Abhinav Merugu

Dr. Ben Juliano

Research Methods in Computer Science

March 9, 2023

A Comparative Study of Angular and ReactJS: An Analysis of Front-End Development Frameworks

Research Plan

INTRODUCTION

Front-end development is rapidly evolving, with new libraries, frameworks, and technologies seemingly being released every other day; this often requires the developers to stay updated to meet the requirements. There are several stable frameworks. React [5] is the most famous library, developed by Facebook in 2011. Before its release, AngularJS 32] was in its place and was developed by Google in 2010. AngularJS support has officially ended as of January 2022, and it was rewritten as Angular, released in 2016 by Google. Angular [4] has a component-based architecture like React, whereas AngularJS was based on MVC architecture. Even though Angular is component-based, it still needs to work on becoming relevant to many developers. This research paper focuses on a comparative analysis of the features and performance of Angular and ReactJS to let the developers make informed decisions and determine the reason for decreasing interest in Angular.

Major questions that need to be addressed are **1.** What are the reasons behind ReactJS's popularity and Angular's struggle for popularity? **2.** What are the key differences in terms of architecture and syntax? Which framework is better suited for a specific requirement of a web application? To find answers to the above questions, there is a need to address several minor questions, such as **1.** What are the performance metrics of Angular and ReactJS? **2.** What are the design patterns and best practices for both technologies? **3.** What is the learning curve of

Angular and ReactJS? **4.** How is the rendering of HTML handled? **5.** How is the data handled? **6.** Which framework is scalable and maintainable? **7.** Does React Fiber's [1] approach to rendering DOM affect the rendering speed in React 16? Etc.

Though there are several research papers on ReactJS, there is a scarcity of in-depth research on Angular, those that are widely available are on AngularJS, which is Angular's predecessor. When a major update of the framework involving the changes in the implementation of certain features is released, all the research on those features becomes void. This paper aims to address the feature changes and bridge the gap in the literature through a comprehensive comparative analysis of the implementation and working of ReactJS and Angular. Additionally, this study will serve as one of the vital resources for developers having conflicting opinions about using either of the technologies for web applications by providing empirical evidence and evaluation within the limitations.

The research scope will focus on the features, usability, design patterns, learning curve, and functionality of the two frameworks. Limitations include evaluating specific test cases, which might cover only some of the use cases in front-end development; also, the versions of the frameworks often change as bug fixes, and new features are rolled out, but most of the working principles remain the same. There is also an exception for a major version release. There is also a limitation of different hardware that affects the overall performance of a framework. The study will work within the confines of common use cases.

GLOSSARY OF TERMS AND DEFINITIONS

Angular: Also known as Angular 2, a famous front-end framework developed by Google, the successor to AngularJS, is used for building dynamic web applications.

AngularJS: Famous front-end framework developed by Google, used for building dynamic web applications.

Application Programming Interface (API): A set of protocols and tools for building software applications.

Babel: A tool for transpiling JavaScript code to support newer language features.

Best Practices: A set of guidelines or recommendations to improve software development's quality, maintainability, and efficiency.

Component: A modular and reusable code encapsulating a web application's specific functionality or user interface element.

Command Line Interface (CLI): A software tool used for interaction with applications via the command line.

Client-Side Rendering (CSR): A technique for rendering web pages on the client side using JavaScript.

Content Delivery Network (CDN): A distributed network of servers that delivers web content to users based on their geographical location.

Data Binding: A technique for connecting data from the model to the view in a web application.

Dependency Injection: A pattern for managing dependencies between components in a software application.

Design Patterns: A proven and tested way of solving design problems consistently and efficiently.

Document Object Model (DOM): A programming interface for web documents. It represents the page so that programs can change the document structure, style, and content.

Ease of Use: Simplicity and easy to understand for the developers to learn and use.

ES6/ES2015: The sixth major version of the ECMAScript language specification, also known as ECMAScript 2015.

Flexibility: The ability of the framework to be modified and to adapt to different technologies and platforms.

Framework: Software tool developers use to build a software product or functionality. Library:

Front-end Development: The user interface of web applications.

Hypertext Markup Language (HTML): The standard language used to create web pages and other online documents.

JavaScript: A programming language or scripting language used to create interactive and

dynamic web pages and applications.

JavaScript Object Notation (JSON): A lightweight data interchange format that is easy for humans to read and write for machines to parse and generate.

JSX: A syntax extension to JavaScript used by React to describe the user interface components in a declarative way.

Learning Curve: The rate of a developer's progress in gaining experience or a new skill.

Model View Controller (MVC): A software design pattern that separates an application into three interconnected components: the model (data and business logic), the view (user interface), and the controller (intermediary between the model and the view).

Node.js: A JavaScript runtime built on Chrome's V8 JavaScript engine for building server-side applications.

Node Modules: Packaged JavaScript code for Node.js.

Object Oriented Programming: A programming paradigm that uses objects (instances of classes, to represent and manipulate data, functionality, and behavior).

Performance: Speed and efficiency of a framework to execute and render the desired output.

ReactJS: Also known as React, a famous front-end library developed by Facebook, used for building dynamic web applications.

React Hooks: A set of functions that allow developers to use state and other React features in functional components, simplifying code and reducing the need for class components.

RESTful API: A type of API that follows the principles of Representational State Transfer (REST), a set of constraints for building web services.

Reusability: Use of existing assets within the software product development process.

Rendering: The process of generating the final visual output of a web application.

Routing: A technique for defining and navigating between different views or pages in a web application.

Scalability: Ability to handle large-scale web applications with high traffic and complex functionality.

Single Page Application (SPA): A web application that loads a single HTML page and dynamically updates the content as the user interacts without reloading the entire page.

Server-Side Rendering (SSR): A technique for rendering web pages on the server before sending them to the client.

State Management: A process for managing the state of a web application, including data and user interface state.

TypeScript: A superset of JavaScript that adds optional static type checking, interfaces, and other features to the language.

User Interface (UI): The visual and interactive section of the web application.

Virtual DOM: A lightweight representation of the actual DOM used by React and other JavaScript libraries to optimize rendering performance.

Web Application: A software application that runs on a web server and is accessed through a web browser. It can be a simple website or a more complex web-based system that provides various services to the users.

Webpack: A module bundler for JavaScript applications.

Note: Definitions are sourced from the internet.

REVIEW OF LITERATURE

React, and Angular are popular technologies in the front-end development ecosystem. Still, Angular is preferred by those who have previously worked in AngularJS, whereas React is preferred by most professional developers and those new to front-end development. React has a learning curve that is relatively low compared to Angular due to its complexity, and that is the reason for its popularity among those new to web development. Also, the documentation of React is more user-friendly and enables ease of learning, whereas the documentation of Angular is complex for novice developers. In addition to the documentation, Angular is written in TypeScript, a superset of JavaScript that adds a vital typing feature and other object-oriented programming concepts for catching errors at compile time, better documentation, improved developer productivity, code readability, and quality.

ReactJS and Angular are designed to help developers build complex applications quickly and offer features like component-based architecture, state management, data-binding,

scalability, support for integrating third-party node modules, SPAs, SSR, CSR, and many more. However, there are some critical differences in how the features are implemented. One such in- stance is how the data is handled. React uses a one-way data flow principle, meaning that data is only passed down to the child components and not back to the parent component. This makes the developer write more code to implement complex data flows, often requiring the installation of third-party modules. On the other hand, Angular uses a two-way data binding principle, which means the data can be passed down and up from parent to child and vice versa. While this approach makes it harder to reason in some instances as it may have unexpected changes in other components, and the effects are significant in large-scale applications, it will force the user to follow best practices and design patterns.

React uses a virtual DOM approach to render its UI, a simplified version of the actual DOM tree. React estimates the slightest modifications necessary to update the real DOM when a component's state changes. This strategy, known as “reconciliation,” is performance-optimized since it decreases the amount of time needed to render the UI and prevents DOM modifications that aren't essential. With Angular, a combination of templates and directives generates the user interface. A template is a declarative HTML-like syntax that specifies the UI's structure, and directives are used to enable the dynamic behavior of the template. The new view is rendered on the screen when Angular updates the template when a component's state changes. Angular employs a technique known as "change detection" to identify the changes in the state and update the view accordingly. Angular is slower than React when rendering the UI because of real DOM by Angular and virtual DOM by ReactJS.

Research paper [4] using literature review a methodology for analysis of React, Angular, and Vue. The study focused on Trends and Popularity, Architecture, Features, Performance, Data Binding, and Code Syntax. It concluded that React is a more flexible and performant front-end library that depends on the community for feature implementations. In contrast, Angular is a complete package that is less performant due to its size and says it is more of a platform and less of a framework because of the numerous out-of-the-box features,

unlike React. The work [6] on Angular features concludes that the framework is opinionated and requires the developer to follow a specific style for organizing the project structure.

Another brief study [8] concludes interestingly by addressing two questions, **1.** Is there a difference in available throughput and load performance between (functionally) the same application created with different frameworks? The answer is yes; React is best for efficient rendering, and Angular came on top for throughput total render. **2.** How do the results from the current study compare to those from previous studies? Some of the results are irrelevant to the previous studies as there are changes in the framework and implementations used in previous studies compared to this [8] study.

This is some of the literature that was reviewed, and a common conclusion that can be drawn is that React is more flexible, has better rendering, and is easy to implement. At the same time, Angular is a heavy all-in-one package that forces developers to follow a pattern to implement performant large-scale applications.

RESEARCH PLAN

With inspiration from the article [9] to compare frameworks and libraries, the plan for the research is a five-step process; they are **1.** Identify at least five real-world use cases that require better UI performance, **2.** Review literature for the specific use case, **3.** Implement the component both in React and Angular, and finally, **4.** Compare the performance using time, size, and ease of implementation as the metrics.

Framework, Environment, and Required Tools

React is a modern web-development library. The environment setup for writing React, an IDE tool that most developers use, is Visual Studio Code [7]. The programming language JavaScript is used, along with JSX syntax extension and a package manager ‘npm.’

For Angular, the environment setup of IDE and package manager is the same and use TypeScript, HTML, and CSS, along with the Angular CLI, a command-line interface for

creating and managing Angular projects.

Data Acquisition

This covers steps 1 and 2 of the plans to gather data from different sources and review, including official documentation, online forums, articles, open source code base, and tutorials.

Methodologies

This covers steps 3 and 4 of the plan and involves conducting performance benchmarks and measuring metrics such as load times, rendering speed, and memory usage to compare the performance and documentation quality to implement features of React and Angular.

Test Plan

1. Identify at least five real-world use cases that require better UI performance:

a. E-commerce website with many product listings.

b. Social media platform with a high volume of user-generated content.

c. Web application with complex graphics and animations.

d. Data visualization dashboard with real-time updates.

e. Video streaming platform with an extensive library of content.

2. Review literature for specific use case:

For each identified use case, a literature review is conducted to gather information on best practices and strategies for improving UI performance in React and Angular.

3. Implement the component both in React and Angular:

Using the literature review as a guide, this study will implement a sample component for each identified use case in React and Angular. The same data set and design for each component to ensure a fair comparison and analysis.

4. Compare the performance using time, size, and ease of implementation as the metrics:

Time: Measuring the time it takes for the component to render and update and the time it takes for the page to become interactive for the user after processing and rendering.

Size: The amount of memory the component uses to render and dependencies.

Ease of Use: Evaluating ease of implementation, code readability, scalability, and maintainability of a framework/library.

Schedule and Scope Assessment

Week 1: Identify at least five real-world use cases that require better UI performance. Review literature for specific use cases and determine appropriate metrics for evaluation, finally, set up the testing environment with the needed tools.

Week 2-3: Implement the components in React and Angular for each identified use case. Collect data on the ease of implementation, load times, rendering speed, and memory usage for each component.

Week 4-5: Analyze the data and compare the performance of React and Angular for each use case, documenting the results and identifying any patterns or trends in the data.

Week 6: Final report summarizing the findings and providing recommendations for future development based on collected data.

Note that the scope of this research plan may need to be adjusted based on the implementation of specific use cases identified and the amount of data collected. Unexpected issues or challenges may arise during the testing process, which could impact the schedule and scope of the project. The research plan must be flexible and adaptable throughout testing to ensure the best results.

Feasibility

Assessing this research's feasibility is vital as it compares a library and a framework. Factors such as insufficient resources to develop the use cases, technical expertise, errors caused by any third-party packages involved, and unreliable test data are among the ones to look out for in mitigating the risks effectively. For a successful outcome, specific risk mitigation strategies can be employed, such as using only the latest stable versions of software, reaching out to experts for any advice and help in the implementation of the use cases following design patterns and best practices, having alternate use cases implemented, in case

any of the planned test cases might throw errors affecting the time available to complete the study.

In addition to the above methods, the study will use visualization tools to represent the findings with clarity, seeking input from experienced professionals for their perspective on the strengths and weaknesses of the technologies involved in the study and a careful literature review of the existing research will ensure the consistency of the findings in this study with the broader research community.

HYPOTHESIS

The hypothesis that could be tested in the comparative analysis of React and Angular are:

1. React is faster than Angular in rendering screens:

This hypothesis could be tested by measuring the rendering speed of React and Angular components in the use cases discussed.

2. Angular is better scalable and manageable for large-scale projects than React.

This hypothesis could be tested by implementing and testing large-scale project components and evaluating the ease of scaling and managing.

3. React has a smaller footprint and uses less memory than Angular.

This hypothesis could be tested by measuring the memory usage and size of React and Angular components and evaluating their memory usage efficiency.

4. Angular provides better out-of-the-box support for features to develop complex applications.

Using both technologies, this hypothesis could be tested by implementing components that involve Forms, state management, routing, etc.

Each of these hypotheses will be tested using appropriate metrics and testing methodologies to evaluate the performance and efficiency of both React and Angular.

REFERENCES

[1] Acdilite, GitHub, *React Fiber Architecture*, 2016, https://github.com/acdlite/react-fiber-archi- tecture [ACCESSED 03-11-2023]

[2] A. K. Sahani and P. Singh (2020), *Web Development Using Angular: A Case Study. Journal of Informatics Electrical and Electronics Engineering, Vol. 01, Iss. 02, S. No. 5, pp. 1-7, 2020*, https://jieee.a2zjournals.com/index.php/ieee/article/view/5/10 [ACCESSED 03-10-2023]

[3] Angularjs.org, https://angularjs.org/ [ACCESSED 03-10-2023]

[4] Angular.io, https://angular.io/guide/what-is-angular [ACCESSED 03-10-2023]

[5] Reacts.org, https://reactjs.org/docs/introducing-jsx.html [ACCESSED 03-10-2023]

[6] Rishi Vyas, *Comparative Analysis on Front-End Frameworks for Web Applications,* 2022, http s://issuu.com/ijraset/docs/comparative_analysis_on_front-end_frameworks_for_w [AC- CESSED 03-10-2023]

[7] StackOverFlow.com, *2022 Developer Survey,* 2022, https://survey.stackoverflow.co/2022/#overview [ACCESSED 03-11-2023]

[8] Tomas Marx-Raacz von Hidvég, *Are the frameworks good enough?*, 2022 https://www.diva-portal.org/smash/record.jsf?pid=diva2%3A1717993&dswid=9675 [ACCESSED 03-10-2023]

[9] Viktor Lazarevich, *How to Compare Software Solutions, Frameworks, Libraries and Other Components,* 2022, https://www.digiteum.com/how-to-compare-software-solutions-frameworks- libraries-and-other-components/ [ACCESSED 03-11-2023]