Reflection

Setting up Auth0

1. **Sign up Auth0**
2. **Create an application which type is "Single Page Web Applications" and choose React**
3. **Click setting in the dashboard**
4. **Config Allow Callback URLs with http://localhost:3000**
5. **Allowed Logout URLs: http://localhost:3000**
6. **Allowed Web Origins: http://localhost:3000**
7. **Create API in Auth0 named Todolist**
8. **Fill the identifier section with https://quickstarts/api**

Setting Environment

There is a .env file in the fronded needed to be changed

1. **REACT_APP_AUTH0_DOMAIN= (Copy the domain name from the Auth0 and paste it there)**
2. **REACT_APP_AUTH0_CLIENT_ID=(Copy the Client ID from Auth0 and paste it there)**
3. **REACT_APP_AUTH0_AUDIENCE=https://quickstarts/api (If identifier has the same URL don't need to change otherwise paste the identifier there)**

There is file named check-jwj.js need to be modified. It should be look like this

1. **jwksUri: ` https://YOUR DOMAIN NAME/.well-know/jwks.jason `**
2. **audience: ' https://quickstarts/api '**
3. **issuer: [`https:// YOUR DOMAIN NAME `]**
4. **algorithm: ['RS256']**

Fronded implementation for Task one

To complete the contents of task1， First of all import Auth0Provider from auth0 and config Domain, Client ID, redirect URL and audience for it. Wrapping app by <Auth0Provider>. After setting up Auth0 React SDK, create login and logout components. Auth0 provide us with very simple login and logout hook, through the above methods we can realize the user's login and logout. To meet the requirement which an unauthenticated user should instead be redirected to the Auth0 login page, I created a loading component. Then use { withAuthenticationRequire } from auth0 to warp the export default App. When the user is not authenticated, the content of app.js cannot be displayed and automatically jumps to the loading.js component to complete the login. To realize that the logged-in user can see the logout button, I created a new authenticationButton.js component to determine whether the user is logged in. If the user completes the login, only the logout button component will be returned.

Backend Implementation for Task two

To implement task two, firstly, I need to create an API. Fill in the identifier section with http://quickstarts/api and use the identifier as an audience. After

that, I need to configure the middleware to realize access control. Create a checkJwt.js and configure the express-jwt middleware so it can be used on our router. In order to protect routers, in the todos-routers.js file, I configure each router with the checkJwt express-jwt middleware. After implemented those steps, our routers are protected. When making an API request, only the authenticated users can achieve those operations ("GET all to do items," "GET a single to-do item by id," "Create a new to-do item," "Update a to-do item," "DELETE a to-do item"). If it doesn't authenticate, a 401 error will be returned. To realize which users created which to-do items, I modify the database schema by adding a new element named username. After that, I modify the retrieveAlltodos function by adding a new parameter: req.user.sub. The auth0 provided us a user object that has a sub property when a user is authenticated. By the value of sub, determine if the username stored in the database is equal to the value of req.user.sub. Through that method, the frontend displays only the corresponding user's to-do item.

Feedback

Auth0 provides SDK for node.js and a lot of documentation which allows me quickly to achieve the login and registration on the fronded. Route protection can also be effectively implemented through the methods provided by auth0. However, when configuring the route protection of the express API in the backend, I think the official website documents are not very complete and even a little messy, which caused me to face challenges in completing task2.

The documentation only mentions how to use Middleware to authenticate API permissions, but it doesn't mention how to handle the issues of 'token' and 'header' in the stage where the Fetch API is required in the frontend. This part, I think, was more difficult than anticipated. On the other hand, what is difficult for me is how to implement unit tests under the condition of using auth0 authentication. The official document should introduce a test method in detail.

The documentation is generally adequate and can guide me to complete the task. But how the API defined in Express implements authentication on the front end is not mentioned in the documentation. Other parts of the document are omitted, such as reference relationships between JS files, local environment configuration. The content of the documentation is not suitable for those who are new to development. In my opinion, if auth0 can make the document more detailed based on the above suggestions, it will give more help to newcomers who use auth0.