

React Router for Firebase

In this lesson, we will implement routing using React router in our application.

WE'LL COVER THE FOLLOWING ^

- Background
- Getting Started

Background

Since we are building a full-fledged application in this course, it is good practice to have separate pages (e.g. landing page, account page, admin page, sign-up page, sign-in page) to divide the application into multiple URLs (e.g. /landing, /account, /admin). These URLs or subpaths of a domain are called **routes** in a *client-side web application*.

Let's implement *routing* in our application using **React Router** before we dive into incorporating Firebase for a real-time database, authentication, and authorization. Don't worry if you haven't used **React Router** before; the concepts are quite basic and you should be able to understand and follow the instructions throughout this course.

The application should ideally have multiple routes. For instance, the user should be able to visit the public landing page, and also be able to use the sign-up and sign-in pages to enter the application as an authenticated user. If a user is authenticated, only then will he be allowed to visit protected pages such as the account page or admin page, where the latter will only be accessible to authenticated users with an admin role.

You can centralize all the routes of your application in a well-defined `src/constants/ routes.js` file.

```
export const LANDING = '/';  
export const SIGN_UP = '/sign-up';
```



```
export const SIGN_UP = '/signup';
export const SIGN_IN = '/signin';
export const HOME = '/home';

export const ACCOUNT = '/account';
export const ADMIN = '/admin';
export const PASSWORD_FORGET = '/pw-forget';
```

routes.js

Each route represents a page of your application. For instance, the sign-up page should be reachable in development mode via <http://localhost:3000/signup> and in production mode via <http://yourdomain/signup>.

First, we will have a **sign-up page** (register page) and a **sign-in page** (login page). We can take any web application as a blueprint to structure these routes for a well-rounded authentication experience. Consider the following scenario: A user visits your web application, is persuaded to buy your service, and finds the button in the top-level navigation bar to sign in to the application. But the user does not have a personal account yet, so a sign-up button is presented as an alternative on the sign-in page.

the Road to React; LOG IN

Log In

Email Address

Password

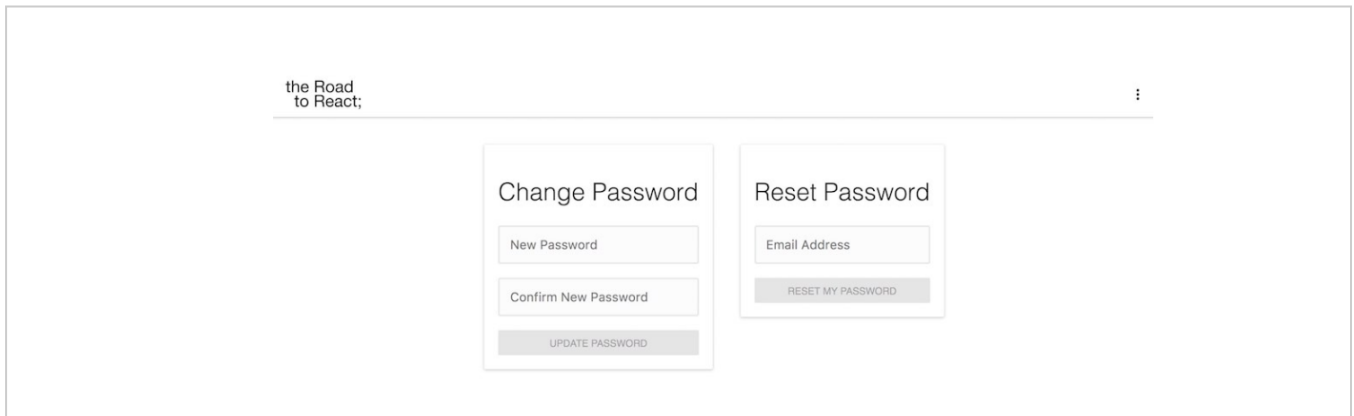
LOG IN

[Forgot Password?](#)

Don't have an account? [Sign Up](#)

Second, there will be a **landing page** and a **home page**. The **landing page** is the **default route** for your application (e.g. <http://yourdomain/>). This is the place a user is navigated to when visiting your web application. The user does not need to be *authenticated* to go to this route (page). On the other hand, the **home page** is a **protected route**, which can only be accessed if the users have been authenticated. We will implement *protection of the routes* using authorization mechanisms for this application in upcoming chapters.

Third, next to the **home page**, there will be a protected **account page** and an **admin page**. On the **account page**, a user can *reset* or *change* a password. This is secured by authorization as well, so it can only be reached by authenticated users. On the **admin page**, only the user authorized as an admin will be able to manage this page (the users of this application). The admin page is protected on a more fine-grained level because it is only accessible to authenticated admin users.



the Road to React;

Change Password

Reset Password

Lastly, the **password forget** component will be exposed on a non-protected page (the **password forget page**). It is used by users who are not authenticated and forgot their password.

Getting Started

We find it exciting to build a wholesome and versatile application because it can be used as a boilerplate project that provides us with authentication, authorization, and a database, which are the foundational pillars for any web-based application.

Now that we've completed the routes for our React with Firebase application, all of them need to be accessible to the user. First, we need a router for our web application which will be responsible for mapping routes to React components. **React Router** is a popular package used to enable routing, so we will install it using the command line shown in the code snippet below:

```
npm install react-router-dom
```



The best way to start is by implementing a **Navigation component** which will

be used in the **App component**. The App component is the perfect place to render the Navigation component because it always renders the Navigation component but replaces other components (pages) based on the routes. Basically, the App component is the container into which all our **fixed components** (e.g. navigation bar, sidebar, footer), as well as the components that are displayed depending on the route in the URL (e.g. account page, login page, password forget page), will go.

First, the App component will use the Navigation component (that is not implemented yet). It will also use the Router component provided by React Router. The Router makes it possible to navigate from URL-to-URL on the client-side of the application without another request to a web server for every route change. The application is only fetched once from a web server, after which all routing is done on the client-side with React Router.

In `src/components/ App/index.js` file, implement the following code:

```
import React from 'react';
import { BrowserRouter as Router } from 'react-router-dom';

import Navigation from '../Navigation';

const App = () => (
  <Router>
    <Navigation />
  </Router>
);

export default App;
```

App/index.js

Second, implement the **Navigation component**, which uses the **Link component** of React Router to enable navigation to different routes. We defined these routes previously in the *constants* file. Let's import all of them and give a specific route to every Link component.

In `src/components/ Navigation/index.js` file, modify the code as follows:

```
import React from 'react';
import { Link } from 'react-router-dom';
```

```
import { Link } from 'react-router-dom';

import * as ROUTES from '../constants/routes';

const Navigation = () => (
  <div>
    <ul>
      <li>
        <Link to={ROUTES.SIGN_IN}>Sign In</Link>
      </li>
      <li>
        <Link to={ROUTES.LANDING}>Landing</Link>
      </li>
      <li>
        <Link to={ROUTES.HOME}>Home</Link>
      </li>
      <li>
        <Link to={ROUTES.ACCOUNT}>Account</Link>
      </li>
      <li>
        <Link to={ROUTES.ADMIN}>Admin</Link>
      </li>
    </ul>
  </div>
);

export default Navigation;
```

Navigation/index.js

Now that we are done with this, we run our application again and verify that the links show up in the browser window and that once the link is clicked, the URL changes. Note that even though the URL changes, the displayed content does not change. The navigation is only there to enable navigation through your application but none of them know what to render on each route. This is where the *route to component mapping* comes into use. In the **App component**, you can specify which components should show up according to their corresponding routes with the help of the **Route component** from React Router.

In the `src/components/ App/index.js` file, add the following modifications:

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

import Navigation from '../Navigation';
import LandingPage from '../Landing';
```

```

import SignUpPage from '../SignUp';
import SignInPage from '../SignIn';
import PasswordForgetPage from '../PasswordForget';

import HomePage from '../Home';
import AccountPage from '../Account';
import AdminPage from '../Admin';

import * as ROUTES from '../constants/routes';

const App = () => (
  <Router>
    <div>
      <Navigation />

      <hr />

      <Route exact path={ROUTES.LANDING} component={LandingPage} />
      <Route path={ROUTES.SIGN_UP} component={SignUpPage} />
      <Route path={ROUTES.SIGN_IN} component={SignInPage} />
      <Route path={ROUTES.PASSWORD_FORGET} component={PasswordForgetPage} />
      <Route path={ROUTES.HOME} component={HomePage} />
      <Route path={ROUTES.ACCOUNT} component={AccountPage} />
      <Route path={ROUTES.ADMIN} component={AdminPage} />
    </div>
  </Router>
);

export default App;

```

App/index.js

If a route matches a path prop, the respective component will be displayed; thus, all page components in the App component are exchangeable by changing the route, but the Navigation component stays fixed independent of any route changes. This is how we enable a static frame with various components (e.g. Navigation) around dynamic pages that are driven by routes. It is all made possible by [React's powerful composition](#).

Previously, we created basic components for each page component used by our routes. Now you should be able to view a different display as you click on each Navigation component. The routes for the PasswordForget and SignUp components are not used in the Navigation component but will be defined in upcoming lessons. For now, we can pat ourselves on the back because we have successfully implemented fundamental routing for this application!

In the next lesson, you can run and verify the app we have built so far on our live terminal!

