Bachelor’s thesis

Information and Communications Technology
May 2020

Phan Hong Duc

REACT FRAMEWORK

– Concept and implementation

TURKU AMK
TURKU UNIVERSITY OF
APPLIED SCIENCES

BACHELOR'S THESIS | ABSTRACT

TURKU UNIVERSITY OF APPLIED SCIENCES

Information and Communication Technology

2020 | 45

Phan Hong Duc

REACT FRAMEWORK

- Concept and implementation

This thesis was conducted as research on React JS framework by participating the practical project with named Tama web application and implementation the small project with named shopping cart. The purpose of this thesis was to introduce and implement a small project in React JS.

This thesis discussion React JS concepts and their related specifications, such as JSX, state, props, events, class component and Redux.

The implementation of Web application projects using React framework was demonstrated in a Shopping cart and Tama web applications. Those projects also use a third party with named Redux to build a complete web application. The result of this thesis was the creation of a shopping cart and Tama application with Redux which were successfully deployed to the server.

KEYWORDS:

React, React Js, Redux, State, Props, Life cycle, Actions, Reducers, Stores, App, web application.

CONTENT

LIST OF ABBREVIATIONS (OR) SYMBOLS 6

1 INTRODUCTION 1

2 MAIN CONCEPTS 2

2.1 JSX and render method 2

2.2 State and Props and life cycle methods 3

3 ADVANCED CONCEPTS IN REACTJS 7

3.1 State and Props and life cycle methods 7

3.2 Context 8

3.3 Forwarding refs 9

3.4 Higher order component 9

3.5 Render props 12

4 COMPONENTS 13

4.1 Function component 13

4.2 Class Component 14

4.3 Styled component 15

4.4 Web component 16

5 REDUX 18

5.1 Managing state by Redux 18

5.2 State management tool 20

5.3 Store 21

5.4 Actions 22

5.5 Dispatch to props 23

5.6 Middleware 24

5.7 Reducer 26

6 IMPLEMENTATION 28

6.1 System requirements 28

6.2 Structure and run 28

6.3 Deployment 43

7 CONCLUSION 47

REFERENCES 48

FIGURES

Figure 1. JSX in JS file. 2
Figure 2. Render method in React JS project. 3
Figure 3. The props is called in component 4
Figure 4. The props is exchanged from parent component to child component 5
Figure 5. State is in constructor of AlbumPage component. 5
Figure 6. All life cycle methods. [3] 6
Figure 7. Non-splitting code vs splitting code. [4] 7
Figure 8. An example of the context in React. 8
Figure 9. An example of Refs. 9
Figure 10. An example of Props proxy. 10
Figure 11. An example of Inheritance Inversion. 10
Figure 12.An example HOC. 11
Figure 13. An example of render props. 12
Figure 14. An example function component in one React JS project. 14
Figure 15. An example of React class component. 15
Figure 16. An example of style component. 16
Figure 17. An example using web component in React JS. 17
Figure 18. Node tree to simulate the states in React project. 19
Figure 19. Update the state nodes without using Redux. [11] 19
Figure 20. Update state nodes using Redux. [11] 20
Figure 21. An example does not use Redux for managing global states in React. 21
Figure 22. An example creating store in React. 22
Figure 23. An example of the actions. 23
Figure 24. An example installing the mapDispatchToProps method. 24
Figure 25. Example of using the dispatchToProps method in React. 24
Figure 26. An example using Redux saga with generator functions. 26
Figure 27. Flow Redux in React Project. [11] 27
Figure 29. Shopping-cart App project structure. 29
Figure 30. Shopping Cart user interface of login and register. 30
Figure 31. Shopping Cart user interface in home-page. 31
Figure 32. User interface of the message notification. 31
Figure 33. User interface in cart. 32
Figure 34. User interface of the functionality of calculating the total of price. 32
Figure 36 Tama App project structure. 33
Figure 37 Interface login 34
Figure 38 User Interface 35
Figure 39 Create restaurant with some input fields 35
Figure 40 Main restaurant with some features, such as import and export 36
Figure 41 Main company interface with some features such as import and export 36
Figure 42 Create new company interface 37
Figure 43 Employees with some feature 37
Figure 44 Create new employee with some fields 38

Figure 45 Change password and logout 38
Figure 46 Change Language 38
Figure 48 Create new user 39
Figure 49 Commits around the september 2019 40
Figure 50 Commit around 9.2019 40
Figure 51 Commit in 9.2019 41
Figure 52 Commit in 8.2019 41
Figure 53 Commit in 8.2019 42
Figure 54 Commits from the first time-7.2019(the starting time of the Tama React Project) 42
Figure 55. Localhost on Chrome and console for debugging. 43
Figure 56. Run localhost with terminal of Visual code. 44
Figure 57. Github Pages for deploying in setting tab. 45
Figure 58. Deployment project. 46

LIST OF ABBREVIATIONS (OR) SYMBOLS

API	Application Program Interface
App	Application
DOM	Document Object Model
HTML	Hypertext Markup Language
JS	JavaScript
JSX	JavaScript XML
NPM	Node Package Management
Props	Properties

1 INTRODUCTION

One of the popular frameworks in Javascript, ReactJS can be built on many web platforms on Windows 10, macOS, and mobile devices. In addition, ReactJS helps to improve the performance of modern web states by compile source code to static HTML, as well as a single page application.

Today, in the modern web application, the requirement of the user is very high, such as how many minutes the App is loaded or the layout has to beautiful and easy to use. Therefore engineers around the world have to work very hard to look for the solution to improve the performance. And the single page application concept appears to help improving the performance of the App, such as the App has only one page, instead of multi pages, leading to help the App to load faster. Realizing that it is an inevitable development of the application, Facebook engineers developed a new framework named React, using the advantages of SPA.

ReactJS meets the requirements of the modern application, from improving the performance of traditional web application as well, to meet a new approach of the modern web application that separate frontend side with backend side, leading to improve the performance of developers developing the application as well. Based on this practical demand, the project has been carried out.

This thesis was done mainly by researching and using React JS framework and React environment to develop ReactJS App, which is the one of the best framework in Frontend side around the world.

The application focuses on the use of ReactJS and Redux to build the Shopping cart single page application which is a requirement of a modern web application.

2 MAIN CONCEPTS

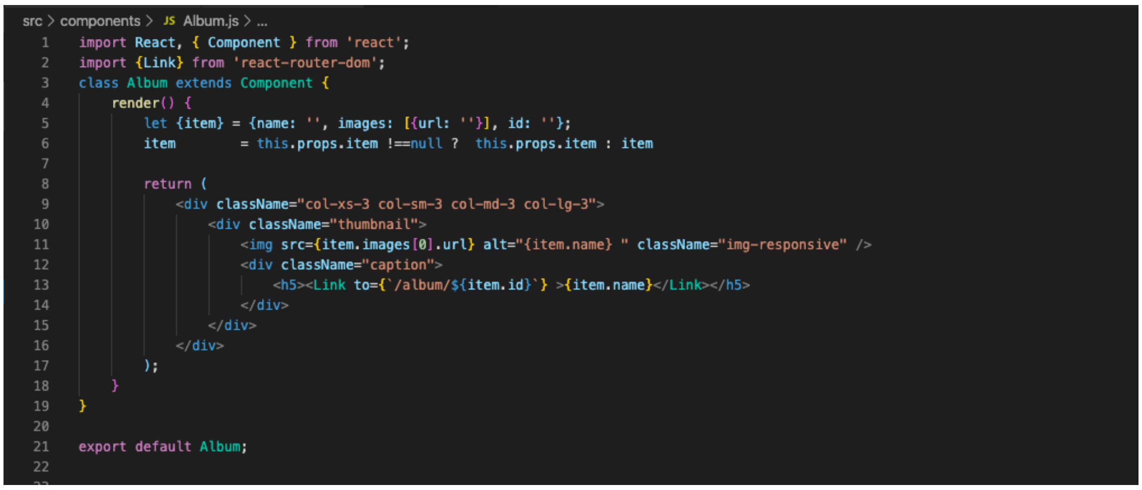
This chapter explains technologies related to main concepts of React framework, such as JSX, state, props, and life-cycle methods. These are also understood as client side rendering, since they affect a user's interaction directly as client-side.

2.1 JSX and render method

JSX is a syntax that intended to be used by preprocessors to combine HTML code into JavaScript standard object instead of using string concatenation to render UI components.
"Most people find it helpful as a visual aid when working with UI inside the JavaScript code. It also allows React to show more useful error and warning messages." [1]. The benefits from JSX includes:

- JSX makes easier and faster to write templates for user who are familiar with HTML.
- The performance is faster while compiling code to JavaScript.

Figure 1 illustrates the JSX looks like when using in React framework. The code below is an Album component is practical Tama App which has the JSX code inside render method.



src > components > JS Album.js > ...

```
import React, { Component } from 'react';
import {Link} from 'react-router-dom';
class Album extends Component {
    render() {
        let {item} = {name: '', images: [{url: ''}], id: ''};
        item        = this.props.item !==null ?  this.props.item : item

        return (
            <div className="col-xs-3 col-sm-3 col-md-3 col-lg-3">
                <div className="thumbnail">
                    <img src={item.images[0].url} alt="{item.name} " className="img-responsive" />
                    <div className="caption">
                        <h5><Link to={`/album/${item.id}`} >{item.name}</Link></h5>
                    </div>
                </div>
            </div>
        );
    }
}

export default Album;

```

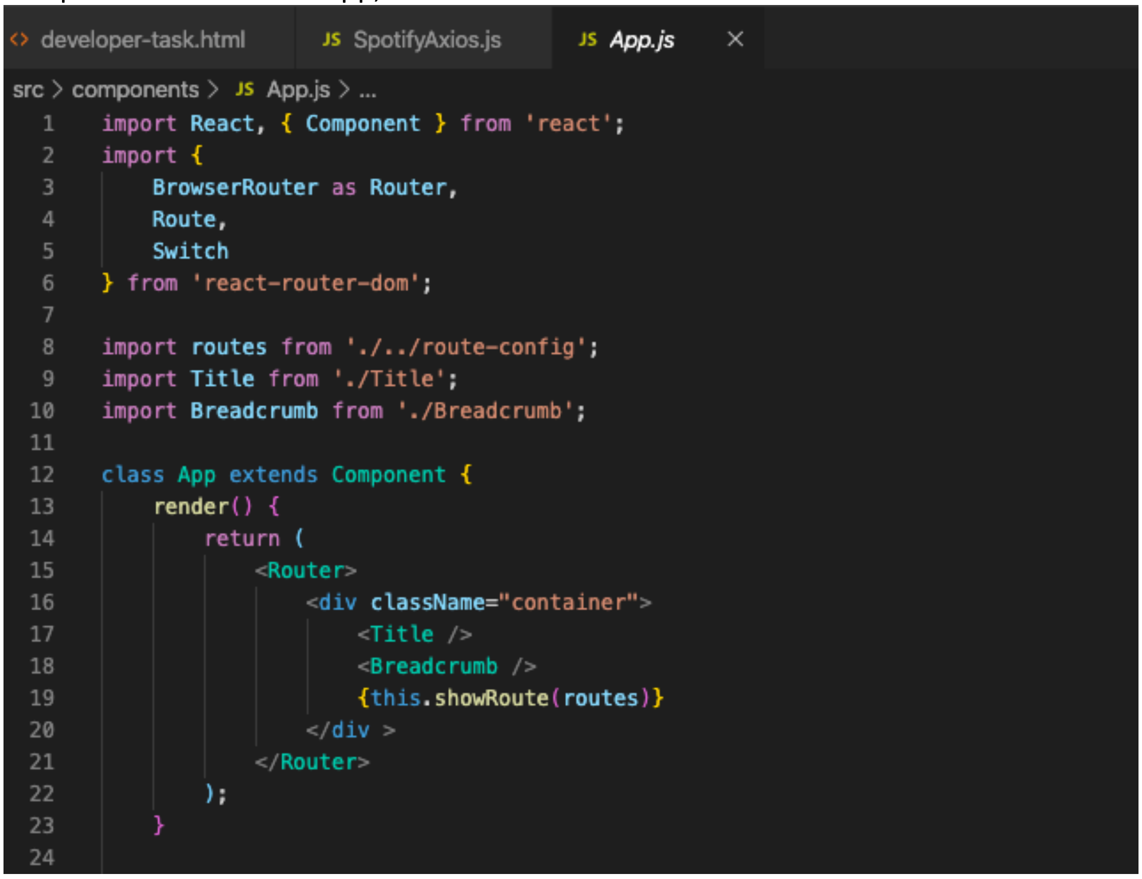
Figure 1. JSX in JS file.

There are many methods and functions as external API of React framework, such as render, constructor. The constructor is a method that is automatically invoked during the creation of an object from a class and in React framework, it can be used to bind event handlers to the component and initialize the local state of the component. The render method is provided by ReactDOM package and can be used to render to the DOM. "ReactDOM.render() controls the contents of the container node you pass in. Any

existing DOM elements inside are replaced when first called. Later calls use React's DOM diffing algorithm for efficient updates. ReactDOM.render() does not modify the container node (only modifies the children of the container). It may be possible to insert a component to an existing DOM node without overwriting the existing children." [2]
In React JS, render method is invoked after constructor executed. The method has maximum three parameters:

- Element: the expected React element or JSX expression need to be rendered.
- Container: element contains the rendered element.
- Callback: the function will be invoked after the render is complete. This is an optional parameter.

Figure 2 illustrates the render method in class component App of React JS. There is App component in the Tama App, which has the render method.



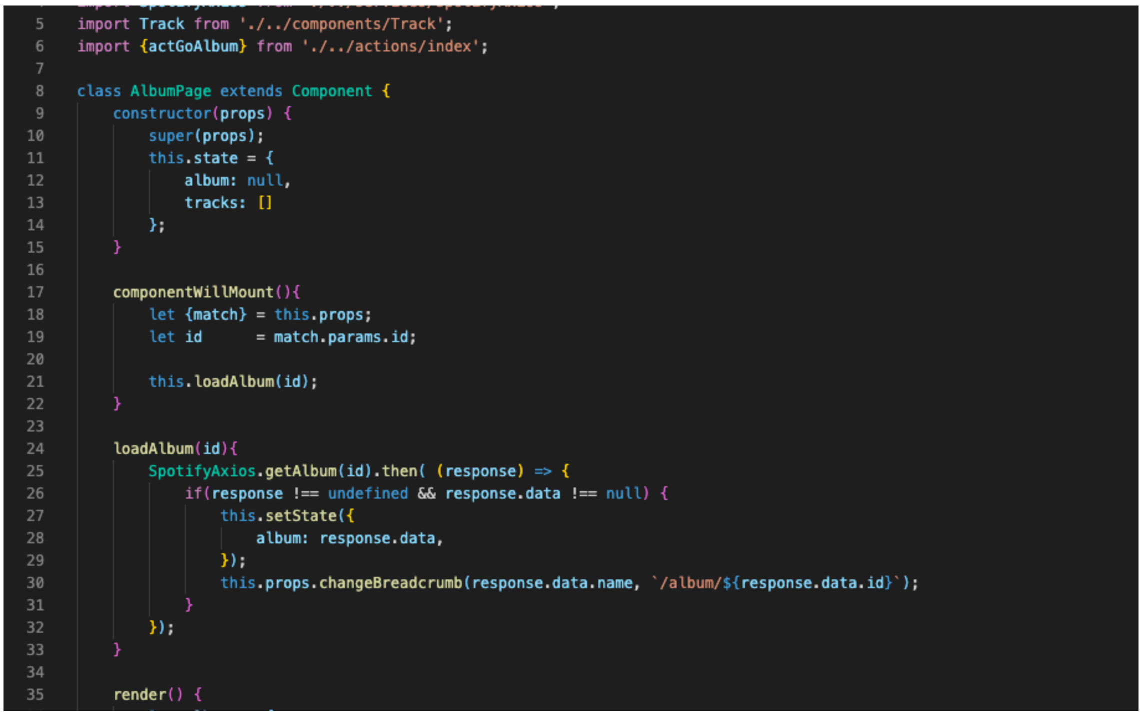
```
import React, { Component } from 'react';
import {
    BrowserRouter as Router,
    Route,
    Switch
} from 'react-router-dom';

import routes from './../route-config';
import Title from './Title';
import Breadcrumb from './Breadcrumb';

class App extends Component {
    render() {
        return (
            <Router>
                <div className="container">
                    <Title />
                    <Breadcrumb />
                    {this.showRoute(routes)}
                </div >
            </Router>
        );
    }

```

Figure 2. Render method in React JS project.

2.2 State and Props and life cycle methods

When developing software applications, developers try to manage the flow of the application as much as possible, especially performance. The application is a sequence

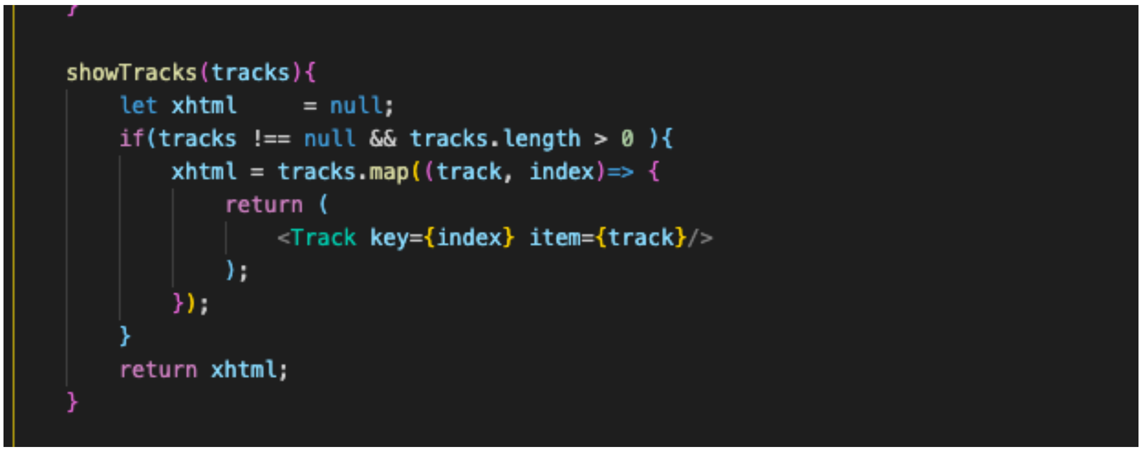
of instructions in a program, this is determined at run time by the input data and by the control structures that developers used in the program. In React environment, there are some React native APIs to help developers improve the applications' performance such as state, props and life cycle methods. State is what allows to create components that are dynamic and interactive. When React sees an element representing a user-defined component and passes JSX attributes, children to this component as a single object, in React this object is called props. In application with many components, each component has a lifecycle which developer can monitor and manipulate during its three main phases: mouting, updating and unmounting.
After the initial render of the application, the condition for the next render is the props or/and state changes. The props were identified by the caller of this component will be included in "this.props". In specific, the props were defined by the child tags within the JSX expression, instead of within the tag itself are called this.props.children.
In figure 3 illustrates the AlbumPage component in Tama App which gets the props.



```
import Track from './../components/Track';
import {actGoAlbum} from './../actions/index';

class AlbumPage extends Component {
    constructor(props) {
        super(props);
        this.state = {
            album: null,
            tracks: []
        };
    }

    componentWillMount(){
        let {match} = this.props;
        let id      = match.params.id;

        this.loadAlbum(id);
    }

    loadAlbum(id){
        SpotifyAxios.getAlbum(id).then( (response) => {
            if(response !== undefined && response.data !== null) {
                this.setState({
                    album: response.data,
                });
                this.props.changeBreadcrumb(response.data.name, `/album/${response.data.id}`);
            }
        });
    }

    render() {
```

Figure 3. The props is called in component

Figure 4 illustrates how to transfer the date through the props between parent and child components.

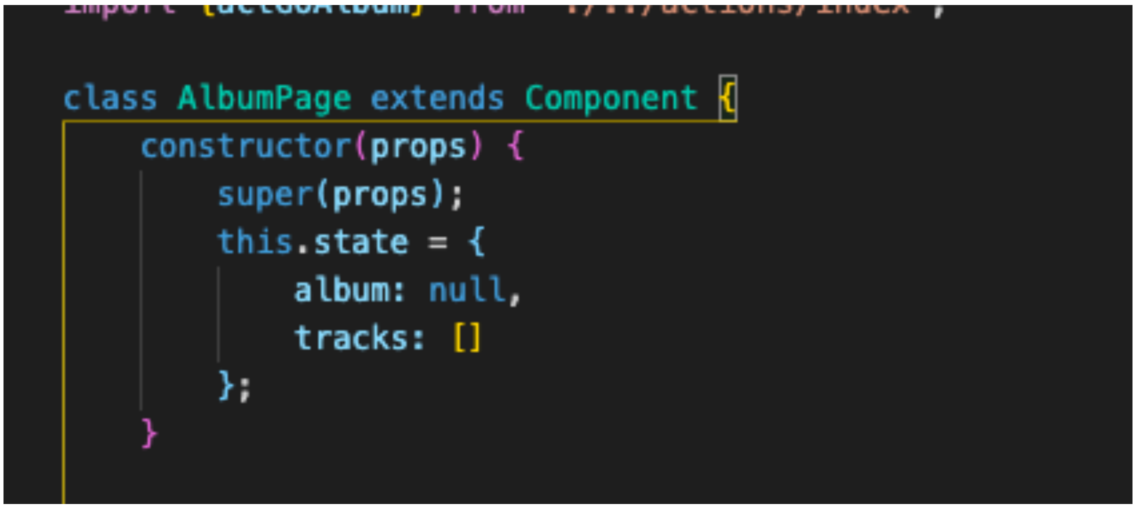


```
showTracks(tracks){
    let xhtml      = null;
    if(tracks !== null && tracks.length > 0 ){
        xhtml = tracks.map((track, index)=> {
            return (
                <Track key={index} item={track}/>
            );
        });
    }
    return xhtml;
}
```

Figure 4. The props is exchanged from parent component to child component

Programmer can use the state to manage changing UI by forcing render method invoked. In React JS, state likes the internal date, and this date is only existed in the component contains the state. When user defines state that should be a simple Javascript object and state’s date specific may change during the time, and values which are not used to render or data flow should be defined, and put into fields as a component instance, instead of the state. Specially, this.state should not be changed directly, because when setState is called then, it may replace the mutation that made before, making this.State is a immutable value.

Figure 5 illustrates the state is initialized in the AlbumPage component in React.



```
class AlbumPage extends Component {
    constructor(props) {
        super(props);
        this.state = {
            album: null,
            tracks: []
        };
    }
```

Figure 5. State is in constructor of AlbumPage component.

Moreover, life cycle methods plays important roles in managing the flow of the React application. There are many methods such as constructor, render, getDerivedStateFromProps, shouldComponentUpdate.

Figure 6 shows all life cycle methods adjusts the flow of React application.

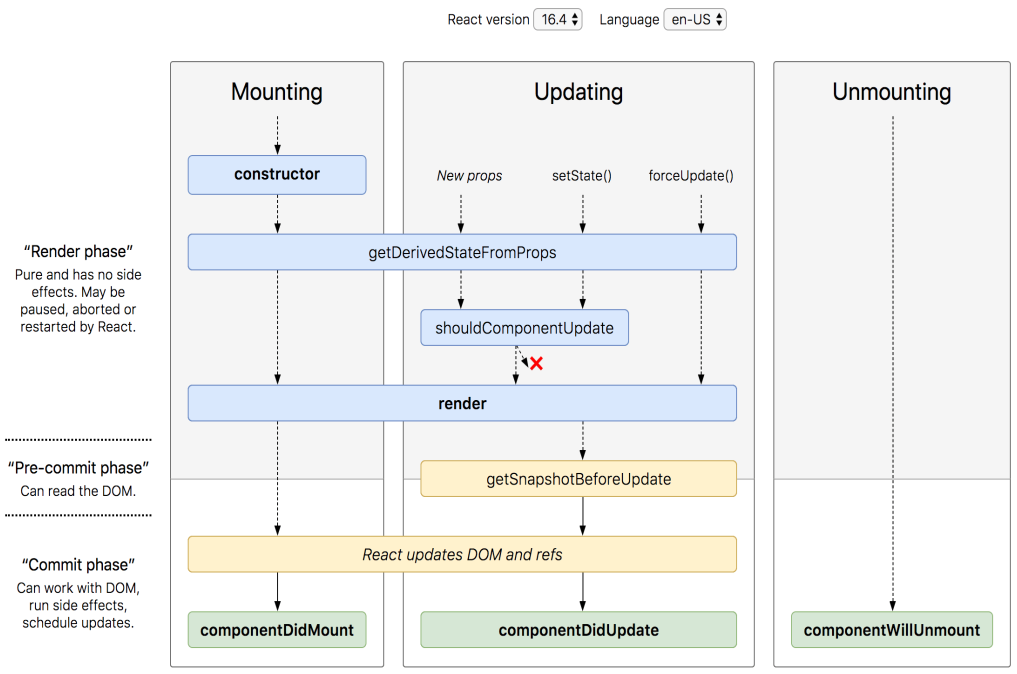


Figure 6. All life cycle methods. [3]

3 ADVANCED CONCEPTS IN REACTJS

The React core team, along with the large contributors community, has done a great job of upgrading every day the technology by releasing many small versions and fixing bugs, including many advanced concepts. The main goal of the advanced concepts in ReactJS is to solve outstanding issues, make the development easier for future product development work, or to simply clean up the spaghettis code. Therefore, this chapter describes the advanced concepts of React framework, such as code splitting to improve the performance of the application or some advanced patterns such as Higher other components or render props.

3.1 State and Props and life cycle methods

When the applications have been grown up day by day, leading to the bundle which uses tools as Webpack, Rollup or Browserify of the application which was developer built will grow too; it makes the app wasting more time to load, so the Code-splitting is a feature to help to split the app to smaller and generate multiple bunbles which can be dynamically loaded during the running time.
Code-Splitting can help developers load only things which are currently needed by the users and make the app's performance will be increased significantly thanks to reducing the amount of code in the React app and do not load code which the users never use it. In the figure 7 below, it depicts the disadvantage of non-splitting code. It will import all of dependencies and the performance of the project is very slow because the project needs to load many unnecessary libraries and dependencies.

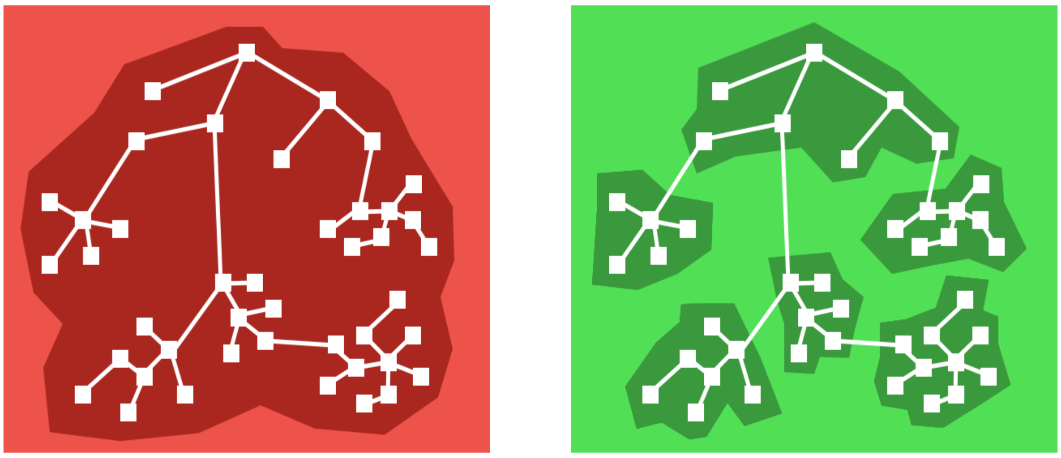


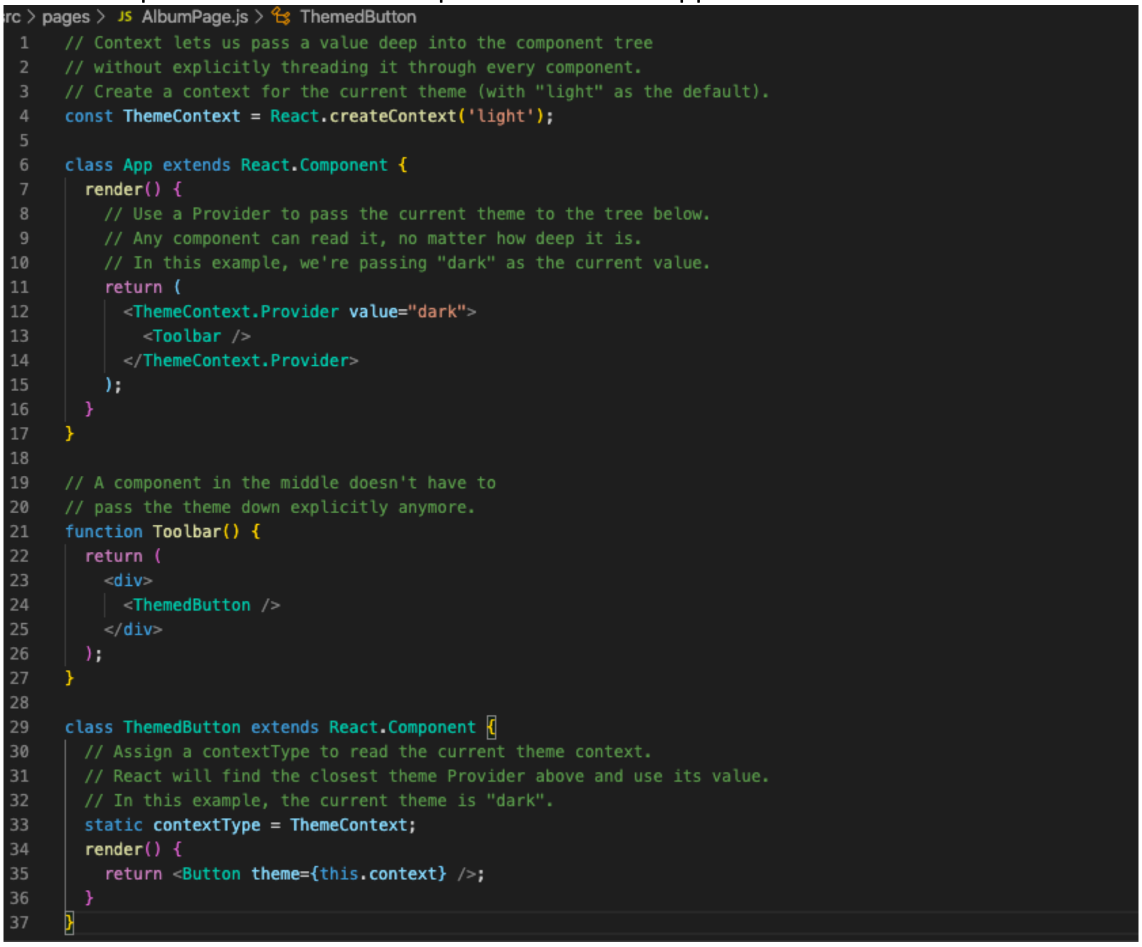
Figure 7. Non-splitting code vs splitting code. [4]

Otherwise, the project just needs to load the necessary libraries and dependencies. Therefore, in this case, the project does not need loading many unnecessary libraries and dependencies, and the performance is improved much.

3.2 Context

Context in React Js has the scope that contains all data of parent component and the children components can get and use. That means the parent component passing data to component tree without having to pass props down child level will be provided by context.
When looking at a typical React application, data will be passed from parent to child throughout props, but sometimes having a problem for prop's specific types as language options language, UI themes, so context is generated to share these values within the components without having to pass props via tree's all level.

Figure 8 illustrates how the data goes down from parent component. After the data going down by using context, all children compoents can use the data by this.context keyword. However, "If you only want to avoid passing some props through many levels, component composition is often a simpler solution than context." [4] In practical development an application, developer prefer using composition in case going down the data from parent to children components in React application.

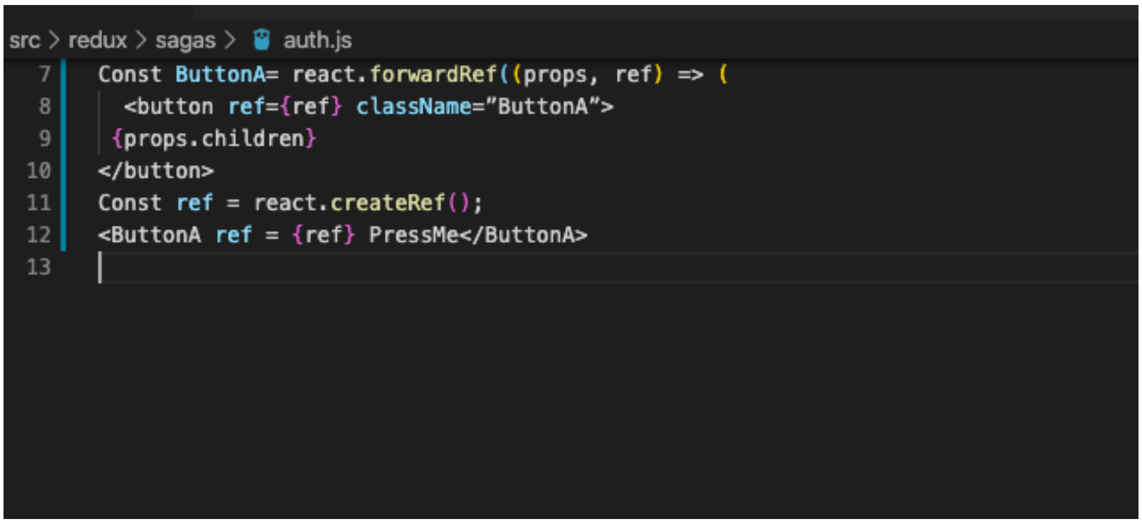


```
// Context lets us pass a value deep into the component tree
// without explicitly threading it through every component.
// Create a context for the current theme (with "light" as the default).
const ThemeContext = React.createContext('light');

class App extends React.Component {
  render() {
    // Use a Provider to pass the current theme to the tree below.
    // Any component can read it, no matter how deep it is.
    // In this example, we're passing "dark" as the current value.
    return (
      <ThemeContext.Provider value="dark">
        <Toolbar />
      </ThemeContext.Provider>
    );
  }
}

// A component in the middle doesn't have to
// pass the theme down explicitly anymore.
function Toolbar() {
  return (
    <div>
      <ThemedButton />
    </div>
  );
}

class ThemedButton extends React.Component {
  // Assign a contextType to read the current theme context.
  // React will find the closest theme Provider above and use its value.
  // In this example, the current theme is "dark".
  static contextType = ThemeContext;
  render() {
    return <Button theme={this.context} />;
  }
}
```

Figure 8. An example of the context in React.

3.3 Forwarding refs

According to React documentation [5], refs are used to get references to a DOM node or can be an instance of a component in a React application. ie refs will return to a node that we refer to. Ref forwarding is an automatic technical to pass a ref via a component to their children to make the component libraries can be re-used.
The common cases of ref forwarding: forwarding refs to DOM components, component library maintainers, higher-order components, show a custom name in DevTools.
In figure 9, there is a ref generating a React ref as React.createRef and assign it to a ref valuable, then passing this ref to <ButtonA ref={ref}> by determining that data as a JSX attribute. After that React will pass the ref to prop as (props,ref) => the function inside fordwardRef API will be a second argument. Finally, continuing pass the ref argument downs to the button as <button ref={ref}> by determining it as a JSX attribute.



```
src > redux > sagas > auth.js
Const ButtonA= react.forwardRef((props, ref) => (
  <button ref={ref} className="ButtonA">
 {props.children}
</button>
Const ref = react.createRef();
<ButtonA ref = {ref} PressMe</ButtonA>
```

Figure 9. An example of Refs.

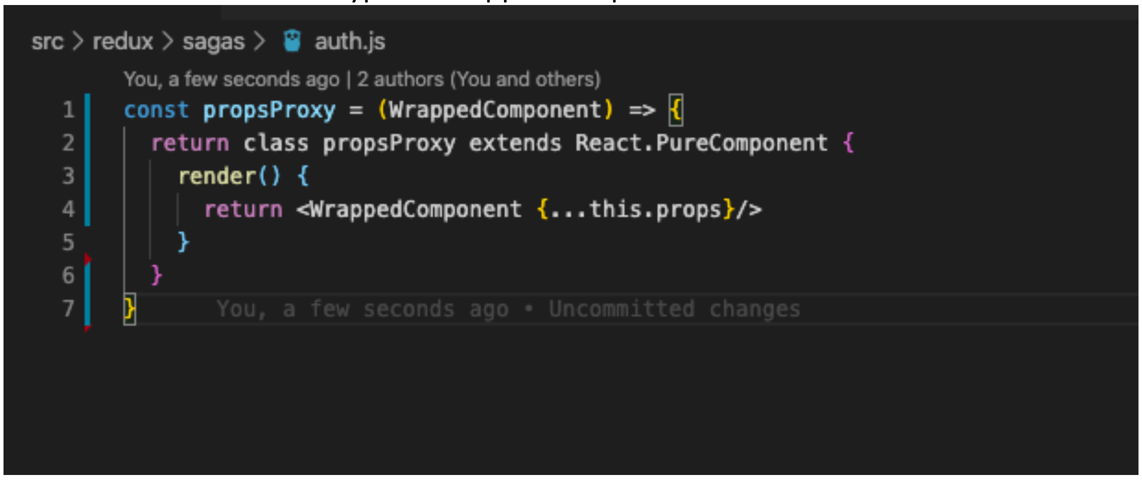
3.4 Higher order component

During developing one complex application, thought about the coding direction in React to help the application is optimized, so how reuses component or code in other places is very important. And, Higher order component (HOC) is created to reuse components and optimize the application.
Besides, HOC does not modify or modify the input component, that pattern only inherits the behavior of that component. The component is packaged in to receive all the props of the container, with props that can be used to render the content that the project needs. HOC is not related to how the data is used, from where, and the component that is packaged is also not related to where the data is transmitted. There are some benefits of HOC:

- Code reuse, logic and automated abstraction (bootstrap abstraction).
- Render Highjacking (Render Highjacking).

- Abstract (abstraction) and control (manipulation) State.
- Control Props.

In figure 10, there is Props Proxy method to show the HOC's render method will return a React Element in which type is WrappedComponent.



src > redux > sagas > auth.js

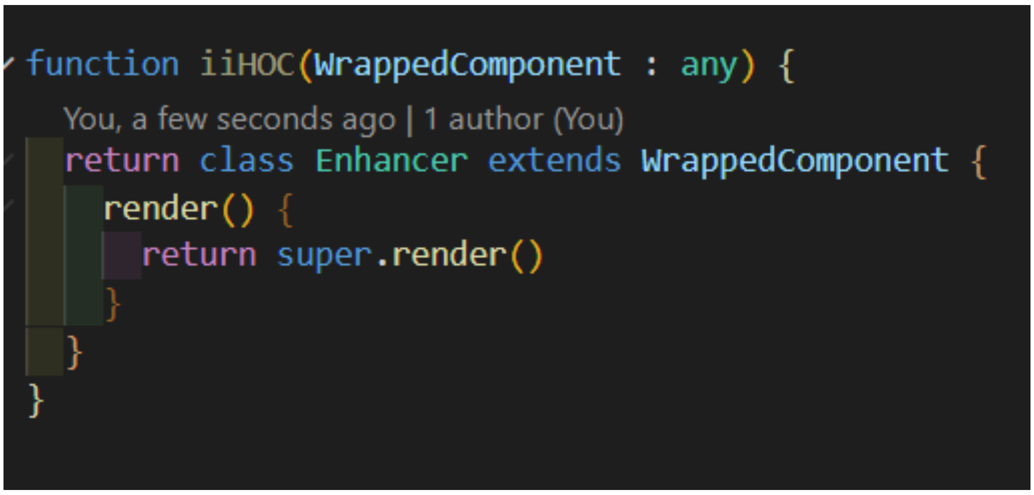
You, a few seconds ago | 2 authors (You and others)

```
const propsProxy = (WrappedComponent) => {
  return class propsProxy extends React.PureComponent {
    render() {
      return <WrappedComponent {...this.props}/>
    }
  }
}      You, a few seconds ago • Uncommitted changes
```

Figure 10. An example of Props proxy.

In figure 12, using the method Inheritance Inversion to show the HOC returns class (Enhancer) inheritance WrappedComponent; Enhancer will extend WrappedComponent rather than WrappedComponent extending certain Enhancer clas. Leading to, the relationship between them will be reversed.

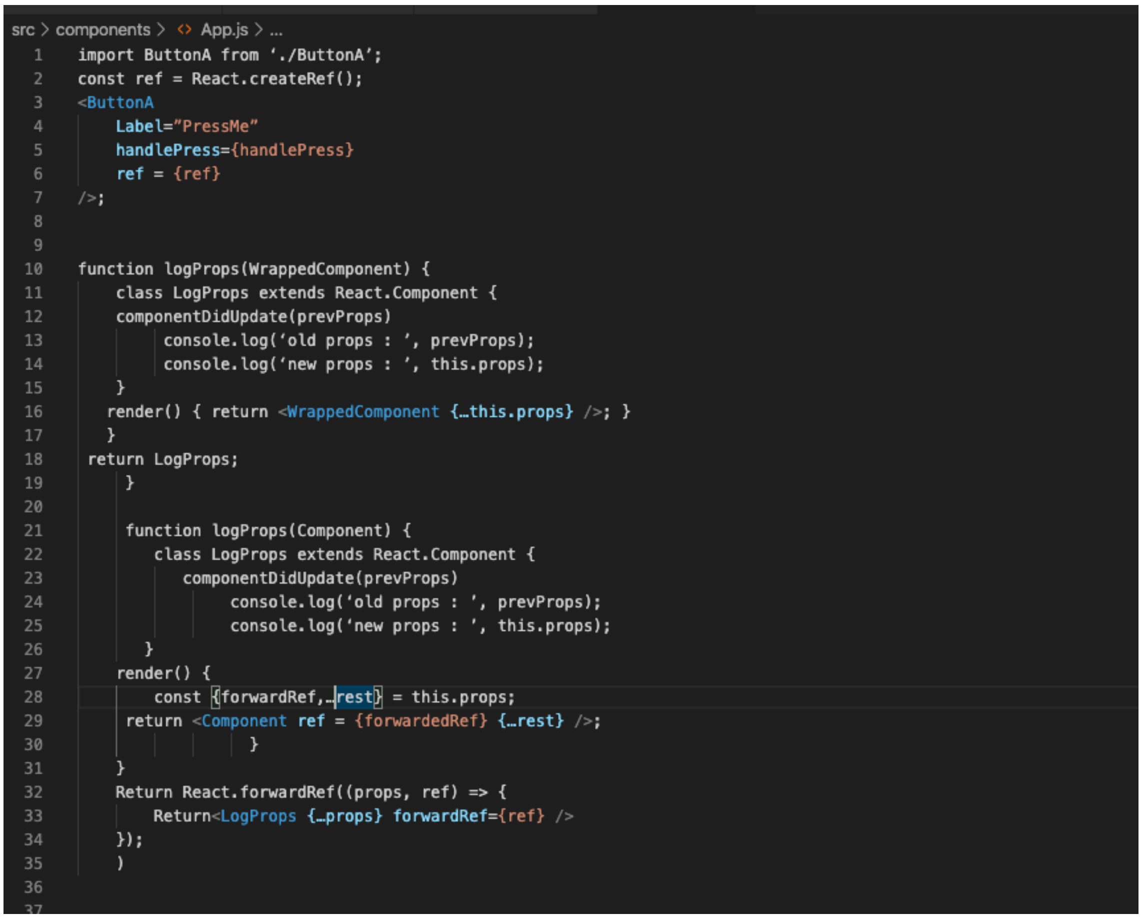
Inheritance Inversion allows the HOC to access the WrappedComponent instance through this, which means that the HoC has access to the state, props, component lifecycle hooks and even the render method.



```
function iiHOC(WrappedComponent : any) {
  You, a few seconds ago | 1 author (You)
  return class Enhancer extends WrappedComponent {
    render() {
      return super.render()
    }
  }
}
```

Figure 11. An example of Inheritance Inversion.

In figure 11 and figure 12 illustrate how the HOC looks like and used in the Taama App to optimize the App by reusing components . The logProps will pass all props to all their components which are wrapped, so that the output render will be the same. Then using logProps to log all props which are passed to component as ButtonA, then passing refs to the inside ButtonA component by react.forwardRef API. Besides, the React.forwardRef gets props and ref parameters by render function and returns it to React.



src > components > App.js > ...

```
import ButtonA from './ButtonA';
const ref = React.createRef();
<ButtonA
    Label="PressMe"
    handlePress={handlePress}
    ref = {ref}
/>;


function logProps(WrappedComponent) {
    class LogProps extends React.Component {
    componentDidUpdate(prevProps)
        console.log('old props : ', prevProps);
        console.log('new props : ', this.props);
    }
   render() { return <WrappedComponent {…this.props} />; }
   }
 return LogProps;
     }

     function logProps(Component) {
        class LogProps extends React.Component {
           componentDidUpdate(prevProps)
                console.log('old props : ', prevProps);
                console.log('new props : ', this.props);
        }
    render() {
        const {forwardRef,…rest} = this.props;
     return <Component ref = {forwardedRef} {…rest} />;
                 }
    }
    Return React.forwardRef((props, ref) => {
        Return<LogProps {…props} forwardRef={ref} />
    });
    )
```

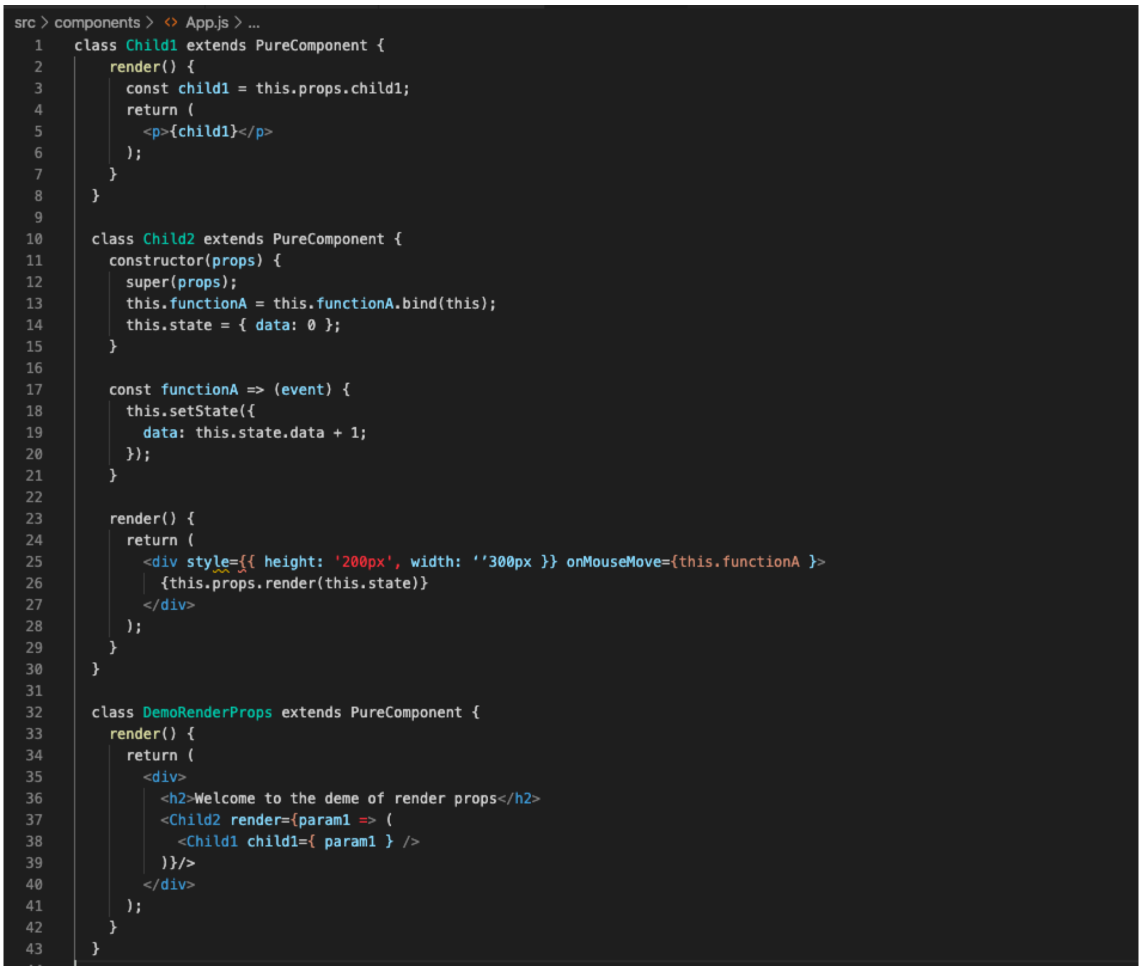
Figure 12.An example HOC.

One of the reasons HOCs are popular is because this technique uses the ES6 class (class) from the beginning. Since React 16, the class is the default mechanism when building components, completely replacing React.createClass () in previous versions. This makes sense, since the class is already supported by most current browsers by default. However, the use of HOCs also has limitations:

- HOCs are confusing: Using multiple HOCs for a component easily leads to the situation of not knowing which HOC provided by this HOC.
- Duplicate name props: If there are two HOCs using the same name for the prop, the codes will be overwritten.

3.5 Render props

The render props are not a React API, but rather a technique for handling logical sharing between React Components using prop with value as a function. In simple terms, render props are a method with a similar purpose to the Higher Order Component method, which helps to reuse logic across multiple components. Render Props are used to create a very famous library in the React ecosystem, which is the React-Router. In essence, the purpose is the ability to reuse state, methods of a component on another component need to use the same state. However, the pattern uses the props for reusing the logic code from one component then passing the business logic to another component. A component with a render prop takes a function that returns a React element and calls it instead of implementing its own render logic.
In figure 13, there is an example of render props in React which is used in the Taama App. In the DemoRenderProps component, there is Child1 component and that component can manage the data of DemoRenderprops components but this component still uses the same layout and when reusing the component in other places.



src > components > App.js > ...

```
class Child1 extends PureComponent {
    render() {
      const child1 = this.props.child1;
      return (
        <p>{child1}</p>
      );
    }
  }

  class Child2 extends PureComponent {
    constructor(props) {
      super(props);
      this.functionA = this.functionA.bind(this);
      this.state = { data: 0 };
    }

    const functionA => (event) {
      this.setState({
        data: this.state.data + 1;
      });
    }

    render() {
      return (
        <div style={{ height: '200px', width: ''300px }} onMouseMove={this.functionA }>
          {this.props.render(this.state)}
        </div>
      );
    }
  }

  class DemoRenderProps extends PureComponent {
    render() {
      return (
        <div>
          <h2>Welcome to the deme of render props</h2>
          <Child2 render={param1 => (
            <Child1 child1={ param1 } />
          )}/>
        </div>
      );
    }
  }
```

Figure 13. An example of render props.

4 COMPONENTS

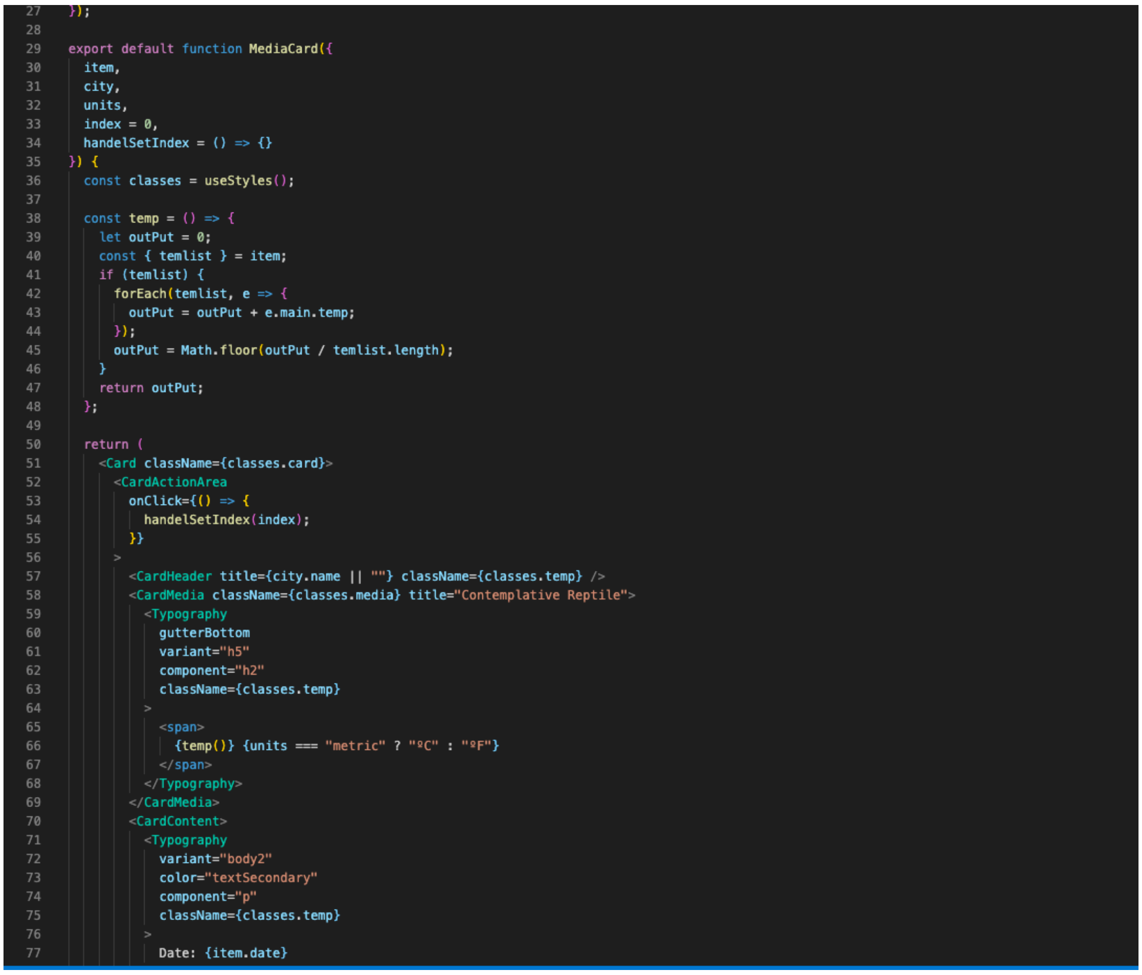
Component-based development is now a very popular way to build user interfaces (UI) and web applications. When engineers develop React project, components help saving much time because those engineers do not need to build the the small UI parts, such as button, text field or top bar of the project. The developers just use the components and combine the components to develop the layout. Modern framework as Angular, ReactJS or VueJS are also base on components to use, so get more understand about components to help using ReactJS easier.

4.1 Function component

In software development industry, there are two main ways to develop one software, that are Object oriented programming(OOP) and functional programing. However, Javascript developers, especially React developers prefer using functional programming with React Hooks, that is a function component can contain state and handle business logic [6]. A function component is a simple JavaScript function in which props are called as an argument and return a React element.
The functional component is a JavaScript function, and can not use setState() keyword here, so the function component is called stateless functional component and absolutely, the function component does not have its own state. Besides, if need to have a state in the component, there is a way to generate a class component or can initialize state in the parent component and pass it down to the child component via props.
Moreover, there are some reasons why should use functional component in ReactJS:

- The component's function makes developers easier to read and test, debug and can be reusable; because functional components do not have state and lifecycle-hook (functional component is a simple Javascript function).
- Improve performance because function component is shorter and can compile quicker than class component.
- When using function component, there is no need to think how to split the component to container component and normal UI component.

In figure 14, there is the MediaCard function component used in the Tama App without using state. That component only uses item, city, units. index and handleSetIndex props for re-render the component. This main functionality of the function component is that contains UI in the return method and gets data through props and render the data.

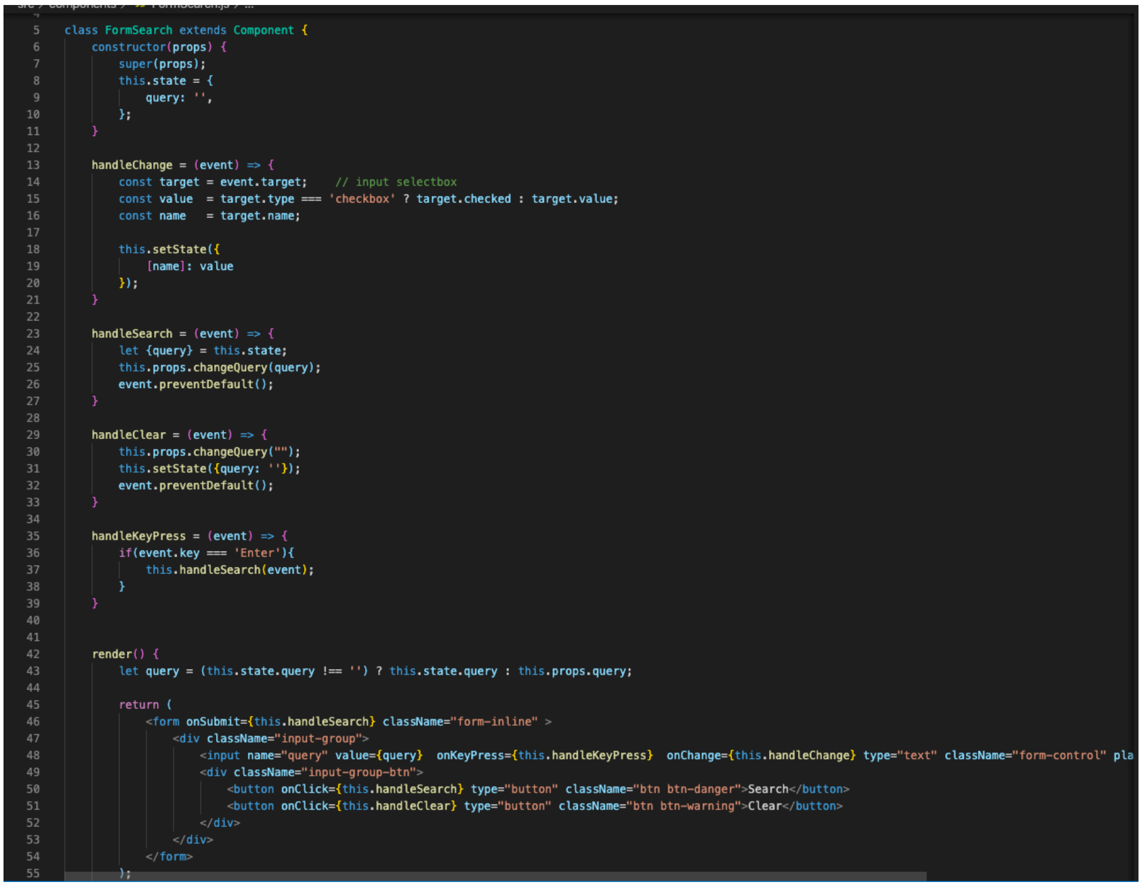


```
});

export default function MediaCard({
  item,
  city,
  units,
  index = 0,
  handelSetIndex = () => {}
}) {
  const classes = useStyles();

  const temp = () => {
    let outPut = 0;
    const { temlist } = item;
    if (temlist) {
      forEach(temlist, e => {
        outPut = outPut + e.main.temp;
      });
      outPut = Math.floor(outPut / temlist.length);
    }
    return outPut;
  };

  return (
    <Card className={classes.card}>
      <CardActionArea
        onClick={() => {
          handelSetIndex(index);
        }}
      >
        <CardHeader title={city.name || ""} className={classes.temp} />
        <CardMedia className={classes.media} title="Contemplative Reptile">
          <Typography
            gutterBottom
            variant="h5"
            component="h2"
            className={classes.temp}
          >
            <span>
              {temp()} {units === "metric" ? "ºC" : "ºF"}
            </span>
          </Typography>
        </CardMedia>
        <CardContent>
          <Typography
            variant="body2"
            color="textSecondary"
            component="p"
            className={classes.temp}
          >
            Date: {item.date}
```

Figure 14. An example function component in one React JS project.

4.2 Class Component

There is a traditional component called class component. Class component uses the syntax of ES6. The class component are more complex than the functional components in that the class component also have: constructor, life-cycle, render () function and state (data) management. Moreover, the class component requires extending from Class Component as including extends component from React framework statement to generate an inheritance to Class Component for accessing to Class Component function. Besides, the class component also needs a render method to return React elements.
Figure 15 illustrates the React class component how looks like. Those code below is the FormSearch component in the practical Tama React App. There are to have enough mandatory methods, such as constructor, render, return and may have life cycle methods, also the state to manage the render of application.



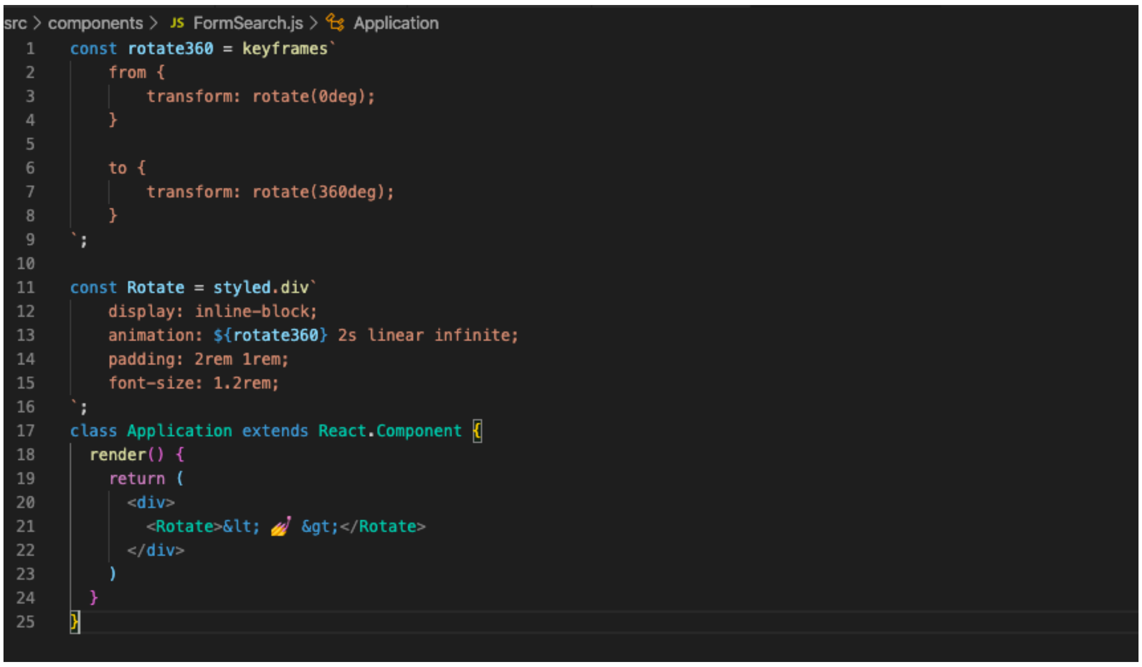
```
class FormSearch extends Component {
    constructor(props) {
        super(props);
        this.state = {
            query: '',
        };
    }

    handleChange = (event) => {
        const target = event.target;    // input selectbox
        const value  = target.type === 'checkbox' ? target.checked : target.value;
        const name   = target.name;

        this.setState({
            [name]: value
        });
    }

    handleSearch = (event) => {
        let {query} = this.state;
        this.props.changeQuery(query);
        event.preventDefault();
    }

    handleClear = (event) => {
        this.props.changeQuery("");
        this.setState({query: ''});
        event.preventDefault();
    }

    handleKeyPress = (event) => {
        if(event.key === 'Enter'){
            this.handleSearch(event);
        }
    }


    render() {
        let query = (this.state.query !== '') ? this.state.query : this.props.query;

        return (
            <form onSubmit={this.handleSearch} className="form-inline" >
                <div className="input-group">
                    <input name="query" value={query}  onKeyPress={this.handleKeyPress}  onChange={this.handleChange} type="text" className="form-control" pla
                    <div className="input-group-btn">
                        <button onClick={this.handleSearch} type="button" className="btn btn-danger">Search</button>
                        <button onClick={this.handleClear} type="button" className="btn btn-warning">Clear</button>
                    </div>
                </div>
            </form>
        );
```

Figure 15. An example of React class component.

4.3 Styled component

Styled Component is introduced to write CSS code that has the local scope, it means that CSS modules just affect a single component. Styled-Components is a library where organizing and manage CSS code easily in React projects. The component was built with the goal of keeping the styles of React components attached to the components themselves. The compinent provides a clear and easy to use interface for both React and React Native. The styled component not only changes the implementation of components in React, but also the way of thinking in building styles for those components.

In figure 16, the code defines animations in Tama App in outside company, the styled-Components provides a mechanism to generate unique names for the keyframes in each component, therefore there is no need to worry about whether the names of the keyframes are duplicated or not. Moreover, there is customStyledComponent that is defined by style component. When using styled component, it is easy to handle, and change style of elements, in order to improve performance of project, React developers do not need to use normal component to create simple component included only styles.



```
const rotate360 = keyframes`
    from {
        transform: rotate(0deg);
    }

    to {
        transform: rotate(360deg);
    }
`;

const Rotate = styled.div`
    display: inline-block;
    animation: ${rotate360} 2s linear infinite;
    padding: 2rem 1rem;
    font-size: 1.2rem;
`;
class Application extends React.Component {
  render() {
    return (
      <div>
        <Rotate>&lt; 💅 &gt;</Rotate>
      </div>
    )
  }
}
```

Figure 16. An example of style component.

Moreover, using Styled Components, React developers are allowed to have following profits:

- No pain maintenance: when maintaining an application, developers need to hunt across hundreds files, even thousands files to find out which one affects a component, it wastes a lot of developer’s time. But with Styled Component, the file affects a component will be detected easily.
- No name bugs: Style Components just allow to use unique class names during styling. That is why developers will not have the case two or more class names have the same name when using Style Components in their application and the critical bugs about class names are not the problem anymore.
- Automatically inject CSS: track all Styled Components to realize which components are rendered in order to automatically inject their styles and do not do anything else. It means the amount of necessary code that needs to be loaded is the smallest.
- Dynamic styling: when adapting the styling of components usually are managed manually by a lot of classes, but with Styled Components, that can be worked in a simple way based on its props or a global theme.

4.4 Web component

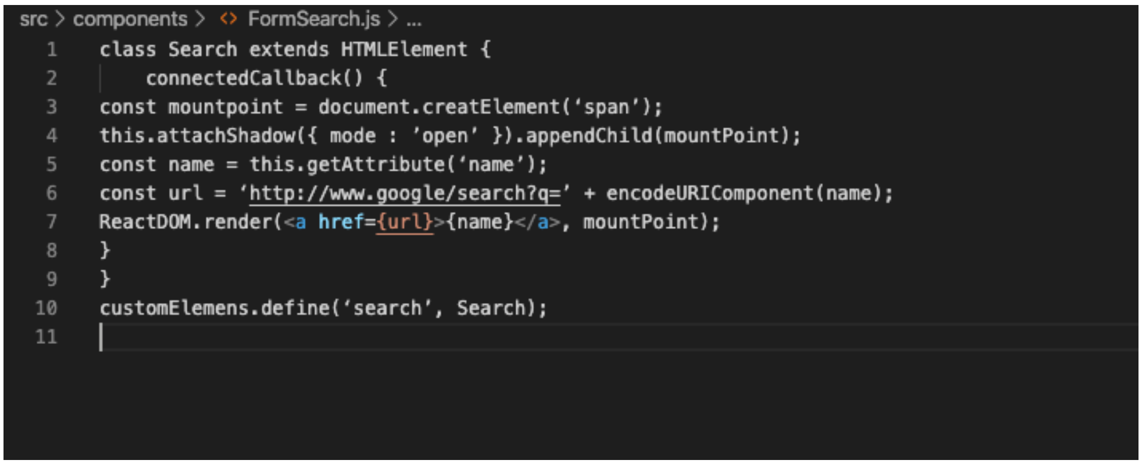
Web components are a relatively new concept, it was only introduced by Alex Russel in 2011 [7] in a front-end workshop. Google has actively developed this technology with the open source Polymer project, and promises to bring a breakthrough in Web development. Web technology has been developing strongly, the complexity is therefore increasing. This brings a lot of headaches for web developers, when the number of components in a website is increasingly larger and more demanding.

Web components are a set of technology rules for building website components that are packaged (separate from the rest of the application code) and reusable. Web Components provide packaging to reuse components. Web Components often display some compulsory APIs as video Web components can display play() and pause() functions. Therefore there is to have to use ref to interact with the DOM node to access the compulsory APIs of the Web Component.

Besides, a Web Component's events can pass in a wrong way via a React render; so the project should be manually attached handing events to address the events into a React Components.

Basically web components use 3 technologies:

- Custom Elements: is a set of JavaScript APIs to allow the creation of custom web components, such as the <my-todos> tag in the example above.
- Shadow DOM: is a collection of JavaScript APIs to attach a "shadow" DOM tree to a DOM element, this DOM tree is managed separately and rendered separately from the main DOM tree, so the component can be repackaged. To use for different applications.
- HTML templates: Used to create templates from which to render on web pages. This concept is relatively similar to some other frameworks like angular or react.js.



src > components > <> FormSearch.js > ...

```
class Search extends HTMLElement {
    connectedCallback() {
const mountpoint = document.createElement('span');
this.attachShadow({ mode : 'open' }).appendChild(mountPoint);
const name = this.getAttribute('name');
const url = 'http://www.google/search?q=' + encodeURIComponent(name);
ReactDOM.render(<a href={url}>{name}</a>, mountPoint);
}
}
customElemens.define('search', Search);

```

Figure 17. An example using web component in React JS.

5 REDUX

Nowadays, when the requirements for Single Page Applications build in JS are more complex, our code must manage more states than before. These states include the data returned from the server and cached data, as well as the data created locally and does not guarantee integrity compared to the server. In addition, the state used to manage the UI also increases the complexity as we need to manage which Routes are active, which tabs are selected, Paginations.
Managing state clusters that are always changing is difficult. For example, one model can update another model, then one view can update the above model, which means it also updates the other model, and here when the second model updates, it can also drag. Another view updates with it. At this point, we don't have a lot going on, losing control of when, why, and how the state is updating. If the system has loose or inconsistent code, it may be difficult to reproduce bugs, debug, or add new features.
Try to consider the case, there are new requirements from customers, which gradually become popular when working in the front-end. For example, we have to control when the component should update, server-side rendering, get data before advancing to change routes, etc ... Now React developers are trying to manage an extremely complex logic. magazines that have never been seen before.
There are two confusing concepts: "mutation and asynchoronicity". These 2 things are very separate, but when mixed together, it is very difficult to understand. Libraries like React have somewhat solved this, but only in the View, and the state containing the data, so finding the solutions have to solve it ourselves. Redux is needed to solve it.

5.1 Managing state by Redux

Suppose having an application where the nodes shown in the figure represent a single page application modeled as a tree-node model - in figure 18 [8]. Imagine the application running by exchanging data between nodes, each node (subpage) contains a state and child nodes receive data passed from parent to child.

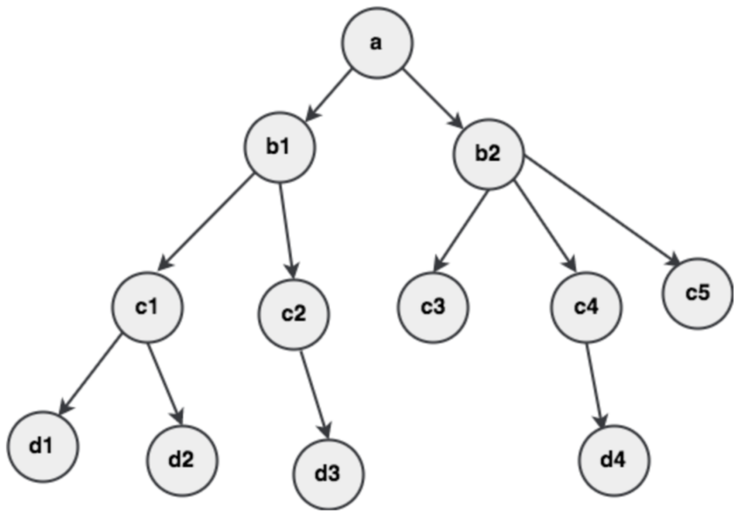


Figure 18. Node tree to simulate the states in React project.

Suppose if there is an action on node d3 that is activated and the requirement to change state d4 and c3 then the data stream will be transmitted from node d3 back to root node a, then from node a again transmits data to the child node (figure 19 [9]):

- Update state for node d4: d3-c2-b1-a-b2-c4-d4.
- Update state for node c3: d3-c2-b1-a-b2-c3

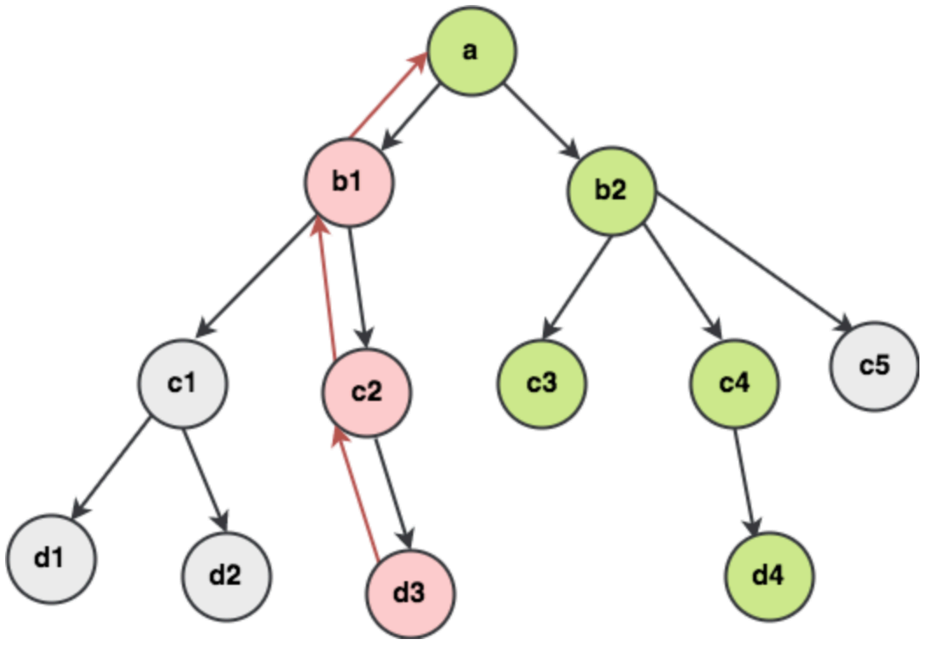


Figure 19. Update the state nodes without using Redux. [11]

For small problems (just using React JS without Redux), updating the state between pages can be easy, but try to imagine a larger application with lots of branches and child

nodes. Back and forth between the pages become more complex, making the flow of code difficult to read and debug.
Back to the above problem, in figure 20 [10], redux helps to map the Action from node d3 to the Redux's store and then at c3 and d4 nodes just need to connect to the store and update the changed data.

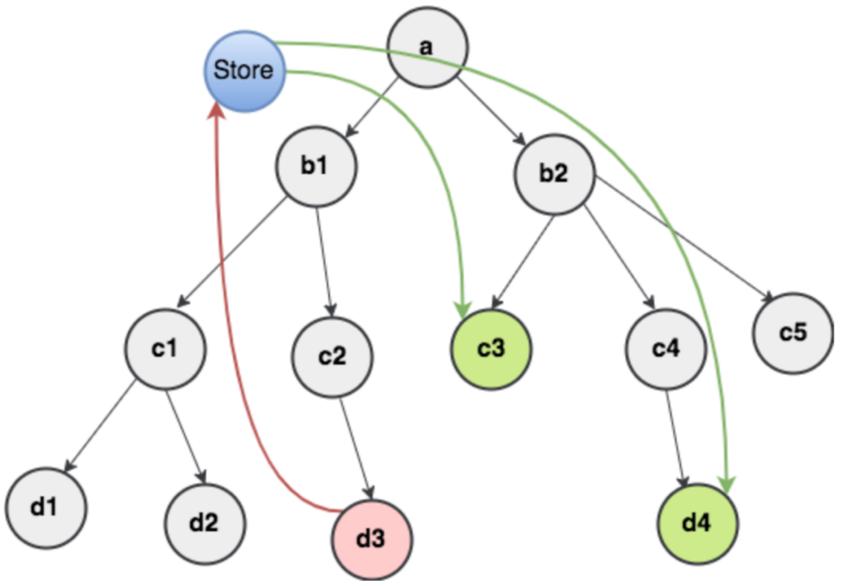
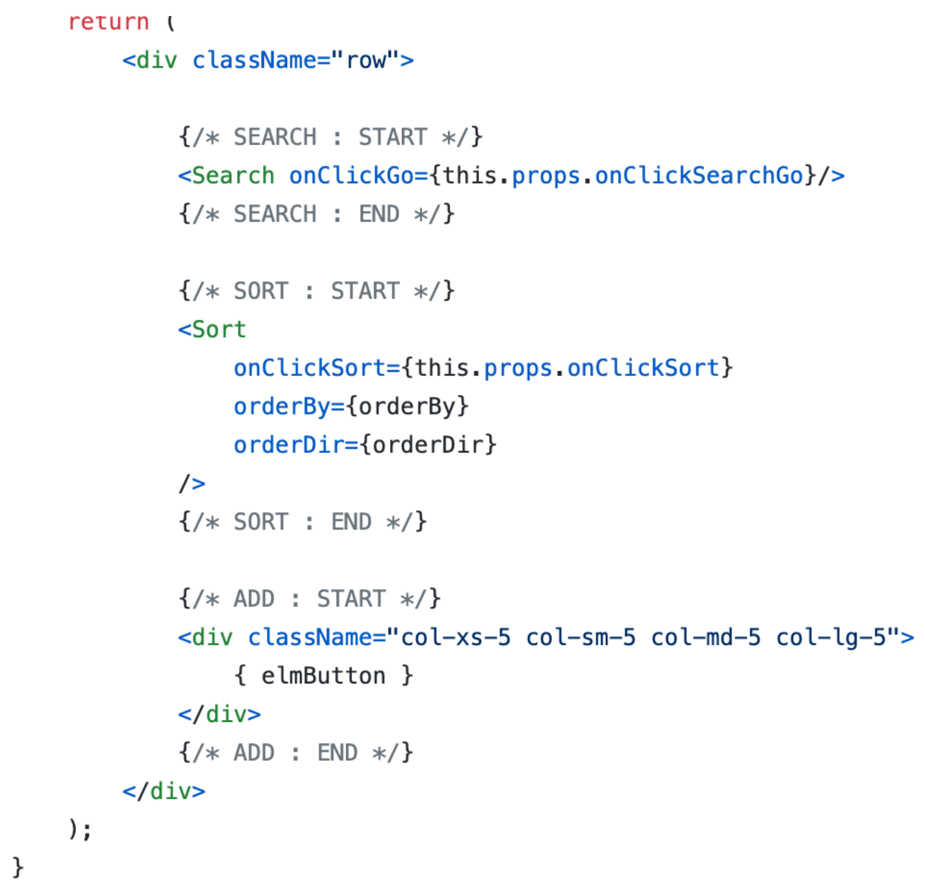


Figure 20. Update state nodes using Redux. [11]

5.2 State management tool

Most libs like React, Angular, etc are built in such a way that the components go to internal management of their states without any external libraries or tools. Those frameworks will work well for applications with few components, but as the application gets larger, the management of states shared through components will turn into odd jobs.
In an app where data is shared via components, it's easy to get confused so we can really know where a state is live. Ideally, data in a component should live in only one component. So sharing data through components will become more difficult.
For example, in react to share data through sibling components, a state must live in the parent component. A method to update this state itself will be provided by the parent component itself and pass as props to child components.



```
      return (
          <div className="row">

              {/* SEARCH : START */}
              <Search onClickGo={this.props.onClickSearchGo}/>
              {/* SEARCH : END */}

              {/* SORT : START */}
              <Sort
                  onClickSort={this.props.onClickSort}
                  orderBy={orderBy}
                  orderDir={orderDir}
              />
              {/* SORT : END */}

              {/* ADD : START */}
              <div className="col-xs-5 col-sm-5 col-md-5 col-lg-5">
                  { elmButton }
              </div>
              {/* ADD : END */}
          </div>
      );
  }
```

Figure 21. An example does not use Redux for managing global states in React.

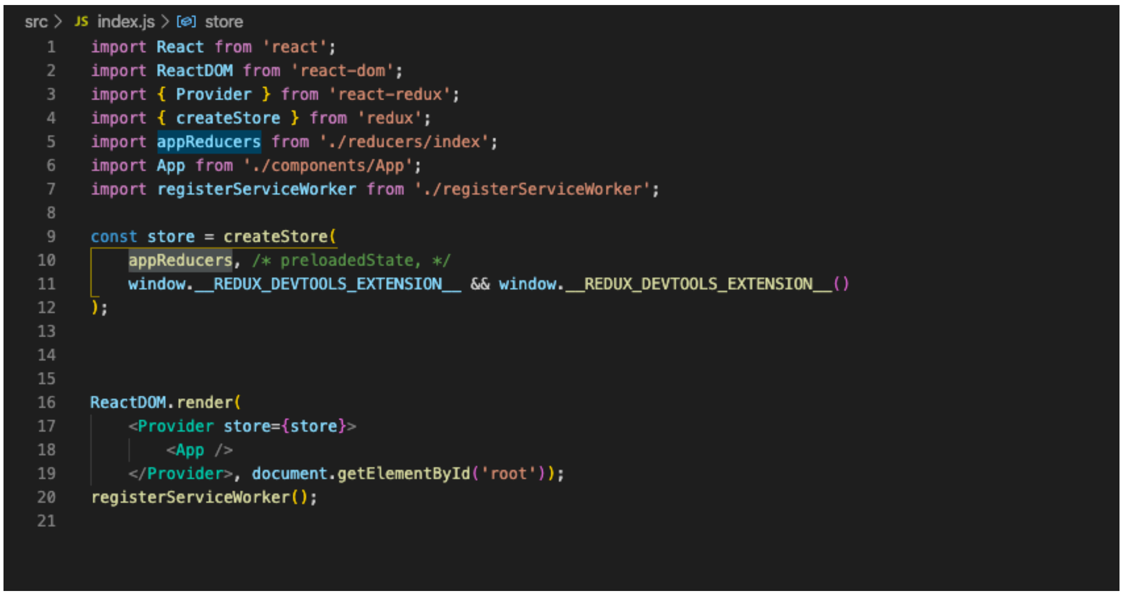
Now let's imagine that if a state must be shared between components quite far apart in a tree component and this state must be passed from one component to another until it reaches where it is called.
Basically, the state we are talking about must be lifted to the nearest parent component and continue until it reaches the ancestral component containing all the components it needs this state then pass this state down. This will make the state more difficult to maintain and less predictable.
This makes the state management department in the app become messy as the app becomes extremely complicated. That is why we need a state management tool like Redux.

5.3 Store

Store is a large JavaScript object with many key-value pairs representing the current state of the application. Unlike the state objects in React that are spread across different components, we only have one store. Store provides application status and every time the state updates, the view is reloaded. Store in Redux is like a human brain. The entire state of the application is in this store.
There are main store’s responsibilities :

- Hold application state.
- Accessing state from getState().
- Updating state via dispatch(action).
- Sign-up listeners via subscribe(listener).
- Handling listeners do not sign-up via the return by subscribe(listener).

In figure 22, there is the store created by the createStore method and used in the Taama App to manage reducers and middleware. After that, the reduxDevTool is imported to the the store, in order to observe the changing of the states in redux store. Finally, using the context with Provider helper API of react-redux to go down the store to the chidlren components and those components can use the store directly.



```
import React from 'react';
import ReactDOM from 'react-dom';
import { Provider } from 'react-redux';
import { createStore } from 'redux';
import appReducers from './reducers/index';
import App from './components/App';
import registerServiceWorker from './registerServiceWorker';

const store = createStore(
    appReducers, /* preloadedState, */
    window.__REDUX_DEVTOOLS_EXTENSION__ && window.__REDUX_DEVTOOLS_EXTENSION__()
);



ReactDOM.render(
    <Provider store={store}>
        <App />
    </Provider>, document.getElementById('root'));
registerServiceWorker();
```

Figure 22. An example creating store in React.

5.4 Actions

Actions are contents of information transmitting data from the application to the store. The store has actions as its only resource of information. Those actions are sent to the store by the store.dispatch() method.

Actions are absolute JavaScript objects. There must be a type property which shows the type of action being implemented. These types need to be identified as string constants. When the app is big enough, it may need to be transferred into a different module. Apart from type, engineers can decide how the structure of an action object is. Functions that create actions are action creators. It is simple to combine it with the term "action.

Figure 23 illustrates that there are two actions: the first action is actLogin has two parameters that are username and password and the payload is the object that contains type, and username and password. Another action is actLogout and the payload is only type.

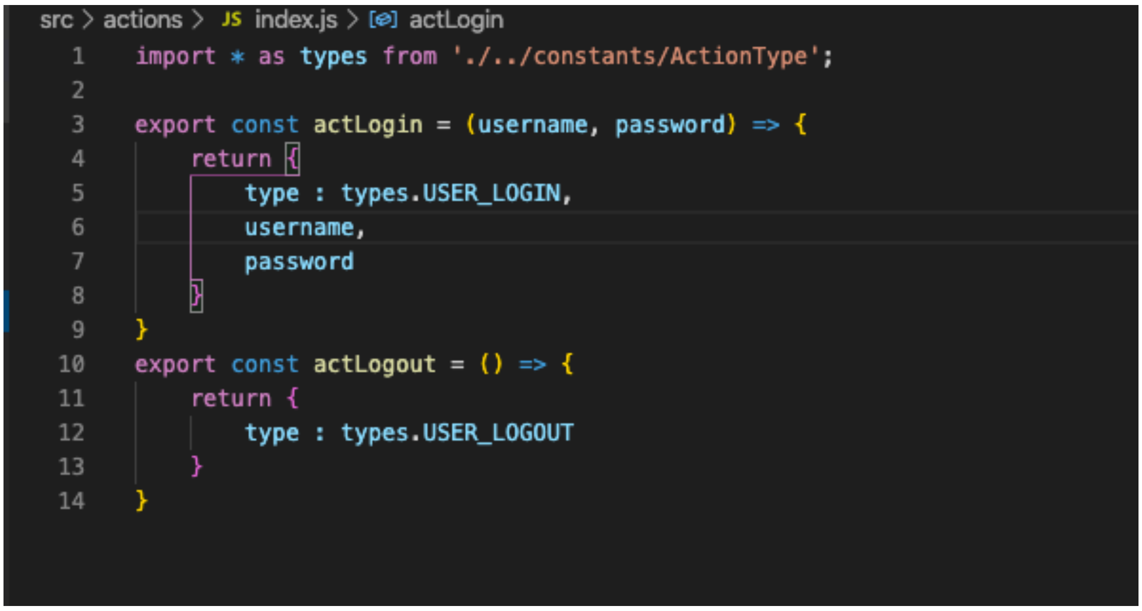
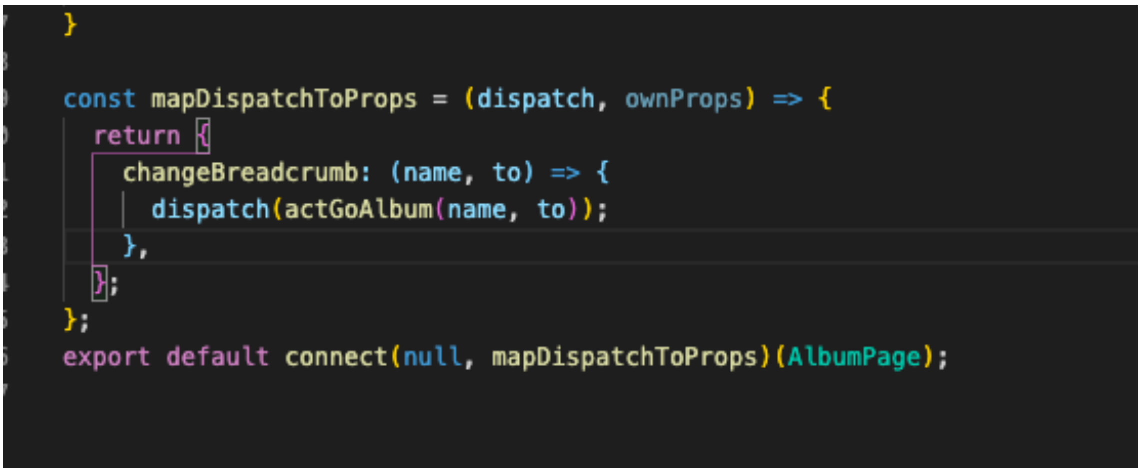

```
src > actions > JS index.js > actLogin
import * as types from './../constants/ActionType';

export const actLogin = (username, password) => {
    return {
        type : types.USER_LOGIN,
        username,
        password
    }
}
export const actLogout = () => {
    return {
        type : types.USER_LOGOUT
    }
}
```

Figure 23. An example of the actions.

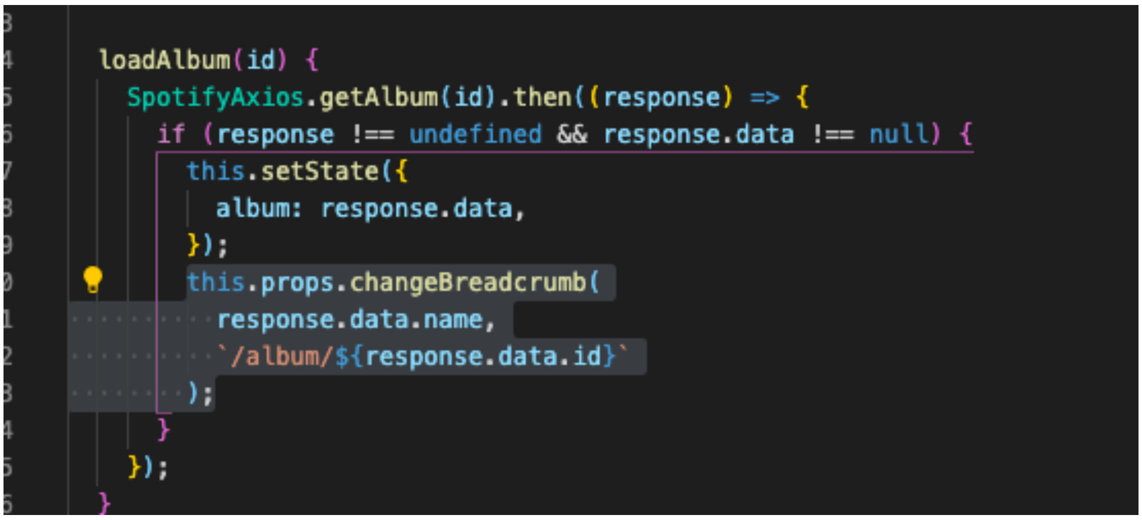
5.5 Dispatch to props

In Redux, mapDispatchToProps method is used to dispatch actions to the store. dispatch() method is a way to trigger a state change which is provided by Redux store, to use this method, developers can use the syntax store.dispatch to dispatch an action. The mapDispatchToProps method will be invoked as a first argument in dispatch function, it will return a new function that invokes and passes an action object to Redux store. The return of mapDispatchProps method needs to be a plain object.
In figure 24, there is a way to dispatch one method in React. After user created one action, the action will go to reducer, then dispatching that action and the actGoAlbum method is invoked. However, in figure 25, the dispatchToProps can be called through the props like: “this.props.changeBreadcrumb”.



```
  }

  const mapDispatchToProps = (dispatch, ownProps) => {
    return {
      changeBreadcrumb: (name, to) => {
        dispatch(actGoAlbum(name, to));
      },
    };
  };
  export default connect(null, mapDispatchToProps)(AlbumPage);
```

Figure 24. An example installing the mapDispatchToProps method.



```
    loadAlbum(id) {
      SpotifyAxios.getAlbum(id).then((response) => {
        if (response !== undefined && response.data !== null) {
          this.setState({
            album: response.data,
          });
          this.props.changeBreadcrumb(
            response.data.name,
            `/album/${response.data.id}`
          );
        }
      });
    }
```

Figure 25. Example of using the dispatchToProps method in React.

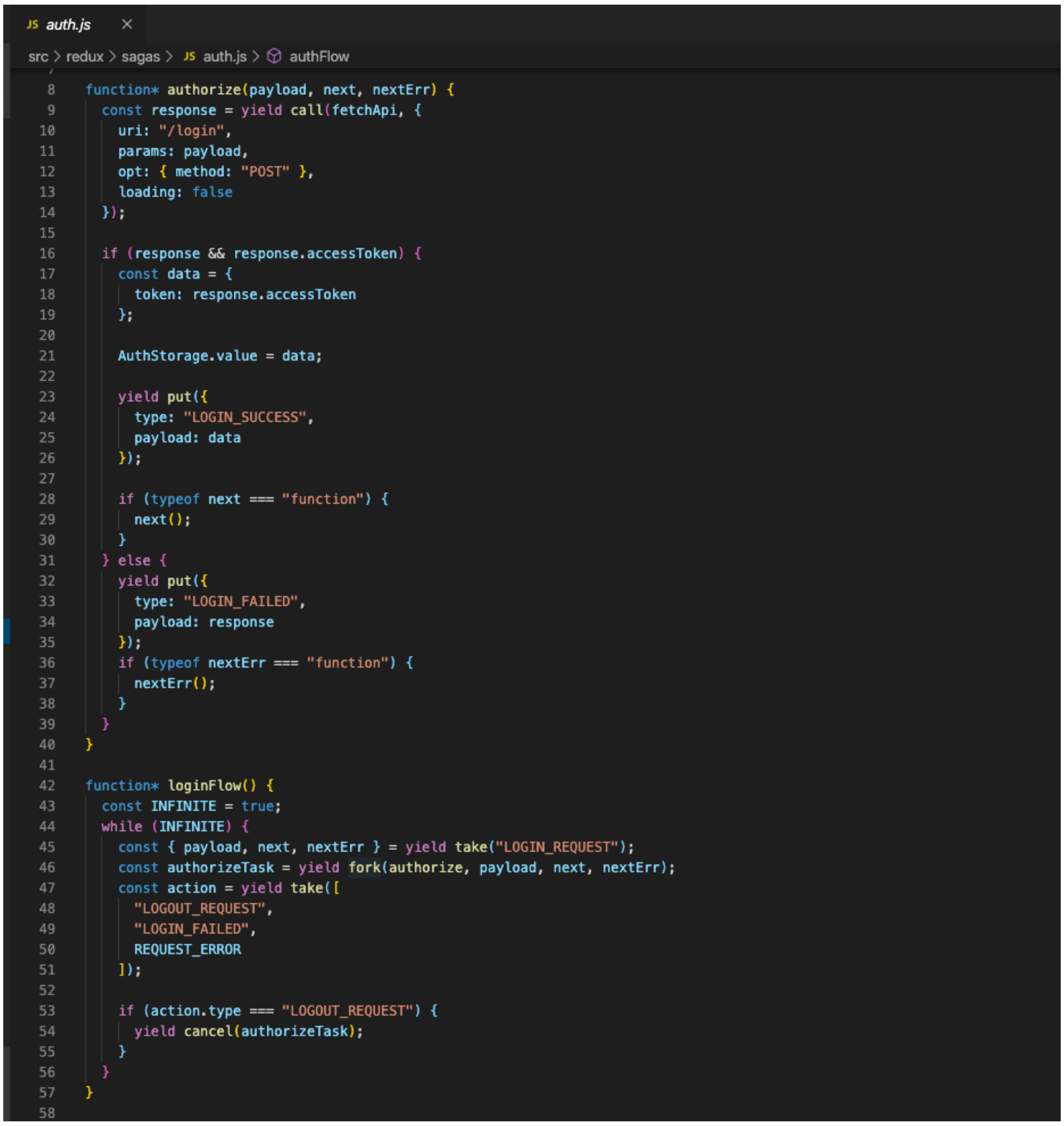
5.6 Middleware

Middleware is the middle code between requests and responses. It receives requests, executing corresponding commands on that request. Upon completion, it responds (returns) or passes the delegated results to another Middleware in the queue.
Nowadays, modern Web frameworks use it as part of the application to connect other parts. For web applications, using Middleware effectively helps us reduce the amount of code that must be written in the application.
With the common thought is the bridge between user interaction and the system in Web programming. The middleware will act as an intermediary between the request / response and the logical handling within the web server.

Therefore, Middleware in the frameworks for Web applications (Laravel, Django, Rails, ExpressJS ...), will be functions used to preprocess, filter requests before putting them into logical processing or adjusting responses before send to user.

In Redux, the concept of middleware also exists and plays a similar role, but the problem that middleware in redux solves differs from server-side. In Redux, middleware is the layer between Reducers and Dispatch Actions. The position where the Middleware operates is before the Reducers receive Actions and after an Action is dispatched (). Middleware in Redux is best known for handling ASYNC Action - actions that are not available as soon as an Action Creator is called, usually here are API requests.
Currently, there are quite a lot of middleware libraries for Redux, especially using with React, there are redux-thunk redux-saga and redux-observable and MobX. Each library has its own method of solving side effects.
The requirements to understand some basic concepts like generator function. The generator function is a function that has the ability to postpone execution while keeping the context intact. It's a bit confusing, in simple words, the generator function is a function that can pause before the function ends (unlike the pure function when called will execute all the statements in the function), and can continue. keep running at another time. It is this new function that helps solve the asynchronous, the function will stop and wait for the async to finish running and then continue to execute.
Redux saga provides helper effect functions, which will return an effect object that contains special information instructing Redux's middeware to perform other actions. The helper effect functions will be executed in the generator function.
Some helpers in redux saga:

- takeEvery (): executes and returns the result of every called action.
- takeLastest (): means that if we perform a series of actions, it will only execute and return the result of the last action.
- take (): pause until the action is received
- put (): dispatch an action.
- call (): call function. If it returns a promise, pause the saga until the promise is resolved.
- race (): run multiple effects simultaneously, then cancel all if one of them ends.



```javascript
function* authorize(payload, next, nextErr) {
  const response = yield call(fetchApi, {
    uri: "/login",
    params: payload,
    opt: { method: "POST" },
    loading: false
  });

  if (response && response.accessToken) {
    const data = {
      token: response.accessToken
    };

    AuthStorage.value = data;

    yield put({
      type: "LOGIN_SUCCESS",
      payload: data
    });

    if (typeof next === "function") {
      next();
    }
  } else {
    yield put({
      type: "LOGIN_FAILED",
      payload: response
    });
    if (typeof nextErr === "function") {
      nextErr();
    }
  }
}

function* loginFlow() {
  const INFINITE = true;
  while (INFINITE) {
    const { payload, next, nextErr } = yield take("LOGIN_REQUEST");
    const authorizeTask = yield fork(authorize, payload, next, nextErr);
    const action = yield take([
      "LOGOUT_REQUEST",
      "LOGIN_FAILED",
      REQUEST_ERROR
    ]);

    if (action.type === "LOGOUT_REQUEST") {
      yield cancel(authorizeTask);
    }
  }
}
```

Figure 26. An example using Redux saga with generator functions.

5.7 Reducer

Reducer indicates how the application’s state transforms in reply to actions transmitted to the store. In Redux, every application’s state is saved as a single object..
When the state object has been decided, the reducer is written for the App. The actions called a reducer since it is the kind of function being transferred to Array.prototype.reduce(reducer,? initialValue). The reducer needs to be clear. These are things that should not be done inside a reducer:

- Modify the arguments.
- Conduct side effects (API calls and routing transformation)
- Request non-pure functions..

In figure 27, it depicts the flow of redux. Firstly, user touches the view or layout on browser and the action is born, then the actions goes to reducer and the application will have the new states

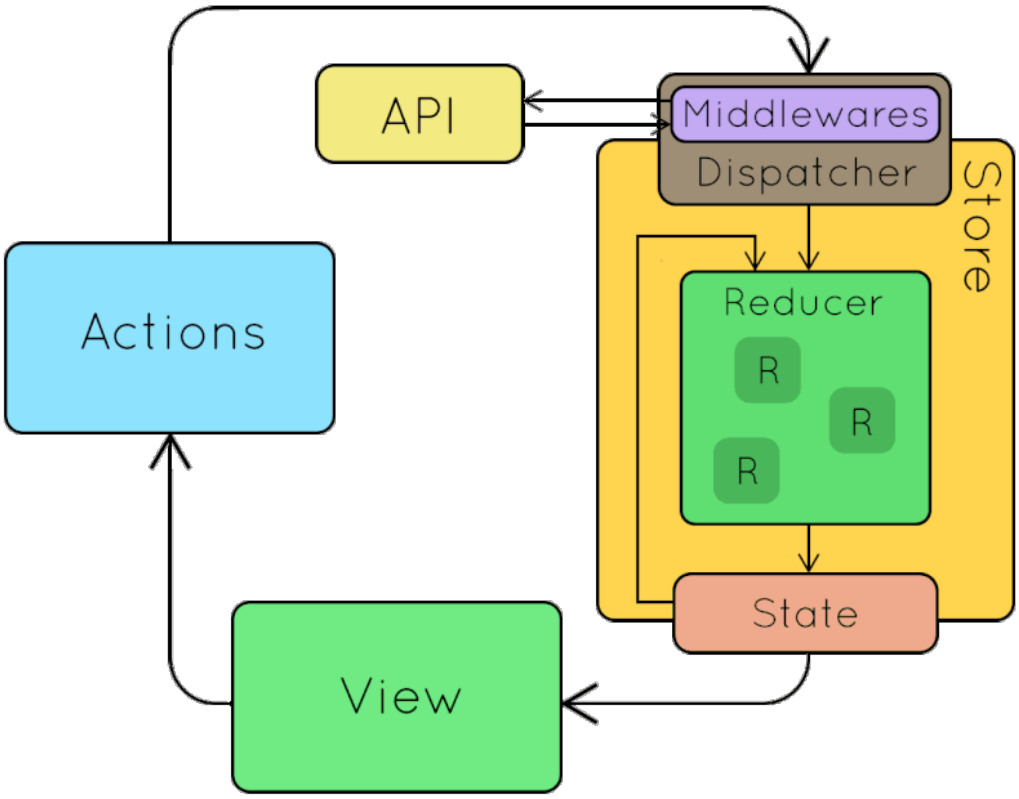


Figure 27. Flow Redux in React Project. [11]

6 IMPLEMENTATION

This chapter describes the implementation of shopping-cart application after enhancing programming skill in practical Tama React project, and demonstrates how React JS can help developer develops static HTML file and single page application. The implementation contains development, the structure of project and deployment.

6.1 System requirements

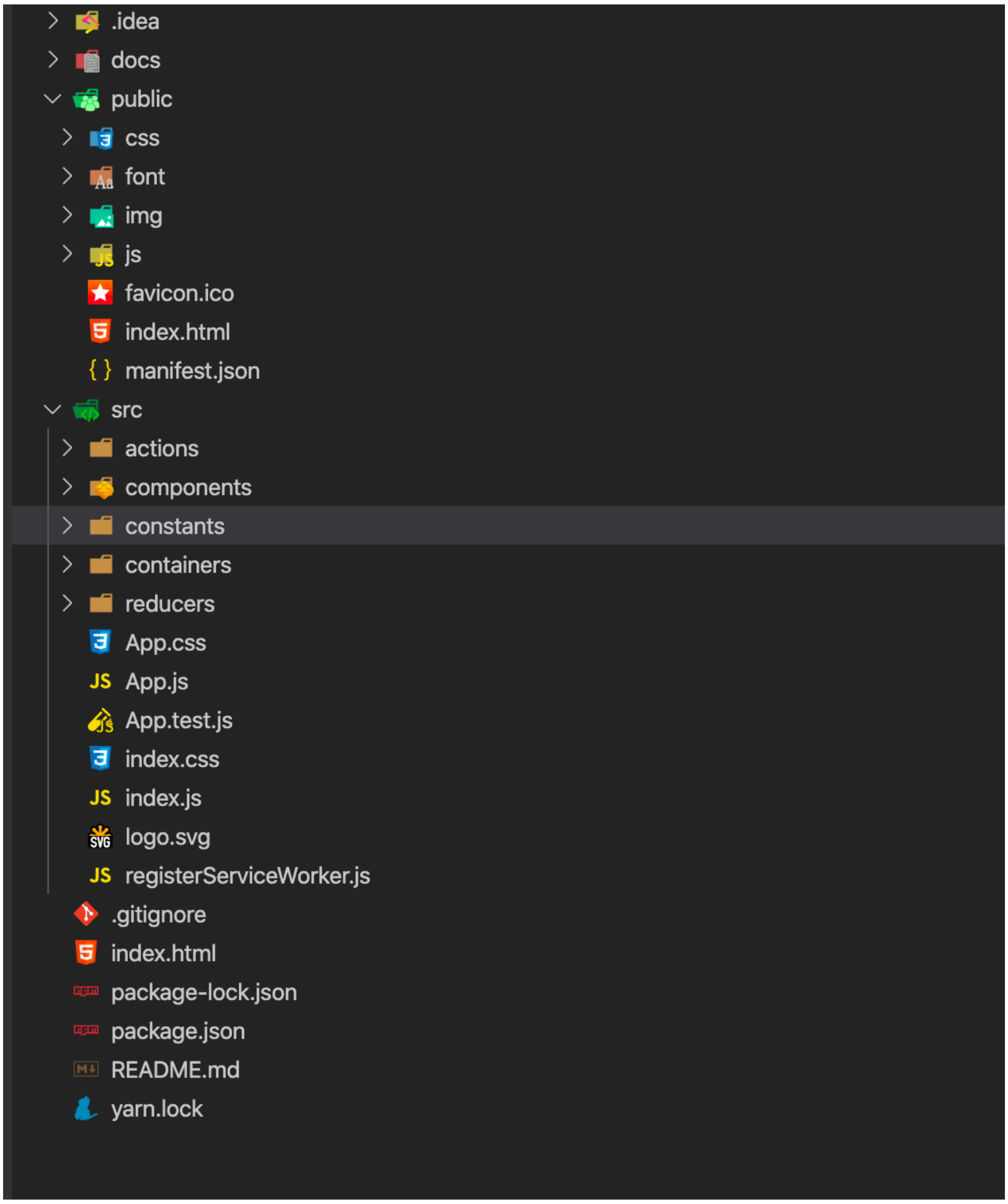
These Appilcation require NodeJS 10 and the NPM which can find more information at https://nodejs.org.
Git is the Source Code Administration apparatus for managing changes of the source code during developing the project. This instrument can be introduced by means of its official site https://git-scm.com

6.2 Structure and run

Running on local

This project was created by command line: `npx create react-app shopping-cart`. It is a fully working React application that developer can continue developing locally. It has all the necessary dependencies and files to run the project in `package.json`.
The project is a standard `Node_modules` pakage manager, so developer can import it to the IDE of choice with the command: `npm install name_dependency`.
In order to run from the command line, use `npm start` or `yarn start` and open ` http://localhost:3000/` in browser, but making sure the project is installed node already.

Project structure of Shopping cart



```
> .idea
> docs
v public
  > css
  > font
  > img
  > js
    favicon.ico
    index.html
    manifest.json
v src
  > actions
  > components
  > constants
  > containers
  > reducers
    App.css
    App.js
    App.test.js
    index.css
    index.js
    logo.svg
    registerServiceWorker.js
  .gitignore
  index.html
  package-lock.json
  package.json
  README.md
  yarn.lock
```

Figure 28. Shopping-cart App project structure.

Figure 29 depicts the structure of Shopping cart Application project using React and Redux. This structure contains source code of the project and configuration for development and deployment:

- “public” folder contains the source code of the projet after compiling to static HTML file and the project uses it to run the App.

- “src” folder contains the main source of the project, before compiling the source code.
- “actions” folder contains the actions of Redux.
- “components” folder is a place to contains the components of project. It only handles layout of the project.
- “containers” folders is a place to contains the logic code of the project. It is a place to connect to Redux store and bring the data to components.
- “reducers” contains reducer of Redux.
- “App.js” is a main source code file of project.
- “index.js” is a root file of project.
- “.gitignore” to ignore files and folders of project which developers do not upload to Github server.
- “pakage.json” contain dependencies and some command scripts of the project.
- “README.md” is a first file developer should read. It is a file to guide how to run and introduce all of things of the project.

Application user interface

Figure 30, in the right top there is a menu bar, it depicts the functionality related to account, such as login, and log-out and register of the user.

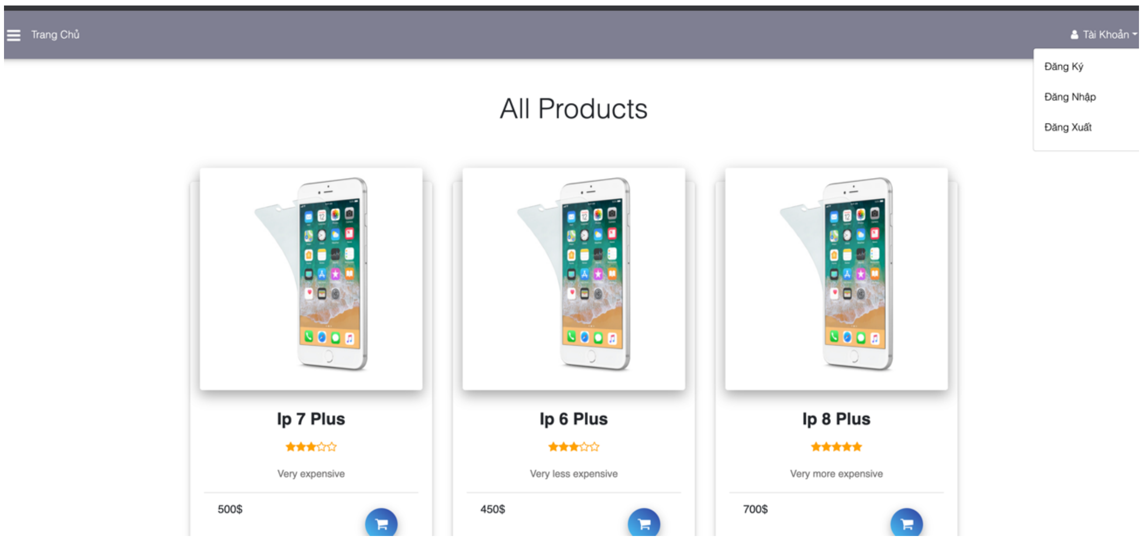


Figure 29. Shopping Cart user interface of login and register.

Figure 31 shows the home page of the Application with some items with theis information, such as the name, the price of the items.

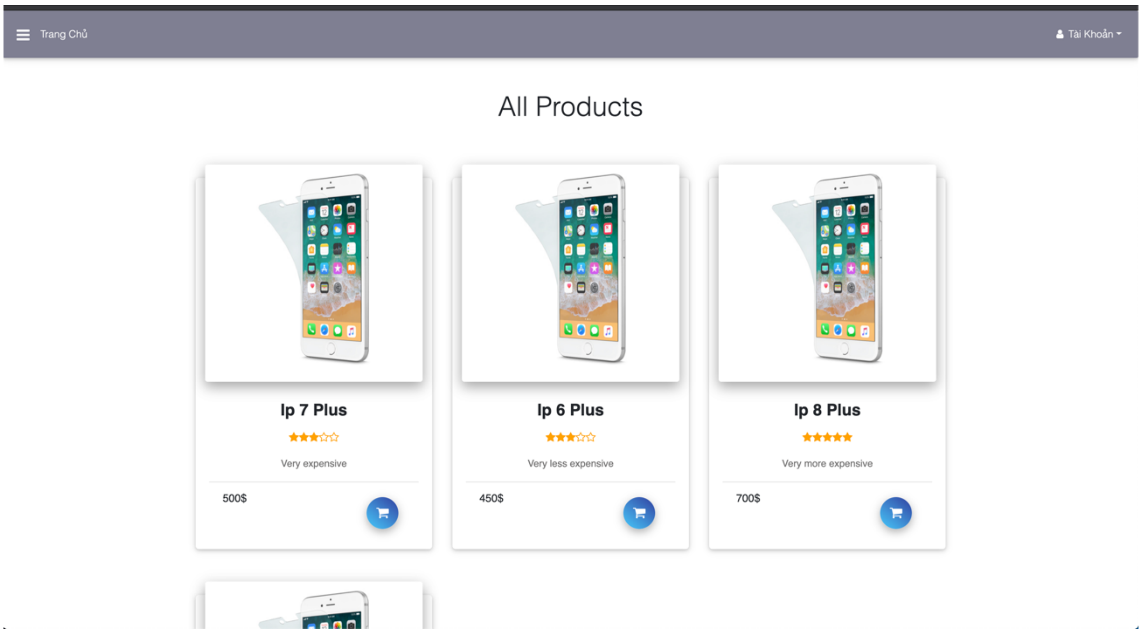


Figure 30. Shopping Cart user interface in home-page.

In figure 32, there is a notification to inform that buyer bought the items, or update the item.

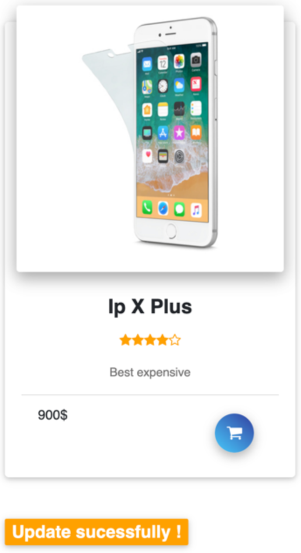
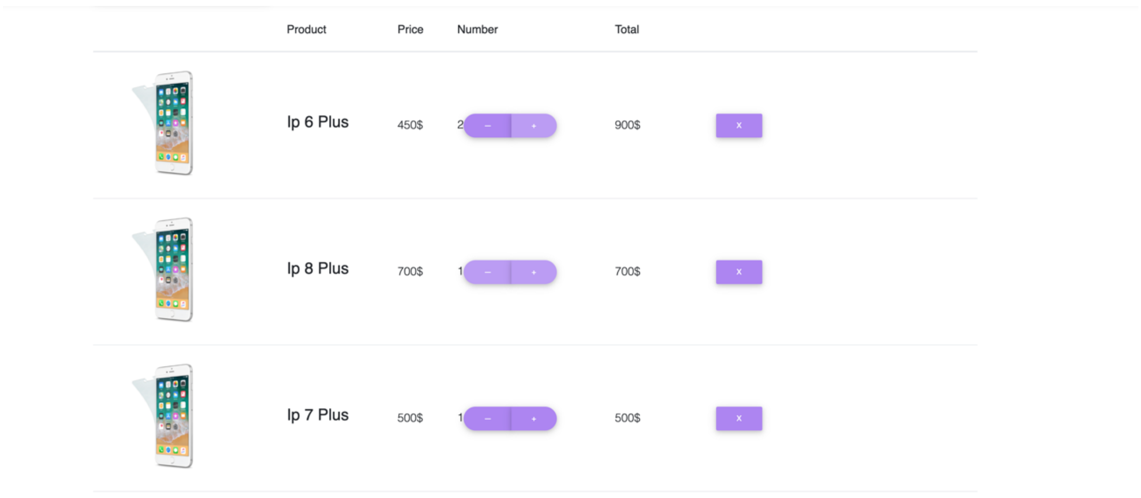


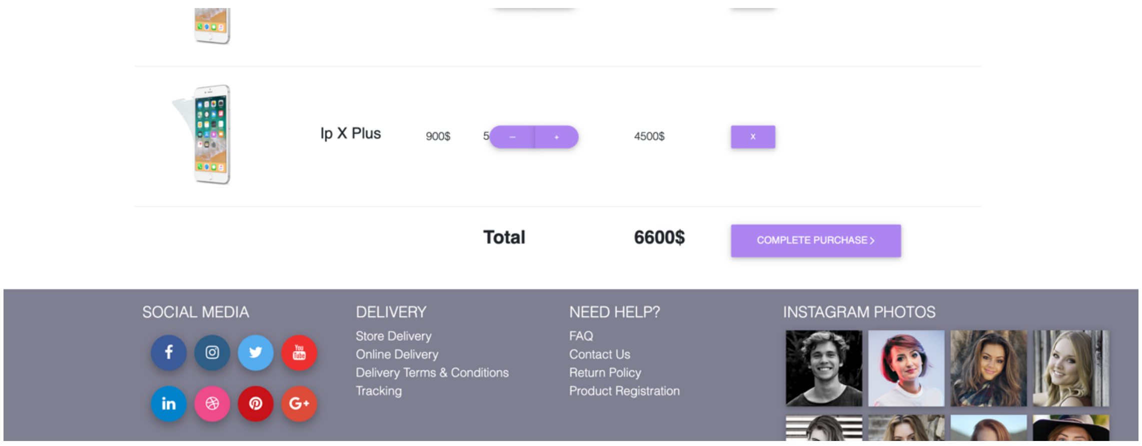
Figure 31. User interface of the message notification.

In figure 33, the photo shows the information of the items added to the cart. The buyer can update and remove the items.



	Product	Price	Number	Total	
	Ip 6 Plus	450$	2 – +	900$	x
	Ip 8 Plus	700$	1 – +	700$	x
	Ip 7 Plus	500$	1 – +	500$	x

Figure 32. User interface in cart.



	Ip X Plus	900$	5 – +	4500$	x
			Total	6600$	COMPLETE PURCHASE ›

Figure 33. User interface of the functionality of calculating the total of price.

Source code

The project developed and pushed to Github. There is the link: https://github.com/hongduc-phan/React-Redux-shopping-cart
In figure 35, it shows the commits during developing the application.

Project structure of Tama

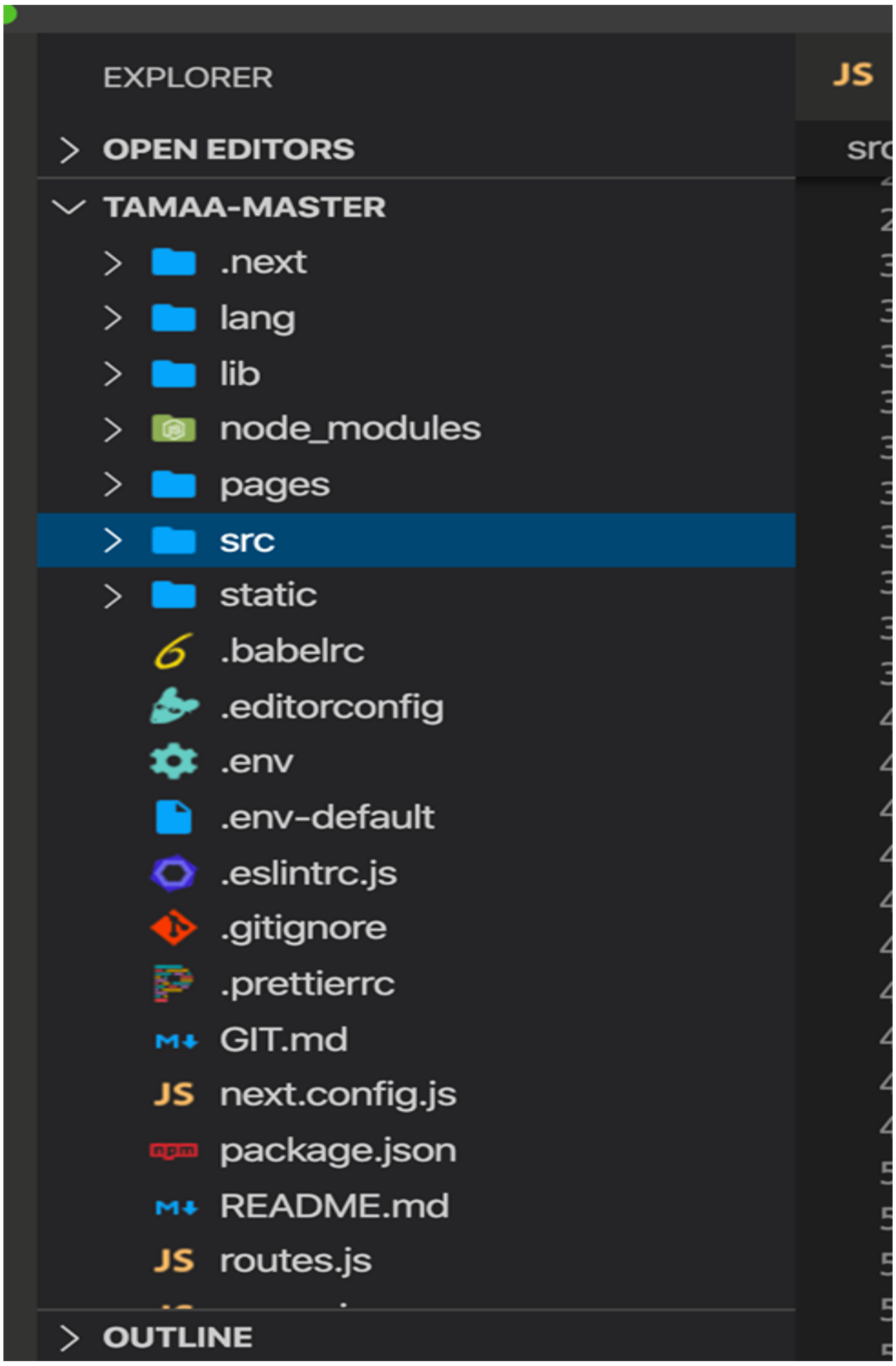


Figure 34 Tama App project structure.

Figure 36 depicts the structure of Tama Application project using React and Redux. This structure contains source code of the project and configuration for development and deployment:

- “public” folder contains the source code of the projet after compiling to static HTML file and the project uses it to run the App.

- “src” folder contains the main source of the project, before compiling the source code.
- “static” folder contains the compile source code of project.
- “pages” folder is a place to contains the pages of project.
- “index.js” is a root file of project.
- “.gitignore” to ignore files and folders of project which developers do not upload to Github server.
- “pakage.json” contain dependencies and some command scripts of the project.
- “README.md” is a first file developer should read. It is a file to guide how to run and introduce all of things of the project.

Application user interface

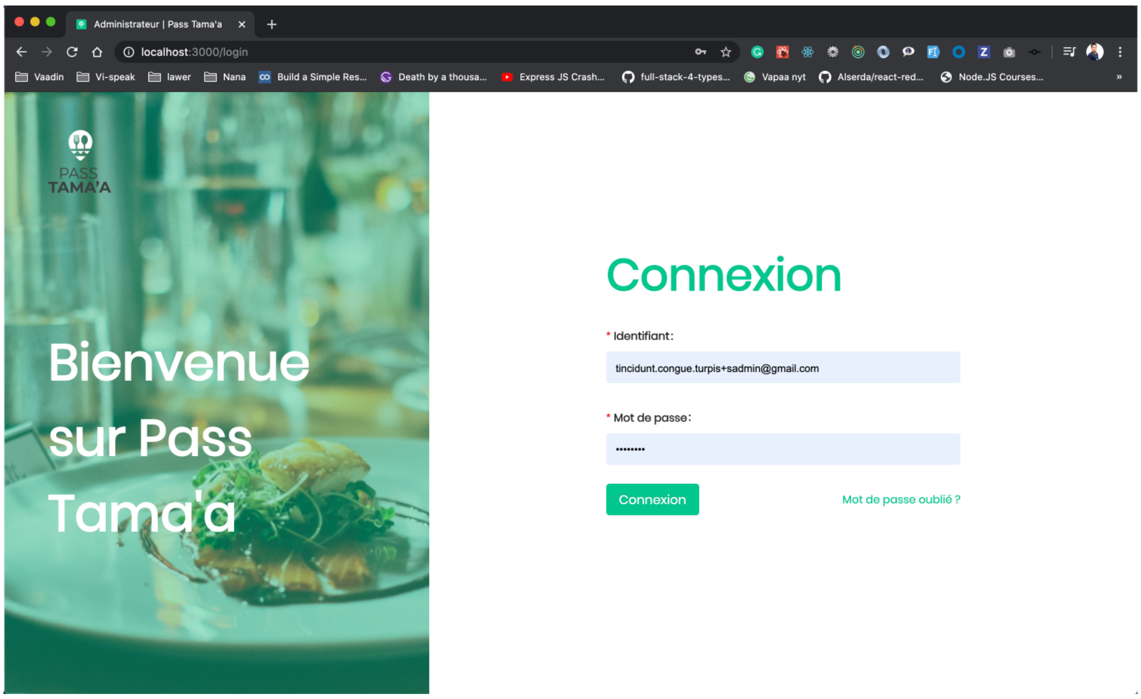
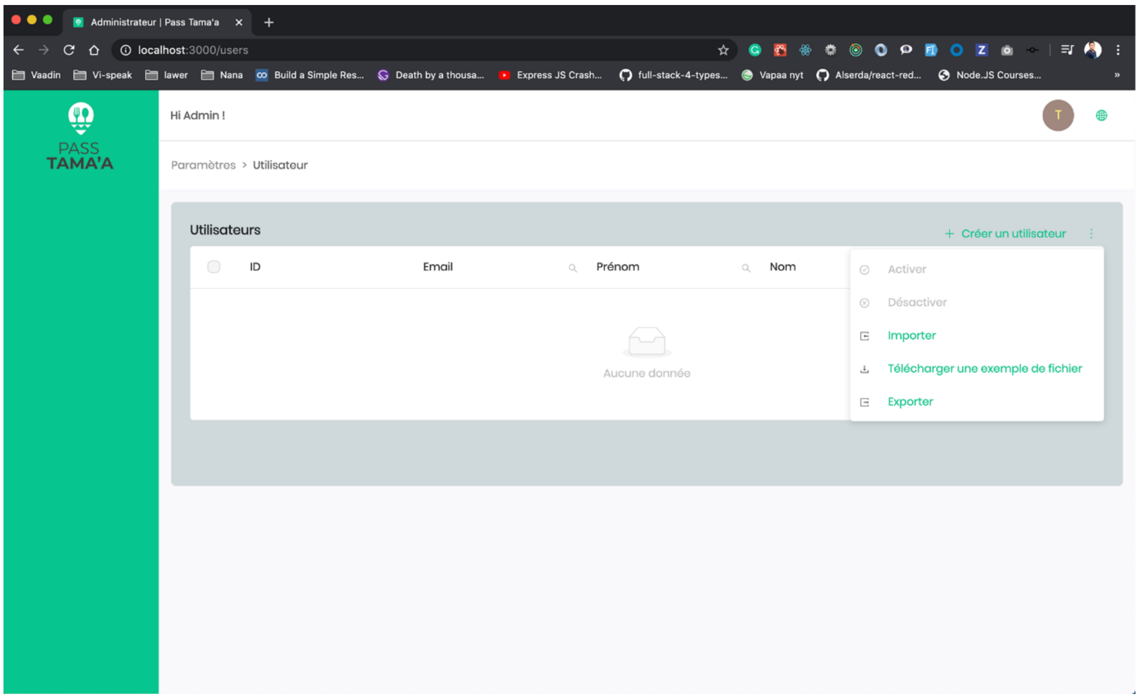


Figure 35 Interface login



Hi Admin !

Paramètres > Utilisateur

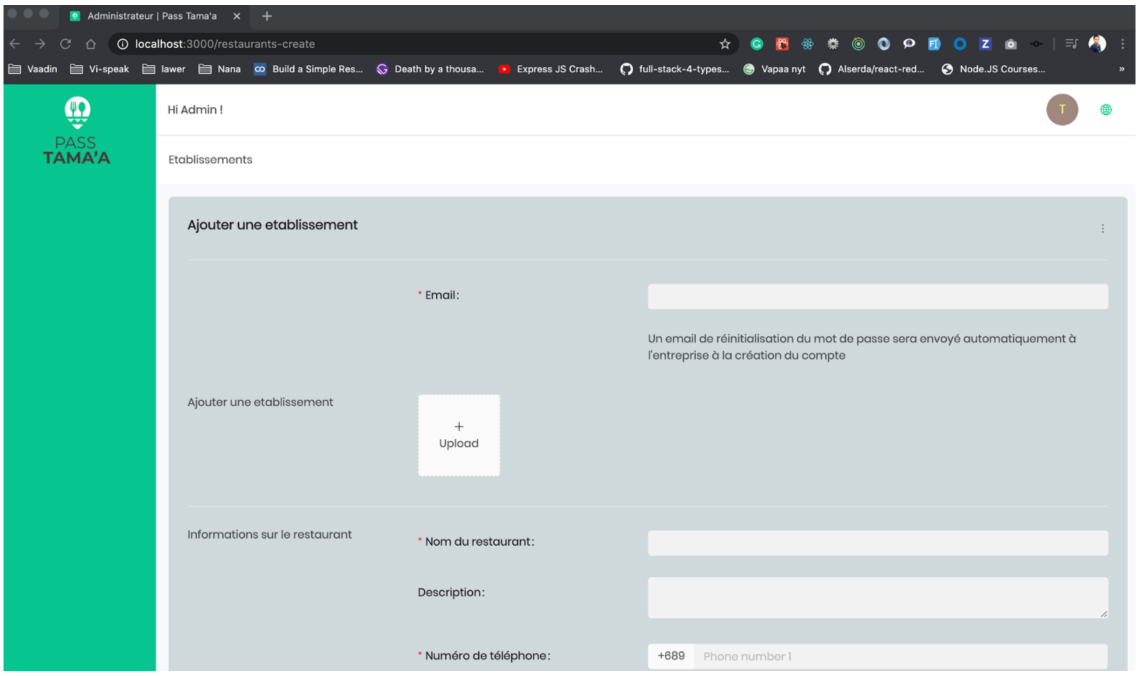
Utilisateurs

+ Créer un utilisateur

ID	Email	Prénom	Nom

Aucune donnée

Figure 36 User Interface



Hi Admin !

Etablissements

Ajouter une etablissement

* Email:

Un email de réinitialisation du mot de passe sera envoyé automatiquement à l'entreprise à la création du compte

Ajouter une etablissement

Informations sur le restaurant

* Nom du restaurant:

Description:

* Numéro de téléphone: +689 Phone number 1

Figure 37 Create restaurant with some input fields

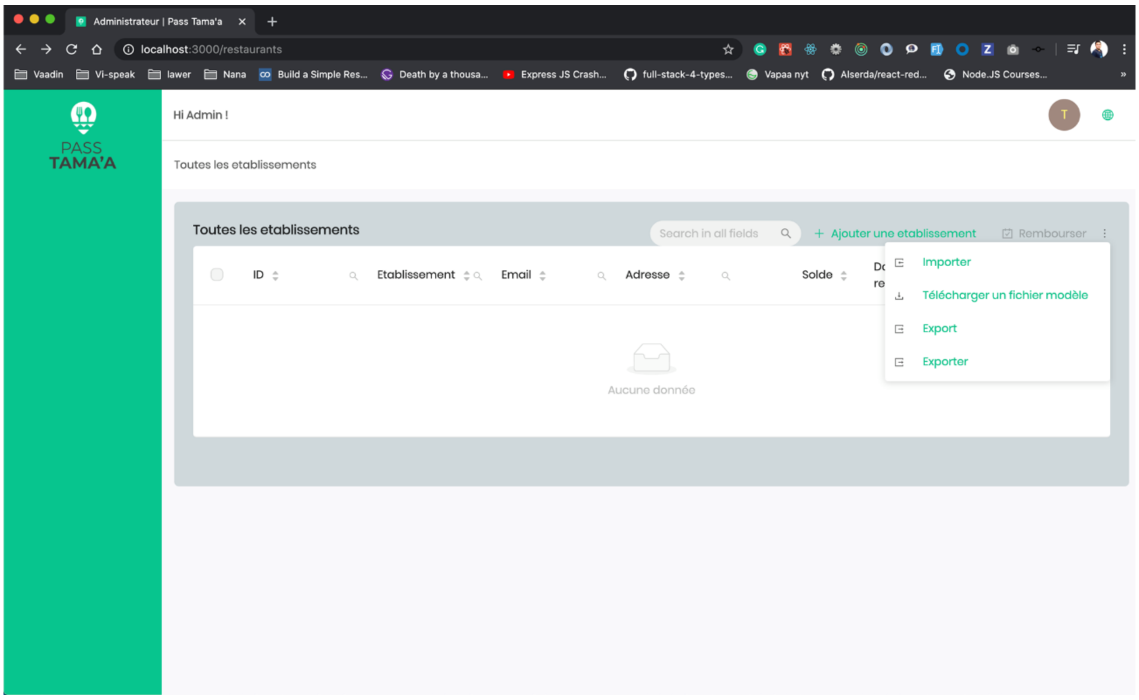

Administrateur | Pass Tama'a
localhost:3000/restaurants
PASS TAMA'A
Hi Admin !
Toutes les etablissements
Toutes les etablissements
Search in all fields
+ Ajouter une etablissement
Rembourser
ID
Etablissement
Email
Adresse
Solde
Importer
Télécharger un fichier modèle
Export
Exporter
Aucune donnée

Figure 38 Main restaurant with some features, such as import and export

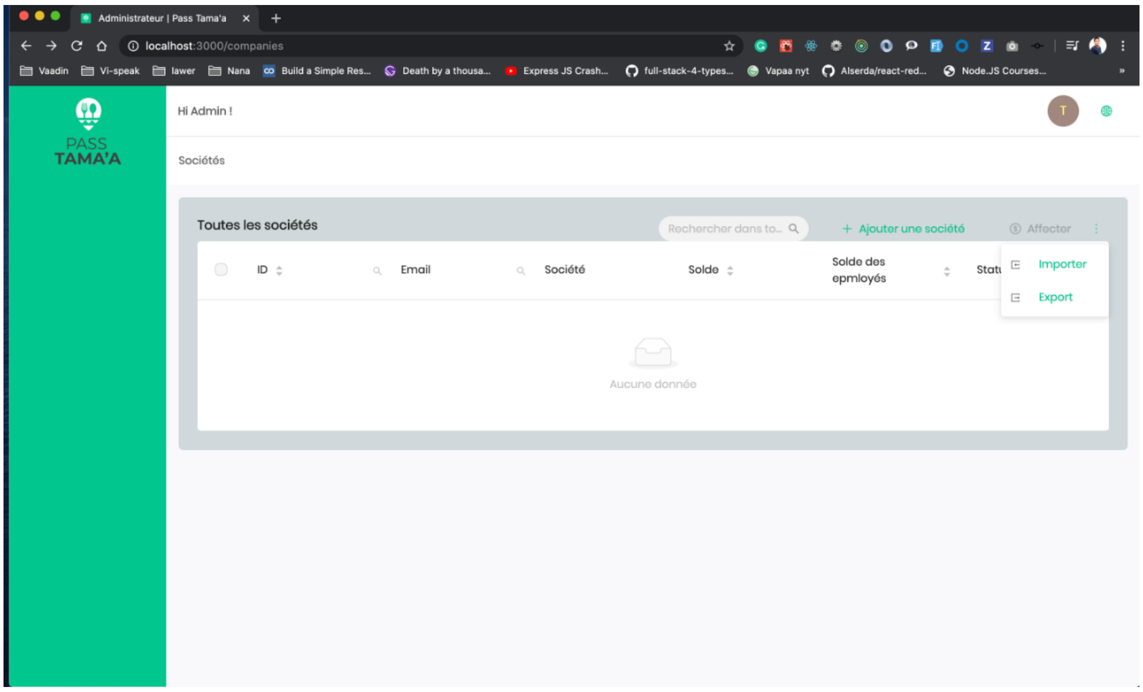
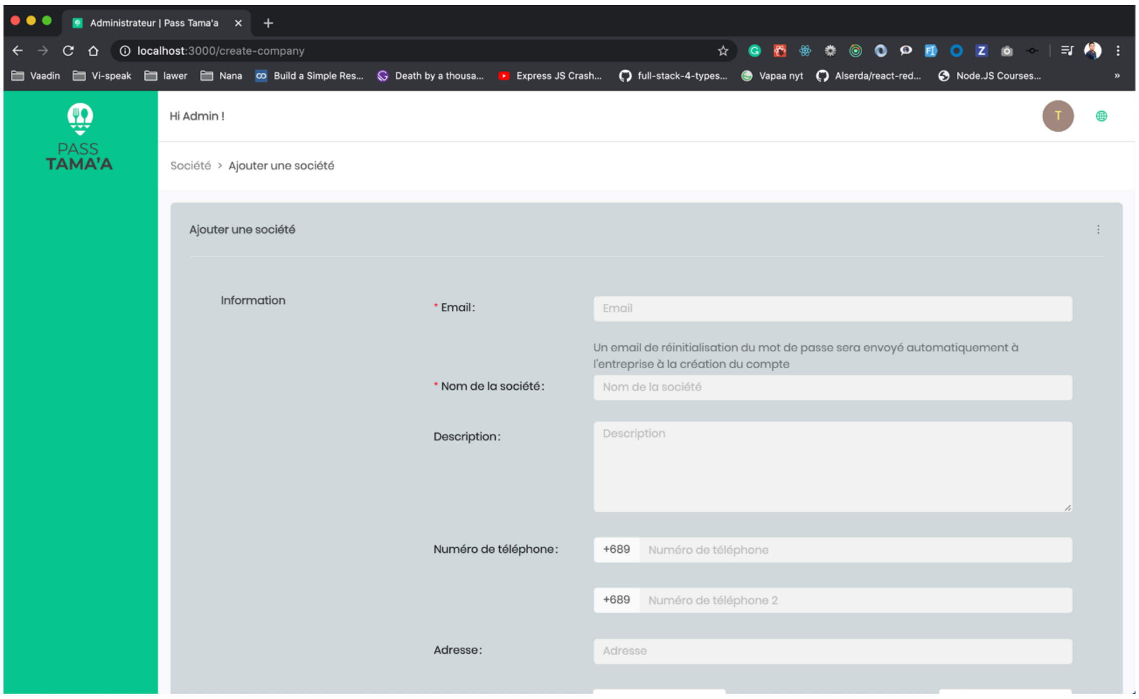


Figure 39 Main company interface with some features such as import and export

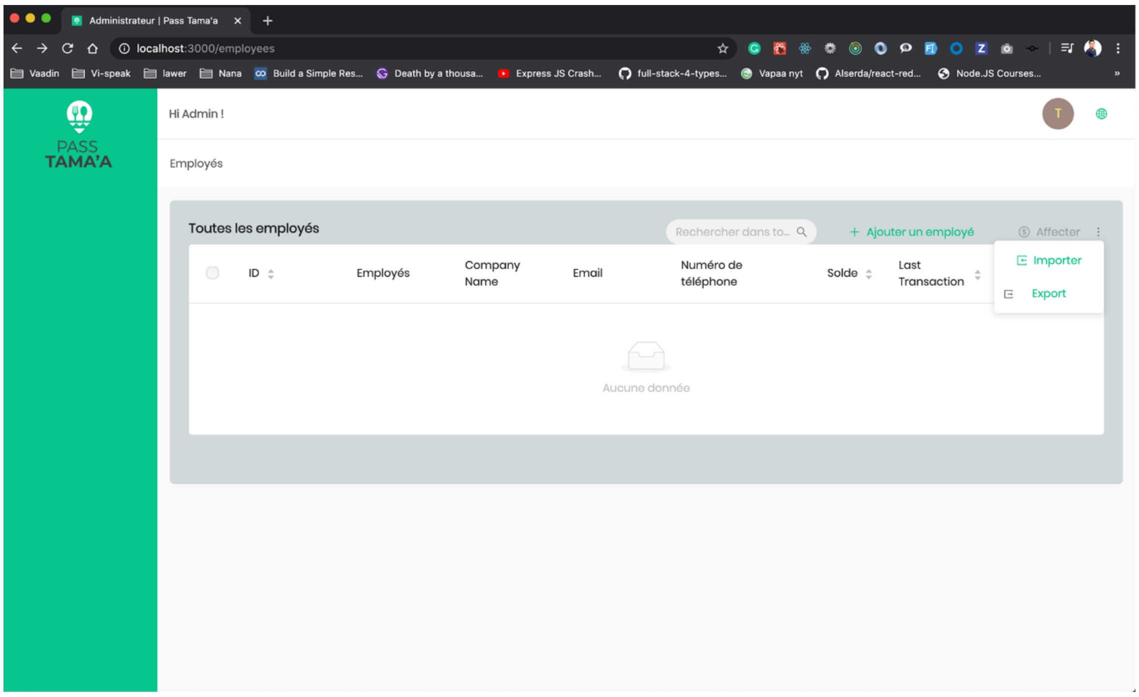


Ajouter une société

Information

* Email:	Email
	Un email de réinitialisation du mot de passe sera envoyé automatiquement à l'entreprise à la création du compte
* Nom de la société:	Nom de la société
Description:	Description
Numéro de téléphone:	+689 Numéro de téléphone
	+689 Numéro de téléphone 2
Adresse:	Adresse

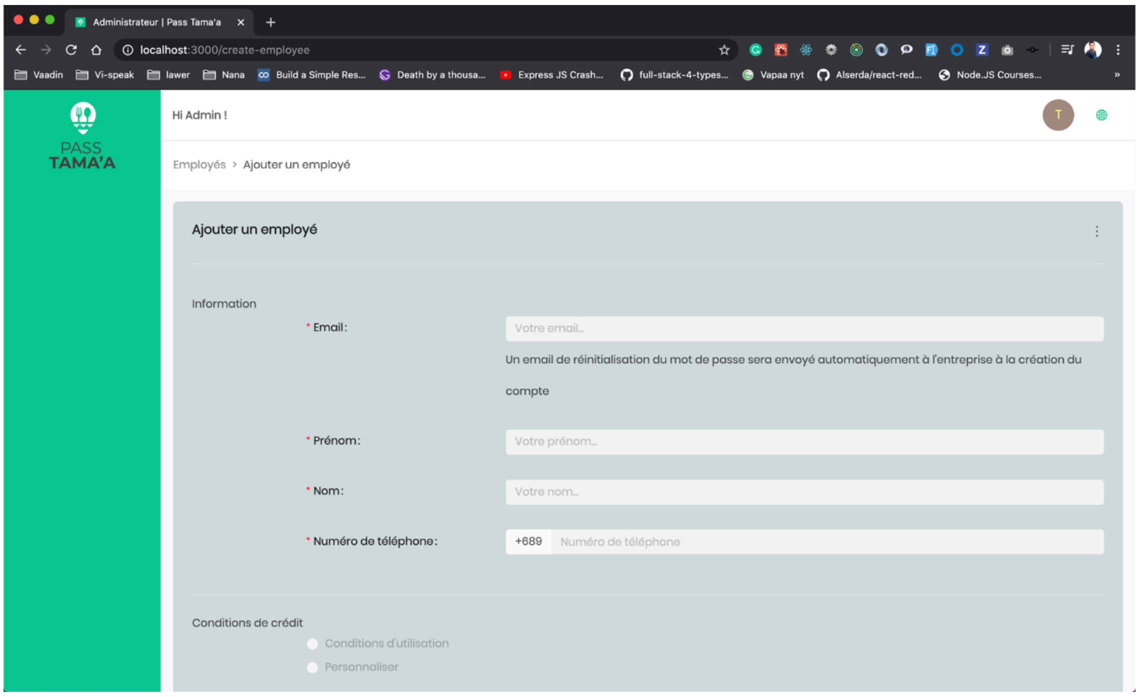
Figure 40 Create new company interface



Toutes les employés

	ID	Employés	Company Name	Email	Numéro de téléphone	Solde	Last Transaction
Aucune donnée							

Figure 41 Employees with some feature


Hi Admin !

Employés > Ajouter un employé

Ajouter un employé

Information

* Email: Votre email...

Un email de réinitialisation du mot de passe sera envoyé automatiquement à l'entreprise à la création du compte

* Prénom: Votre prénom...

* Nom: Votre nom...

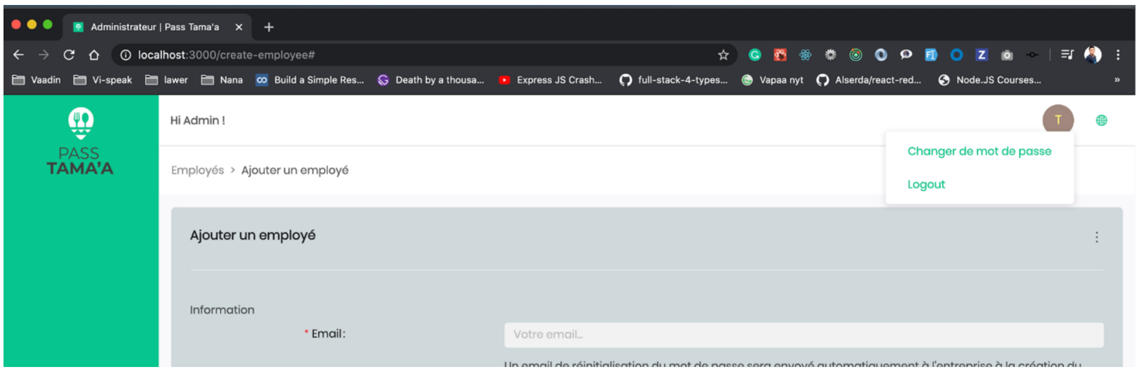
* Numéro de téléphone: +689 Numéro de téléphone

Conditions de crédit

Conditions d'utilisation

Personnaliser

Figure 42 Create new employee with some fields


Hi Admin !

Changer de mot de passe

Logout

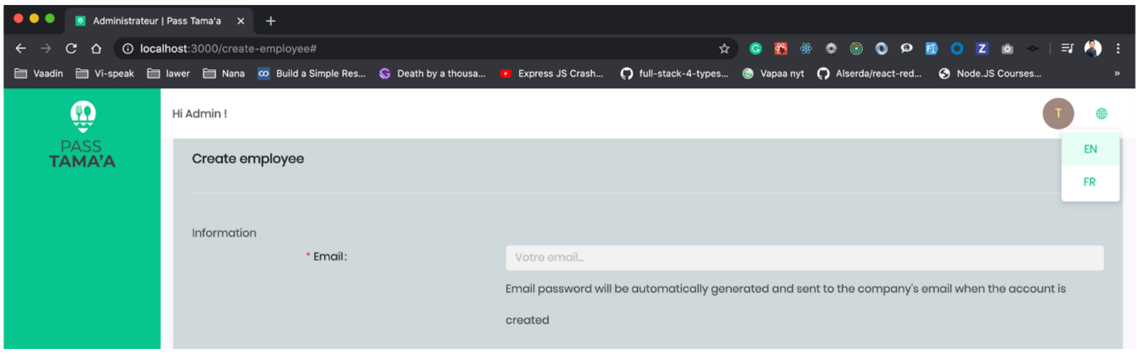
Employés > Ajouter un employé

Ajouter un employé

Information

* Email: Votre email...

Figure 43 Change password and logout


Hi Admin !

EN

FR

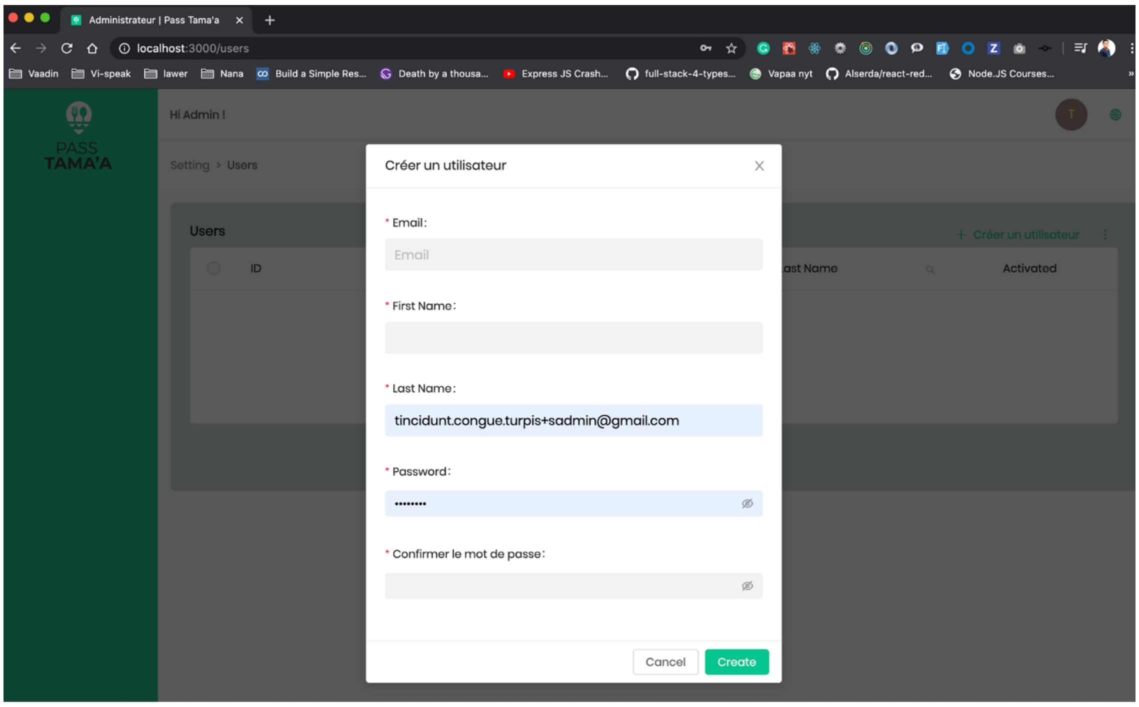
Create employee

Information

* Email: Votre email...

Email password will be automatically generated and sent to the company's email when the account is created

Figure 44 Change Language


Créer un utilisateur

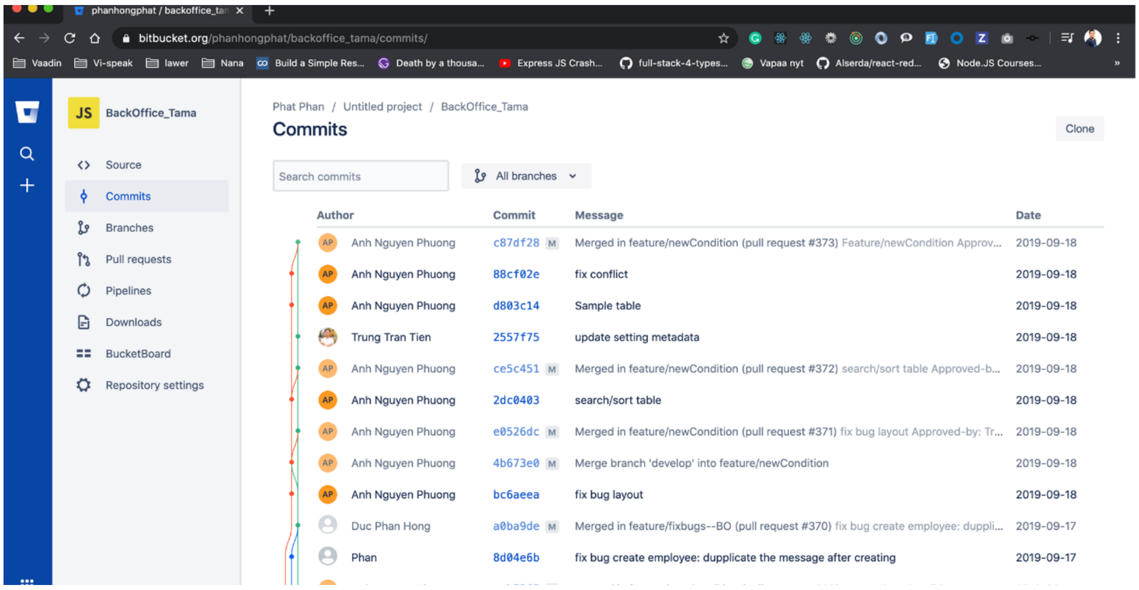
* Email:
Email
* First Name:
* Last Name:
tincidunt.congue.turpis+sadmin@gmail.com
* Password:
••••••••
* Confirmer le mot de passe:

Cancel Create

Figure 45 Create new user

Source code

The project developed and pushed to Github. There is the link: https://bitbucket.org/phanhongduc/backoffice_tama/src/develop/

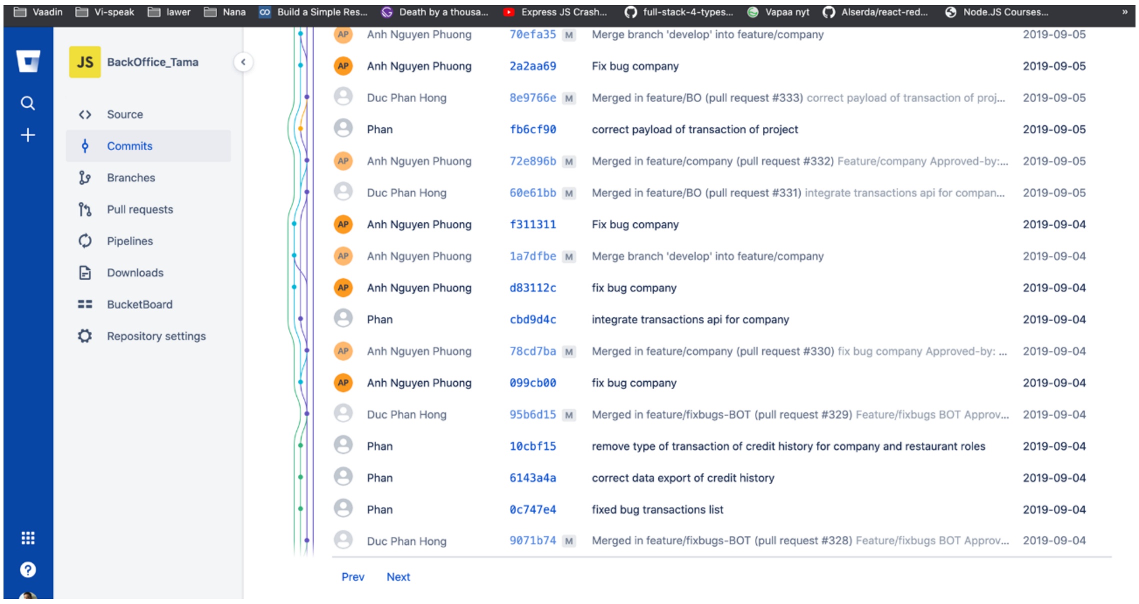


Phat Phan / Untitled project / BackOffice_Tama

Commits

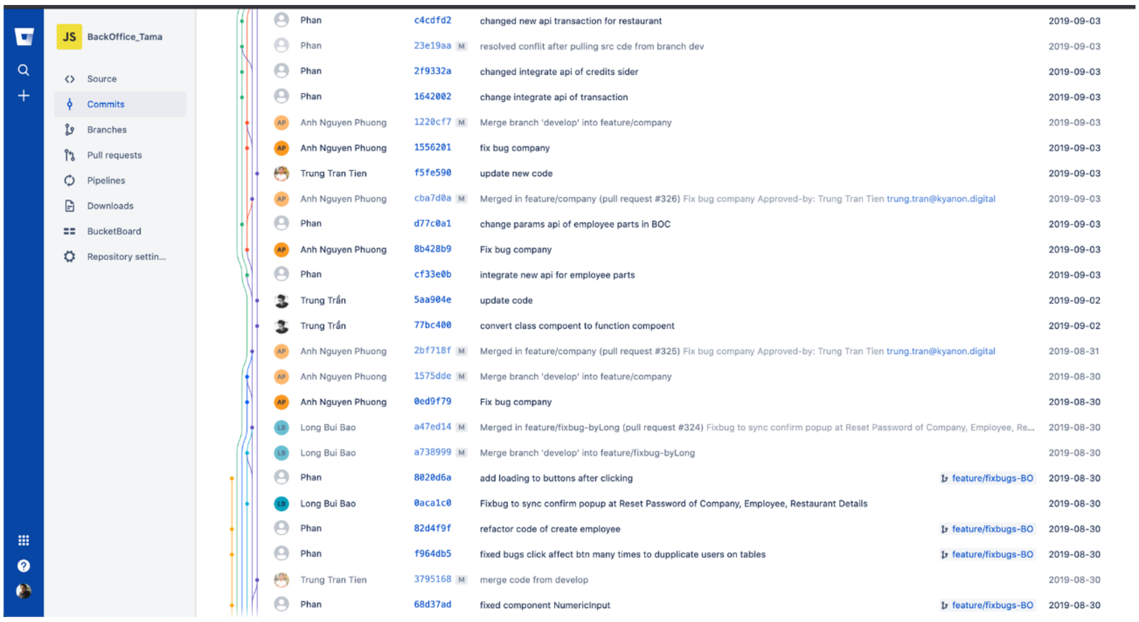
Author	Commit	Message	Date
Anh Nguyen Phuong	c87df28 M	Merged in feature/newCondition (pull request #373) Feature/newCondition Approv...	2019-09-18
Anh Nguyen Phuong	88cf02e	fix conflict	2019-09-18
Anh Nguyen Phuong	d803c14	Sample table	2019-09-18
Trung Tran Tien	2557f75	update setting metadata	2019-09-18
Anh Nguyen Phuong	ce5c451 M	Merged in feature/newCondition (pull request #372) search/sort table Approved-b...	2019-09-18
Anh Nguyen Phuong	2dc0403	search/sort table	2019-09-18
Anh Nguyen Phuong	e0526dc M	Merged in feature/newCondition (pull request #371) fix bug layout Approved-by: Tr...	2019-09-18
Anh Nguyen Phuong	4b673e0 M	Merge branch 'develop' into feature/newCondition	2019-09-18
Anh Nguyen Phuong	bc6aeea	fix bug layout	2019-09-18
Duc Phan Hong	a0ba9de M	Merged in feature/fixbugs--BO (pull request #370) fix bug create employee: duppli...	2019-09-17
Phan	8d04e6b	fix bug create employee: dupplicate the message after creating	2019-09-17

Figure 46 Commits around the september 2019



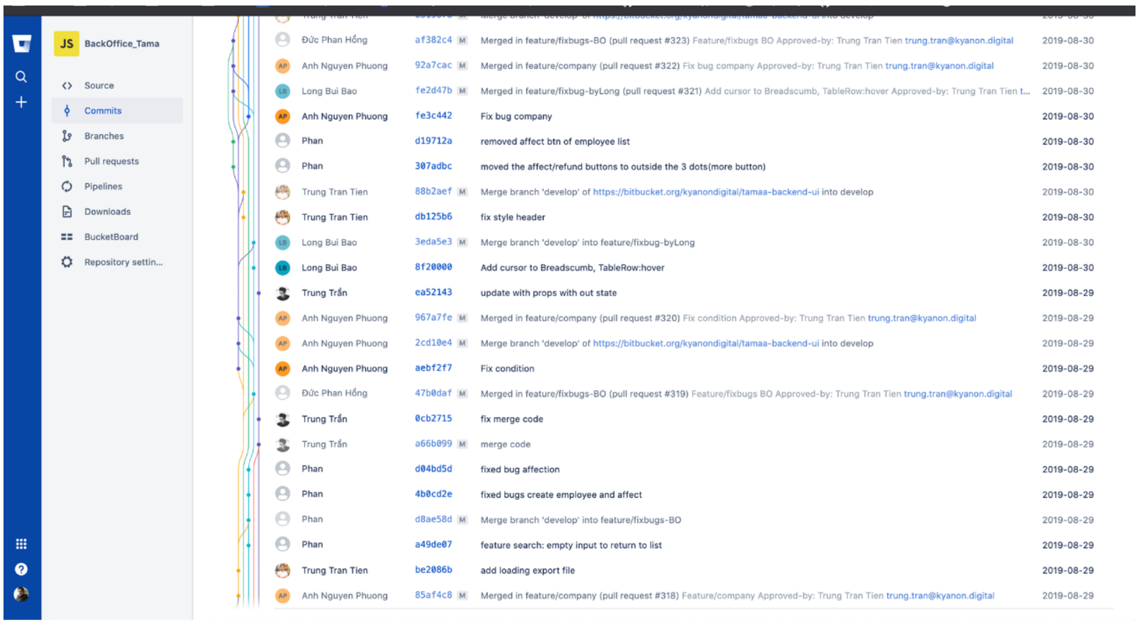
Author	Commit	Message	Date
Anh Nguyen Phuong	70efa35 M	Merge branch 'develop' into feature/company	2019-09-05
Anh Nguyen Phuong	2a2aa69	Fix bug company	2019-09-05
Duc Phan Hong	8e9766e M	Merged in feature/BO (pull request #333) correct payload of transaction of proj...	2019-09-05
Phan	fb6cf90	correct payload of transaction of project	2019-09-05
Anh Nguyen Phuong	72e896b M	Merged in feature/company (pull request #332) Feature/company Approved-by:...	2019-09-05
Duc Phan Hong	60e61bb M	Merged in feature/BO (pull request #331) integrate transactions api for compan...	2019-09-05
Anh Nguyen Phuong	f311311	Fix bug company	2019-09-04
Anh Nguyen Phuong	1a7dfbe M	Merge branch 'develop' into feature/company	2019-09-04
Anh Nguyen Phuong	d83112c	fix bug company	2019-09-04
Phan	cbd9d4c	integrate transactions api for company	2019-09-04
Anh Nguyen Phuong	78cd7ba M	Merged in feature/company (pull request #330) fix bug company Approved-by: ...	2019-09-04
Anh Nguyen Phuong	099cb00	fix bug company	2019-09-04
Duc Phan Hong	95b6d15 M	Merged in feature/fixbugs-BOT (pull request #329) Feature/fixbugs BOT Approv...	2019-09-04
Phan	10cbf15	remove type of transaction of credit history for company and restaurant roles	2019-09-04
Phan	6143a4a	correct data export of credit history	2019-09-04
Phan	0c747e4	fixed bug transactions list	2019-09-04
Duc Phan Hong	9071b74 M	Merged in feature/fixbugs-BOT (pull request #328) Feature/fixbugs BOT Approv...	2019-09-04

Figure 47 Commit around 9.2019



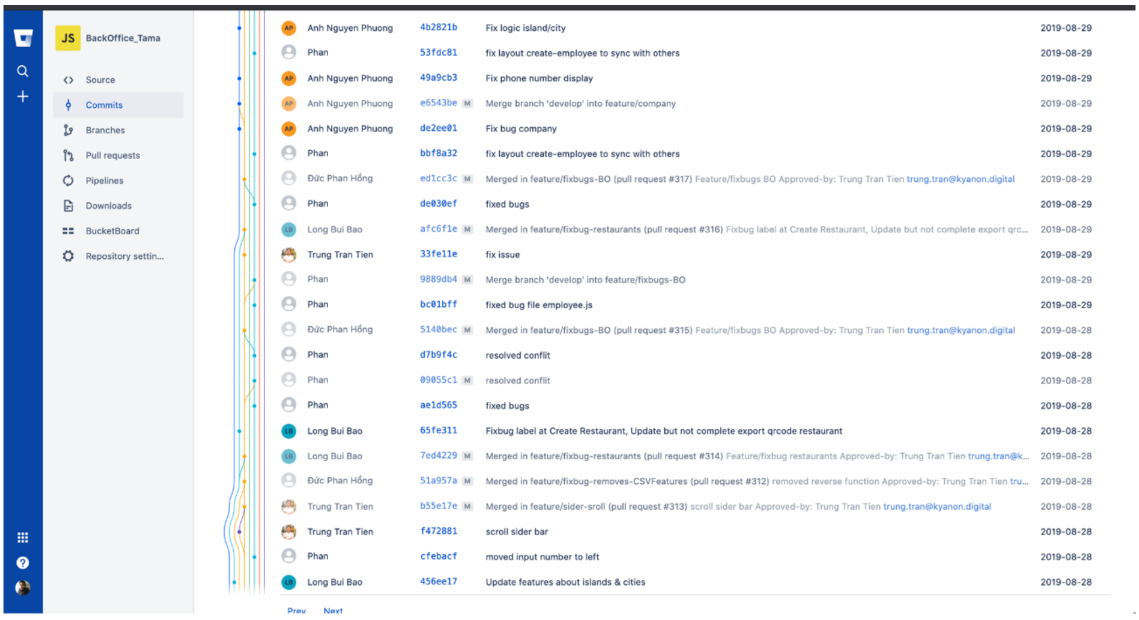
Phan	c4cdfd2	changed new api transaction for restaurant		2019-09-03
Phan	23e19aa M	resolved conflit after pulling src cde from branch dev		2019-09-03
Phan	2f9332a	changed integrate api of credits sider		2019-09-03
Phan	1642002	change integrate api of transaction		2019-09-03
Anh Nguyen Phuong	1220cf7 M	Merge branch 'develop' into feature/company		2019-09-03
Anh Nguyen Phuong	1556201	fix bug company		2019-09-03
Trung Tran Tien	f5fe590	update new code		2019-09-03
Anh Nguyen Phuong	cba7d0a M	Merged in feature/company (pull request #326) Fix bug company Approved-by: Trung Tran Tien trung.tran@kyanon.digital		2019-09-03
Phan	d77c0a1	change params api of employee parts in BOC		2019-09-03
Anh Nguyen Phuong	8b428b9	Fix bug company		2019-09-03
Phan	cf33e0b	integrate new api for employee parts		2019-09-03
Trung Trần	5aa904e	update code		2019-09-02
Trung Trần	77bc400	convert class compoent to function compoent		2019-09-02
Anh Nguyen Phuong	2bf718f M	Merged in feature/company (pull request #325) Fix bug company Approved-by: Trung Tran Tien trung.tran@kyanon.digital		2019-08-31
Anh Nguyen Phuong	1575dde M	Merge branch 'develop' into feature/company		2019-08-30
Anh Nguyen Phuong	0ed9f79	Fix bug company		2019-08-30
Long Bui Bao	a47ed14 M	Merged in feature/fixbug-byLong (pull request #324) Fixbug to sync confirm popup at Reset Password of Company, Employee, Re...		2019-08-30
Long Bui Bao	a738999 M	Merge branch 'develop' into feature/fixbug-byLong		2019-08-30
Phan	8020d6a	add loading to buttons after clicking	feature/fixbugs-BO	2019-08-30
Long Bui Bao	0aca1c0	Fixbug to sync confirm popup at Reset Password of Company, Employee, Restaurant Details		2019-08-30
Phan	82d4f9f	refactor code of create employee	feature/fixbugs-BO	2019-08-30
Phan	f964db5	fixed bugs click affect btn many times to dupplicate users on tables	feature/fixbugs-BO	2019-08-30
Trung Tran Tien	3795168 M	merge code from develop		2019-08-30
Phan	68d37ad	fixed component NumericInput	feature/fixbugs-BO	2019-08-30

Figure 48 Commit in 9.2019



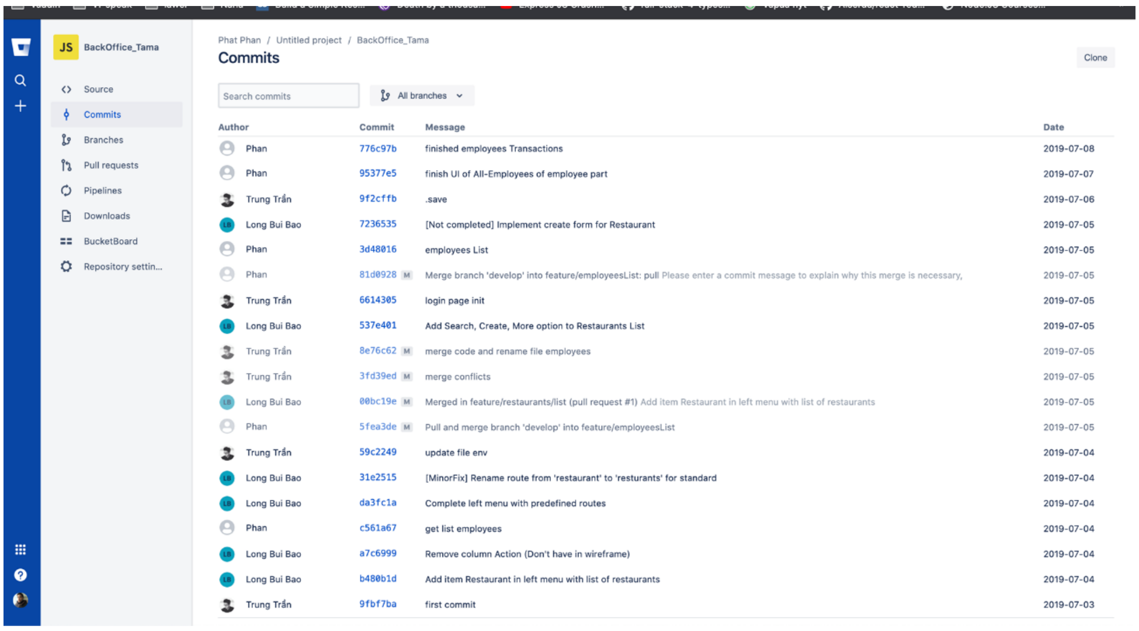
Đức Phan Hồng	af382c4 M	Merged in feature/fixbugs-BO (pull request #323) Feature/fixbugs BO Approved-by: Trung Tran Tien trung.tran@kyanon.digital	2019-08-30
Anh Nguyen Phuong	92a7cac M	Merged in feature/company (pull request #322) Fix bug company Approved-by: Trung Tran Tien trung.tran@kyanon.digital	2019-08-30
Long Bui Bao	fe2d47b M	Merged in feature/fixbug-byLong (pull request #321) Add cursor to Breadscumb, TableRow:hover Approved-by: Trung Tran Tien t...	2019-08-30
Anh Nguyen Phuong	fe3c442	Fix bug company	2019-08-30
Phan	d19712a	removed affect btn of employee list	2019-08-30
Phan	307adbc	moved the affect/refund buttons to outside the 3 dots(more button)	2019-08-30
Trung Tran Tien	88b2aef M	Merge branch 'develop' of https://bitbucket.org/kyanondigital/tamaa-backend-ui into develop	2019-08-30
Trung Tran Tien	db125b6	fix style header	2019-08-30
Long Bui Bao	3eda5e3 M	Merge branch 'develop' into feature/fixbug-byLong	2019-08-30
Long Bui Bao	8f20000	Add cursor to Breadscumb, TableRow:hover	2019-08-30
Trung Trần	ea52143	update with props with out state	2019-08-29
Anh Nguyen Phuong	967a7fe M	Merged in feature/company (pull request #320) Fix condition Approved-by: Trung Tran Tien trung.tran@kyanon.digital	2019-08-29
Anh Nguyen Phuong	2cd10e4 M	Merge branch 'develop' of https://bitbucket.org/kyanondigital/tamaa-backend-ui into develop	2019-08-29
Anh Nguyen Phuong	aebf2f7	Fix condition	2019-08-29
Đức Phan Hồng	47b0daf M	Merged in feature/fixbugs-BO (pull request #319) Feature/fixbugs BO Approved-by: Trung Tran Tien trung.tran@kyanon.digital	2019-08-29
Trung Trần	0cb2715	fix merge code	2019-08-29
Trung Trần	a66b099 M	merge code	2019-08-29
Phan	d04bd5d	fixed bug affection	2019-08-29
Phan	4b0cd2e	fixed bugs create employee and affect	2019-08-29
Phan	d8ae58d M	Merge branch 'develop' into feature/fixbugs-BO	2019-08-29
Phan	a49de07	feature search: empty input to return to list	2019-08-29
Trung Tran Tien	be2086b	add loading export file	2019-08-29
Anh Nguyen Phuong	85af4c8 M	Merged in feature/company (pull request #318) Feature/company Approved-by: Trung Tran Tien trung.tran@kyanon.digital	2019-08-29

Figure 49 Commit in 8.2019



Anh Nguyen Phuong	4b2821b	Fix logic island/city	2019-08-29
Phan	53fdc81	fix layout create-employee to sync with others	2019-08-29
Anh Nguyen Phuong	49a9cb3	Fix phone number display	2019-08-29
Anh Nguyen Phuong	e6543be M	Merge branch 'develop' into feature/company	2019-08-29
Anh Nguyen Phuong	de2ee01	Fix bug company	2019-08-29
Phan	bbf8a32	fix layout create-employee to sync with others	2019-08-29
Đức Phan Hồng	ed1cc3c M	Merged in feature/fixbugs-BO (pull request #317) Feature/fixbugs BO Approved-by: Trung Tran Tien trung.tran@kyanon.digital	2019-08-29
Phan	de030ef	fixed bugs	2019-08-29
Long Bui Bao	afc6f1e M	Merged in feature/fixbug-restaurants (pull request #316) Fixbug label at Create Restaurant, Update but not complete export qrc...	2019-08-29
Trung Tran Tien	33fe11e	fix issue	2019-08-29
Phan	9889db4 M	Merge branch 'develop' into feature/fixbugs-BO	2019-08-29
Phan	bc01bff	fixed bug file employee.js	2019-08-29
Đức Phan Hồng	5140bec M	Merged in feature/fixbugs-BO (pull request #315) Feature/fixbugs BO Approved-by: Trung Tran Tien trung.tran@kyanon.digital	2019-08-28
Phan	d7b9f4c	resolved conflit	2019-08-28
Phan	09055c1 M	resolved conflit	2019-08-28
Phan	ae1d565	fixed bugs	2019-08-28
Long Bui Bao	65fe311	Fixbug label at Create Restaurant, Update but not complete export qrcode restaurant	2019-08-28
Long Bui Bao	7ed4229 M	Merged in feature/fixbug-restaurants (pull request #314) Feature/fixbug restaurants Approved-by: Trung Tran Tien trung.tran@k...	2019-08-28
Đức Phan Hồng	51a957a M	Merged in feature/fixbug-removes-CSVFeatures (pull request #312) removed reverse function Approved-by: Trung Tran Tien tru...	2019-08-28
Trung Tran Tien	b55e17e M	Merged in feature/sider-sroll (pull request #313) scroll sider bar Approved-by: Trung Tran Tien trung.tran@kyanon.digital	2019-08-28
Trung Tran Tien	f472881	scroll sider bar	2019-08-28
Phan	cfebacf	moved input number to left	2019-08-28
Long Bui Bao	456ee17	Update features about islands & cities	2019-08-28

Figure 50 Commit in 8.2019



Commits

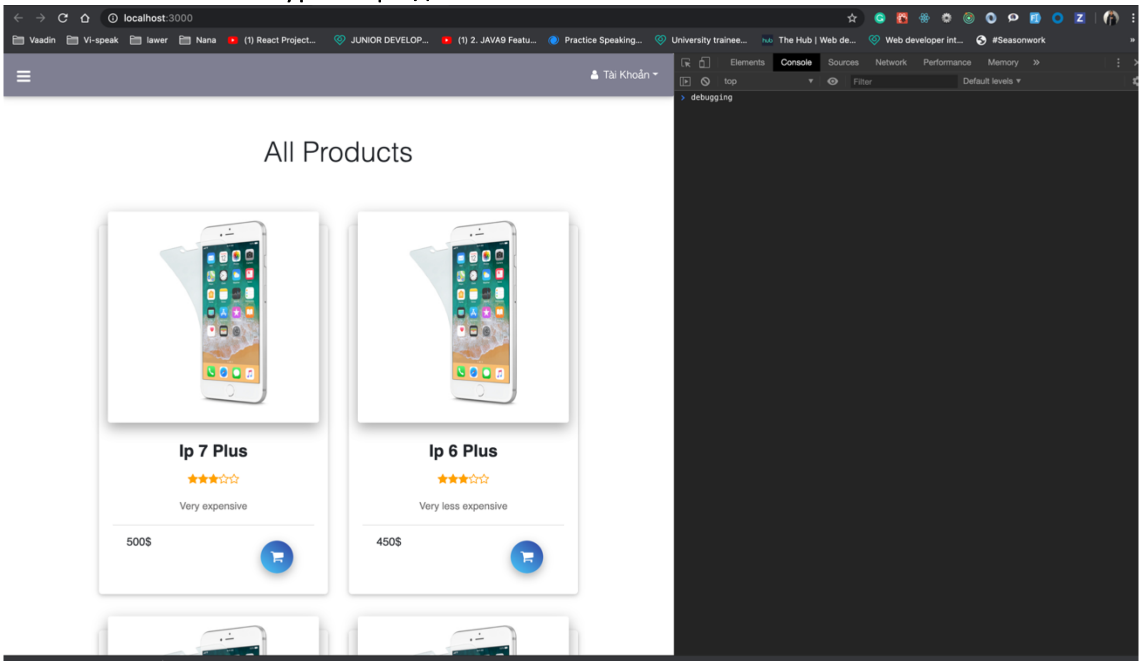
Author	Commit	Message	Date
Phan	776c97b	finished employees Transactions	2019-07-08
Phan	95377e5	finish UI of All-Employees of employee part	2019-07-07
Trung Trần	9f2cffb	.save	2019-07-06
Long Bui Bao	7236535	[Not completed] Implement create form for Restaurant	2019-07-05
Phan	3d48016	employees List	2019-07-05
Phan	81d0928 M	Merge branch 'develop' into feature/employeesList: pull Please enter a commit message to explain why this merge is necessary,	2019-07-05
Trung Trần	6614305	login page init	2019-07-05
Long Bui Bao	537e401	Add Search, Create, More option to Restaurants List	2019-07-05
Trung Trần	8e76c62 M	merge code and rename file employees	2019-07-05
Trung Trần	3fd39ed M	merge conflicts	2019-07-05
Long Bui Bao	00bc19e M	Merged in feature/restaurants/list (pull request #1) Add item Restaurant in left menu with list of restaurants	2019-07-05
Phan	5fea3de M	Pull and merge branch 'develop' into feature/employeesList	2019-07-05
Trung Trần	59c2249	update file env	2019-07-04
Long Bui Bao	31e2515	[MinorFix] Rename route from 'restaurant' to 'resturants' for standard	2019-07-04
Long Bui Bao	da3fc1a	Complete left menu with predefined routes	2019-07-04
Phan	c561a67	get list employees	2019-07-04
Long Bui Bao	a7c6999	Remove column Action (Don't have in wireframe)	2019-07-04
Long Bui Bao	b480b1d	Add item Restaurant in left menu with list of restaurants	2019-07-04
Trung Trần	9fbf7ba	first commit	2019-07-03

Figure 51 Commits from the first time-7.2019(the starting time of the Tama React Project)

6.3 Deployment

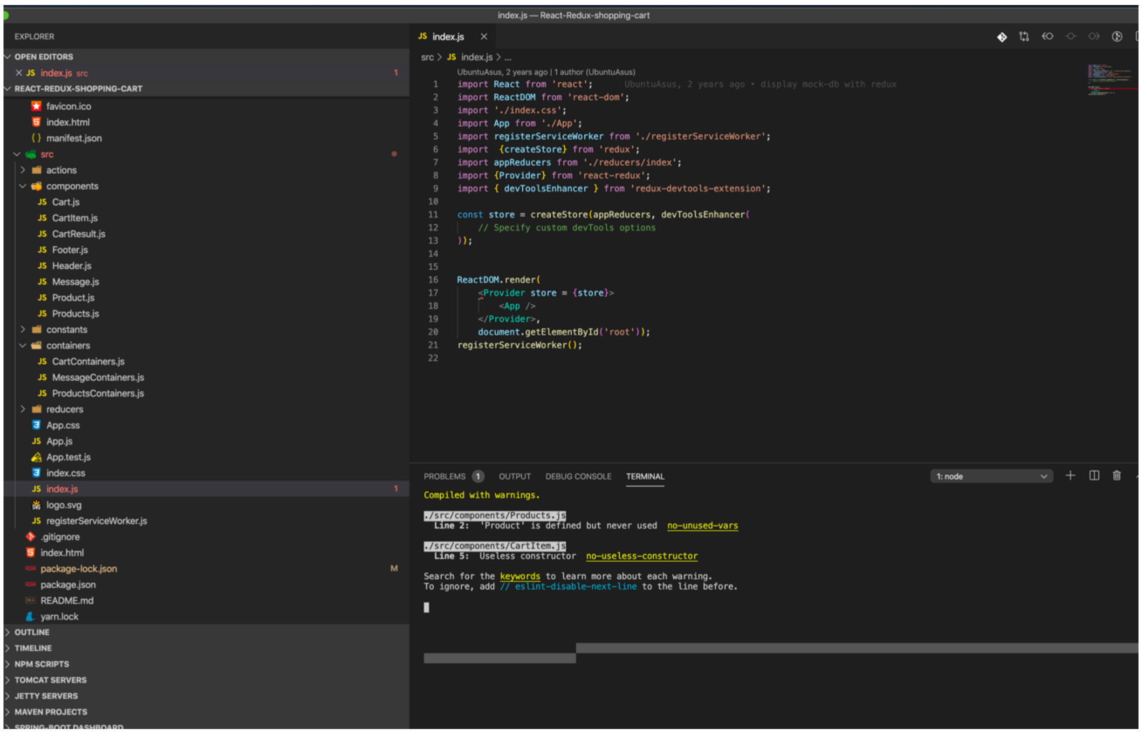
Running the application in the local server

After developing successfully and during developing it, the project always runs in local server for debugging and developing. This command runs of the project in local server is “npm start” is introduced in README.md. After running the command, the project is running and the requirement for seeing the project looks like is to open the browser, such as Chrome and type https://localhost: 3000.



```
> debugging
```

Figure 52. Localhost on Chrome and console for debugging.



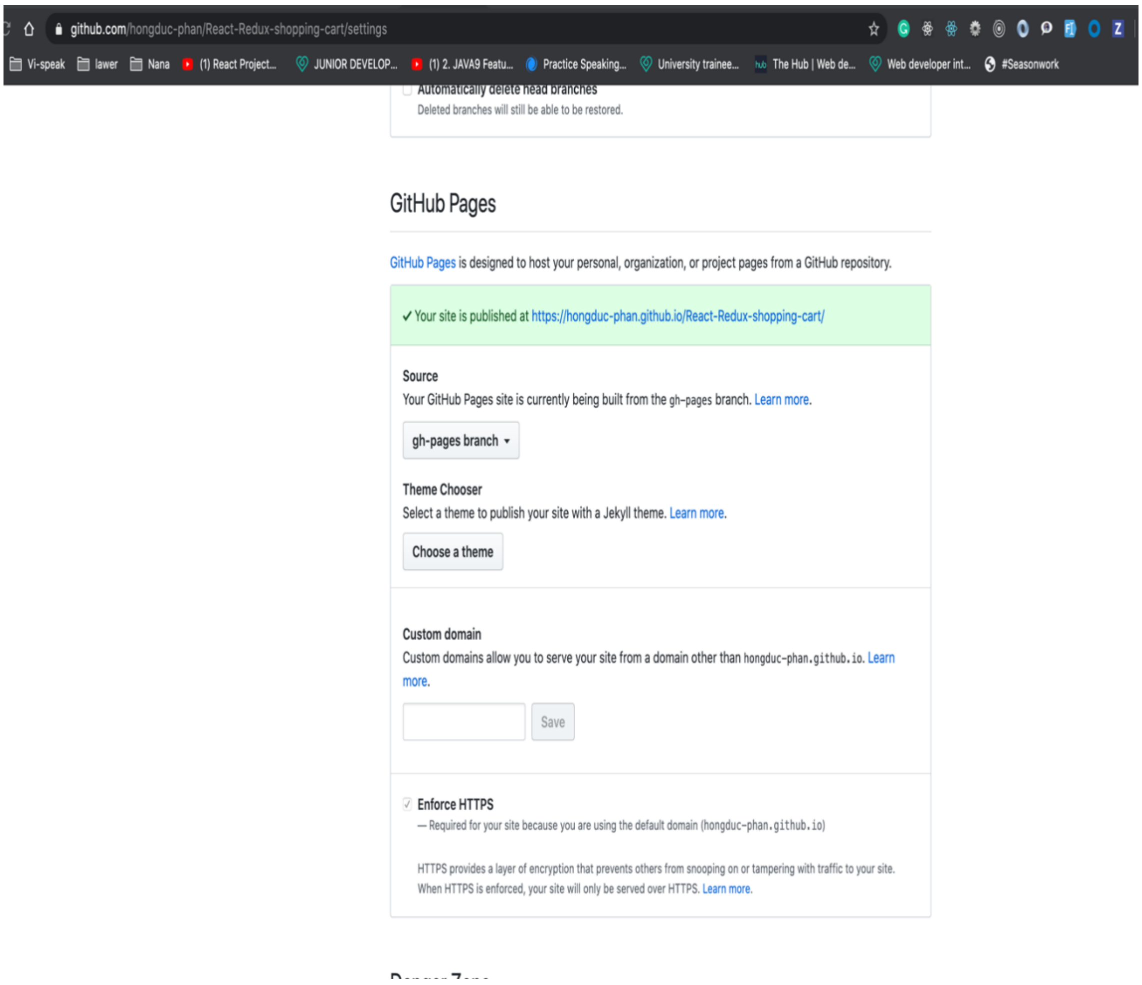
```
import React from 'react';
import ReactDOM from 'react-dom';
import './index.css';
import App from './App';
import registerServiceWorker from './registerServiceWorker';
import  {createStore} from 'redux';
import appReducers from './reducers/index';
import {Provider} from 'react-redux';
import { devToolsEnhancer } from 'redux-devtools-extension';

const store = createStore(appReducers, devToolsEnhancer(
    // Specify custom devTools options
));


ReactDOM.render(
    <Provider store = {store}>
        <App />
    </Provider>,
    document.getElementById('root'));
registerServiceWorker();
```

```
Compiled with warnings.

./src/components/Products.js
  Line 2:  'Product' is defined but never used  no-unused-vars

./src/components/CartItem.js
  Line 5:  Useless constructor  no-useless-constructor

Search for the keywords to learn more about each warning.
To ignore, add // eslint-disable-next-line to the line before.
```

Figure 53. Run localhost with terminal of Visual code.

Build the application for deployment

After developing successfully, the project is deployed in Github server. This is the link for deployment the project in Github: https://hongduc-phan.github.io/React-Redux-shopping-cart/



github.com/hongduc-phan/React-Redux-shopping-cart/settings

Automatically delete head branches

Deleted branches will still be able to be restored.

GitHub Pages

GitHub Pages is designed to host your personal, organization, or project pages from a GitHub repository.

✓ Your site is published at https://hongduc-phan.github.io/React-Redux-shopping-cart/

Source

Your GitHub Pages site is currently being built from the gh-pages branch. Learn more.

gh-pages branch

Theme Chooser

Select a theme to publish your site with a Jekyll theme. Learn more.

Choose a theme

Custom domain

Custom domains allow you to serve your site from a domain other than hongduc-phan.github.io. Learn more.

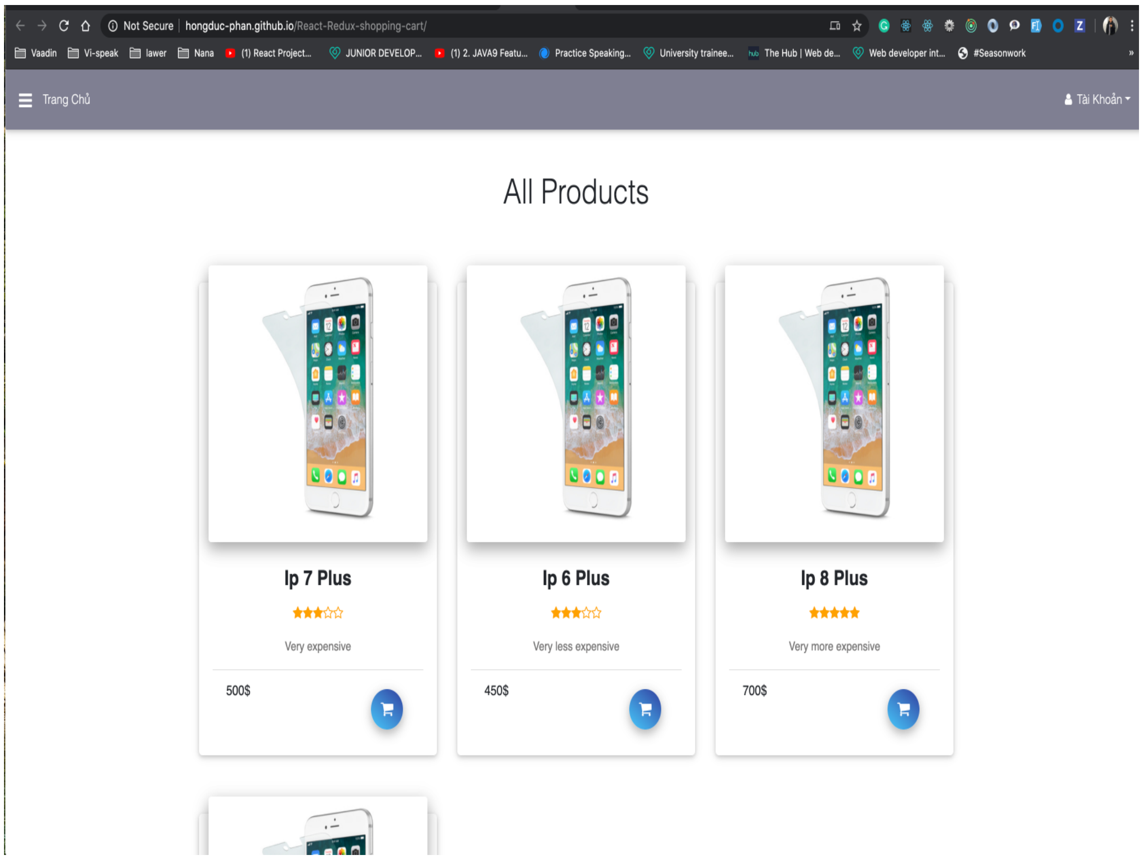
Save

✓ Enforce HTTPS

— Required for your site because you are using the default domain (hongduc-phan.github.io)

HTTPS provides a layer of encryption that prevents others from snooping on or tampering with traffic to your site. When HTTPS is enforced, your site will only be served over HTTPS. Learn more.

Figure 54. Github Pages for deploying in setting tab.



All Products

Ip 7 Plus
Very expensive
500$

Ip 6 Plus
Very less expensive
450$

Ip 8 Plus
Very more expensive
700$

Figure 55. Deployment project.

7 CONCLUSION

The main goal of the thesis was to study and use React Js Framework to implement React project Web App through participating the practical Tama React project and implementation the shopping cart application. Base on simple examples, theoretical concepts, and advanced concepts, the thesis shows anything from basic to advanced knowledge related to reactJS.
Even though React has not been popular in most of the web applications nowadays because in recent years, having many popular frameworks are released to compete with ReactJS as VueJS, Angular; Besides understanding both the basic and profound concepts of ReactJS is a treasure key to use that in other flatforms as React Native in mobile development.
React framework introduces a modern approach to develop web applications with single page application concept. The React is being supported and standardized and it is not an overstatement to conclude that the framework might be the future of web application development.

REFERENCES

[1] Facebook, "React," 2020. [Online]. Available: https://reactjs.org/docs/introducing-jsx.html. [Accessed 11 June 2019].

[2] Facebook, "React," Facebook, 2020. [Online]. Available: https://reactjs.org/docs/react-dom.html?fbclid=IwAR0VSqPAQDrIH2i66dLNeUNGG_nd18AsYndOUfc280QdaE-XXI--7C9HIu8. [Accessed 19 20 2019].

[3] M. Hamedani, "Programmingwithmosh," 2015. [Online]. Available: https://programmingwithmosh.com/javascript/react-lifecycle-methods/?fbclid=IwAR1xoJDi3mR-Td9sq9kkwHBW5rQzH_cchEV8G1kRiQETBdV5DT6Y9bbV5mw. [Accessed 12 7 2019].

[4] QED42, "QWD42," 2020. [Online]. Available: https://www.qed42.com/blog/code-splitting-in-react?fbclid=IwAR0LVnaptYEg_7-Ec3sEdqFOQV49phoAStJ7YtDMtY22RwQF_nqr8tWlwuU. [Accessed 12 6 2019].

[5] Facebook, "React," Facebook, 2020. [Online]. Available: https://reactjs.org/docs/context.html. [Accessed 11 June 2019].

[6] Facebook, "React," Facebook, 2020. [Online]. Available: https://reactjs.org/docs/forwarding-refs.html. [Accessed 13 June 2019].

[7] Facebook, "React," Facebook, 2020. [Online]. Available: https://reactjs.org/docs/hooks-state.html. [Accessed 19 July 2019].

[8] CBS, "CBS, cast," CBS, 2020. [Online]. Available: https://www.cbs.com/shows/swat/cast/215705/. [Accessed 3 August 2019].

[9] Insights, "Insights, innovatube," Insights, 07 March 2016. [Online]. Available: https://insights.innovatube.com/redux-th%E1%BA%ADt-l%C3%A0-%C4%91%C6%A1n-gi%E1%BA%A3n-ph%E1%BA%A7n-1-76a3fa2c31ab. [Accessed 19 August 2019].

[10] i. Insights, Insights, 07 March 2016. [Online]. Available: https://insights.innovatube.com/redux-th%E1%BA%ADt-l%C3%A0-%C4%91%C6%A1n-gi%E1%BA%A3n-ph%E1%BA%A7n-1-76a3fa2c31ab. [Accessed 19 August 2019].

[11] Viblo, "Viblo Asia," 2020. [Online]. Available: https://viblo.asia/p/redux-co-ban-m68Z00JdZkG?fbclid=IwAR3q7zySCM54vS395wfuLEk3VRVT71tv0EOE2ZwvpaO1SvAChy4jrhmAf6s. [Accessed 5 6 2019].

[12] Insight, "Insight," Insights, 07 March 2016. [Online]. Available: https://insights.innovatube.com/redux-th%E1%BA%ADt-l%C3%A0-%C4%91%C6%A1n-gi%E1%BA%A3n-ph%E1%BA%A7n-1-76a3fa2c31ab. [Accessed 19 August 2019].

[13] DevTo, "Devto," 2020. [Online]. Available: https://dev.to/oahehc/redux-data-flow-and-react-component-life-cycle-11n?fbclid=IwAR0wtIqCTWuQimDh1O0qlpZ-S4HJoPQ0cq_-Oix_zFg10N5SeDef_ltKwJ4. [Accessed 6 1 2020].