**Expanded Workflow with Full Code Details**

**1. Technical Stack**

• **Frontend**: React (TypeScript), Redux Toolkit for state management.

• **AI Processing**: TensorFlow.js (Chess piece detection and recognition), Chess.js (Chess logic).

• **External Integration**: Lichess API (OAuth2 authentication, chess game syncing).

• **Rendering**: HTML5 Canvas (chessboard visualization and updates).

• **Data Communication**: Fetch API for Lichess requests.

**2. Application Initialization**

**File: src/index.tsx**

This is the entry point of the application:

import React from "react";

import ReactDOM from "react-dom/client";

import { Provider } from "react-redux";

import App from "./App";

import { store } from "./store";

ReactDOM.createRoot(document.getElementById("root")!).render(

  <React.StrictMode>

    <Provider store={store}>

      <App />

    </Provider>

  </React.StrictMode>

);

• **Key Components**:

• Provider: Connects the Redux store to the React application.

• App: The main application component.

**File: src/store.ts**

This file configures the Redux store, combining all slices:

import { configureStore } from "@reduxjs/toolkit";

import userReducer from "./slices/userSlice";

import gameReducer from "./slices/gameSlice";

import cornersReducer from "./slices/cornersSlice";

export const store = configureStore({

  reducer: {

    user: userReducer,

    game: gameReducer,

    corners: cornersReducer,

  },

});

export type RootState = ReturnType<typeof store.getState>;

export type AppDispatch = typeof store.dispatch;

• **Slices**:

• userSlice: Manages user authentication state.

• gameSlice: Tracks chess game logic (moves, board state).

• cornersSlice: Stores chessboard corner detection data.

**3. User Authentication**

**File: src/utils/lichess.ts**

Handles interaction with the Lichess API, including authentication:

const lichessHost = "https://lichess.org";

const scopes = ["study:write", "study:read", "challenge:read", "bot:play", "board:play"];

const clientId = "lichess-api-demo";

const clientUrl = `${location.protocol}//${location.host}/`;

const getOauth = () => {

  const oauth = new OAuth2AuthCodePKCE({

    authorizationUrl: `${lichessHost}/oauth`,

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

    clientId,

    scopes,

    redirectUrl: clientUrl,

    onAccessTokenExpiry: (refreshAccessToken) => refreshAccessToken(),

    onInvalidGrant: console.warn,

  });

  return oauth;

};

• lichessLogin:

export const lichessLogin = () => {

  const oauth = getOauth();

  oauth.fetchAuthorizationCode();

};

• **Token Retrieval**:

export const lichessTrySetUser = async (navigate, dispatch) => {

  const oauth = getOauth();

  if (!(await oauth.isReturningFromAuthServer())) return;

  const accessContext = await oauth.getAccessToken();

  const token = accessContext?.token?.value;

  if (!token) return;

  dispatch(userSetToken(token));

  const account = await lichessGetAccount(token);

  dispatch(userSetUsername(account.username));

  navigate("/");

};

**4. App Component**

**File: src/App.tsx**

Initializes the application context, routing, and renders the layout:

import { useRef } from "react";

import { Routes, Route } from "react-router-dom";

import { GraphModel } from "@tensorflow/tfjs-converter";

import Layout from "./common/Layout";

const App = () => {

  const piecesModelRef = useRef<GraphModel>();

  const xcornersModelRef = useRef<GraphModel>();

  const modelRefs = {

    piecesModelRef,

    xcornersModelRef,

  };

  return (

    <Routes>

      <Route path="/" element={<Layout context={modelRefs} />} />

    </Routes>

  );

};

export default App;

• **Model References**:

• TensorFlow models (piecesModelRef and xcornersModelRef) are initialized here.

• **Routing**:

• The Layout component is used as the base layout for the application.

**5. Chessboard Detection and Processing**

**File: src/common/videoAndSidebar.tsx**

Combines the video processing and sidebar UI:

import Video from "../common/video";

const VideoAndSidebar = ({ mode }: { mode: Mode }) => {

  const context = useOutletContext<ModelRefs>();

  const dispatch = useDispatch();

  const props = { */\* Model references and states \*/* };

  return (

    <Container>

      {Sidebar()}

      <Video {...props} />

    </Container>

  );

};

export default VideoAndSidebar;

**File: src/common/video.tsx**

Processes video input and updates the game state:

useEffect(() => {

  findPieces(piecesModelRef, videoRef, canvasRef, playingRef, setText, dispatch,

    cornersRef, boardRef, movesPairsRef, lastMoveRef, moveTextRef, mode);

}, []);

• **Piece Detection**:

• Uses findPieces to detect chess pieces from video frames.

• Updates the Redux store with game data.

**6. Piece Detection Logic**

**File: src/utils/findPieces.ts**

Implements the core detection logic:

export const findPieces = (

  modelRef, videoRef, canvasRef, playingRef, setText, dispatch, cornersRef

) => {

  const loop = async () => {

    const { boxes, scores } = await detect(modelRef, videoRef, keypoints);

    const update = getUpdate(scores, squares);

    state = updateState(state, update);

    dispatch(gameUpdate(payload));

    renderState(canvasRef.current, centers, boundary, state);

    requestAnimationFrame(loop);

  };

  requestAnimationFrame(loop);

};

**7. Lichess Integration**

**File: src/utils/lichess.ts**

Handles API requests for game synchronization:

• **Push Round**:

export const lichessPushRound = (token, pgn, roundId) => {

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

  const options = {

    body: pgn,

    method: "POST",

  };

  fetchResponse(token, path, options);

};

**8. Rendering**

**File: src/utils/render/renderState.ts**

Renders the chessboard state onto the canvas:

export const renderState = (canvas, centers, boundary, state) => {

  const ctx = canvas.getContext("2d");

  ctx.clearRect(0, 0, canvas.width, canvas.height);

*// Render logic here*

};

**Expanded Workflow**

1. **Startup**:

• index.tsx initializes the Redux store and mounts the App component.

• App.tsx initializes TensorFlow model references and routing.

2. **Authentication**:

• lichessLogin triggers OAuth2 login.

• lichessTrySetUser retrieves the token and sets up the user.

3. **Chessboard Detection**:

• videoAndSidebar.tsx combines video processing with the sidebar.

• video.tsx handles video input and calls findPieces.

4. **Piece Detection**:

• findPieces.ts detects pieces using TensorFlow.js.

• Updates the game state in Redux via gameUpdate.

5. **Synchronization**:

• Game state is pushed to Lichess using lichessPushRound.

6. **Rendering**:

• The canvas is updated with the current chessboard state using renderState.

This revised explanation ensures accurate code references, detailed workflows, and better alignment with the repository.