# Authentication

Supports 3 authentication methods: [Auth0](https://auth0.com/), [Firebase Authentication](https://firebase.google.com/docs/auth), [JSON Web Token](https://jwt.io/). You can switch between them at any time in a few of seconds. In our demo website we have Firebase by default, but we shipped with JWT version to allow you to start the application without having to configure the Auth0/Firebase api keys.

We’ve been testing multiple alternatives to support each integration easily without adding complexity to the project. We started with the concept of having all methods active at the same time under Redux with “service” utils, then we decided that is too difficult and hard to modify, and we reimplemented it using React Context API.

## How it works

Accessing dashboard pages is only possible for authenticated users. If this condition is not fulfilled, the user is redirected to a login page. To make this work we implemented the “Guard” concept from Angular. It is simply a component that wraps a route and checks for user authentication status before allowing the navigation. To find more information about guards, please visit the “Routing” page.

In the src/App.js (src/App.tsx for TS version) we have this line to specify the auth provider:

import { AuthProvider } from 'src/contexts/JWTAuthContext';

This line imports the AuthProvider from the JSON Web Token method. In the src/contexts folder we have the all 3 methods.

We use this provider to wrap the app content:

<Router history={history}>

<AuthProvider>

<GlobalStyles />

<ScrollReset />

<GoogleAnalytics />

<CookiesNotification />

<SettingsNotification />

{renderRoutes(routes)}

</AuthProvider>

</Router>

The reason for having the AuthProvider under Router is to allow the Auth Context to have access to router history (if needed).

Now that we have the AuthProvider, we can use the context directly by importing useContext from React and specifying the context that we want to use or we can use the custom hook useAuth from src/hooks/useAuth.js (src/hooks/useAuth.ts for TS version).

## Setup (optional)

### **Auth0**

To use Auth0 we need to configure the Auth0 Client first. The config object auth0Config can be found in src/config.js (src/config.ts for TS version).

export const auth0Config = {

client\_id: process.env.REACT\_APP\_AUTH0\_CLIENT\_ID,

domain: process.env.REACT\_APP\_AUTH0\_DOMAIN

};

Access your Auth0 application dashboard and get your own client\_id and domain. Remember to setup your Auth0 Application according the the Auth0 documentation, Single Page Application - React.

You can use the .env method to be able to have different keys depending on your environment (development, staging or production). If you will be using .env, in the .env.example file you have the variables that we use in our project.

This config object is being used in the src/contexts/Auth0Context.js (src/contexts/Auth0Context.tsx for Ts version) to initialise a new Auth0 Client.

### **Firebase**

To use Firebase we need to configure the Firebase SDK first. The config object firebaseConfig can be found in src/config.js (src/config.ts for TS version).

export const firebaseConfig = {

apiKey: process.env.REACT\_APP\_FIREBASE\_API\_KEY,

appId: process.env.REACT\_APP\_FIREBASE\_APP\_ID,

authDomain: process.env.REACT\_APP\_FIREBASE\_AUTH\_DOMAIN,

databaseURL: process.env.REACT\_APP\_FIREBASE\_DATABASE\_URL,

messagingSenderId: process.env.REACT\_APP\_FIREBASE\_MESSAGING\_SENDER\_ID,

projectId: process.env.REACT\_APP\_FIREBASE\_PROJECT\_ID,

storageBucket: process.env.REACT\_APP\_FIREBASE\_STORAGE\_BUCKET

};

Go to your Firebase Console to find your project keys.

This config object is being used in src/lib/firebase.js (src/lib/firebase.ts for TS version). This is the file that initilises the Firebase application that we use for authentication.

## Switching between auth methods

The shipped project version uses by default the JWT method, but if you want to switch to Auth0 or Firebase after setup part, all you have to do is to update 2 import paths to specify the provider and it’s done.

Open src/App.js (src/App.tsx for TS version) and update the following line:

import { AuthProvider } from 'src/contexts/JWTAuthContext';

to use Auth0Context or FirebaseAuthContext.

Example:

Now open src/hooks/useAuth.js (src/hooks/useAuth.ts for TS version) and do the same by updating this line:

import AuthContext from 'src/contexts/JWTAuthContext';

to use Auth0Context or FirebaseAuthContext.

Example:

import AuthContext from 'src/contexts/Auth0Context';

## Important

If you switch to Auth0 or FirebaseAuth and the app does not load, this could mean that you forgot to setup the Firebase or Auth0 config objects.