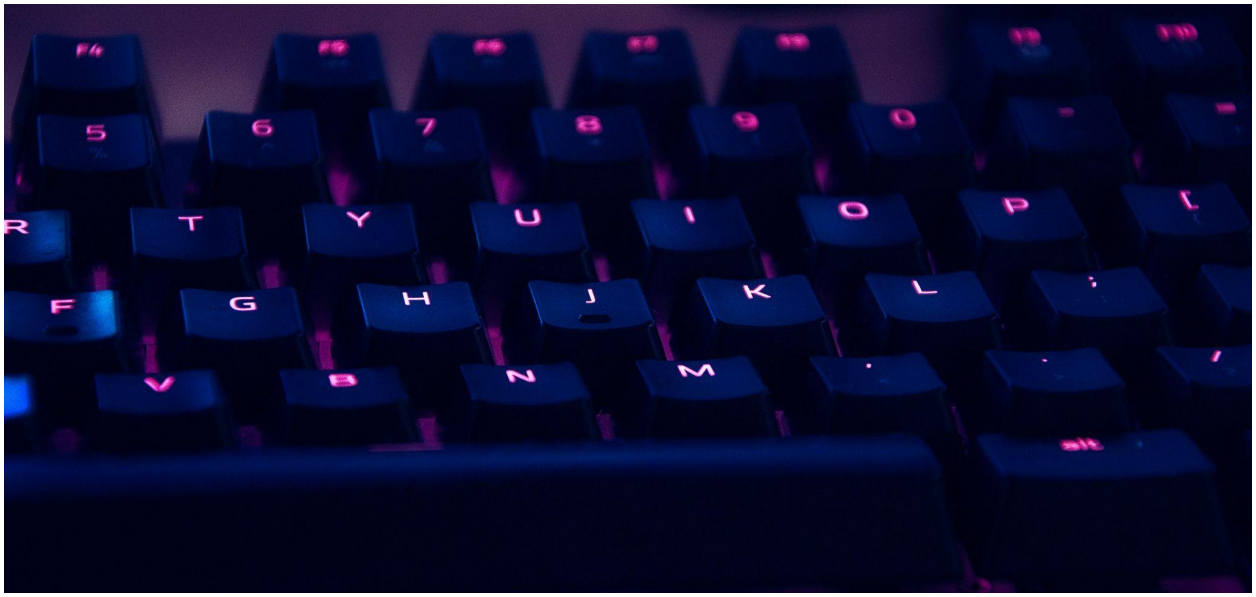


CAB402 Programming Paradigms

React.js Report

Due Date: 31st May 2021

Tafadzwa Manyenga (N9761781)

A report describing the usage of React.js and its programming paradigm, discussing widely adopted code conventions, with brief code blocks. Additional resources used within the discussion are located at the end.

Executive Summary

This report provides an in-depth investigation and analysis of the ReactJS javascript library, and how ReactJS falls under the functional programming paradigm. Methods of analysis generally include experiments, and observation using a basic React application. All supporting documentation, including code samples and architecture related to ReactJS can be found in the appendices. The results of experiments and observation on code showed that ReactJS does in fact fall under the functional programming paradigm, and more specifically falls under the subcategory of declarative programming. React components do in fact show how they can implement functors, higher-order functions, immutable and mutable properties, list comprehension, all with a basic knowledge of JavaScript and HTML.

The report finds the implementation to be a form of function style in terms of factors like function composition. The major areas of implementation are in building User Interface, with a key focus on building reusable components.

Recommendations for implementing shared state amongst multiple components were discussed with React Redux and React Hooks. JSX was also noted to be a useful extension of React that allows less code whilst increasing efficiency in going from concept to deployment.

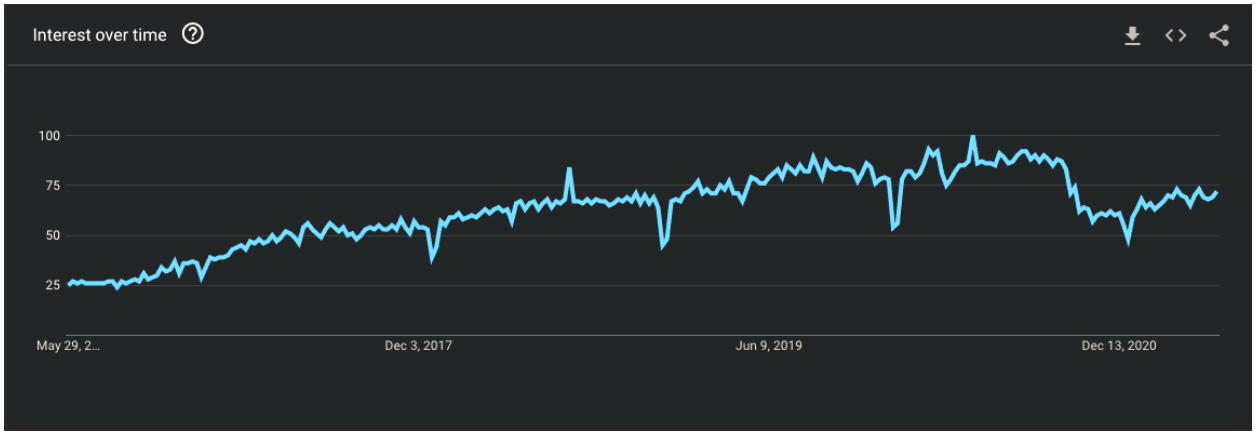
The report also investigates how ReactJS is component based and has a virtual DOM that reduces the demand on resources through re-rendering components if the virtual DOM did not exist.

Table of Contents

1.0 Introduction

1.1 Purpose of the report

React is a Javascript library for building user interfaces ("Getting Started – React", 2021). React.js is popular, open-source, and is used by developers to build websites, and React Native is used to develop native mobile applications for Android and iOS. Most front-end development begins with a javascript framework as a base convention. React is one of the leading frameworks alongside Angular, Vue, Next, Ember, Svelte, and many more. At its conception React was created by Jordan Walke, and the most recent stable release of React is version 17.0 released on 20th October 2020.



(Figure 1.1: Google Trends, ReactJS searches over the past 5 years.)

Single-page applications (SPA) and mobile applications are made with React and cater to a wide audience, for example, Netflix. A SPA is a website or web application that dynamically reloads the current page with new data from the web server, as compared to the default resource expensive way of reloading the whole new page ("Single-page application - Wikipedia", 2021). It is used in many of our preferred applications and websites in many industries and became popular in 2015. Javascript and HTML are the predominantly used programming languages with the JSX syntax extension being used for React “elements”. In the Google Trend above, figure 1.0 depicts how ReactJS was a search term that has risen in popularity over 5 years. The full screengrab is in *Appendix A: Figure 1.0*.

There are different programming paradigms in the world of programming, and in this report, we will explore declarative programming concerning functional programming. In the report, the terms React.js, ReactJS, and React are used interchangeably for functional programming discussion.

1.2 Issues to be discussed and their significance

Early web development involved reloading a page to display the changes made from user input and options. React was created with functional programming in mind and Single-Page applications as an inspiration. With code examples, we will better understand how React.js is an implementation of

functional programming, and we will define functional programming, declarative programming, and its counter imperative programming.

The ideas around React and its implementation are based on React solving the problems from the view layer of the MVC framework (A. M & Sonpatki, 2016). React uses a basic 'element' called **Components**, to handle the view data and program logic, as well as to be rendered to the View itself. This idea of having both the view and the logic together is counter-intuitive at first but it makes sense once we explore the initial React app. We will explore how to react components and the virtual DOM are used to handle program flow and the React program lifecycle. Basic knowledge of javascript assists developers and enthusiasts in understanding how functions are implemented in React. Refer to the Appendix item basic component to understand the basic component that accepts a single attribute (props).

1.3 Research methods

Experiments: The discussion will be aided by experiments on the boilerplate React application in its state after the `npx create-react-app my-app`, `cd my-app`, `npm run` commands. To truly grasp the functional programming in React we require an example application.

Observation: Some observations will be made on the basic project after running create-react-app to determine the different functional aspects of the ReactJS.

1.4 Limitations and assumptions

The reader is assumed to have prior basic knowledge of JavaScript (JS), functions in JS with ES6, and HTML5 with CSS3. The main assumption within this report is that the functional programming paradigm is based on base React without extensions and additional libraries for client-side functionality. Also, the higher-order React functions are assumed to be wrapped components. These HOCs (Higher Order Components) are functions that take a component as input and returns a component that may be altered in some way.

```
const NewComponent = higherOrderComponent(originalComponent);
```

Sample 1.0: Higher Order Component, Adapted from Bulat, (2021)

Softwares

To experiment on the initial React app and create examples required for the research report, a text editor is required. Examples are Atom, VSCode, and Sublime Text amongst others.

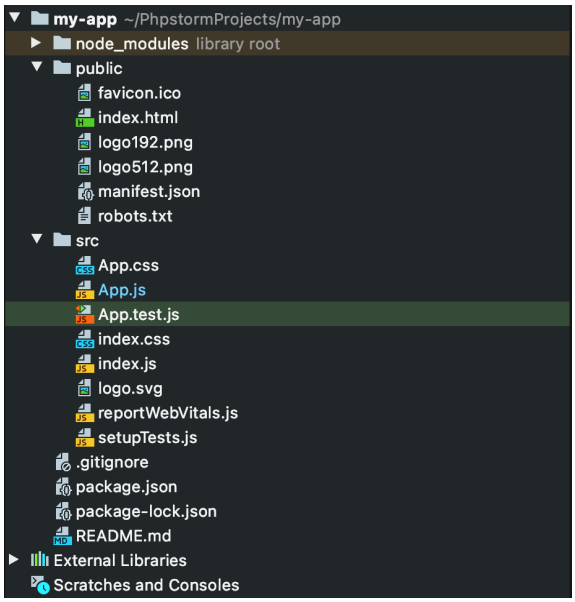
- **Atom:** a basic text editor for multiple languages. Has version control integration with Git, and extensions.
- **VSCode:** Visual Studio Code is a preferred text editor, used for writing code. Has a built-in terminal.
- **Sublime Text:** a bare-bones text editor that focuses on the code being written.
- **ReactJS:** a javascript library for creating single-page applications, and native mobile applications, ("React – A JavaScript library for building user interfaces", 2021).
- **NodeJS: "**asynchronous event-driven javascript runtime", ("Node.js", 2021).
- **NPM: "**Node package manager, the world's largest software registry of JavaScript software and meta-info", ("npm", 2021).

The software resources can be found at Atom.io, Visual Studio Code, Sublime Text, React, Node.js, and npm.

Folder Structure

The screengrab shows the initial folder structure of a React app after running the `create-react-app` command. The main directory is “my-app” and the main sub-directories are “node_modules”, “public”, “src”. “Src” contains the main files that contain the elements, logic, and routing between different areas of the SPA. Node_modules contains the default packages as well as any additional javascript packages installed during development. The “public” folder contains the HTML file that is served to the browser. Please see the appendix item HTML Chart. The page icon and title can be modified from this directory.

Figure 1.2: Folder Structure of a Basic React App



Basic Commands

For a table of basic commands and their results refer to the related Appendix item here: Basic Commands. These commands are used to create an initial React app, as well as to run the app and update additional packages used for state management and routing. Other packages can be installed but for this research, we will stick to the default packages.

Basic Terms and Ideas

For the table with terms and meanings please refer to the related Appendix item here: Basic Terms and Ideas. These ideas are similar to the terms and ideas associated with javascript and web programming in general. Understanding the terms gives the research depth by adding to the semantics of explaining logic associated with functional programming concepts. Semantics by definition is the linguistic aspect of a programming language, whereas syntax is the actual structure of meaningful and usable code. These are basic terms and ideas that will be used for the discussion on research findings.

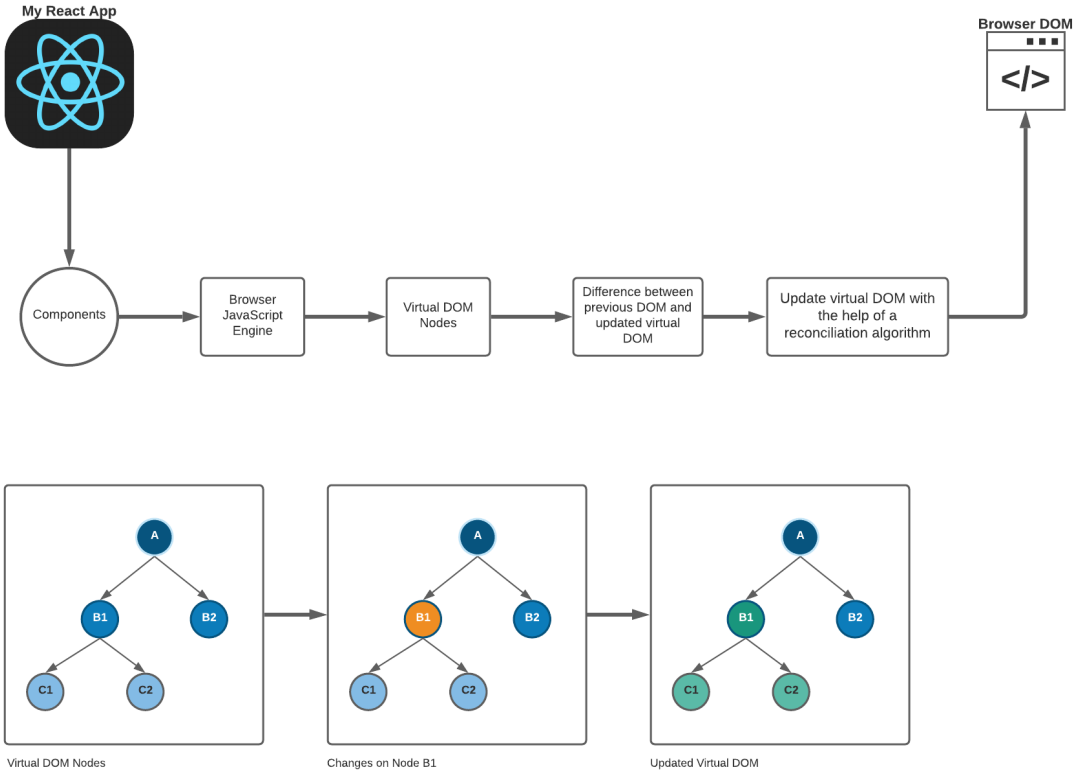
2.0 Discussion

2.1.1 Functional Programming Definition

To understand the functional paradigm of ReactJS we will start with definitions to make distinctions between functional programming language paradigm compared to imperative programming.

“Functional programming is a paradigm where functions are in their pure form mathematically”, (Gabbrielli & Martini, 2010). Imperative programs map directly to the Von Neumann architecture of real computers (Kelly, 2021).

Declarative programming focuses on the desired result of the program, and declarative languages are focused on what is to be done based on the language interpreter concentrating on the result. “Declarative languages can be divided into functional and logic-programming languages”, (Gabbrielli & Martini, 2010). ReactJS falls under this category and renders first to the virtual DOM (Document Object Model). The main reason for this is actual DOM manipulation is expensive on resources and optimizing those virtual DOM operations whilst also rendering to a virtual DOM allows for fewer errors and repetitive code. Please refer to *Appendix E* for the *React Virtual DOM*, for your understanding of how React optimizes the flow of applications. Below is a smaller sample of the React Virtual DOM. The nodes show how changes on one node (B1) affect subsequent nodes and are updated to the virtual DOM before they are rendered to the Browser DOM.



(React Virtual DOM and rendering.)

“Imperative programming is a style of programming where the programmer instructs the computer to execute a sequence of statements (high-level instructions) that can change the state of the program (variables stored in memory)”, (Kelly, 2021). An example of an imperative language is FORTRAN which was utilized for numerical-scientific and engineering arithmetic, through the use of routines and subroutines to execute the sequences of code.

Functional, Class, Nested and Render Components

In React we have different default components that are required to separate the presentation from the logic. The Functional components are components used when a function is called. They form the reusable parts of a React app and return HTML code. Class components are given access to React.component functions by inheritance, this is achieved using the `extends React.component` portion of code. Nested components are UI children components nested under a parent component, for example Section components nested under a Div element. Rendering components are used when compiling the React app into imperative javascript code, and take the reusable components from different modules using `export default functionName;`. These components are imported into the index.js file using import statements. Please refer to the code under App.js and Index.js in the appendix. The ReactDOM.render method takes the <App/> component and renders it to the HTML document under the browser DOM DIV element with ID ‘root’. This is the element in the public directory file index.html. Now we have a basic understanding of how React has a virtualDOM and also how it achieves reusable code.

2.1.2 Code conventions and JSX

The benefit of using React with JSX is that the developer can write HTML syntax within React (Javascript) code, and this code is compiled to Javascript at runtime. The difference between a React application with the use of JSX versus without JSX is shown below. JSX’s main advantage is the reduction in the amount of code being written. Also, it seems to be a bit more intuitive to those who have prior knowledge of HTML. Looking at the examples below, we can notice the difference in a simple function that renders elements with JSX versus without JSX.

App.js with JSX

```
// render with JSX
render: function App() {
 return (
   <div className="App">
     <header className="App-header">
       <img src={logo} className="App-logo"
         alt="logo" />
       <p>
         Edit <code>src/App.js</code> and save to
         reload.
       </p>
       <a
         className="App-link"
         href="https://reactjs.org"
         target="_blank"
         rel="noopener noreferrer"
       >
         Learn React
       </a>
     </header>
   </div>
 );
}
```

App.js without JSX

```
// render without JSX
render: function App(){
 return(React.createElement
         ("div", {key: "App"}, [
             React.createElement ("header",
               {key:"App-header"}),
             React.createElement ("img",
               {key: "App-logo"},
               {alt: "logo"},
               {src: "./logo.svg"}),
             React.createElement ("p", null,
               "Edit <code> src/App.js </code> and
               save to reload."),
             React.createElement ("a",
               {key: "App-link"},
               {href: "https://reactjs.org"},
               {target: "_blank"},
               {rel: "noopener noreferrer"}),
             "Learn React"
         ])
       );
}
```

(Sample 5.0: Comparison of JSX vs without JSX.)

As we can observe, the right code sample has more complicated code as compared to the left, and there are more nested calls for React elements when coding without JSX and ES6. Both render the required application but the difference is the code without JSX is plain React state. “This is React without the

additional default packages to manage state and routing", (McQuay, 2021). For each page element, we can see how the method `React.createElement` is repeatedly called for the div, header, img, p, and a elements. "This method would also be used for custom elements passing an element type in the first argument, and **props** or properties in the second argument, and child elements to the third argument", (McQuay, 2021). The `key` keyword allows React code without ES6 to have elements that can be uniquely identified when running the program and performing operations on different elements. React without JSX or ES6 allows for basic functional code without the complexity of state management and or routing. To explain the React library with a focus on producing fast code, we will discuss using JSX. "Compared to the previous non-JSX example, the JSX code is much more readable, easy to understand, and close to the actual HTML markup", (Vipul & Sonpatki, 2016). Please refer to Sample 6.0 for a no JSX structure of React Elements.

2.2.1 Functional Programming in React.js

JavaScript is a multi-faceted programming language for many functions and styles of programming. The advent of JavaScript libraries and frameworks, it has allowed organized and structured conventions of creation programs with the best efficiency by using reusable code. To understand how ReactJS adheres to the functional programming paradigm we first understood how to create a React app, and explored the folder structure. We have also discussed JSX and are now ready to describe the functional aspects of ReactJS.

ReactJS Expressions

An example of expressions can be understood using an example. In the example below, we have a basic notifier on new grades posted on a university blackboard portal. The expression is wrapped in curly braces. In this example, we can observe a conditional expression combined with an embedded expression. The expression is `{newGrades.length}`. In ReactJS components, expressions are similar to JavaScript expressions. Other examples are `newGrades.course`, `student.firstName`, or `reverse(sampleString)`. Javascript expressions according to Vipul et. al, (2016) are "used for passing props or evaluating some JavaScript code that can be used as an attribute value."

```
function Grades(props) {
   const newGrades = props.newGrades;
   return(
       <div>
           <h2>Grades</h2>
           {newGrades.length > 0 &&
           <h3>
               You have {newGrades.length} new grades from your units.
           </h3>
           }
       </div>
   );
}
```

Adapted from reactjs.org ("React.Component – React", 2021).

Abstraction

With a basic understanding of JavaScript, we can highlight how ReactJS handles the abstraction of any given expression (**exp**) with an identifier (**x**). In the example below, we can see how the function doSquare used the parameter "number" to create an expression "number * number". In React we can pass functions as arguments to other functions, and these functions are components. This will be discussed further in the section on Higher-Order Components (HOCs).

Pseudocode: **`fn x => exp`**
JS:

```
function doSquare (number) {
         if(number > 0)
         {
             return number*number;
         }
         else {throw new Error("out of range"); }}
```

Evaluation

In ReactJS any code that is in curly braces is evaluated by JSX. This holds for child expressions as well as JavaScript expressions. To understand how expressions are evaluated in ReactJS we will take the population of rows in a table.

```
// Evaluating expressions
ReactDOM.render(<App headings = {['When', 'Who', 'Description']}
                     data = {data.length > 0 ? data : ''} />,
   document.getElementById('container'));
```

(JavaScript expressions, A. M & Sonpatki, 2016)

The props are passed as expressions within curly braces, and then when rendering the App component, the JSX expression **`{data.length > 0 ? data : ‘ ’}`** is evaluated first within the nested environment. Nested expressions are evaluated using the props passed at the top level. For a large number of attributes that can be increasing over time, ES6 allows for a “...” operator that can be used within JSX code, as shown in the example below. Of the three evaluation strategies, ReactJS Components that use lazy evaluation have benefits on parallelization of children, memorization, and computation by reducing overheads. However, eager evaluation is important to achieve a balance when building custom components.

```
var props = { headings: headings, changeSets: data, timestamps:
   timestamps };
ReactDOM.render(<App {...props } />,
   document.getElementById('container'));
```

(JavaScript expressions, A. M & Sonpatki, 2016)

RegEx Pattern Matching

RegEx is also known as Regular Expressions. RegEx is a sequence of characters that form a pattern that is used to search strings and determine if the strings contain the pattern or not. In Web Development it is useful for user interfaces that require emails, passwords, and usernames to be entered, for example at the sign-up page. This is part of programming for authenticity, and complex applications of RegEx can be used for multi-factor authentication, security tokens, cryptographic keys, and the like. RegEx can be used for pattern matching in an <input/> field of a form as shown below in the two JSX examples. This is a basic check for a password that contains the characters and symbols, with a wildcard “?” condition for any of the symbols within the “`[ ]`” square brackets. Then following the RegEx we have the code to validate the user input in the interface.

```
export const validPassword = new RegExp('^(?=.*?[A-Za-z])(?=.*?[0-9])(?=.*?[~,`!@#$%^&.*();:"]).{6,}$');
```

(RegEx pattern matching code for password characters and symbols)

```
const validate = () => {
 if (!validPassword.test(password)) {
   false;
```

```
    throw new Error("invalid password");
  }
  else {true; }
};
```

A simple example of evaluating a password using RegEx pattern matching

Props and State

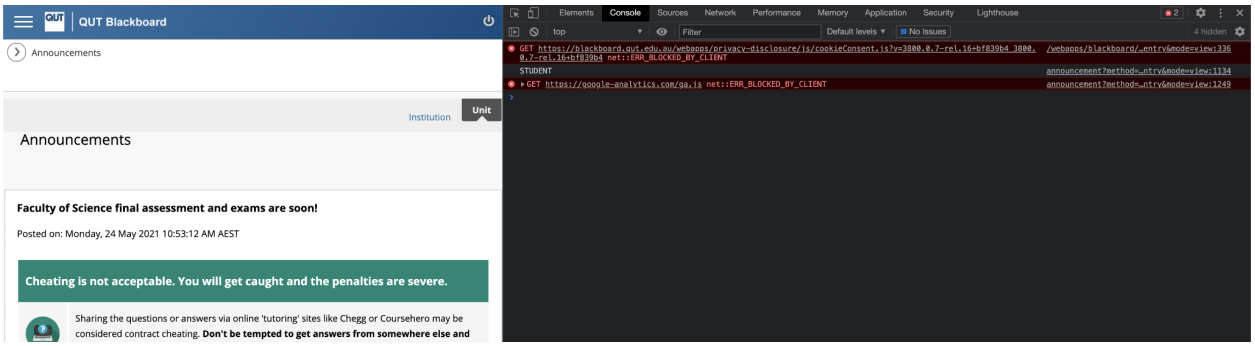
On the topic of state with React apps, the state is dependent on whether components will change as the app is running. In a React app, if Components do not change then there is no need to use state. Props are the attributes that are passed from the parent component to the child component, and these can be used by components without the need to re-render components. This is part of what makes React popular as it leads to efficient apps that minimize re-rendering after the initial rendering of components. Props are immutable in React and generally shouldn't be changed by their passed component.

However elements like checkboxes or dropdown menus have a simple state, usually, a CSS class that checks if something should be displayed on the interface or not. When state is necessary a React Hooks are used to achieve state transitions and updates on the user interface.

2.2.2 React Component Lifecycle and Higher-Order Components

React Component Lifecycle

React Components go through different lifecycle events and they are displayed in a graphic under the Appendix item React.js Lifecycle. The lifecycle shows points where components are initialized and rendered, as well as when external data is fetched. In React the components dynamically retrieve information using promise-based HTTP requests to APIs. The methods for dynamically handling data are GET and POST, where GET retrieves information such as through an API, and POST sends a post request. When a data request is completed components may update based on success or errors. HTTP status codes are used to determine the response to the GET and POST requests, and these include 200 OK for success, 403 forbidden, 404 not found, and many more. An example of a GET request and some kind of component update for the CAB402 Announcements page can be observed using "inspect sources" in the Chrome browser.



(Example of GET request on QUT blackboard site)

Generally, components begin in a Start state, and based on user interface events such as mouse click events, or button clicks, they transition to componentWillMount, render, componentDidMount, shouldComponentUpdate, componentWillUpdate, render in that order. Whether they are triggered is based on how the user interacts with the application as some applications have functionality that is broad and some components might not be used, for example, components linked with the login/sign up when a

user intends on being a guest/anonymous user. HTTP requests are also handled with the back-end programming side of development. ReactJS is used for front-end client-facing applications.

Higher-Order Components

In React Higher-Order Components (HOCs) are functions that take an already defined component, and return a new component that takes the props of the original component. These are useful for injecting any additional data into components, thus extending the functionality of components. This can also contribute to the design of components by keeping the logic separate from the component. React components are reusable and usually only contain the logic for one purpose. An example is how routing for React apps is achieved with HOCs and this shows how we can "simulate" between different pages whilst maintaining the single-page aspect of the application. The most widely adopted package to abstract the modular logic of Routing in React is react-router-dom. Below we have an example of how we implement higher-order components.

For an in-depth code sample showing the implementation of higher-order components, please refer to the Appendix item higher-order components implemented with react-router-dom.

```
import { withFunctions } from 'my-module';
class OriginalComponent extends React.Component {
   ...
}
export default withFunctions(OriginalComponent);
```

Higher Order Components (Bulat, 2021)

Functors in ReactJS (javascript)

In ReactJS we can implement code for 'something' that can be mapped over. Arrays in javascript have the methods under **`array.prototype`** that allows for **`map`**, **`filter`**, **`reduce`** functions. In this example, we will use an array as the 'something' that can be mapped over. Considering the code sample under appendix item Sample 8.0, we can see how an original array [1,2,3] becomes [2,4,6] when the map method is used. We can observe how a set of 'things' can be mapped to attain a 'new' set of things under the original structure. This is very useful for props and data that will be used by components when they are rendered and reduces the need for re-rendering by applying new logic to the modules if some behavior is expected on the user interface.

2.2.4 Mutability with React.js

In React mutable components change value when you use them, and immutable components don't change value. The values associated with a component are usually the props. Props that are related to strings for example, do not change. If we were to take string str1 which has the value "hello world", applying the string method **`slice()`** to create str2 and str3 doesn't change the original string str1. The example below shows this example of immutable strings. Numbers, arrays and objects are also immutable and when referencing the original value, the original will be preserved.

```
var str1 = "hello world";
var str2 = str1.slice(0,5);
console.log(str2); // "hello"
var str3 = str1.slice(5,10);
console.log(str3); // "world"
console.log(str1); // "hello world"
```

Sample 9.0 Immutability of strings in JS

2.3 React State Management, React-Redux and React Hooks

Shared Data flows from the parent component to children through inheritance. The properties are set at the parent component and passed onto the child components. The child component then sets its internal state using the props passed down and initialized internally. This works well for simple applications, but for more complex applications React-Redux is implemented for children that access and update the same state and properties.

React-Redux allows developers to store the state of multiple components in one big container. This solves the issues that come from multiple child components or parallel components using and updating the shared state. This preserves the UI framework through binding interaction logic and preserving state.

React Hooks were introduced in version 16.8 and there are two main types that are useState and useEffect hooks. A Hook is a special function that lets you “hook into” React features, ("Using the State Hook – React", 2021). Instead of refactoring code of a function component to use state, we can implement state within the function. An example can be found in the appendix under Sample 10.0

Conclusion

ReactJS is a strong framework for developing component-based user interfaces. It allows the developer to go quickly from concept, to prototype, to product and allows for scalability. The highlight is reusable code, and these reusable components provide a simple but robust architecture. The programming paradigm of React can be observed to be functional, but falling under the declarative paradigm of functional programming. React functions and components are immutable and elements as well, allowing for consistency throughout the application. A strong understanding of JavaScript is required however a basic understanding can still support the developer in understanding concepts such as arrow functions, monoids, classes, expressions and evaluation. The virtual DOM is especially advantageous as it reduces the demand on resources thus increasing the performance of React apps. React has a strong potential for business and commercial use because of its memory and performance optimization. Using additional packages is also important as it allows for custom components, assisting in developing with efficiency, scalability and adhering to the CIA triad for application security.

References

A hackable text editor for the 21st Century. Atom. (2021). Retrieved 27 May 2021, from https://atom.io/.

A. M, Vipul., & Sonpatki, P. (2016). *ReactJS by example*. Packt Publishing.

About npm | npm Docs. Docs.npmjs.com. (2021). Retrieved 27 May 2021, from https://docs.npmjs.com/about-npm.

Bulat, R. (2021). *How to Use React Higher-Order Components*. Medium. Retrieved 25 May 2021, from https://rossbulat.medium.com/how-to-use-react-higher-order-components-c0be6821eb6c.

Code, V. (2021). *Visual Studio Code - Code Editing. Redefined*. Code.visualstudio.com. Retrieved 27 May 2021, from https://code.visualstudio.com/.

Design Principles – React. Reactjs.org. (2021). Retrieved 25 May 2021, from https://reactjs.org/docs/design-principles.html.

Gabbrielli, M., & Martini, S. (2010). *Programming Languages: Principles and Paradigms*. Springer.

Getting Started – React. Reactjs.org. (2021). Retrieved 25 May 2021, from https://reactjs.org/docs/getting-started.html.

House, C. (2021). *React Stateless Functional Components: Nine Wins You Might Have Overlooked*. Medium. Retrieved 25 May 2021, from https://medium.com/@housecor/react-stateless-functional-components-nine-wins-you-might-have-overlooked-997b0d933dbc.

Kelly, W. (2021). *Week 1 - Turing Machines, Lambda Calculus and Von Neumann Architecture*. Presentation, Queensland University of Technology - Gardens Point, Brisbane, Australia.

Kelly, W. (2021). *Week 2 - Introduction to F#*. Presentation, Queensland University of Technology - Gardens Point, Brisbane, Australia.

McQuay, D. (2021). *Use React without JSX Or ES6 | Pluralsight | Pluralsight*. Pluralsight.com. Retrieved 27 May 2021, from https://www.pluralsight.com/guides/just-plain-react.

Node.js. Node.js. (2021). Retrieved 27 May 2021, from https://nodejs.org/en/.

npm. Npmjs.com. (2021). Retrieved 27 May 2021, from https://www.npmjs.com/.

React Lifecycle Methods - The Basics. DEV Community. (2021). Retrieved 25 May 2021, from https://dev.to/stlnick/react-lifecycle-methods-4lh4.

React.Component – React. Reactjs.org. (2021). Retrieved 25 May 2021, from https://reactjs.org/docs/react-component.html.

React Hooks – React. Reactjs.org. (2021). Retrieved 30 May 2021, from https://reactjs.org/docs/hooks-state.html.

Single-page application - Wikipedia. En.wikipedia.org. (2021). Retrieved 27 May 2021, from https://en.wikipedia.org/wiki/Single-page_application.

Sublime Text - the sophisticated text editor for code, markup and prose. Sublimetext.com. (2021). Retrieved 27 May 2021, from https://www.sublimetext.com/.

Walke, J. (2021). *React (JavaScript library) - Wikipedia*. en.wikipedia.org. Retrieved 25 May 2021, from https://en.wikipedia.org/wiki/React_(JavaScript_library).

Appendix

Appendix A: Code Examples

Sample 1.0: Higher Order Component

```
const NewComponent = higherOrderComponent(originalComponent);
```

Sample 1.1 Basic Component

```
function Welcome(props) {
  return <h1>Hello, {props.name}</h1>;
}
```

Sample 2.0: app.js

```
import logo from './logo.svg';
import './App.css';

function App() {
 return (
   <div className="App">
     <header className="App-header">
       <img src={logo} className="App-logo" alt="logo" />
       <p>
         Edit <code>src/App.js</code> and save to reload.
       </p>
       <a
         className="App-link"
         href="https://reactjs.org"
         target="_blank"
         rel="noopener noreferrer"
       >
         Learn React
       </a>
     </header>
   </div>
 );
}

export default App;
```

Sample 3.0: index.js

```
import React from 'react';
import ReactDOM from 'react-dom';
import './index.css';
import App from './App';
import reportWebVitals from './reportWebVitals';

ReactDOM.render(
 <React.StrictMode>
   <App />
 </React.StrictMode>,
 document.getElementById('root')
);

reportWebVitals();
```

Sample 4.0: /public/index.html

```
<!DOCTYPE html>
<html lang="en">
 <head>
   <meta charset="utf-8" />
   <link rel="icon" href="%PUBLIC_URL%/favicon.ico" />
   <meta name="viewport" content="width=device-width, initial-scale=1" />
   <meta name="theme-color" content="#000000" />
   <meta
     name="description"
     content="Web site created using create-react-app"
   />
   <link rel="apple-touch-icon" href="%PUBLIC_URL%/logo192.png" />
   <!--
     manifest.json provides metadata used when your web app is installed on a
     user's mobile device or desktop. See
https://developers.google.com/web/fundamentals/web-app-manifest/
   -->
   <link rel="manifest" href="%PUBLIC_URL%/manifest.json" />
   <!--
     Notice the use of %PUBLIC_URL% in the tags above.
     It will be replaced with the URL of the `public` folder during the build.
     Only files inside the `public` folder can be referenced from the HTML.

     Unlike "/favicon.ico" or "favicon.ico", "%PUBLIC_URL%/favicon.ico" will
     work correctly both with client-side routing and a non-root public URL.
     Learn how to configure a non-root public URL by running `npm run build`.
   -->
   <title>React App</title>
 </head>
 <body>
   <noscript>You need to enable JavaScript to run this app.</noscript>
   <div id="root"></div>
   <!--
     This HTML file is a template.
     If you open it directly in the browser, you will see an empty page.

     You can add webfonts, meta tags, or analytics to this file.
     The build step will place the bundled scripts into the <body> tag.

     To begin the development, run `npm start` or `yarn start`.
     To create a production bundle, use `npm run build` or `yarn build`.
   -->
 </body>
</html>
```

Sample 5.0: App.js with JSX vs without JSX

App.js with JSX

```
// render with JSX

render: function App() {
 return (
   <div className="App">
     <header className="App-header">
       <img src={logo} className="App-logo"
         alt="logo" />
       <p>
         Edit <code>src/App.js</code> and save to
         reload.
       </p>
       <a
         className="App-link"
         href="https://reactjs.org"
         target="_blank"
         rel="noopener noreferrer"
       >
         Learn React
       </a>
     </header>
   </div>
 );
}
```

App.js without JSX

```
// render without JSX

render: function App(){
 return(React.createElement
          ("div", {key: "App"}, [
              React.createElement ("header",
                {key:"App-header"}),
              React.createElement ("img",
                {key: "App-logo"},
                {alt: "logo"},
                {src: "./logo.svg"}),
              React.createElement ("p", null,
                "Edit <code> src/App.js </code> and
                save to reload."),
              React.createElement ("a",
                {key: "App-link"},
                {href: "https://reactjs.org"},
                {target: "_blank"},
                {rel: "noopener noreferrer"}),
              "Learn React"
          ])
        );
}
```

Sample 6.0: Structure of React Element without JSX

```
React.createElement ( [type], [props], [child...]  )
```

Sample 7.0: higher-order components

https://github.com/TafadzwaManyenga/paid/blob/master/src/App.js

Sample 8.0: Functors

```
console.log([1,2,3].map(x => x * 2)))
//[2,4,6]
```

Sample 9.0 Immutability of strings in JS

```
var str1 = “hello world”;
var str2 = str1.slice(0,5);
console.log(str2); // “hello”
var str3 = str1.slice(5,10);
console.log(str3); // “world”;
```

Sample 10.0 React Hooks

```
class Example extends React.Component {
  constructor(props) {
    super(props);
    this.state = {
      count: 0
    };
  }
```

In a function component, we have no `this`, so we can't assign or read `this.state`. Instead, we call the `useState` Hook directly inside our component:

```
import React, { useState } from 'react';

function Example() {
  // Declare a new state variable, which we'll call "count"
  const [count, setCount] = useState(0);
```

Reading State

When we want to display the current count in a class, we read `this.state.count`:

```
  <p>You clicked {this.state.count} times</p>
```

In a function, we can use `count` directly:

```
  <p>You clicked {count} times</p>
```

Updating State

In a class, we need to call `this.setState()` to update the `count` state:

```
  <button onClick={() => this.setState({ count: this.state.count + 1 })}>
    Click me
  </button>
```

In a function, we already have `setCount` and `count` as variables so we don't need `this`:

```
  <button onClick={() => setCount(count + 1)}>
    Click me
  </button>
```

("Using the State Hook – React", 2021)

Appendix B: Figures

Figure 1.1: Google Trends, ReactJS searches over the past 5 years.

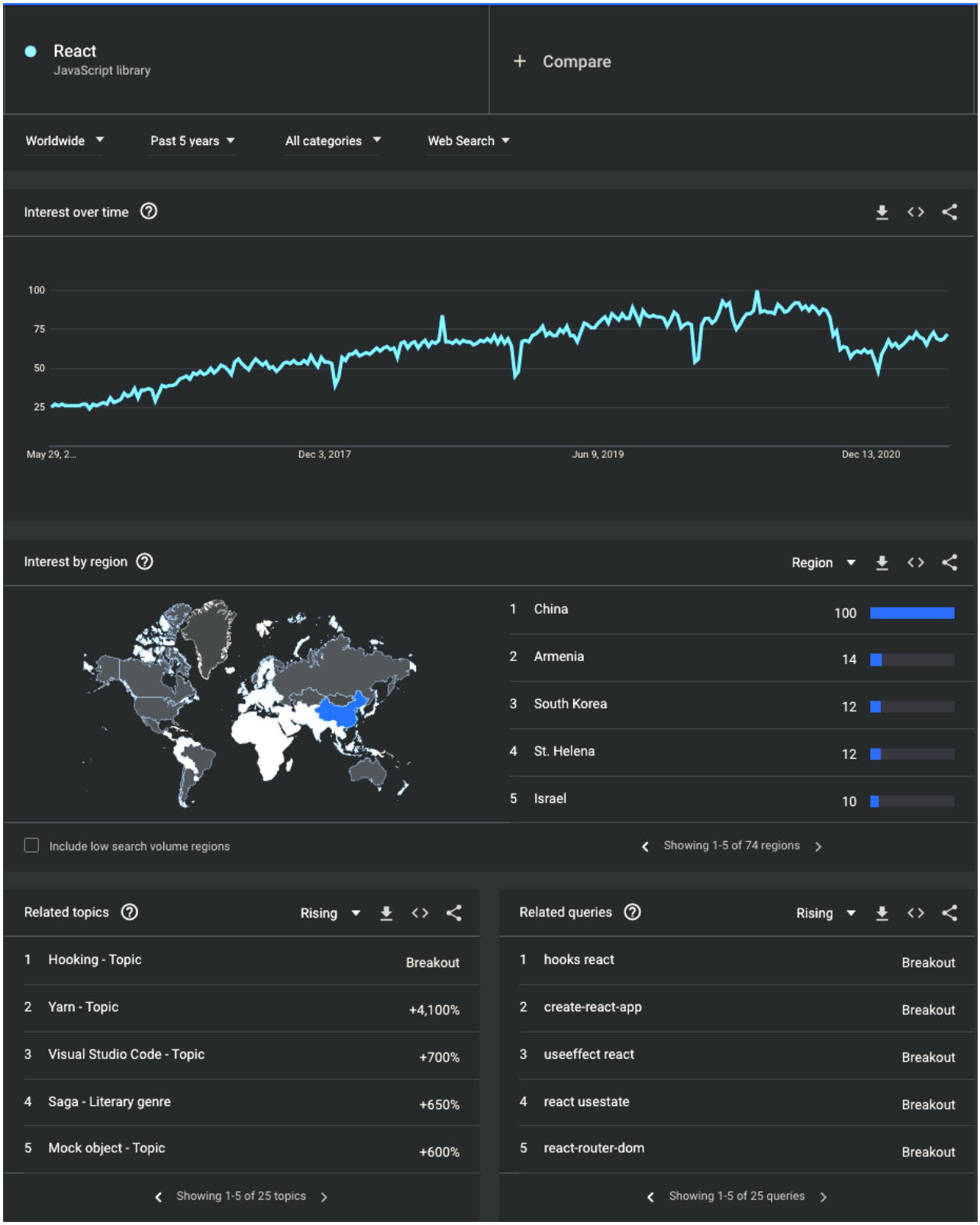
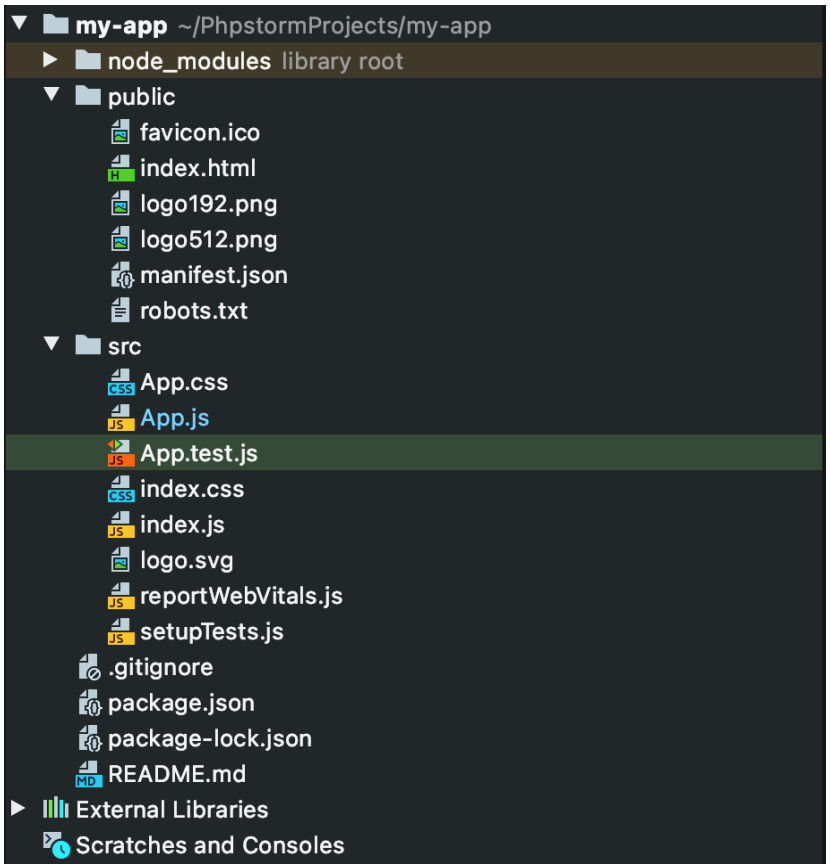
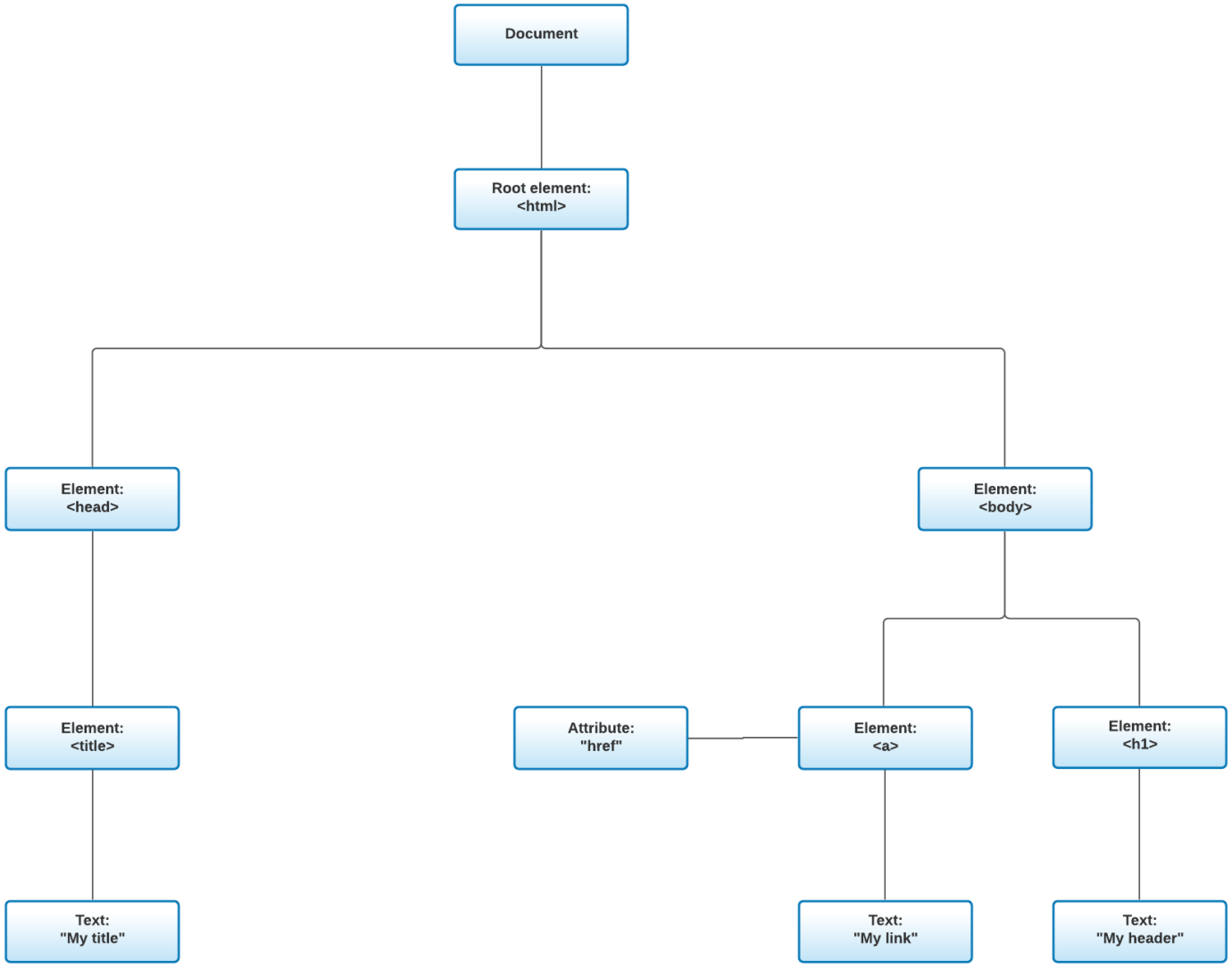


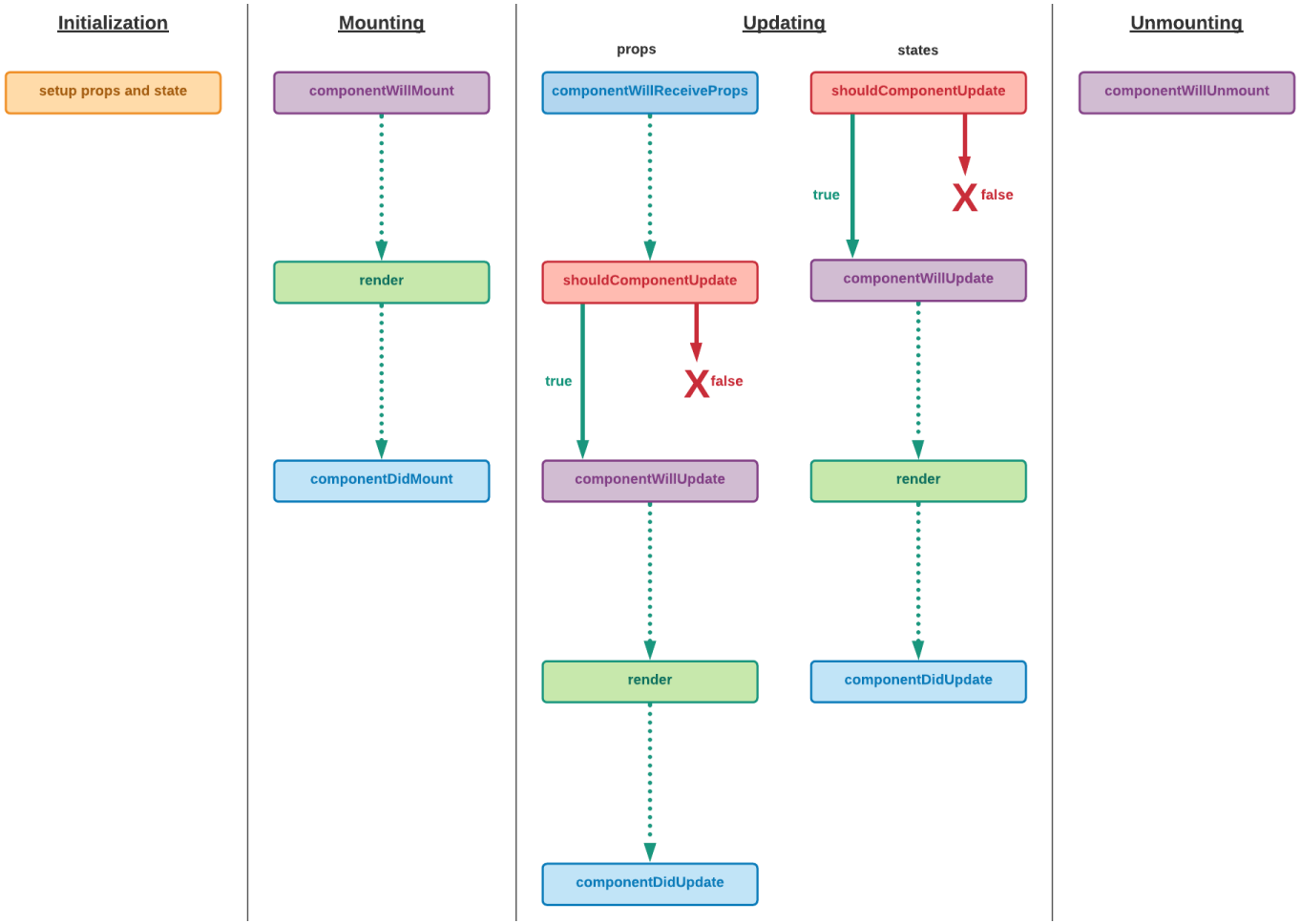
Figure 1.2 Folder Structure of a Basic React App



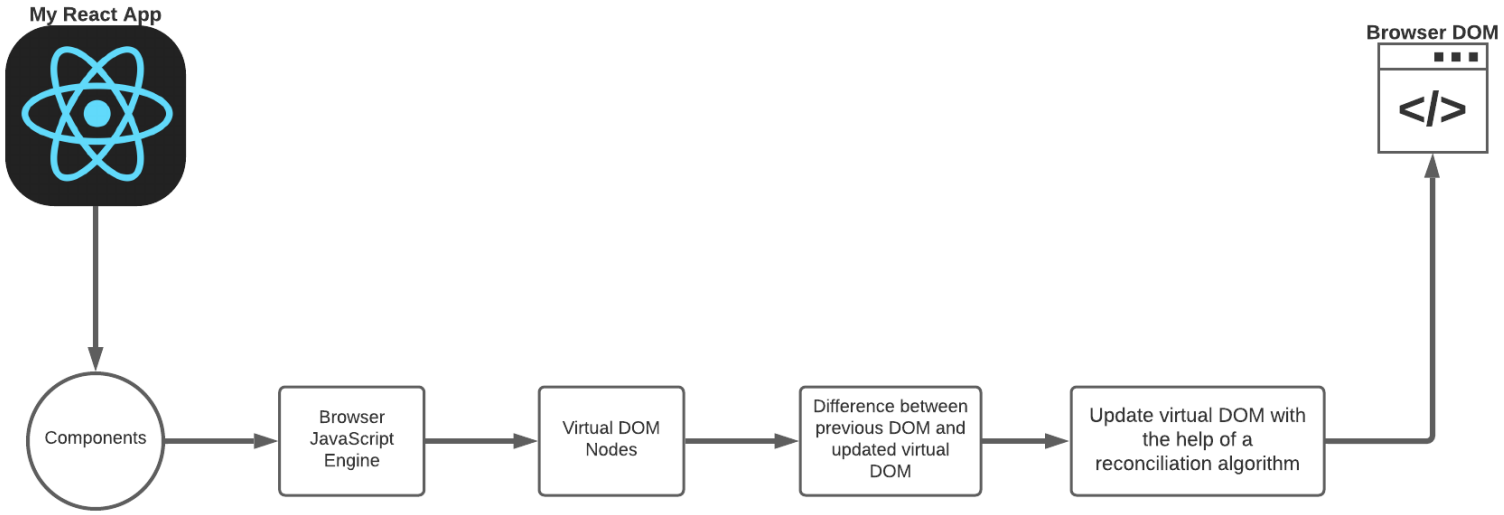
Appendix C: HTML Document Chart

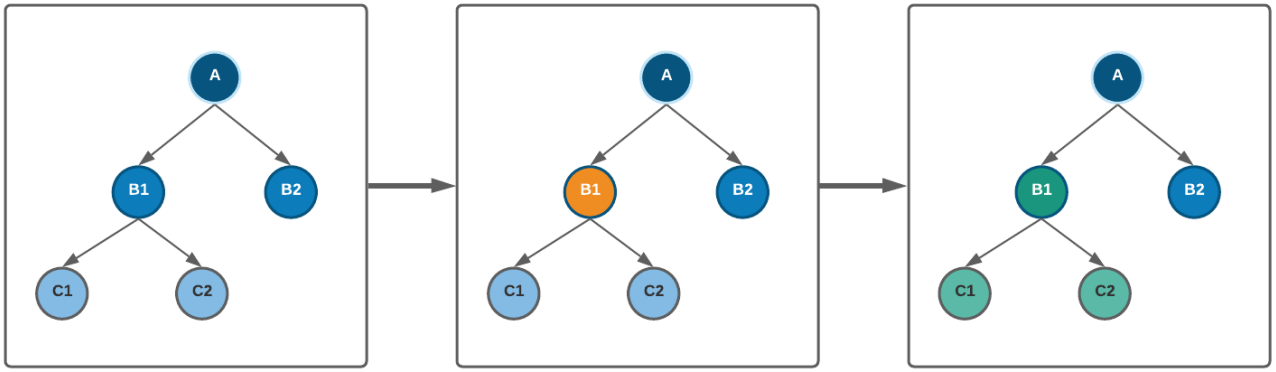


Appendix D: React.js Component Lifecycle

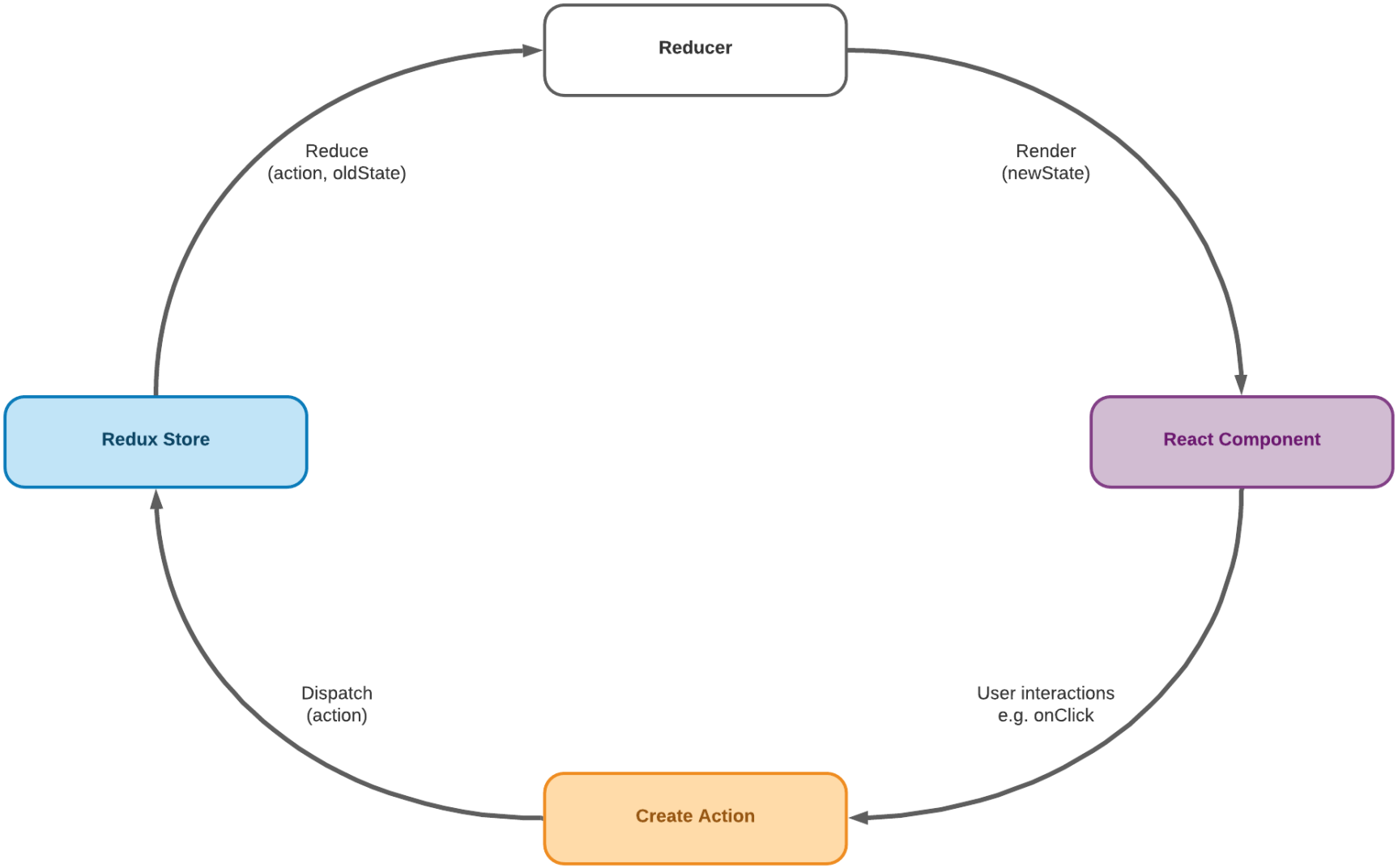


Appendix E: React Virtual DOM





Appendix F: React Redux Architecture



Appendix G: Basic Commands

Command	Result
`npm install -g npm`	Installing the Node Package Manager. It is also an optional install when you download and install Node.JS.
`node -v` `npm -v`	Used to check if Node and NPM are installed. It displays the versions of each.

`npx create-react-app my-app`
`cd my-app`
`npm start`
	Command used to create a new React app “my-app”.
	Change the directory to the “my-app” directory.
	Runs the application in development mode, and to access it we open a browser tab with the target http://localhost:3000

`npm install`	Installs the node modules listed in the `package.json` file.
`npm test`	Runs tests in an interactive watch setting.
`npm run build`	Builds a production version of the app to the “build” folder.

Appendix E: Basic Terms and Meanings

Concept	Meaning
Back-end programming	Programming related with server-side technology in a client-server architecture
Call	The process of “activating” a method with the arguments passed as parameters.
Client-side	This is the user-facing side and tends to also be associated with the user's local computer.
Component	React types are used to handle data and logic as well as being displayed or rendered to the view.
Constructor	Render is the keyword for the constructor and is used when initializing the program.
Declarative Programming	A programming paradigm in which code is written to meet the desired result without specifying logic to do with the control flow.
DOM	The document object model, a tree structure with elements to be rendered to the view. Usually, it is associated with XML or HTML code and for React web applications this is achieved when the application is compiled.
Element	React elements are the smallest building blocks of React apps (reactjs.org, 2021)
Front-end programming	This is development related to the client-facing functionality of a website.
Function	“A function is a piece of code identified by name, is given a local environment of its own and can exchange information with the rest of the code using parameters”, (Gabbrielli & Martini, 2010). A function is a subprogram or procedure that performs the desired operation on its given parameters and returns a value. Some functions return no value and are meant to perform logic on given parameters.
Functional Programming	“Functional programming is a paradigm where functions are in their pure form mathematically”, (Gabbrielli & Martini, 2010)
Imperative Programming	“Imperative programming is a style of programming where the programmer instructs the computer to execute a sequence of statements (high-level instructions) that can

	change the state of the program (variables stored in memory)", (Kelly, 2021).
Method	A way of doing things. An action within the code.
MVC	Model-View-Controller. An architecture and software pattern is used for designing user interfaces and connecting the logic between basic parts of the code.
Parameters	Values given to a function that are used in the logic of the function.
Props	Stands for properties in React and are used to transfer data between different components of React. Props are unidirectional from a parent component to a child component and are read-only, meaning they cannot be modified.
Return	The value returned from a function after the function has been used along with its parameters. Return values are how a function can interact with the rest of the program.
Server-side	This is the server-facing side and tends to also be associated with the network and server-side logic.
State	The basic states of a physical machine are fetch, decode, execute and stop and represent the basic model which stores the state as a value at any given point. A program can also store memory. The basics of any program state is that it has a finite pair of variables *(X, n)* that transition as the program executes. This can be better understood as the variable *X* has the value of *n*.