Tampere University

The Anh Nguyen

LITERATURE REVIEW ON HISTORY, EVOLUTION AND PROSPECTED FUTURE OF REACTJS

Bachelor of Science Thesis
Faculty of Information Technology and Communication Sciences (ITC)
Supervisor: Prof. Kari Systä
April 2022

Abstract

The Anh Nguyen: Literature review on History, Evolution and Prospected future of ReactJS
Bachelor of Science Thesis
Tampere University
Bachelor's Degree Programme in Science and Engineering
April 2022

Nowadays, the Internet has become popular worldwide. A result of this is that software products and applications tend to shift from being installed to desktop machine to the web. This leads to a significant growth of web-based applications, developed with many web development frameworks and libraries. Among those web development frameworks that are currently popular, ReactJS is one of the most noteworthy ones. The main purpose of this thesis is to study the history and how ReactJS has been evolved, therefore predict its prospected future.

Keywords: ReactJS, Framework, Development, History of React.

Contents

LIST OF SYMBOLS AND ABBREVIATIONS

AJAX	Asynchronous JavaScript and XML
CSS	Cascading Style Sheets
DOM	Document Object Model
HTML	Hypertext Markup Language
JSX	JavaScript XML
MVC	Model View Controller
SPA	Single Page Application
SSR	Server side rendering
UI	User Interface
VDOM	Virtual Document Object Model
W3C	World Wide Web Consortium

LIST OF FIGURES

1 Introduction

Nowadays, with the burst of the Internet, almost everyone can get access to the Internet wherever they are. Also for this reason, a huge number of web and mobile applications for various purposes has been created so that users can use those services more conveniently. It can be said that most daily tasks can be done only by using the web. The speed of the Internet is getting faster and faster, along with exceeding performance of modern devices, resulting in the greater demand for faster applications.

A trend was witnessed that the software applications have been shifted from having to install to desktop machines to being deployed into the Web in order to effectively increase the mobility and availability. This leads to the creation of numerous web application development libraries and frameworks. In a web application, the front-end plays a crucial role as it is the part used directly by the users. A fast and good-looking front-end is the key to attract more users to access the application. Among popular JavaScript front-end development frameworks such as AngularJS, EmberJS, VueJS, ReactJS is currently the one that receives the most attention from the developer community [1].

In this thesis, the history of React, the purposes why it was created, the development process as well as most remarkable milestones and features happened during that period, including the release of supportive tools and technologies, and some predictions about the future of React will be discussed. The thesis will also analyze the evolution of the web development technologies chronologically in order to give a brief background for creation of ReactJS. The objective of the thesis is to carry out an in-depth research of how ReactJS becomes the most popular framework.

The structure of the thesis is organized as follow. Chapter 2 will briefly explain ReactJS' core concepts and state several advantages of learning ReactJS. In chapter 3, the summary of the history of the web application development will be discussed as the background for the evolution of ReactJS. Chapter 4 will give an attempt to predict the prospected future of ReactJS based on its development as well as the current situation. Finally, chapter 5 will summarize the content of the thesis.

2 What is ReactJS?

ReactJS is one of the most popular open-source JavaScript front-end libraries in recent years (Most commonly used web framework according to Stack Overflow 2021 Survey [1] and Monocubed's List of 10 Best Front end Frameworks to Use For Web Development [2]).

2.1 ReactJS's core concepts

2.1.1 Virtual DOM

The DOM (Document Object Model) is a programming interface that represents the current state (structure, style, and content) of a web page or a web-app as a tree data structure containing nodes and objects, which can be used to interact with programming languages to apply changes to the user interface (UI) [3]. DOM manipulation is considered the heart of modern web applications. However, directly manipulating the DOM is not so efficient since not only the content is changed, but also the CSS is recalculated and layouts are changed using complicated algorithms, which affects the overall performance. Furthermore, most JavaScript frameworks are making this even worse because they tend to re-render the browser whenever the DOM is updated, which results in a lot of unnecessary re-rendering and performance waste.

This is where the virtual DOM takes place. The virtual DOM (VDOM) is a concept where a lightweight copy or a "virtual" representation of the actual DOM is stored in the memory and kept synced with the "real" DOM. The virtual DOM shares similar characteristics with the actual DOM and for each object in DOM, ReactJS has a corresponding object in its virtual DOM. The biggest difference between the virtual DOM and the DOM is that the virtual DOM is incapable of changing the the UI directly.

How virtual DOM helps ReactJS with improving the performance much more is that ReactJS keeps two versions of virtual DOM: the first one contains the updated virtual DOM and the other one contains the virtual DOM before the update,

then React compares and find differences between these two, called ”Diffing”. The virtual DOM then calculates the best way to apply these changes to the browser DOM, where only the updated components get re-rendered [4]. This process is called ”Reconciliation”. Reconciliation can save a big amount of re-rendering, which is more performance-costing than updating the DOM. The demonstration of this Reconciliation is shown in Figure 2.1:

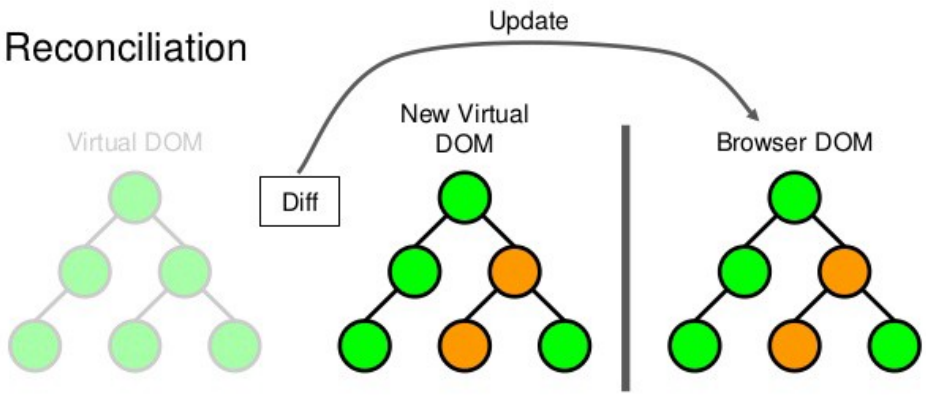

Figure 2.1 *Screenshot of Reconciliation. Reprinted from The Comprehensive Guide to React's Virtual DOM. [5]*

2.1.2 JSX

JSX (or JavaScript XML) is a markup syntax extension to JavaScript. JSX produces React elements and is recommended to be used with React to describe the UI. Although it might look like a template engine or HTML, but it allows us to utilize JavaScript's full power [6].

JSX is not required in React, but it helps simplify the process of writing React components [4]. There are three main benefits of using JSX [7]. The first benefit of using JSX is that it improves developer experience. Thanks to an XML-like syntax, JSX presents the nested declarative structures better than the old function calls and object construction, particularly React.createElement(). Below is the comparison between the code for creating the HelloWorld and a link element using JSX and React.createElement() function call.

```
ReactDOM.render(
```

```
  <h1>Hello world!</h1>,
  document.getElementById('content')
)
```

Program 2.1 *Rendering HelloWorld using JSX [7].*

```
ReactDOM.render(
  React.createElement('h1', null, 'Hello world!'),
  document.getElementById('content')
)
```

Program 2.2 *Rendering HelloWorld using React.createElement() function call [7].*

Another advantage of using JSX is that it helps team members with not much coding experience (such as designers) with editing the code since its syntax is quite similar to HTML, which is familiar to them. Last but not least, JSX is so simple that it will prevent developers from making small mistakes when writing the code and therefore reduce repetitive-stress injuries.

JSX is said to be able to utilize the full power of JavaScript. The reason is that we can use any valid JavaScript expression in JSX elements using curly braces and JSX elements can be treated as JavaScript object [7]. The use of JavaScript expression is shown in Program 2.3.

```
function formatName(user) {
  return user.firstName + ' ' + user.lastName;
}

const user = {
  firstName: 'Harper',
  lastName: 'Perez'
};

const element = (
  <h1>
    Hello, {formatName(user)}!
  </h1>
)

ReactDOM.render(
  element,
  document.getElementById('root')
)
```

Program 2.3 *Use of JavaScript expression in JSX. [7].*

However, JSX is still a combination of HTML and JavaScript, therefore it is not valid JavaScript as they are embedded in HTML [8]. This leads to the fact that JSX is not supported on any browser because the browser engine is not implemented to understand it, which means JSX can only be used in browsers after compiling into valid JavaScript by a compiler (Figure 2.2).

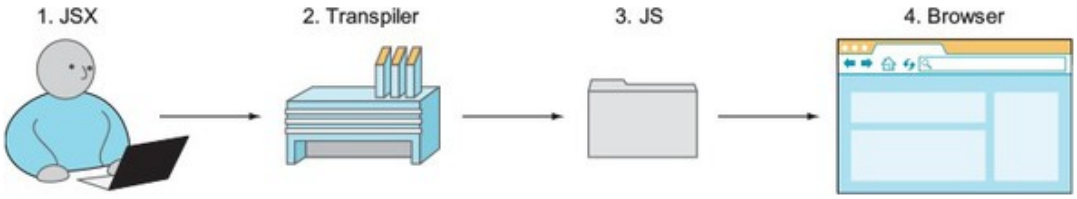

Figure 2.2 *JSX is compiled into regular JavaScript. [7]*

2.1.3 Component-based architecture

The web applications are getting more and more sophisticated, but no matter how complicated they are, they can still be divided into small and simple pieces such as buttons, text boxes, lists, etc. These small pieces are called components. A React component is a small, reusable, and mutable element that displays the data to the UI.

There are several ways to create a React component. The most recent way to create a React component is to write JavaScript functions. We can use either traditional function or arrow function (ES6 or above). Program 2.4 and Program 2.5 demonstrate a simple example of a React component created using function component.

```
const Component = (props) => {
  return (
    <div>
      Hello, {props.name}!
    </div>
  )
}
```

Program 2.4 *React component created using arrow function.*

```
function Component (props) {
  return (
    <div>
      Hello, {props.name}!
```

```
    </div>
  )
}
```

Program 2.5 *React component created using traditional function.*

Before function components were introduced, there were other ways of creating React components. The first method used when React was first introduced is to use React.createClass(). A simple example of React component created using React.createClass() is shown in Program 2.6.

```
const Component = React.createClass({
  displayName: "Component",
  render() {
    return React.createElement(
      "div",
      null,
      "This is a React component"
    )
  }
});
```

Program 2.6 *React component created using React.createClass().*

After that, when JavaScript released the class syntax in ES 2015, React introduced a new way to create a React component. We can then create a new component instance using class syntax from React.Component API. A simple example of React component created using class syntax is shown in Program 2.7.

```
class Component extends React.Component {
  render() {
    return (
      <div>
        Hello, {this.props.name}
      </div>
    );
  }
}
```

Program 2.7 *React component created using class syntax.*

One important feature of React components is their states. States and props both hold data that influences the component. However, props are data that are passed down to the component from its parent (similar to function parameters) and are read-only, while states are the data of the component and managed inside it and are

mutable. The React components in programs above are examples of a ”stateless” component, which means it has no state. Stateless components are easier to manage. However, state gives developers the ability to change the UI of the component whenever the state change without having to do anything. States are used when developers need to keep track of the information within the component between renders. States are first initialize with a default value when the component is created and can be changed over time. In a React class component, the states are initialized with *this.state* inside component's constructor and only mutated indirectly by *this.setState()* function; meanwhile, for function components, a state is initialized along with a mutate function by *useState()* hook. Tow examples of using states in a class component and a function component is shown in Program 2.8 and 2.9 below.

```
class Component extends React.Component {
  constructor(props) {
    super(props);
    this.state = {num: 0};
  }
  addOne() {
    this.setState({ num: num + 1});
  }
  render() {
    return (
      <div>Hello, {this.props.name}</div>
      <div>{num}</div>
      <button onClick={addOne()}>Add one</button>
    );
  }
}
```

Program 2.8 *Using state in a class component.*

```
import React, { useState } from 'react';
const Component = (props) => {
  const [num, setNum] = useState(0);
  return (
    <div>Hello, {props.name}!</div>
    <div>{num}</div>
    <button onClick={() => setNum(num+1)}>Add one</button>
  )
}
```

Program 2.9 *Using state in a function component.*

2.1.4 One-way data binding

Data binding is a process of synchronizing the data source between the provider and consumer automatically. In React and other front-end frameworks, data binding is the connection between the data to be shown in the UI and the logic of the component which contains those data. Frameworks like Angular and Ember use two-way data binding, where the view reflects the change in model's data and the model is updated whenever there is a change in data in the view immediately to keep the data in the view and the model synchronized all the time [9]. Although two-way data binding works fine in most applications, there might be cases where the data flow is unpredictable because both the controller and the view can mutate the model [10]. Instead of using the above approach, ReactJS uses the one-way data binding (or unidirectional data binding) through *Flux* or *Redux* architectures which helps control data flow from a single point. This approach helps developers to gain more control of the application, hence improves application's flexibility.

In ReactJS' one-way data binding (or unidirectional data binding), only the components can perform changes to the view directly. As mentioned in 2.1.2, a change in a state of a component can trigger re-rendering the view of that component. However, we still need to use the view of the component to update the state of the component such as clicking some buttons to make the web app to dark mode, or making the text field to show what we typed in it. The only way to do it is to add event handlers to the view elements and use them to change the data in the component since React does not allow us to mutate the component directly. An example is shown in Figure 2.3 below where a simple application contains one text input and one paragraph showing the value of the text input.

```
import { useState } from "react";
import "./styles.css";

export default function App() {
  const [text, setText] = useState("Default text");
  const eventHandler = (e) => setText(e.target.value);
  return (
    <div className="App">
      <input type="text" value={text} onChange={eventHandler} />
      <p> The value of text box is {text}.</p>
    </div>
  );
}
```

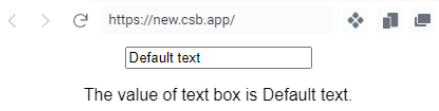

Figure 2.3 *State and event handling in React.*

In this example, the data flow is from the component to the view: Whenever the value of the "text" state changes, the content in the text input and the paragraph

will be updated. Furthermore, we can also apply changes to the component from the view's text input through the defined "eventHandler".

2.2 Why Learn ReactJS

2.2.1 ReactJS' advantages

There are several key reasons why developers should choose to learn ReactJS. First and foremost, ReactJS is easy to learn. Comparing to other JavaScript frameworks such as AngularJS, which require developers to spend more time studying the concept of the framework [10], React is much easier to start with. Thanks to its simple and straightforward nature, ReactJS allows developers with some basic knowledge of JavaScript to get familiar to its structure quickly. It also provides a great comprehensibility, which eases the study process for even beginners. We can say that it does not require much effort to start running a React application and develop it.

Another key factor that helps React to stand out of so many frameworks at the moment is its highly efficient performance. As explained in 2.1.1, the uses of virtual DOM drastically boost the performance of React applications. Furthermore, because of the independent property of React, a team or many teams of developers can work on different features of the same application without having to understand what others are doing [11]. This is archived since changes in a React component will not affect the logic of the entire application. In addition, the components can be reused to accelerates the process of development [12].

Currently, React is supported by a large development community. React is being used in production by many big companies such as Facebook, New York Times, Airbnb, Netflix, etc [13]. Therefore, it is still being supported and developed by these companies. Not only the companies but also independent contributors, developers are regularly updating, adding new features to the framework, which helps making React up-to-date technologies [11]. Beside, React is also supported with a huge amount of tool-sets and third-party libraries such as Material-UI, Styled Components, Redux, resulting in a great support for React developers.

React also supports all modern browsers such as Edge, Firefox, Chrome, etc. However, even for older browsers which do not support ES5 methods or microtasks, developers can still include polyfills in the application bundle to make it works.

2.2.2 React's best use cases

Although React is a powerful tool, it does not fit all types of applications. In this section, the top use cases for React will be discussed.

First of all, React is a good approach to implement interactive and data-intensive dashboard applications. Since React is component-based, developers can easily re-use chart components with appropriate UI division [11]. Additionally, data intensive dashboards require real-time update of the components, which can be solve easily with React's virtual DOM.

Single Page Applications (SPAs) is also a common use case of React thanks to the existence of React-Router. React-Router is a routing library created to assist SPAs built using React [14], which enable efficient navigation experience to users. React's server side rendering also allows developers to implement complicated views.

Since React is created and first used by Facebook [15], the top use case for React so far is for building social networking applications. In social networking applications, the content of the page is reloaded depending on the user's requests. The view changes can be implemented as SPAs which can be built using React. Social networking apps also need real-time notifications. This can also be solved by automating the communication between the client side and the server side using web socket, which accelerates the data flow drastically, allowing the application to create real-time notifications.

Another good use of React is for building E-Commerce applications. Similar to social networking applications, E-Commerce apps can be divided to several views which can be built as SPAs. Code re-usability also offer great assistance as long as the UI is logically modulated.

3 History and evolution of ReactJS

This chapter will discuss the brief history and evolution of the Web development technologies and then discuss the evolution of ReactJS in detail.

3.1 Web application development's brief history.

3.1.1 Creation of the web and HTML.

The idea of the web originated in 1989 by Tim Berners-Lee, started from a suggestion to create a global hypertext space that contains information that can be accessed in the network and referred to by an identifier [16]. Then this dream expanded to the creation of a common information space for people to share information. This state of the web is called web 1.0, which is mainly read-only and there was no interaction between the user and the web page. Figure 3.1 shows a screenshot of the world's first website, called World Wide Web.

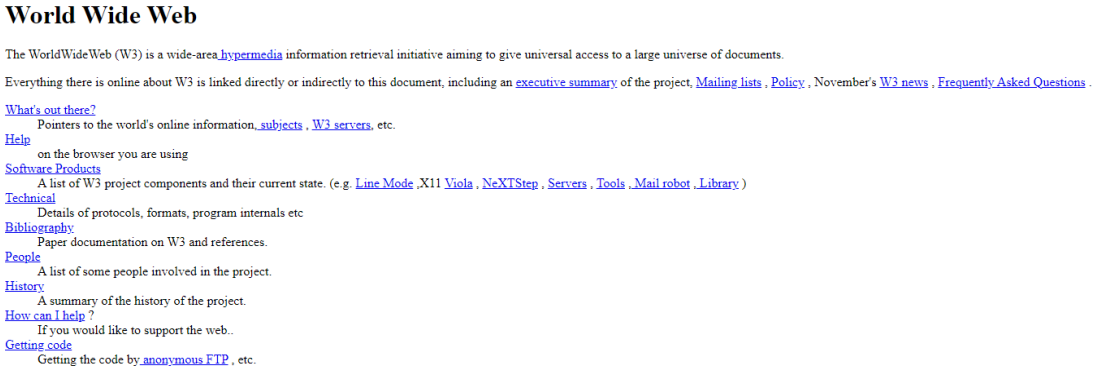

Figure 3.1 *A screenshot of the world's first website [17].*

Tim Berners-Lee is also the creator of the Hypertext Markup Language (HTML) in the early 1991, which was used for marking and formatting the World Wide Web documents. HTML was developed and had many iterations released throughout the following years. HTML v2 was released in 1995, followed by HTML v3 and v4 in 1997, and finally HTML5 was released in 2011 [18].

3.1.2 Creation of CSS and CSS frameworks.

Although HTML has evolved, but it can only be used for formatting the document. The need of styling the web page leads to the birth of Cascading Style Sheets (CSS). The first draft of CSS proposal was introduced by Håkon Wium Lie in 1994 [18]. The first version of CSS was released in 1996 by World Wide Web Consortium (W3C), allowing user to change the font-size and the color of text or background. About two years later, in 1998, W3C launched the next version of CSS called CSS2, with many new features such as the ability to adjust the position of elements relative to the flow of the document, media support, etc. The latest version of official CSS released by W3C, also known as CSS3, was launched in 1999 and is still being used at the moment [19].

CSS being introduced is a big leap in the web development, however, there are still many problems related. One of the biggest problem was that, in the beginning, standardized CSS specifications were not supported in most browsers, which led to the fact that a web page can be perfectly fine in one browser and a mess in another [20]. An example of a cross browser compatibility issue in a browser test called Acid2 test is shown in Figure 3.2, where the behavior of CSS differs in Firefox v1.0.2 and Internet Explorer 6 sp2 [21]. At the time, most browsers were affected in some aspect, but thanks to the help of an organization called the Web Standards Project and developer communities, most of CSS issues have been resolved in major browsers.

Figure 3.2 *Cross browser incompatibility between Firefox v1.0.2 and IE 6 sp2 [21].*

Beside the standard CSS, sometimes browser vendors want to add experimental CSS properties and at the same time not breaking the developers' code with standard

CSS. In order to do that, they add some prefix to these properties depending on which browser they are using. For example, *-webkit-* prefix is used in Chrome, Safari, recent versions of Opera, most IOS browsers and any Webkit base browser, *-moz-* for Firefox, and *-ms-* for Internet Explorer and Microsoft Edge. Although these prefixes are only created for experimental intention, the developers have been using them in production code for web applications. This leads to difficulties in maintaining compatibility and makes smaller browsers add popular prefixes to load common websites [22].

The creation of CSS allows developers to decorate the web applications. However, the web applications keep getting more and more complicated, which leads to a significant increment in the amount of CSS code in each development. Around the 2000s, a variety of CSS frameworks and libraries were introduced such as Blueprint [23], YUI Grids [24], and YAML [25]. They helps developers with implementing the layout of web applications by offering a grid system. Furthermore, these frameworks are supported by most browsers, so developers do not have to worry about the compatibility issues discussed above. The use of these frameworks drastically improve the productivity of the development of web application. They also provide a standardized code base since in order to use a framework, developers have to follow the framework's naming convention, style guides, and architecture [26].

3.1.3 Creation of JavaScript and JavaScript frameworks.

JavaScript is one of the most popular programming languages. According to StackOverflow's 2021 Developer Survey, JavaScript is the most used programming language with over 68 percents of professional developers claiming to use it [27]. It is called the language of the web since most modern web applications are using it.

History of JavaScript

In 1994, a browser called the Netscape Navigator was created and quickly became the most used browser [28]. At the time, all web pages are only static since they are just made of HTML for structuring and CSS for styling. Due to a rise in demand for browsers to support interactions between users and the web pages like computer programs, Netscape (the company created Netscape Navigator) decided to create a scripting language to enable dynamic behavior in the web pages so

that their browser can get even more popular. At the same time, Java, an object-oriented programming language created by company Sun Microsystems, was widely used. A feature of Java was Java Applet, which is a small program written in Java and embedded in a web page to provide additional features to the application. At first, Netscape intended to cooperate with Sun Microsystems to integrate Java into Netscape browser similarly to Java Applets. However the integration was not done since the Netscape management needed a scripting language instead of Java, but still have the Java-like syntax [28] and could be used with computer programs built with Java.

Brendan Eich, who was just recruited to Netscape on April, 1995, was given the task to create that scripting language based on the requirements from Netscape. The process of creating the prototype called Mocha was done in 10 days in May, 1995. The prototype was demonstrated successfully to Netscape and first published under the name of LiveScript in the first beta release of Netscape Navigator 2.0 in September 1995 [29]. Later, LiveScript was renamed to JavaScript for the official release in March 1996 to take advantage of the popularity of Java [28]. About a year later, JavaScript 1.1 was introduced as a more completed version compare to JavaScript 1.0.

After introducing JavaScript, Netscape started to work on making JavaScript standard for all browsers. At first, Netscape approached the W3 consortium and Internet Engineering Task Force, but neither of them was interested in making a programming language standard [30]. Then Netscape submitted JavaScript to ECMA International, resulting in the release of ECMAScript language specification in June 1997 [29], which continues to present with ECMAScript 2022 being the most recent specification on March 17, 2022.

Creation of JavaScript libraries and frameworks

Similar to CSS, JavaScript also got cross browser compatibility issues that required a bit more effort to make the code run regardless of which browser the code was running. jQuery, released in August 2006, was one of the earliest JavaScript libraries created to help developers deal with cross browser compatibility. It also had functions that helps with making websites interactive. [20]

Before jQuery, another JavaScript technology called Asynchronous JavaScript and XML (AJAX) was introduced. This technique mainly depends on using XML-

HttpRequest object, a concept developed by the Microsoft Outlook Web Access team in 1998 [31]. In 2004, Google implemented a standardized AJAX version on Gmail and Google Maps. This encouraged developers to use AJAX more widely and libraries like jQuery started to support for AJAX [20].

Due to the rise of many libraries and frameworks, managing dependencies also became more essential. In 2012, Bower, a package manager for front-end dependencies, was introduced by Twitter. Bower was used to download the required dependencies from different locations on the web. In 2014, *npm registry* was created and quickly become the world's largest software registry [32].

One problem with jQuery was that it could not handle data persistently when the application contained several shared views. To deal with this problem, a number of frameworks were implemented such as *Backbone*, *Knockout*, and *Ember* [20]. *Backbone* was released in 2010 by Jeremy Ashkenas, being the first framework that aimed at creating SPAs. *Backbone* can be used with or without jQuery and avoid getting to complicated when the applications' size increase.

Around that time, in October 2010, AngularJS was created and published by Adam Abrons and Misko Hevery [33]. Soon after that, Misko Hevery joined Google, resulting in Google's supervision on the framework. AngulartJS quickly became the most used JavaScript MVC framework by offering many features such as two-way data binding, dependency injection, and routing packages. However, this framework got more and more complicated as it grow, which is difficult for new developers to start using this framework. The AngularJS development team then decided to redesign the whole framework, resulting in Angular 2. Despite the similar name, Angular 2 was not backward compatible with AngularJS at all and there was no way to migrate AngularJS applications to Angular 2 but to rebuild the whole project, causing a large number of developers to abandon AngularJS [20].

In 2013, ReactJS was introduced in JavaScript Conference in the United States. The evolution of ReactJS will be discussed in the next section.

After ReactJS, there were a lot of new JavaScript frameworks published such as Vue.js, Next.js, etc.

3.2 Development of ReactJS

3.2.1 ReactJS' background and purpose

In 2004, Mark Zuckerberg launched a local networking platform called *Thefacebook*, which evolved into *Facebook* and *Meta* today. In the beginning of 2010, Facebook began to gain more and more popularity, resulting in a demand of a faster and more efficient speed. Facebook's developers soon introduced xhp, a syntax which was used later in React, into their PHP stack and made it open source. Xhp allows developers to create composite components.

Back then, Facebook's developers were building traditional client-side applications for Facebook Ads based on the Model-View-Controller (MVC) model using two-way data-binding and templates. At first, the application was simple but it was getting more and more complicated as the team started to implement more features. This leads to the recruitment of new developers to handle these new features. They have to deal with big and complicated code bases, which added up and resulted in a slow down in the development as a company. Overtime, they realized their problem with the current approach was that there were so many *Cascading Updates* that they could not keep track of which changes caused the views to update, leading to a huge waste of work on bookkeeping what to update and unpredictable code behaviors. It was clear that their code needed improvements to become more efficient. [34]

At the same time, the Facebook's chat lists were implemented so that whenever there was a change in the list, a string of *innerHTML* would be created and dumped into the DOM. Although this approach could cause some user experience issues, it was so simple and worked fine without bothering the users too much in reality. The developers knew that the model was right, however the user experience need to be improved. Jordan Walke, an engineer at the Facebook Ads department who was frustrated with the low maintainability of the front-end code, wanted to solve the problem with the 'unmaintainable' code base by creating a prototype of something that would make the mentioned process more efficient along with acceptable user experience. [28]

3.2.2 Timeline and development of React

In 2011, Jordan Walke created "FaxJS", a prototype that made manipulating the DOM easier and less error prone. FaxJS was intentionally implemented as a hobby

project and accepted later on by Facebook for further development and introduced to the public [28]. FaxJS was first used for the search element on Facebook [35].

2013 - The year of the beginning

In the beginning of 2012, Facebook Ads became complicated and hard to maintain. ReactJS was officially created by Jordan Walke shortly after that for solving this problem. In the same year, Facebook bought Instagram, a photo sharing network, for one billion USD [36]. After this acquisition, Instagram showed their interest in Facebook's technologies and wanted to get access to them, putting pressure of decoupling React and making it open-source on Facebook.

2013 is a big year for React. React was finally made open-source and published in JavaScript Conference US in May, 2013 as a result of the mentioned pressure. The first impressions of the audiences were not too exciting. They had quite sarcastic responses to the release of React and thought it was a huge step backward. This is because there were misunderstandings that React was made aimed to "innovators" while it was first introduced to "early adopters". Facebook realized this mistake and quickly started a "React tour" right after to resolve this problem and ended up gaining many positive responses [35]. After being released, React continued to be tested and experimented throughout the year across the industry.

There are some remarkable events that show the grow of React in 2013. In June 2, JSFiddle announced to support React on their IDE so that developers can play and get familiar to React [37]. Soon after that, on July 30, a gem called "react-rails" was released to support React and JSX in Ruby on Rails [38]. With this gem, developers can use and update ReactJS effortlessly and write JSX without having to perform an external build step to transform that code to JavaScript. On August 19, PyReact was initially released [39]. The purpose was to enable the use of React and JSX in Python applications by providing an API to transform JSX files to JavaScript and provide access to React source files. An example of JSX usage in python application is shown in Program 3.1 below. During JavaScript Conference EU 2013 on September 14 and 15, Pete Hunt gave a speech of "React: Rethinking best practices", explaining why and how React resolve particular problems.

```
from react import jsx

# For multiple paths, use the JSXTransformer class.
transformer = jsx.JSXTransformer()
```

```
for jsx_path, js_path in my_paths:
    transformer.transform(jsx_path, js_path)

# For a single file, you can use a shortcut method.
jsx.transform('path/to/input/file.jsx', 'path/to/output/file.js')
```

Program 3.1 *Usage of JSX in a python application. [39].*

2014 - The year of expansion

With those efforts, React had gained a quite good reputation and reached the ”early majority” of its potential users. The next step is to expand their empire and aim to bigger clients, such as enterprises. However, just technical benefits at this time is not enough, they had to prove that React is stable. In early 2014, Facebook started *#reactjsworldtour* conferences. The purpose was to resolve misunderstandings from the JavaScript Conference. After this event, they had pulled a big proportion of ”haters” to become their ”advocates”. On January 2, 2014, React Developer Tools, which is an extension to the Chrome Developer Tools [40]. This is a very convenient tool that is still being used today. With React Developer tool, developers can see the structure of the components and as well as inspect and alter the props and states of a particular component. The UI of React Developer Tool back in 2014 is shown in Figure 3.3 below.

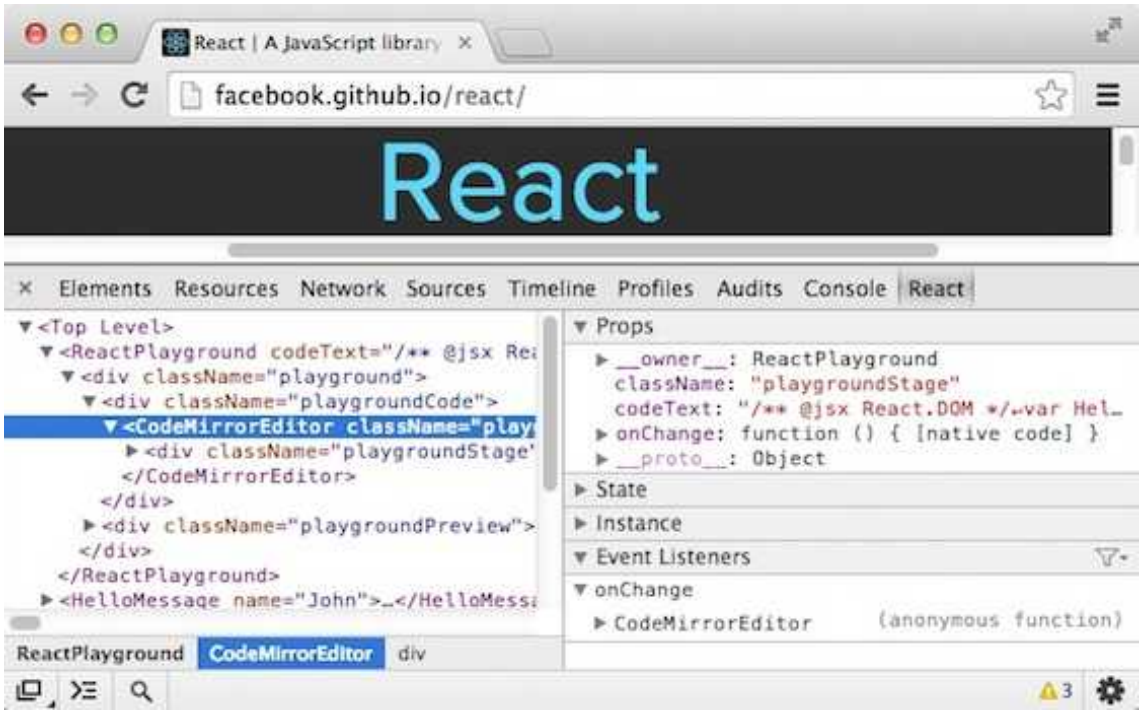

Figure 3.3 *UI of React Developer Tool. Screenshot from React blog [40].*

On April 4, Facebook announced the release of ReactJS.NET, aiming to support the use of React and JSX in .NET MVC web applications. On April 7-9, React London 2014 was held. In this conference, the fundamental principles of the Reactive Manifesto are discussed by a lot of influential developers [41]. On July 13, React Hot Loader was released [42]. This is also a popular plugin because of the ability to live reload React components while preserving their states. On December 12, PlanOut, which is "a multi-platform framework and programming language" [43], was created for online experimentation purposes. PlanOut 0.5 editor supported React, which provide another tool for implementing React applications.

2015 - The year of stability

At this point, React was considered to be "Stable". In 2015, there were plenty of crucial milestones that boosted React's reputation further into the mainstream. In January, Netflix, a popular subscription streaming and filming platform, posted an article called "Netflix likes React" [42]. The article covered the main advantages of React and why React was suitable for Netflix UI implementation [44]. The article shown Netflix's interest as well as support to React. Soon after, Airbnb, a company providing home-stays and vacation renting service, decided to use ReactJS for their

web application. Other companies started to try using React in their applications. Atlassian decided to rebuild their HipChat with React after carefully choosing the suitable JavaScript framework among several candidates such as Angular, Ember, etc [45]. British Broadcasting Corporation (BBC) also rebuilt their homepage with React [46].

These events effectively increased the reputation of React, followed by the first ReactJS Conf ever held on January 28-29. The most noteworthy point was the release of a new technology called React Native. React Native is a framework created to develop mobile applications. React Native was first introduced in ReactJS Conf and officially open-sourced and bring into use on March 26 the same year [47] for iOS version and later for Android on September 14 [48]. It only took React Native three weeks after it got first released to achieve magnificent support from the developer community. Within three weeks, there were "over 12.500 stars, 1000 commits, 500 issues, 380 pull requests, and 100 contributors, plus 35 plugins and 1 app in the app store" [49].

Apart from React Native, there were two other technologies introduced in the ReactJS Conf, namely GraphQL and Relay. They were both created for handling data. GraphQL was designed to be a data query language aimed to deal with complicated nested data while Relay was made as a framework providing data-fetching functionality to React applications using GraphQL for specifying components' data dependencies. Another remarkable event happened in 2015 was the birth of Redux library. Redux was a state management JavaScript library inspired by Facebook's Flux architecture. Redux was introduced on June 2 by Dan Abramov and Andrew Clark, and its version for React called React-Redux was separated to an independent repository in July.

Facebook also held another conference called *ReactEurope* in Paris, France on July 2-3. During the conference, the core team and some members from the community gave speeches about the technologies released earlier namely React Native, Flux, Relay, and GraphQL [50]. Furthermore, a new React Developer Tools was announced on August 3 with many new features to support development [51]. Its first stable version called version 0.14 was released on September 3 [52]. The UI of the new React Developer Tool is shown in figure 3.4 below.

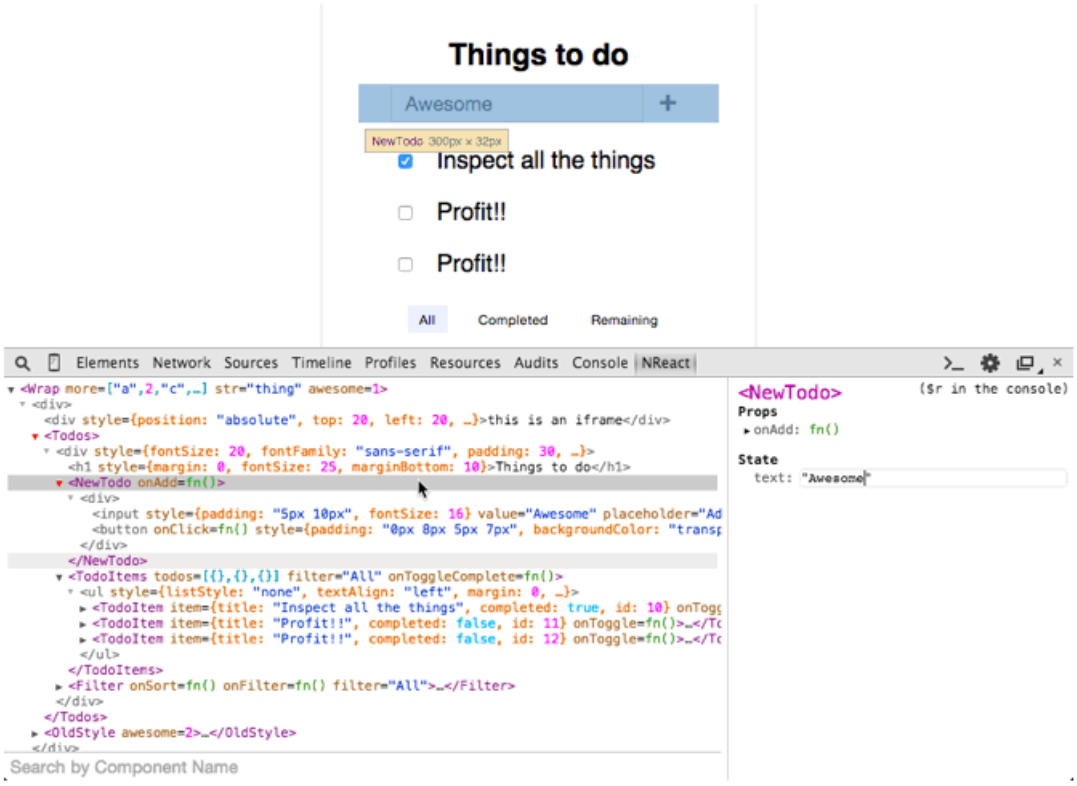

Figure 3.4 *UI of React Developer Tool v3. Screenshot from React blog [52].*

The last remarkable technology released in 2015 was React-Router on November 9, enabling users to navigate between views in the same application.

2016 - The year React got mainstream

After a successful 2015, React continued to gain more Global acknowledgment in 2016 by organizing international conferences such as ReactJS Conf in San Francisco, United State on February 22-23 and ReactEurope 2016 on June 2-3 in Paris, France. Throughout the year, there were plenty of novel technologies as well as supportive libraries introduced. During ReactJS Conf in San Francisco, Salier-Hellendag introduced Draft.js, a rich text editor framework built on top of ReactJS that allows users to apply various styles to the text input field [53]. In March, MobX, which is a state management tools, was introduced. Soon after that, a tool called React Storybook was introduced in the same month. With React Storybook, the development of React applications were made faster and simpler by isolating components, which allowed users to work on one component at once [54]. On July 11, 2016, Facebook

introduced React's Error code system in order to improve programmers' developing experience by providing helpful error messages [55]. In November, Blueprint were introduced as a UI toolkit providing ready-made components that can be used conveniently. The release of these libraries and development tools were highly appreciated by the community.

2017 until now - Further development

ReactJS continued to grow as there were more and more tools introduced, updated, and improved. Plenty of world-wide conferences were organized to deliver new features, technologies and share the point of view of the development team to the community. Especially in 2018-2019, React 16 introduced a lot of novel features to support function components.

React v16.6 on October 23, 2018 witnessed the release of *React.memo()*, which allowed function components to bail out from re-rendering if their props remain the same, and *React.lazy()*, enabling code-splitting with Suspense component [56]. In React v16.8 on February 6, 2019, React Hooks were introduced. React Hooks let developer utilize state and other class components' features on function components [57]. The most used hook is *useState()* hook, which let developers add React states to function components. An example of using React state in function component and equivalent class component is shown in Program 3.2 below:

```
// Function component with useState() hook
function Example() {
  const [count, setCount] = useState(0);
  return (
    <div>
      <p>You clicked {count} times</p>
      <button onClick={() => setCount(count + 1)}>Click me</button>
    </div>
  );
}
// Equivalent class component
class Example extends React.Component {
  constructor(props) {
    super(props);
    this.state = {
      count: 0
    };
  }
  render() {
```

```
    return (
      <div>
        <p>You clicked {this.state.count} times</p>
        <button onClick={() => this.setState({ count: this.state.count
            + 1 })}>Click me</button>
      </div>
    );
  }
}
```

Program 3.2 *useState() hook in function component and equivalent class component*

Another frequently used hook is *useEffect()* hook. With *useEffect()* hook, developers can perform side effects in function components, similar to *componentDidMount()* and *componentDidUpdate()* methods in class components. There are several other React hooks and developers can also create custom hooks. React 16 also included some other supportive features such as Concurrent Mode, allowing React applications to re-render DOM trees without blocking the main thread, therefore improve responsiveness of those applications; Suspense component for data fetching, displaying a load indicator while fetching data to improve user experience.

In August 15, 2019, Facebook released another new React DevTools, offering better performance and navigation experience. The new React DevTools also provided support for React hooks and inspecting nested components [58].

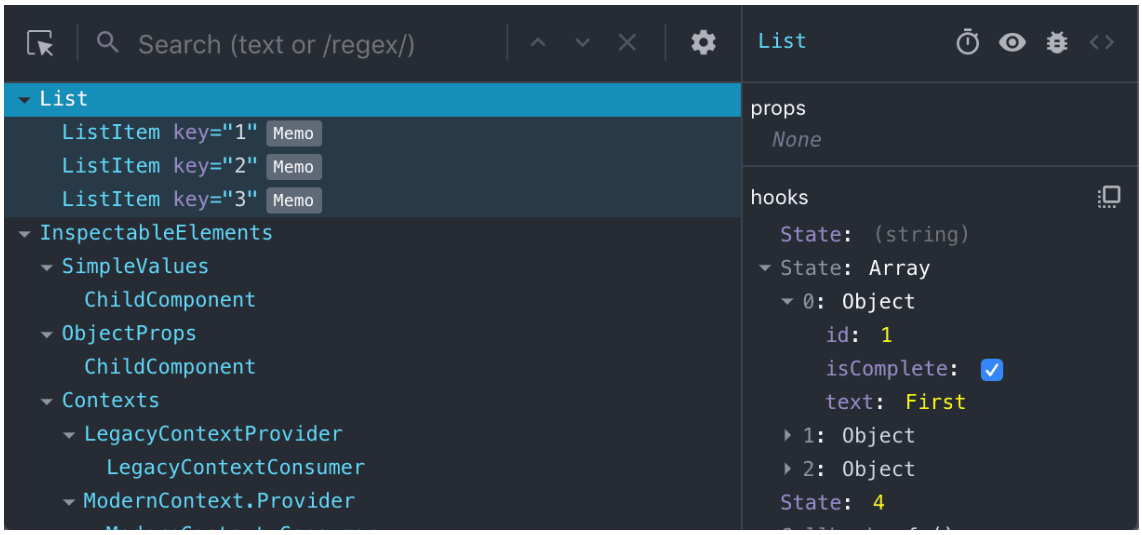

Figure 3.5 *UI of React Developer Tool v4. Screenshot from React blog [58]*

React 17 were released with no major new features. It only focused on making updating React easier, with *Gradual Upgrades* being the most remarkable point, which

will be discussed in the next chapter. On December 21, 2020, Facebook introduced Zero-Bundle-Size React Server Components [59]. A React Server Component is a component that is rendered on the server side and sent to the client to display. An advantage of React Server Component is that it can recover data rapidly and easily, and the component can access data directly since it is rendered on the server side. Zero-Bundle-Size means that when the Server Component is sent to the client, the component itself and its external libraries needed are not include in the client bundle, therefore the client bundle size is not increased at all. This significantly improves the performance of the application.

In React 18, a new server side rendering architecture was introduced, allowing the applications to stream HTML and enable Selective Hydration. Hydration is the process of connecting the JavaScript logic to the HTML code provided by the server side for the whole application. The current process of server side rendering is that, the application must wait for the server to fetch all data for its components and render those components to HTML, then it need time for all JavaScript code to load, and finally hydrates the HTML code before the user can interact with it. This is known as ”all or nothing” rendering and hydration. With the new architecture, the application can now display piece by piece of the UI without having to wait for everything to be ready. [60] Automatic batching is also a remarkable improvement that was introduced in React 18, which reduces the amount of re-rendering, therefore enhance the performance of the applications. After these, there were no significant update made. However, the development tools continued to be improved.

4 The prospected future of ReactJS

In this chapter, the prospected future of ReactJS development will be discussed.

Benefit from its simple learning path, ReactJS easily attracts new users to choose it. In three most recent years, ReactJS was the second most popular web framework in 2019 with 31.3% of all developers' responses [61] and 2020 with 35.9% of all of all developers' responses [62]. In 2021, ReactJS surpassed jQuery to become the most used web framework with over 40% of all respondents claiming using it [1]. With this solid user base and strong community, and a large enterprise's backing up such as Facebook, ReactJS does not seem to disappear from the most used frameworks anytime in the near future. Because of the popularity of ReactJS, most of front-end libraries and tools nowadays provide support for ReactJS, which leads to the fact that the ecosystem around ReactJS will still be expanding and not only depends on the ReactJS development team. This does not mean that the role of the development team is insignificant. In recent versions of ReactJS, Facebook and the ReactJS team have shown their effort in improving the efficiency and stability of ReactJS instead of introducing new features without any maintenance process [63]. This approach is fairly important and favorable for ReactJS in the competition with other web frameworks such as Vue.js.

Another reason why ReactJS will not lose its popularity is that, even with novel innovations, ReactJS has always been backward compatible. Every version of React before React 17 was fully compatible with its previous ones with only some minor breaking points that can be handled quite easily even though React 16 was a "complete rewrite" of the framework since they all share the same public API [64]. When React 17 were introduced, it was even simpler to maintain the compatibility throughout the application. React 17, as mentioned in the previous chapter, enabled gradual React upgrades. Previously in preceding versions, when the user upgrade from one version of React to a newer version, the whole app would be upgraded at once. The problem with this was that there might be some components or some parts of the code base were implemented a long time ago and were not maintained well, which could potentially cause problems with events. That problem was fixed in React 17 that developers then had the option to upgrade the whole application at once like before, or upgrade the application piece by piece. The best option is still upgrade everything at once, but for large applications that are not maintained

regularly, gradual upgrading is a worth considering option [65].

Despite being popular, ReactJS is still incomplete. There will always be room for possible improvements. For instance, error handling is expected to receive improvements. Currently, ReactJS' approach to resolve runtime errors during render process is to throw the error into the component or lifecycle methods, therefore prevent the application from rendering faulty data to the UI. However, this approach does not provide a pleasant user experience. No one can tell if React will still be the best framework in several years later or there will be a new and impressive enough technology to surpass ReactJS' position, but it can be seen from the present that React is doing an excellent job. ReactJS is not perfect, but that also means that they will keep improving in the future.

5 Conclusion

The purpose of this thesis is to study the how and why ReactJS is created and developed to become the most popular web front-end JavaScript framework at the moment, therefore predict the prospected future of ReactJS. The web development became popular since it is more convenient to access services and applications through the Internet than to download and install to a desktop machine. Taking benefits from it, a lot of web development technologies, tools, and frameworks were created and ReactJS was one of them. ReactJS was created in 2011 by Jordan Walke, an employee working for Facebook and was made open-sourced in 2013. The architecture of ReactJS was first questioned by the developer community at first since the approach of ReactJS was unique and different from other frameworks at that time. However, it quickly gained the support of the community by organizing conferences to resolve misunderstandings. Ever since ReactJS was introduced to the public, it has constantly been improving itself by developing novel tools, innovative features along with suitable maintenance plan in order to not only attract new users to choose ReactJS but also keeping the current users to continue using it.

With various strengths and advantages over direct opponent frameworks, along with a good development team and a strong community, ReactJS will likely stay in the group of most used front-end development frameworks in a long period of time. In addition, ReactJS is still being improved consistently to make the user experience even better. ReactJS is not perfect, but it is considered to be quite complete since it can serve well the purposes that it is created for.

In conclusion, ReactJS is a great technology to learn. Its is easy to start and developers will have the chance to learn new knowledge and enrich personal skills along the learning path. The excellent performance and efficiency promise that the future of ReactJS is bright and mastering it is worth the effort.

References

[1] "Stack Overflow 2021 Survey on most commonly used web framework." (2021), [Online]. Available: https://insights.stackoverflow.com/survey/2021#section-most-popular-technologies-web-frameworks. (Accessed: 23/02/2022).

[2] J. Partel. "List of 10 Best Front end Frameworks to Use For Web Development." (2022), [Online]. Available: https://www.monocubed.com/blog/best-front-end-frameworks/. (Accessed: 23/02/2022).

[3] "Introduction to the DOM - web apis: MDN." (n.d.), [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/API/Document_Object_Model/Introduction. (Accessed: 26/02/2022).

[4] Aggarwal, Sanchit, "Modern web-development using reactjs," *International Journal of Recent Research Aspects*, vol. 5, no. 1, pp. 133–137, 2018.

[5] A. Verma. "The Comprehensive Guide to React's Virtual DOM - Medium." (2021), [Online]. Available: https://javascript.plainenglish.io/react-the-virtual-dom-comprehensive-guide-acd19c5e327a. (Accessed: 28/02/2022).

[6] "Introducing JSX – React." (n.d.), [Online]. Available: https://reactjs.org/docs/introducing-jsx.html. (Accessed: 03/03/2022).

[7] A. Mardan, "React Quickly: Painless web apps with React, JSX, Redux, and GraphQL," in New York: Manning Publications, 2017, ch. 3, sec 1. [Online]. Available: https://learning.oreilly.com/library/view/react-quickly-painless/9781617293344/.

[8] "Why can't browsers read JSX." (Oct. 22, 2021), [Online]. Available: https://www.geeksforgeeks.org/why-cant-browsers-read-jsx/. (Accessed: 05/03/2022).

[9] "Angularjs data binding." (n.d.), [Online]. Available: https://www.w3schools.com/angular/angular_databinding.asp. (Accessed: 06/03/2022).

[10] N. I. Naim, "ReactJS: An Open Source JavaScript Library for Front-end Developement," Bachelor's Thesis, Metropolia University of Applied Sciences, May 2017, 36 pp. [Online]. Available: https://urn.fi/URN:NBN:fi:amk-2017053111605, (Accessed: 05/03/2022).

[11] K. Shah. "Why Choose Reactjs For Your Next Project: Features and Benefits." (Jun. 9, 2020), [Online]. Available: https://www.thirdrocktechkno.com/blog/why-choose-reactjs-for-your-next-project-features-and-benefits/. (Accessed: 08/03/2022).

[12] Maratkar, Pratik Sharad and Adkar, Pratibha, "ReactJS - An Emerging Frontend JavaScript Library," *Iconic Research And Engineering Journals*, vol. 4, no. 12, pp. 99–102, 2021.

[13] O. Melnyk. "Top 10 Big Companies Using React." (Feb. 13, 2022), [Online]. Available: https://careerkarma.com/blog/companies-that-use-react/. (Accessed: 08/03/2022).

[14] S. Ganatra, *React Router Quick Start Guide: Routing in React applications made easy*. Packt Publishing, 2018, (Accessed: 10/03/2022).

[15] A. Banks and E. Porcello, *Learning React: Modern Patterns for Developing React Apps*, 2nd ed. O'Reilly Media, 2020.

[16] S. Aghaei, "Evolution of the world wide web : From web 1.0 to web 4.0," *International journal of Web Semantic Technology*, vol. 3, no. 1, pp. 1–10, 2012. DOI: 10.5121/ijwest.2012.3101.

[17] "The World Wide Web project." (n.d.), [Online]. Available: http://info.cern.ch/hypertext/WWW/TheProject.html. (Accessed: 17/03/2022).

[18] T. Lea. "A Brief History of Web Development." (Jan. 27, 2021), [Online]. Available: https://devdojo.com/tnylea/a-brief-history-of-web-development. (Accessed: 14/03/2022).

[19] S. Kumar. "History of CSS." (Nov. 18, 2021), [Online]. Available: https://www.thecrazyprogrammer.com/2021/11/history-of-css.html. (Accessed: 14/03/2022).

[20] M. Wanyoike. "History of front-end frameworks." (Sep. 26, 2019), [Online]. Available: https://blog.logrocket.com/history-of-frontend-frameworks/. (Accessed: 14/03/2022).

[21] P. Freitag. "Web Standards Browser Test." (n.d.), [Online]. Available: https://www.petefreitag.com/item/324.cfm. (Accessed: 15/03/2022).

[22] "Vendor Prefix - MDN Web Docs Glossary: Definitions of Web-related terms | MDN." (Feb. 18, 2022), [Online]. Available: https://developer.mozilla.org/en-US/docs/Glossary/Vendor_Prefix. (Accessed: 17/03/2022).

[23] "Blueprint: A css framework | spend your time innovating, not replicating." (n.d.), [Online]. Available: http://blueprintcss.org/.

[24] "Css grids - yui library." (n.d.), [Online]. Available: https://clarle.github.io/yui3/yui/docs/cssgrids/.

[25] "Yaml css framework." (n.d.), [Online]. Available: http://www.yaml.de/.

[26] "Beloved CSS Frameworks | AGEEK." (Mar. 15, 2022), [Online]. Available: https://ageek.dev/css-frameworks. (Accessed: 17/03/2022).

[27] "Stack Overflow Developer Survey 2021." (n.d.), [Online]. Available: https://insights.stackoverflow.com/survey/2021#most-popular-technologies-language-prof. (Accessed: 17/03/2022).

[28] N. Álvarez-Acebal, "From JavaScript to React. js: Best Practices for Migration," Sep. 2021. DOI: https://doi.org/10.6084/m9.figshare.16553373.v1. [Online]. Available: https://figshare.com/articles/thesis/Bachelor_thesis_pdf/16553373/1.

[29] A. Wirfs-Brock and B. Eich, "JavaScript: the first 20 years," *Proceedings of the ACM on Programming Languages*, vol. 4, no. HOPL, pp. 6–25, 2020. DOI: 10.1145/3386327.

[30] Å. A. A. Kløvstad, "Essay on History of JavaScript," *History of Programming Languages: Collection of Student Essays Based on HOPL IV Papers*, p. 38,

[31] Prof. F. R. Shamil. "History of AJAX." (Mar. 3, 2022), [Online]. Available: https://t4tutorials.com/history-of-ajax/. (Accessed: 20/03/2022).

[32] "About npm | npm Docs." (n.d.), [Online]. Available: https://docs.npmjs.com/about-npm/. (Accessed: 20/03/2022).

[33] A. Barker. "The Super-Brief History of JavaScript Frameworks For Those Somewhat Interested." (Apr. 3, 2018), [Online]. Available: https://dev.to/_adam_barker/the-super-brief-history-of-javascript-frameworks-for-those-somewhat-interested-3m82. (Accessed: 20/03/2022).

[34] T. Occhino [Meta Developers], *React.js Conf 2015 Keynote - Introducing React Native*, Jan. 29, 2015. [Online]. Available: https://youtu.be/KVZ-P-ZI6W4.

[35] "The History of React.js on a Timeline." (Oct. 11, 2021), [Online]. Available: https://blog.risingstack.com/the-history-of-react-js-on-a-timeline/#theneedforabettercode. (Accessed: 01/04/2022).

[36] Segall. "Facebook acquires Instagram for $1 billion." (Apr. 9, 2012), [Online]. Available: https://money.cnn.com/2012/04/09/technology/facebook_acquires_instagram/index.htm. (Accessed: 01/04/2022).

[37] "JSFiddle Integration – React Blog." (Jun. 2, 2013), [Online]. Available: https://reactjs.org/blog/2013/06/02/jsfiddle-integration.html. (Accessed: 01/04/2022).

[38] "Use React and JSX in Ruby on Rails – React Blog." (Jul. 30, 2013), [Online]. Available: https://reactjs.org/blog/2013/07/30/use-react-and-jsx-in-ruby-on-rails.html. (Accessed: 01/04/2022).

[39] "Use React and JSX in Python Applications – React Blog." (Aug. 19, 2013), [Online]. Available: https://reactjs.org/blog/2013/08/19/use-react-and-jsx-in-python-applications.html. (Accessed: 01/04/2022).

[40] Markbåge. "React Chrome Developer Tools – React Blog." (Jan. 2, 2014), [Online]. Available: https://reactjs.org/blog/2014/01/02/react-chrome-developer-tools.html. (Accessed: 01/04/2022).

[41] Instil Software. "React 2014." (n.d.), [Online]. Available: http://reactconf.org/2014/london/. (Accessed: 01/04/2022).

[42] S. Arancio. "ReactJS: A brief history - Stephen Arancio." (Aug. 5, 2021), [Online]. Available: https://medium.com/@sjarancio/reactjs-a-brief-history-3c1e969a477f. (Accessed: 01/04/2022).

[43] Facebookarchive. "GitHub - facebookarchive/planout: PlanOut is a library and interpreter for designing online experiments." (n.d.), [Online]. Available: https://github.com/facebookarchive/planout. (Accessed: 05/04/2022).

[44] J. Kwok. "Netflix Likes React - Netflix TechBlog." (Jun. 19, 2018), [Online]. Available: https://netflixtechblog.com/netflix-likes-react-509675426db. (Accessed: 05/04/2022).

[45] Rmanalang. "Rebuilding HipChat with React.js." (Feb. 10, 2015), [Online]. Available: https://blog.developer.atlassian.com/rebuilding-hipchat-with-react/. (Accessed: 10/04/2022).

[46] A. Hillel. "How we built the new BBC Homepage." (Feb. 16, 2015), [Online]. Available: https://www.bbc.co.uk/blogs/internet/entries/47a96d23-ae04-444e-808f-678e6809765d. (Accessed: 10/04/2022).

[47] Alpert. "Introducing React Native – React Blog." (Mar. 26, 2015), [Online]. Available: https://reactjs.org/blog/2015/03/26/introducing-react-native.html. (Accessed: 05/04/2022).

[48] D. Witte and Weitershausen. "React native for android: How we built the first cross-platform react native app." (Sep. 14, 2015), [Online]. Available: https://engineering.fb.com/2015/09/14/developer-tools/react-native-for-android-how-we-built-the-first-cross-platform-react-native-app/.

[49] "React Native v0.4 – React Blog." (Apr. 17, 2015), [Online]. Available: https://reactjs.org/blog/2015/04/17/react-native-v0.4.html. (Accessed: 05/04/2022).

[50] Johnston. "ReactEurope Round-up – React Blog." (Aug. 13, 2015), [Online]. Available: https://reactjs.org/blog/2015/08/13/reacteurope-roundup.html. (Accessed: 10/04/2022).

[51] Forsyth. "New React Devtools Beta – React Blog." (Aug. 3, 2015), [Online]. Available: https://reactjs.org/blog/2015/08/03/new-react-devtools-beta.html. (Accessed: 10/04/2022).

[52] "New React Developer Tools – React Blog." (Sep. 2, 2015), [Online]. Available: https://reactjs.org/blog/2015/09/02/new-react-developer-tools.html. (Accessed: 05/04/2022).

[53] I. Salier-Hellendag. "React.js Conf 2016 - Isaac Salier-Hellendag - Rich Text Editing with React." Dutch. (Feb. 24, 2016), [Online]. Available: https://www.youtube.com/watch?v=feUYwoLhE_4&feature=youtu.be. (Accessed: 10/04/2022).

[54] "Introduction to Storybook." (n.d.), [Online]. Available: https://storybook.js.org/docs/react/get-started/introduction/. (Accessed: 10/04/2022).

[55] Zhang. "Introducing React's Error Code System – React Blog." (Jul. 11, 2016), [Online]. Available: https://reactjs.org/blog/2016/07/11/introducing-reacts-error-code-system.html. (Accessed: 10/04/2022).

[56] Markbåge. "React v16.6.0: lazy, memo and contextType – React Blog." (Oct. 23, 2018), [Online]. Available: https://reactjs.org/blog/2018/10/23/react-v-16-6.html. (Accessed: 10/04/2022).

[57] "React v16.8: The One With Hooks – React Blog." (Feb. 6, 2019), [Online]. Available: https://reactjs.org/blog/2019/02/06/react-v16.8.0.html. (Accessed: 10/04/2022).

[58] Vaughn. "Introducing the New React DevTools – React Blog." (Aug. 15, 2019), [Online]. Available: https://reactjs.org/blog/2019/08/15/new-react-devtools.html. (Accessed: 10/04/2022).

[59] Abramov, Tan, Savona, and Markbåge. "Introducing Zero-Bundle-Size React Server Components – React Blog." (Dec. 21, 2020), [Online]. Available: https://reactjs.org/blog/2020/12/21/data-fetching-with-react-server-components.html. (Accessed: 10/04/2022).

[60] D. Abramov. "New Suspense SSR Architecture in React 18 · Discussion 37 · reactwg/react-18." (Jun. 5, 2021), [Online]. Available: https://github.com/reactwg/react-18/discussions/37. (Accessed: 13/04/2022).

[61] "Stack Overflow Developer Survey 2019." (n.d.), [Online]. Available: https://insights.stackoverflow.com/survey/2019#technology-_-web-frameworks. (Accessed: 13/04/2022).

[62] "Stack Overflow Developer Survey 2020." (n.d.), [Online]. Available: https://insights.stackoverflow.com/survey/2020#technology-web-frameworks. (Accessed: 13/04/2022).

[63] I. Fatyanova. “The Future of ReactJS Developers.” (Nov. 4, 2020), [Online]. Available: https://dzone.com/articles/the-future-of-reactjs-developers. (Accessed: 13/04/2022).

[64] D. Abramov. “Hey, thanks for feedback! - Dan Abramov.” (Apr. 24, 2017), [Online]. Available: https://medium.com/@dan_abramov/hey-thanks-for-feedback-bf9502689ca4. (Accessed: 13/04/2022).

[65] “React v17.0 – React Blog.” (Oct. 20, 2020), [Online]. Available: https://reactjs.org/blog/2020/10/20/react-v17.html. (Accessed: 13/04/2022).