Honour Projects

# Online Shopping Application In React Native

Guchen Wang/100985902 - December 13, 2019

Retrieved from <http://thinkapps.com/blog/development/develop-for-ios-v-android-cross-platform-tools/>

# Abstract

After the Facebook introduce the React Native, it provides an easier way to create a cross-platform application compared with using an official native method. The project focuses on developing a mobile application in React Native. The React Native can develop the mobile application in Javascript, and will render a native component to construct the Interface conveniently. The most important is it doesn't need twice development for iOS and Android.

Meanwhile, with the fast development of mobile devices along with the shifting of shopping habit from in store to online, the objective of the project is to develop an online shopping app that can be used on multiple platforms using React Native.

# Acknowledgements

I acknowledge the project is indeed individual work and didn’t use any copyrighted materials and code. My application uses react-native-swiper which provide the components for the home screens to show the images. This component is released under an MIT license. Thanks my supervisor Louis Nel for helping me on the project.

# Table of contents

# List of figures

# List of tables

# Introduction

React Native is an open-source JavaScript framework, which can be used to develop cross-platform applications supporting both iOS and Android (Eisenman,2015). This means you can write codes easily and make it available simultaneously on iPhones. iPads, and Android Phones, and Android tablets without having to write it in two or more languages like Swift and Java. Unlike web apps that run in some adapter or translator, apps run nativity are faster and more reliable (Axelsson & Carlström, 2016). Instead of isolating developers based on the target platform, it will let developers iterate more quickly, and share knowledge and resources more effectively. React Native also applies layering and diff mechanism that can easily pass the json data through the mobile app and keep state out of the DOM. More specifically, the JavaScript layer passes native layer a json after diff, the data is then mapped by Native to the actual layout view. It's very friendly to developers considering most of the codes for the layout is JSX and all Native components are tagged. Additionally, React Native reuses the React framework, largely reducing the resources needed to build a new mobile application (Eisenman,2015). Thus, the learning cost is considered as low because any developer who knows how to write React code can now use the same skill set for web and apply it to iOS and Android.

For all those good sides of applying React Native, I become interested in building my own application that explores the cross-platform and efficient code sharing using existing JavaScript library. I choose to design an online shopping application utilizing React Native as my Honours Project. The online shopping application will include loading items lists, users sign in/sign up, and users' data interaction with database such as items added to the cart and items added to the Favourite list. The objective of the project is to research the basics of the React Native and try to develop a mobile application which can

transform data with the server and manage the state in React Native. In this report, I start with the motivation to explain why I undertake this project. Then I separately describe the details of the backend and frontend including the main basics of React Native, comparing the function component and class component, layout the screens by Flexbox, MobX and routing by react-navigation. The report finished by the conclusion about knowledge I studied in this project and my feeling of using React Native, good point and drawback of using React Native, and the future improvement for my application.

# Motivations

I first became aware of the React Native when I found the method about how to transfer a website in react to a mobile application and it has become more popular these years because there are lots of resources wasted on developing on both platforms in different frameworks and languages. My motivations mainly come from the following

The application using React Native can run on both iOS and android system without two separate codes, it will encourage code sharing and knowledge sharing. As a result, there is no isolation for developers on different platforms. This is what I'm delightful to see.  I studied the mobile application in the COMP 1601 course, from which it passed on knowledge about swift on iOS and Android Studio. It looks like I need a lot of time studying the basics and library to fully grasp these two platforms. I think React Native has the advantage on this problem. It can adapt common screen size and have good performance on both platforms.

In this project, I choose to develop an online shopping application demo to research the basics of React Native. An online shopping app will need a kind of layout of the screens, a feature of user login in and register, and management of the state for the cart of a user. So I can study how to layout for common size screens, transform data with server and manage the state in React Native. In addition, I want to know whether apps run natively are faster and more reliable than React Native.

# METHODOLOGY

## BACKEND

My primary choice of the backend is using express, a fast and lightweight framework for Node.js. One of the notable advantages for the Node.js is the scalable and real-time situations of increasing demand of servers by utilizing its unique I/O model (Hahn,2016). Hence, it's easy to use for small but robust server-side applications in this project. The structure of the backend is on the appendix ii.

**1. Mongoose**

MongoDB is a general purpose and document-based database for modern application, which meets all the needs for my mobile application. Mongoose provides such object modelling tool to easily model the app data for MongoDB. Mongoose defines a model starting with a schema, then it will map to a MongoDB collection within which it defines the shape of documents (Holmes,2013). In this application, I use it to construct a model of users. Every user has four objects: email, password, username and cartItem. For example, when the key 'email' with the type as 'String', if two schema Types for 'required' and 'unique' are set to true, email address will be essential and particular. Also, it will check if the input is in the correct format of email by using **SchemaType.prototype.validate().** This method can validate the value of the key and return the value back once the validation runs successfully. After setting the model, we need a MongoDB hosting to save our data. The one I use here is called mlab. It has a free option as a small hosting and is easy to connect by mongoose. I set the connection between the mongoose and the host in mongoose.js, export it, and let the main file 'server.js' require it to connect the mlab hosting.

**2. Routing**

Once it's connected to the mlab hosting and set the model of users, the next step is to set the app starting and listening to port 3000 for connections. I use Postman to test whether the connections and the routing is successful. Postman is a platform to monitor the response of the routes of server (). To verify if we can get the response from the server, we only need to put in the URLs and choose the route request. After it can successfully get the response from the server, I proceed with the setting of routing. Routing usually means how the application server responds to kinds of requests from clients. Routing defined by using the methods of the Express app object which correspond to HTTP methods. For example, 'app.get()' is used to handle the GET requests and app.post is used to handle the POST request. For these route methods, they include three main components: parameter, route paths, middleware function and handle function. Route path can let endpoint know which request will be made. The middleware functions are functions to determine the access to the request object, respond object or next middleware functions. For example, in the 'getUsername' routing method, I used a middleware function in it to check whether it has the token after login in before I get the username for current login user. The handle function is what you want to respond to you. It can let the server send you string or HTTP status code. The server of my application is used to handle the information of the users. Since the most important actions between my frontend and backend are register and login, it needs two routing methods: 'register' and 'login'. In the register, it captures the email, password and username from client request and uses these contents to create a new local 'User' object. Then, it saves the object to the database and gets corresponding HTTP status from the response. If the status code is 201, it means the register request is successfully done, otherwise, it is failed showing a code 400. [Figure.1]In the login method, after it gets the username and password from the client request, it will

use finOne method from the mongoose. That is, to find the data by the condition in the parameter. In my login function, it uses the username as a condition to find the correct user and send back code 201 so that the user will be allowed to log in. In addition, the login routing function can send back a token which is used for authentication. [Figure.2]After setting these two routing, the server can handle the main requests from client-side. In the server, it also has a middleware file that is 'authenticate.js'. *It is used to authenticate the token from which user for other routings that need the authentication of the client-side request. For example, in getUsername routing, it can get the username after authenticating if the user login in and which one is logged.*

```
router.post('/register', (req,res)=>{
    const {email, password, username} = req.body;
    const {item}="";
    let newUser = new User({
        email,
        password,
        username,
        item
    });

    newUser.save().then(user=>{
        if(!user){
            return res.status(400).send();
        }
        return res.status(201).send(user);

    }).catch(err=>{
        if(err){
            return res.status(400).send({error:err});
        }
        return res.status(400).send();
    });
});
```

Figure.1 - Register routing

```
router.post('/login',(req,res)=>{
    const{username,password} = req.body;

    User.findOne({username}).then((user)=>{
        if(!user){
            return res.status(400).send();
        }
        bcrypt.compare(password, user.password).then(match=>{
            if(!match){
                return res.status(401).send();
            }
            let token = jwt.sign({_id:user._id},'secret');
            return res.status(201).header('x-auth',token).send({token});
        }).catch(err=>{
            return res.status(401).send({error:err});
        })
    }).catch((err)=>{
        if(err){
            return res.status(401).send(err);
        }
        return res.status(401).send;
    })
})
```

Figure.2 - Login routing

## FRONTEND

The client-side focuses on building a react-native application that can run on both iOS and Android platforms. Main basic concepts of react-native include JSX, Component, State and Props. I will explain in detail as follows.

### 1. JSX and component

The ES6(ES2015) is supported in the react-native in a sense that all the stuff of ES6 is compatible to be used. For JSX, it's embedded XML within Javascript, which has the same function as HTML on the web. They both let us write the markup language inside the code, but the difference is the react uses <View> instead of the container  and  in HTML. Therefore, we can code the template and functions of the application easily in Javascript and render them to native components on interface UI.

Components are the basics of  React Native apps, and all of them need a render function which can return JSX to render. Anything we see on the screen are components, so we will build many components in a React Native app. In my application, it focuses on the components of screen pages and also common components that can be repeatedly used in multiple scenes. To create a component, react let us use function component and class component. I will talk about the differences between them after I explain the state and props.

**2. States and props**

We can customize the components even if it is created by ourselves. When we create customized components, different parameters are needed. The parameters in React Native are called props. For example, there is a component named Image in the JSX to show the picture. The props in the Image are the source to control the picture shows. If it is in one's own component, the props is also the reason that let a single component can be used in many other components. As shown in the below picture,

```
import React, { Component } from 'react';
import { Text, View } from 'react-native';

class Message extends Component {
  render() {
    return (
      <View style={{alignItems: 'center'}}>
        <Text>Text: {this.props.name}!</Text>
      </View>
    );
  }
}

export default class Messages extends Component {
  render() {
    return (
      <View style={{alignItems: 'center', top: 50}}>
        <Message name='the first message ' />
      </View>
    );
  }
}
```

Text: the first message !

Figure.3 - Demo of props

this.props.name is the props in this demo, we can reuse the function message in the ‘Messages’ component. Putting a name in the component is like putting a name as a parameter in demo function. However, the props can not change in all the lifetime of a component. If we have data to change, we need state.

The state will be initialized in the constructor generally. Whenever we want to change it, we need to call setState. The figure is the login component of my application. It set the state by the input from the user. The state in this component is username and password. After the component gets the change which means user input the text, the value of input will set to be the value of username and password through this.setState. When the setState is called, the component will be re-rendered.

### 3.Main application structure

I use the Expo CLI here to run my application directly on the simulator or Phones by Expo application. It is an official recommended way to test your react-native application quickly during development. The main structure of my application is shown in Appendix.ii.

The main application is in the ‘src’ folder while some components that can be reused in the main screens are in the common folder. For every screen page, I split the page to some small components for constructing an entire page and I put them into different folders for separate screen pages. The home folder has components to construct the home page. The category folder is the second page that can show all products according to the categories. The cart folder is a page showing the product list in the cart. Mine folder is used to show the information of logged users including the order lists. They are all in the main folder, among which some of them are class components and the other are function components.

### 3.1. Class component and function component

A function component is plain Javascript function that can get props as parameter and return JSX to native component. A class component is required to extend from React Native component and have a render function to return JSX. It has more benefits though it needs more codes in that the function component cannot initial a state and have setState method. The function components cannot be used when it needs its own state. The only way is to pass by the props from the parent component, but this can be a bit complicated also. Thus, when the component is without the state, it is better to use the function component. The advantage is less code and easy to read and write, in other words, better performance overall.

### 3.2 React Native-swiper

In many applications, especially shopping apps, there are always some images to advertise on the home page. React Native-swiper is the swiper component I used here to do some simple customizations. From the properties table, the swiper can be set to auto play the images and loop them. The images are set in a way it will be passed in as props. In my Homeswiper.js, the props are the image sources and the function can map them to show every picture. Sometimes we can use open source components which are easy to install the dependency. From the good example of using swiper components, the advantage of react-native components is obvious. Since the parameter can be passed by the props, native components can be reused in many senses. Additionally, the style and template of components can be flexibly set so that it is suitable to any size of screens.

### 3.3 Flexbox

Recently, different series of phones or different manufacturers' phones have different sizes of screens, such as 'all screen' from Apple and 'infinite screen' from Samsung. If we develop an application only for the iPhone in Swift, the process will be much easier because we only think about the application on one single platform. Since the goal of React Native application is one code for multiple platforms, React Native introduces Flexbox to adjust to different screen sizes, playing an important role in designing the layout of all screen pages in applications.

The flex is to make the items over fill the available space along the main axis. The property of flex can divide the space to corresponding size. Like the graph below


Figure.4 - Demo for flex with flexDirection: 'row'

The flex direction is used to set the layout direction of all children of the node. It sets the View container with flex equal to 1. Since there is no same level container , it means all the main axis is set to this container. In this View, there are three other containers with flex 1, 2, 3 in it, which represent 1/6 , 2/6, and 3/6 of the axis respectively. The 6 is from the total of 1, 2 and 3, and the axis is the row axis because the flex direction has been set to row. JustifyContent is used to align the children within the main axis. JustifyContent set to ‘flex-start’ means the children will be aligned from the start of the main axis. ‘Flex-end’ means the children will be aligned from the end of the main axis. The details of them can be seen on table.1. Similar to the ‘justifyContent’, alignItems is to align the children within the cross axis. See Figure.5 and Figure.6 for the demo of the four options of the alignItem. AlignSelf has the same options and effects as alignItems, the only difference is that AlignSelf can be set to one child to change the alignment within its parent. In some scenarios, the sizes of children overflow the size of a container. Thus, we need flexWrap to allow the items to be wrapped to several lines along the main axis.

| | |
|---|---|
| **flex-start** | Align children of a container to the start of the container's main axis. |
| **flex-end** | Align children of a container to the end of the container's main axis. |
| **center** | Align children of a container in the center of the container's main axis. |

| | |
|---|---|
| **space-between** | Even the space of children across the container's main axis, distributing remaining space between the children. |
| **space-around** | Even the space of children across the container's main axis, distributing remaining space around the children. Compared to space-between using space-around will result in space being distributed to the beginning of the first child and end of the last child |
| **space-evenly** | Evenly distributed within the alignment container along the main axis. The spacing between each pair of adjacent items, the main-start edge and the first item, and the main-end edge and the last item, are all exactly the same. |

Table.1 - The four options for justifyContent, alignContent

```
import React, { Component } from 'react';
import { View, Text } from 'react-native';

export default class FlexDirectionBasics extends Component {
  render() {
    return (
      // Try setting `alignItems` to 'flex-start', 'flex-end', 'center', 'stretch'
      <View style={styles.container}>
        <Text style={styles.headerStyle}>alignItems: 'center'</Text>
        <View style={[{flexDirection: 'column', alignItems: 'center'},
styles.elementsContainer]}>
          <View style={styles.powderblue} />
          <View style={styles.skyblue} />
          <View style={styles.steelblue} />
        </View>
      </View>
    );
  }
}

const styles = {
  steelblue : {
    backgroundColor: 'steelblue',
    height: 50,
    width: 100
  },
  skyblue : {
    backgroundColor: 'skyblue',
    height: 50,
    width: 100
  },
  powderblue : {
    backgroundColor: 'powderblue',
```

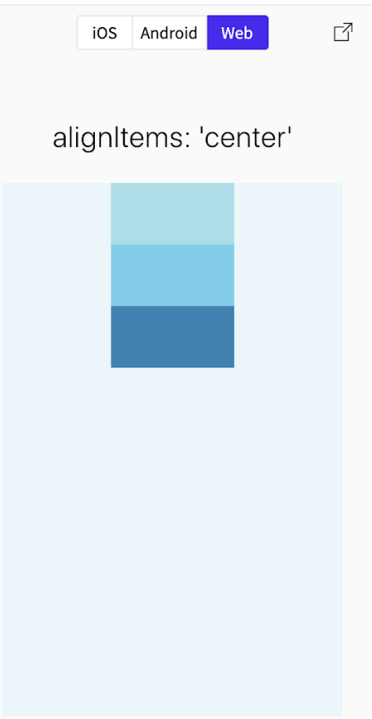


Figure.5 - demo for alignItems: ‘center’

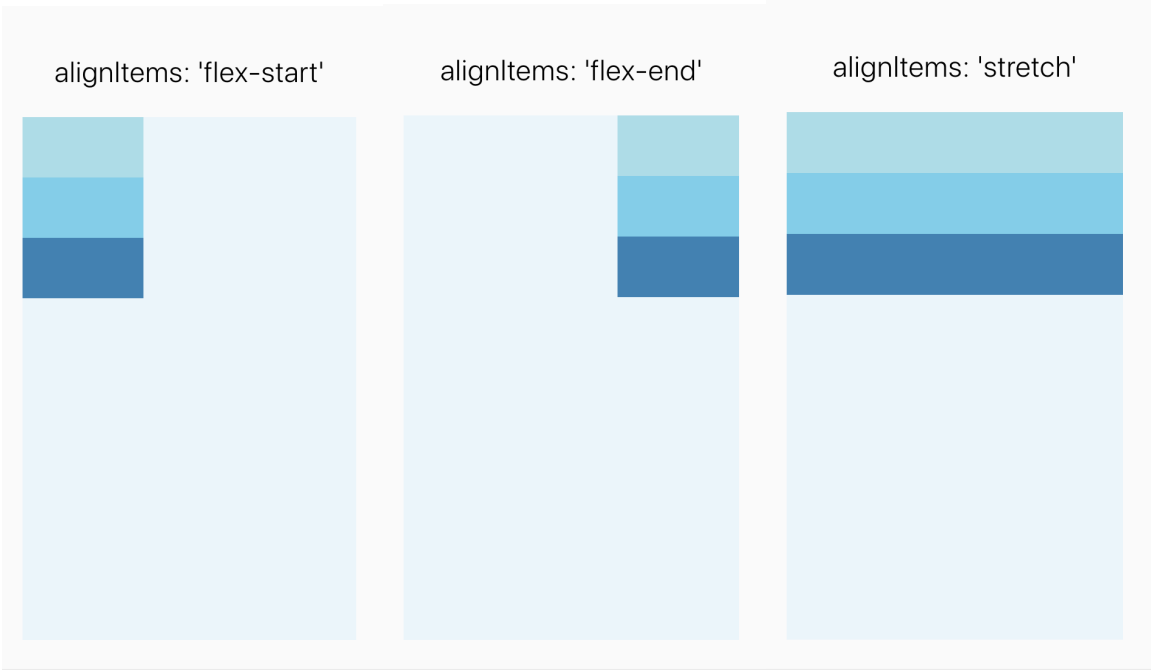


Figure.6 - demo for alignItems: ‘flex-start’, ‘flex-end’, ‘stretch’

### 3.4 React-ative-axios

Axios can send http requests from node.js environment or browser. It deals with XMLHttprequest and node's HTTP interface by providing a single API request.  It also intercepts the request and the response, transforms the data, and auto transforms JSON data. It is easy to add axios to react-native projects for accessing the common HTTP and HTTPS API.

In the login.js, using axios is to wrap a post request to the login routing of the server with the username and password. As I introduced in the backend section, the response from the server is a header named 'x-auth'. After sending the request, we can set the handle function in the '.then()'. In this function, I set the token to the header which bcrypt the information of the user and I save it in the asyncStorage. AsyncStorage is an asynchronous, key-value storage system. The item in the storage can be loaded globally. The 'asyncStorage.setItem' can persist an object with a key and use getItem to fetch the data. I save the token with key 'x-auth' in the asyncStorage, and when I want to authenticate the information from the user, the app will get those info by the key. [Figure.7] It is the same thing in the register.js as in login.js. It sends a HTTP request including an object of username, email and password to the routing for registration on the server.

```
47      login = () => {
48        const { username, password } = this.state;
49        const self = this;
50        if(username && password){
51          axios.post('http://localhost:3000/user/login',{
52            username,
53            password
54          }).then(response => {
55
56            try{
57              const token = response.headers['x-auth'];
58              if(token){
59                      AsyncStorage.setItem('x-auth',token).then(()=>{
60                          this.props.navigation.navigate('Home');
61               }).catch(()=>{
62                 alert('err');
63               })
64              }
65            }catch(err){
66               alert('error');
67            }
68
69          }).catch(()=>{
70              alert('Wrong username or password');
71          });
72        }else{
73          alert('username and password field are both required');
74        }
75      };
76    }
```

Figure.7 - login function using axios

### 3.5 MobX

MobX is a simple and scalable state management for react or React Native. It is based on transparently applying functional reactive programming. It optimizes the synchronization of application state and

react components. It can only update when strictly needed and keep the latest state by a reactive virtual dependency state graph.

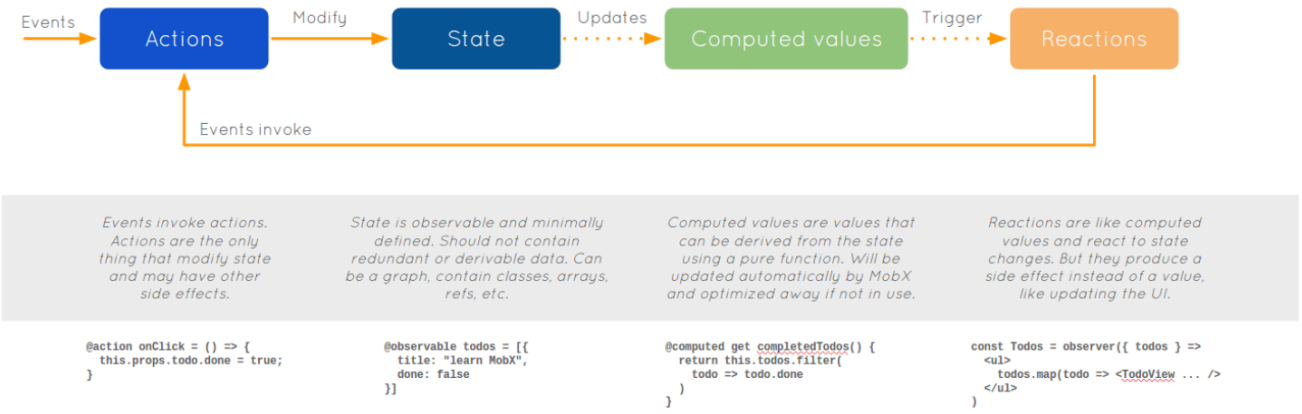


Figure.8 - The graph of the main progress of MobX<https://mobx.js.org/README.html>

First step is marking the data to be observed by MobX. Then create a view to respond to change in the state. The next thing is to modify the state, ‘actions’ is the only needed way to do this in MobX. After the state is modified, the computed value will then compute the relative values and update them. Additionally, the ‘reactions’ can give a reaction to the UI when state changes.

In my application, the index.js with a class RootStore in folder MobX is the root of the store system. I define a CartStore and OrderStore in the class. They are used to store the data of the items of cart list and order list. In the CartStore class, I define an observation for the itemList which is array, it is the data of the cart we want to observe in this application when users adding or removing items in the cartList. The delete is an action to remove the item in the cart list. Moreover, there are two computed values, one is to select all items and another one is to calculate the total price of the products adjusting to any items change in the cart list[Figure.9].

```
import CartStore from './CartStore';
import OrderStore from './OrderStore';

/**
 * @class RootStore
 */
class RootStore {
  constructor() {
    this.CartStore = new CartStore(this);
    this.OrderStore = new OrderStore(this);
  }
}

export default new RootStore();

```

Figure.9 - Capture of RootStore

In the component of the item detail screen, it has a function for adding this item to the cart list. The addCart function will call a updateCartScreen to update the data of cart list synchronously. When the component has picked up the store at the beginning, the item data will push to the array which is the observation in the CartStore mentioned above. Now the state of the cart list has changed globally. After adding item to the cart, the screen can be routed to the cart. In the cart.js, it has two computed value functions for updating the new state of CartStore and OrderStore. The reason is that the states of cart and order will change in this component. Users can change the value of items in cart, check out and pay to finalize the order. And in the checkout function, the data of new order will also push to the array of order in the store system and make the array in the CartStore clean.

### 3.6 React-Navigation

After finishing constructing all components, I am going to focus on connecting all the screens. React-Navigation is for routing and navigation in React Native. In the web browser, the url is pushing to the browser history stack when the user clicks the link. The url will pop from the stack and the browser will jump to it if the user wants to go back to the previous page. However, in React Native, there is no stack like this, so we need React navigation. Using a stack navigator can let app transit between screens and manage the history. Here are the basic procedures on how the react navigation works. The 'createStackNavigator' is a function to render a react component. We can put the screens as objects in the stack and configure the routes. We then pass this stack function as parameter to the app container which also exports a react component to the root component of the app.

In my application, onces the user opens the app, it will authenticate upon user login. Instead of a stack, I use a switchNavigator to switch between the login screens, authentication loading screens and main screens. The initial screen of it is an authentication loading screen which is used to check if the application has the token of authentication. If it has the token, it will either go to the main screens or to the login screen. Then I put the main screens in a bottom tab navigator, a tab bar at the bottom of the app's screen so that users can switch to main screens by using it. This bottom bar and other screens to which users need to go indirectly such as registration need to be put in a stack navigator. Finally, the stack navigator is like a container of all screens and I set the stack navigator to the switch navigator.

# Conclusion

React Native enables web developers to create robust mobile applications using their existing JavaScript knowledge. It offers faster mobile development and more efficient code sharing across iOS, Android, and the Web (Eisenman,2015). After developing my project application, I can essentially develop a mobile application and be experienced with major features of React Native. I came to know the principle of the react navigation for routing, managed the state by MobX and studied how to layout the screens for common screen sizes. In the future, improvements for the application will include saving the cart list to the users' database, and updating the cart when the state changes. I also want to research how the check out  process works.

All in all, the React Native is indeed a convenient way to develop a cross-platform mobile application. However, the project is relatively young and documentation still has room to improve, for example, no best practices has been developed and some features still aren't supported. The API of this framework changes every version. Therefore, some demos on the internet might become out of date and it's time consuming to verify and review all the implements. As part of the downsides of the new technology, React Native comes with uncertainty as to whether it's worth studying, when it will be out of support by Facebook, and how long it takes for the next new framework to show up and eventually replace it.

# Reference

Eisenman, B. (2015). Learning React Native: Building native mobile apps with JavaScript. " O'Reilly Media, Inc.".

Justyna, R. (2017, Feb 23). *When, How and Why use Node.js as Your Backend*. Retrieved from http://www.netguru.com

Axelsson, O., & Carlström, F. (2016). Evaluation targeting React Native in comparison to native mobile development.

Hahn, E. (2016). Express in Action: Writing, building, and testing Node. js applications. Manning Publications,.

Holmes, S. (2013). Mongoose for Application Development. Packt Publishing Ltd.

# Appendix.i.

Running instruction:

1.Start server:

In terminal:

i. cd ../OA/server

ii. node server.js

2.Start application

Mac OS:

i. Xcode->preference->Components->choose a simulator to install

In terminal:

ii. cd ../OA

iii. npm start

iv. choose run on iOS simulator

Windows:

i. install Android Studio

ii. install Android simulator

In terminal:

iii. cd ../OA

iv. choose run on android simulator

both can run on the Mobile phone if you scan the QR code in Expo App.

## Appendix.ii.

The structure of server-side and client-side

```
OA/
|---.expo
|---.expo-shared
|---assets
|---node_modules
|---server
|---src
|        |---common
|        |---img
|        |---main
|            |---Cart
|            |---Category
|            |---Home
|            |---ItemDetail
|            |---Login
|            |---Mine
|            |---Order
|        |---mobx
|        |---mock
|        |---config.h
|        |---navigation.js
|        |---Root.js
|---App.js
|---app.json
|---babel.config.js
|---package-lock.json
|---package.json
```

```
OA/server
|---db
|    |---models
|         |---user-model.js
|    |---mongoose.js
|---middleware
|    |---authenticate.js
|---node_modules
|---routes
|    |---addItem.js
|    |---getUsername.js
|    |---use-routes.js
|---config.js
|---package-lock.json
|---package.json
|---server.js
```