An Architectural Style for Single Page Scalable Modern Web Application

Suresh Kumar Mukhiya[1] Hoang Khac Hung[2]

[1](skmu@hvl.no) Ph.D. candidate at Western Norway University of Applied Sciences, Bergen, Norway
https://www. hvl.no
[2](hunghk.it@gmail.com), Ho Chi Minh City University of Information Technology
https://en.uit.edu.vn/

Abstract- **The increasing complexity and popularity of web application and the movement of enterprise applications from desktop based to web based architecture have created sophisticated challenges yet the need for scalable modern web applications architecture. One way to manage this complexity is to design and develop the web application using single page architecture. The paper describes a single page web application architecture suitable for modern web application development using React JS for building the complex interactive user interfaces, Redux for handling state management, Redux Saga for handling side effects, Node JS as server-site scripting language, Immutable JS to handle immutability and Webpack for module bundling. It is aimed at facilitating the developers and business stakeholders to comprehend the need and importance of making application scalable and maintainable over time.**

Keywords— Singe Page Application, ReactJS, Redux, Redux Saga, Client-Server, Immutable JS, Webpack, Node JS

I. INTRODUCTION

The attention and acceptance of the movement from desktop applications towards web applications are massive and increasing over time. The web (WWW) belongs to a large class of networks adhering to a distributed principle of knowledge representation and provides user-friendly access to stored knowledge [24]. This knowledge affects the behaviour of people around the world. The behaviour of the people has a significant impact on business, commerce, industry, banking and finance, education, entertainment, government, and other personal and working life. Hence, enterprises invest resources to structure this knowledge so that it can be accessible and available to end users in most accurate and efficient way. Web applications have suffered from poor interactivity and responsiveness towards the end users despite its enormous popularity. Web applications are still based on the classical multi-page interface model, in which for every request entire page is refreshed [26], [30], [33], [29]. These architectures are based on a page sequence diagram and have many limitations in terms of user friendly human-computer interactions and responsiveness [29].

The need for better representation of knowledge and the problems with multi-page architecture are the motivation to optimize the web application architecture in a better way to adhere to scalability, availability, security, performance, and maintainability. In this context, a web application architecture defines the interactions between applications, middleware systems and databases to ensure multiple applications can work in a harmony. Moreover, the architecture is the blueprint for supporting growth which may come from increased demands, future interoperability and enhanced reliability requirements. The main question addressed in this paper is how to appropriately structure a single page web application that provides scalability, availability, security, performance, and maintainability. The architecture follows the client-server paradigm. The client-server architecture is the most general web application architecture where server-side components are configured on a network [10]. A web browser works as the client where no special configuration is required. We investigate a single page architecture for development of scalable modern web application using ReactJS for creating interactive user interfaces, Redux JS for state management, Redux Saga for handling side effects, immutable JS for immutability. A single-page architecture application works inside a web client without a requirement of page reloading. This architecture can be visible in the instance of Gmail, Google Maps, Instagram, Facebook, Netflix or GitHub.

We next introduce the architecture (Section II), architecture properties (Section III), then related works (section IV), and discussion (Section V).

II. ARCHITECTURE STYLE

With the growing popularity and complexity of web applications and the openness of web application architecture, the design, development, and maintenance of enterprise applications are becoming complex. One way to mitigate this complexity is following common web architecture that addresses scalability, maintainability and other quality attributes. In this paper, we present single page architecture using ReactJS and Redux. The architecture consists of the following major components:

A. Architectural components

Below we present some the major components of the web application architecture. In each subsection, we present why

these components are essential in promoting scalability of the architecture.

1) ReactJS: ReactJS is a component-based JavaScript library that follows the declarative programming paradigm. The declarative views can be utilized to create complex interactive UI which serves as the presentational component of the web. The library is easy to learn, promotes code reusability, offers lightweight DOM for better performance, enforces unidirectional data flow, ensures Search Engine Optimizations (SEO), forms the views of Model View Controller(MVC) architecture and supports virtual DOM [3], [18], [22], [31]. Virtual DOM is managed by the reconciliation process [19], [20] where an idea or virtual representation of a UI is stored in memory and synced with the real DOM by the library such as ReactDOM [3], [18], [13]. Because of these features and benefits, the enterprises like Facebook, Netflix, Yahoo, Atlassian, Khan Academy, Pinterest, Dropbox, and others have successfully migrated to the usage of ReactJS.

2) Redux: Facebook was encountering difficulties with the MVC structure. The Model and View relationships can become complicated especially when an application commences scaling. Increase in Model and Views in an application can result in infinite loops. To surmount this problem, Facebook launched Flux which is an unidirectional way of updating views and handling user actions. Redux JS is a remodelled implementation of the Flux architecture. The most prominent distinction in Flux and Redux is that Flux has several stores but Redux has just one root store. Redux practices this theory of unidirectional data flow [36] and grew into a de-facto pattern as a state management technology for ReactJS applications. It is a convenient and straightforward method of structuring data in an application and presenting it on the client. The application has a central root state. A state change triggers view updates. Only special functions can modify the state. A user interaction triggers these special, state changing functions. Only one change takes place at a time. This means that the central state cannot trigger any further actions. Only a user input can trigger another action. This makes the state much more manageable. It also makes it difficult to introduce infinite loops provided single source of state management is followed like Redux.

3) Redux-saga: Redux-saga [37] library facilitates easier management and efficient execution of side effects in a web application. In addition to that, the library shines at failure handling. In this context, side effects refer to asynchronous things like data fetching and impure things like locating the browser cache or cookies. Two of the most well-known techniques of dealing with side effects in Redux apps are Redux Thunk [38] and Redux Saga [37]. There are several blogs comparing differences between saga and Thunk where developers present their subjective opinions and contextual benefits. Redux Saga allows writing a complex sequence of synchronous and asynchronous events in a clear and declarative style without call backs.

4) NodeJS: Node.js has gained popularity in the server side scripting language and offers client-server development integration, aiding code reusability in web applications, and is the perfect tool for developing fast, scalable network applications [9]. NodeJS is a framework for developing high-performance, concurrent programs that don't rely on the mainstream multithreading approach but use asynchronous Input/output with an event-driven programming model [39]. Kai Lei and et. al performed a study to compare the performance of Node.js, Python-Web, and PHP using benchmark tests and scenario tests. The experimental results yield some valuable performance data, showing that PHP and Python-Web handle much fewer requests than that of Node.js in a certain time. The study demonstrates that Node.js is quite lightweight and efficient, which is an ideal fit for I/O intensive websites among the three, while PHP is only suitable for small and middle scale applications, and Python-Web is developer friendly and good for large web architectures [23].

5) Immutable JS: Immutable JS [15] for handling immutability in the states. Immutable data is a core idea in the functional programming. Immutable data helps to adhere to the principle that if the app state has not changed, the DOM should not change. Immutable JS is an open-source JavaScript library from Facebook which helps in data flow simplification, optimization of data change detection, performance enhancement through memoization. Memoization is one of the techniques to store values of a function instead of recomputing them each time the function is called. A couple of important aspects with regards to immutability are it increases predictability, performance and allows for mutation tracking. In addition to that, Immutable JS provides a convenient way to modify deeply nested properties. Two major need for Immutable JS are:

Data Reference Problem: React is not just about building interactive UI/UX interfaces, it is about performance. The purpose of its development was to be performant and only update the DOM when required and only update the portion which is required to be updated. An optimized react app should contain the simple stateless functional components and can have *shouldComponentUpdate* returning false.

```
shouldComponentUpdate(nextProps, nextState) {
  return false;
}
```

The most notable function in the React component lifecycle is *shouldComponentUpdate* and is expected to return false whenever possible. This ensures that the component having *shouldComponentUpdate* returning *false* would never re-render thus making the React app extremely performant. When building a React app, our goal is to compare old props with new props and states and if they are not changed, the component should never re-render. However, the equality checks in JavaScript is very sophisticated. Consider equality on complex objects and arrays:

```
const object1 = { prop: 'simple value' };
const object2 = { prop: 'simple value' };
```



```
console.log(object1 === object2); // false
```

The *object1* and *object2* appear to be the same but their reference is different. As these two objects are judged to be different, equalling them naively within the *shouldComponentUpdate* function will make our component re-render needlessly. Data comes from Redux reducers. If these reducers are not set correctly, they will be presented with a different reference which will cause the component to re-render every time. This will be one of the major problems with respect to performance.

Reference Handling Problem: When we start working on a real application, sooner or later we come across deeply nested objects. Assuming, the need to compare the object with the previous values, one way to achieve this is looping through each object recursively. We can visualize this computation can be expensive and would require some other solution. One way to solve them is by inspecting the reference. However, it requires us to preserve the reference if nothing has changed, as well as change reference if any of the nested object/array prop values changed. This task is very complicated. Although there are some of the libraries that help in deep comparing the objects, and if we want to do them in a clean, optimized and nice way, Immutable JS is one of the most popular solution. The Facebook developers had faced these obstacles at their very early stage of development and hence developed Immutable JS to overcome this issue.

In addition to these major components, the architecture utilizes other libraries. Webpack is preferred to be used as module bundler [41]. Express is a minimal web framework for Node JS [12]. There are several libraries that facilitate the development of presentational component including Redux form [35] for building forms, Material UI [28], Ant Design [11], and Bootstrap [5] for building interactive UI components.

B. Client-Server Architecture

In general, single page application follows multi-layer client-server architecture. Browsers act as the client and communicate with a server using REST API over HTTP [42] protocol. Figure 1 shows different layers of the single-page web application. As illustrated in figure 1, the API layer communicates with several application layers (for example Watcher, Webhooks, Notifiers) which in turn communicates with the database layer. To ensure interoperability, the database layer is likely to communicate with other external services through a secured firewall.

C. Architecture of the React App with Redux and Saga

As aforementioned, the single-page architecture with React and Redux follows the client-server paradigm. In this section, we present the conceptual model and the logical model of the architecture.

1) Conceptual Model: Figure 2 shows the conceptual model of the architecture. The architecture has four layers - view layer, application services layer, store layer, and domain layer.

The presentational and container components constitute the view layer. Presentational components are concerned with how things appear while container components are concerned with how things operate. Interactive UI components like Ant design, Bootstrap, Redux-form, and Material designs can be used to construct these components. These components are independent and promote code-reusability. A button component can be used several times in the same application. In addition to this, these components can be exported to be used for independent applications. Application services layer is beneficial for preparing all kinds of operations which are near to the data flow like Ajax calls to retrieve data from the server or state projections. This layer is optional based on a developers' preferences. The store layer contains the data which results from CRUD (Create, Read, Update, Delete) operations on the application. The store holds all these data in the state. As mentioned in section Redux, there is only one global store where is responsible for data flow in the entire application. The domain layer explains the state and holds the business logic which is the core of the application. Since we are presenting immutable structure, our domain layer will of entities and domain services.

2) Logical Model: Figure 3 shows the logical model of the single page architecture. The diagram shows an application structure with async-based action that is triggered using Redux Saga. In this section, we try to explain the logical model of the architecture taking an example of a simple Hospital Management System. Having to display the list of doctors in our application, the first thing we do is to model Doctor entity. All the object type of the entity doctor needs to be immutable. React is used to create UI/UX elements which form the virtual DOM. The UI parts are referred to as Component. Each component has a constructor which initializes the component with required attributes. These UI components trigger events which are captured by action creators. Action creators dispatch action. For example, if a user wants to delete a doctor record from the application, an event of delete is triggered by pressing a link or a button. Actions are payloads of information that send data from an application to the store. In this case, action has type and payload. Action type can be DELETE and data could be the identifier of the doctor that needs to be deleted. In principle, we never change state manually, we trigger change by dispatching actions.

Redux Saga provides us with the async dispatch which ensures HTTP requests will not clutter up the execution flow. Saga allows creating all the logic of event stream processing. Saga runs in the background right after the app is launched and observe all actions that store dispatches. Once it finds correct actions dispatched, it listens to it and performs required asynchronous activities like fetching data or deleting the data from backend by calling the API layer. For example, in this case, once saga observers DELETE action been dispatched, it will call the API with correct

payload to delete the doctor. The reducers get the type of actions and update the store data which in turn are reflected in the actional DOM.

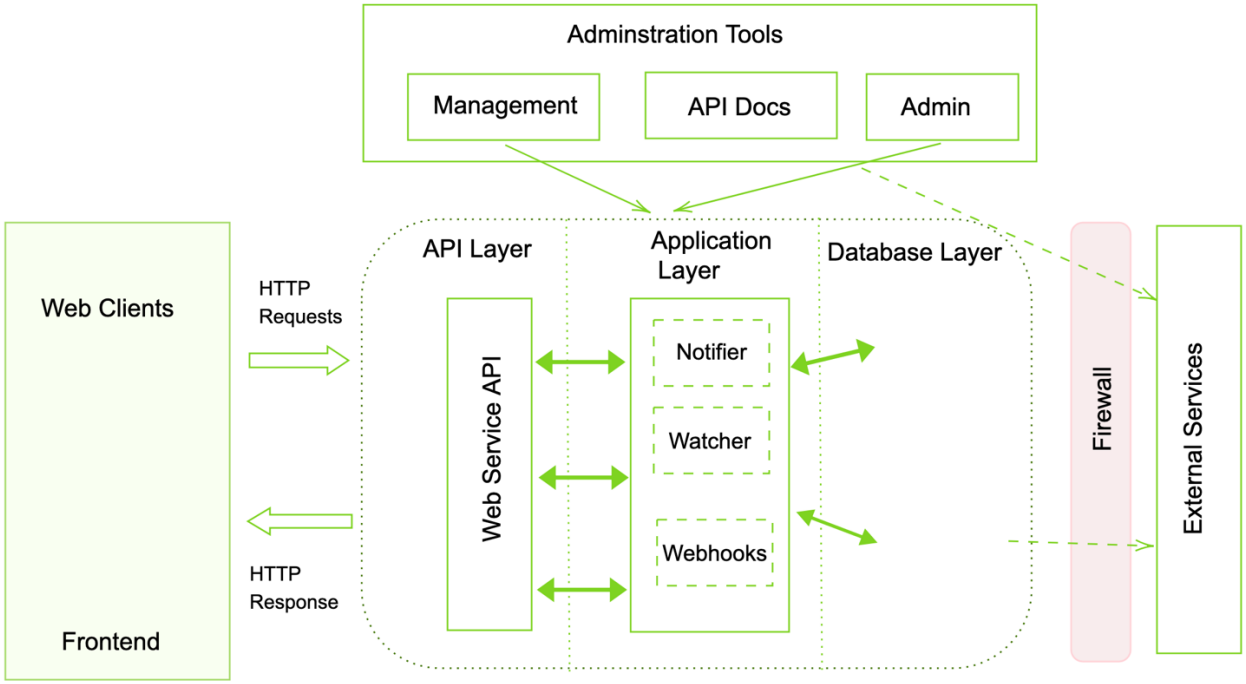

Figure 1. Client-server architecture of the web application

III. ARCHITECTURAL PROPERTIES

In this section, we discuss several architectural properties of single page architecture with respect to ReactJS, Redux, and Redux-saga. These properties can be regarded as the evaluation of the system in terms of usability, accessibility, testability, and maintainability.

A. User Interactivity

Software architecture term usability is closely related to user interactivity [14] whereas human-computer interaction defines user interactivity as the degree to which end users can proactively participate in the communication process and exchange roles in their mutual discourse [8]. ReactJS holds strong ground on building interactive user interface as aforementioned in the previous section. The browser is an event driven application and everything that an end user does in the browser fires an event including from dragging the mouse to even hovering over an element. ReactJS documentation provides various supported events including focus events, mouse events, pointer events, selection events, touch events, wheel events, media events and many more [21]. These events driven support can be used in building a complex user interactive interfaces for user engagement on the web. As suggested by the study done by Teo and et al. [1], a higher level of user interactivity has positive effects on the users perceived satisfaction, efficiency, attitude and effectiveness towards a website.

B. Accessibility

Web accessibility is also referred to as a11y which refers to the facilitation of web content for assistive technology. ReactJS fully promotes constructions of a web-accessible application using standard HTML techniques and provides full support for it [16]. Web accessibility of an application made using the single page architecture can also be evaluated by different methods, including subjective assessments, standards review, user testing, and barrier walkthrough [6], [40].

C. Network Performance

The network performance of a web application is highly influenced by the rate of data transmitted on the network and bandwidth. It is also dependent on how the application updates DOM when a user moves from one UI to another. As mentioned in [3], [18], [22], [31], [19], [20], ReactJS maintains the virtual DOM which updates by the reconciliation process which means only the DOM elements which have updates will be re-rendered. This makes the application network performance compared to other multi-page architectures. Redux is heavily optimized to reduce unnecessary re-renders. However, the work done by Redux JS is affected by processing actions in the middleware's, reducers, and subscribers. An application developed using this architecture must be optimized to avoid unnecessary reconciliation. This can heavily enhance network performance of the application [17].

D. Readability

Readability refers to as a human judgment of how easy a piece of code is to understand. The readability of a program

is highly related maintainability and are regarded as the key factor in overall software quality [7], [2]. ReactJS and Redux code can be organized to have a higher level of readability. The readability matrix is very subjective to the developer's competence. However, ReactJS promotes code readability in its own implementation. For example, lifecycle method name *shouldComponentUpdate* [18]. The name suggests the function will return boolean value by indicating the component should update or not. The libraries proposed in this architecture in this paper promotes readability in terms of naming variables, naming methods, organizing code, keeping component smaller, making a small and reusable component, standard indentation, comments, using a common vocabulary for naming function and variables, and single responsibility principle [27].

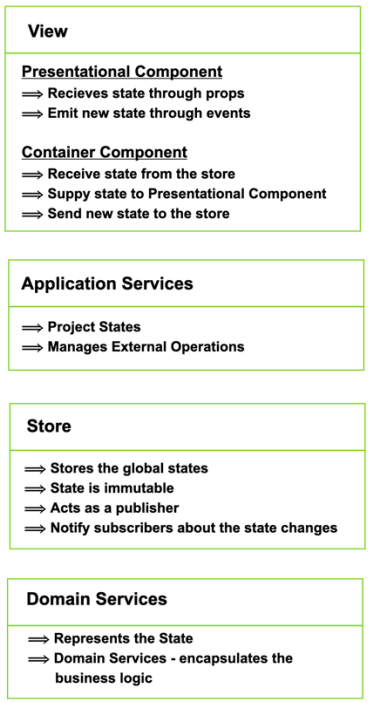

Figure 2. Conceptual Model of the single page architecture

E. Simplicity

The effort that is required to comprehend, plan, analyse, design, implement and maintain a web application is referred to as simplicity. Single-page architecture offers simplicity in terms of implementation. In addition to that, as aforementioned, ReactJS follows the declarative paradigm in which the developers tell React what state they want the UI to be in, and make sure the DOM matches the state. This provides abstraction from the attribute manipulation, event handling, and manual DOM updating that developers have to think about when building an app using another technological stack. The abstraction provides simplicity in development of web and allows the developers to focus on the business logic of the application rather than internal implementation details.

F. Scalability

The ability of a system to grow in terms of components can be viewed as scalability. In a web application scalability refers to the number of clients a web application can handle without deteriorating its quality and results. Single-page architecture can be easily configured to serve a growing number of client requests and has been already proven by its use in enterprise application like Facebook, Netflix, Yahoo, Atlassian, Khan Academy, Pinterest, Dropbox, and others.

G. Maintainability

Maintainability refers to the ease with which a software system can be modified to correct faults, improve performance, or adapt in a new environment or scale with respect to resources [36]. Fred Brooks claimed the total cost of maintaining software is typically 40 percent or more of the cost of developing it [37]. A lot of other studies has been done that accepts software maintainability as one of the major challenges [38]. Single-page architecture follows a flexible structure where each component has a single responsibility. In addition to that, it separates the presentational component, container component, states, and business logic separately. This makes easier to maintain the application. In a well-structured web application, a developer knows where to start debugging if there is any bug in the application. For example, if there is an issue in the display of a button on the browser, a developer is likely to check the presentational component and correct Cascading Style Sheet (CSS). Moreover, one of the strongest maintainability aspects that can be argued is the Separation of Concerns (SoC). SoC is a design principle for separating a software program into distinct sections where each section addresses a distinct and separated concern. This separation facilitates in making the app maintainable, extensible, flexible as well as reusable. The architecture illustrated in figure 3 promotes separation of concerns where React component and Redux components interact to scalable web applications. In this architecture, ReactJS component is responsible for UI design by rendering HTML contents, dispatch actions on user interactions or lifecycle events and performs the animation. The second component Redux maintains the data in a store and provides unidirectional data flow. This helps to manage and organize data better and make the debugging process very smooth.

H. Testability

There are several tactics used to make the system easier to test. For example, limiting the complexity of the structure or of the modules and limiting randomness as much as possible should help the tester achieve his goals. Moreover, adding special interfaces and the function that the player can play his own created maze, should make testing easier, faster and less expensive.

IV. RELATED WORK

Jakob Nielsen [32] explores various usability techniques and their importance to the modern web application in his book. He supports his idea by various principles and experiments

in the book. Jim Conallen [10] illustrates how object-oriented

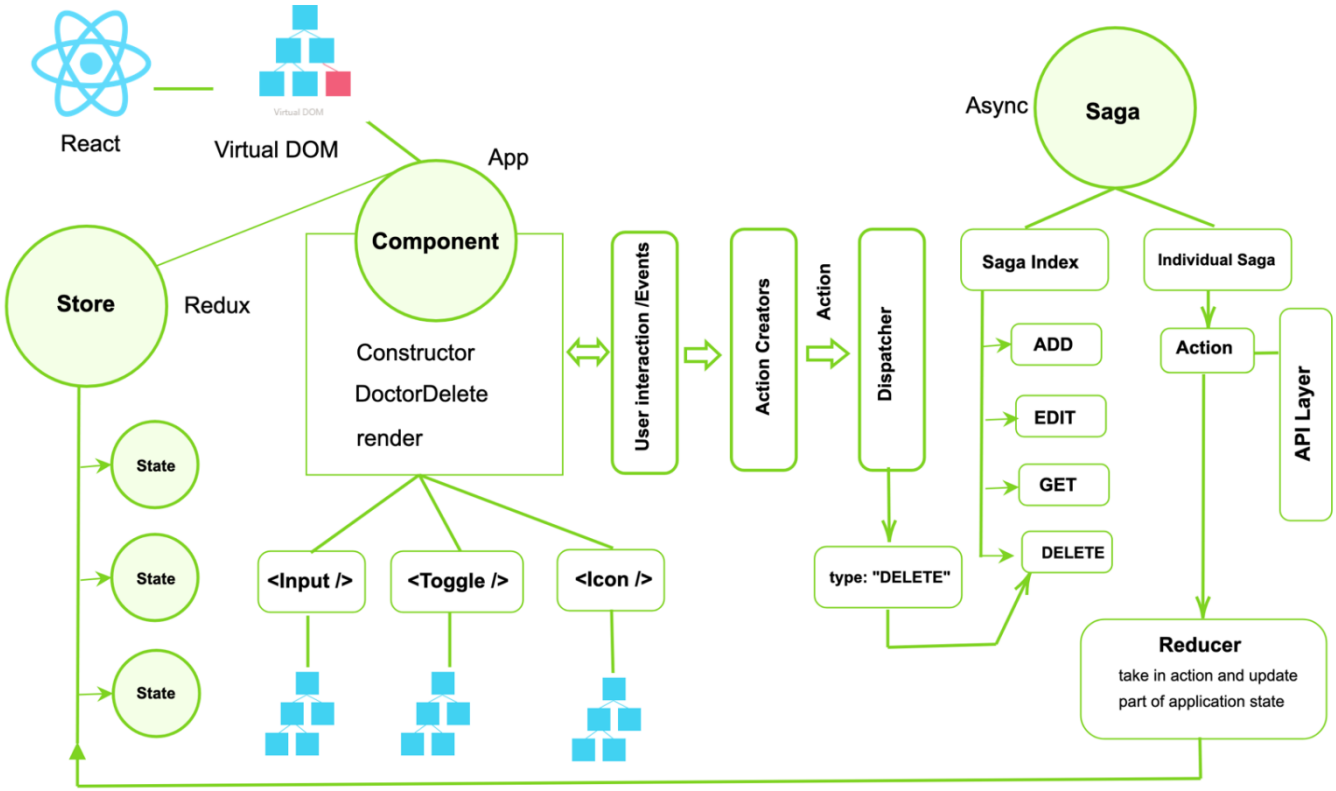

Figure 3. The logical model of the single page architecture

skills can be consumed in the web application development through different web modeling paradigm like UML. The author demonstrates how Model Driven Software Engineering can be used in the improvement of quality, portability, productivity, and maintainability. Recently, a number of technical books have appeared on the subject of web application development. Asleson and Schutta [4] focus on client-side technology. A handful of research has been published using AJAX for web programming [10], [29], [4]. One of the recent development in the software-architecture is service-oriented architecture (S0A). The architecture has transformed the way software engineers and developers today design the application [25]. This paper [34] explains some of the benefits and challenges of using Enterprise Software Oriented Architecture. SOA promotes scalability, better user response, low latency, organized programming model, and interoperability. However, this architecture needs to ensure all the web services [25] standards adopt the same technical standards. Moreover, in order to achieve the benefits of reusability, responsiveness, and extendibility, services must be specified at the correct level of abstraction and granularity [34]. In multi-page applications, any change in the frontend requests rendering of a new page from the server in the browser. These applications ender each page due to the amount of data. The multi-page applications are good and easy for Search Engine Optimization (SEO) management as it gives increased chances to rank different keywords of an application. In addition to this, a multi-page application can be scalable but it requires more development time and has slow speed, update, and performance.

V. DISCUSSION AND CONCLUSION

In this paper, we have discussed using single page architecture for developing a scalable system. The main contribution of this paper is the field of web application development and software architecture. From a software architecture perspective, we investigated the quality attributes of the single page architecture. The paper provides an abstract view of setting React application. From the web application development perspective, we presented a unified single page architecture for building the scalable web application that enhances modifiability. Single page architecture is fast and responsive, has caching capabilities with linear user experience, are scalable, secure, and maintainable. Use of Redux for state management depends on the complexity of the application and need of the data management. If an application is not data-driven, use of Redux will be overkill. To summarize, Redux was created to manage and debug application states in very sophisticated web application systems. Developers argue with the perspective that, for even small changes in functionality, redux requires an excessive amount of code. However, provided the large application scenario Redux is extremely useful. Use of Immutable JS is often questioned by the developers' community. Immutable data is the core concept in the functional programming and its use justifies one of the core principles of React and Redux: if the app state has not changed, the DOM should not change as well. Other than that it doesn't quite follow the ES6/7/8 pattern and there are

several discussions in the online community about difficulty documentation and debugging. Moreover, any fetches to a server for JSON requires conversion from Immutable JS format to readable JSON object using *toJS* function. In addition to this, developers' community also believe there is nothing wrong with mutating objects and there are other libraries that help with specialized data structures. Immutable JS is not the only way to prevent immutability which is the core principle of functional programming. There are other libraries. Immutable JS is one of the libraries recommended by the React team. However, the use of the library must be studied in the web application before using it. Considering the size of the application, and flow of data, use of Redux, Redux Saga and Immutable JS is recommended. It is very important to understand the functional requirements of the application, number of requests need, flow of data before choosing the architecture and its component. Further research encompasses the use of the architecture in a developing an application and evaluation of each component. One of the possible branch to explore beyond this work is the application of the model-driven approach in the construction of a single page application.

REFERENCES

[1]. An empirical study of the effects of interactivity on web user attitude. International Journal of Human Computer Studies (2003).

[2]. AGGARWAL, K. K., SINGH, Y., AND CHHABRA, J. K. An integrated measure of software maintainability. In Annual Reliability and Maintainability Symposium. 2002 Proceedings (Cat. No.02CH37318) (Jan 2002), pp. 235–241.

[3]. AGGARWAL, S. Modern Web-Development using ReactJS. International Journal of Recent Research Aspects (2018).

[4]. ASLESON, R., AND SCHUTTA, N. T. Foundations of Ajax. 2006.

[5]. BOOTSTRAP. Bootstrap, https://getbootstrap.com. last accessed on 2018.8.12.

[6]. BRAJNIK, G. A comparative test of web accessibility evaluation methods. In Proceedings of the 10th international ACM SIGACCESS conference on Computers and accessibility - Assets '08 (2008).

[7]. BUSE, R. P., AND WEIMER, W. R. Learning a metric for code readability. IEEE Transactions on Software Engineering (2010).

[8]. CARROLL, J. M. Human-computer interaction: Psychology as a science of design. International Journal of Human Computer Studies (1997).

[9]. CHANIOTIS, I. K., KYRIAKOU, K. I. D., AND TSELIKAS, N. D. Is Node.js a viable option for building modern web applications? A performance evaluation study. Computing (2015).

[10]. CONALLEN, J. Modeling Web application architectures with UML. Communications of the ACM (1999).

[11]. DESIGN, A. Ant design, https://pro.ant.design. last accessed on 2018.10.12.

[12]. EXPRESSJS. Expressjs, https://expressjs.com/. last accessed on 2018.10.12.

[13]. FACEBOOK. React dom - a javascript library for building user interfaces, https://reactjs.org/docs/react-dom.html. last accessed on 2018.11.10.

[14]. FOLMER, E. Software Architecture Analysis of Usability. Current (2005).

[15]. IMMUTABLE. Immutable js, https://facebook.github.io/immutable-js/. last accessed on 2018.11.10.

[16]. INC., F. Accessibility, https://reactjs.org/docs/accessibility.html/. last accessed on 2018.8.9.

[17]. INC., F. Avoid reconciliation, https://reactjs.org/docs/optimizingperformance.html#avoid-reconciliation/. last accessed on 2018.8.19.

[18]. INC., F. React - a javascript library for building user interfaces, https: //reactjs.org/, 2018. last accessed on 2018.10.10.

[19]. INC., F. React fiber architecture, https://github.com/acdlite/react-fiberarchitecture/, 2018. last accessed on 2018.9.22.

[20]. INC., F. Reconciliation, https://reactjs.org/docs/reconciliation.html, 2018. last accessed on 2018.10.18.

[21]. INC., F. Syntheticevent, https://reactjs.org/docs/events.html/, 2018. last accessed on 2018.11.9.

[22]. KHUAT, T. Developing a frontend application using reactjs and redux. Master's thesis, Laurea University of Applied Sciences, 2018.

[23]. LEI, K., MA, Y., AND TAN, Z. Performance comparison and evaluation of web development technologies in PHP, Python and Node.js. In 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, Algorithms, and Networks, I-SPAN 2014 and 8th International Conference on Frontier of Computer Science and Technology, FCST 2014 (2015).

[24]. LI, D., BROWNE, G., AND WETHERBE, J. Why Do Internet Users Stick with a Specific Web Site? A Relationship Perspective. International Journal of Electronic Commerce (2006).

[25]. MAAMAR, Z., HACID, H., AND HUHNS, M. N. Why web services need social networks. IEEE Internet Computing (2011).

[26]. MARCHETTO, A., TONELLA, P., AND RICCA, F. State-based testing of Ajax Web applications. In Proceedings of the 1st International Conference on Software Testing, Verification and Validation, ICST 2008 (2008).

[27]. MARTIN, R. C., NEWKIRK, J. W., AND KOSS, R. S. Agile Software Development, Principles, Patterns, and Practices. 1998.

[28]. MATERIAL UI. Material ui, https://material-ui.com. last accessed on 2018.10.14.

[29]. MESBAH, A., AND VAN DEURSEN, A. An architectural style for ajax. In 2007 Working IEEE/IFIP Conference on Software Architecture (WICSA'07) (2007).

[30]. MESBAH, A., AND VAN DEURSEN, A. Migrating multi-page web applications to single-page AJAX interfaces. In Proceedings of the European Conference on Software Maintenance and Reengineering, CSMR (2007).

[31]. MOUSAVI, S. A. Maintainability Evaluation of Single Page Application Frameworks- Angular2 vs. React. PhD thesis, Linnaeus University Vxj, Sweden, 2016.

[32]. NEILSEN, J. Designing Web Usability: The practice of simplicity. Interactive Marketing (2000).

[33]. PAULSON, L. D. Building rich Web applications with Ajax. Computer (2005).

[34]. PEUERLICHT, G. Enterprise SOA: What are the benefits and challenges? . University of Technology, Sydney (2007).

[35]. RASMUSSEN, E. Redux form, https://redux-form.com/7.4.2/. last accessed on 2018.10.11.

[36]. REDUX. Redux, https://redux.js.org/. last accessed on 2018.7.19.

[37]. REDUX-SAGA. Redux saga, https://redux-saga.js.org/. last accessed on 2018.7.20.

[38]. REDUX-THUNK. Redux thunk, https://github.com/reduxjs/redux-thunk/. last accessed on 2018.8.20.

[39]. TILKOV, S., AND VINOSKI, S. Node.js: Using JavaScript to build highperformance network programs. IEEE Internet Computing (2010).

[40]. VIGO, M., BROWN, J., AND CONWAY, V. Benchmarking web accessibility evaluation tools. In Proceedings of the 10th International CrossDisciplinary Conference on Web Accessibility - W4A '13 (2013).

[41]. WEBPACK. Immutable js, https://webpack.js.org/. last accessed on 2018.7.10.

[42]. Internet Engineering Task Force. RFC 6749. Hypertext Transfer Protocol -- HTTP/1.1. https://tools.ietf.org/html/rfc2616#section-5. Accessed December 10, 2018.

