**Overall Workflow for lichess.tsx**

The file is a centralized module that manages **Lichess API integration**. It handles authentication, user account management, study handling, PGN importing, and live game updates. These utilities are critical for connecting the application to Lichess, enabling seamless interaction with its features.

**1. Authentication and Authorization**

**Key Functions:**

1. getOauth():

• Initializes the OAuth2 authorization flow using @bity/oauth2-auth-code-pkce.

• Sets up:

• Authorization URL: https://lichess.org/oauth

• Token URL: https://lichess.org/api/token

• Client ID: "lichess-api-demo"

• Redirect URL: Application’s URL.

• Automatically refreshes access tokens when they expire.

• Handles invalid grants.

const oauth = new OAuth2AuthCodePKCE({

    authorizationUrl: `${lichessHost}/oauth`,

    tokenUrl: `${lichessHost}/api/token`,

    clientId,

    scopes,

    redirectUrl: clientUrl,

    onAccessTokenExpiry: refreshAccessToken => refreshAccessToken(),

    onInvalidGrant: console.warn,

});

2. lichessLogin():

• Starts the OAuth2 authorization process using getOauth().

const oauth = getOauth();

oauth.fetchAuthorizationCode();

3. lichessLogout(dispatch):

• Logs the user out by removing OAuth2 state and resetting the Redux user state.

localStorage.removeItem("oauth2authcodepkce-state");

dispatch(userReset());

**Cooperation with Other Files:**

• **Redux**: Updates user state (userSlice) with the token and username.

• **UI Components**: Initiates login/logout flows when the user interacts with authentication buttons.

**2. User Account Management**

**Key Functions:**

1. lichessGetAccount(token):

• Fetches the authenticated user’s account details.

• Used in the login flow to retrieve and store the username.

const path = "/api/account";

const account = fetchBody(token, path);

2. lichessTrySetUser(navigate, dispatch):

• Checks if the user is returning from the Lichess OAuth server.

• Retrieves the access token and user details.

• Navigates to the app’s home page after login.

const accessContext: AccessContext = await oauth.getAccessToken();

const newToken: string | undefined = accessContext?.token?.value;

dispatch(userSetToken(newToken));

const account: any = await lichessGetAccount(newToken);

dispatch(userSetUsername(account.username));

navigate("/");

**3. Study and Broadcast Management**

**Key Functions:**

1. lichessSetStudies():

• Fetches the user’s studies and broadcasts.

• Separates studies that are not part of broadcasts for display.

const path = `/api/broadcast/my-rounds`;

const broadcasts: Study[] = [];

fetchResponse(token, path)

  .then(readStream((response: any) => broadcasts.push({ id: response.round.id, name: response.round.name })))

  .then(() => setBroadcastlessStudies(token, username, setStudies, broadcasts));

2. setBroadcastlessStudies():

• Filters studies that are not linked to broadcasts.

const path = `/api/study/by/${username}`;

fetchResponse(token, path)

  .then(readStream((response: any) => { if (!broadcastIds.includes(response.id)) { studies.push({ id: response.id, name: response.name }); }}))

  .then(() => setStudies(studies));

**4. PGN Management**

**Key Functions:**

1. lichessImportPgn(token, pgn):

• Imports a PGN (Portable Game Notation) file to Lichess.

const path = "/api/import";

const options = { body: new URLSearchParams({ pgn }), method: "POST" };

const data = fetchBody(token, path, options);

2. lichessImportPgnToStudy(token, pgn, name, studyId):

• Adds a PGN to a specific study.

const path = `/api/study/${studyId}/import-pgn`;

const options = { body: new URLSearchParams({ pgn: pgn, name: name }), method: "POST" };

fetchResponse(token, path, options);

**5. Broadcast Round Updates**

**Key Functions:**

1. lichessPushRound():

• Pushes a PGN update to a broadcast round.

const path = `/api/broadcast/round/${roundId}/push`;

const options = { body: pgn, method: "POST" };

fetchResponse(token, path, options);

**Cooperation with Other Files:**

• videoAndSidebar.tsx:

• Invokes lichessPushRound() during “broadcast” mode to update the Lichess round with live moves.

**6. Live Game Streaming**

**Key Functions:**

1. lichessStreamGame():

• Streams live updates of a specific game.

const path = `/api/board/game/stream/${gameId}`;

fetchResponse(token, path).then(readStream(callback));

2. lichessPlayMove():

• Sends a move to Lichess in a live game.

const path = `/api/board/game/${gameId}/move/${move}`;

const options = { method: "POST" };

fetchResponse(token, path, options);

**7. Utility Functions**

**Key Functions:**

1. fetchResponse(token, path, options):

• Fetches an API response, adding the authentication token to the request.

const config: any = { ...options, headers: { Authorization: `Bearer ${token}` } };

const res: any = await window.fetch(`${lichessHost}${path}`, config);

2. readStream(processLine):

• Reads streaming responses (e.g., live game updates) line by line.

const stream = response.body.getReader();

const matcher = /\r?\n/;

const decoder = new TextDecoder();

**Integration with the Rest of the Project**

1. videoAndSidebar.tsx:

• Uses lichessPushRound() for broadcasting game moves in real-time.

2. userSlice.ts:

• Stores user authentication tokens and account details retrieved via this file.

3. gameSlice.ts:

• Syncs with Lichess to update game state (e.g., moves, broadcasts).

This file is the backbone for Lichess-related interactions, enabling OAuth authentication, real-time game updates, and study management while integrating seamlessly with Redux and other project components.