Trinity College Dublin
Coláiste na Tríonóide, Baile Átha Cliath
The University of Dublin

Benchmark Comparison of JavaScript Frameworks

React, Vue, Angular and Svelte

Wenqing Xu

A research Paper submitted to the University of Dublin,
in partial fulfilment of the requirements for the degree of
Master of Science Interactive Digital Media

2021

Declaration

I have read and I understand the plagiarism provisions in the General Regulations of the University Calendar for the current year, found at: http://www.tcd.ie/calendar

I have also completed the Online Tutorial on avoiding plagiarism 'Ready, Steady, Write', located at http://tcd-ie.libguides.com/plagiarism/ready-steady-write

I declare that the work described in this research Paper is, except where otherwise stated, entirely my own work and has not been submitted as an exercise for a degree at this or any other university.

Signed: 

Wenqing Xu

23 May 2021

Permission to lend and/or copy

I agree that Trinity College Library may lend or copy this research Paper upon request.

Signed: 

Wenqing Xu

23 May 2021

Acknowledgements

I would first like to thank my parents and friends for all their love and constant support, without which I would not be where I'm now. I must express my very profound gratitude to my supervisor Glenn, without whom this thesis would not have been possible. Whenever I had a question about my research or writing, he steered me in the right the direction. I could not have asked for a better supervisor. Thanks for the all the encouragements and valuable feedbacks that makes my thesis a memorable one.

Wenqing Xu

Abstract

Due to the complexity and variety of JavaScript front-end frameworks, it can be difficult for a developer to choose the most suitable framework for their project. For a novice who wants to learn a front-end framework, choosing the appropriate one to start with can also be a struggle. However, the choice of framework is a crucial part of project development. Therefore, the main challenge for developers is usually to choose the appropriate framework to accomplish their work.

This paper will benchmark various front-end frameworks to analyse their performance, explore the underlying reasons and give advice to developers on choosing a front-end framework based on their project.

Contents

List of Figures

List of Tables

1 Introduction

The aim of this research paper is to investigate the performance of popular front-end frameworks in an attempt to determine how developers should make a choice according to their projects.
This aim will be refined into specific research questions and objectives in section 1.2 and 1.3.

1.1 Background

1.1.1 The complexity of JavaScript frameworks

Web front-end development is a field that has been gradually subdivided with the development of the Web. In the early days, front-end development was rather straightforward, with the page being the unit of work and the content of the site being mainly used for display purposes. At this time there was very minimal JavaScript logic on each interface and basically no framework was needed. As the Internet continues to evolve, pages are not only used to display content, but also include interactive functions and special effects with users. In addition, the logic of front-end code is becoming increasingly complex. With the proliferation of smartphones and tablets of all sizes in recent years, an increasing number of people are using mobile devices to access the internet, which places a higher demand on the cross-platform, reusability and ease of maintenance of front-end code [1]. If there is a cross-platform, reusable and easy-to-maintain framework, it can significantly improve development efficiency. At the same time, the environment in which JavaScript works is complicated by the fact that different browsers have different support for the ECMA standard, and as a result there is no consensus on what JavaScript can do, what it should do and how it should do it.
A Framework is a technology that allows for a defined code structure to be laid out in order to achieve a fast and complete solution. The technology has evolved as the demand for front-end functionality has expanded [2]. The variety and complexity of the JavaScript frameworks and libraries available today stems in large part from the specificity of web front-end development itself. From the days when it was optional to the fully functional HTML5 standard of today, the role of JavaScript in web applications has changed dramatically. A web application can have all of its business logic on the server side, with little reliance on JavaScript, or it can be built entirely in JavaScript on the client side, with the server only responsible for the data interface. It is also likely that a compromise in between can be chosen. At the same time, due to JavaScript being a fairly flexible language, developers from various backgrounds draw on many diverse software design ideas to build their ideal JavaScript framework, which results in different frameworks/libraries often having distinct solutions to the same problem [3].

Front-end technology is iterating at a rapid pace. A few years ago, elements were still being manipulated with jQuery, but in recent years React and Vue have started to

take over the front-end market. The worst part of this phenomenon is that front-end frameworks have a very short-term lifecycle. constantly, we see a brand-new framework claim to have revolutionised UI development, then thousands of developers use them in their projects, blog about them, share their experiences, and day after day, a newer or even more revolutionary framework comes along to replace the last "king". The JavaScript front-end framework lifecycle can be described as brutal. They all seem to have a rapid rise in popularity as they become more widely used, and then gradually transition from a slower rise to a steady decline as developers adopt newer technologies. Overall, the lifecycle of most front-end frameworks is too fleeting due to the rapid iteration of releases and updates.

1.1.2 The importance of performance metrics for websites

For websites, performance metrics have serious practical implications. Better performance means less load time, higher user satisfaction, which can increase revenue for commercial websites. Interaction speed is an important performance metric, with a reduction in download time of a few milliseconds increasing user interaction and retention. E-commerce company Zalando found that a 100-millisecond improvement in load time on their site led to a 0.7% increase in customer revenue [4]. The speed at which an interaction can be performed is important when dealing with large numbers of user requests and technical difficulties such as slow connections. Layout Shift is also an essential metric. If unexpected shifts in page content that occur due to JavaScript loading resources asynchronously or adding DOM elements to pages above existing content dynamically, some sites that require security and stability may suffer unpredictable damage and irreparable consequences [5].

From a website usage perspective, different types of websites will have varying levels of performance focus. As the performance of websites relying on different front-end frameworks varies considerably, this places greater demands on developers to choose the appropriate front-end framework. It is clear from the design of the front-end framework why they behave differently:

Encapsulating native APIs and common tasks: JavaScript's native APIs suffer from cumbersome object and method names, a lack of syntax for common tasks, and poor compatibility with older browsers. As a result, the primary goal of early frameworks was to encapsulate cumbersome native APIs and common tasks into simpler and more intuitive APIs, while also dealing with compatibility issues in the process. JQuery is a typical solution to this part of the problem. The introduction of the HTML5 and ECMAScript 5 standards has made these issues somewhat better. As the new standards become more widespread, the need for this part of the functionality will diminish in the future. In general, wrapped objects include the following categories: DOM selection and manipulation, DOM event handling, Ajax, and language enhancements.

Infrastructure: This section is usually the underlying functionality of the various frameworks and determines how the framework's code is organised together. The goal is to improve the maintainability, collaboration and testability of the code.

Rich Application Architecture: The main purpose of this section is to further improve code reuse using design patterns, allowing the developer's efforts to focus primarily on implementing the functionality of the application itself.

Visual Interaction: Traditionally large frameworks have included this component, usually based on extensions to their own architecture, with a dependency on the framework itself. Some new frameworks, however, focus solely on the architecture and are completely open to extensions, promoting the idea of allowing developers to choose the most appropriate tools themselves.

In summary, front-end frameworks have different design and architectural biases and priorities, resulting in inconsistent performance in all aspects of their performance. As a result, the performance of implementing the same application or performing DOM operations with different front-end frameworks varies [6].

1.2 Research Problems

Due to the complexity and variety of JavaScript front-end frameworks, it can be difficult for a developer to choose the most suitable framework for their project. For a novice who wants to learn a front-end framework, choosing the appropriate one to start with can also be a struggle. However, the choice of framework is a crucial part of project development. The framework is the backbone of a project, which means that once it has been chosen for project development, it will not be changed at will. Frameworks can be highly invasive to a project. If the framework is changed, the whole project may need to be rebuilt [7]. Therefore, the main challenge for developers is usually to choose the appropriate framework to accomplish their work.

Developers should base their choice of framework on a popular, highly used and viable front-end framework, as the size of the community is important to developers, meaning a more friendly and rich thriving open-source environment where reference and help can be more easily sought. Not only that, choosing a framework with a high level of viability means that it is less likely to become obsolete when it comes to maintenance and updates in the future.

In addition, the diversity of overall architectural options makes it possible for different applications to have very distinct needs for front-end architecture, which means that a single front-end framework or library is difficult to satisfy everyone. Even skilled front-end developers can find it difficult to master the features of all frameworks. However, the comparison of JavaScript frameworks is not a mature field yet and is still relatively new. The closest area to comparing JavaScript frameworks is software architecture comparison software architecture comparison is also a fairly young discipline. One of the most popular methods of comparison is the Software Architecture Analysis Method (SAAM), which originated in 1996. The project was executed at Decerno, a small IT consulting company founded in Sweden in 1984 that has proven to be able to build bespoke systems for its clients. Their focus has always been on web development. When building custom systems, they do a thorough assessment of the tools and frameworks required for a particular project. Building single page applications using the emerging JavaScript frameworks is their current concentration

[8]. It is fair to state that there are few studies that do an in-depth comparison of the various front-end frameworks, but in the context of the complexity and diversity of front-end frameworks, studies that summarise and generalise the features of front-end frameworks are needed by developers.

In furthermore, a number of studies comparing front-end frameworks now take the approach of using different front-end frameworks for simple DOM manipulation, which is accurate but too idealistic to be a realistic guide considering that applications currently in production use are largely web-based. However, if the same networking application is implemented with different front-end frameworks and the performance of each framework is tested to get results. Then such results could be affected by changes in placement advertising, changes in Internet traffic routing or browser extensions, and the reader could easily question the reliability. There is a lack of research on the performance of front-end frameworks that is both instructive and relatively accurate for real-world, operational web-based software.

1.3 Research Objectives

In light of the above issues, the evaluation and benchmarking of web development frameworks in the field of software development and software engineering is the main motivation for this project. In particular, with the growing demand for the web and the need for web developers to continuously find new ways to develop their applications more efficiently (Graziotin and Abrahamsson, 2013), this paper aims to guide developers in selecting a suitable front-end framework. The research objectives of this paper are as follows.

This paper is based on the 9 most satisfied mainstream frameworks and libraries from The State of JavaScript Survey, and selects the 4 most popular and used frameworks based on Github Stars and npm downloads. This gives developers, especially newcomers who want to learn front-end frameworks, an overview of the trends in front-end framework popularity.

This paper provides some metrics comparing front-end frameworks and points out measurement tools. Researchers undertaking front-end frameworks in the future can also refer to these metrics and tools for data measurement.

This paper collects data on the performance of each framework in each metric and analyses all the data obtained, pointing out the strengths and weaknesses of each framework. The comparison of frameworks is not intended to select the best of all frameworks, but rather to guide developers in selecting the appropriate framework for their projects in production based on the features of each framework.

This paper provides a rigorous scientific approach to front-end framework comparison, implementing the same networking application with different front-end frameworks, guiding developers to choose the right framework for their projects in real production. Afterwards, the underlying architecture is analysed by comparing the performance of each front-end framework in different DOM operations. Finally, the results of the two experiments are checked to see if they validate each other. This approach to the comparison of front-end frameworks is more rigorous and makes the

results more convincing.

The framework comparison in this article goes step by step from the surface performance analysis to the underlying architectural dissection, making it understandable even for beginners.

2 View of art

2.1 Front-end foundations and catalysts

2.1.1 Development of browsers

HTML5 has already become a symbol that basically all web interfaces with gorgeous interfaces or interactions, whether on computers or mobile, are based on it. The development of HTML5 and the series of front-end changes it has brought about cannot be separated from the development of modern browsers and the updating of older browsers, typically represented by IE.

2.1.2 HTML and CSS

HTML is a web authoring technology known as Hyper Text Markup Language. As the basis of web front-end technologies, HTML is essentially a series of tags that can be standardised into a text file. It is not only a mark-up language for web production, but can also be used to create hypertext documents. HTML5, as the representative of HTML language, is the fifth version of Hypertext Markup Language. HTML5 is an open standard protocol rather than a simple upgrade from version 4, which was released as an official recommended standard by the World Wide Web Consortium W3C in October 2014 and is the first HTML standard to use the Web as a platform for application development. Information on the Web can be delivered to smartphones and other wireless communication devices using HTML5 technology as HTML5 has been developed on top of existing Internet standards, which means it has been improved and upgraded for the characteristics of wireless networks. HTML5 technology has added new forms of implementation such as audio and video to the original technology, revolutionising the history and status of online preservation and making storage in a non-networked state possible.

CSS is the Cascading Style Sheet, a technology that allows the layout, colours and fonts of a web page to be set up in a way that beautifies the web page, as well as being a presentation language technology. CSS3 is an upgraded version and a representative of CSS technology, the main feature of which is to control the display of the web page. CSS technology not only beautifies web pages, but also makes it possible to create web pages quickly and efficiently. Moreover, it is a convenient way to create web pages. The CSS includes the box model, the list module, the hyperlink approach and the multi-column layout modules [9].

HTML5, the core language of the next generation of the Internet, is set to drive the rapid development of the Internet. With the advent of the mobile internet, web pages are bound to be influenced by this era as a vehicle for carrying all kinds of information, with improvements in scanning graphics, playing video and audio, and support for more platforms. HTML5 has obvious advantages in various aspects, making web front-end designers more inclined to integrate it with CSS3, which will promote more colourful and interactive web pages. CSS3, as a new generation of style sheet files, has

strengthened the control of page element settings, Web search and other aspects to achieve the goal of a user-friendly, quick query web page. The development and advancement of HTML5 and CSS3 is the result of the combined efforts of a number of web designers and engineers, playing a more important role in improving the interface and enhancing the user interaction experience. Especially in the integration of various language technologies, it has been realised that the relationship between HTML5 and CSS3 cannot be ignored. For web front-end designers, it is important to develop an integrated view of development, to sort out the relationship between the two, to apply it to specific web designs and to promote the integration of the two technologies in practice.

2.2 JavaScript

2.2.1 Introduction to JavaScript

JavaScript is a high-level programming language based on the ECMAScript standard, which is a mixture of a simplified functional programming language and an object-oriented programming language. JavaScript consists of ECMAScript, the Document Object Model and the Browser Object Model, with ECMAScript implementing the syntax description of the JavaScript language, the Document Object Model (DOM) being the interface to the methods used to process web content, and the Browser Object Model (BOM) implementing the interaction with the browser. The JavaScript programming language features simplicity, efficiency, dynamism, interactivity, cross-platform, interpretability and light weight [10].

2.2.2 The ECMAScript standard

The year 2015 marks the 20th anniversary of JavaScript, and the year of the ECMAScript 6, which is the standard for the JavaScript language, making it possible to write complex, large-scale applications and become an enterprise-level development language. ES6 is the biggest change to the ECMAScript standard to date, bringing with it a range of exciting new features for developers [11]. JavaScript is slowly becoming less limited to front-end development, and the introduction of Node.js has given a taste of what is possible with JavaScript for full-stack development.

2.2.3 TypeScript

Among the four frameworks compared in this study, React and Vue use JavaScript, while Svelte and Angular employ TypeScript, a free programming language developed by Microsoft that is a superset of JavaScript. Improvements to TypeScript over JavaScript include the addition of type declarations and compile-time type checking, the addition of a full class structure that more closely resembles a traditional object-oriented language, the addition of interfaces, modules, and Arrow functions that resemble Lambda functions. Anders Hejlsberg, the chief architect of C#, was involved in the design of TypeScript [12].

2.2.4 Browser-side magic: AJAX

AJAX is Asynchronous JavaScript and XML, which refers to a set of integrated browser-side web development technologies that can be used to create more interactive web applications based on JavaScript's XmlHttpRequest. AJAX is a mashup of existing technologies, a combination of technologies together to form its features and advantages, and has been used since as early as 1998. Google's in-depth use of this technology in products such as Maps and Gmail, and the introduction of the eye-catching name AJAX, has officially put it in the spotlight and started to attract countless attention [13]. When a user submits a form in a web application, they send a request to the web server, which receives and processes the incoming form and returns a new page. Most of the HTML code in both pages is the same, and returning the entire page each time is a waste of bandwidth resources. AJAX applications, on the other hand, only send and retrieve the necessary data to the server, and use JavaScript on the client side to process the response from the server, then update the local information on the page. In this way, not only is the exchange of data between browser and server significantly reduced, but the client can also respond more quickly to user actions [14]. The advent of AJAX and the emergence of some front-end development frameworks such as EXTJS and DOJO also made Single Page Application (SPA) popular at this time.

2.2.5 The Document Object Model

The Document Object Model was originally developed as a model standard by the W3C organisation to address differences between browser environments in the era of browser chaos, primarily for InternetExplorer and NetscapeNacigator. The W3C explains it as the Document Object Model (DOM) is a language platform that enables programs and scripts to dynamically access and update the content, structure and style of documents, providing a standard set of XML and HTML objects with a standard interface to access and manipulate them. It enables programmers to access standard components such as elements, style sheets, scripts on XMI and HTML pages quickly and to manipulate them accordingly [15].

The DOM is an application programming interface for developing and programming applications on XML or HTML documents. As a cross-platform, language-independent interface specification published by the W3C, the DOM provides a standard programmatic interface in different environments and applications, which can be implemented in any language. The DOM can be used to parse structured XML or HTML documents, the instructions, elements, entities and attributes in the document are represented by an object model. The entire document is seen as a structured tree of information, rather than a simple stream of text. The generated object model is the nodes of the tree, and the object contains both methods and properties. Therefore, all operations on the document are performed in the object tree. In the DOM, everything in the tree is an object, whether it is the root node or an entity's properties. Developers can dynamically create XML or HTMI documents, traverse the structure, add, modify, delete content using the DOM. Its object-oriented nature allows people to save a lot of

effort when dealing with XML or HTML parsing-related matters, and is a powerful programming tool in line with the idea of code reuse [16].

2.2.5.1 Virtual DOM

Virtual DOM was created to optimise DOM manipulation. It automatically calculates the differences between the new page and the old page, and then applies these differences to the old page in the form of DOM operations, not only minimising DOM operations on the page, but also making the whole process completely automated without the need for developer involvement.

The core idea of Virtual DOM is to calculate the differences between the old and new pages, and then apply these differences to the old pages, so it consists of three main parts: 1) Simulating the real DOM with JS. 2) Comparing the differences between two virtual DOM trees. 3) Applying the differences to the old pages. DOM-Diff is the core of the entire virtual DOM theory. The DOM-Diff process is summarised as when the state of the page changes, a new Js object tree is reconstructed, the new object tree is compared with the old object tree, and the differences between the two object trees are recorded using the DOM-Diff algorithm [17].

2.2.6 JQuery

JQuery was created by the American John Resig in early 2006, is another excellent Javascript library after prototype, belongs to the lightweight and flexible js library, compatible with CSS3 and a variety of browsers, not only can make the user more convenient to deal with events and HTML system, but also can provide website exchange using Ajax to better match the web application. JQuery makes it easy for users to manipulate HTML elements, handle events, implement various animation effects, and provide AJAX interaction so that users can obtain information from the server without refreshing the page [18]. JQuery is designed to use method chaining, which not only reduces the length of code, but also allows for special effects in which later operations in the JQuery chain are based on the results of previous operations. The JQuery library is well encapsulated, so that its namespace does not conflict when used with other libraries. The JQuery principle is to allow users to write less, do more. It is easy to separate structure from behaviour with JQuery. Only the structure is defined in the page without interaction logic, and the interaction of elements is defined in the script file, reflecting front-end layered design thinking.

JQuery is licensed under the MIT License, remains free and open source, and is a multi-browser compatible Javascript library. One of the prominent features of JQuery is its syntax, which is designed to make it easier for users to handle events, create animations, use Ajax and select DOM elements. In addition, JQuery provides plug-ins, or APIs, that make it easier for developers to control modularity and to interact with dynamic and static web pages. At the same time, JQuery can be modified with plug-ins to perform functions such as Tab navigation, table sorting, image effects and comes with its own effects, including show and hide functionality [19].

2.3 Modularity and engineering

2.3.1 Framework versus library

It is well known that the distinction between libraries and frameworks has always been somewhat blurred in front-end development. Projects like jQuery and YUI are officially described as “libraries”, but are often referred to as “frameworks” in various places. Some MVC projects that have emerged in the last two years are called frameworks, but are actually more like libraries. In addition, the functionality covered varies between projects that also claim to be frameworks. The traditional software engineering distinction between libraries and frameworks focuses on the control of the application runtime process. Frameworks provide the architecture, control the runtime, and allow developers to write problem-specific code in the right places; libraries, on the other hand, are subordinate to the architecture, do not control the runtime, and only provide functions that can be called [20]. However, due to the particular nature of web front-end development described above, such a definition seems overly strict: there are few projects that qualify as frameworks, yet they often need to be compared with projects that are libraries. Therefore, when comparing JavaScript open-source projects, it is not particularly important whether they are frameworks or libraries.

2.3.2 The Model-View-Controller pattern

In the traditional MVC pattern, an application system is divided into three main aspects: the Model layer, the View layer and the Control layer.

Model refers primarily to the data model. In the system, data processing is carried out and data can be accessed directly with the help of this model. In the process of application, the data is processed in a simple way with the help of business logic in combination with packaging or other applications to obtain valid data and relevant information.

View is a view of the data. Once the data model has been constructed, the system contains a large amount of data and relevant parameters, which need to be rationalised by the developers. This allows the system to be run more efficiently, making full use of the data information. Generally speaking, there is no fixed logic in the views. In order to make the most of the latest functions on the view, the data model it monitors needs to be accessed. The data can then be accessed and analysed effectively.

The Controller is the vehicle through which data models and views are viewed and applied. It controls the processes of the applications and strengthens the link between the data model and the view. In the case of event processing activities, the system needs to be able to react quickly. At this stage, the processing of events does not simply include the user's actions, but also the data model. In this case, the controller needs to actively and spontaneously capture user events. Relevant data and information is updated and reflected with the active role of the model layer. The model layer needs to be updated in a timely manner after the above-mentioned content and operations have been completed. At the same time, the updates and notifications of changes are

transmitted to the view, which then makes the corresponding changes according to the system's operational requirements. From a positive point of view, such an application maintains a high degree of consistency between the view and the data model thanks to the advantages of the controller. This facilitates the post-operational and data processing process and to a certain extent improves the efficiency of the system as well as the processing of data. As a result, the system can be widely used at a social level [21].

2.3.3 Front-end MVC framework

In this model, the division of labour between the front and back ends is very clear, and the key collaboration point between the front and back ends is the Ajax interface. After specifying the interaction interface, the front and back-end engineers can start working separately according to the agreement, and test the development environment, while conducting front and back-end integration tests at specific time points. Nowadays, it is easy for JavaScript code to bloat for a SPA with complex front-end functionality and interaction. Naturally, like the server-side conversion from JSP to MVC frameworks, a large number of MVC frameworks have emerged for front-end development, more typically including BackboneJS, AngularJS, EmberJS, KnockoutJS [22]. In summary, the MVC framework is proposed to address the complexity of front-end development by providing a set of rules to organise code and layering, making the front-end code responsibilities clear for development and testing.

2.3.3.1 React.js

The React.js framework, which originated as an internal Facebook project and was open sourced in May 2013, is a JavaScript library for building user interfaces. It can be used to combine a number of short independent code fragments into a complex UI interface. It has high performance, simple code logic and can easily solve cross-browser compatibility issues. React.js allows the user interface to be combined directly with, for example, button components, dialog components and so on. By combining these components, developers can have rich interactive web pages. At the same time, the introduction of the relevant JSX syntax into the user interface makes reusing components simpler and ensures that the internal structure of the components is clear. In addition, on top of these components, React Js can differentiate the code from the real target straight away, that is, using the rendering capabilities of the DOM in the browser to develop web pages, thus facilitating the development of native mobile applications [23].

2.3.3.2 Vue.js

Vue.js is a set of progressive JavaScript frameworks for building web interfaces. In contrast to other front-end frameworks, Vue.js is a bottom-up incremental development of layer-by-tier applications with a focus on the view layer for easy integration with third-party libraries or existing projects. Vue.js uniquely provides MVVM data binding and a composable component system. From a technical perspective, Vue.js focuses on

the view model layer on top of the MVVM pattern and uses bi-directional data binding to connect the view and model to drive the page. Compared to other MVVM frameworks, Vue.js is easier to get started with and can quickly update the DOM through asynchronous batch processing, as well as combining reusable and decoupled components together, allowing for multiple module installations and more flexible scenarios [24].

2.3.3.3 Angular.js

Angular.js is primarily used to build single-page dynamic page designs, with more focus on building CRUD (Create, Retrieve, Update, Delete) applications, created by MiskoHevery et al. in 2009. Angular.js framework has many features, the core of which are MVW, modularity, automated bi-directional data binding, semantic tagging and dependency injection. It extends HTML with directives and binds data to HTML with expressions. Angular.js is described as a complete framework because it contains everything from templates, two-way data binding, routing, modular services, filters and dependency injection. An Angular.js directive can be used in HTML code in four ways: a new HTML element, an element's attributes, an element's classes and comments [25].

2.3.3.4 Svelte.js

Svelte.js is a tool for building fast web applications. It is similar to JavaScript frameworks such as React.js and Vue.js, which share the common goal of making it easy to build slick, interactive user interfaces. There is one key difference, however, Svelte.js converts the application to ideal JavaScript at build time, rather than interpreting the application code at runtime [26]. This means there is no performance cost to the framework abstraction and no penalty when the application is first loaded.

2.3.4 Front-end engineering

Most of the time when it comes to the concept of engineering, it is often referred to as tooling. Inevitably, however, any path to engineering will take a tooling path. The goal of the front-end engineering path is to be able to build and maintain effective, functional and high-quality software using an engineering disciplined approach.

The elements of engineering would be the following.

1) Uniform development specifications and compilation tools. In fact, given the differences in browsers, when writing front-end code, programmers are writing across multiple "platforms", so the syntax, flow, engineering structure, etc. need to be standardised.

2) Modular/component-based development. In a real project, programmers often need to collaborate on development, which was previously often divided by pages, but this resulted in a lot of duplicated code and could be very cumbersome to maintain. Dividing by modules and components reduces unnecessary duplication of effort and facilitates maintenance.

3) Unified component publishing and repositories. The emergence of centrally managed repositories such as Maven has given programmers a unified central

repository and version management tool, which avoids a lot of duplication of work [27].

2.3.5 Tool chains

2.3.5.1 Package manager: npm and Yarn

Npm is a package management tool that is installed with Node.js and can solve many problems with Node.js code deployment, with the following common usage scenarios.

- Allows users to download third party packages written by others from the npm server for local use.
- Allows users to download and install command line programs written by others from the NPM server for local use.
- Allows users to upload packages or command line programs they have written to the NPM server for others to use [28].

Apache Hadoop Yarn (Yet Another Resource Negotiator) is a new Hadoop resource manager, a common resource management system and scheduling platform that provides unified resource management and scheduling for upper-level applications. The introduction of Yarn brings great benefits to clusters in terms of utilisation, unified resource management and data sharing.

Yarn can be thought of as the equivalent of a distributed operating system platform, with applications such as MapReduce running on top of the operating system, providing the resources (memory, CPU) needed for these applications to operate. It has the following features.

- Yarn is not aware of the mechanism by which user-committed programs run.
- Yarn only provides scheduling of computing resources.
- The supervisor role in yarn is called ResourceManager.
- The specific role in Yarn that provides computing resources is called NodeManager.
- Yarn is fully decoupled from the running user program, meaning that all types of distributed computing programs can be run on Yarn.
- Computing frameworks such as spark and storm can all be integrated to run on Yarn, as long as their respective frameworks have resource request mechanisms that conform to the yarn specification.
- Yarn becomes a common resource scheduling platform. The various computing clusters that previously existed in the enterprise can be consolidated on a single physical cluster, improving resource utilization and facilitating data sharing [29].

2.3.5.2 Bundler: Webpack

Webpack is a front-end resource loading/packaging tool, which will statically analyse modules according to their dependencies and then generate the corresponding static resources from these modules according to the specified rules. Webpack provides a powerful loader mechanism to make it more flexible. The popularity of Webpack is of course due to React and Facebook, but it is also pleasing that Webpack also has the delightful feature of supporting LazyLoadComponent, and this lazy loading technique

is framework-independent, which avoids the need for developers to think about fixed components or code partitions when coding, after all it is difficult to plan all the component partitions at the beginning of a fast iterative project. Webpack also supports hot swapping of code with the React Hot Loader, which can greatly improve the efficiency of code development [30].

3 Design

3.1 Experimental Methodology

The next section will start from the performance of each front-end framework, aiming to help users choose the appropriate framework for development.

Firstly, representative front-end frameworks are selected based on market demand and user evaluation. After that, this paper designs two types of experiments, the first is to build the same website using different front-end frameworks, based on comparing the performance of the different frameworks in each of the selected metrics, to discover the advantages and disadvantages of the different frameworks and the projects they are applicable to. In this paper, it is hoped that this site is universal, representative, and instructive for real-life development and production. It features a networked, moderately sized project, and the Realworld Application will be chosen for this paper and will be described in detail in Section 3.3. The second experiment is a benchmark test that measures the duration of each Dom operation. The tests are run locally without any impact on performance due to the network. By comparing the performance of each front-end framework in different Dom operations, this paper will delve into the architectural layer to explore the reasons and draw conclusions.

Finally, this paper will evaluate whether the experiments validate each other based on the results obtained. If the results are unified, a more consistent conclusion can be drawn. However, if the results are contradictory, it indicates that there are a few problems with the experiment. The benefit of setting up two types of experiments is that the results are more convincing in this manner. With bare results from Dom operation-based benchmarking, such experimental content is too simplistic and idealistic when compared with feature-rich websites on the market. If the Realworld Application is used alone, comparing the results of different frameworks implementing it, the results are likely to be affected by network and other factors, incurring some errors. In this paper, by combining two implementations to verify and complement each other, the reliability can be greatly improved.

3.2 Front-end framework selection

The State of JavaScript Survey team has recruited tens of thousands of developers since 2016 to participate in the questionnaire survey on front end frameworks and libraries and publish the results, which build a comprehensive picture of the community and ecosystem to find out where the choice of front-end frameworks is going [31]. The survey was chosen because it compares the most popular frameworks available today and the sample is broad, based on the ideas of tens of thousands of developers worldwide. The 2020 survey covered 23,765 people in 137 countries, surveying nine leading front-end frameworks and libraries. Data on the annual satisfaction ratio over the time span from 2016 to 2020, is shown in Figure 3.1.

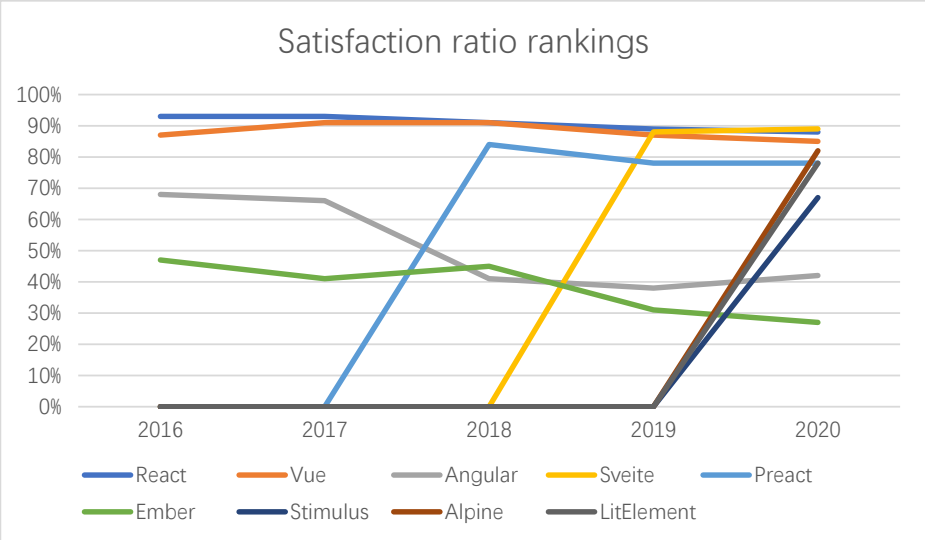

Figure 3.1: Satisfaction ratio rankings, State of JavaScript Surveys 2016 - 2020

As can be seen from this graph, Svelte has the highest satisfaction rate of all frameworks in the 2020 satisfaction ratio rankings with 89%, a 1% increase in satisfaction from 2019 when it was ranked second. React ranks second, with only 1% less satisfaction than Svelte. In previous years, however, React has always ranked first in terms of satisfaction. Vue is very close to React, with a satisfaction rate of 85%. Over the five years, Vue's satisfaction rose and then fell, tying with React for first place in 2018. The Alpine framework, which joins the survey in 2020, has a syntax almost entirely borrowed from Vue (and extended by Angular) and will not be discussed in this article. Preact is in the middle of the pack, dropping from 84% satisfaction in 2018 to 78% satisfaction in 2019 and 2020. The next LitElement is a simple base class for creating fast, lightweight web components and will not be included in the discussion of this article. Stimulus, a nascent front-end framework, has a 67% satisfaction rating. It can be seen that Angular has a satisfaction rate of only 42%, it has been decreasing from 2016 to 2019, with a slight recovery in 2020. ember has a decreasing trend in this time period, reaching its lowest point in 2020 with only 27% satisfaction, ranking the worst in this questionnaire.

The second data source for selecting front-end frameworks for this paper is Github stars. GitHub is the world's largest code hosting platform, used by more than 50 million developers, and Github stars reflect the popularity of projects. By the end of March 1, 2021, the front-end frameworks selected for this phase are over 20,000 github stars, visualized in Figure 3.2.

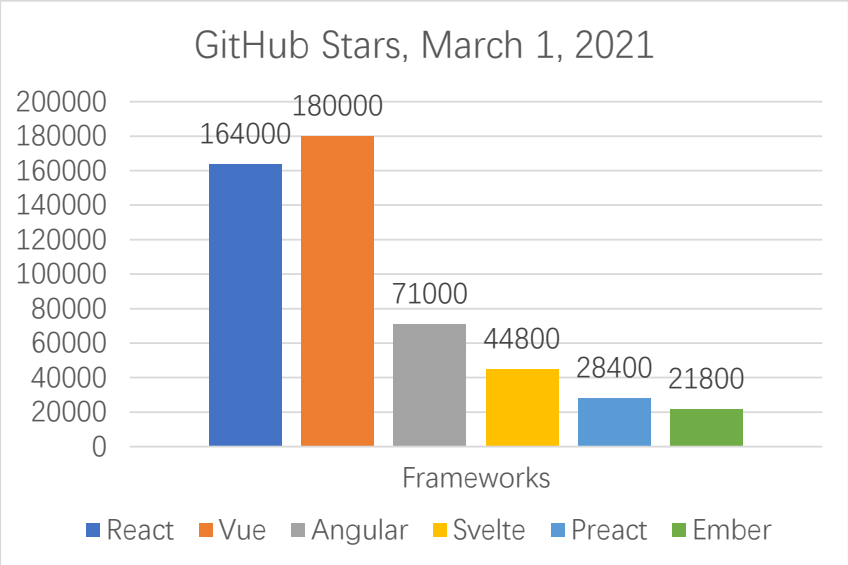

Figure 3.2: Front-end frameworks with over 20,000 Github stars and their number of Github stars

The framework with the most Github stars is Vue, with 180,000 stars, followed by React with 164,000. The newer framework Svelte has 44,800 stars, followed by React and Ember. The new front-end framework Stimulus will be dismissed in the next section because it has less than 20,000 stars and its popularity is too low compared to other frameworks.

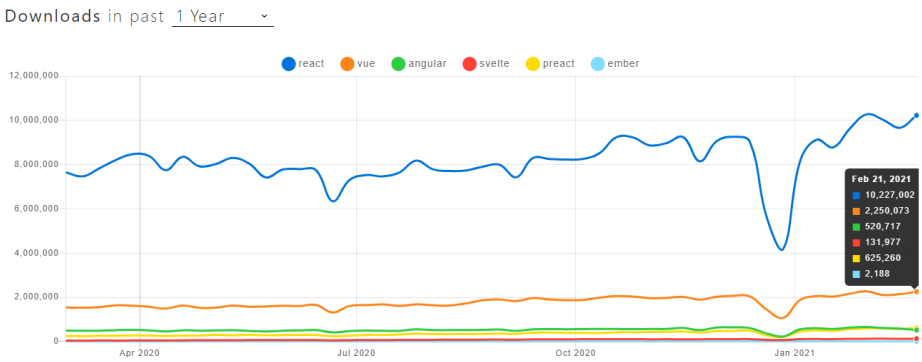

Figure 3.3: Npm downloads of every frontend framework in past 1 year

This article also uses the number of npm downloads for each front-end framework for the past year as another selection criterion which is shown in Figure 3.3. The ranking is stable, taking February 21, 2021 as an example, the number of React downloads is 10,000,000 in the order of magnitude, ranking first, far beyond other front-end frameworks. Vue downloads is about 1/5 of React, ranking second. Preact, Angular and Svelte all have downloads in the magnitude of 100,000, while Ember, the least, has only about 2000 downloads. Ember's npm downloads are too small compared to other frameworks, which means it is far less used than other frameworks, so it will not be discussed in the subsequent part of the paper.

Furthermore, since Preact is essentially a re-imagined version of React, optimized

for faster performance. It was not included in this paper due to its similarity and lower popularity.

In conclusion, the front-end frameworks experimented in this paper are React, Vue, Angular and Svelte based on the user satisfaction in State of JavaScript Surveys, the number of Github Stars over 20,000 and npm downloads.

3.3 Realworld Application with Benchmarks

The Realworld Application is a case-study of a blog-type application called Conduit [32]. It was originally created to help developers fully understand and master new frameworks. In this era of rapid updates to the JavaScript ecosystem, learning a completely new framework from scratch can be very time consuming and having a classic example to back it up can speed up the learning process.

APIs for implementing Conduit are available in the Realworld Github repository, allowing the software to be implemented with different frameworks and ensuring the modularity of each front-end and back-end. Each front-end uses the same hand-crafted Bootstrap 4 theme to implement the same UI/UX, and the back-end API has a hosted version for public use that does not require an API key. Developers can use the new framework to implement the same functionality, greatly increasing learning efficiency.

By the end of 7 March 2021, Realworld Application has already collected 551,000 Github stars on the Github repository. Some of its technologies make full use of the web, such as querying data and persisting it to a database, authentication systems, session management, and full CRUD to resources. Conduit also has great features such as Authentication with JWT, Profiles with images, Write/edit/read articles, Comments on articles, Ability to "favorite" articles and Ability to follow other users & have their articles show up in the feed. Arguably, the Conduit blog site can be used as an example of a simple yet powerful web application.

In the next section, the Realworld Application is first implemented using the four front-end frameworks chosen in the previous section — React, Vue, Angular and Svelte. Then the appropriate metrics are selected for the benchmarks and the method of measuring each metric is demonstrated.

In general, Angular has an extremely fixed and complex structure and requires developers to code in a specific way with less freedom and flexibility. As a library as opposed to a framework, React doesn't offer everything that Angular does. React only provides a library for rendering content to the DOM and subsequently controlling it effectively. Applications developed in React need to include other libraries. Vue provides built-in state management and also comes with a built-in router, but it does not include form validation or HTTP client functionality. Svelte template syntax is very similar to that of Vue. Both Svelte and Vue use an HTML-based template syntax and have the concept of a single file component.

3.3.1 Building Conduit with React

In this section you will find an overview of how to build the Realworld Application

with React.

First install create-react-app from npm and use create-react-app to generate a React project in the command line.

```
create-react-app example-app
cd example-app
```

Secondly, code structures and communication between components are achieved using Redux, a predictable state container for JavaScript applications, which can help write applications that perform consistently, run in different environments (client, server and native) and are easy to test [1]. Its workflow is shown in Figure 3.4.

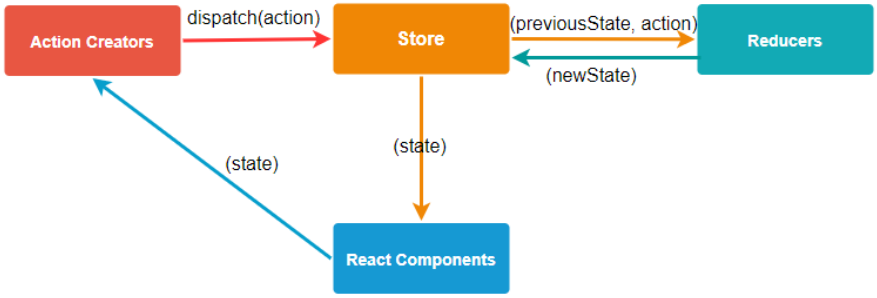

Figure 3.4: Workflow diagram for Redux

An explanation of the Redux workflow is as follows.

The user issues an Action.

```
store.dispatch(action);
```

The Store automatically calls the Reducer and passes in two parameters: the current State and the received Action. The Reducer returns the new State.

```
let nextState = todoApp(previousState, action);
```

As soon as the State changes, the Store will call the listener function.

```
store.subscribe(listener);
```

listener can get the current state via store.getState(). If React is used, this can then trigger a re-render of the View.

```
function listerner() {
  let newState = store.getState();
  component.setState(newState);
}
```

Next, bind the component to the Redux state using the react-redux module. The state is then bound to the App component using connect() in the module. Once the state is finished connecting to the components, write code to implement the individual components.

After that, the agent.js file is built, which is used to interact with the RealWorld

APIs. The agent.js file is built, which is used to interact with the RealWorld APIs. The Http proxy is then combined with the Store using an intermediate component built using Promise asynchronous operations.

```
import superagentPromise from 'superagent-promise';
import _superagent from 'superagent';

const superagent = superagentPromise(_superagent, global.Promise);
const API_ROOT = 'https://conduit.productionready.io/api';
const responseBody = res => res.body;

const requests = {
  get: url =>
    superagent.get(`${API_ROOT}${url}`).then(responseBody)
};

const Articles = {
  all: page =>
    requests.get(`/articles?limit=10`)
};

export default {
  Articles
};
```

Finally, the routes are constructed and the components are connected to them.

3.3.2 Building Conduit with Vue

In this section you will find an overview of how to build the Realworld Application with Vue.

Firstly, the project is initialised, the Vue scaffolding is installed and a project based on the Vue technology stack is quickly created.

```
npm install --global @vue/cli
vue create realworld-vuejs
```

Next, import the template components. Take the home page of the Realworld Application as an example, start by adding the header, footer and home page template components to the components folder. The file index.vue code is as follows.

```
<template>
    <div>
        <app-header/>
        <router-view/>
        <app-footer/>
    </div>
</template>
```

```
<script>
    import AppHeader from './components/app-header'
    import AppFooter from "./components/app-footer"
    export default {
        name:'AppLayout',
        components:{
            AppHeader,
            AppFooter
        }
    }
</script>
```

The content in <router-view/> in index.vue is dynamically changing, configure it as a nested route and render the home page content here.

```
import Vue from "vue";
import Router from "vue-router";

Vue.use(Router);

export default new Router({
  routes: [
    {
      path: "/",
      component: () => import("@/views/home"),
      children: [
        {
          path: "",
          name: "home",
          component: () => import("@/views/home")
        }
      ]
    });
```

It is also necessary to request data from the APIs using axios to display the data dynamically in the software

```
import axios from "axios";

const request = ({
    baseURL = "https://conduit.productionready.io/api"
  })

 export default request
```

After that, the construction of the routes will continue and the components will be connected to them. This research paper will not go into detail.

3.3.3 Building Conduit with Angular

In this section you will find an overview of how to build the Realworld Application with Angular.

First, after cloning the project on Github, run npm install to install the required dependencies and checkout the seed branch (which is named m-9). AngularJS Seed is an application skeleton for typical AngularJS web applications, allowing for the quick launch of AngularJS webapp projects and development environments for those projects. It includes a sample AngularJS application, pre-configured with the Angular framework installed and a number of development and testing tools for meeting instant web development.

```
git checkout m-9
```

Next, build the basic layout and routing of the Conduit application. Take creating header, footer and home page for example, start by completing the html and ts files according to the API provided and allow these components to be easily imported across the application via barrels.

Then, import the HomeModule, HeaderComponent and FooterComponent into the AppModule and declare rootRouting for the application.

```
  import { ModuleWithProviders, NgModule } from '@angular/core';
  import { BrowserModule } from '@angular/platform-browser';
 +import { RouterModule } from '@angular/router';

  import { AppComponent } from './app.component';
 +import { HomeModule } from './home/home.module';
  import {
    SharedModule,
 +  FooterComponent,
 +  HeaderComponent,
  } from './shared';

 +const rootRouting: ModuleWithProviders = RouterModule.forRoot([],
{ useHash: true });

  @NgModule({
    declarations: [
      AppComponent,
 +    FooterComponent,
 +    HeaderComponent
    ],
    imports: [
      BrowserModule,
      SharedModule,
 +    HomeModule,
 +    rootRouting,
```

```
    ],
    providers: [],


[...]
```

Next, the AppComponent template needs to be replaced with Angular layout components and a router-outlet, after which the data is bound. The rest of the component construction method is similar and will not be repeated.

3.3.4 Building Conduit with Svelte

In this section you will find an overview of how to build the Realworld Application with Svelte.

First configure the environment and install the dependencies needed for the project. node install npx command, the code below is a convenient way to run the Node command. This will download and run the degit command, which will download the latest code from the Svelte project template located in the Github repository to the svelte-project folder.

```
npx degit sveltejs/template svelte-project
```

Now go to that folder and run npm install to download the other dependencies for the template. Define the markup, styles and JavaScript in each component. take the example of creating the home page. First edit the header, footer and import the index.svelte according to the provided template, then build the route and do the data binding.

```
<svelte:head>
   <title>Conduit</title>
</svelte:head>

<div class="home-page">
   <div class="banner">
      <div class="container">
         <h1 class="logo-font">conduit</h1>
         <p>A place to share your knowledge.</p>
      </div>
   </div>

   <div class="container page">
      <div class="row">
         <MainView {p} {tag} bind:tab />

         <div class="col-md-3">
            <div class="sidebar">
               <p>Popular Tags</p>
```

```
                <Tags {tags} on:select='{setTags}' />
            </div>
        </div>
    </div>
</div>
</div>
 <svelte:footer>
</svelte:footer>
```

The binding method for other components is similar.

3.3.5 Benchmark Metrics for the Realworld Application

The environment in which all metrics were measured was Windows. Prior to conducting the tests, the test environment was processed to make it as close as possible to the ideal platform for running the benchmark: as many background processes were ended and applications were ended as much as possible. In addition, as suggested by Intel47 , all disk logging was disabled, as it was found that read/write operations on the hard disk could have an impact on the execution time of the program.

The set up used is as follows:

Processor: Intel® Core™ i5-8250U CPU @ 1.60GHz 1.80 GHz

Machine Belt: RAM 8.00 GB

System type: 64-bit operating systems, x64-based processors

Version：Windows 10 Home Edition

Network: Ethernet LAN connection: 240Mbps Download/20Mbps Upload

Browser versions: Google Chrome version 89.0.4389.90

After implementing each framework to execute the Realworld Application, this section will select the appropriate metrics for benchmarking, which are listed below.

3.3.5.1 Performance

This article will test performance using Chrome's Lighthouse Audit [33]. All configurations are used as shown in Figure 3.5.

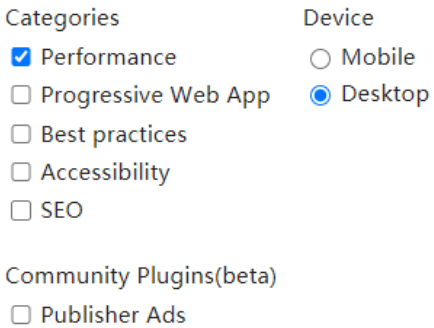

Figure 3.5: Lighthouse Audit configuration

Lighthouse returns a score of 0 to 100. Performance scores are derived based on the following tests.

- First Contentful Paint. It marks the time at which the first text or image is painted.
- Speed Index. It shows how quickly the contents of a page are visibly populated.
- Largest Contentful Paint. It marks the time at which the largest text or image is painted.
- Time to Interactive. It is the amount of time it takes for the page to become fully interactive.
- Total Blocking Time. It is sum of all time periods between FCP and Time to Interactive, when task length exceeded 50ms, expressed in millisecond.
- Cumulative Layout Shift. It measures the movement of visible elements within the viewport.

Once Lighthouse has finished collecting performance metrics (typically in milliseconds), it converts each raw metric value into a metric score between 0 and 100 by looking at the position of the metric value on its Lighthouse score distribution. The score distribution is a log-normal distribution derived from the performance metrics of the actual website performance data on the HTTP archive. The Lighthouse scoring curve model uses HTTPArchive data to determine two control points that then set the shape of a log-normal curve. The 25th percentile of HTTPArchive data becomes a score of 50 (the median control point), and the 8th percentile becomes a score of 90 (the good/green control point). The scores are colour coded with indicator scores and performance scores being coloured according to the following ranges:

- 0 to 49 (red): poor
- 50 to 89 (orange): needs improvement
- 90 to 100 (green): good

The Lighthouse Performance Score is a weighted average of the metric scores for each parameter. The parameters and their corresponding weights will be shown in the Table 3.1.

Table 3.1: Contribution of individual parameters in Lighthouse performance

Audit	**Weight**
First Contentful Paint	15%
Speed Index	15%
Largest Contentful Paint	25%
Time to Interactive	15%
Total Blocking Time	25%
Cumulative Layout Shift	5%

Scores may vary due to Internet traffic routing changes, browser extension, etc. Weights were chosen to provide a balanced representation of user perceptions of performance. The weights have changed over time as the Lighthouse team regularly conducts research and gathers feedback to understand what has the greatest impact on user-perceived performance.

3.3.5.2 Line of Code

This article will use the code statistics tool Cloc to count Line of Code. Cloc counts lines of code in the src folder of each repository without counting blank lines and comment lines. Line of Code gives an indication of how concise a given library/framework/language is[34]. A screenshot of Cloc's usage is shown in Figure 3.6.

```
C:\Users\13037\Desktop\react-redux-realworld-example-app-master\src>cloc .
      38 text files.
      38 unique files.
       6 files ignored.

http://cloc.sourceforge.net v 1.64  T=0.08 s (489.6 files/s, 30162.9 lines/s)
-------------------------------------------------------------------------------
Language                     files          blank        comment           code
-------------------------------------------------------------------------------
Javascript                      38            285              8           2048
-------------------------------------------------------------------------------
SUM:                            38            285              8           2048
-------------------------------------------------------------------------------
```

Figure 3.6: Screenshot of the use of Cloc

3.3.5.3 Package Size

Smaller Package Size means faster download speeds and less parsing. Bundlephobia is a site where developers can find the cost of adding a npm package to the bundle. Upload the package.json of different front-end framework projects here and this site will automatically monitor the size of these dependencies[35]. A screenshot of the usage is shown in Figure 3.7.

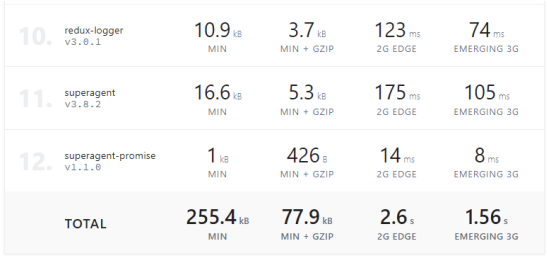

Figure 3.7: Screenshot of Package Size calculation using Bundlephobia

3.3.5.4 Memory Usage

One of the metrics is to compare the memory footprint of running different framework implementations of the Realworld Application. A high footprint may make the computer lag and run poorly. In this paper, the heap snapshot under the memory tab of Google Developer Tools will be used to show memory distribution among the index page's JavaScript objects and related Dom nodes. See Figure 3.8 for a screenshot of the memory usage when running the software.

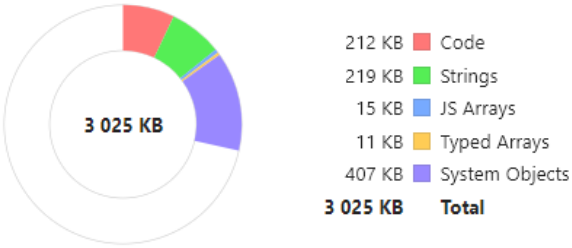

Figure 3.8: Screenshot of memory usage when running software

With the appropriate metrics to be measured, this paper will obtain data, explore the meaning behind the data, and analyze the reasons in the implementation section.

3.4 Local Dom Benchmarks

The tool for local Dom benchmarks and which local operations will be benchmarked for each framework are described in this section.

The tool chosen to test each framework Dom operations is a benchmarking software VanillaJs. The benchmark creates a large table with random entries and measures the timing of various operations, including rendering durations. A screenshot of the software is shown in Figure 3.9.

The VanillaJs measurement timeline is a custom solution that uses Selenium Webdriver to measure the duration from the start of the "EventDispatch" to the end of the next "Paint" from Chrome's timeline reporting raw performance log entries and uses the Aurelia framework to wait for the counter to trigger, then updates and re-renders Dom [36].

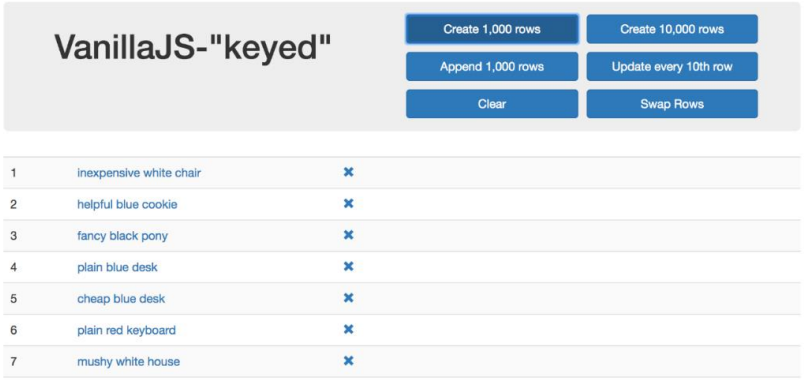

Figure 3.9: VanillaJs benchmarking software

The following operations are selected for each framework being benchmarks:

- Creating 1000 rows
- Creating 10000 rows
- Updating the 10th row in 10000 rows
- Removing a row
- Updating 1000 rows
- Deleting 10000 rows

This paper will obtain data, explore the meaning behind the data, and analyze the reasons in the implementation section.

4 Data analysis

In this section, the performance of the Realworld Application using different front-end frameworks is first compared to the features of these front-end frameworks in a networked, medium-sized forum-type project. This is followed by a summary of the characteristics of each framework and an analysis of the more suitable projects for the various frameworks, which is a realistic guide for developers. The front-end frameworks are then measured in local benchmarks, which measure the time taken to perform a series of Dom operations using the front-end frameworks in relation to additions, deletions and changes, and compare the time taken to perform the operations in each front-end framework. This paper will provide insight into the architectural features of the front-end frameworks that account for the differences in the time taken for each operation, and provide recommendations for developers to choose a framework based on the characteristics of their development projects. The results of the two experiments will subsequently be analysed and compared to see if there is consistency in the performance of the frameworks. The local benchmarks can also be considered as a validation and complement of the Realworld Application results.

4.1 Networking Realworld Application

4.1.1 Performance

When measuring performance of each framework, the average of 10 measurement scores was used as the final performance for the analysis in this paper, and the results are shown in Figure 4.1. The average for each parameter's results over the 10 measurements of per framework, which are shown from Figure 4.1 to 4.7, is also calculated.

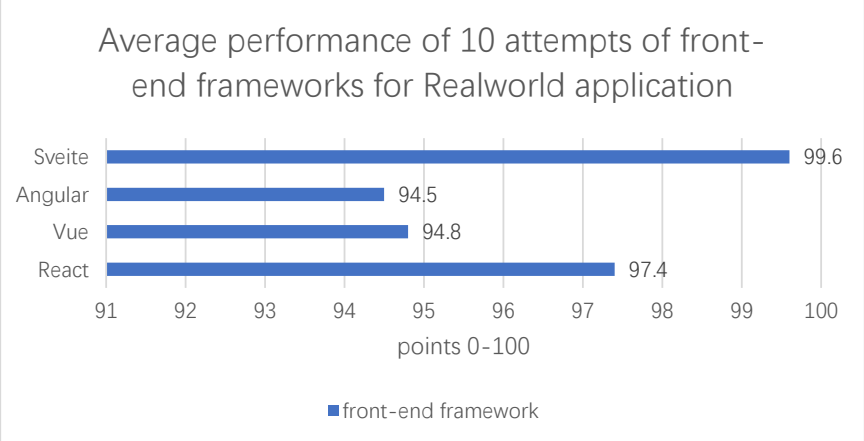

Figure 4.1: Average performance of 10 attempts of front-end frameworks for Realworld application using Lighthouse

The X scale here does not begin at 0, which highlights the differences between the measures and makes the discrepancies look bigger than they are. From the Lighthouse

results, the Svelte's implementation of Realworld Application was the most impressive, with a score of 99.6. Angular and Vue were relatively low, with scores of 94.5 and 94.8 respectively, while React scored in the middle.

The next section provides a step-by-step analysis of how the frameworks performed in each of the parameters provided by Lighthouse.

Table 4.1: Average value for 10 First Contentful Paint Time attempts of each framework

Framework	React	Vue	Angular	Svelte
Average	0.76s	0.59s	0.62s	0.56s

Table 4.2: Average value for 10 Speed Index Time attempts of each framework

Framework	React	Vue	Angular	Svelte
Average	0.76s	0.59s	0.6s	0.57s

In these two tables, First Contentful Paint marks the time at which the first text or image is painted and Speed Index shows how quickly the contents of a page are visibly populated. These two metrics capture more of loading experience after the initial paint. From the results, it can be seen that Lighthouse gives all four front-end frameworks a green colour, which means that all four frameworks perform well and give the user a comfortable experience. It can be said that the four frames selected for this paper are highly consistent in their performance on these two parameters with very little difference, so after initial painting, all four frames give the user a fast-loading experience.

Table 4.3: Average value for 10 Largest Contentful Paint Time attempts of each framework

Framework	React	Vue	Angular	Svelte
Average	1.04s	0.74s	0.84s	0.73s

The two metrics mentioned earlier - First Contentful Paint and Speed Index do not identify when the main content of the page has According to W3C Web Performance, a more accurate way to measure when the main content of the page has loaded is to see when the largest element is rendered. The Largest Contentful Paint parameter marks the time at which the largest text or image is painted. The performance of the four frameworks on this parameter remains in the same order as the previous two: Svelte takes the least time at 0.73s Angular is a little higher, while React takes the longest at 1.04s. From these three parameters, it can be concluded that Svelte and Vue load the main content of the page the fastest, Angular takes the average time and React the slowest.

Table 4.4: Average value for 10 Time to Interactive attempts of each framework

Framework	React	Vue	Angular	Svelte
Average	0.91s	0.73s	0.9s	0.64s

Time to interactive is the amount of time it takes for the page to become fully interactive. This is an important indicator because some sites optimize content visibility at the expense of interactivity. This can create a frustrating user experience: the site

appears to be ready, but when the user tries to interact with it, nothing happens. In this indicator, the performance rankings of several frameworks are still consistent with the first three. Svelte and Vue are in moderate range, which is from 0.39 to 0.73s. React and Angular are slower, and the time exceeds 0.73s. Relatively speaking, Svelte and Vue will give users a fast interactive experience. However, in the feedback given by Lighthouse, several frameworks are coded green in the Largest Contentful Paint Time metric, so the difference in usage time makes little distinction to the user.

Table 4.5: Average value for 10 Total Blocking Time attempts of each framework

Framework	React	Vue	Angular	Svelte
Average	7ms	0ms	24ms	0ms

Total Blocking Time measures the total amount of time that a page is blocked from responding to user input, such as mouse clicks. Vue and Svelte barely detect any blocking, while Angular's 24ms is relatively long. However, blocking times between 0-300ms are very responsive. In this respect, these frameworks perform well.

Table 4.6: Average value for 10 Cumulative Layout Shift Scores attempts of each framework

Framework	React	Vue	Angular	Svelte
Average	0.0055s	0.7034s	0.7068s	0.0031s

Cumulative Layout Shift measures the sum of layout shift scores for every movement of visible elements within the viewport during the lifespan of the page. Unexpected Movement of page content sometimes cause annoying experience or even damage. The score of this parameter for React and Svelte is less than 0.1, These two are in the range of good performance. On the contrary, if this parameter is greater than 0.25, the performance is within the range of poor, and the values of Vue and Angular are both above 0.7, which is almost three times the boundary value. It can be seen that the layout of React and Svelte is more stable, while the layout of Vue and Angular is prone to change.

4.1.2 Line of code

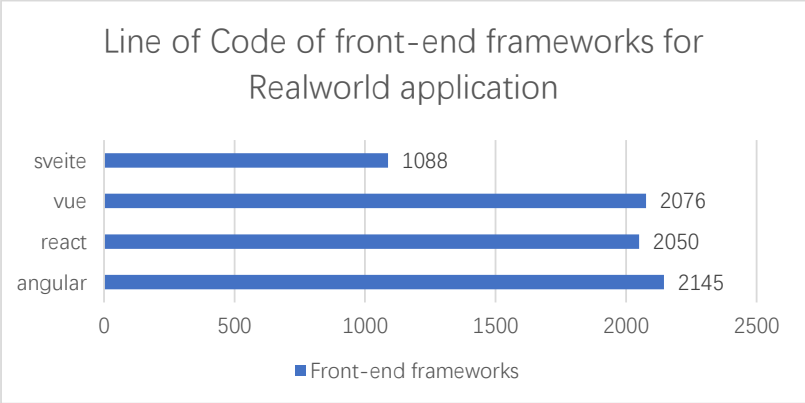

Figure 4.2: Line of code of different front-end frameworks for Realworld application

Line of code indicates how succinct given framework is. It is intuitive to see how many lines of code each framework requires to implement an almost identical application according to the specification. The difference between Vue, React and Angular is not significant, at approximately 2100 lines, while Svelte has almost half the code lines of the other three frameworks. This shows that Svelte is well encapsulated and can achieve the same functionality with fewer lines of code [37], while Vue, React and Angular have similar levels of code simplicity.

4.1.3 Package size

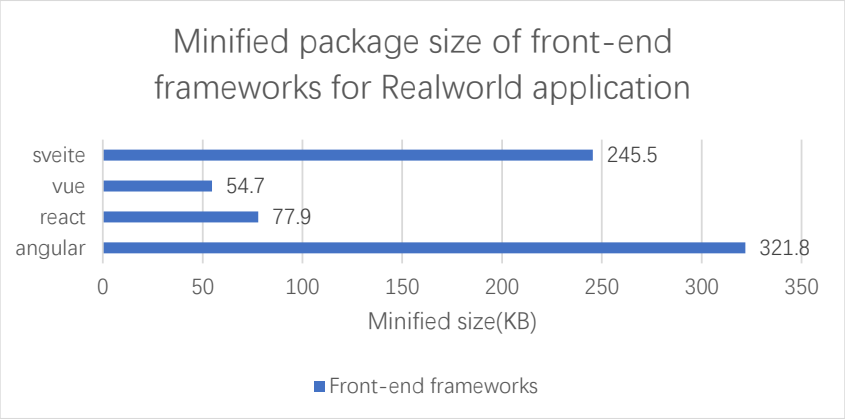

Figure 4.3: Size of each framework's minified package in the npm registry

The minified package size results for the Realworld Application implementation using each front-end framework in the npm registry are shown in Figure 4.3, where the size data is obtained by testing the dependencies listed in package.json. The size of the package corresponds to the amount of functionality [38]. When developing, programmers can choose a framework according to the volume of software to be implemented. The graphs show that minified package size of each framework compressed by Gzip is less than 350kB. The results are clearly divided into two groups: Vue's size is the smallest at 54.7kB, React is similar at 77.9kB. By contrast, Angular's size is the largest at 321.8KB and Svelte is also relatively heavy at 245.5kB. The comparison of minified package sizes reflects the fact that Svelte and Angular support include a larger spectrum of functionality, while Vue and React are designed for more streamlined development that is better suited to small and medium sized projects.

4.1.4 Memory

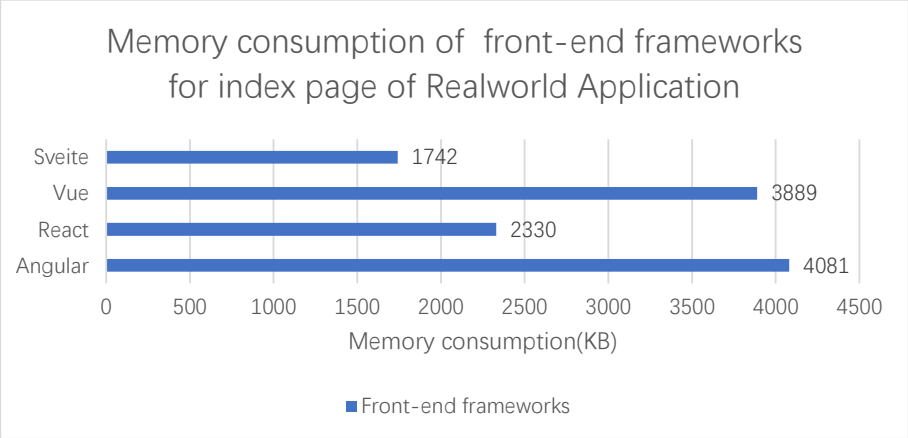

Figure 4.4: Memory consumption of front-end frameworks for index page of Realworld application

The memory consumption for running the Realworld Application home page varies between frameworks, these memory consumptions are for code, Strings, Js Arrays, Typed Arrays and System Objects. The comparison of memory consumption shows that Svelte consumes the least amount of memory at 1742kB and React consumes around 600kB more. Angular consumes the most memory (4081kB), which is twice as much as Svelte and 600kB more. Vue is only 200kB less than Angular. The reason for the big difference is that Svelte and React are relatively lightweight frameworks that focus on reducing development complexity. Vue and Angular, on the other hand, are much heavier in comparison and will have increased processing power.

In summary, after implementing Realworld Application with different front-end frameworks, Vue and Svelte have a better performance in terms of page load time and Time to Interact, which can bring users a faster interactive experience. React and Svelte do an excellent job of stabilising pages, with far fewer modules floating around than Angular and Vue, allowing for a more stable interaction experience for users.

For programmers, the fact that Svelte can achieve the same functionality in far fewer lines of code than other frameworks means that it does a remarkable job of packaging functionality. In addition, Angular and Vue consume more memory, while Svelte and React have a smaller memory footprint. This also implies that Angular and Vue are more heavyweight and can implement more features, while Svelte and React are more lightweight and more suitable for rapid development of small and medium-sized applications [39].

4.2 Local Dom benchmarks

Table 4.7: Duration of different DOM operations

Duration of Dom operations(ms)	**React**	**Vue**	**Angular**	**Svelte**
Creating 1000 rows	204.7	131.8	145.4	125
Creating 10000 rows	1779.9	1103.9	1229.9	1192.6
Updating the 10^{th} row in 10000 rows	268.4	171.6	141.5	162.7
Removing a row	34.3	23.2	25.4	21.9
Updating 1000 rows	153.2	116.1	129.8	127.3
Deleting 10000 rows	155.8	127.7	233.0	139.7
Slowdown geometric mean	**1.53**	**1.05**	**1.21**	**1.07**

The method used to calculate the Slowdown geometric mean is as follows:

Slowdown: Duration of an operation/Fastest duration in that line.

Slowdown geometric mean: Multiply the slowdown values of the n elements in a column and multiply the resulting value by the root of the n [40].

It is worth noting that React was the worst performer in all the Dom operations performed in this article, except for the second lowest performance in Deleting 10000 rows. In particular, updating the 10th row in 10000 rows takes almost twice as long as Angular, which is the best performer in this operation. The cost of Virtual Dom is very high. In operations like Updating the 10th row in 10000 rows, removing a row, a substantial amount of work can be done by simply updating a single message binding, whereas Virtual Dom has to traverse the entire Virtual Dom and do the Diff algorithm. In this process, the process is passed down recursively until the target is updated. Because of the dynamic nature of JavaScript, React also introduces the concept of runtime scheduling concurrent time slicing for optimization [41]. However, this runtime scheduling scheme takes a long time to run, so React does not centre on speeding up Virtual Dom itself.

Vue, which also uses Virtual Dom, is the best performer overall, especially in bulk Dom operations. But its performance drops slightly in minor Dom operations, coming in third in updating the 10th row in 10000 rows. The reason for the drop in performance in update operations is the high cost overhead of Virtual Dom mentioned above, and the speed of Virtual Dom speeds up when Dom operations reach a certain volume, which is one of the reasons why Vue performs extremely well in bulk operations. And the main reason is also because it uses a completely static node structure to optimise Virtual Dom speed, without the need to recursively go down the hierarchy to perform Diff operations on sublists [42].

In the benchmark results set up for this article, Svelte's overall performance is very little different from the best Vue, and the value of the Slowdown geometric mean is only 0.2 larger than Vue's. Interestingly, although Svelte performs well in every Dom operation, it is still found to perform better in a small number of Dom operations than in a large number of Dom operations, as opposed to Vue. Svelte's compilation style is

to compile templates as imperative native DOM operations, which simply means that the code is compiled into miniature, frameless raw JS. Compared to Virtual DOM-based frameworks, this output does not require the diff/patch operations of the Virtual DOM and therefore has extremely fast performance.

The overall performance of running each Dom Operation with Angular rank in the middle, worse than Vue and React, but better than React. However, it is worth noting that it performs best in Updating the 10th row in 10000 rows. However, it was the slowest in the bulk delete operation, more than 100ms longer than the fastest Vue. Angular uses an MVVM framework that watches for data changes and retains references to the actual DOM elements, performing the corresponding operations when there are data changes. Dependency collection requires re-collection of dependencies on both initialisation and data changes, this also incurs some consumption when there is a large amount of data, but the cost is almost negligible for small updates, so it performs well for small update operations. The problem with the MVVM framework is how to effectively reuse already created ViewModel instances and DOM elements when the data source for list rendering changes. Without any reuse optimizations, MVVM would actually need to destroy all the previous instances, recreate all the instances, and finally perform another rendering, which is probably why the bulk delete operation takes a long time [43]. In terms of optimisation, Angular also provides a mechanism for list reentry to effectively reuse instances and Dom elements, so the rest of the operations take an acceptable amount of time.

In summary, it can be seen that React sacrifices the speed optimisation of Virtual Dom for more stable and secure performance. Vue and Svelte have a significant speed advantage, with Vue being better at batch operations and Svelte being better at small operations. Angular is generally fast, but has very impressive performance in modifying specific elements.

4.3 Conclusion

By comparing the performance of the Realworld Application using different front-end frameworks, we are able to get an insight into the performance of these front-end frameworks in a networked, medium-sized forum-type project, which is a more realistic guide for developers. The paper then goes on to measure local benchmarks for these front-end frameworks, measuring a series of Dom operations relating to additions, deletions and changes, and comparing the time taken by each front-end framework. This is a validation and complement to the Realworld Application results.

It is clear that the results of the two experiments are highly consistent, with Vue and Svelte delivering the fastest user interaction experience when implementing the Realworld Application. React is not as fast in Realworld Applications, but Realworld Applications implemented with it rarely suffer from Cumulative Layout Shift, indicating better stability. React is also the slowest in terms of time spent on the various Dom operations. However, the underlying analysis above suggests that the reason it is slower and more stable is because React's focus is on optimising JavaScript dynamics rather than improving the speed of Virtual Dom's Diff. Angular, on the other hand, is

in the middle of the range in terms of speed performance in both experiments.

In addition, Svelte's outstanding performance in terms of lines of code and memory consumption is also due to the fact that it compiles the templates as imperative native DOM operations, which saves a lot of code due to the fact that no diff/patch operations are required compared to Virtual Dom-based frameworks. Another reason is that when it is compiled, it will only compile the functional components that are used. As the number of project components increases, however, the number of lines of code variation becomes smaller and there is more code to run. This is why Svelte performs best in smaller projects. Angular, on the other hand, downloads the largest amount of data in the minified package and has the largest memory footprint, which also means that it is heavy, contains a larger range of features and is better suited for larger projects.

After experimentation and research, this thesis does a great job of answering the question of how to choose an appropriate front-end framework for developers for different web projects, when each framework has its own characteristics, by comparing the performance of different frameworks on each metric.

However, there are still limitations in this thesis; the Realworld Application is a medium-sized web project and the performance of the frameworks in other project sizes may still vary. In the future, I would like to build more projects with different front-end frameworks to deepen my learning of these frameworks through practice, and then use them to build other size of web projects to measure their performance in the metrics mentioned above and continue to explore the underlying reasons. Besides, I will continue my research and select more metrics for comparison to discover how different front-end frameworks perform in more aspects, and to provide developers with more characterized framework selection advice.

Bibliography

[1] A. Rauschmayer, *Speaking JavaScript: An In-Depth Guide for Programmers*. O'Reilly Media, Inc., 2014.

[2] Fu C., 'Exploration of Web front-end development technology and optimization direction', Sep. 2016, pp. 166–169. doi: 10.2991/icence-16.2016.35.

[3] D.-P. Pop and A. Altar, 'Designing an MVC Model for Rapid Web Application Development', *Procedia Engineering*, vol. 69, pp. 1172–1179, Jan. 2014, doi: 10.1016/j.proeng.2014.03.106.

[4] A. Vera-Baquero, O. Phelan, P. Slowinski, and J. Hannon, 'Open Source Software as the Main Driver for Evolving Software Systems Toward a Distributed and Performant E-Commerce Platform: A Zalando Fashion Store Case Study', *IT Professional*, vol. 23, no. 1, pp. 34–41, Jan. 2021, doi: 10.1109/MITP.2020.2994993.

[5] 'Cumulative Layout Shift (CLS)', *web.dev*. https://web.dev/cls/ (accessed May 16, 2021).

[6] M. Arruat *et al.*, *Front-End Software Architecture*.

[7] K. Gallagher, A. Hatch, and M. Munro, 'Software Architecture Visualization: An Evaluation Framework and Its Application', *IEEE Transactions on Software Engineering*, vol. 34, no. 2, pp. 260–270, Mar. 2008, doi: 10.1109/TSE.2007.70757.

[8] E. Molin, *Comparison of Single-Page Application Frameworks : A method of how to compare Single-Page Application frameworks written in JavaScript*. 2016. Accessed: May 16, 2021. [Online]. Available: http://urn.kb.se/resolve?urn=urn:nbn:se:kth:diva-194105

[9] B. Frain, *Responsive Web Design with HTML5 and CSS3*. Packt Publishing Ltd, 2015.

[10] P. Saxena, D. Akhawe, S. Hanna, F. Mao, S. McCamant, and D. Song, 'A Symbolic Execution Framework for JavaScript', in *2010 IEEE Symposium on Security and Privacy*, May 2010, pp. 513–528. doi: 10.1109/SP.2010.38.

[11] H. Lee, S. Won, J. Jin, J. Cho, and S. Ryu, *SAFE: Formal Specification and Implementation of a Scalable Analysis Framework for ECMAScript*.

[12] 'Safe & Efficient Gradual Typing for TypeScript | Proceedings of the 42nd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages'. https://dl.acm.org/doi/abs/10.1145/2676726.2676971 (accessed May 16, 2021).

[13] J. J. Garrett, 'Ajax: A New Approach to Web Applications', p. 5.

[14] L. D. Paulson, 'Building rich web applications with Ajax', *Computer*, vol. 38, no. 10, pp. 14–17, Oct. 2005, doi: 10.1109/MC.2005.330.

[15] J. Keith, *DOM Scripting: Web Design with JavaScript and the Document Object*

Model. Apress, 2006.

[16] 'Document Object Model †DOM‡ Level 1 Specification †Second Edition‡', p. 212.

[17] J.-P. Voutilainen, T. Mikkonen, and K. Systä, 'Synchronizing Application State Using Virtual DOM Trees', in *Current Trends in Web Engineering*, Cham, 2016, pp. 142–154. doi: 10.1007/978-3-319-46963-8_12.

[18] J. Li and C. Peng, 'jQuery-based Ajax general interactive architecture', in *2012 IEEE International Conference on Computer Science and Automation Engineering*, Jun. 2012, pp. 304–306. doi: 10.1109/ICSESS.2012.6269466.

[19] K. De Volder, 'JQuery: A Generic Code Browser with a Declarative Configuration Language', in *Practical Aspects of Declarative Languages*, Berlin, Heidelberg, 2006, pp. 88–102. doi: 10.1007/11603023_7.

[20] A. Nitze and A. Schmietendorf, *Modularity of JavaScript Libraries and Frameworks in Modern Web Applications*. 2014.

[21] A. Leff and J. T. Rayfield, 'Web-application development using the Model/View/Controller design pattern', in *Proceedings Fifth IEEE International Enterprise Distributed Object Computing Conference*, Sep. 2001, pp. 118–127. doi: 10.1109/EDOC.2001.950428.

[22] D.-P. Pop and A. Altar, 'Designing an MVC Model for Rapid Web Application Development', *Procedia Engineering*, vol. 69, pp. 1172–1179, Jan. 2014, doi: 10.1016/j.proeng.2014.03.106.

[23] A. Fedosejev, *React.js Essentials*. Packt Publishing Ltd, 2015.

[24] O. Filipova, *Learning Vue.js 2*. Packt Publishing Ltd, 2016.

[25] B. Green and S. Seshadri, *AngularJS*. O'Reilly Media, Inc., 2013.

[26] M. Levlin, 'DOM benchmark comparison of the front-end JavaScript frameworks React, Angular, Vue, and Svelte', 2020, Accessed: May 17, 2021. [Online]. Available: https://www.doria.fi/handle/10024/177433

[27] E. Mendes and N. Mosley, *Web Engineering*. Springer Science & Business Media, 2006.

[28] I. Lapsley, 'The Npm Agenda: Back to the Future', *Financial Accountability & Management*, vol. 24, no. 1, pp. 77–96, 2008, doi: https://doi.org/10.1111/j.1468-0408.2008.00444.x.

[29] 'Apache Hadoop YARN | Proceedings of the 4th annual Symposium on Cloud Computing'. https://dl.acm.org/doi/abs/10.1145/2523616.2523633 (accessed May 17, 2021).

[30] V. Subramanian, 'Modularization and Webpack', in *Pro MERN Stack: Full Stack Web App Development with Mongo, Express, React, and Node*, V. Subramanian, Ed. Berkeley, CA: Apress, 2017, pp. 115–150. doi: 10.1007/978-1-4842-2653-7_7.

[31] ‘State of JS 2020: JavaScript Flavors’. https://2020.stateofjs.com/en-US/technologies/javascript-flavors/ (accessed May 17, 2021).

[32] *gothinkster/realworld*. Thinkster, 2021. Accessed: May 17, 2021. [Online]. Available: https://github.com/gothinkster/realworld

[33] ‘Lighthouse | Tools for Web Developers’, *Google Developers*. https://developers.google.com/web/tools/lighthouse (accessed May 17, 2021).

[34] AlDanial, *AlDanial/cloc*. 2021. Accessed: May 17, 2021. [Online]. Available: https://github.com/AlDanial/cloc

[35] ‘BundlePhobia’. https://bundlephobia.com (accessed May 17, 2021).

[36] S. Krause, *krausest/js-framework-benchmark*. 2021. Accessed: May 17, 2021. [Online]. Available: https://github.com/krausest/js-framework-benchmark

[37] G. Singh, D. Singh, and V. Singh, ‘A Study of Software Metrics’, vol. 11, p. 6, 2011.

[38] P. Goswami, S. Gupta, Z. Li, N. Meng, and D. Yao, ‘Investigating The Reproducibility of NPM Packages’, in *2020 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, Sep. 2020, pp. 677–681. doi: 10.1109/ICSME46990.2020.00071.

[39] A. E. Hassan and R. C. Holt, ‘A lightweight approach for migrating web frameworks’, *Information and Software Technology*, vol. 47, no. 8, pp. 521–532, Jun. 2005, doi: 10.1016/j.infsof.2004.10.002.

[40] A. Mutapcic, S. Murali, S. Boyd, R. Gupta, D. Atienza, and G. D. Micheli, ‘Optimized Slowdown in Real-Time Task Systems via Geometric Programming’, p. 4.

[41] D. T. Kudiabor, ‘State management with React-Redux’, 2020. http://www.theseus.fi/handle/10024/355184 (accessed May 18, 2021).

[42] S. bin Uzayr, N. Cloud, and T. Ambler, ‘Vue.js’, in *JavaScript Frameworks for Modern Web Development: The Essential Frameworks, Libraries, and Tools to Learn Right Now*, S. bin Uzayr, N. Cloud, and T. Ambler, Eds. Berkeley, CA: Apress, 2019, pp. 523–539. doi: 10.1007/978-1-4842-4995-6_14.

[43] E. Wohlgethan, ‘Entscheidungshilfe für die Webentwicklung anhand des Vergleichs von drei führenden JavaScript Frameworks: Angular, React and Vue.js’, p. 84.