Table of Contents

MateAdmin

A react-redux powered single page material admin dashboard. Support Monorepo using Lerna and `Yarn Workspaces.`

Demo : https://mate.redq.io/

Credits:

- React
- Redux
- Redux-Saga
- React Router 5
- React Create App
- Material Design
- Material UI
- Google Map
- React Big Calendar
- React Flip Move
- React Google Charts
- Recharts
- React Vis
- React Chart 2
- React Trend
- Echart
- React Grid Layout
- Firebase Authentication
- Auth0 Authentication
- Algolia Search

Installation

- Install Node JS
- Install npm
- Install yarn
- Install Packages & Dependencies
- yarn <cmd> (required monorepo command see below)

MateAdmin is based on Create React App, It would be better if you can check their Website https://create-react-app.dev

too. There are lot of tricks that can help your app, like API connection, Deployment etc.

Installing Node & NPM:

To work with MateAdmin the first thing you need is to have Node install on your system. To make sure you have already Node js installed on your system you may follow the below instructions :-

As Node will make sure you have node and npm commands are available via command line, just run the below command on your terminal

```
node -v

npm -v
```

On successful installation, it will print out the respective versions. make sure you have al the latest stable version install to get better performance.

Note that if you find the npm version less than 5.0.0 you need to update it to the latest version using the below command. you may need to use `sudo` to grant permission

```
npm install npm@latest -g

or

sudo npm install npm@latest -g
```

Installing YARN:

You will need to Install Yarn for the Fast, Reliable, and Secure Dependency Management. Before you start using Yarn, you'll first need to install it on your system. And to make sure it running on your system with latest version run the below command

```
yarn -version

or

yarn -v
```

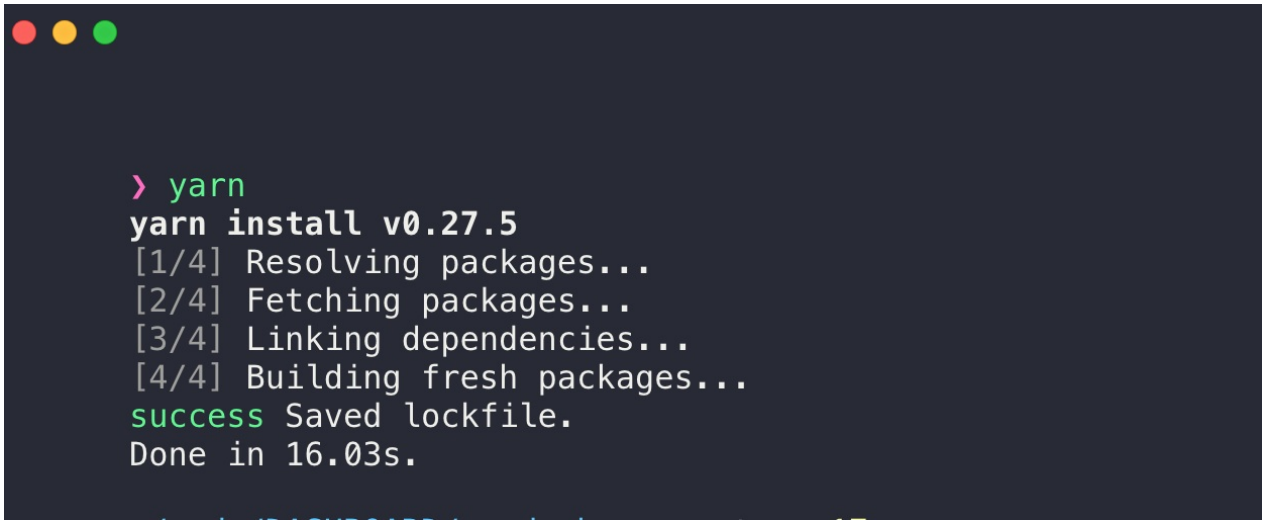
On successful installation, it will print out the version.

Installing Packages & Dependencies:

After Installing Yarn, now open the `MateAdmin` app in your terminal. Now at your terminal In the root directory of `MateAdmin` app just run

```
yarn
```

it will download all the necessary packages and dependencies in the node_modules folder.



yarn start:

Now to start the `MateAdmin` app all you need to do is to run the below command in you terminal root directory of the `MateAdmin` app.

```
yarn start:mate-admin
```

after the compiled process completed successfully, it will show the below success commands & redirect to the `http://localhost:3000/` of your browser where you will find the login screen of the `MateAdmin` app.

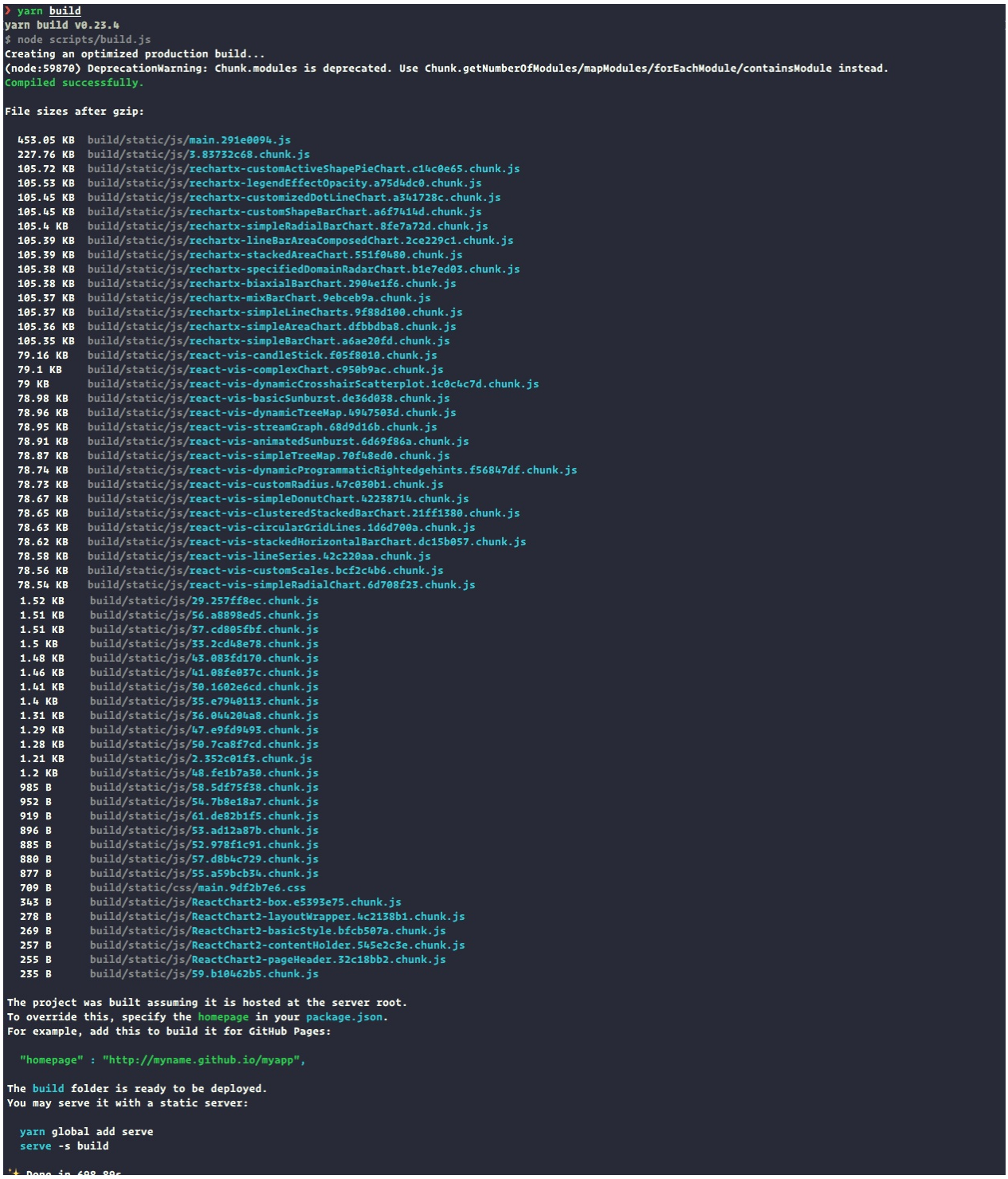
If you want to run this mateAdmin in production, Then follow the below settings :

yarn build:

To create an Optimized Product Build of the Mate admin app. you will need need to do is to run the below command in you terminal root directory of the `MateAdmin` app.

```
yarn build:mate-admin
```

after sometime when it build the production version successfully you will be notified via the terminal.



To run other packages see below commands:

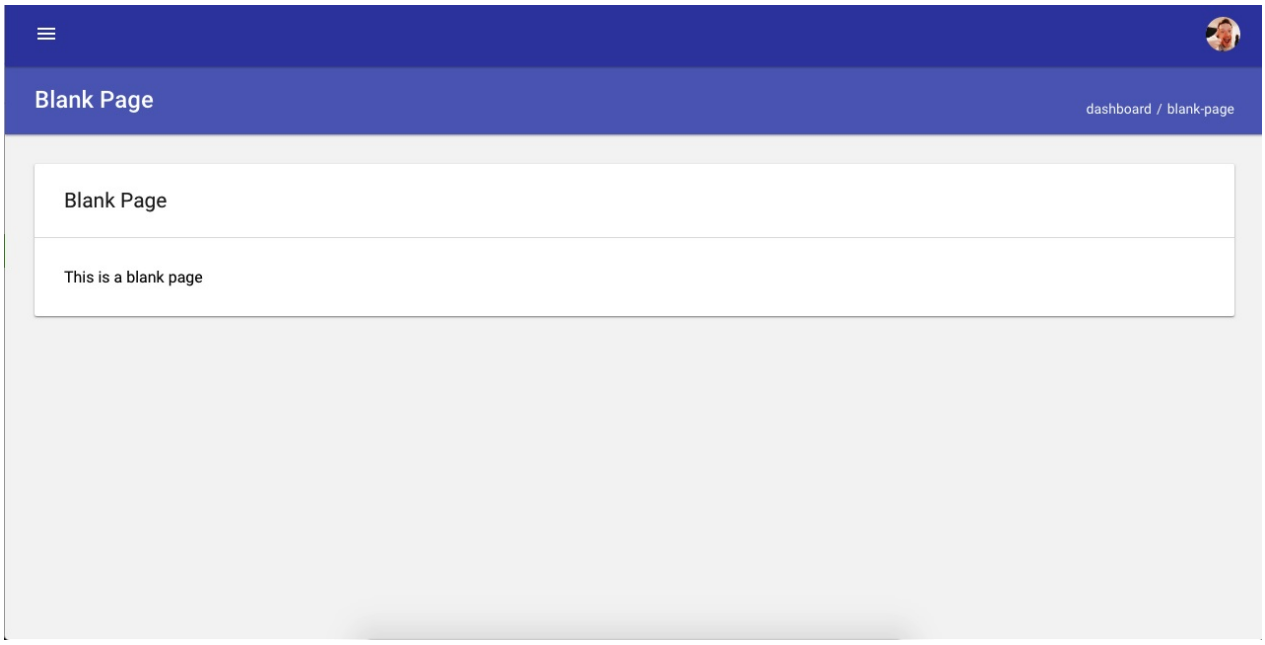
```
  "scripts": {
    "clean": "lerna clean --yes && rimraf node_modules",
    "clean:build": "lerna exec -- rimraf \"{.next,dist,out,build,.docz,.cache, public}
\"",
    "start:mate-admin": "yarn workspace @mate/admin run start",
    "start:mate-boilerplate": "yarn workspace @mate/boilerplate run start",
    "start:mate-servers": "yarn workspace @mate/servers run start",
    "build:mate-admin": "yarn workspace @mate/admin run build",
    "build:mate-boilerplate": "yarn workspace @mate/boilerplate run build",
    "build:mate-servers": "yarn workspace @mate/servers run build",
    "serve:mate-admin": "yarn workspace @mate/admin run serve",
    "serve:mate-boilerplate": "yarn workspace @mate/boilerplate run serve",
    "serve:mate-servers": "yarn workspace @mate/servers run serve"
  },
```

Getting Started

In this section, We have discussed about some quick tips and how you can create your own page by creating Public and Private Routes.

Starting From a Blank Page

If you want to start working on a blank page just try to use the Mate boilerplate. You will have the chance to start from the scratch.

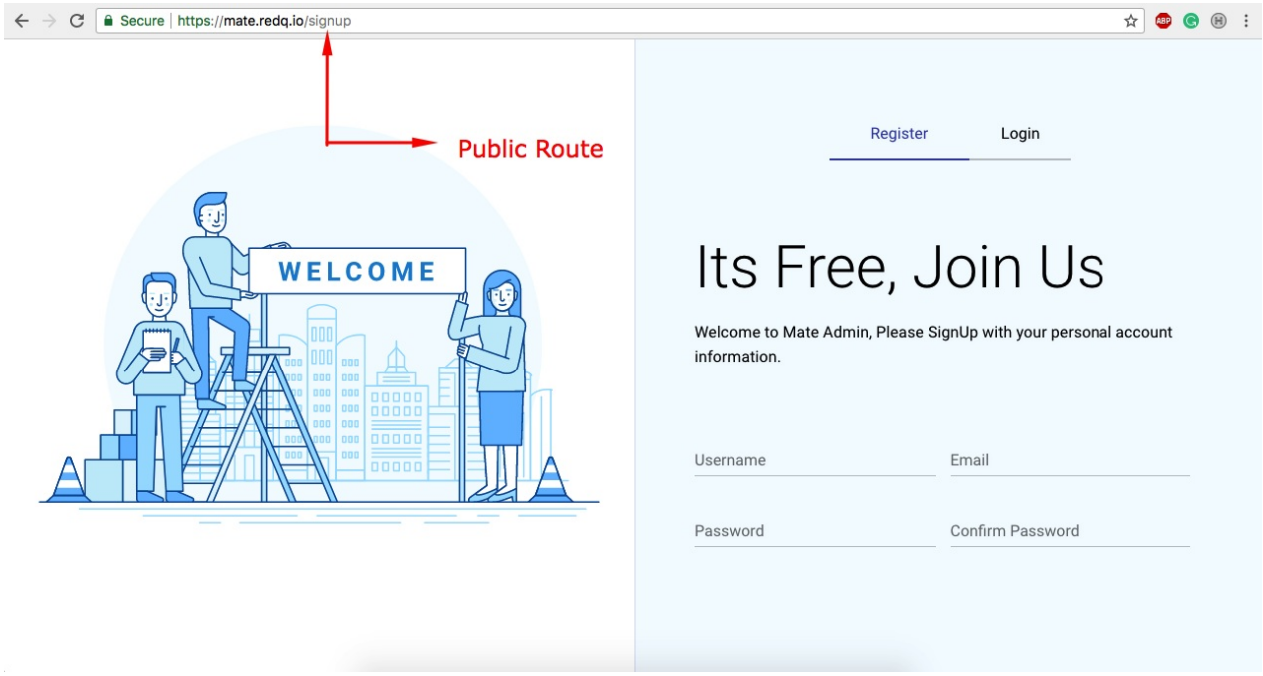


If you want to create a new page, Then you can create it by two ways.

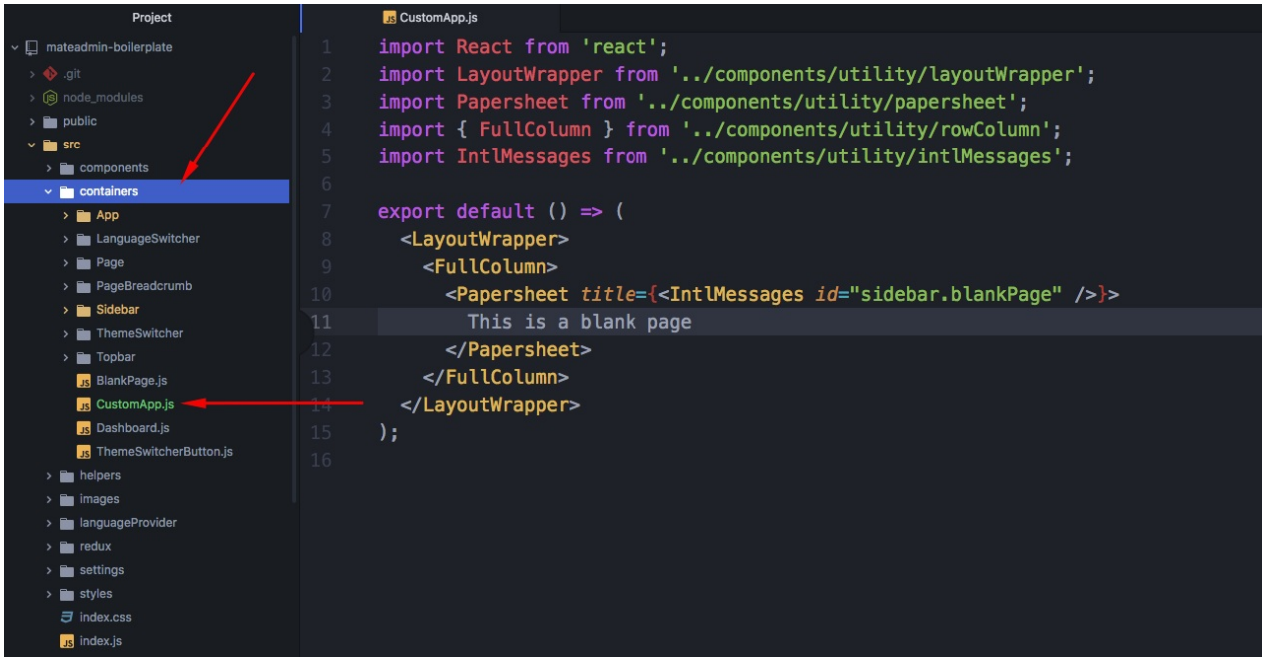
1. Public Route
2. Private Route

If you want to access the page Without any authentication, Then you have to use the public Route.

For example, In the Mate Dashboard The signIn Page, The SignUp page , 404 page, Forgot Password etc pages are build by using private route. Because these pages don't need any authentication to access.



To create a public page or route, At first create a file named CustomApp.js(or in any specific name) in src->container folder.

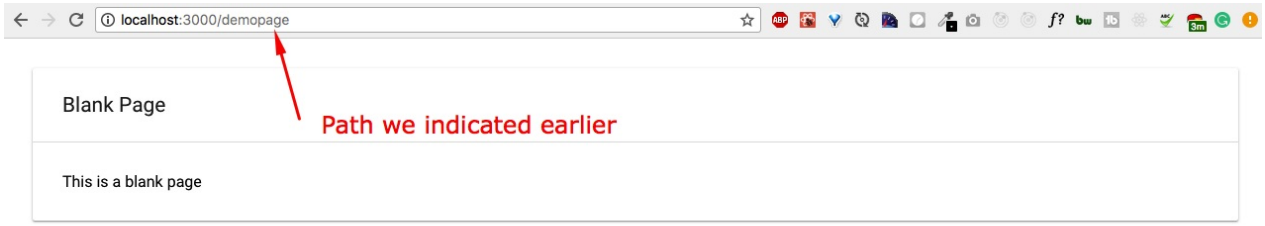


Put this demo code for buliding a blank page

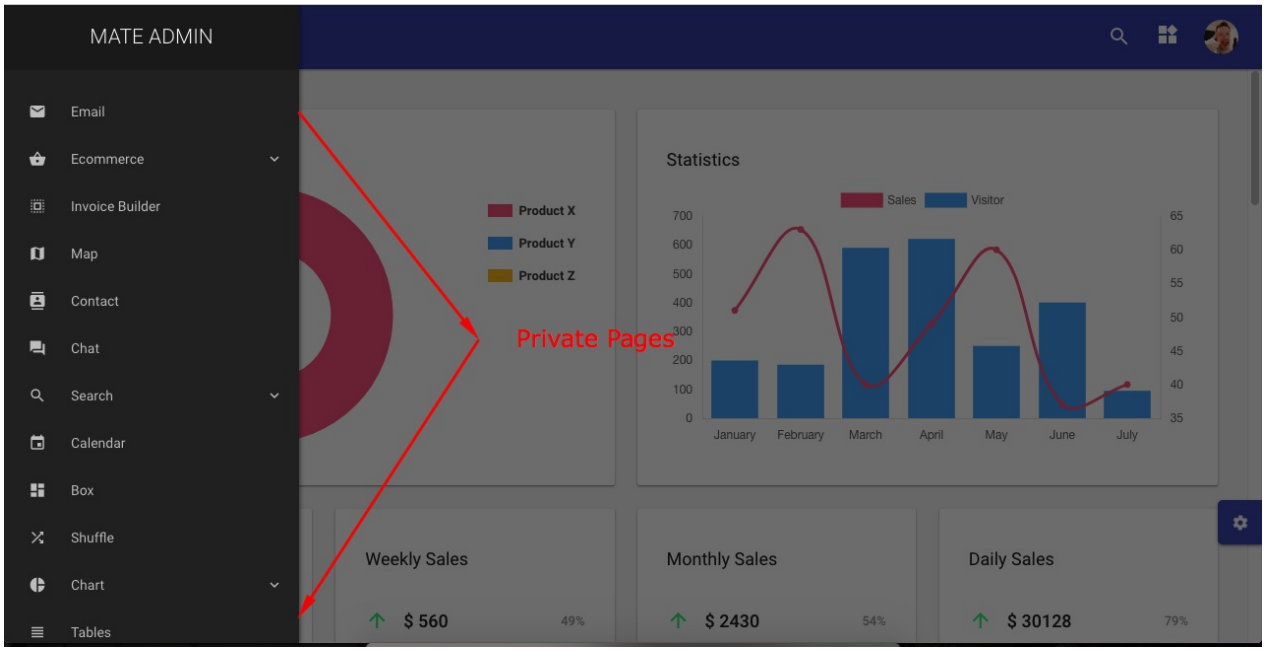
```
import React from 'react';
import LayoutWrapper from '../components/utility/layoutWrapper';
import Papersheet from '../components/utility/papersheet';
import { FullColumn } from '../components/utility/rowColumn';
import IntlMessages from '../components/utility/intlMessages';
export default () => (
<LayoutWrapper>
<FullColumn>
<Papersheet title={<IntlMessages id="sidebar.blankPage" />}>
This is a blank page
</Papersheet>
</FullColumn>
</LayoutWrapper>
)
```

After that, You have to go at the src-> router.js file to show the routes and indicating the path. Look at this screenshot

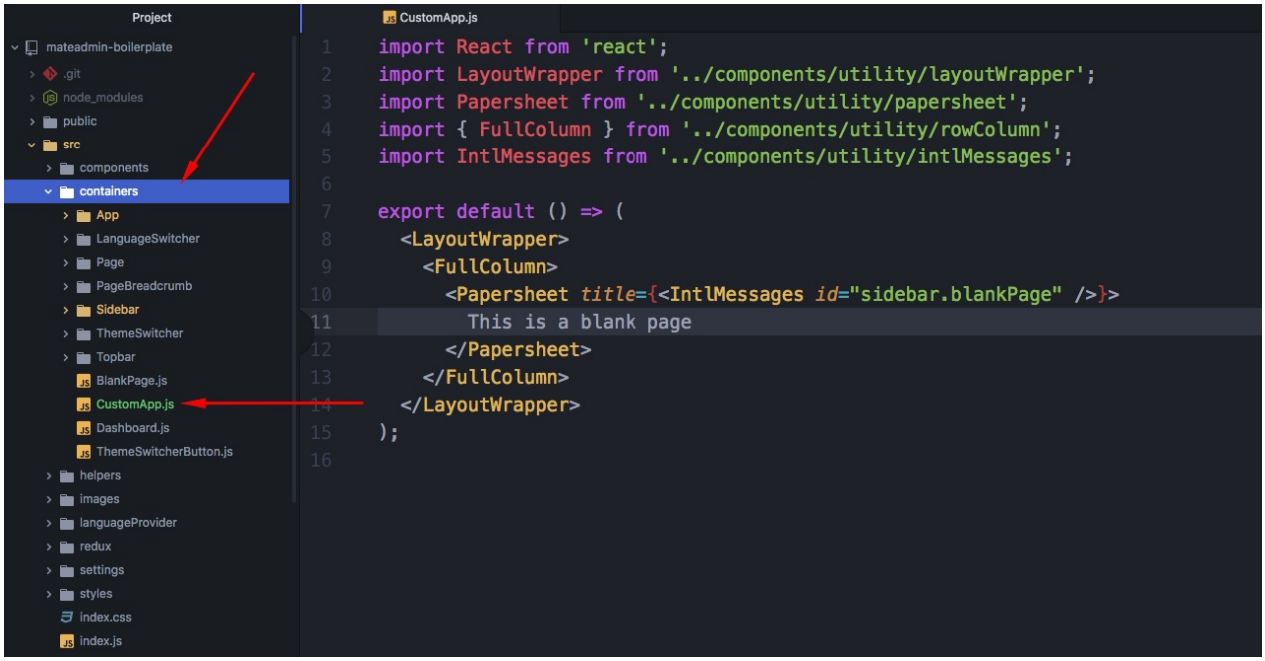
As it is a public page, Then you can access it by http://localhost:3000/demopage



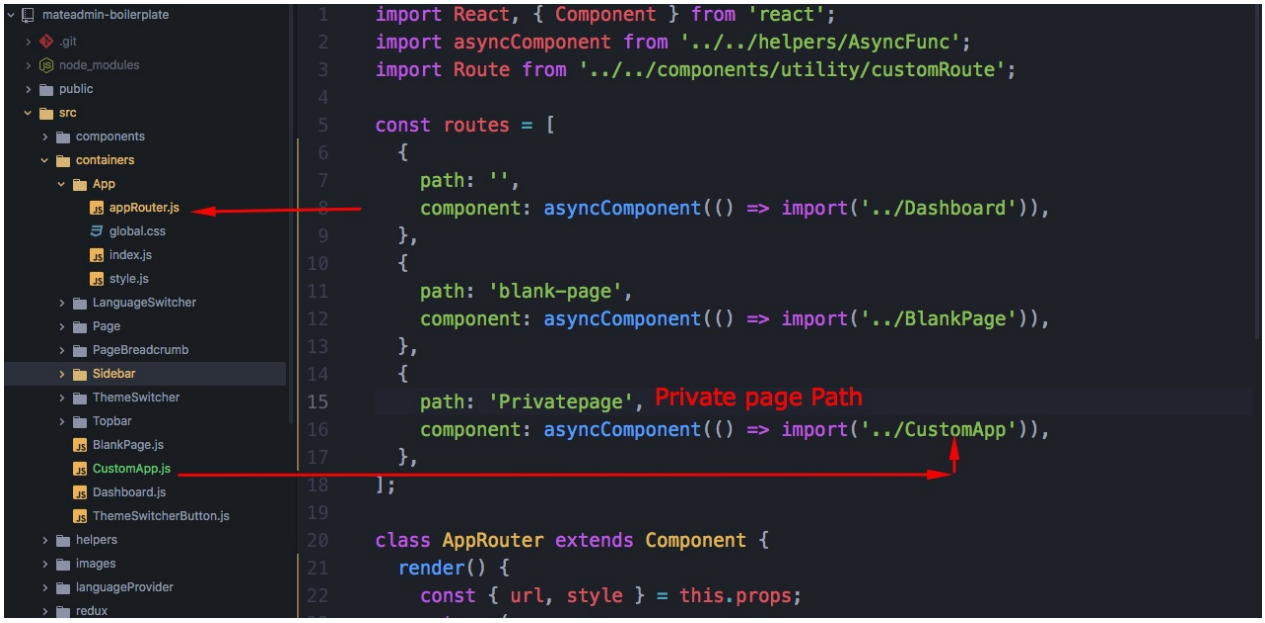
If you want to show your page in the dashboard or left sidebar, Then you have to use Private Route. Because to access your private page, you will need the authentication.



To create a private page or route, At first create a file named CustomApp.js(or in any specific name) in src->container folder.

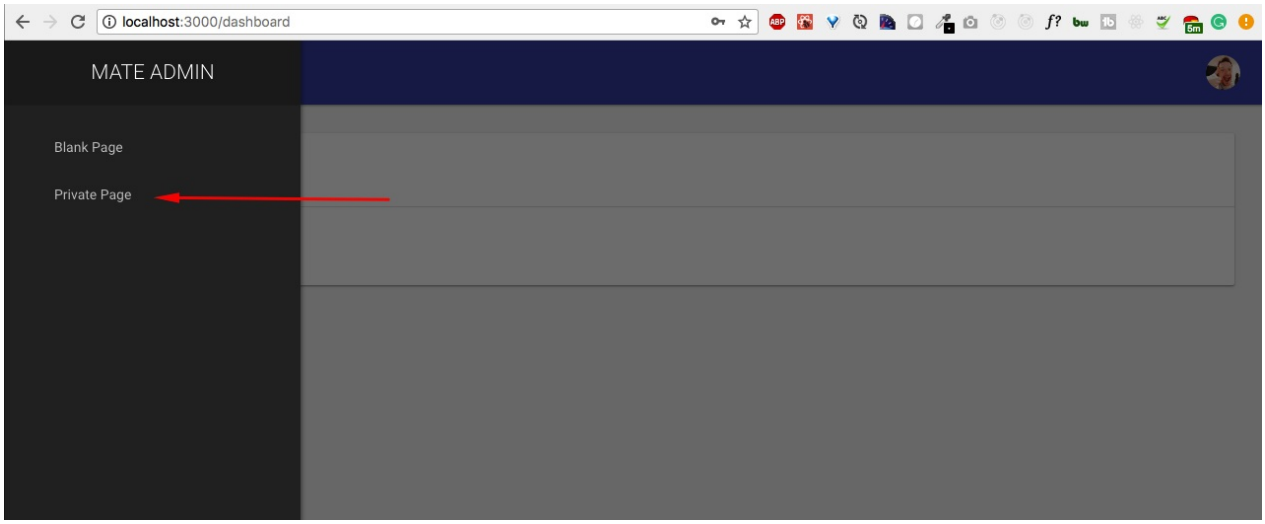


Then you have to go at src->container->app->appRouter.js and Then look at the screenshot

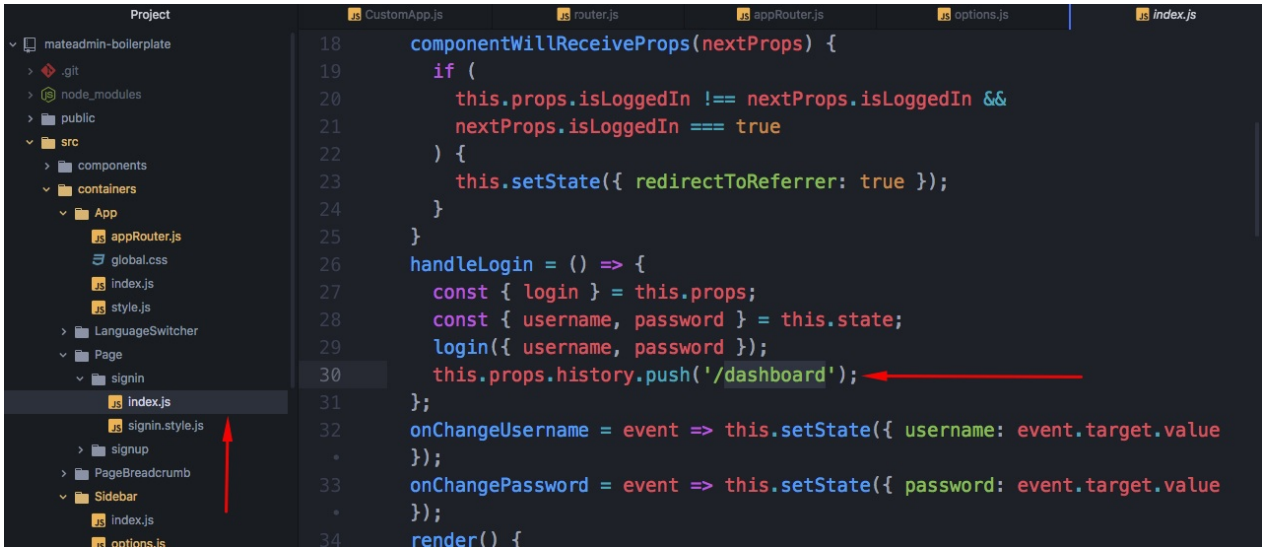


For showing it in the sidebar, You have to go at sidebar->option.js .

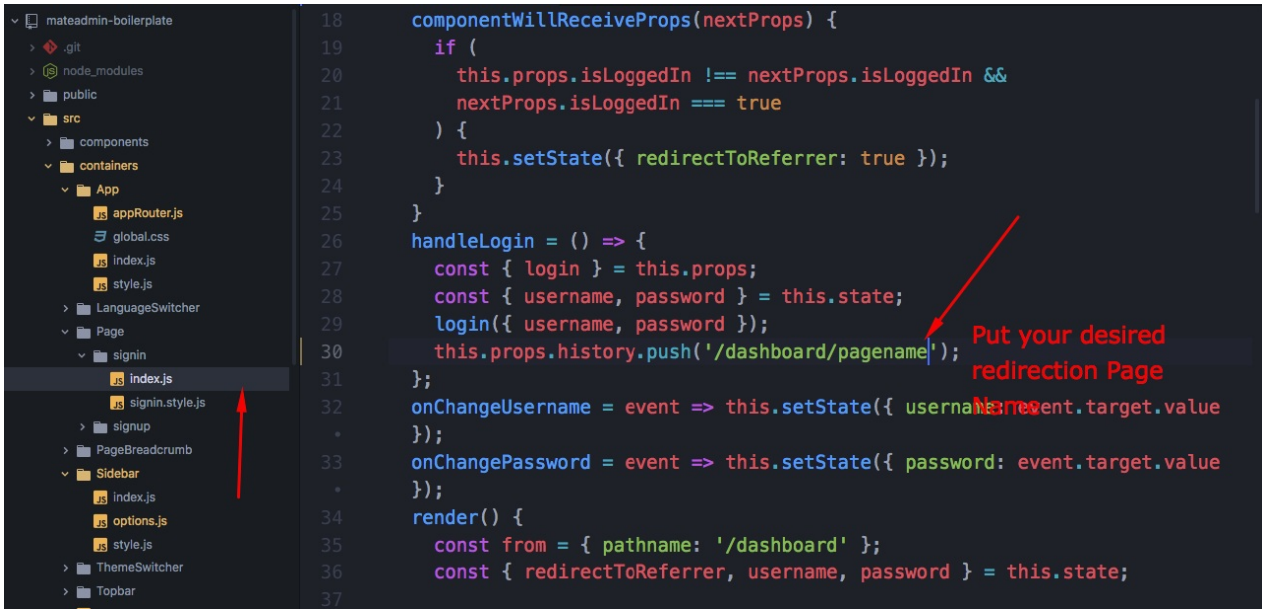
Then in the dashboard sidebar option, you will see like this



In the signIn and SignUp Page, When you click on the Login or SignUp button, Generally it will redirect into Dashboard.



If you want to redirect to the another page rather than dashboard, Then you have to do like the screenshot



Authentication

We provided two third party Auth integration in our Project. There are

1. Firebase
2. Auth0

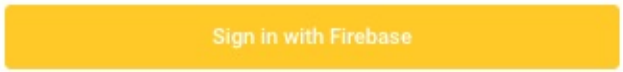
A brief description about them is given bellow.

Firebase

Firebase-Doc

`Folder path: /src/components/firebase/`

If you want a login button like the following image given bellow.



On the `onclick` function of the button You can render a form with two input fields(one for Email and one for Password). And passing those credentials through the Firebase Api, a user can be get signed up or singed in. The following code is connects the User inputs to the Firebase Api

`Firebase.login(Firebase.EMAIL, { email, password })`

And you can use basic Javascript promise methods like then() or catch() methods and use the Firebase returned data to do your desired operations like the following code

```
    Firebase.login(Firebase.EMAIL, { email, password })
      .then(result =>
{
        // Do something
      })
      .catch(error =>
{
        // Handle your error precisely
      })
```

To Use the Firebase Api you need to configure your app to theFIrebase Official Websitefirst. And put your app credentials to the config file of our app.

Path to the config file: `/src/settings/index.js`

The following are the important Credentials you must provide in order to make Firebase Authentication work.

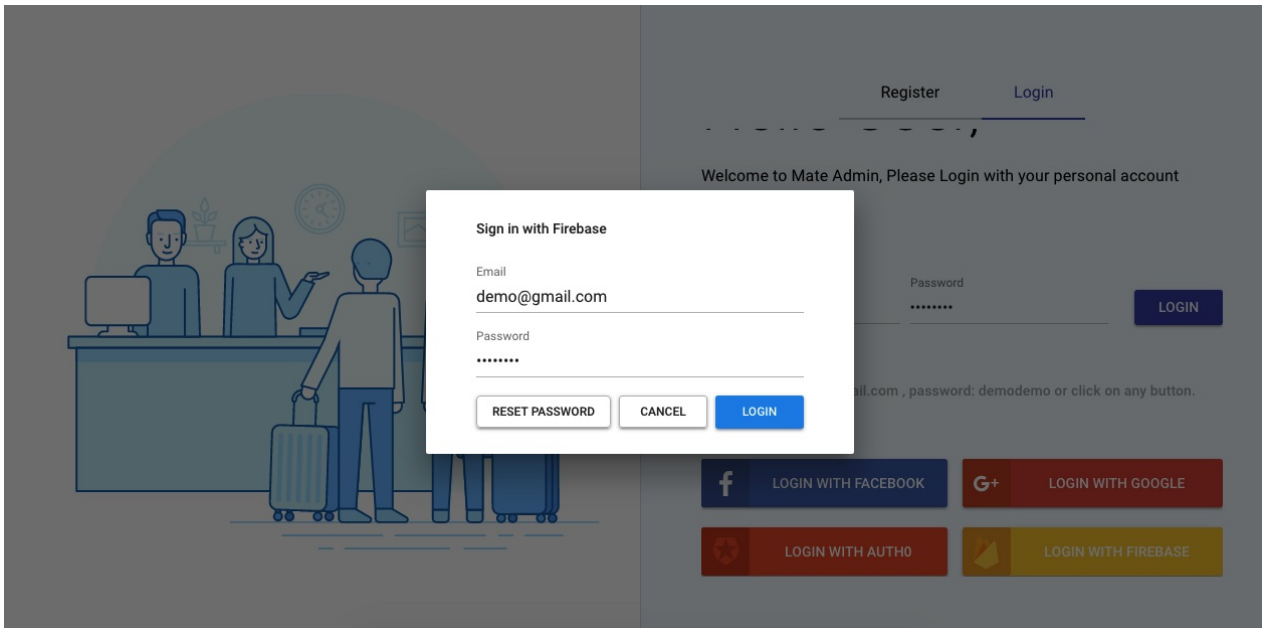
Keys
apiKey
authDomain
databaseURL
projectId
storageBucket
messagingSenderId

Currently We are providing Firebase authentication through Email and password only. You can also integrate Social Logins (Facebook, Google, Github, Twitter) with Firebase. The code will be found on the following Path: `Folder path: /src/helpers/firebase/`

And the code is like the following.

Authentication Type	Helper function	Manual Code
Basic(Email, Password)	firebaseAuth().signInWithEmailAndPassword()	Firebase.login(Fireba email, password })
Facebook	firebaseAuth().FacebookAuthProvider()	Firebase.login(Fireba
Google	firebaseAuth().GoogleAuthProvider()	Firebase.login(Fireba
Github	firebaseAuth().GithubAuthProvider()	Firebase.login(Fireba
Twitter	firebaseAuth().TwitterAuthProvider()	Firebase.login(Fireba

After all those Works clicking the Login with Firebase button A prompt Window like the following will open

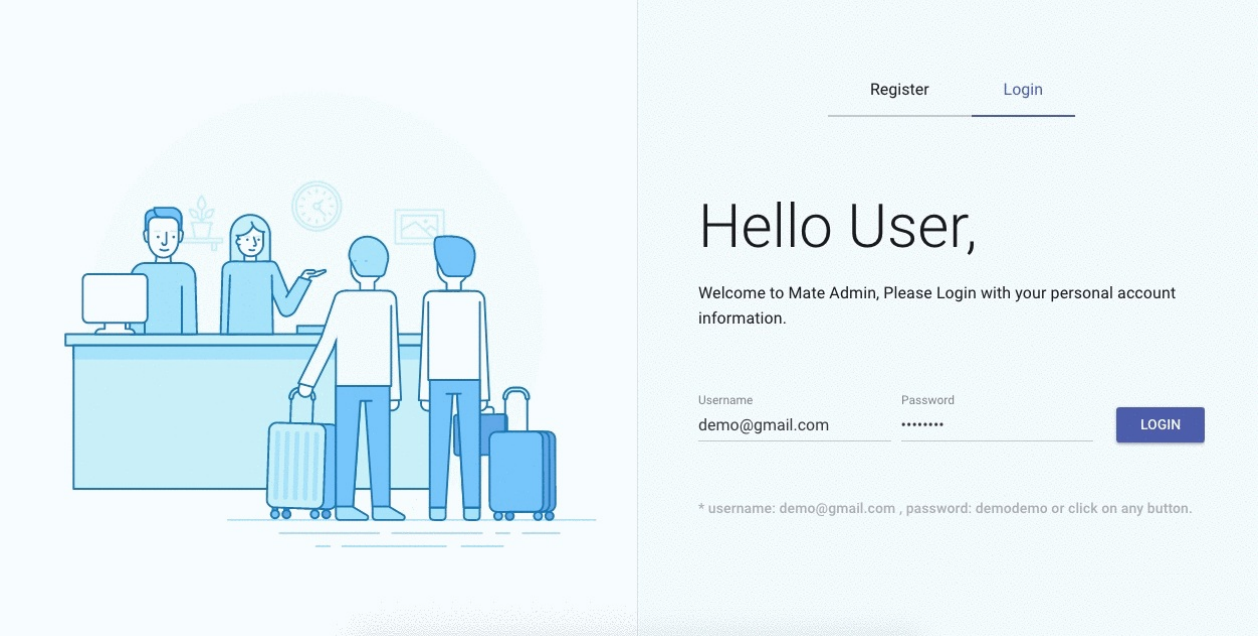


Auth0

Auth0-official-website

`Folder path: /src/helpers/auth0/`

If you want a login button like the following image given bellow.



Just need to Place a button and On the onclick function of the button You can render the following functions provided by Auth0 itself.

Function	Details
new Auth0Lock()	Instantiating Lock
getUserInfo()	Obtaining the profile of a logged in user
show()	Showing the lock widget
on()	Listening for events
logout()	Log out the user

We are using the firebaseLockwidget.

In the folder path `/src/helpers/auth0/` there is a Auth0Helper class that uses all the functions.

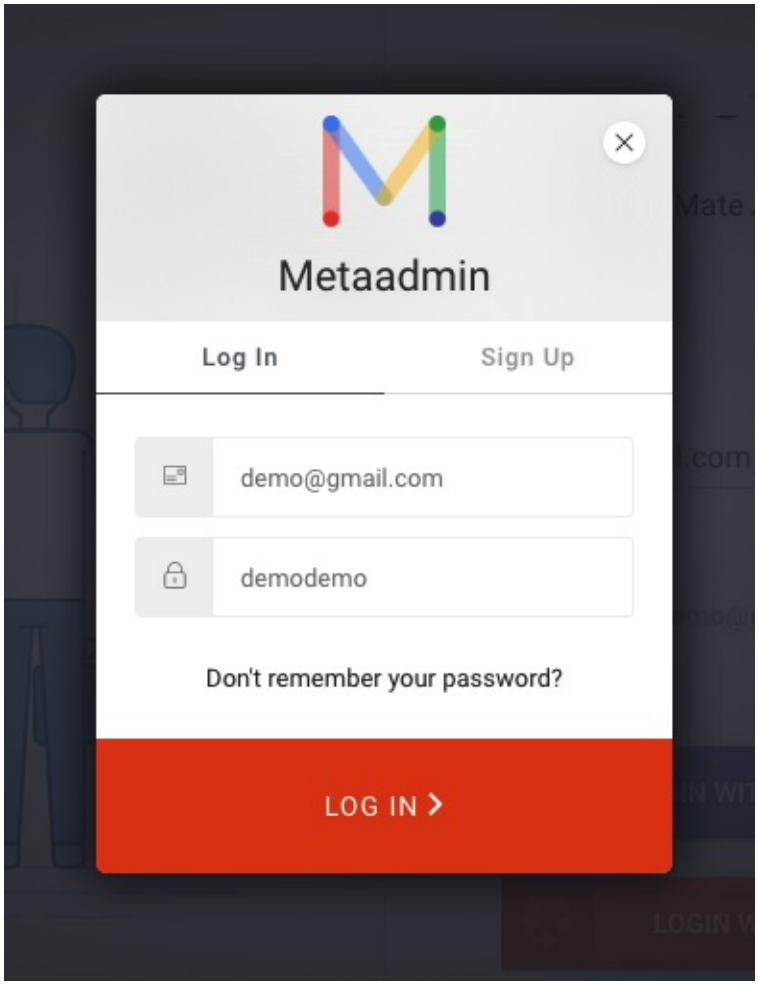
To Use the Firebase Api you need to configure your app to the Auth0-Official documentation first. And put your app credentials to the config file of our app.

Path to the config file: `/src/settings/index.js`

The following are the important Credentials you must provide in order to make Firebase Authentication work.

Keys
clientID
authDomain

After all those Works clicking the Login with Auth0 button A prompt Window like the following will open

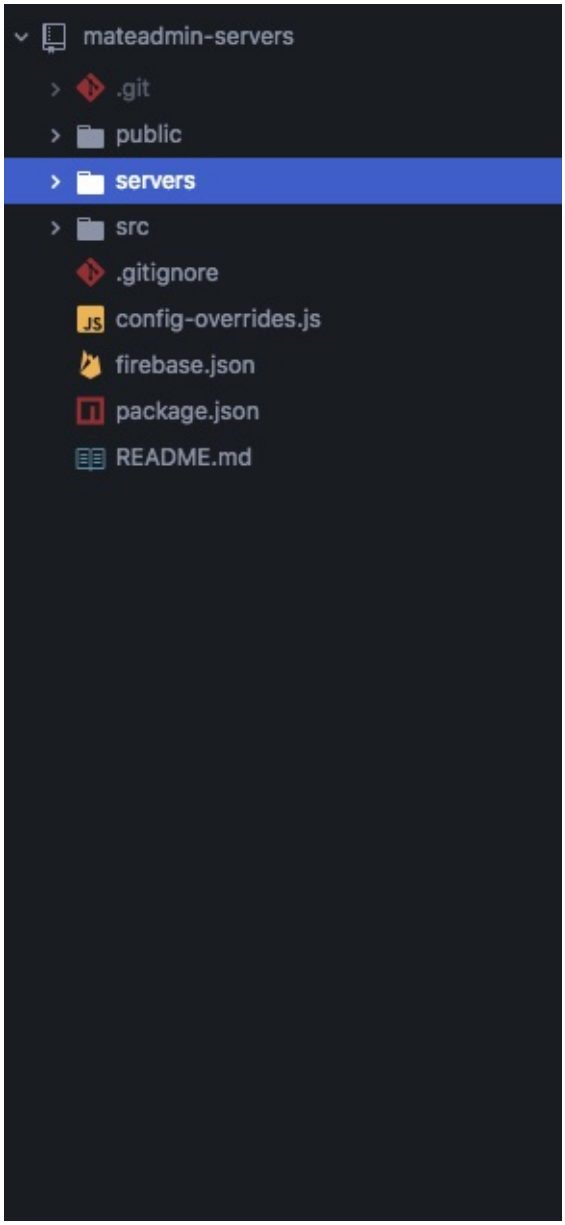


Express JWT Implementation

Server Side

In this section we will show you how to implement JSON Web Token (JWT) using Node Express framework.

First, Open the mateadmin-servers folder in your favourite editor. where you will find the below folder structure



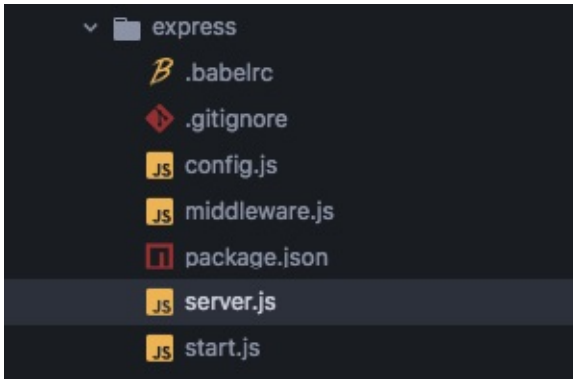
You can see we have provided and extra `server` folder where we have implemented our **Express JWT Authentication.**

To Run the Server follow the below steps,

i) Run `yarn` at the terminal in the mateadmin-servers-> `server directory. (This will bring all the necessary node modules)`

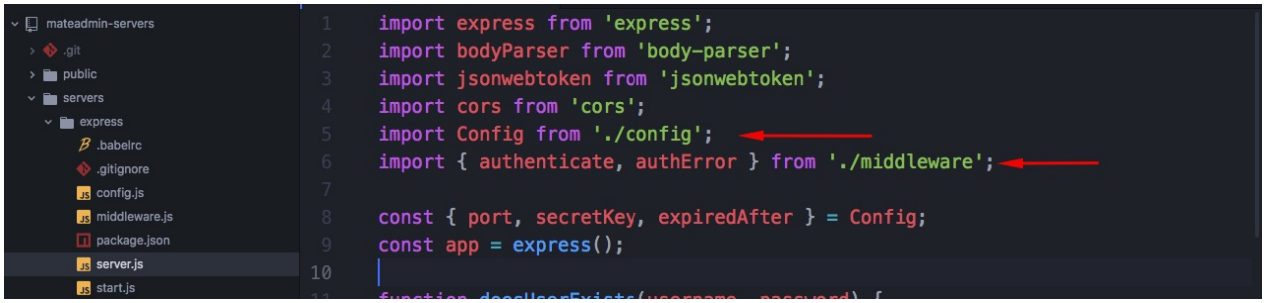
ii) Then Run the `yarn start` command

Lets see what's inside the `server folder` & what's happening there:



From the above snap you can see that theres a `server.js & `middlewire.js` file where we have implemented the necessary server & middleware coding for the JWT implementation.

If you open the `server.js` file in the first part you will see we have declared some necessary requirement,



at line 5 & 6 we have required the configuration from the `config.js` file where have declared the `port` , `secret key` , `expiration time` & `middleware.js` file where we have implemented the **Token Authorization**, **Token Expiration** and **Error checking**.

let's have a look at the `middleware.js` file

middleware.js

```
import jsonwebtoken from 'jsonwebtoken';
import Config from './config';

const { secretKey } = Config;

const authenticate = (req, res, next) => {
  const token = req.headers.authorization || '';
  jsonwebtoken.verify(token, secretKey, (error, decoded) => {
    if (error) {
      next({ error: 'token varified failed' });
    } else {
      const { expiredAt } = decoded;
      if (expiredAt > new Date().getTime()) {
        next();
      } else {
        next({ error: 'token expired' });
      }
    }
  });
};

```

The `authenticate` function checks the `token authorization` & `token expiration` . while the `authError` function is doing the error checking if something went wrong.

From the `server.js ,` you can see that the two middlewere `authenticate & authError` is used by our `app` at the `\api` route. so whenever you want to access this route you have to generate the valid token.

Now, let's have a look at the later part of the of the `server.js` file

server.js

```
app.post('/api/login', (req, res) => {
  const { username, password } = req.body;
  const response = {};
  // You can use DB checking here

  if (doesUserExists(username, password)) {
    response.token = jsonwebtoken.sign(
      {
        expiredAt: new Date().getTime() + expiredAfter,
        username,
        id: 1
      },
      secretKey
    );
  } else {
    response.error = 'Not found';
  }
  res.json(response);
});
app.use('/api/secret', [authenticate, authError]);
app.post('/api/secret/test', (req, res) => {
  res.json({
    status: 200,
    message: 'succcesful'
  });
});
```

you can see here we have implemented the `/login` related functionality at the beginning of this part and later we have created a

demo testing post request at `/api/demoTesting/` route.

Let's see what we have done at the `/login`

Here we have generated the `token` in the response when the login is successful.

Client Side

For the client part if you are already familiar with our Mate codebase than all you have to do is to check the below file,

mateadmin-servers `/src/helpers/authHelper.js file` ,

where you can find the necessary client side coding,

you can also check the,

mateadmin-servers `/src/redux`

where we have done the reducer & saga related code for the jwt authentication.

If you are not familiar with the Mate codebase than we suggest you to check our previous section of this documentation.

Deployment

`yarn buid` or `npm run build` creates a `build` directory with a production build of your app. Set up your favourite HTTP server so that a visitor to your site is served `index.html`, and requests to static paths like `/static/js/main.<hash>.js` are served with the contents of the `/static/js/main.<hash>.js` file.

If you want to deploy with Create React App, Please follow this link https://github.com/facebook/create-react-app/blob/master/packages/react-scripts/template/README.md#deployment

For deploying your site in firebase visit Firebase Deployment page.

For deploying your site with AWS , Please follow this link https://blog.logicalicy.com/static-react-app-in-20-mins-with-aws-baac9c5f615b

Multi Language Support.

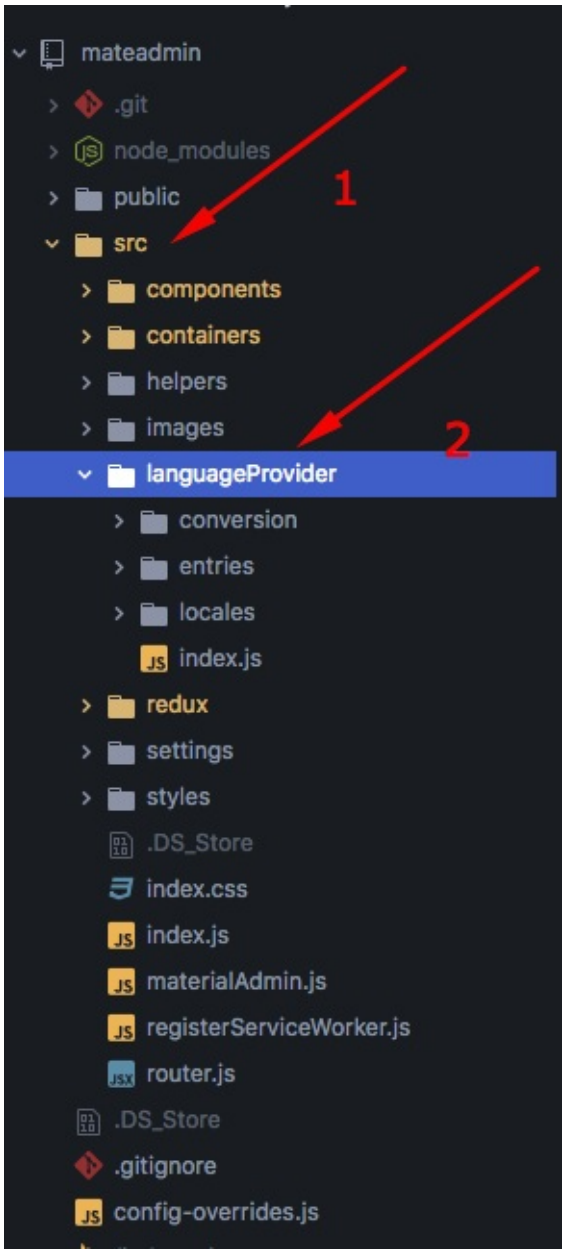
Mate supports multi language. It became very useful to people from around the world.

For multi-language conversion Mate use https://github.com/yahoo/react-intl library.

Lets discuss the procedure step by step of how to Add/Convert into new languages.

Step 1 :

Go to Project Root folder > `src` folder > `languageProvider` folder



Open `index.js` file and Add the language name into `const AppLocale` which you want to add into Mate App.

```
const
 AppLocale = {
  en: Enlang,
  zh: Zhlang,
  sa: Salang,
  it: Itlang,
  es: Eslang,
  fr: Frlang
};
```

for example you want to Add/Convert German language.

then add `de` language code into AppLocale constant like this.

```
const AppLocale = {
  en: Enlang,
  zh: Zhlang,
  sa: Salang,
  it: Itlang,
  es: Eslang,
  fr: Frlang,
  de: GermanLang
};
```

and Add this code `addLocaleData(AppLocale.de.data);`

Step 2 :

create a new file in `src > languageProvider > entries > de_DE.js file` and paste this code.

```
import antdSA from 'antd/lib/locale-provider/de_DE';
import appLocaleData from 'react-intl/locale-data/de';
import deMessages from '../locales/de_DE.json';

const deLang = {
  messages: {
    ...deMessages
  },
  antd: antdDE,
  locale: 'de-DE',
  data: appLocaleData
};
export default deLang;
```

Step 3 :

Now, create a JSON file named `de_DE.json` in `src > languageProvider > locales` folder.

Step 4 :

This is the big step for the language conversion. To change language in every MeteAdmin components this is the general procedure. Let's discuss an **Alert box** component inside the **FeedBack** section.

Step 4.1

copy all the element from_en_US.json _file and paste this into_de_DE.json _file. Now all the text strings are listed there. Just translate every json element's value into German language.

Example :

`"sidebar.formsWithValidation": "Forms With Validation"` will be convert like this

`"sidebar.formsWithValidation": "Formulare mit Validierung"`

Step 5 :

To add this new language into Sidebar switcher option, follow this file path `src > containers > languageSwitcher > config.js` file.

add the new icon image from image folder in the top of the file like this `import italianLang from '../../image/flag/german.svg';`
Then add this new Language into config -> option array by this

```
{
  languageId: 'german',
  locale: 'de',
  text: 'German',
  icon: germanLang,
},
```

Step 6 (additional step) :+

To chose the default language of Mate go to this file `src > settings > index.js` and change this code `const language = 'english';` into `const language = 'german';`

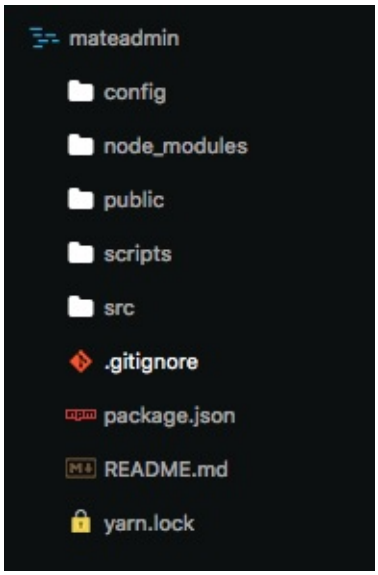
This will initiate german language as default language.

Ta-Da!!

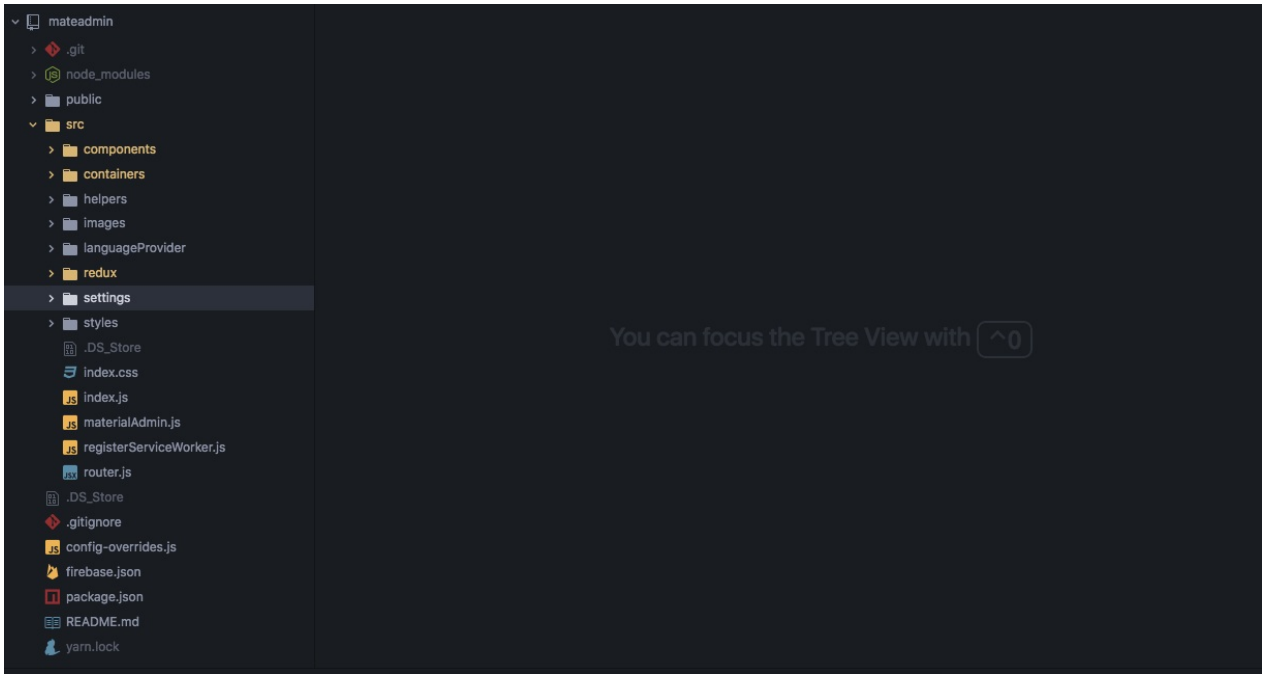
Thanks for reading & understanding the process. It's not that hard. For any inquiries or issues you can contact with us via our support portal https://redqsupport.ticksy.com/

Structure

The folder structure of MateAdmin is following like that.



Build: All the Build files are available on this folder.



node_modules: It contains all the npm packages that is used on this projects.

public: Contains public files used on the projects like menifest file, index.html file, icon files.

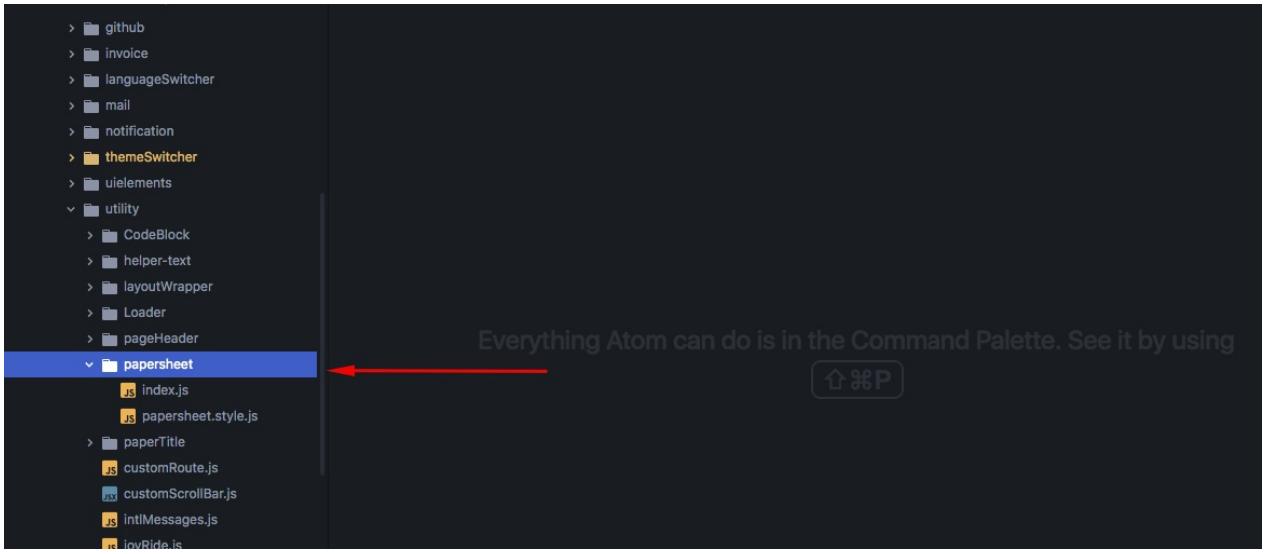
src: Contains all the codes including js, less and the image files. It has some folders inside. They are:

- components: Reusable react components
- containers: Constains all the files of the react component of the project.
- settings: General config files.
- helpers: Utility codes for the projects.
- image: Images used in the project.
- reducers: Contains the functional code of redux.
- sagas: React sagas for handling async request.
- selectors: React selectors
- store: Redux Stores
- styles: Less code files.

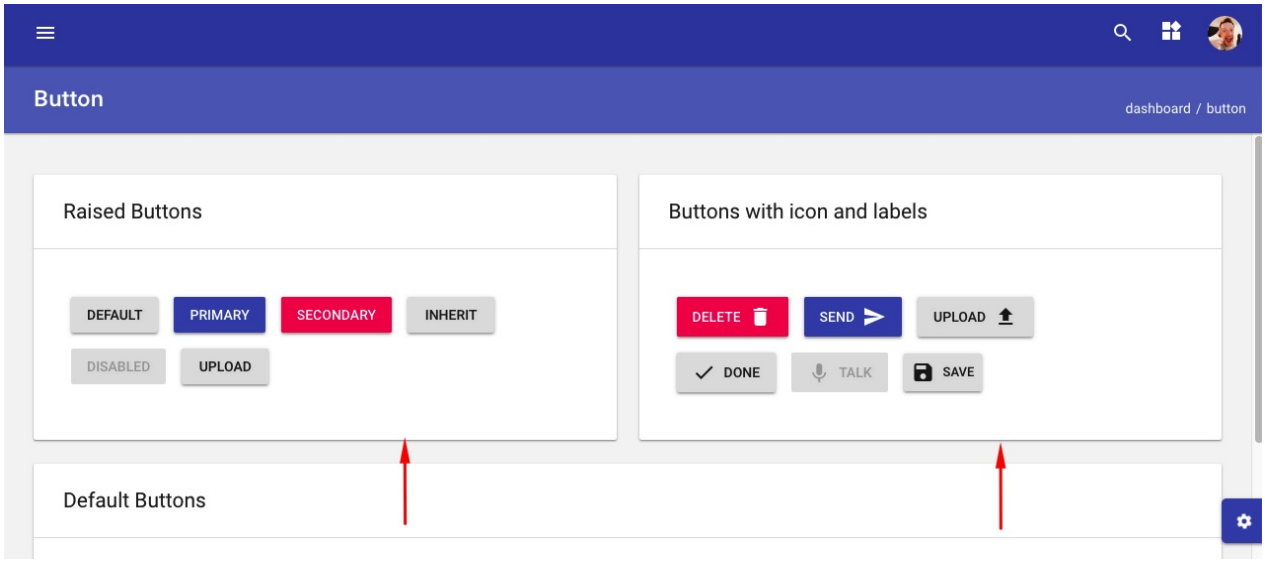
package.json: Contains all the informations about the project like third party packages, scripts etc .

server.js: The file fires up the node server.

To add a paper and shadow like material component you need to add papersheet from 'components/utility/papersheet' folder. It add material feel in your component.



And the output box should be like this :



> We have used styled component in our Isomorphic. Credit goes to the libraries like styled-components and styled-theme.

To add a stylesheet for your component you need to create a 'style.js' file and import 'styled' from 'styled-components' & import 'palette' from 'styled-theme'.

```
import styled from 'styled-components';

import { palette } from 'styled-theme';
```

Then add 'const' name for the div or existing component & write your css inside it. Then 'export' the const name and 'import' it in your component 'index.js' file and use it. For more information please check the styled components' documentation https://www.styled-components.com/docs .

How to change a Theme Color:

To change a theme color or add a new color for your site all you need to go to the 'settings/themes/themeDefault.js' file & add your color there. Remember 'palette' is a function and it's 1st parameter is the name of the color array and 2nd parameter is the index of the color array.

Here is an example of how you use 'palette' to add a color in your component.

```
import styled from 'styled-components';
```

```
import { palette } from 'styled-theme';
```

``const Button = styled.button```

```
color: ${palette('grey', 9)};
```

```
`export default Button;`


**To style an existing component -**


`import styled from 'styled-components';`

`import { palette } from 'styled-theme';`

`import MatButton from "material-ui/Button";`

``const Button = styled(MatButton)
```

```
color: ${palette('grey', 9)};
```

```
`;

export default Button;
```

To use it in your component -

```
import Button from "./style.js";

.

.

.

.

return(

<Button>Hello</Button>

)
```

MateAdmin supports `AsyncComponent` . All the components are being used in this app are based on `React Router 4` . `asyncComponent` provide you the facility to `Asynchronously` load components and feed them into `Match on route changes` .

you will find the `asyncComponent` related function and necessary code from the below file

> To find out the code of `asyncComponent` related router, please go to your-apps-root-path/src/containers/App/AppRouter.js
>
> To find out the code of `asyncComponent` related function, please go to your-apps-root-path/src/helpers/AssyncFunc.js

The Basic Route Component for The Widgets,

```
<Route
    exact
    path="/"
    component={asyncComponent(() => import('./containers/Page/signin'))}
/>
```

The `asyncComponent` function

```
export default function(loader, Loader = LoaderPlaceholder) {
  const Loading = props => {
    if (props.isLoading) {
      if (props.timedOut) {
        return <div>Loader timed out!</div>;
      } else if (props.pastDelay) {
        return <Loader />;
      } else {
        return null;
      }
    } else if (props.error) {
      return <div>Error! Component failed to load</div>;
    } else {
      return null;
    }
  };
  return Loadable({
    loader,
    loading: Loading,
  });
}
```

AsyncComponent

MateAdmin supports `AsyncComponent` . All the components are being used in this app are based on `React Router 4` . `asyncComponent` provide you the facility to `Asynchronously` load components and feed them into `Match on route changes` .

you will find the `asyncComponent` related function and necessary code from the below file

> To find out the code of `asyncComponent` related router, please go to your-apps-root-path/src/containers/App/AppRouter.js
>
> To find out the code of `asyncComponent` related function, please go to your-apps-root-path/src/helpers/AssyncFunc.js

The Basic Route Component for The Widgets,

```
<Route
    exact
    path="/"
    component={asyncComponent(() => import('./containers/Page/signin'))}
/>
```

The `asyncComponent` function

```
export default function(loader, Loader = LoaderPlaceholder) {
  const Loading = props => {
    if (props.isLoading) {
      if (props.timedOut) {
        return <div>Loader timed out!</div>;
      } else if (props.pastDelay) {
        return <Loader />;
      } else {
        return null;
      }
    } else if (props.error) {
      return <div>Error! Component failed to load</div>;
    } else {
      return null;
    }
  };
  return Loadable({
    loader,
    loading: Loading,
  });
}
```