State Management of API Web Service using Redux on React Native App

Miftachudin, Muhammad Khoirul Hasin and Afif Zuhri Arfianto
Mechanical Engineering, Politeknik Perkapalan Negeri Surabaya, Surabaya, Indonesia

Keywords: Redux, React Native, State Management, API Web Service.

Abstract: There are two data types which control components in the React Native framework, props and state. React Native uses state and it often raises problems in client-side applications. The state changes of the enterprise application component leads to collisions between the states. To avoid these collisions, it is necessary to have a management state. Whereas, Redux is the most widely used state management for Javascript based applications, for instance; framework React Native. This study is to discuss comprehensive processes to make React Native-Redux application. The first step is to discuss some ways to make component in React Native. To obtain the data from API web service data sources, applying library third-party React Native is taken. Data which is resulted from the source stored in Redux state. Then it will be discussed deeply how to manage more than one state that is in a different component. The results of this study are in the form of an Android application using the Redux management state. The application runs normally when tested using an Android emulator from Android Studio. The data displayed matches the defined action type.

1 INTRODUCTION

There are two types which control the component at React Native framework, props and state. Props is set by immutable application. However, for data which has mutable value, React Native uses state (Facebook, 2019). Mostly, state produces problems for client-side application. The state changes at the enterprise application component lead to collision between states. To avoid the collision, it requires state management. The most state management used for Javascript- based application is Redux, for instance, React Native framework (B. Alex, et al, 2017).

In Github, the definition of Redux is a predictable state container (D. Abramov, 2019). This means that any change in state can be easily identified. Redux will first save the actions of states in one container so that they are informed to other states to adjust the order of the change reaction. For more details, the role of Redux in an application, namely:1) state changes will be made the time, place, and sequence should 2) state changes are only made by one person in charge (read: redux) 3) when there is a change, all states will receive information to adjust the reaction. 4) when there is an error in the state, Redux can immediately fix it (D. Abramov, 2015). In this study the data source used is from the web service API provided by Github. The format uses a shared data format in the form of JSON (JavaScript Object Notation).

There are several studies that have discussed Redux, those are: Matthias Kevin Caspers who explained the relationship of React (not React Native) and Redux, he explained only the elements used in Redux without any more comprehensive explanation on how to apply it in React (M. K. Casper, 2017). Wenhao Wu only explained the principle of Redux and compared it with other state management, namely Mobx (Wu, 2018). Mikael Nordström created a bug report tool to diagnose the state in the React Native third-party library (M. Nordström, 2018). However, this study discusses comprehensive some steps to create a Redux React Native application. First is to discuss how to create components in React Native. To retrieve data from a web service API data source, the next step is to implement a React Native third-party library. Data generated from these sources, stored in a state where the state is managed by Redux. Deeper, will be discussed how to manage more than one state that is in a different component.

Miftachudin, ., Khoirul Hasin, M. and Zuhri Arfianto, A.
State Management of API Web Service using Redux on React Native App.
DOI: 10.5220/0010966700003260
In *Proceedings of the 4th International Conference on Applied Science and Technology on Engineering Science (iCAST-ES 2021)*, pages 1425-1430
ISBN: 978-989-758-615-6; ISSN: 2975-8246

2 REACT NATIVE

React Native is a Javascript framework used to create mobile applications using web technology. This Facebook framework is a development of a web framework that was first released, React (without Native). This mobile framework makes it easy for developers who have been usually with web programming to build a mobile user interface. Unlike hybrid frameworks, such as Ionic, although it is Javascript, React Native is truly native in terms of user experience. To create a user interface, React Native has JSX components such as View, Text, and Image which are directly attached to the native platform. In addition, once developing an application based on React Native, the developer can simultaneously deploy it to two mobile operating systems, Android and iOS.

There are two ways to install React Native, Expo CLI and React Native CLI. In this study, the installation uses the React Native CLI method, using the Mac OS operating system with a choice of applications in the form of Android. To get started, Node must first be installed on the desktop. Like native applications, Android development requires an Android emulator found in Android Studio or an Android mobile phone to run the results of its development.

3 API WEB SERVICE

API (Application Programming Interface) web service is a group of protocols and standards that allow the exchange of data between systems or applications (RapidApi, 2019). Github provides API web service which is free used by anyone. Developers can directly interact by accessing the interface in the form of a URL with a basic URL https://api.github.com/. By adding the URL segment that Github has specified in its documentation (https://developer.github.com/v3/) (Github, 2019), the developer can obtain data in the form of JSON. To be able to access, the application in this study uses a third-party library for React Native that is axios.

4 JSON

JSON (JavaScript Object Notation) is small data exchange format, easy to read and write by humans, and easy to translate and generated by a computer (D. Crockford, 2019). This Javascript format allows communication between machines or applications in one standard. JSON structures can be objects (key and value pairs), arrays, and a combination of objects and arrays. The example can be seen in Table 1.

In this study, JSON is taken from the Github web service API at the GET /users / ${user_name}/ repos and GET/ repos/ $ {user_name} /$ {repo_name} endpoints. Each endpoint aims to get the entire data repository from a particular user and get one specific repository detail on that user. Each data display can be seen in Table 2 and Table 3 (Github, 2019).

Table 1: Example of JSON Structure.

Types	Structure	Remark
object	{"key0":0,"key1":1,"key2":2}	key0, key1, and key2 are key which have value 0, 1, and 2
array	[0,1,2]	It is like array structure in general
object + array	[{"key00":0,"key01":1,"key02":2}, {"key10":0,"key11":1, "key12":2}]	JSON consists of two objects, each object is in brackets separated by commas

Table 2: Illustration of Data Endpoint GET.

/USERS/${USER_NAME}/REPOS

GET ***/users/relferreira/repos***

```
[
  {
    "id": 63728367,
    "node_id":
"MDEwOlJlcG9zaXRvcnk2MzcyODM2Nw==",
    "name": "angular2-webpack-
starter",
   ……
   "owner": {
     "login": "relferreira",
     ……    },
    "html_url":
"https://github.com/relferreira/angu
lar2-webpack starter",
    ……
    "license": {
      "key": "mit",
      "name": "MIT License",
      ……    },
    "forks": 0,
    ……
  },
  {
    "id": 100611195,
    "node_id":
"MDEwOlJlcG9zaXRvcnkxMDA2MTExOTU=",
    "name": "apollo-server",
    ……
    "owner": {
      "login": "relferreira",
      ……
    },
```

```
    "html_url":
"https://github.com/relferreira/apol
lo-server",
    ……
    "license": {
      "key": "mit",
      "name": "MIT License",
      ……    },
    "forks": 0,
    ……
  },
……]
```


Table 3: Illustration of Data Endpoint GET. */repos/${user_name}/${repo_name}*

GET */repos/relferreira/angular2-webpack-starter*


```
{
  "id": 63728367,
  "node_id":
"MDEwOlJlcG9zaXRvcnk2MzcyODM2Nw==",
  "name": "angular2-webpack-
starter",
  ……
  "owner": {
    "login": "relferreira",
    ……
  },
  "html_url":
"https://github.com/relferreira/angu
lar2-webpack-starter",
  ……
  "license": {
    "key": "mit",
    "name": "MIT License",
    ……
  },
  "forks": 0,
  ……
  "parent": {
    "id": 31829770,
    "node_id":
"MDEwOlJlcG9zaXRvcnkzMTgyOTc3MA==",
    ……
    "owner": {
      "login": "PatrickJS",
      ……
    },
    "html_url":
"https://github.com/PatrickJS/angula
r-starter",
    "description": ":tada: An
Angular Starter kit featuring
Angular (Router, Http, Forms,
Services, Tests, E2E, Dev/Prod, HMR,
Async/Lazy Routes, AoT via ngc),
Karma, Protractor, Jasmine,
Istanbul, TypeScript, TsLint,
Codelyzer, Hot Module Replacement,
@types, and Webpack",
    ……
    "license": {
      "key": "mit",
      "name": "MIT License",
      ……
    },
    "forks": 5361,
    "open_issues": 83,
    ……
  },
  "source": {
    "id": 31829770,
    "node_id":
"MDEwOlJlcG9zaXRvcnkzMTgyOTc3MA==",
    "name": "angular-starter",
    ……
    },
    "html_url":
"https://github.com/PatrickJS/angula
r-starter",
    "description": ":tada: An
Angular Starter kit featuring
Angular (Router, Http, Forms,
Services, Tests, E2E, Dev/Prod, HMR,
Async/Lazy Routes, AoT via ngc),
Karma, Protractor, Jasmine,
Istanbul, TypeScript, TsLint,
Codelyzer, Hot Module Replacement,
@types, and Webpack",
    ……
    "license": {
      "key": "mit",
      "name": "MIT License",
      ……
    },
    "forks": 5361,
    "open_issues": 83,
    ……  },
  "network_count": 5361,
  "subscribers_count": 1
}
```


5 REDUX

Redux is a predictable container state for the Javascript app. Usually for state management, React uses Redux. Likewise the React family, which is used to develop mobile applications, namely: React Native, pairs Redux so that the statistics are well managed.

Some reasons that require using Redux include: 1) in the application a lot of data changes occur all the time 2) the developed application needs a place to store the entire state 3) top-level component is no longer sufficient to maintain the whole state.

The overall process of Redux management state with API web service is illustrated in Figure 1.

There are five main components in Redux according to the illustration, namely Actions, API, Store, Reducer, and View. Actions are functions that change state. Actions are Javascript objects so Actions must have a type. Because the state has value, in this study the value comes from the API (Github API web service) which is processed using the help of middlewares from axios.

The Dispatcher method sends Actions to the Store for safekeeping and provides information to other states that a new state will be processed. Reducer acts as a component that changes state to

new state. Reducer accepts two parameters, namely state and Actions from Dispatcher. Reducers process actions according to their type. Furthermore, in View, Redux is integrated so that the View component responds every time a state changes.

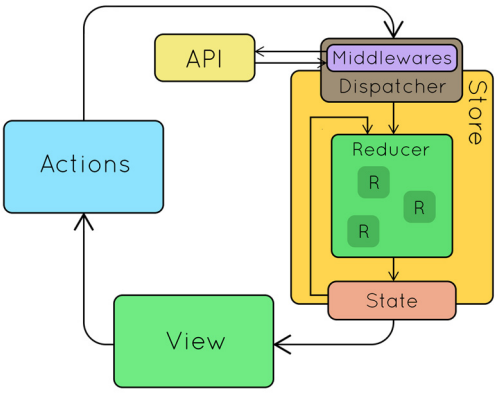

Figure 1: Five main components at Redux by illustration, Actions, API, Store , Reducer, and View.

6 EXPLANATION

This study does not discuss the details of React Native, so readers must ensure they understand React Native. After installing React Native, create a directory like in Table 4. This study creates two views stored in */app/pages*. In View RepoPage.js uses JSX <FlatList /> while RepoInfoPage.js uses only <Text></Text> to display the data. To connect both of them, this application uses react-navigation, when the user clicks the list of data displayed in RepoPage.js then the system goes to RepoInfo Page.js.

Before the Dispatcher method sends Actions and states to View, all Reducers in the / app / reducers folder are combined into one Reducer to become rootReducer in /app/reducers/index.js, then stored in the Store defined in /App.js and informs other states that there will be new states. Of course, each of these views has a Dispatcher method.

By using the Dispatcher method found in View, each Reducers accepts Actions and states. Each Actions is stored in/ app/ actions. While all types of actions are defined in the folder/ app constants. The State in Actions stores data that comes from API web service. The API is accessed using axios. In order for the axios to be integrated with Redux, in the Store axios middleware is applied. The next discussion will be detailed with one example on each component of Redux.

6.1 Type

There are three types of state in constants owned by RepoReducer, those are GET_REPOS, GET_ REPOS_SUCCESS, and GET_REPOS_FAIL.

```
/app/contants/repo.js
export const GET_REPOS = 'my-
awesome-app/repos/LOAD';
export const GET_REPOS_SUCCESS =
'my-awesome-app/repos/LOAD_SUCCESS';
export const GET_REPOS_FAIL = 'my-
awesome-app/repos/LOAD_FAIL';
```

6.2 Actions

A provided type is imported to Actions, but the type taken is only GET_REPOS type to request data from endpoint /user/${user}/repos. This action is defined in listRepos function.

```
/app/actions/RepoAction.js
import {GET_REPOS} from
'../constants/repo';
export function listRepos(user) {
    return {
      type: GET_REPOS,
      payload: {
        request: {
          url: `/users/${user}/repos`
        }
      }
    };
  }
```

6.3 Reducers

Reducers defined in the RepoReducer function have state and Actions in their parameters. Absolutely, each type that is imported has its own new state.

```
/app/reducers/RepoReducer.js
import {GET_REPOS,
GET_REPOS_SUCCESS, GET_REPOS_FAIL }
from '../constants/repo';
const initialState = {data: []};
export default function
RepoReducer(state = initialState,
action) {
    switch (action.type) {
      case GET_REPOS:
        return { ...state, loading:
true };
      case GET_REPOS_SUCCESS:
        return { ...state, loading:
false, data: action.payload.data };
      case GET_REPOS_FAIL:
        return { ...state, loading:
false, error: 'Error getting repos
info' };
      default:
        return state;
    }
  }
```

Reducers that have been made are combined into one in the combineReducers function. In this study, there are two Reducers namely: RepoReducer and RepoInfoReducer.

```
   /app/reducers/index.js
   import { combineReducers } from
'redux';
   import RepoReducer from
'./RepoReducer';
   import RepoInfoReducer from
'./RepoInfoReducer';

   export default combineReducers({
       RepoReducer, RepoInfoReducer
   });
```

6.4 Stores

Store defined by createStore function which has Reducers and Middleware parameter. Reducers imported into these parameters are Reducers that have been combined into one while Middleware imported is axios middleware which contains a root endpoint on github (https://api.github.com). Furthermore, the Store is imported in the Provider which acts as the only one that regulates the state in the application.

```
   /app/App.js
   import React, { Component } from
"react";
   import { View, Text, StyleSheet,
Image, TouchableOpacity, FlatList }
from "react-native";
   import {createAppContainer,
StackNavigator } from 'react-
navigation';
   import {createStackNavigator} from
'react-navigation-stack';
   import RepoPage from
'./app/pages/RepoPage';
   import RepoInfoPage from
'./app/pages/RepoInfoPage';

   import { createStore,
applyMiddleware } from 'redux';
   import { Provider, connect } from
'react-redux';
   import reducer  from
'./app/reducers';
   import axios from 'axios';
   import axiosMiddleware from 'redux-
axios-middleware';

   const client = axios.create({
     baseURL: 'https://api.github.com',
     responseType: 'json'
   });

   const store = createStore(reducer,
applyMiddleware(axiosMiddleware(client)
));

   const MainNavigator =
createStackNavigator({

     Repo: {screen: RepoPage,
navigationOptions: {
       header: null
       }
     },
     RepoInfo: {screen: RepoInfoPage,
navigationOptions: {
       header: null
       }
     }
   });
   const Navigation =
createAppContainer(MainNavigator);

   class App extends Component {
     render() {
       return (
         <Provider store={store}>
           <View
style={styles.container}>
             <Navigation />
           </View>
         </Provider>
       );
     }
   }

   // here cut code style

   export default App;
```

6.5 Views

Every time a change occurs, the state is mapped and dispatched with the Dispatcher method contained in View on mapStateToProps and mapDispatchTo Props, both of which are connected to Redux using the connect function.

```
   /app/pages/RepoPage.js
   import React, { Component } from
'react';
   import { View, Text, StyleSheet,
FlatList, TouchableOpacity } from
'react-native';
   import { connect } from 'react-
redux';
   import { listRepos } from
'../actions/RepoAction';

   class RepoPage extends Component{
       componentDidMount() {

this.props.listRepos('relferreira');
       }
       renderItem = ({ item }) => {
         const {navigate} =
this.props.navigation;
         return (
```

```
          <View style={styles.item}>
            <TouchableOpacity
onPress={() => navigate('RepoInfo',
{user: 'relferreira', repo:
item.name})}>
              <Text>{item.name}</Text>
            </TouchableOpacity>

          </View>
        );
      }
      render(){
          const { data, loading }  =
this.props;
          if(loading) return
(<View><Text>Loading...</Text></View>);
          return (
          <FlatList

styles={styles.container}
              data={data}
              keyExtractor={(item) =>
item.node_id}

renderItem={this.renderItem}
          />
          );
      }
  }
  // here cut code style

  const mapStateToProps = (state) => {
    return {
      data: state.RepoReducer.data,
      loading:
state.RepoReducer.loading
    };
  }
    const mapDispatchToProps = {
      listRepos
    };

  export default
connect(mapStateToProps,
mapDispatchToProps)(RepoPage);
```

7 CONCLUSION

The result of the study is Redux-Android Application. This application runs normally when tested using an Android emulator from Android Studio. When the data in the process of loading the Actions RepoAction.js retrieved is of type GET_REPOS, but when the data is successfully loaded Actions, the data taken are of type GET_REPOS_SUCCESS.

After all the data appears in the RepoPage.js View, the author clicks on one of the data repositories, the system transitions the page to RepoInfoPage.js and brings up the data in the RepoInfoAction.js Actions. It can be seen in figure 2.

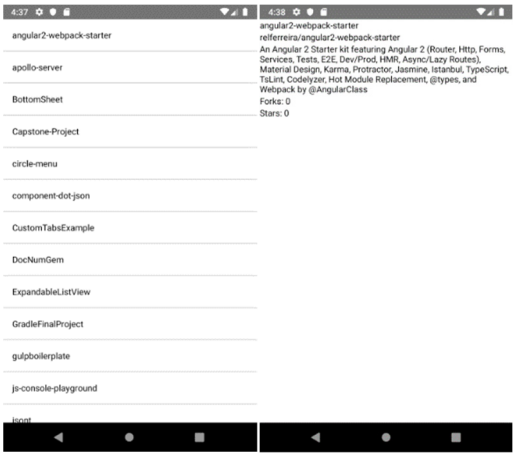

Figure 2: The left side is View from RepoPage.js while the right side is View from RepoInfoPage.js

REFERENCES

Facebook. (2019). React Native. Facebook, [Online]. Available: https://facebook.github.io/react-native/.

B. Alex and P. Eve, Learning React: Functional Web Development with React and Redux, California: O'Reilly, 2017.

D. Abramov. (2019). Redux. [Online]. Available: https://github.com/reduxjs/redux

D. Abramov. (2015). Redux. [Online]. Available: https://redux.js.org/.

M. K. Casper. (2017). React and Redux. Ausarbeitungen zum Seminar Rich Internet Applications w/HTML and Javascript.

W. Wu. (2018). React Native vs Flutter, cross-platform mobile application framework. *Metropolia University of Applied Sciences.*

M. Nordström (2018). Diagnostic tool for React Native – Reporting application state. Linköping University | Department of Computer and Information Science *Bachelor thesis, 16 ECTS* | Innovativ Programmering.

RapidApi. (2019). API vs Web Service: What's the Difference?. 14 February 2019. [Online]. Available: https://rapidapi.com/blog/api-vs-web-service/.

Github. (2019). Github Developer Guide Rest Api V3. Github, [Online]. Available: https://developer.github.com/v3/.

D. Crockford. (2019). Introduction JSON. [Online]. Available: https://www.json.org/json-en.html.

M. Zabriskie. (2019). Promise based HTTP client for the browser and node.js. [Online]. Available: https://github.com/axios/axios.

M. Eric and F. Jacob. (2017). Mastering React Native, Birmingham: Puck Publishing.