

Khuong Chu

IMPLEMENTING A WEB CLIENT USING REACT

IMPLEMENTING A WEB CLIENT USING REACT

Khuong Chu
Bachelor's Thesis
Spring 2019
Information Technology
Oulu University of Applied Sciences

ABSTRACT

Oulu University of Applied Sciences
Degree Programme in Information Technology

Author: Khuong Chu
Title of the bachelor's thesis: Implementing A Web Client Using React
Supervisor: Kari Laitinen
Term and year of completion: 2019 Number of pages: 32

In recent years, social networking has increasingly developed in the public. Facebook and Instagram are known as pioneering leaders in this trend. Therefore, the purpose of this thesis was to build a web application by using a new technology in web development. Additionally, this thesis is used to demonstrate the outcome of the author's learning process. The application is created to connect the users all around the world so that they can freely discuss their idols and comfortably share their stories.

The application is developed by using NodeJS for backend and React for frontend. However, the aim of this thesis focuses strongly on the frontend where the users will interact with the application. Additionally, MySQL is also used for storing database. The thesis will show how React communicates with the server REST API via HTTP. Generally, this thesis shows that React is a suitable choice and it works well for the purpose.

The crucial part for the whole project is its source code which is taken apart in this thesis and is accessible at following GitHub link: <https://github.com/Joekhuong/lodi-react-hook>

Keywords:

React, JavaScript, Front-End, Web Development, REST API, HTTP

CONTENTS

ABSTRACT	3
CONTENTS	4
VOCABULARY	5
1 INTRODUCTION	6
2 TECHNOLOGY	7
2.1 Object-Oriented Programming	7
2.2 JavaScript	8
2.3 Firebase	9
2.3.1 Firebase Authentication	10
2.4 React	10
2.5 CSS (Cascading Style Sheets)	12
2.6 Bootstrap	13
3 IMPLEMENTATION	14
3.1 Workflow	14
3.2 Analysis and design	15
3.3 Development process	16
3.4 State Management and React	18
3.5 Authentication	20
3.5.1 Login	20
3.5.2 Sign up	22
3.5.3 Logout	23
3.6 Idol management	25
3.7 User homepage	28
3.8 Posting	28
4 CONCLUSION	30
5 REFERENCES	31

VOCABULARY

API: Application Programming Interface

CSS: Cascading Stylesheets

DOM: Document Object Model

HTML: Hypertext Markup Language

JS: JavaScript

JSX: JavaScript XML

MVC: Model View Control

SDK: Software Development Kit

UI: User Interface

URL: Uniform Resource Locator, the resource address on the internet

UWP: Universal Windows Platform

XML: Extensible Markup Language

1 INTRODUCTION

React maintained by Facebook is known as a JavaScript library for the user interface development. Through the technology called JSX (JavaScript XML, Extensible Markup Language), HTML content is combined with JavaScript expressions. The author learned the JavaScript programming language at Oulu University of Applied Sciences. Besides, he also worked and had experience with it. The focus of this thesis is to study extensively by building a web application based on React. The React theory will be discussed in Chapter 2. As a result of this study, a web application named LODI is built by using React. The sample code as well as workflow will be shown in Chapter 3 for the purpose of evaluation and further discussion. The final chapter will illustrate an overview of this thesis.

The topic of the thesis as well as the LODI web application were selected based on the author's personal passion of programming language. The author has acquired more knowledge about React and programming language in general. The author also hopes that studying this topic will help him on his career in the future. He will continue developing this LODI web application perfectly and will release it to the public in the near future.

2 TECHNOLOGY

This chapter provides technologies used in this thesis project. It also gives an overview of the context and work process of the thesis.

2.1 Object-Oriented Programming

Imperative computer programming, i.e. programming by writing instructions for the computer to execute, can be divided into two main paradigms: procedural and object-oriented. In procedural programming, an application is built as a collection of variables and subroutines. The main focus in the creation of a procedural application is controlling the flow of execution using subroutine calls and flow control features of the programming language. In object-oriented programming the application is built as a collection of classes and objects. The focus is on the relations and interactions between the objects.

Classes are the foundation of object-oriented programming. A class contains declarations of member variables and functions. Member functions are often called methods. Some languages also have a concept of member properties, which are essentially a combination of a member variable with methods to get and set its value. Member variables and methods can be private or public. Private variables are only visible and accessible for the own methods of the class. Private methods can similarly only be called from within other methods inside the class. Public variables and methods can be accessed from both inside and outside of the class.

In addition to explicitly declaring member variables and methods, a class can also inherit features from another class and add new or replace existing members to refine it. The FIGURE 1 shows a simple example of class inheritance as an UML (*Unified Modeling Language*) class diagram. In this example “UIElement” is the base class from where classes “Button” and “Icon” both inherit. The base class has member variables for storing coordinates of the element and a public method for drawing. Button and Icon classes add variables and methods specific to their needs and may also implement a specific method for drawing.

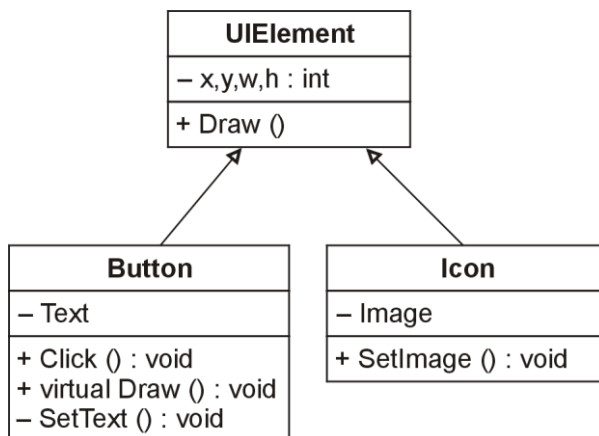


FIGURE 1. A simple UML class diagram

A class itself is only a declaration of methods and variables. An object is a realized instance of these declarations. Multiple instances of objects can be created from the same class, each having their own independent states made up by their local versions of member variables. Objects can also own other objects as their member variables, in other words have child objects.

2.2 JavaScript

JavaScript is a scripting language widely used in implementation of web based applications. It was originally developed to enable executing code at the client side of the web page, i.e. in the user's browser instead of the web server. It can also be used for implementing the server side parts of the application. Due to the virtually ubiquitous support for JavaScript in web clients and even servers, some other programming languages even use it as a target for a language-to-language compilation or translation. Despite the naming, JavaScript has no relation to the Java programming language.

A specification named ECMAScript has been created to standardize JavaScript. In this sense the JavaScript language is in fact an implementation of the ECMA-262 standard. Other implementations of the standard have also been created, but JavaScript has remained the most well-known of them. (1)

As a programming language, JavaScript is a high level interpreted language capable of supporting multiple programming paradigms. JavaScript is not fully object-oriented in technical terms, as it does not have a class construct. Instead prototypes are used for building the object oriented features into the application. The syntax of the JavaScript language is based on the Java and C programming languages (2).

When used for client-side scripting, JavaScript interacts with the web page through HTML (Hyper Text Markup Language) Document Object Model (DOM) and Cascading Style Sheets (CSS). Each HTML element on the page, including the page itself, can be accessed in JavaScript as an object, possibly with child objects. These objects can be manipulated using HTML and CSS properties. The objects on the page can also be created and deleted by the script at run time even after the page has finished loading from the server. JavaScript also interfaces with services provided by the browser, for example to communicate with the server side application over the network.

2.3 Firebase

Firebase is an application platform for the web and mobile development. It is provided by Google. It offers components and services for implementing the infrastructure of the application, such as following: (3)

- Authentication
- Database
- Storage
- Application hosting
- Analytics

The components provided are cloud based. Using them simplifies building an application because much of the technical details and decision making of the lower level implementation are hidden behind the programming interfaces provided.

Especially, authentication with the user management is a critical element of an application that can introduce severe security vulnerabilities if it is not implemented carefully. (4)

2.3.1 Firebase Authentication

Firebase provides two main ways for adding the authentication to the application. *FirebaseUI Auth*, is a component that handles the authentication process completely and even comes with the user interface for signing in. Firebase SDK Authentication provides the same functionalities as API calls (Application Programming Interface) and leaves the implementation of the user interface to the application developer.

Users can be registered and signed in using a traditional email and password combination. Firebase also handles resetting a forgotten password using the users' email address. Another option for signing in is using external identity providers, such as Google, Facebook, Twitter and GitHub. (5)

2.4 React

React, also known as "React.js" is a JavaScript library for the user interface development. It is maintained by Facebook. It allows HTML content to be combined with JavaScript expressions using the technology called JSX (JavaScript XML, Extensible Markup Language). It also introduces a virtual DOM (Document Object Model) feature, which allows the developer to focus on the functionality of the user interface rather than performance considerations.

JSX is an extension to the JavaScript syntax. It allows mixing the HTML content and JavaScript expressions into JSX elements. JSX elements are rendered to the Document Object Model. At that point the JavaScript expressions within the elements are evaluated and their values put into the DOM along with the HTML. The FIGURE 2 contains a simple example of JSX taken from the JSX documentation. In this example the variable *element* is a JSX element containing both the

HTML and JavaScript expression. The *render* function renders the element and puts its output into the *root* object of the Document Object Model. (6)

```
const name = 'Josh Perez';
const element = <h1>Hello, {name}</h1>;

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

FIGURE 2. A simple JSX Example (6)

Virtual DOM maintains an in-memory cache of the HTML Document Object Model, which is an output to the actual model in the web browser. When changes are made to the virtual DOM, a difference against the browsers model is computed and the browsers model is updated accordingly. Even if the entire virtual DOM is rewritten on every update, only the changed parts need to be rendered by the browser. A direct manipulation of the browser DOM would likely cause a massive refreshing and rendering time penalty. (7)

React can also be used to create native applications for mobile devices, namely for the ones using Android, iOS and UWP (Universal Windows Platform) operating systems. This framework is called React Native. It uses the same principle as React for web pages but instead of rendering the page through the virtual Document Object Model, it uses native user interface elements provided by the host operating system within the device. (8)

2.5 CSS (Cascading Style Sheets)

Cascading Style Sheets is a technology for controlling the visual representation of an HTML document. The CSS specification is maintained by the World Wide Web Consortium (W3C). CSS integrates with both HTML and JavaScript. A style defined using CSS can be applied to HTML elements directly to a single element from within the HTML document itself or to different types or classes of elements sharing a common style definition.

A style definition consists of a selector and property-value pairs. A selector defines which elements in the HTML Document Object Model the style is applied to. The TABLE 1 lists the basic types of selectors. Selectors can be combined, and the CSS also specifies additional modifiers to them to make their use more flexible.

TABLE 1 : Basic types of CSS selectors (9)

Selector	Example	Description
.class	.myclass	Selects all elements with class "myclass"
#id	#myid	Selects the element with id is "myid"
*	*	Selects all elements
element	span	Selects all elements
[attribute]	[target]	Selects all elements that have the attribute "target"

In addition to using selectors it is also possible to define the style properties directly to a single HTML element using the "style" attribute of the element. Modifying the CSS style names and individual style properties from JavaScript is possible at runtime by accessing the elements using the Document Object Model functions.

2.6 Bootstrap

Bootstrap is a framework for implementing a front-end of a website or web-application. It is based on templates using HTML, CSS and JavaScript. A notable feature of Bootstrap is the responsiveness of the page layout, meaning that the page can be laid out differently depending on the size and orientation of the screen. The final page layout will respond to changes in screen geometry, namely the width, in real time.

The design philosophy of Bootstrap is “mobile first”, which means that the code and layouts are optimized primarily for use in mobile devices and scale up from that when being viewed on larger screens. For example, the main layout in a mobile device view is by default one column, which is used to present all the content of the page. When the screen size is increased, columns and other elements can be added to make navigation easier and to present additional content that would otherwise need to be viewed by scrolling.

Several components are included to enable a dynamic operation of the website or application. Examples of such are a button, a breadcrumb navigation, a tooltip, a modal dialog and a content spinner.

3 IMPLEMENTATION

3.1 Workflow

In every client-server application, there are two important parts: frontend and backend. The frontend is called a client. It can be a web site viewed with a browser or a mobile device application. Users interact with the client side. The second part is the server. It will handle requests from frontend, such as getting data or sorting new data.

In this thesis project, the backend was implemented as a NodeJS application. On the other hand, the frontend was implemented as a React application. The main focus of the thesis is on the frontend, implemented as a web client. The backend exposes a REST API that is used by the frontend. Communication is done through the HTTP protocol as shown in the FIGURE 3.

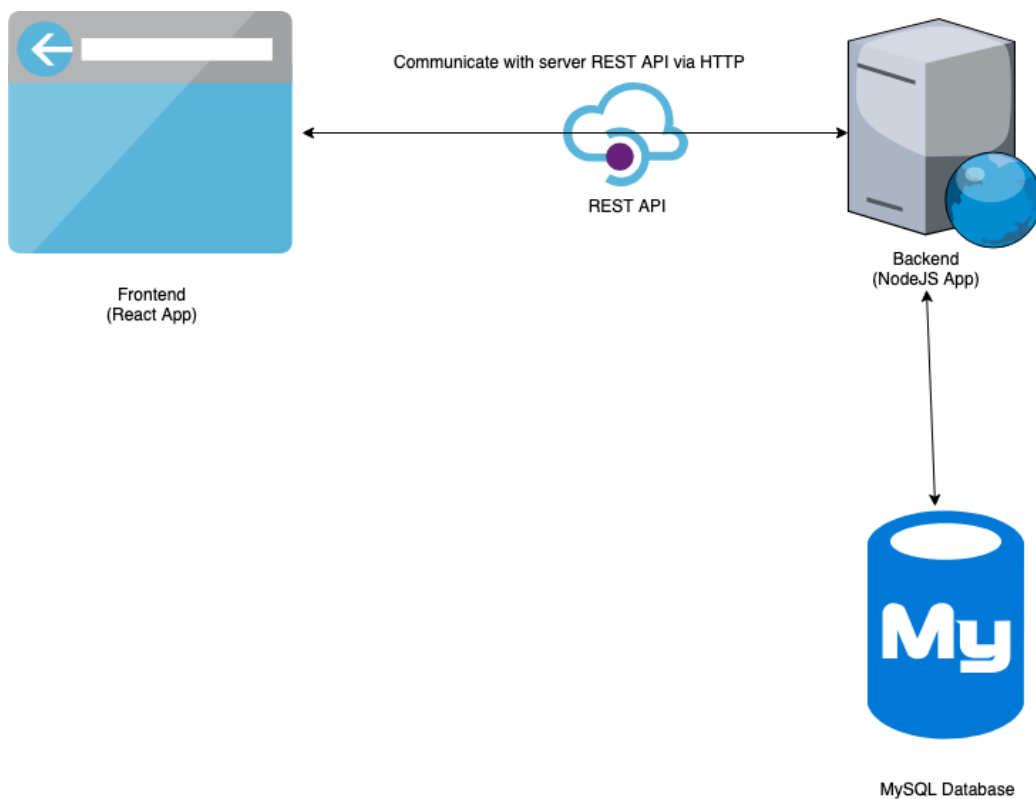


FIGURE 3. The workflow of the application

3.2 Analysis and design

First, an analysis was carried out. After this step, the decision was made for the frontend application to include the following features:

- The user can create an account
- The user can login and logout
- The admin can create, edit and delete an idol
- The user can search for an idol by name or hashtag
- On the user page, the user can see the ranking table based on the region or globally
- On the user page, the user can see the list of idols they are following
- On the user page, all posts made by the user will be shown
- On the idol page, the user can follow or unfollow the idol
- On the idol page, all posts that belong to the idol will be displayed.

The next step was to go through the features and main functions. The author created the following views to meet the requirements:

- A login page
- A signup page
- A home page
- An Idol page
- An idol search page.

The next step was to analyze the REST API, which is exposed by the backend and is used to meet the needs of the client. The TABLE 2 describes briefly the resources used in the React application.

TABLE 2. Backend resource list

RESOURCE	URL	DESCRIPTION
idol	/api/idol	Expose functions to work with idol
user	/api/user	Expose functions to work with user
follow	/api/follow	Expose functions to follow, unfollow and get ranking
region	/api/region	Expose function to get all region data
post	/api/post	Expose functions to work with post

3.3 Development process

This project used Create React (10), a tool to create a React application, which is supported by Facebook. It created a boilerplate of a React application. Also, Babel (11) and Webpack (12) were installed automatically and preconfigured. Those configurations were hidden but they can be edited later if needed. The following are the features provided by the Create React Tool:

- Support ES6+ syntax
- Babel to transpile ES6+
- Webpack for assets bundling
- Hot reloading of development server
- Building production application
- Testing environment with Jest.

The folder structure of the basic application generated by Create React App is shown in the FIGURE 4.

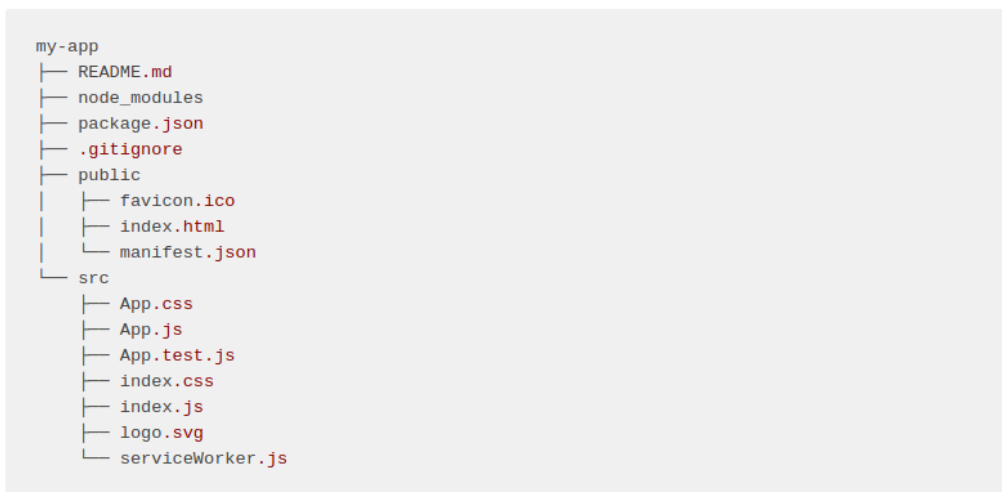
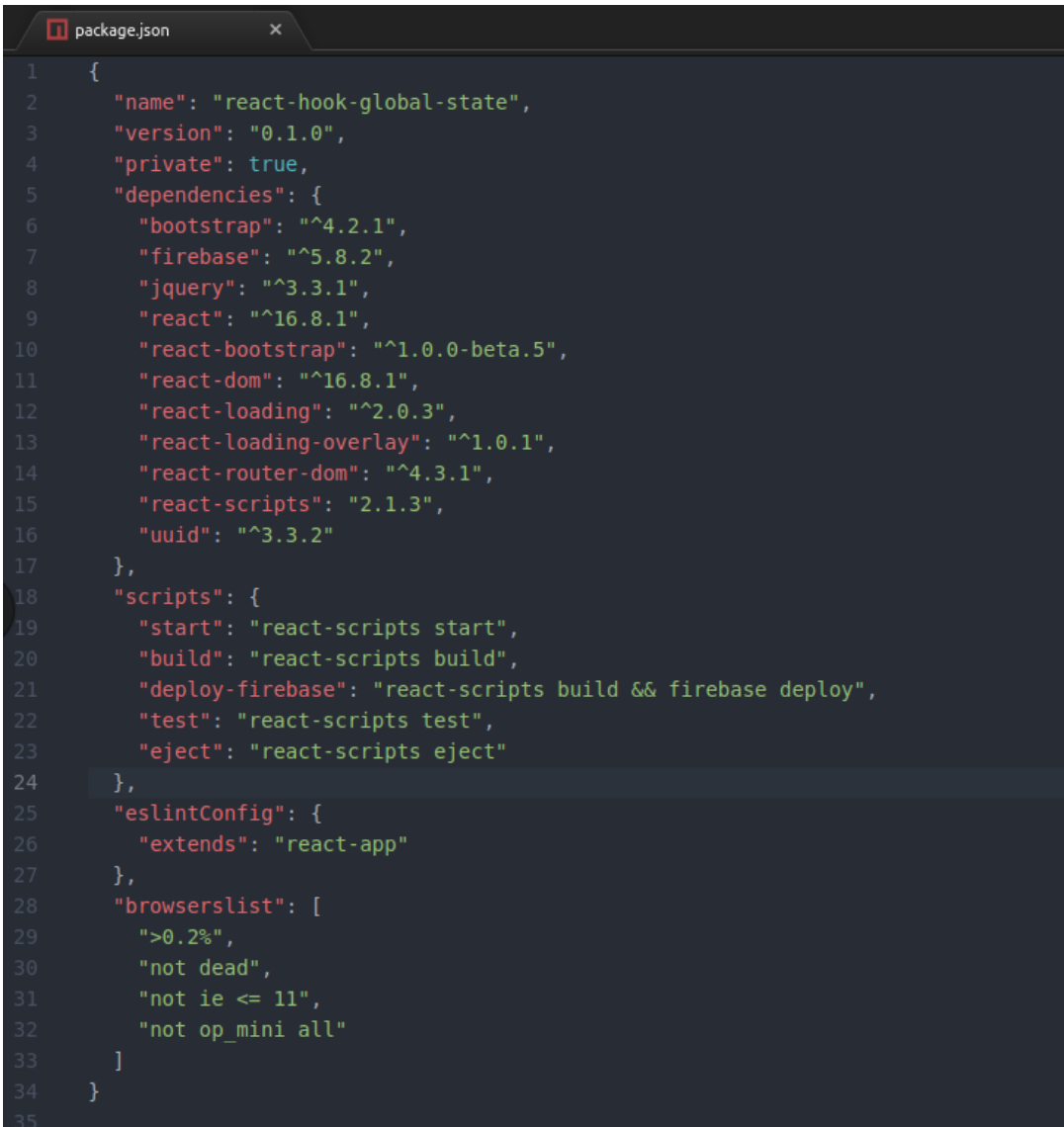


FIGURE 4. The basic application structure generated by Create React Tool

In this thesis project, the following main packages were installed via the NPM tool:

- React-router-dom (13): Used for routing and DOM binding for React Router (14)
- Firebase (15): Helps to work with firebase
- react-bootstrap (16): The rebuild for the React of Bootstrap front-end framework
- React-dom, react: The main library for ReactJS.

Dependencies for this project contained in the package.json file are shown in The FIGURE 5.



```
1  {
2    "name": "react-hook-global-state",
3    "version": "0.1.0",
4    "private": true,
5    "dependencies": {
6      "bootstrap": "^4.2.1",
7      "firebase": "^5.8.2",
8      "jquery": "^3.3.1",
9      "react": "^16.8.1",
10     "react-bootstrap": "^1.0.0-beta.5",
11     "react-dom": "^16.8.1",
12     "react-loading": "^2.0.3",
13     "react-loading-overlay": "^1.0.1",
14     "react-router-dom": "^4.3.1",
15     "react-scripts": "2.1.3",
16     "uuid": "^3.3.2"
17   },
18   "scripts": {
19     "start": "react-scripts start",
20     "build": "react-scripts build",
21     "deploy-firebase": "react-scripts build && firebase deploy",
22     "test": "react-scripts test",
23     "eject": "react-scripts eject"
24   },
25   "eslintConfig": {
26     "extends": "react-app"
27   },
28   "browserslist": [
29     ">0.2%",
30     "not dead",
31     "not ie <= 11",
32     "not op_mini all"
33   ]
34 }
35
```

FIGURE 5. Contents of package.json

3.4 State Management and React

Every React application is a combination of modules and the most important thing in each module is its state. There are many alternatives to manage the state such as passing it down as properties from a parent to a child module or using a library like Redux (17).

This project used the latest version of ReactJS. It provides many features to manage module states, such as Context (18) and UseReducer (19). They can be used together to manage the state of the component. The state management is implemented in the store.js file shown in the FIGURE 6.

```
1  import React from 'react';
2  import reducer, {initialState} from './reducers'
3
4  const Store = React.createContext();
5
6  export const connect = (
7    mapStateToProps = () => {},
8    mapDispatchToProps = () => {}
9  ) => WrappedComponent => {
10   return (props) => {
11     const {dispatch, state} = React.useContext(Store);
12
13     return (
14       <WrappedComponent
15         dispatch={dispatch}
16         {...props}
17         {...mapStateToProps(state, props)}
18         {...mapDispatchToProps(dispatch, props)}
19       />
20     )
21   }
22 }
23
24 const createStore = (reducer, initialState) => {
25   const [state, dispatch] = React.useReducer(reducer, initialState);
26   return {state, dispatch};
27 }
28
29 const Provider = ({children}) => {
30   const store = createStore(reducer, initialState);
31   return <Store.Provider value={store}>{children}</Store.Provider>
32 }
33
34 export {Store, Provider};
35
```

FIGURE 6. A screenshot of the Store component

Two important parts of the application are the “connect function” and the “Provider constant”. The provider function works as a wrapper to help the component using the context of the store. The usage of this function is shown in the FIGURE 7.

```
1  import React from 'react';
2  import {Provider} from './store';
3  import Routing from './Routing';
4  import Header from './Header';
5
6  const App = () => {
7    return (
8      <div className="App">
9        <Provider>
10         <Routing/>
11       </Provider>
12     </div>
13   );
14 }
15
16 export default App;
17
```

FIGURE 7. A screenshot of the Provider usage

The connect function is the core function. It is used to map the state and dispatch the context to the properties of the component. This is the input of the function. Therefore, the state will be mapped using `mapStateToProps` and `mapDispatchToProps`. It works in the same way with the connect function of the Redux store. The FIGURE 8 will show an example of how this function is used.

```
1 import React from "react";
2 import { connect } from "../store";
3 import { withRouter } from "react-router-dom";
4 import {
5   Container,
6   Row,
7   Badge
8 } from "react-bootstrap";
9 import PostComment from "../PostComment";
10
11 const mapStateToProps = (state, props) => ({
12   ...props,
13   ...state
14 });
15
16 const mapDispatchToProps = (dispatch, props) => ({
17
18 });
19
20 class Post extends React.Component {
21
22   state = {
23     created_by: null,
24     is_follow: null
25   }
26
27   componentDidMount() {
28     //GET COMMENT
29   }
30
31   render() {
32
33     var date = new Date(this.props.item.createdAt);
34     return (
35       <>
36       <Container className="post-item border border-dark mb-4">
37         <Row className="post-footer p-3 bg-primary">
38           <Badge variant="dark">Posted by: {this.props.item.user_name}</Badge>
39           <Badge variant="dark" className="ml-2">Posted at: {date.toDateString()}</Badge>
40           {this.props.item.idol_name != null ? <Badge className="ml-2" variant="dark">Idol: {this.props.item.idol_name}</Badge> : ""}
41         </Row>
42         <Row className="post-content p-3">
43           {this.props.item.content}
44         </Row>
45         <PostComment parent_id={this.props.item.id} />
46         <hr />
47       </Container>
48     </>
49   );
50 }
51
52 }
53
54
55 export default connect(
56   mapStateToProps,
57   mapDispatchToProps
58 )(withRouter(Post));
59
```

FIGURE 8. A screenshot of the connect function usage

3.5 Authentication

3.5.1 Login

Users will use this page to log into the system by using their email addresses and passwords. The FIGURE 9 shows the user interface of the login page. When a user touches the “SIGN IN” button, the React application will authenticate the user with Firebase. If the email address and password do not match, an error will be shown as an alert. Otherwise, the system will get a response containing an

uid and use this to receive user information from the backend system via REST api. The FIGURE 10 shows the workflow of the login page.

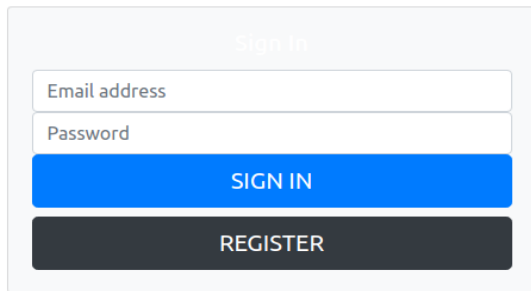


FIGURE 9. A screenshot of the Login page

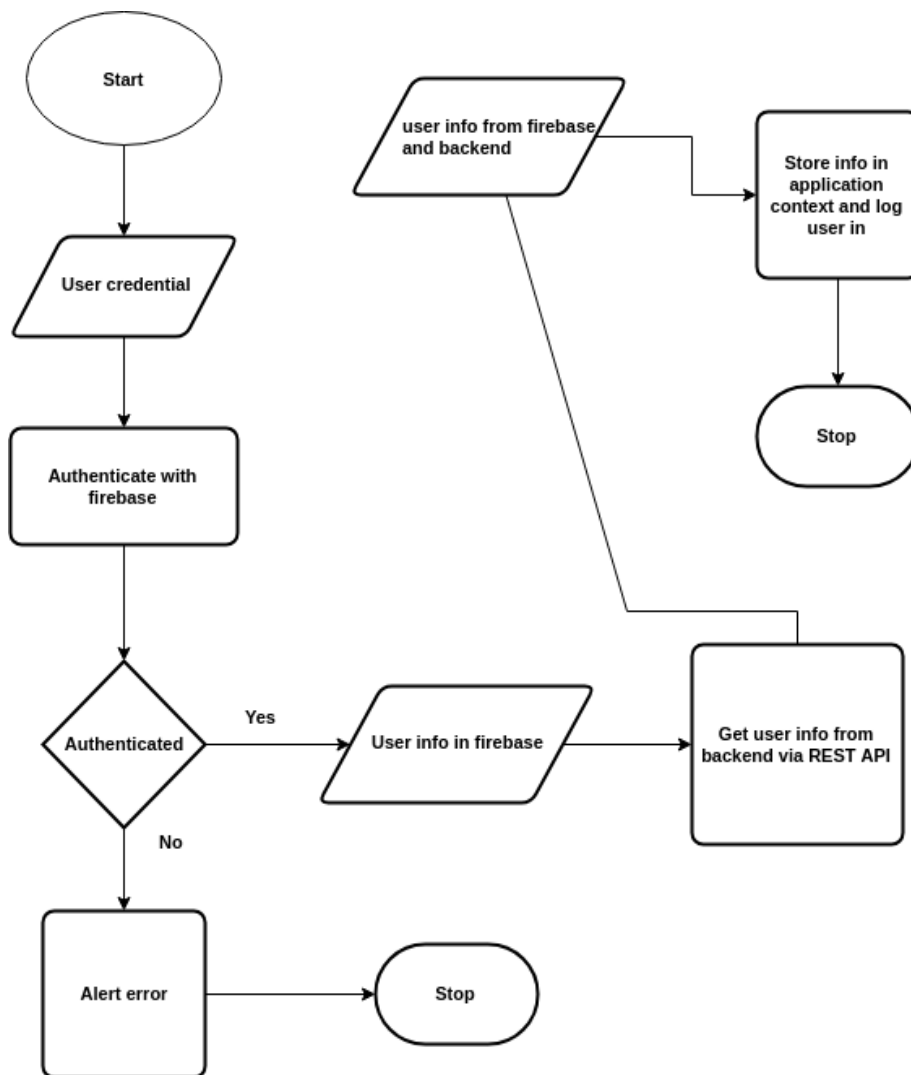
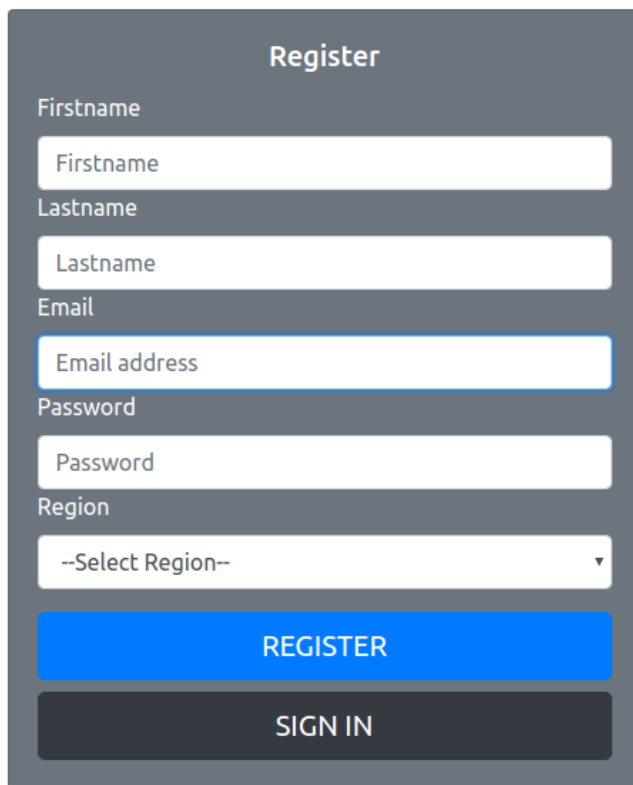


FIGURE 10. The workflow of the Login process

3.5.2 Sign up

The user will use this page to register an account by using the email address and password. The user interface is shown in the FIGURE 11. When the user touches the “REGISTER” button, the React application will register the user with Firebase. If the email address already exists in Firebase, an error will be shown as an alert. Otherwise, the system will receive a response containing uid and will send it combined with the region information to the backend system via REST api, where an entry is created in the database. The FIGURE 12 shows the workflow of the Sign up process.



The image shows a mobile application interface for a registration page. The page has a dark grey background with the title "Register" at the top center. Below the title, there are five input fields stacked vertically, each with a label above it: "Firstname", "Lastname", "Email", "Password", and "Region". The "Email" field is currently selected, indicated by a blue border. Below the input fields, there are two buttons: a blue "REGISTER" button and a dark grey "SIGN IN" button.

FIGURE 11. A screenshot of the Register page

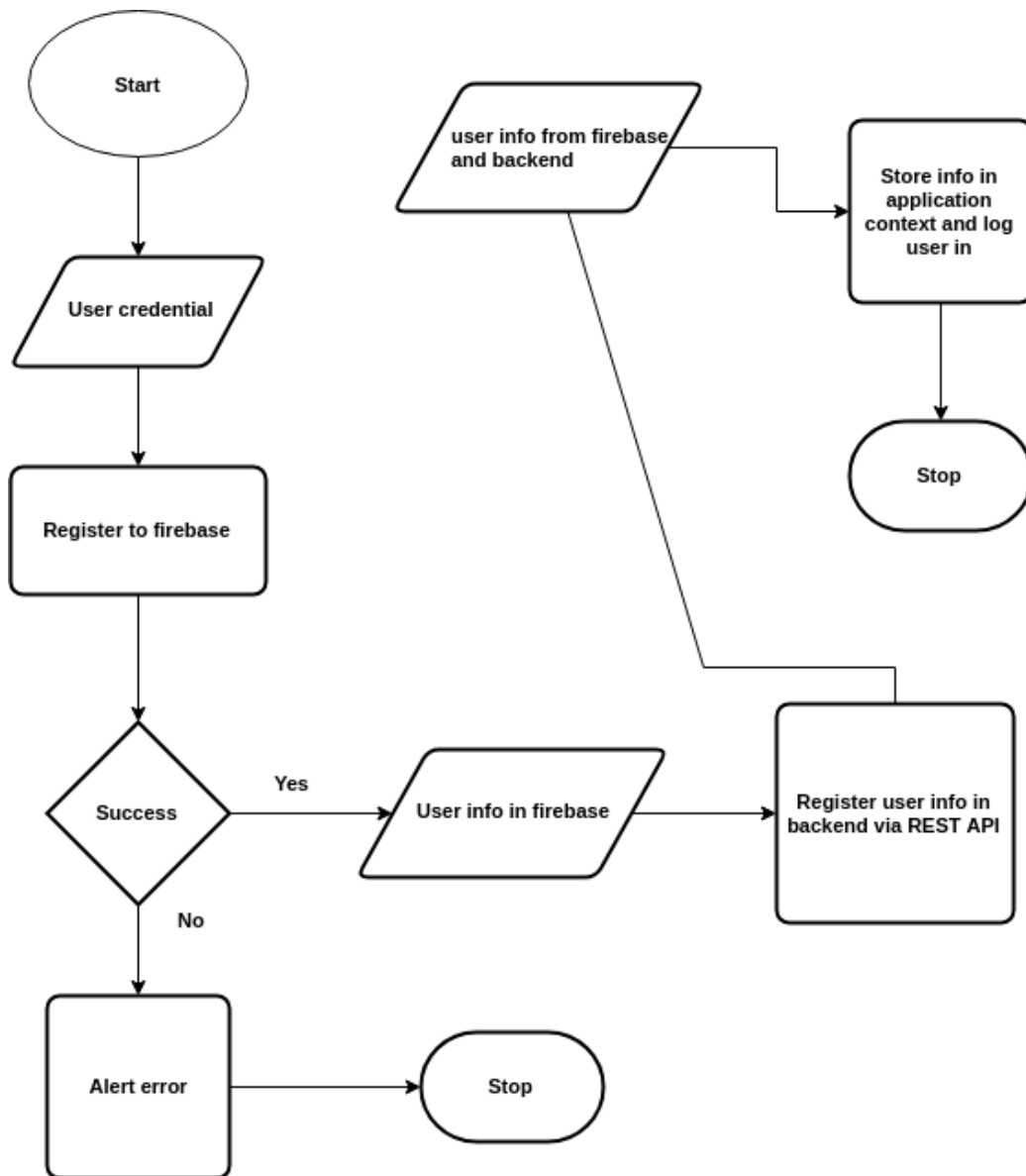


FIGURE 12. The workflow of the register process

3.5.3 Logout

There is no specific page or UI for the logout feature. There will be a button in the menu to logout, which is shown in the FIGURE 13.

A screenshot of a user interface. On the left, the text "Welcome Lodi Admin!" is displayed. To its right is a rectangular button with the text "Sign out" inside it.

FIGURE 13. A screenshot of the Logout button

When the user touches the “sign out” button, React will execute the code and tell Firebase to logout the current user. After that, the application will remove the user out of the system state. The FIGURE 15 shows the workflow of the Sign-out process.

```
const mapDispatchToProps = (dispatch, props) => ({
  logout: () => {
    dispatch({
      type: LOGOUT_ACTION,
      payload: {}
    });
    firebase.auth().signOut();
    setTimeout(() => (props.history.push("/")), 100);
  }
});
```

FIGURE 14. The logout snippet code

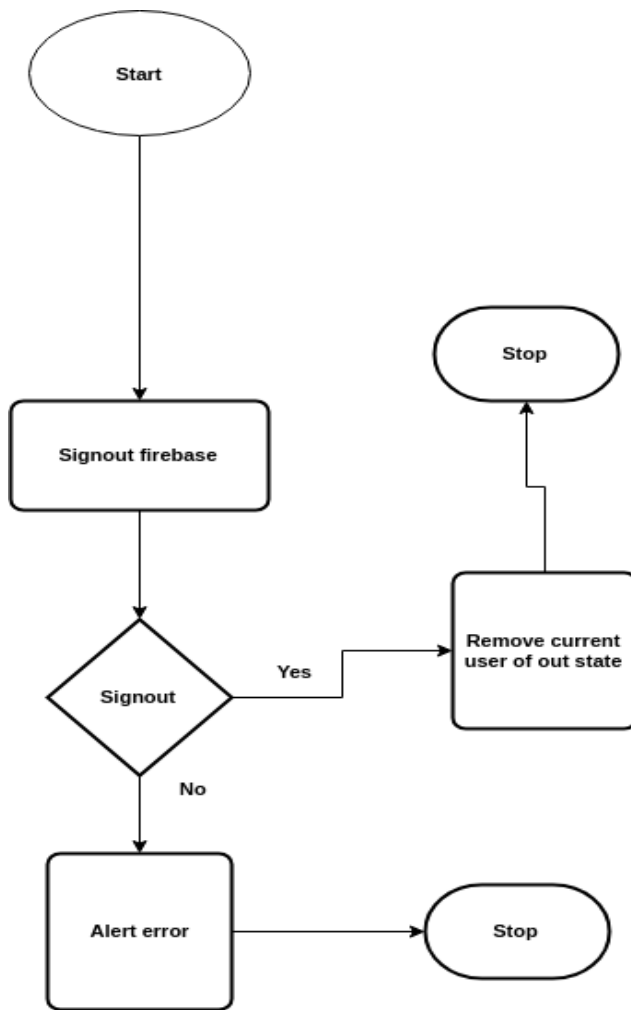


FIGURE 15. The workflow of the Sign-out process

3.6 Idol management

In this thesis project, the idol management page is accessed only by the admin. On this page, the admin can create, edit or delete an idol. This page is mainly working with the idol resource of the backend via REST API. The User interface of the idol management page is shown in the FIGURE 16.

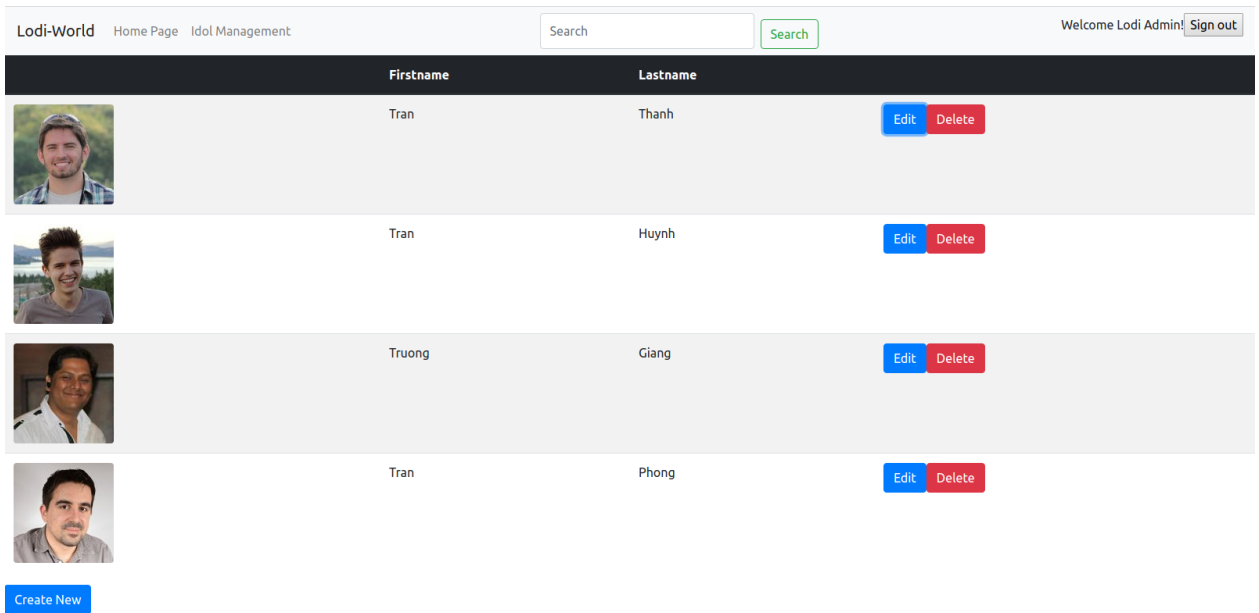


FIGURE 16: A screenshot of the Idol list on the idol management page

When the admin user touches the “Create new” or “Edit” button, a modal form will appear. There, the user can give information for the idol and touch “Save”. According to the action “Create” or “Edit”, the application will make a request to the backend REST API through the idol resource. When receiving a response from the backend, the idol list will be updated with the latest information. The user interface of editing an idol is shown in the FIGURE 17.

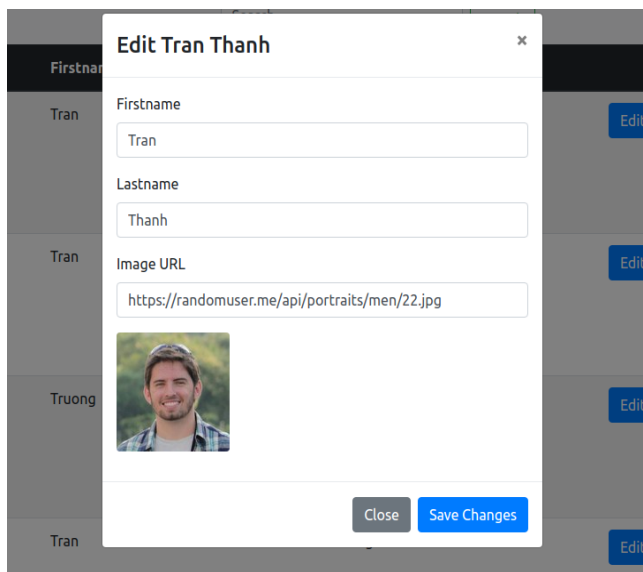


FIGURE 17. A screenshot of the idol edit modal form

In case of deleting an idol, a modal dialog asking for confirmation will be shown. The user needs to confirm if they want to delete the idol or cancel it. The dialog is shown in the FIGURE 18.

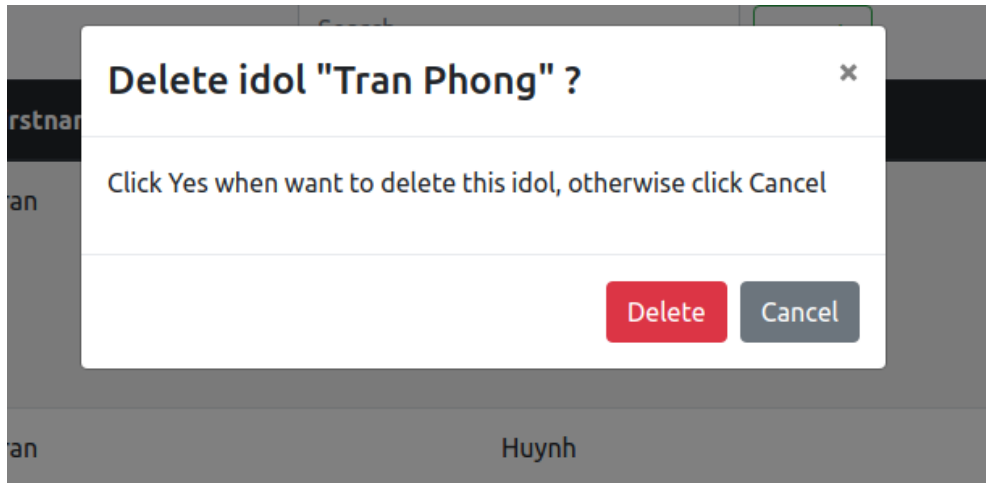


FIGURE 18. A screenshot of the delete confirm modal

Each idol will have their own page to show all their information and all posts belonging to the idol. On this page, a user can follow or unfollow the idol or create a post. The user interface of the idol page is shown in the FIGURE 19.

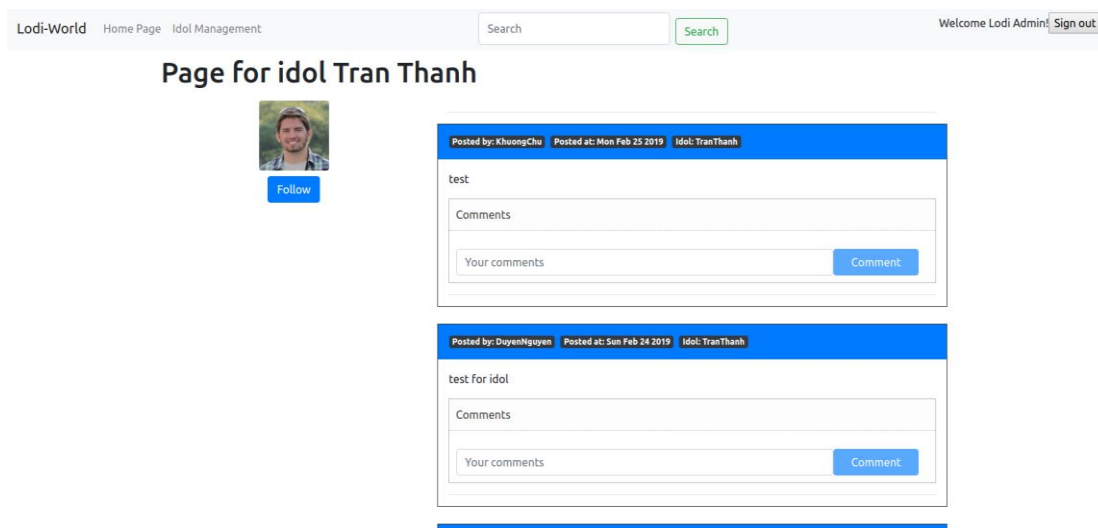


FIGURE 19. A screenshot of the idol page

3.7 User homepage

Each user will have their own homepage. This page will show all the idols that the user has followed. Also, a ranking table will show top 5 idols based on the user's region or alternatively without a region.

As mentioned earlier, a user can create a post on the idol page. Furthermore, they can also create a post for themselves on their home page. The user interface of the user homepage is shown in the FIGURE 20.

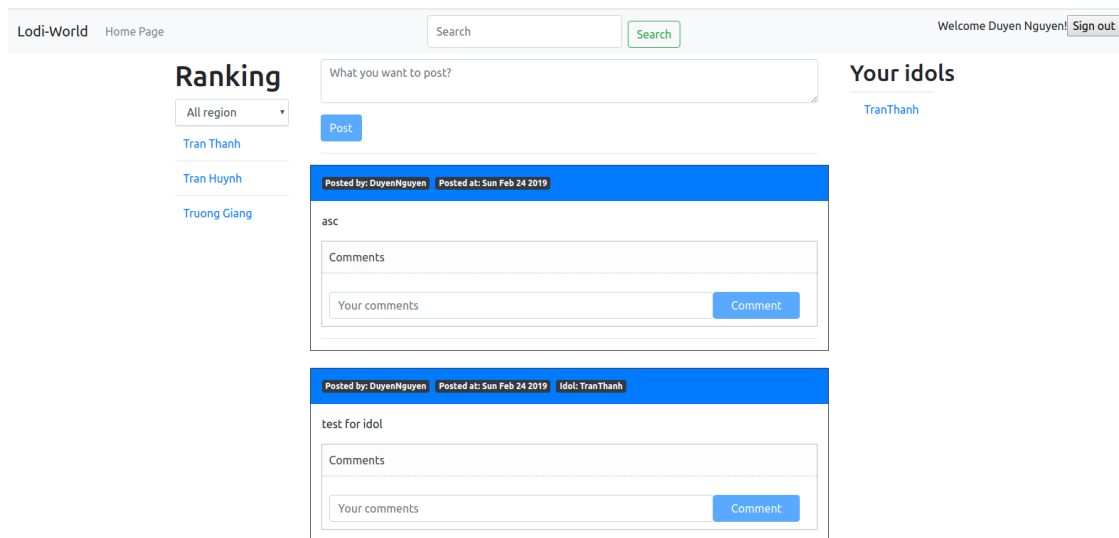


FIGURE 20. A screenshot of the user's homepage

3.8 Posting

One of the features of this application is to allow a user to create a post for an idol as well as for themselves. In order to make a post, the user needs to fill the content into the form shown in the FIGURE 21 and touch the "Post" button.

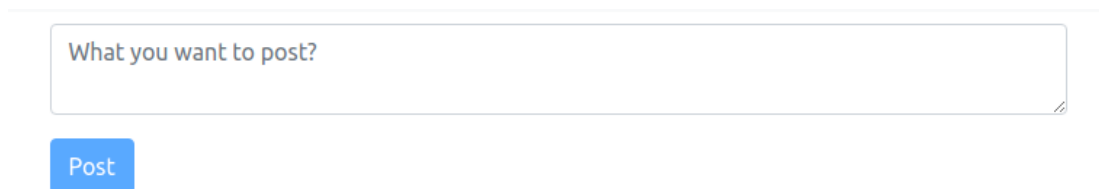


FIGURE 21. A screenshot of the Posting form

When the user touches the “Post” button, the application will show a modal dialog (shown in the FIGURE 22) for the user to confirm the post or cancel it. After that, the application will make a request to post resource in the backend and add the post into a database.

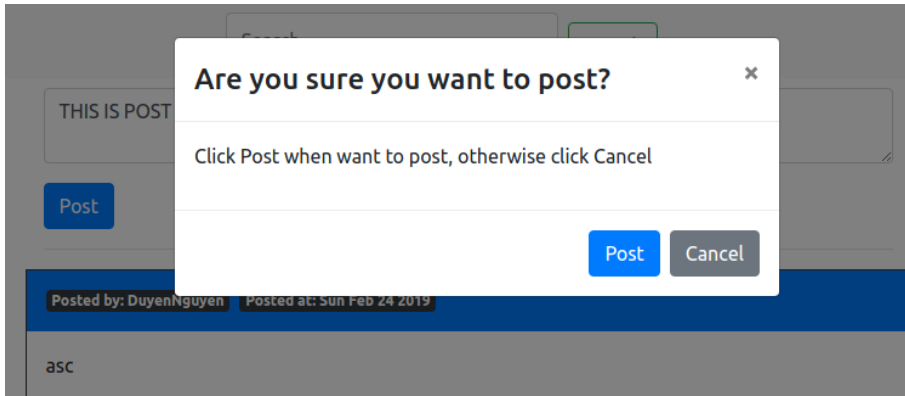


FIGURE 22: A screenshot of the confirm modal when posting

In the header of each post, there are tags including the owner of this post, the day and time as well as the name of the idol whom the post owner is following. Furthermore, the comment box of the post will show all comments related to this post. A screenshot of a single post is shown in the FIGURE 23.

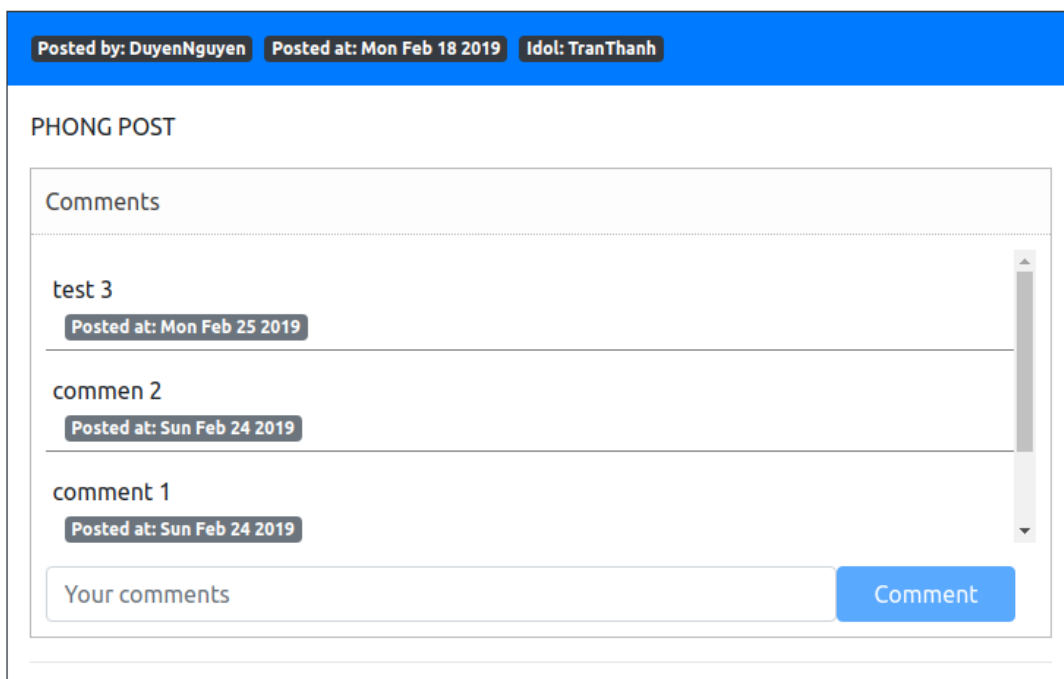


FIGURE 23: A screenshot of the single post

4 CONCLUSION

While developing the project, the author studied the advantages and disadvantages of React. React is under a continuous development. It is easy to use and creating light components is fast. With the JSX technology, the HTML code is inserted easily into JS. Furthermore, React is very effective, because it internally creates a virtual DOM. However, it also has some weaknesses. Unlike other MVC frameworks, React is just a view-oriented one. It does not have the Model and Controller concepts.

In the end, all the defined features were implemented, and the result worked as expected. However, some things were left to be improved in the future.

The current version will receive all post of a user or idol page at one time and it will lead to performance issues. To solve this problem, in the future the pagination technique (20) will be applied for showing a post and its comments. Furthermore, the author will provide a possibility that a user can reply to the comment of the post. Also, the communication feature needs to be updated so that users can make friends and interact with each other.

The main workflow of this project is working through the REST API now. It can later lead to an increase in the number of requests from the React application to the backend. For this problem, the author will work with GraphQL (21), which will provide more possibilities to controlling the data needs within one request. Moreover, the Jest framework (22) will also be applied to prevent bugs before being deployed into production.

5 REFERENCES

1. Standard ECMA-262. Date of retrieval 17.12.2018. ECMA International 2018, <http://www.ecma-international.org/publications/standards/Ecma-262.htm>.
2. A re-introduction to JavaScript - Mozilla Developer Center. Date of retrieval 09.04.2018 https://developer.mozilla.org/en-US/docs/Web/JavaScript/A_re-introduction_to_JavaScript.
3. Google. Firebase. Date of retrieval 10.01.2019 <https://firebase.google.com/>.
4. Krishnan, Sid. Why You Shouldn't Roll Your Own Authentication. Date of retrieval 25.01.2019 <https://blog.codeship.com/why-you-shouldnt-roll-your-own-authentication/>.
5. Firebase Authentication, Google. Date of retrieval 15.01.2019 <https://firebase.google.com/docs/auth/>.
6. Introducing JSX. Date of retrieval 25.04.2018 <https://reactjs.org/docs/introducing-jsx.html>.
7. Virtual DOM and Internals. Date of retrieval 16.05.2018 <https://reactjs.org/docs/faq-internals.html>.
8. React Native Documentation. Date of retrieval 05.08.2018 <https://facebook.github.io/react-native/>.
9. CSS Selector Reference. Date of retrieval 13.10.2018 https://www.w3schools.com/cssref/css_selectors.asp.
10. Create React App - Getting Started. Date of retrieval 21.08.2018 <https://facebook.github.io/create-react-app/docs/getting-started>.
11. Babel. Date of retrieval 30.10.2018 <https://babeljs.io/>.
12. Webpack. Date of retrieval 18.12.2018 <https://webpack.js.org/>.

13. React-router-dom. Date of retrieval 21.08.2018
<https://reacttraining.com/react-router/web/guides/quick-start>.
14. React-router. Date of retrieval 21.08.2018 <https://reacttraining.com/react-router/>.
15. Firebase npm package. Date of retrieval 10.01.2019
<https://www.npmjs.com/package/firebase>.
16. React-bootstrap. Date of retrieval 03.02.2019 <https://react-bootstrap.github.io/>.
17. Redux. Date of retrieval 05.02.2019 <https://redux.js.org/>.
18. React Context. Date of retrieval 11.02.2019
<https://reactjs.org/docs/context.html>.
19. useReducer Hook. Date of retrieval 11.02.2019
<https://reactjs.org/docs/hooks-reference.html#usereducer>.
20. Pagination Wiki. Date of retrieval 20.12.2018
<https://en.wikipedia.org/wiki/Pagination>.
21. GraphQL. Date of retrieval 05.01.2019 <https://graphql.org/learn/>.
22. Jest Framework. Date of retrieval 21.12.2018 <https://jestjs.io/>.