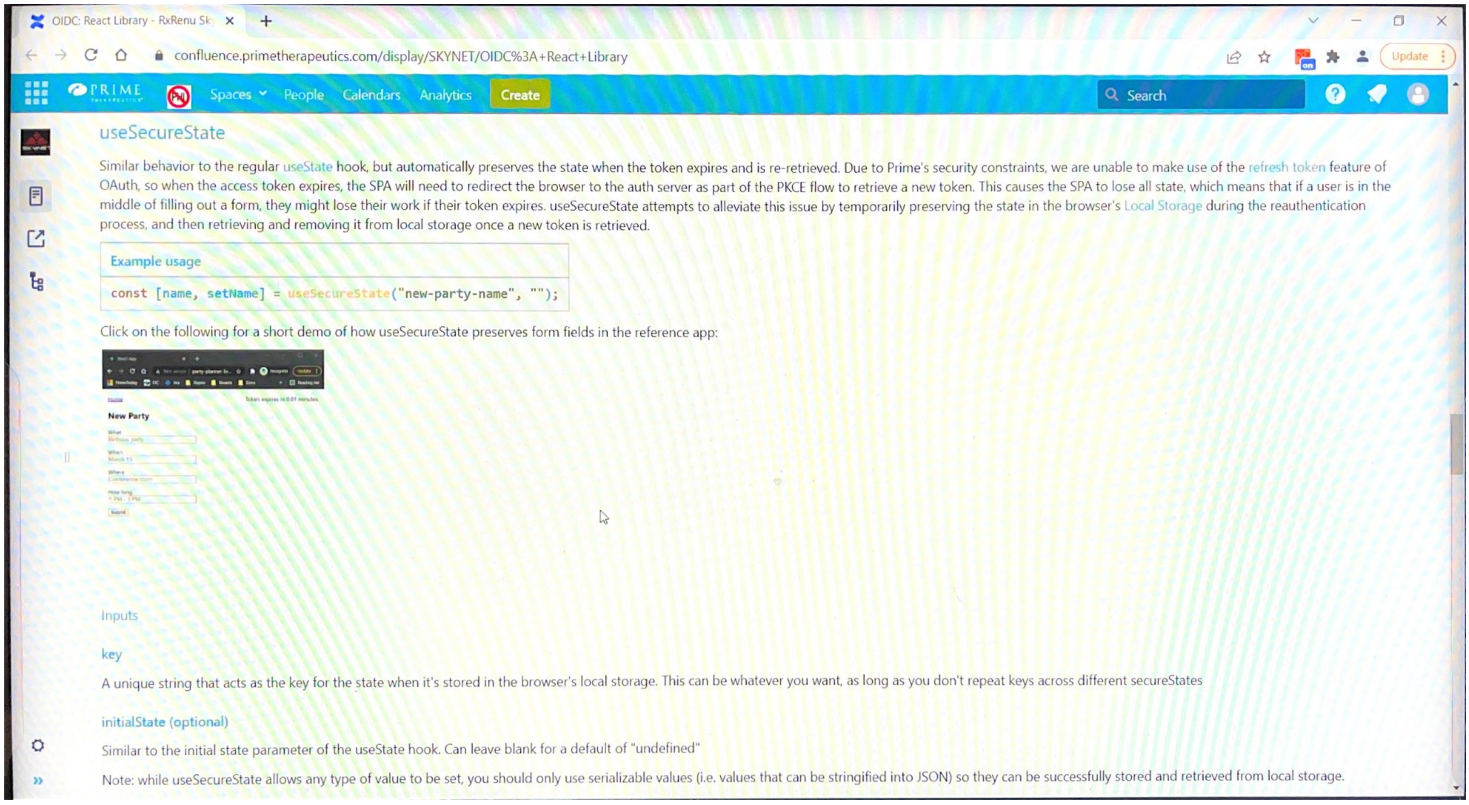
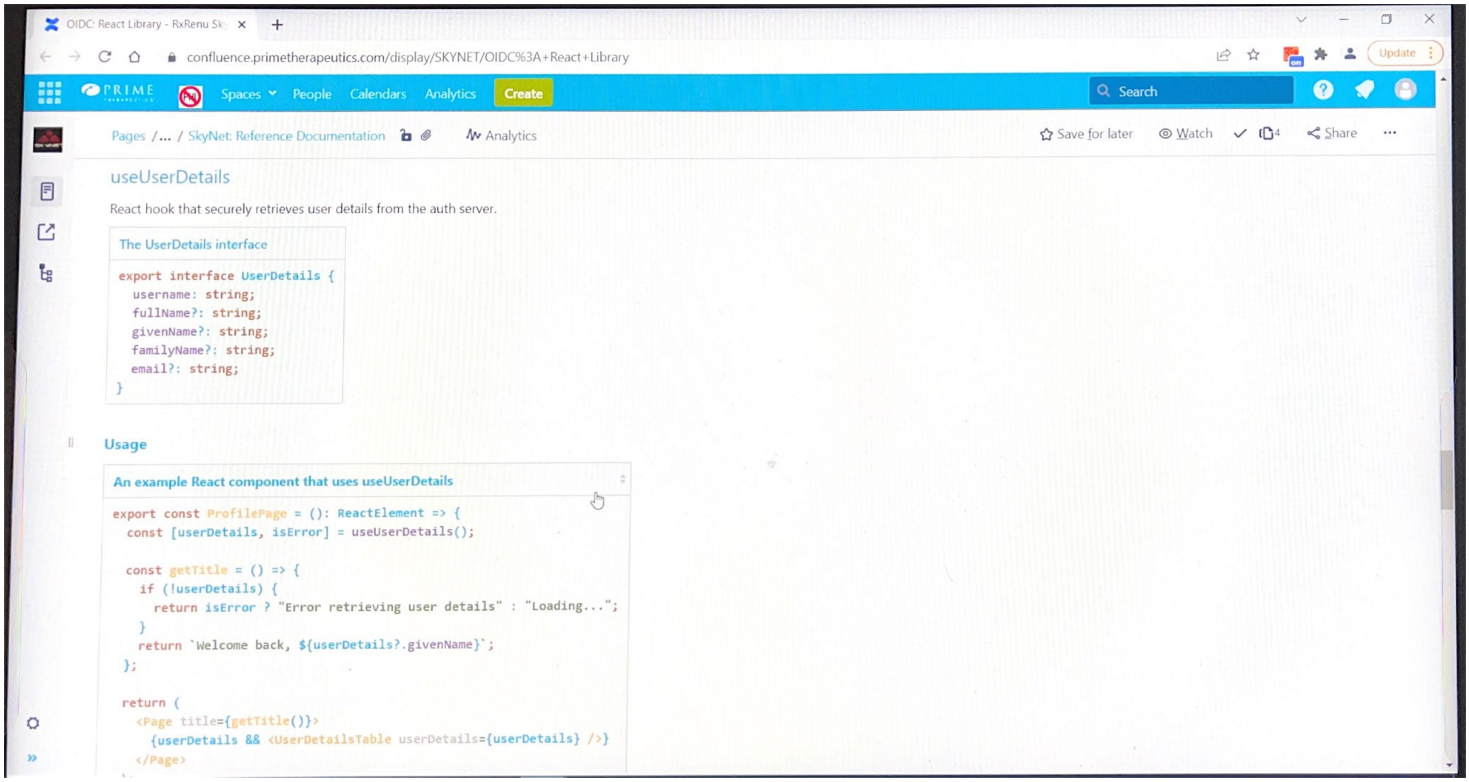
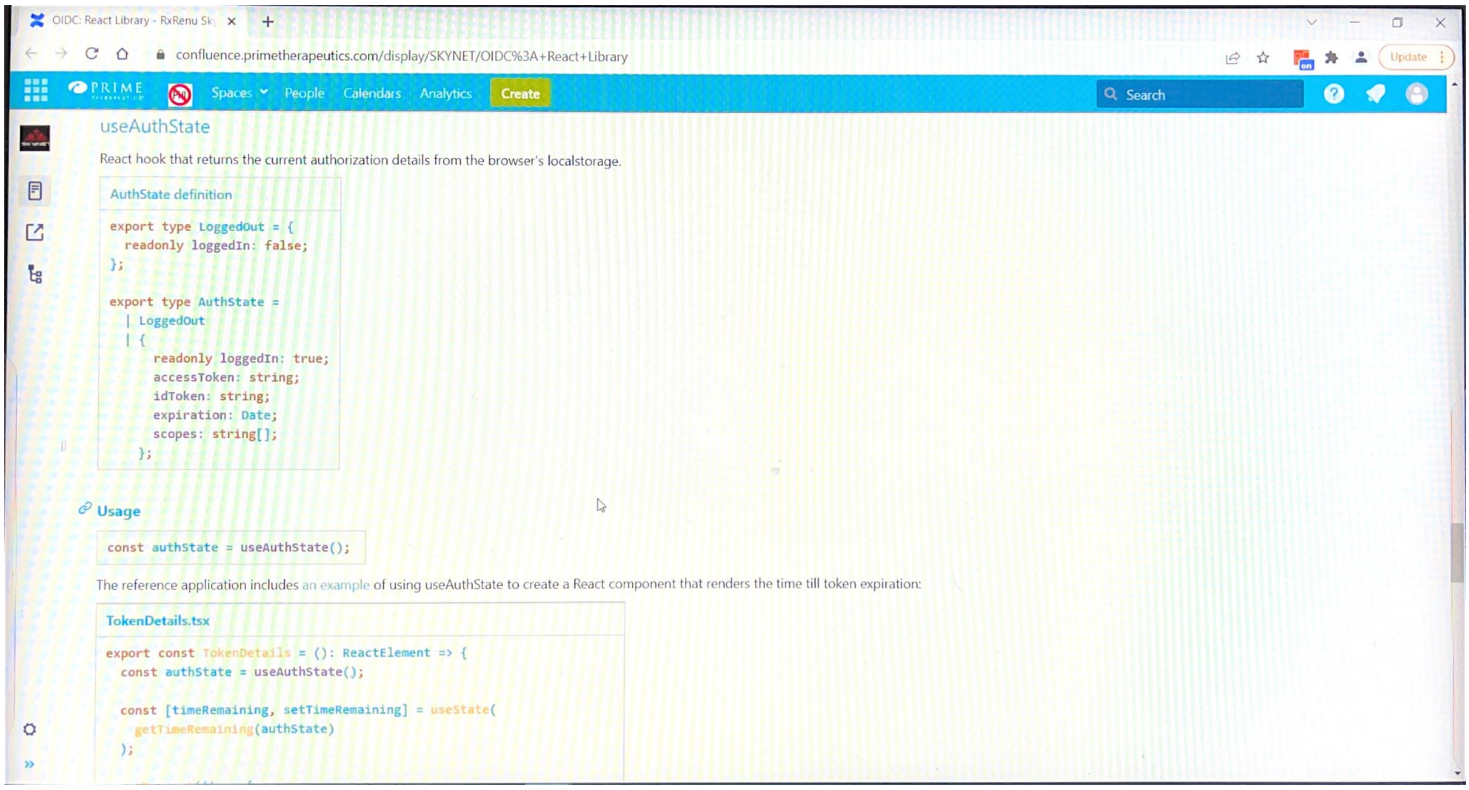
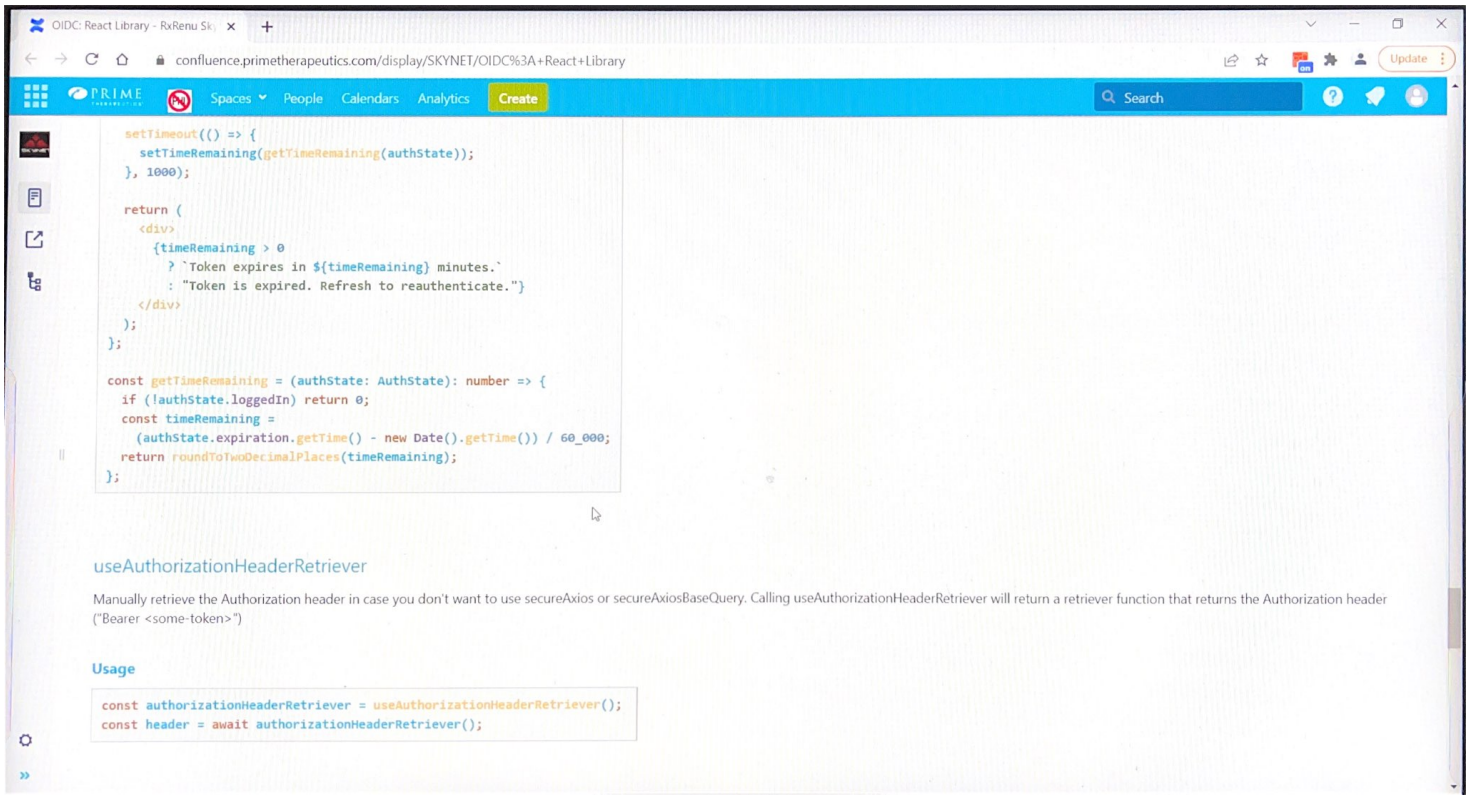

OIDC: React Library - RxRenu Sk
confluence.primetherapeutics.com/display/SKYNET/OIDC%3A+React+Library
Update
PRIME
Spaces
People
Calendars
Analytics
Create
Search
useSecureState
Similar behavior to the regular useState hook, but automatically preserves the state when the token expires and is re-retrieved. Due to Prime's security constraints, we are unable to make use of the refresh token feature of OAuth, so when the access token expires, the SPA will need to redirect the browser to the auth server as part of the PKCE flow to retrieve a new token. This causes the SPA to lose all state, which means that if a user is in the middle of filling out a form, they might lose their work if their token expires. useSecureState attempts to alleviate this issue by temporarily preserving the state in the browser's Local Storage during the reauthentication process, and then retrieving and removing it from local storage once a new token is retrieved.
Example usage
const [name, setName] = useSecureState("new-party-name", "");
Click on the following for a short demo of how useSecureState preserves form fields in the reference app:
New Party
Inputs
key
A unique string that acts as the key for the state when it's stored in the browser's local storage. This can be whatever you want, as long as you don't repeat keys across different secureStates
initialState (optional)
Similar to the initial state parameter of the useState hook. Can leave blank for a default of "undefined"
Note: while useSecureState allows any type of value to be set, you should only use serializable values (i.e. values that can be stringified into JSON) so they can be successfully stored and retrieved from local storage.

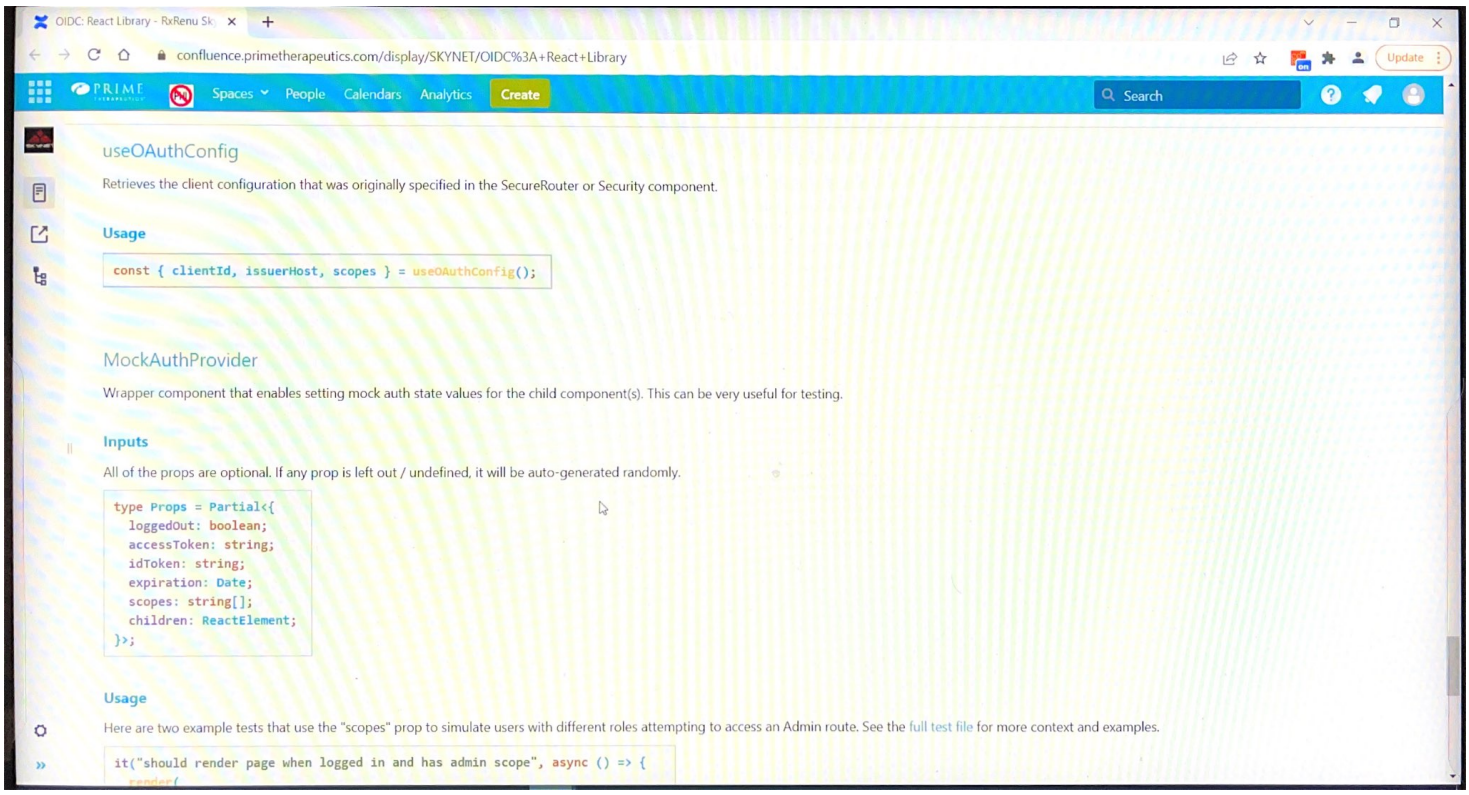

OIDC: React Library - RxRenu Sk
confluence.primetherapeutics.com/display/SKYNET/OIDC%3A+React+Library
Update
PRIME
Spaces
People
Calendars
Analytics
Create
Search
Pages / ... / SkyNet: Reference Documentation
Analytics
Save for later
Watch
Share
useUserDetails
React hook that securely retrieves user details from the auth server.
The UserDetails interface
export interface UserDetails {
username: string;
fullName?: string;
givenName?: string;
familyName?: string;
email?: string;
}
Usage
An example React component that uses useUserDetails
export const ProfilePage = (): ReactElement => {
const [userDetails, isError] = useUserDetails();
const getTitle = () => {
if (!userDetails) {
return isError ? "Error retrieving user details" : "Loading...";
}
return `Welcome back, ${userDetails?.givenName}`;
};
return (
<Page title={getTitle()}>
{userDetails && <UserDetailsTable userDetails={userDetails} />}
</Page>

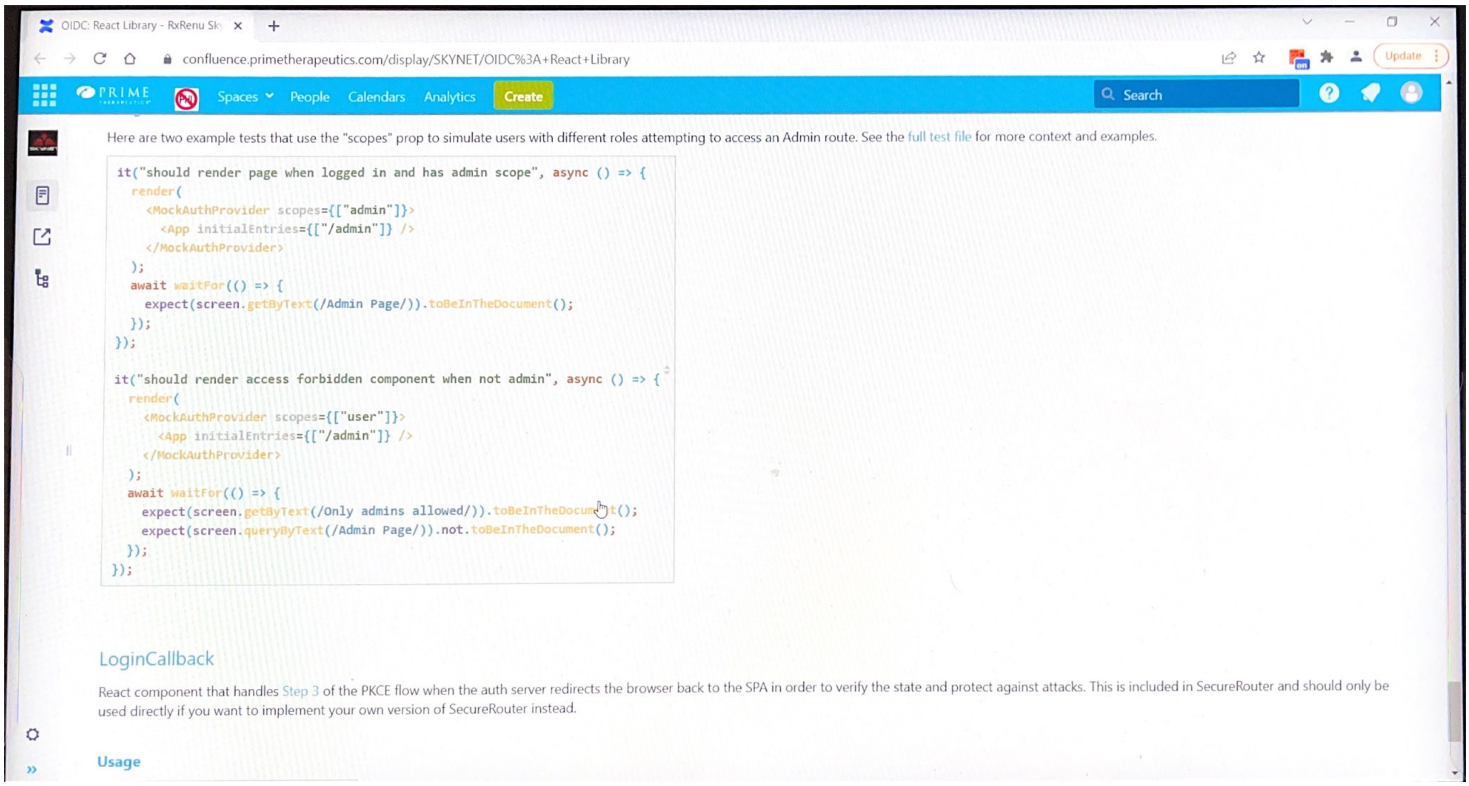

OIDC: React Library - RxRenu Sk
confluence.primetherapeutics.com/display/SKYNET/OIDC%3A+React+Library
Update
PRIME
Spaces
People
Calendars
Analytics
Create
Search
useAuthState
React hook that returns the current authorization details from the browser's localstorage.
AuthState definition
export type LoggedOut = {
readonly loggedIn: false;
};
export type AuthState =
| LoggedOut
| {
readonly loggedIn: true;
accessToken: string;
idToken: string;
expiration: Date;
scopes: string[];
};
Usage
const authState = useAuthState();
The reference application includes an example of using useAuthState to create a React component that renders the time till token expiration:
TokenDetails.tsx
export const TokenDetails = (): ReactElement => {
const authState = useAuthState();
const [timeRemaining, setTimeRemaining] = useState(
getTimeRemaining(authState)
);

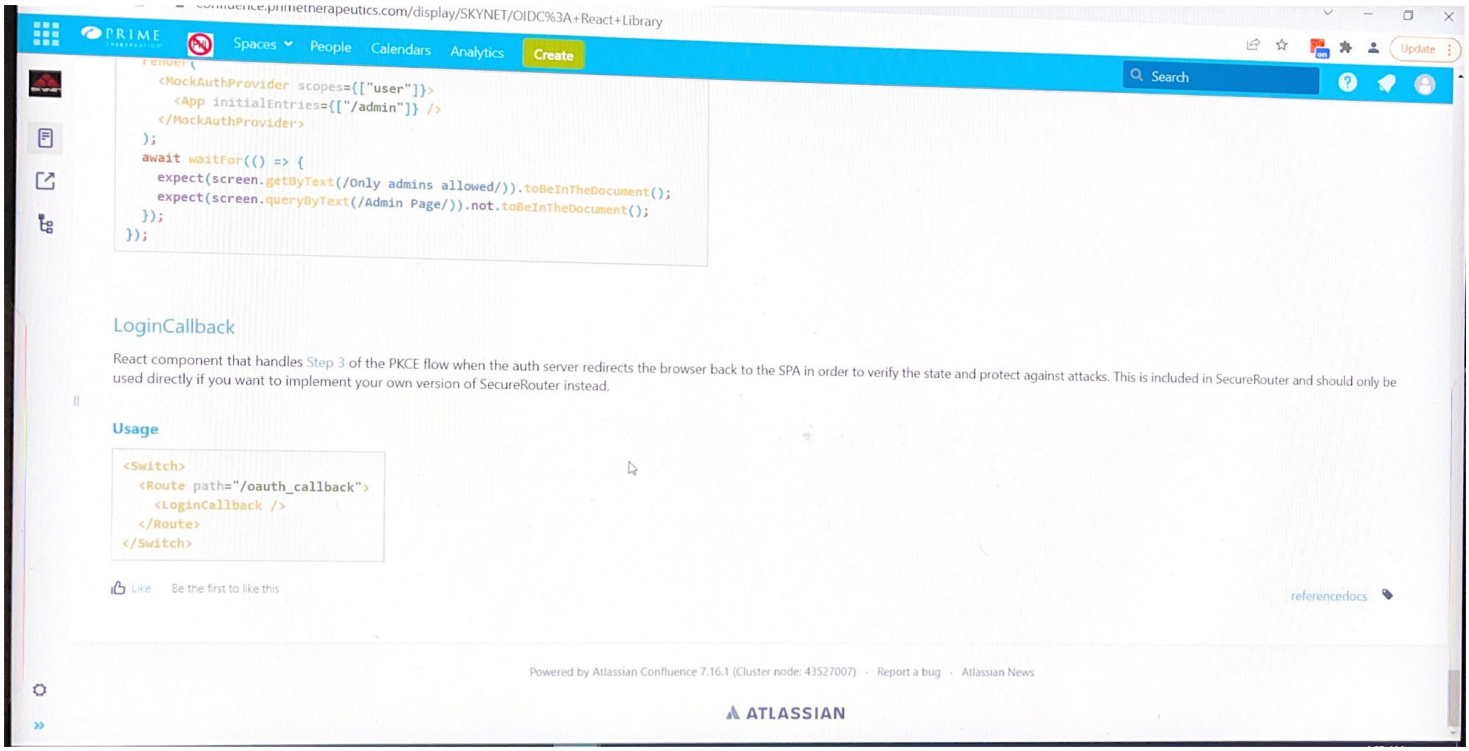

OIDC: React Library - RxRenu Sk
confluence.primetherapeutics.com/display/SKYNET/OIDC%3A+React+Library
Update
PRIME
Spaces
People
Calendars
Analytics
Create
Search
setTimeout(() => {
setTimeRemaining(getTimeRemaining(authState));
}, 1000);
return (

{timeRemaining > 0
? `Token expires in ${timeRemaining} minutes.`
: "Token is expired. Refresh to reauthenticate."}

);
};
const getTimeRemaining = (authState: AuthState): number => {
if (!authState.loggedIn) return 0;
const timeRemaining =
(authState.expiration.getTime() - new Date().getTime()) / 60_000;
return roundToTwoDecimalPlaces(timeRemaining);
};
useAuthorizationHeaderRetriever
Manually retrieve the Authorization header in case you don't want to use secureAxios or secureAxiosBaseQuery. Calling useAuthorizationHeaderRetriever will return a retriever function that returns the Authorization header ("Bearer <some-token>")
Usage
const authorizationHeaderRetriever = useAuthorizationHeaderRetriever();
const header = await authorizationHeaderRetriever();


OIDC: React Library - RxRenu Sk
confluence.primetherapeutics.com/display/SKYNET/OIDC%3A+React+Library
Update
PRIME
Spaces
People
Calendars
Analytics
Create
Search
useOAuthConfig
Retrieves the client configuration that was originally specified in the SecureRouter or Security component.
Usage
const { clientId, issuerHost, scopes } = useOAuthConfig();
MockAuthProvider
Wrapper component that enables setting mock auth state values for the child component(s). This can be very useful for testing.
Inputs
All of the props are optional. If any prop is left out / undefined, it will be auto-generated randomly.
type Props = Partial<{
loggedOut: boolean;
accessToken: string;
idToken: string;
expiration: Date;
scopes: string[];
children: ReactElement;
}>;
Usage
Here are two example tests that use the "scopes" prop to simulate users with different roles attempting to access an Admin route. See the full test file for more context and examples.
it("should render page when logged in and has admin scope", async () => {


OIDC: React Library - RxRenu Sk
confluence.primetherapeutics.com/display/SKYNET/OIDC%3A+React+Library
Update
PRIME
Spaces
People
Calendars
Analytics
Create
Search
Here are two example tests that use the "scopes" prop to simulate users with different roles attempting to access an Admin route. See the full test file for more context and examples.
it("should render page when logged in and has admin scope", async () => {
render(
<MockAuthProvider scopes={["admin"]}>
<App initialEntries={["/admin"]} />
</MockAuthProvider>
);
await waitFor(() => {
expect(screen.getByText(/Admin Page/)).toBeInTheDocument();
});
});
it("should render access forbidden component when not admin", async () => {
render(
<MockAuthProvider scopes={["user"]}>
<App initialEntries={["/admin"]} />
</MockAuthProvider>
);
await waitFor(() => {
expect(screen.getByText(/Only admins allowed/)).toBeInTheDocument();
expect(screen.queryByText(/Admin Page/)).not.toBeInTheDocument();
});
});
LoginCallback
React component that handles Step 3 of the PKCE flow when the auth server redirects the browser back to the SPA in order to verify the state and protect against attacks. This is included in SecureRouter and should only be used directly if you want to implement your own version of SecureRouter instead.
Usage


confluence.primetherapeutics.com/display/SKYNET/OIDC%3A+React+Library
PRIME
Spaces
People
Calendars
Analytics
Create
Search
Update
<MockAuthProvider scopes={["user"]}>
<App initialEntries={["/admin"]} />
</MockAuthProvider>
);
await waitFor(() => {
expect(screen.getByText(/Only admins allowed/)).toBeInTheDocument();
expect(screen.queryByText(/Admin Page/)).not.toBeInTheDocument();
});
});
LoginCallback
React component that handles Step 3 of the PKCE flow when the auth server redirects the browser back to the SPA in order to verify the state and protect against attacks. This is included in SecureRouter and should only be used directly if you want to implement your own version of SecureRouter instead.
Usage
<Switch>
<Route path="/oauth_callback">
<LoginCallback />
</Route>
</Switch>
Like
Be the first to like this
referencedocs
Powered by Atlassian Confluence 7.16.1 (Cluster node: 43527007) · Report a bug · Atlassian News
ATLASSIAN