LEADERBOARD APP

27-12-2020 | React application

LeaberBoard application is a simple webapp where one can respond to the quizzes, they were allotted to or one can create their own with an account of elevated permissions.

Developed by:

Mahesh Kancharla,
B.Tech 3rd Yr, CSE,
Kalasalingam Academy of Research and Education, India.
Intern. at BEST ENLIST,
9918004046@klu.ac.in,
6303962484.

Disclaimer:

This Project was developed as a React Internship Project at BEST ENLIST, Chennai, TamilNadu.

LeaderBoard App

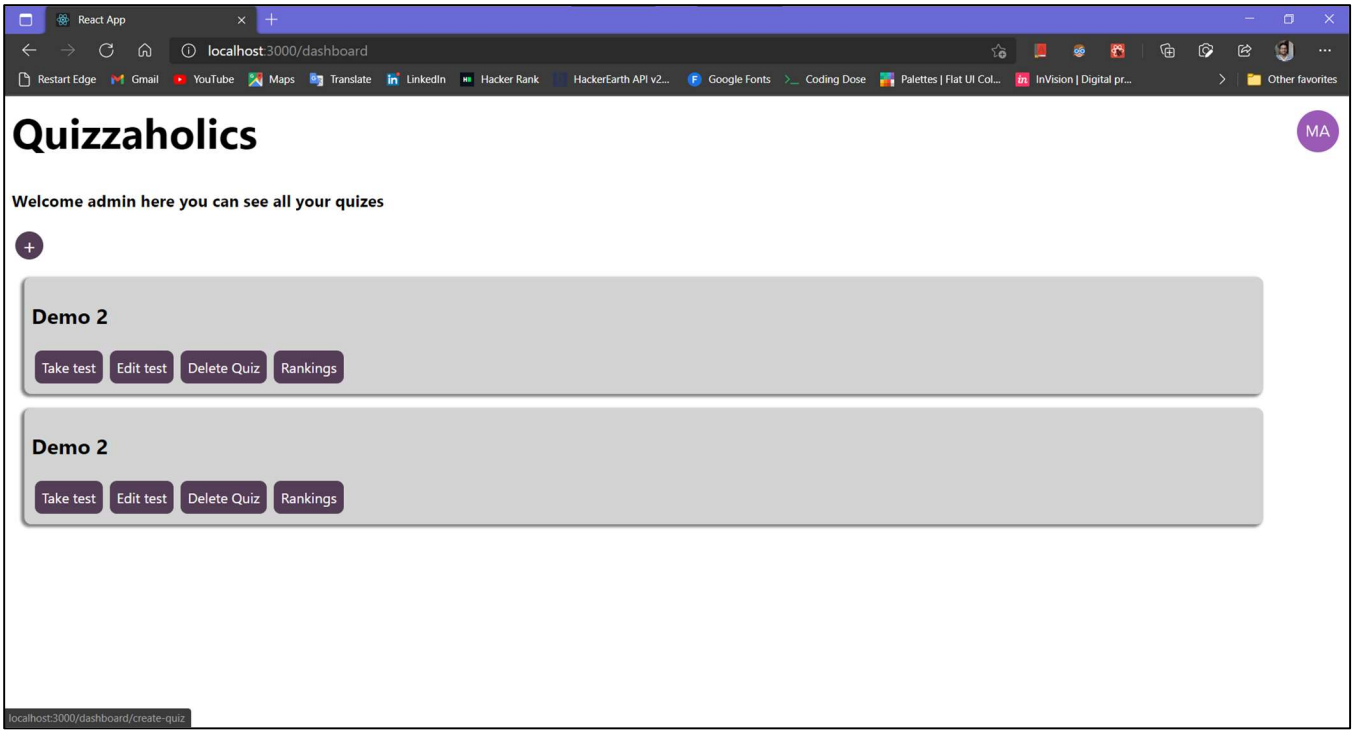
REACT APPLICATION

INTRODUCTION:

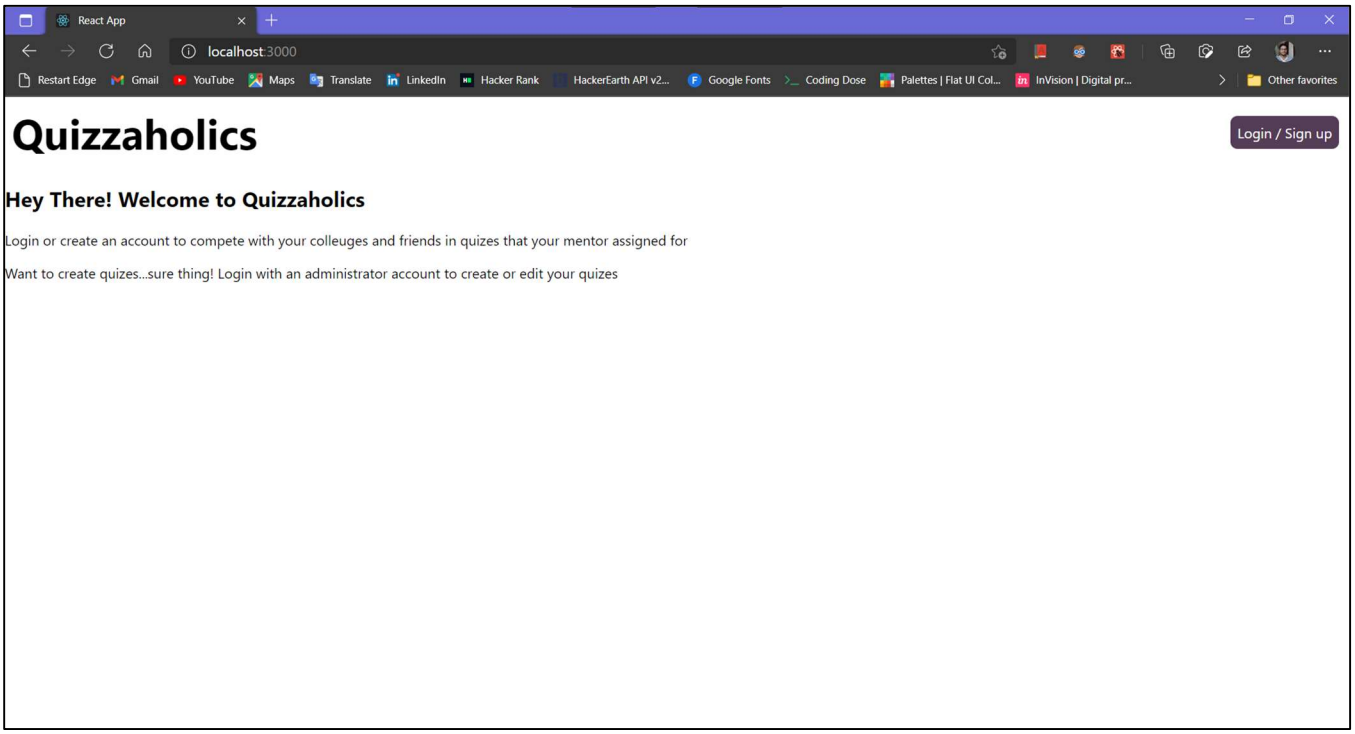
The Quizzoholics website (leader-board-app) is used basically for taking quizzes or making quizzes. It is secured as the users have to authenticate and should be authorized to use the webiste for managing their quizzes or attending them.

SNEEK PEEK INTO THE WEBSITE:

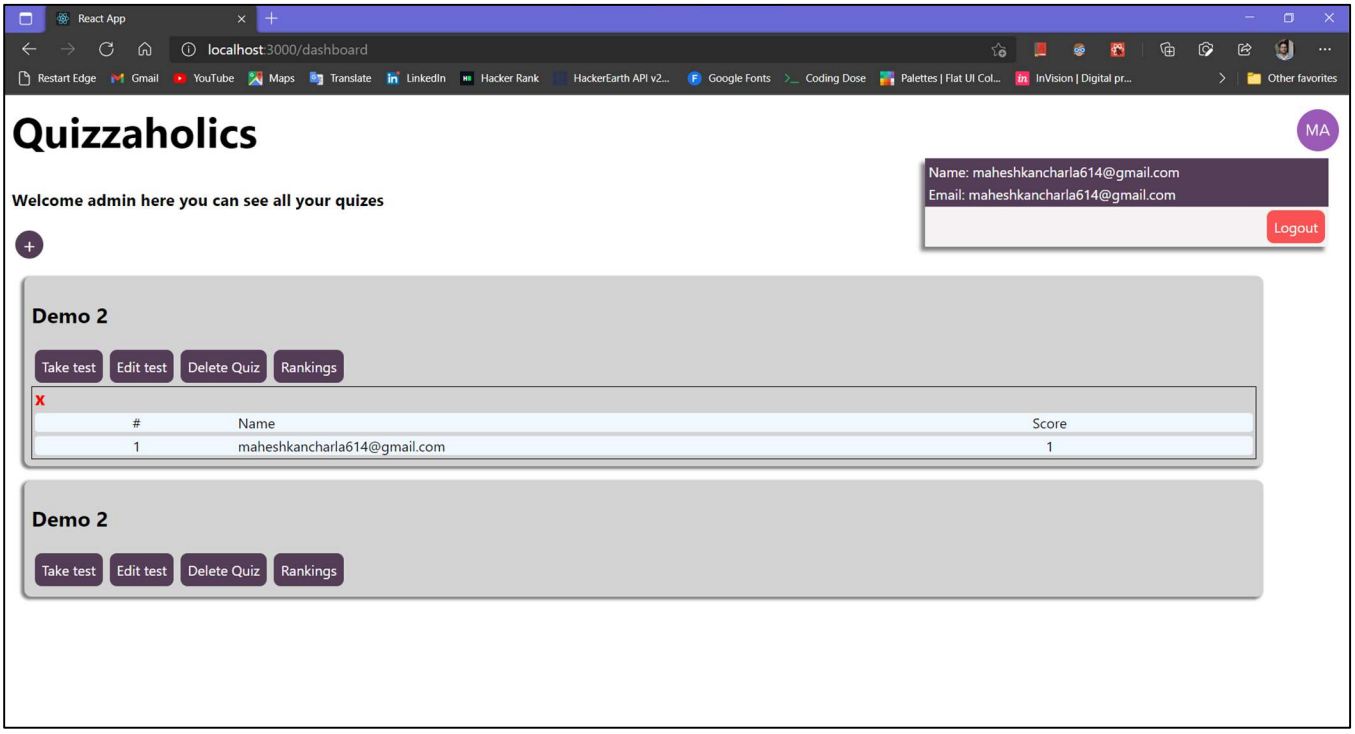
- Admin Page



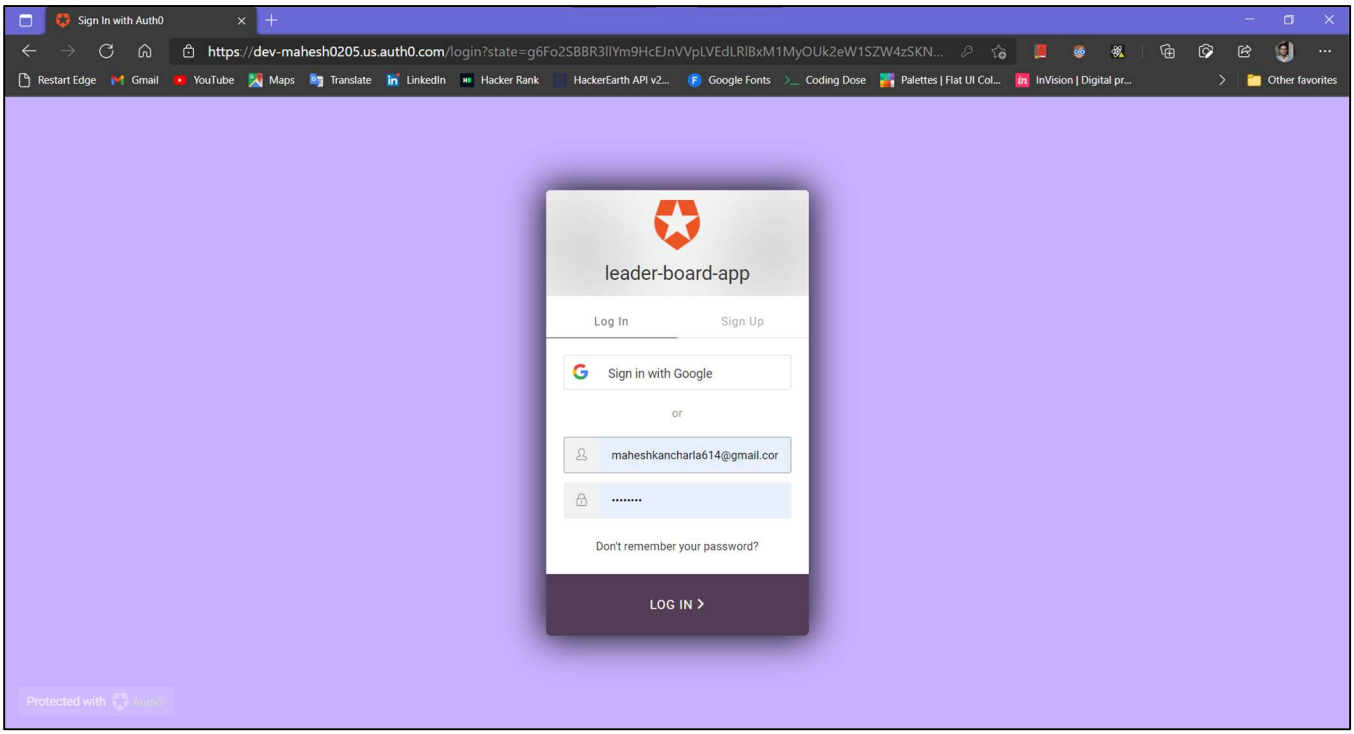
- Home Page (common)



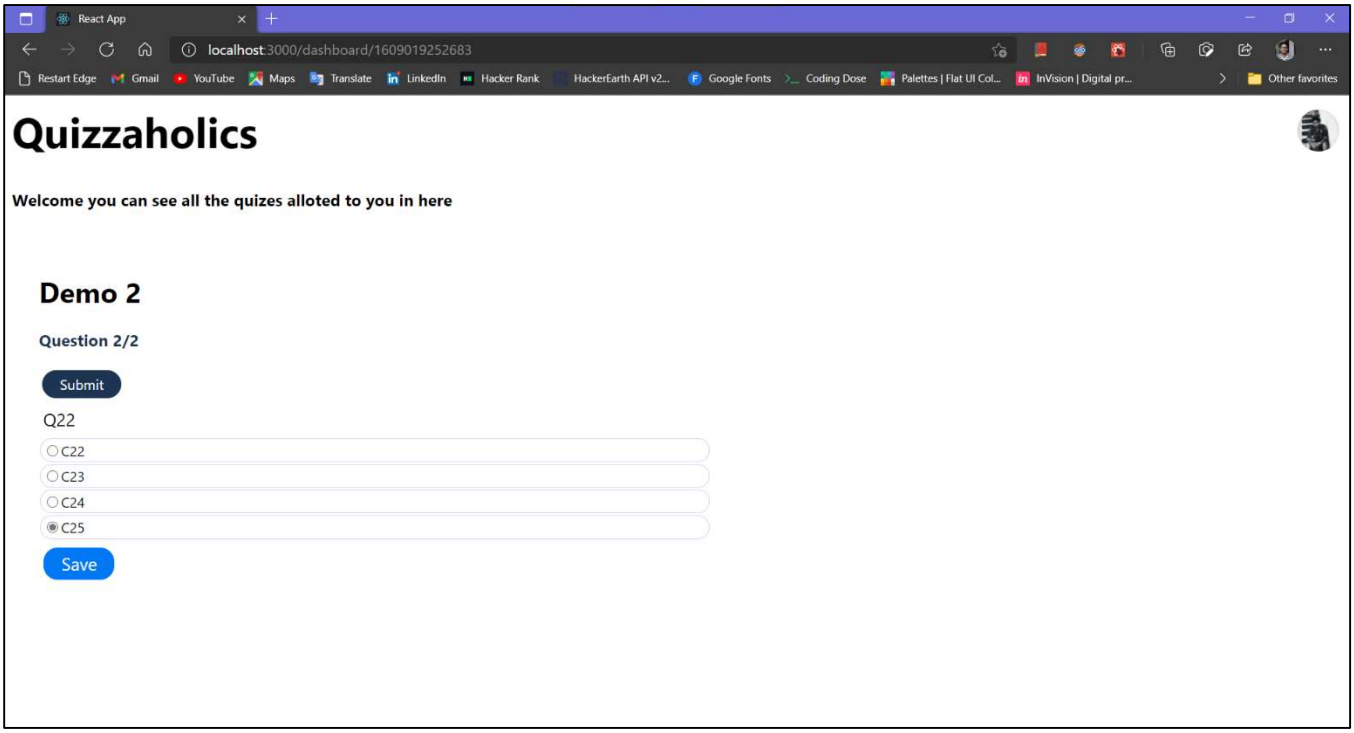
- Rankings



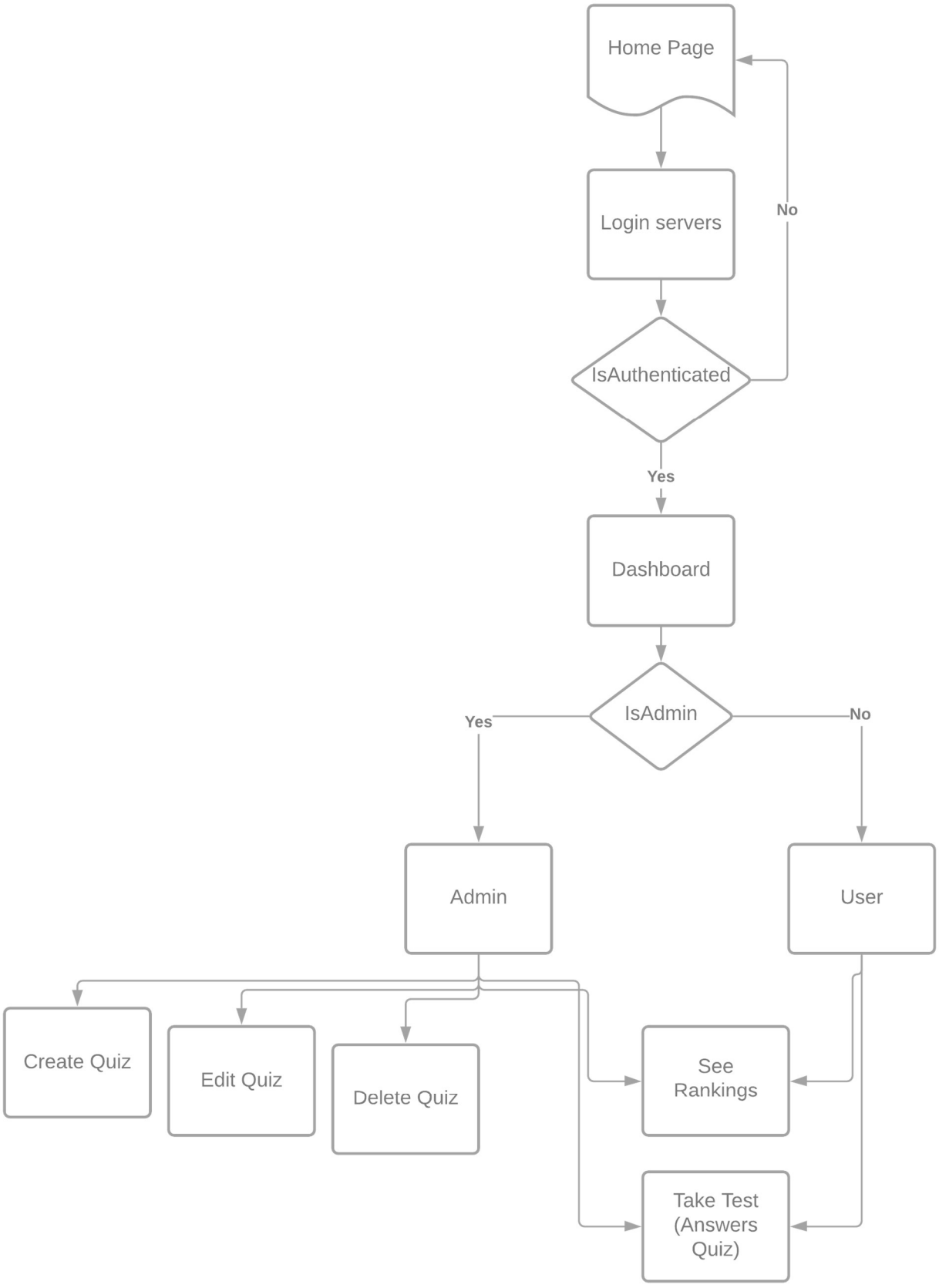
- Login Page:



- Attending Quiz:



Data Flow in the Application (at a glance):



DOCUMENTATION

This section contains about the documentation of the website both for frontend in react.js and backend in python with flask framework.

Frontend doc:

The frontend project folder follows the same folder structure that the 'create-react-app' uses for creating any other react applications. The main entry to the project and all the development was done in the 'src' folder where all the JavaScript files live in and we will consider this as our working folder from now on, for the sake of making this doc clean and simple.

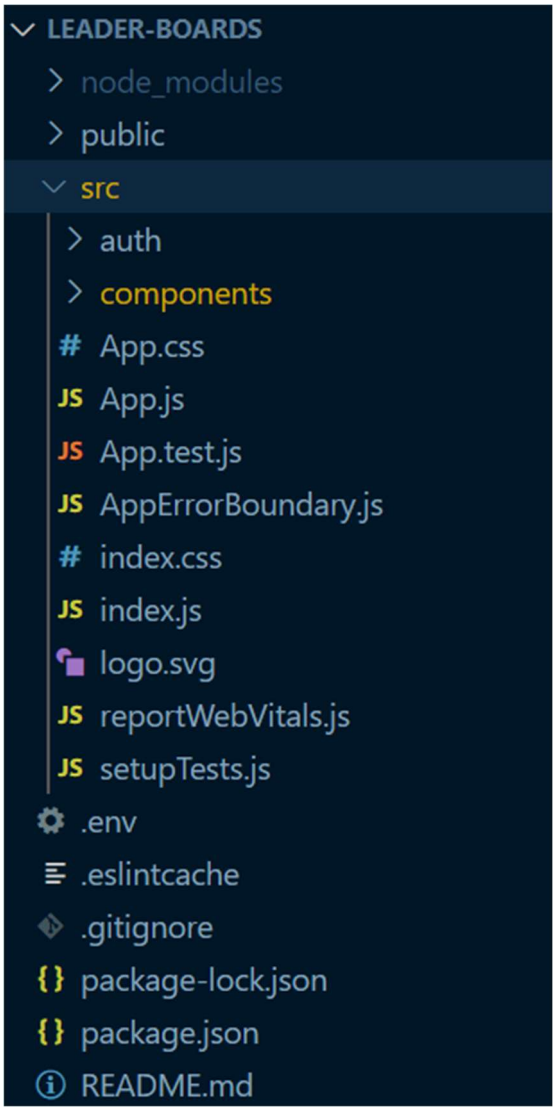


Figure 1 Frontend Folder Structure

Dependencies:

- react-router-dom:
  This dependency was used to maintain the internal routing and to render the components dynamically on the client.
- @auth0/auth0-react:
  This an SDK from the auth0 used for authentication and authorization of the users to the website and secure both the frontend and backend.

Components:

Index.js:

This file just links the App component (in './App.js') to the index.html file in the public folder which is the one that displayed when the website is served. The 'App' component is enclosed with in the BrowserRouter component imported from 'react-router-dom' for enabling the routing feature in the app which is again

```
src > JS index.js
1   import React from "react";
2   import ReactDOM from "react-dom";
3   import "./index.css";
4   import App from "./App";
5   import reportWebVitals from "./reportWebVitals";
6   import
7   import
8   import AppErrorBoundary from "./AppErrorBoundary";
9
10  ReactDOM.render(
11      <AppErrorBoundary>
12          <BrowserRouter>
13              <Auth0ProviderWithHistory>
14                  <App />
15              </Auth0ProviderWithHistory>
16          </BrowserRouter>
17      </AppErrorBoundary>,
18      document.getElementById("root")
19  );
```

(alias) class AppErrorBoundary
import AppErrorBoundary

enclosed with the customized Auth0Provider component from the auth0 sdk to enable the authentication feature.

Figure 2 Index.js

App.js:

The App component in the file is kind of a root component to all the others. Basically, the App component renders a Header component which is a header to the website and based on the user's authorization they will be shown a Dashboard component (only for logged in users) or a Publicpage component which is common for all the unauthorized users.

The authorization part will take place in this component when the user decides to login, they will be redirected a login page and if the user is authenticated then the user will be redirected again to the page with his/her dashboard in it. Route component from react-router-dom package is used to make this a smooth flow and show the components based on the context that returned by the 'useAuth0' custom hook from auth0 SDK.

src > App.js > PublicPage

```jsx
import "./App.css";
import React from "react";

import Header from "./components/Header";
import DashBoard from "./components/DashBoard";

import { useAuth0 } from "@auth0/auth0-react";
import { Route, Switch } from "react-router-dom";
import Loading from "./components/Loading";

function PublicPage(props) {
    return (
        <div className>
            <h2>Hey There! Welcome to Quizzaholics</h2>

            <p>
                Login or create an account to compete with your colleuges and friends in
                quizes that your mentor assigned for{" "}
            </p>
            <p>
                Want to create quizes...sure thing! Login with an administrator account
                to create or edit your quizes
            </p>
        </div>
    );
}

export const AppContext = React.createContext();

function App() {
    const authContext = useAuth0();
    if (authContext.isLoading) {
        return <Loading />;
    }
    return (
        <AppContext.Provider value={{ authContext }}>
            <div className='app-container'>
                <Header authContext={authContext} />
                <Switch>
                    <Route path='/dashboard'>
                        <DashBoard />
                    </Route>
                    <Route path='/'>
                        <PublicPage />
                    </Route>
                </Switch>
                {/* {authContext. */}
            </div>
        </AppContext.Provider>
        // <Loading />
    );
}

export default App;
```

Header.js

The Header component renders the header for the website where it contains some navigation and the title of the website. If the user is not authenticated the header will show a login button (which is also default) and shows the user profile pic which can be used to logout of the session if the users are authenticated.

```
src > components > Header.js > Header
import React, { useContext } from "react";
import { AppContext } from "../App";

function Header(props) {
    const { authContext } = useContext(AppContext);
    const { loginWithRedirect, logout, isAuthenticated, user } = authContext;
    let LoginBtn = () => {
        return (
            <button
                className='login-btn'
                onClick={() => {
                    loginWithRedirect();
                }}
            >
                Login / Sign up
            </button>
        );
    };
    let LogoutBtn = () => {
        return (
            <button
                className='logout-btn'
                onClick={() => {
                    logout({ redirectTo: "window.location.origin" });
                }}
            >
                Logout
            </button>
        );
    };
    const toggleProfilePopup = (e) => {
        let profilePopup = document.getElementById("profile-popup");
        profilePopup.classList.toggle("show-profile-popup");
    };

    return (
        <header className='app-header'>
            <nav className='navbar'>
                <section className='webpage-title'>Quizzaholics</section>
                {isAuthenticated ? (
                    <>
                        <img
                            src={user.picture}
                            alt='Profile pic'
                            className='profile-link'
                            onClick={(e) => {
                                toggleProfilePopup(e);
                            }}
                        />
                        <div className='profile-popup' id='profile-popup'>
                            <section className='profile-content'>
                                <p>Name: {user.name}</p>
                                <p>Email: {user.email}</p>
                            </section>
                            <LogoutBtn />
                        </div>
                    </>
                ) : (
                    <LoginBtn />
                )}
            </nav>
        </header>
    );
}

export default Header;
```

Figure 4 Header.js

Dashboard.js

As It was said in the App component, this component is rendered once the auth0 authenticates the user. It also checks further for the permissions the logged in user has for the website and based on that permissions it will return either an Admin component or a User component. Both are similar, mostly, but the Admin component has an elevated right for creating quizzes or editing quizzes or even deleting quizzes, which are

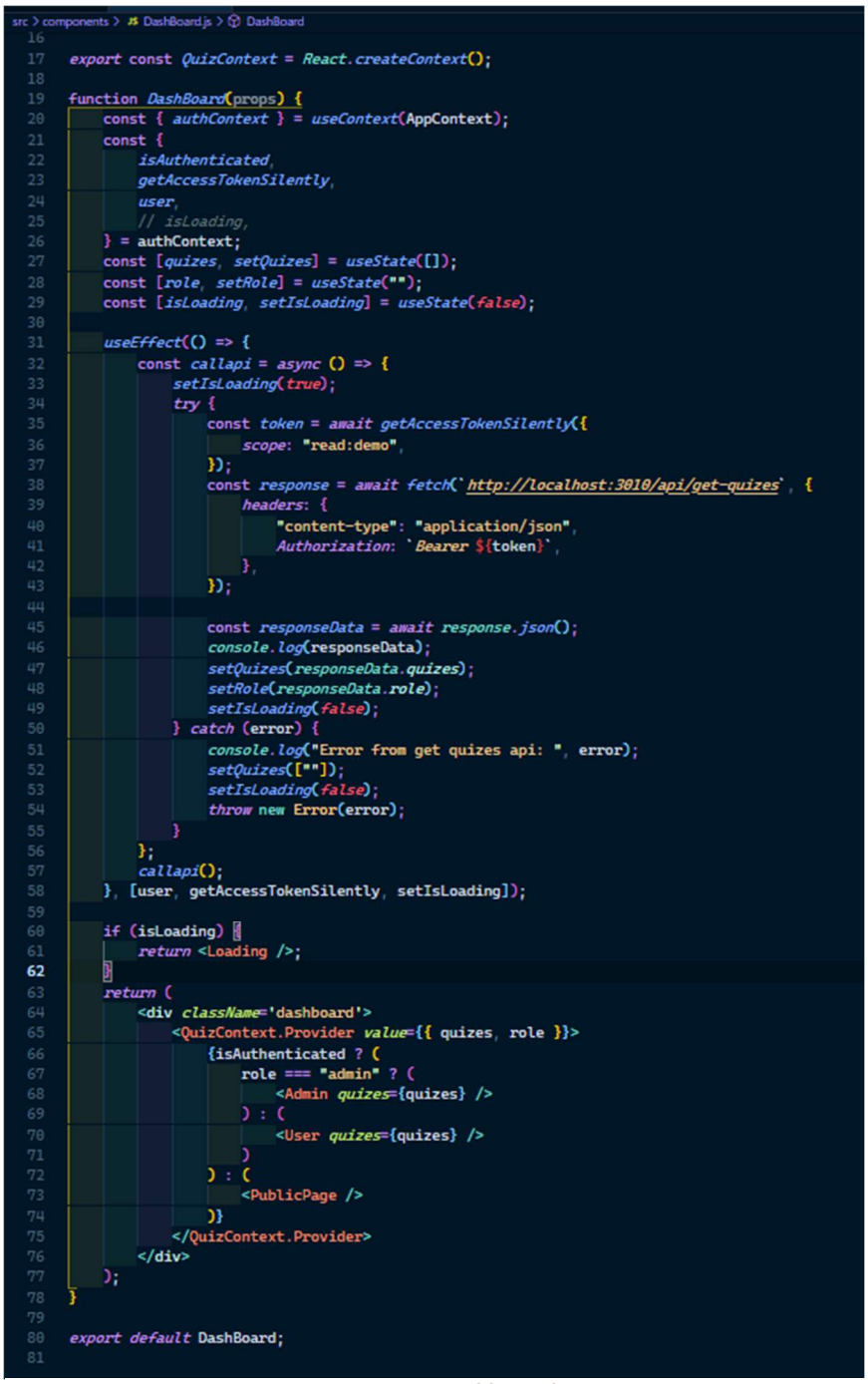


Figure 5 Dashboard.js

created by the user. Whereas the User component can see all the quizzes assigned to him and respond with their answers to a paticular quiz. Both the components show Rankings for a paticular quiz (only top 10 are displayed).

Admin.js

This component renders all the quizzes that the user created and displays them. It also gives the option to create a new one. The user can edit or delete a quiz or can see the leader board for a quiz. The quizzes are served to this component through a asynchronous fetch call to a backend flask API namely the endpoint '/api/get-quizes/' through a GET request. Similarly, the quizzes can be created or edited and sent a POST request containing the questions to the backend API. The user can also take the quiz from this component in the same way the normal user does.

User.js

This component is similar to the Admin component but it does not have features for creating editing and deleting the quizzes. The user will see all the quizzes he was allotted to in this component and can be able to respond to a paticular quiz.

NOTE:

- The CreateQuiz and AddQuestion components are used for both creating a quiz and editing a quiz.
- The Quizes component ('./src/components/QuizesC.js') is used to display all the quizes in both Admin and User components.
- The Quiz (in './src/components/Quiz.js') and Question (in './src/components/Question.js') is used as the interface for responding to a paticular quiz.
- The Quiz component only loads one question at a time for answering and cannot be able to go back for the previous Question.

Backend doc:

The backend is implemented in python with Flask framework to be more precise. The backend works like an API where the endpoints are protected with auth0 by using JWT to authenticate the incoming request and to make sure it is safe and secured.

Dependencies:

- flask
- python-dotenv
- python-jose
- flask-cors
- six

API Endpoints:

- /api/get-quizes/
  Request Mode: GET
  Header: content-type: application/json, Authorization: Bearer {token}
  Response: return the json object with quizzes if the request is valid and authenticated.
- /api/create-quiz/
  Request Mode: POST
  Header: content-type: application/json, Authorization: Bearer {token}.
  Body: The request body should contain the all the questions, quizId, UserId
  Response: return the json object with success message if the request is valid, authenticated and processed without any errors.
- /api/add-quiz-response/
  Request Mode: POST
  Header: content-type: application/json, Authorization: Bearer {token}.
  Body: The request body should contain the quizId, UserId, the responses to the quiz.
  Response: return the json object with success message if the request is valid, authenticated and processed without any errors.
- /api/get-top10?quizid={quizid}
  Request Mode: GET
  Header: content-type: application/json, Authorization: Bearer {token}
  Response: return the json object of top10 responses for the quizid specified in query parameters only if the request is valid, authenticated and processed without any errors.
- /api/delete-quiz/
  Request Mode: POST
  Header: content-type: application/json, Authorization: Bearer {token}.
  Body: the request body should contain the userId and quizId to be deleted.
  Response: return the json object with success message if the request is valid, authenticated and processed without any errors.