Project Setup

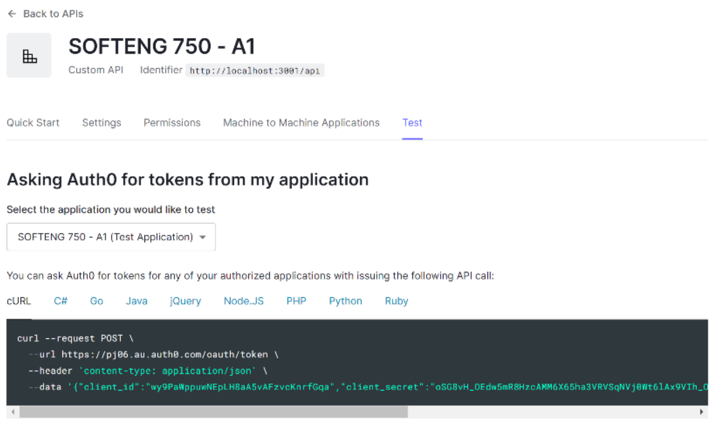
Setup Auth0 Application and API

When setting up the Auth0 Application, make sure that "Application Type" is a "Single Page Application" and add the URL, http://localhost:3000/, to the "Allow Callback URLs", "Allowed Logout URLs", and "Allowed Web Origins" subsections in the "Application URLs" section. When setting up the API, make sure the "identifier" is the following: http://localhost:3001/api.

Required File Changes

You need to modify the ".env" files located in the "frontend" and "backend" folders. Once you have created your Auth0 Application, copy and paste the Domain and ClientId into the ".env" file in the "frontend" folder to their respective variables. Also, copy and paste the identifier for you API (http://localhost:3001/api) in the REACT_APP_AUTH0_AUDIENCE variable in the "frontend" folder's ".env" file.

For the ".env" file in the "backend" folder, paste the API identifier in the AUTH0_ISSUER variable in the file. Go to this link and under the "Configure the Middleware" section copy and paste your values (i.e., jwksUri, issuer) in their respective fields in the ".env" file. Finally, under the *Test* subsection in your API (figure 1), copy and paste the client_id, client_secret, grant_type, and url into the respective fields in the ".env" file in the "backend" folder.



Asking Auth0 for tokens from my application

Select the application you would like to test

SOFTENG 750 - A1 (Test Application)

You can ask Auth0 for tokens for any of your authorized applications with issuing the following API call:

```
curl --request POST \
  --url https://pj06.au.auth0.com/oauth/token \
  --header 'content-type: application/json' \
  --data '{"client_id":"wy9PaWppuwNEpLH8aA5vAFzvcKnrfGqa","client_secret":"oSG8vH_OEdw5mR8HzcAMM6X65ha3VRVSqNVj0Wt6lAx9VIh_O
```

Figure 1 Location to Access values to Run Tests

Reflection

Changes Made to Frontend

In the file, "index.js", I added the component "Auth0Provider" with the following props: domain, clientId, redirectUri, and audience. The "Auth0Provider" component was wrapped around the application so the application could work the Auth0.

A file I added was the "LogoutButton.js" file in the "components" folder as Task One required for us to create a logout button. For styling purposes, I used Material UI's Button component and some custom CSS to position the button on the top right of the screen. To logout the user when the logout button is clicked, I used Auth0's logout function provided by the "useAuth0" hook; therefore, when the button is clicked the logout function provided by Auth0 is called.

The file, "App.js" was also modified to accommodate Task One's requirements. As users should not be able to view their todos until they are logged in, by using the "isAuthenticated" and "isLoading" variables provided by the "useAuth0" hook meant that if any of the two variables were false, then the todos would not be shown. I also used the “loginWithRedirect” function provided by the “useAuth0” hook; therefore, if the user is not authenticated and Auth0 is not logging them in (i.e., isLoading and isAuthenticated variables are *false*) then the “loginWithRedirect” function is called which redirects the user to the Auth0 login page. Furthermore, the “useEffect” hook was used so if the “isLoading” or “isAuthenticated” variables change then the “App” component is re-rendered potentially showing the user their todos (depending on whether the user is logged in).

Finally, the “useCrud.js” file was modified. As users need to be authenticated to access the API endpoints, a token needs to be supplied in each request to the API. Therefore, the user’s session token was added to each request to the API in “Authorization” field under the request “headers”. To retrieve the token, the “getAccessTokenSilently” function provided by the “useAuth0” hook was used. We save the token in a variable called “token”, and we set the token variable by calling the “setToken” function. To ensure that the token does not become stale, we call the “getAccessTokenSilently” function when we changed the URL, or re-fetch – by calling the function in the “useEffect” hook on line 29.

Changes Made to Backend

Firstly, “todos-schema.js” was modified because todos need to be associated to users. Therefore, the “userID” property was added to the “todoSchema”. The “userID” property is of type String, and it is associated to a user’s unique identifier; so, we can easily distinguish which todos belong to which user.

As we wanted to ensure that users can only access and modify their todos, the “todos-dao.js” file was changed accordingly. In terms of retrieving all a user’s todos, we find all the user’s todos by finding todos that correspond to the user’s unique identifier. We ensured that a user can only modify their existing todos. Therefore, if the user is attempting to modify someone else’s todos, a 401 exception is now thrown. We do this by checking that the “userID” property associated to the updating todo is equivalent to the user’s unique identifier. In terms of deleting a todo, if the todo being deleted is not associated to the current user (i.e., the “userID” property and the user’s unique identifier do not match) then the todo is not deleted.

The “index.js” file in the “api” folder was modified. As we want the API endpoints to be secure, a middleware was added to check for this (as shown on line 10). If the user is unauthenticated then a 401 error is thrown.

The “todos-routes.js” file was modified. Firstly, the express-jwt middleware was added and modified to account for my Auth0 Application and API. The use of express-jwt allows us to get my Auth0 public key and complete the verification process. The express-jwt middleware was added to each route to protect the routes from unauthorised users. Finally, all the routes accessed the user’s “userID” by accessing the “user” object in the request. The “userID” variable was used to check that the user was accessing and modifying their todos.

Auth0 Analysis

Adding Auth0 authentication to my web-application was straight forward. As I had never added authentication to a web-application, I initially believed that adding Auth0 authentication would be difficult. However, after completing this assignment, I can safely say that adding authentication to the web-application was significantly easier than what I expected. The reason for this was because of Auth0’s documentation. Their documentation was easily understandable, and easy to follow. Furthermore, when I was logged in, the documentation would be modified based on my API keys, which meant adding Auth0 authentication was as easy as copying and pasting. However, while the documentation was easy to follow, finding the correct documentation to solve the current problem on hand was difficult. Therefore, an improvement I would suggest would be to make their documentation easier to find, as it would become relatively easier to find the correct documentation.