International Journal on Recent and Innovation Trends in Computing and Communication
ISSN: 2321-8169 Volume: 10 Issue: 9
Article Received: *25 July 2022* ***Revised:*** *12 September 2022* ***Accepted:*** *30 September 2022*

Migrating Legacy Angular JS Applications to React Native: A Case Study

Manasa Talluri

Independent Researcher, Usa.

Abstract

Despite JavaScript libraries and frameworks growing in number over time, it is increasingly difficult for both experienced and inexperienced developers to choose the appropriate one. There is currently no explicit review process for developers to ensure that a framework matches the scope and objectives of their project. In order to help developers choose the best framework for their project depending on their tastes and level of experience, this study looks at the evaluation process. Because component-based React projects and traditional AngularJS apps have distinct ideas, patterns, and structures, the transfer is challenging. This study outlines an easy, affordable, and efficient way to switch from AngularJS to React apps. The research provides a framework with comprehensive instructions for transforming the display layer of any conventional AngularJS application into a component-based React application, while also comparing various approaches. Ng-React Copilot, a migration support tool, was used to teach developers and automate the migration. The tool was created by converting the framework's suggested major refactorings into a collection of detection methods and activating scanning against the specified codebase. The utility works with popular integrated development environments and may be used as a command-line tool. The framework and tool were tested on a subset of small, medium, and enterprise-level AngularJS legacy applications, and the findings demonstrate that the study's conclusions are accurate.

Keywords: - JavaScript Libraries, AngularJS Application, Architecture, , Cost-Effective, Algorithms, Migration Assistant, Ng-React Copilot,

I. INTRODUCTION

AngularJS was a comprehensive framework that made it possible to create web applications quickly. Following the MVC design enabled developers to incorporate the same ideas found in server-side frameworks into client-side programming. Because AngularJS was used to build sophisticated apps, the framework's shortcomings became apparent within a few years [1]. Some of these problems resulted from the way the framework was embraced by the community, while others were caused by the framework's fundamental design.

The scope and two-way data binding were extensively used to facilitate communication between the controller, the template, and other crucial AngularJS components like Directives since the framework promoted MVC design [1, 2]. Scope Soup code was recognised as this anti-pattern. Although AngularJS's two-way data flow initially seemed promising, it made it very difficult to anticipate data modifications in big applications. Additionally, the templates were making the program less predictable.

Front-end frameworks and libraries have proliferated in the area of web development in recent years, giving programmers the tools they need to construct dynamic and effective online applications. Notably, React and Angular have become popular choices, each with unique benefits and capabilities. Facebook created the JavaScript framework React, which uses a virtual Document Object Model (DOM) for efficient rendering and stresses a component-based design.

On the other hand, Google's all-inclusive framework Angular offers a more complete solution with integrated services and application architecture guidelines. Because of the widespread use of these technologies and the continuous debate about their relative advantages, understanding the differences between them is crucial for developers and enterprises looking to create scalable and maintainable systems [2, 3]. This thesis' main goal is to

provide a comparative examination of React and Angular, supported by real-world knowledge gained via creating two web applications: one using React and the other using Angular.

By analysing their advantages, disadvantages, and performance in several areas, this research aims to provide a sophisticated understanding of how these frameworks align with various project requirements and development contexts.

Java script has gained popularity since HTML's revolution. The fact that client-side scripting is just as crucial as server-side scripting is widely acknowledged. When smartphones and other mobile devices were introduced, cross-platform application compatibility became a difficult topic. A growing number of new systematic JavaScript scripts were created as a result of this need. They included jQuery. As the most dependable and quick JS library, jQuery contributed to the reduction of code lines [3, 4]. Soon, this turned into a revolution. Numerous jQuery plug-ins were made available, each capable of solving a thousand different issues in a thousand different ways. To improve Java Script's resilience, experience, and productivity, Google released the AngularJS framework. Since AngularJS is MVC, it has received a lot of praise and criticism so far.

Client-side front end developers have taken notice of the declarative programming approach it introduced, and as a result, the majority of businesses have already ceased using jQuery and are switching to AngularJS [3, 4]. In the user's perspective, AngularJS functions similarly to an application and doesn't need additional server-side interaction to calculate the front-end code. For single-page web applications that need to function really well, AngularJS is popular.

In almost every developer forum, one of the most recent subjects of discussion is AngularJS vs. jQuery. Since both frameworks are front end orientated and optimisation and compression are methods that may be used to enhance speed rather than migrate, some individuals argue that it is not worth switching from jQuery to AngularJS. Others argue that with the growing popularity of MVC-based frameworks, AngularJS is the ideal moment to replace jQuery, which has grown outdated.

Today's most widely used browser programming language, JavaScript, was created especially for creating event-based user interfaces that work in a web browser. For this reason, they are best suited to create client-side applications [6, 7]. In recent years, they have also evolved into a general-purpose computing platform that works with browsers as well as other programs like office suites and Rich Internet Application frameworks like Google Web Toolkit, Qooxdoo.org, Cappuccino.org, and others.

It had been seen that in the research conducted, they compared jQuery library with BackboneJS in 2015. They created an easy-to-use application utilising three distinct frameworks: BackboneJS, jQuery, and plain JavaScript. In order to make the browser load faster since there would be less space, they utilised compressed versions of all frameworks [7, 8]. Simple CRUD tasks, such as creating, reading, updating, and deleting forms inside the application, were to be carried out by it.

When applied to my situation, the 2013 study was an intriguing endeavour. In a real-time setting, he attempted to contrast the JavaScript BackboneJS and AngularJS frameworks. He used the open-source to-do MVC application with HTML local storage as the foundational layer for his testing. using the aid of test environments constructed using PhantomJS, the DOM performances of the two comparable apps were examined [5, 6]. According to his investigation, AngularJS performs better than BackboneJS when handling several DOM entries at once, but it exhibits a delay while handling a single item in a DOM 1000 times.

Some developers at Stack Overflow, the largest online community of developers from around the world who support and contribute to each other's work, have expressed a slight concern that using a lot of these plugins in an application could make it weak overall because not all the libraries will always adhere to each other firmly, even though jQuery is one of those frameworks that makes programming easier [5, 6].

Instead of being a library, AngularJS is a JavaScript framework that facilitates the extension of HTML into a more legible and expressive language. Angular HTML is embellished with unique markup that works in tandem with JavaScript [5, 6]. As a result, you may stop manually changing views and concentrate on building your application logic. Angular is perfect for rich interactive frameworks and contributes to cleaner, more efficient code.

Numerous JavaScript frameworks are available. jQuery is the most popular. The most popular framework for UI

International Journal on Recent and Innovation Trends in Computing and Communication
ISSN: 2321-8169 Volume: 10 Issue: 9
Article Received: *25 July 2022* ***Revised:*** *12 September 2022* ***Accepted:*** *30 September 2022*

design is this one. completed a stunning analysis of the jQuery framework's performance. They have effectively raised awareness of the benefits and drawbacks of jQuery in an assessed manner.

1.1 Migration strategy

Several well-known code migration techniques are used in the industry, including the Big-Bang approach, which calls for converting the entire codebase from the old technology to the new technology in a single attempt, the Side-by-Side approach [5, 6], which splits the old and new applications at the entry level so that they can run side by side, and the Bottom-Up approach, which enables developers to gradually incorporate the new technology by applying it to small portions of the application where the changes can be isolated without affecting the entire application.

1.2 Prepare AngularJS code for migration

The view layer of the legacy application, which consists of templates and directives, is the simplest code to begin the conversion, according to AngularjS migration projects that have already been completed. AngularJS directives are more similar to React components [7, 8]. The templates may be readily converted to components by dividing them up into appropriate directives. These little adjustments made to the program make its code easier to work with without altering its functionality.

When a directive is converted to a single React component, first group all of its code together so that old code may be removed and transferred simultaneously [7, 8]. Next, because React is always element-based, prefer directives as elements to attributes, comments, or classes. React encourages the usage of plain Java script for converting custom directives [23, 24]. The second option is to do away with Angular translation, which is a complex idea in AngularJS and is absent from React.

Previous study advises removing it as soon as feasible since it makes the template behaviour very wasteful. In a similar manner, a distinct HTML template may be injected into a parent template using the ng-include directive, indicating that the injected HTML is a stand-alone piece of user interface [8, 9]. Accordingly, earlier studies recommend isolating the Ng and including UIs as directives.

The converted component's AngularJS-specific functionalities should be replaced [9, 10]. It is necessary to substitute JS-specific features or their React counterparts for the many AngularJS capabilities included in the directive templates. The options are shown in the following Table 1.

Table 1 Other Options for AngularJS Functionalities. [22]

AngularJS	React
$http	Fetch or Axios
$q	Native support in ES6
$timeout/$interval	Native timeout
$document/ $window	Native document and window
$formats/$validators/$parsers	Vanila JS to format, Validate and parse data.
Ng-form, Filters, ng-extend, Angular. merge	Vanila JS
Angular. Element	React-dom or jQuery
Angular. For Eac, angular. Extend, angular. Merge	Lodash
Ng-class, ng-show, ng-if, ng-hide	Vanilla JS conditions
Ng-minlength, ng-maxlenght, ng-disable, ng-required	Minlength, Maxlenght, disable and Required Default HTML Validation
ngRoute/angular-Ui-router	React router
Protractor	Enzyme, Nightwatch.js
$httbackend , Dependency injection	Jest or Re-wire
Ng-init, $onDestroy	Component Did Mount(), Component Will Unmount()
Ng-model	Default Value
Ng-click	OnClick

Directive bindings may be immediately transformed to props in the component by using props to specify the inputs and outputs of a React component. It is advised to declare props in the PropTypes section, which aids in program validation and helps programmers determine the inputs and outputs the component supports. The directive logic must then be migrated into the component when the props have been defined [9, 10].Because

International Journal on Recent and Innovation Trends in Computing and Communication
ISSN: 2321-8169 Volume: 10 Issue: 9
Article Received: *25 July 2022* ***Revised:*** *12 September 2022* ***Accepted:*** *30 September 2022*

AngularJS and React have separate lifecycle event systems, it is important to properly move the code into AngularJS components. These factors make moving the logic the most difficult and error-prone part of the whole conversion process, thus it should be done carefully [12, 13].

React2Angular middleware may be used to wrap a React component in a pseudo-Angular directive after the React component is complete. This allows the AngularJS application to connect with an Angular directive rather than a React component [9, 10]. React components may work together effortlessly inside the AngularJS framework thanks to the pseudo directive. The Angulag2React middleware's functionality is shown in Figure 1.

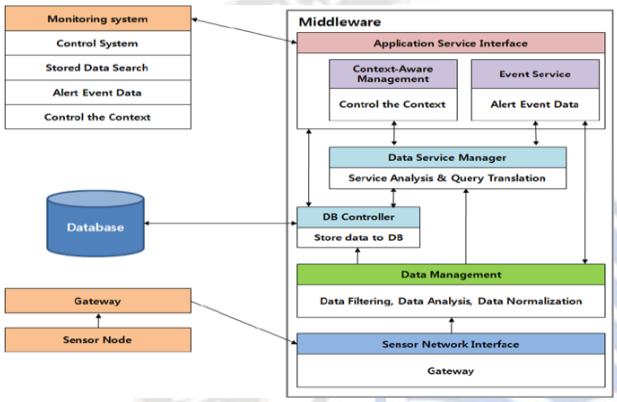

Fig. 1 How React2Angular middleware facilitates data flow. [22]

When an excessive number of characteristics are sent down via the intermediate components, separate the container components from the presentational ones. The presentational components are in charge of handling user interactions and rendering the user interface [9, 10]. Typically, container components transmit information to presentational components after communicating with backend servers and services [20, 21]. Container components are mostly located on top of the component tree and are primarily concerned with how things operate, whereas presentational components are located at the bottom of the tree and are exclusively concerned with appearance.

II. METHODOLOGY

2.1 Choosing the migration strategy

In the highly competitive web application sector, the Big-Bang strategy may be quite risky. The stability may be impacted by the massive codebase duplication caused by the side-by-side method. One of Angular and React's primary benefits is that React can easily coexist with AngularJS by acting as the View layer. Additionally, React is a fairly lightweight library [11]. Taking into account each of these strategies, the Bottom-Up strategy was selected with less migration-related repercussions.

2.2 Migration framework

When transitioning AngularJS apps to React, the 23-step incremental modifications covered in the literature study section may be used as a generic kind of refactoring. To create a framework that was both relevant and simple to recall, those refactorings were grouped [12]. The best grouping criteria, given the environment in which this framework is used, were to categorise them according to their function and the time they are used. The corresponding groups and refactorings are shown below [20].

Group 1: Break the application to directives

Refactoring #1: *Put all of a directive's code in one place.*

Refactoring #2: *Give preference to directives as elements over classes, comments, and attributes.*

Refactoring #3: *Get rid of transclusion*

Refactoring #4: *Use a directive in lieu of ng-include.*

Refactoring #5: *Extract a directive's root controller*

Refactoring #6: *As a directive, extract the controller.*

Refactoring #7: *Take a directive out of big templates.*

Group 2: Define the input/outputs and isolate the directive

Refactoring #8: *Change the external reference to bound inputs.*

Refactoring #9: *Use bound outputs in lieu of the external impact.*

Refactoring #10: *Separate the scope of the directive*

Refactoring #11: *Use bound output in lieu of state mutation.*

Refactoring #12: *Use one-way data binding instead of two-way data binding.*

Group 3: Convert the directives to React components

International Journal on Recent and Innovation Trends in Computing and Communication
ISSN: 2321-8169 Volume: 10 Issue: 9
Article Received: *25 July 2022* ***Revised:*** *12 September 2022* ***Accepted:*** *30 September 2022*

Refactoring #13: *In the React component, convert the directive template to JSX.*

Refactoring #14: *Change the characteristics unique to AngularJS that were used in the converted component.*

Refactoring #15: *The directive bindings should be moved as component props.*

Refactoring #16: *Transfer the component's directive logic.*

Group 4: Bind the React component to AngularJS

Refactoring #17: *Use the React2Angular middleware to bind the React component to AngularJS.*

Group 5: Miscellaneous changes

Refactoring #18: *When too many characteristics are sent down via the intermediate components, separate the presentational components from the container components.*

Refactoring #19: *Import the angular service into the container component after converting it to a JavaScript module.*

Refactoring #20: *Import the angular service into the container component after wrapping it as a JavaScript module.*

Refactoring #21: *Present a UI framework that offers form widgets that are preferred by React.*

Refactoring #22: *Use custom directives to migrate widgets to components.*

Refactoring #23: *Transfer validations, sanitisation, and formatting to services*

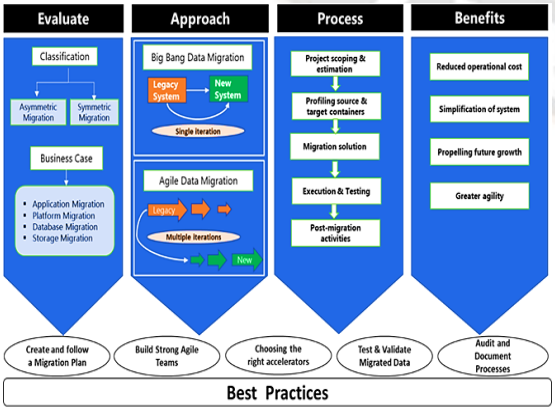

Fig. 2 The migration framework's workflow. [14]

III. IMPLEMENTATION

3.1 Automating the migration workflow

There are many degrees of automation that allow the process to be completely automated without the need for human involvement. Semi-automation is chosen as the method to automate the workflow in accordance with the project's scope in order to prevent unforeseen faults from being introduced into automated code [13]. One refactoring, #13, was discovered as a potential code generation, and nine refactorings, #1, #2, #3, #4, #6, #7, #12, #14, and #15, were identified as potential refactorings to detect.

3.2 Consolidated migration assistance tool

The AngularJS to Angular conversion process is automated using an existing solution called Angular Copilot. It had an impact on the development of the "Ng-React Copilot" AngularJS to React migration tool.

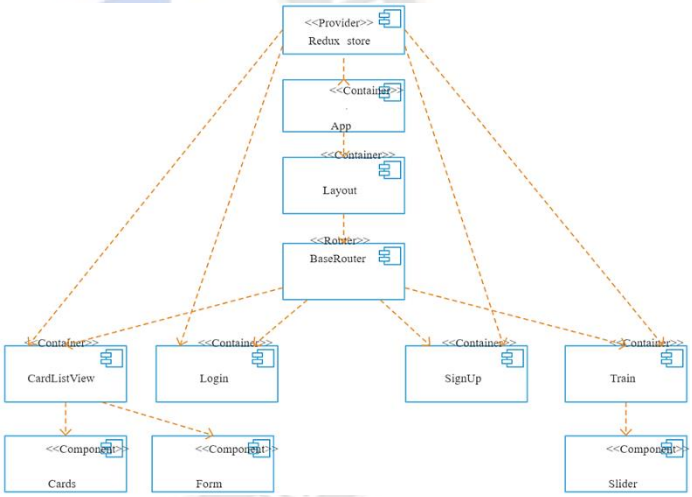

Fig. 3 Diagram of components for Ng-React Copilot. [16]

The Ng-React Copilot tool is available for direct download from GitHub for developers to use. In order to start the tool as "node index.js," the developer might provide the directory path to scan as the first parameter. The three scan modes that Ng-React Copilot operates in are code creation, analysis, and data extraction [11]. The project's metadata is extracted in the Extract Data mode and utilised in the Analysis mode. Analyse mode scanning is used by the program after data extraction. After the scan is finished, [12], the application will compile all of the findings and provide the user with a summary of the findings.

International Journal on Recent and Innovation Trends in Computing and Communication
ISSN: 2321-8169 Volume: 10 Issue: 9
Article Received: *25 July 2022* ***Revised:*** *12 September 2022* ***Accepted:*** *30 September 2022*

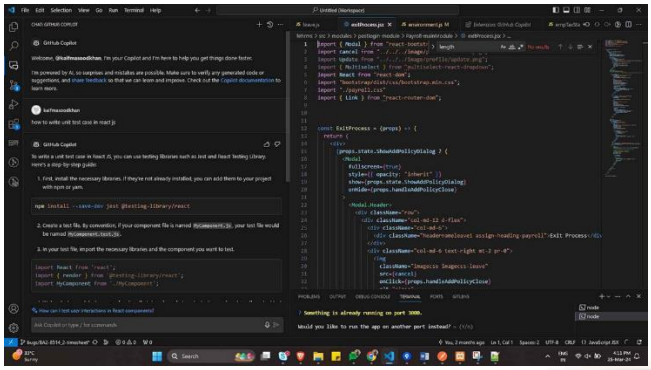

Fig. 4 Summary report for Ng-React Copilot. [18]

IV. RESULT AND EVALUATION

Based on their size and breadth, we established three tiers of benchmark apps to assess the Ng-React Copilot tool. These categories are,

- Small apps with simpler AngularJS code are an example. For this category, benchmarking apps are chosen from among the AngularJS PhoneCat applications available on the official AngularJS websites [13].
- Medium level programs, which are full applications designed for server-specific uses rather than starter apps. "Angular-app" is chosen as the benchmarking application for this category from among the AngularJS apps posted on GitHub [17, 18].
- Candidates for this category include enterprise-level applications with bigger and more complex code bases.

Orange HRM Enterprise was chosen as the benchmarking application for this category since it is an HR solution that is based on AngularJS. The refactorings found at each level of the migration are summarised in the following Table 2.

Table 2 An overview of every detected refactoring.

Total files	**Applications Levels**		
Refactoring occurrences	**Small**	**Medium**	**Enterprise**
Refactoring #1	26	88	1001
Refactoring #2	42	45	631
Refactoring #3		-	-
Refactoring #4		-	-
Refactoring #6			
Refactoring #7		-	-
Refactoring #12		-	-
Refactoring #13		-	-
Refactoring #14	-	-	-
Refactoring #15	-	-	-

4.1 File search efficiency of Ng-React Copilot

The tool's speed was within an acceptable range since the scan times for all three instances were less than 10 seconds. There was a problem since it produced a lot of false positives by failing to exclude library files that weren't in the designated directories [16].

4.2 Migration flow

Compared to enterprise-level apps, the migration of small applications was very simple. The ability of simple migrations is diminished as logic and code complexity rise.

4.3 Speed of migration

The ability to automatically construct the React component from the directive template and highlight the AngularJS-specific usages is the most effective automation step of the Ng-React Copilot tool. Refactoring the current AngularJS code to create separate directives takes up most of the work [16, 19]. The migration accelerated when the isolated directions were withdrawn [16]. Developers must address the current scope soup inefficiencies in order to create separated directives from AngularJS code.

4.4 Accuracy of Ng-React Copilot

During the scanning process, several false positives were found, despite the refactoring detection's generally high accuracy. All instances of the ng-controller directive's use were highlighted in "Refactoring #6," although some of them are required at the AngularJS component tree's root level. The Ng-React Copilot tool helped with "Refactoring #15," however it identified the variables as false positives when they were within JSX in-line routines.

V. CONCLUSION

A framework and tool for moving the display layer of older AngularJS apps to more recent React-based apps are proposed in this project. Application migration at the

small, medium, and enterprise levels was used to test the suggested framework and tool, and the findings indicate that the suggested approach works well at all three levels. The automation tool has to be enhanced to include the manual refactorings that were not discovered in the framework in order for it to be helpful in enterprise-level application migration, even if the framework meets the majority of the requirements for such migrations.

It is necessary to manually configure the Ng-React Copilot. The Ng-React Copilot tool, which displays all of the refactorings simultaneously, does not represent the migration procedure that the framework offers. This leaves the user unsure of where to begin the transfer procedure. The AngularJS application's leaf nodes cannot be automatically identified in order to initiate the transfer. Most time-consuming phases in the business application conversion process, such converting controllers to directives, preventing scope soup in the code, isolating directives, and moving the functionality of the directives, are not automated by the tool. The framework leaves it up to the developer to make judgements since it offers no ways to prevent coupling between external libraries.

It is necessary for the developer to determine whether or not the false positives found in the NgReact Copilot tool are false positives. It is ineffective and misleading. React's methods vary from the direct DOM manipulations used in other libraries like Jquery-based controls, making the conversion challenging when AngularJS UI relies on them. With incompatible JS event systems, the same problem existed. Since they are beyond the purview of this study, the JSX syntaxes and the rules for the newest React features, including React hooks, will not be covered in comprehensiveness.

VI. FUTURE WORK

The following ways have been suggested to get around the restrictions. To allow for customisation according to the kind of application, the Ng-React Copilot tool should be improved to take settings via a configuration file. Enhancing the tool to detect the use of the AngularJS directive in the code is preferable. To begin the migration process, make a component map and highlight the leaf nodes. There should be further investigation on the most popular couplings between AngularJS and other libraries, as well as some recommendations for standardising their refactoring. For instance, Jquery and AngularJS are used in the majority of apps; more study is needed to determine how to transfer such direct DOM manipulation functionality into React. That information needs to be included in the Ng-React Copilot tool.

The Ng-React Copilot tool's detection algorithms need to be enhanced in logic to prevent false positives. To provide the developer a clear picture of the effort, it would be beneficial if the tool could estimate the number of lines in the code that would be associated with the identified refactorings. To identify further framework and tool restrictions and enhancements, the assessment should be carried out by running enterprise applications through their whole lifecycle from bottom to top. For instance, there are no instructions in the suggested framework on how to switch an AngularJS application's routing to React. Currently, developers are free to choose how to set up React and other necessary libraries in an AngularJS application, since this framework does not cover these steps or instructions.

REFERENCES

[1] David Flanagan. JavaScript: The Defnitive Guide, 7th Edition. O'Reilly Media, Inc., 2020.
[2] JavaScript: The Good Parts. O'Reilly Media, Inc., 2008.
[3] Axel Rauschmayer. Exploring ES6 Upgrade to the next version of JavaScript. 2018.
[4] T. J. Crowder. JavaScript: The New Toys. Wrox, 2020.
[5] Koskinen, I., Zimmerman, J., Binder, T., Redström, J., & Wensveen, S. (2011). *Design research through practice: From the lab, field, and showroom*. Morgan Kaufmann.
[6] M. A. Putri, H. N. Hadi, and F. Ramdani. Performance testing analysis on web application: Study case student admission web system. pages 1–5, 11 2018.
[7] J. Voutilainen. Evaluation of front-end javascript frameworks for master data management application development. pages 4–6, 12 2017.
[8] Y. K. Xing, J. P. Huang, and Y. Y. Lai. Research and analysis of the front-end frameworks and libraries in e-business development. pages 68–72, 08 2019.
[9] Rafael Auler, Edson Borin, Peli Halleux, Michal Moskal, and Nikolai Tillmann. Addressing javascript jit engines performance quirks: A crowdsourced adaptive compiler. volume 8409, 04 2014.
[10] Buckler, C. (2016), 'Browser Trends January 2016: 12 Month Review', Site-point Blog, 12 January.
[11] Calero, C., Ruiz, J. and Piattini, M. (2004), 'A web metrics survey using WQM', Web Engineering,

1(2004109140), pp. 147–160. Chrome Developer (2016) Profiling JavaScript Performance [Online].

[12] Conrad, A. (2012), '3 Reasons to Choose AngularJS for Your Next Project', Envato TutPlus, 26 December.

[13] Eric, A. M. and Bert, B. (2001) Introduction to CSS3 [Online].

[14] Faruk, Paul, Alex, Ryan, Patrick, Stu and Richard (2016) What is Modernizr? [Online].

[15] Gayatri, N Nickolas, S. R. A. V. (2009), 'Performance Analysis and Enhancement of Software Quality Metrics using Decision Tree based Feature Extraction', Int. J. of Recent Trends in Engineering and Technology, 2(4), pp. 4–6.

[16] Gizas, A., Christodoulou, S. P. and Pomonis, T. (2014), 'Performance and Quality Evaluation of jQuery Javascript Framework', Science and Engineering Publishing Company, 3(1), pp. 12–22.

[17] Habra, N., Abran, A., Lopez, M. and Sellami, A. (2008), 'A framework for the design and verification of software measurement methods', Journal of Systems and Software, 81(5), pp. 633–648. Halstead, M. H. (1977), Elements of Software Science (Operating and Programming Systems Series), New York: Elsevier Science Inc.

[18] Ladan, Z. and Hagelb¨ack, J. (2015), 'Comparing performance between plain JavaScript and popular JavaScript frameworks', Linnaeus University Swedan, 1(1), pp. 1–26.

[19] Lamb, D. (2014), 'jQuery vs. AngularJS: A Comparison and Migration Walkthrough', Airpair Blog.

[20] Lei Kai, Ma Yining, T. Z. (2015), 'Performance comparison and evaluation of web development technologies in PHP, Python and Node.js', Proceedings - 17th IEEE International Conference on Computational Science and Engineering, CSE 2014, Jointly with 13th IEEE International Conference on Ubiquitous Computing and Communications, IUCC 2014, 13th International Symposium on Pervasive Systems, 1(1), pp. 661–668.

[21] J. Cincovi´c and M. Punt. Comparison: Angular vs. react vs. vue. which framework is the best choice? In: Zdravkovi´c, M., Konjovi´c, Z., Trajanovi´c, M. (Eds.) ICIST 2020 Proceedings, pages 250–255, 2020.

[22] Juilee Waranashiwar and Manda Ukey. Ionic framework with angular for hybrid app development. 2018.

[23] iFour Team, Devon Fata, Pixoul's CEO. Which technology is better, angularjs or reactjs for frontend development? 04 October 2021.

[24] iFour Team, Justin Nabity, Originator, and CEO of Doctors Thrive. Which technology is better, angularjs or reactjs for frontend development? 04 October 2021.