App.jsx Deep Dive (Shatranj)

This file is the main client application. It wires Firebase, chess logic, tournaments, quick matchmaking, chat, video, sounds, and all screens. Use this as a map when onboarding.

High-level structure
- **Imports:** React hooks, ‘chess.js‘, ‘react-chessboard‘, Firebase client SDK (Auth + Firestore), Tone.js for sounds, local components (‘ConnectBoardModal‘), and a few utilities.
- **Firebase init:** Reads ‘VITE_*‘ env vars to initialize the app, auth, and Firestore instances.
- **Sound system:** ‘soundThemes‘ and ‘playSound‘ provide move/capture/check/game-over sounds using Tone.js.
- **Helpers:** ‘logStep‘ for timestamped debug logs; ‘makeGameId‘, ‘shuffle‘, and ‘incrementTournamentScore‘ transaction helper.
- **Services (Firestore helpers):**
  - ‘createGame‘ â writes a new ‘games/{id}‘ document with initial FEN, timers, chat, captured pieces, WebRTC signal slots, tournament metadata, invite-only flag, spectators fields.
  - ‘createTournament‘ â creates ‘tournaments‘ doc with host, scores map, players array, lobby status.
  - ‘joinTournament‘ â transactionally adds player and initializes their score to 0.
  - ‘startTournament‘ â shuffles players, creates round 1 matches (pairings or BYE), creates games via ‘createGame‘, and sets tournament status/round.
  - ‘checkRoundCompletion‘ â after all matches complete, transactionally advances the round or finishes the tournament; then creates next-round matches sorted by scores (Swiss-ish pairing).
  - ‘updateTournamentMatchResult‘ â marks a match complete, updates scores (win/draw), and triggers round completion.
- **UI components defined inside App.jsx:** Auth form, lobby and tournament UIs, game pages, clocks, dialogs, chat, video chat, settings, etc. (details below).
- **Main ‘App‘ component:** Manages auth state, routing between views, live Firestore subscriptions, move handling, timers, premoves, AI moves, hardware moves, WebRTC signaling, and URL deep links for games/tournaments.

Data shapes (client-side expectations)
- **Game doc:** ‘mode‘, ‘timeControl‘, ‘player1/2 { uid, email }‘, ‘playerIds‘, ‘fen‘, ‘moves[]‘, ‘chatMessages[]‘, ‘capturedPieces‘, ‘status‘, ‘winner‘, ‘winReason‘, ‘drawOffer‘, ‘rematchOffer‘, ‘rematchedGameId‘, ‘webrtc_signals { offer, answer, iceCandidates[] }‘, ‘createdAt‘, ‘lastMoveTimestamp‘, per-player clocks, ‘lastMove‘, ‘tournamentId‘, ‘tournamentRound‘, ‘inviteOnly‘, ‘spectators‘, ‘spectatorRequests‘, ‘createdByUid‘.
- **Tournament doc:** ‘name‘, ‘maxPlayers‘, ‘timeControl‘, ‘createdBy‘, ‘players[]‘, ‘scores{}‘, ‘status‘, ‘currentRound‘, ‘winnerId‘, ‘createdAt‘; subcollection ‘rounds/round_{n}/matches‘ with ‘playerWhite/Black‘, ‘gameId‘, ‘status‘, ‘winnerId‘, ‘isBye‘.
- **Matchmaking doc:** ‘{ uid, email, timeControl, createdAt }‘ (paired by Cloud Function).
- **Hardware:** ‘boards/{boardId}‘ and ‘hardware_moves/{boardCode}‘ (moves delivered to

client for hardware-test mode).

Components inside App.jsx
ErrorBoundary
Catches render errors and shows a friendly reload prompt.

AuthForm
Email/password login and registration via Firebase Auth. Uses ‘onAuthStateChanged‘ in the main App to drive routing.

CreateTournamentModal
Collects name, max players, time control, and whether the host plays; calls ‘createTournament‘ then navigates to the lobby.

TournamentLobby
- Subscribes to the tournament doc in real-time.
- Shows players, status, shareable link, standings, and current round matches.
- Host controls: start tournament, force match result, kick player.
- Joining: ‘joinTournament‘ transaction; Host starts via ‘startTournament‘.
- For each match: play (if youâ re in it and ongoing) or watch (spectator) buttons.

GameSetup (main lobby)
- Lists open waiting games (not invite-only) and open tournaments (status=lobby) via ‘onSnapshot‘.
- Create game: uses ‘createGame‘ with selected time control and optional invite-only flag.
- Join game: transactionally claims ‘player2‘ on a waiting game.
- Quick Match: tries to claim an existing waiting game with same time control; otherwise writes a ‘matchmaking_requests‘ doc and listens for a newly created game involving you.
- Other buttons: vs computer (local AI), pass-and-play offline, connect hardware board (opens ‘ConnectBoardModal‘), create tournament.

ProfilePage
Fetches last 50 finished games for the user; links to review.

GameReviewPage
Replays finished games by walking move list; uses ‘Chessboard‘ in read-only mode.

GameClocks
Derives remaining time from stored clocks + ‘lastMoveTimestamp‘; detects and reports timeouts to Firestore.

PromotionDialog
Modal for pawn promotion choice; feeds back to move handler.

ChatBox
Inline chat writing to ‘games.chatMessages‘ (arrayUnion). Scrolls to bottom on new messages.

GameOverDialog
Shown when status is ‘finished‘; indicates winner/reason. Supports rematch offer/accept for non-tournament online games.

GameActions
Draw offer/accept/decline and resign. Updates Firestore and tournament match results when applicable.

CapturedPiecesPanel
Displays captured pieces (icons from chess.com CDN) sorted by piece importance.

SettingsDialog
Local settings persisted to ‘localStorage‘: sound on/off, sound theme, premoves, highlight legal moves.

VideoChat
WebRTC 1:1 video/audio between the two players only. Flow:
- Creates ‘RTCPeerConnection‘ with public STUN.
- Grabs local media; adds tracks to PC.
- Player1 creates offer â stored in ‘games.webrtc_signals.offer‘.
- Player2 reads offer, creates answer â ‘webrtc_signals.answer‘.
- ICE candidates exchanged via ‘webrtc_signals.iceCandidates‘ array; each candidate tagged with sender UID and de-duped.
- Blocks host/spectators (message shown) to avoid breaking the 1:1 signaling model.

Main App component (behavior)
Global state
‘user‘, ‘isAuthReady‘, ‘view‘, ‘gameId‘, ‘tournamentId‘, ‘gameData‘, ‘reviewGameData‘, promotion state, settings, premove, move highlights, modals (connect board, create tournament), mobile accordion state, and refs for game data, hardware move handler, request listeners, etc.

Derived values
- ‘fen‘ from ‘gameData‘.
- ‘game‘ from ‘chess.js‘ built from FEN.
- ‘playerOrientation‘ (white if player1, black if player2, white for offline/hardware).

Effects and subscriptions
- **Auth listener:** ‘onAuthStateChanged‘ sets ‘user‘ and resets game/view when signed out.
- **Game subscription:** ‘onSnapshot‘ on ‘games/{gameId}‘ unless it is a local/offline game; updates ‘gameData‘.
- **Tournament redirect:** When a tournament game finishes, auto-open next active game in that tournament if found.
- **URL deep links:** If ‘?tournamentId‘ present after auth, open tournament lobby; if ‘?gameId‘, join/spectate that game (transactionally claim if waiting).

- **AI moves:** For ‘mode === ’computer’‘, black makes a random legal move after 500ms.
- **Move highlights:** Tracks last move for square highlighting.
- **Hardware moves:** For ‘mode === ’hardware_test’‘, subscribe to ‘hardware_moves/{boardCode}‘ and apply new moves in order.
- **Settings persistence:** Syncs to ‘localStorage‘.

Move handling
- ‘makeMove‘ validates with ‘chess.js‘, plays sounds, updates ‘capturedPieces‘, determines game over, and writes to Firestore for online games (including clocks, lastMove, drawOffer reset, game events subcollection). For tournament games, updates match results.
- ‘onDrop‘ (board interaction) guards turn logic, premoves, promotions, and calls ‘makeMove‘.
- ‘handleSelectPromotion‘ finishes stored promotion intent.
- Premove: if it is not your turn and premoves are enabled, stores the move and executes it when your turn arrives.
- Timeouts: ‘handleTimeout‘ marks winner by time in Firestore and tournament match.

Game lifecycle helpers
- ‘handleStartVsComputer‘, ‘handleStartOfflineGame‘, ‘handleConnectHardwareGame‘ create local gameData for respective modes.
- ‘leaveGame‘ returns to lobby or tournament lobby depending on ‘tournamentId‘.
- ‘handleRematch‘/‘rematchOffer‘ flow for non-tournament online games; auto-joins ‘rematchedGameId‘ when present.

UI rendering
- ‘renderContent‘ switches among views: lobby (GameSetup), tournament lobby, game, profile, review.
- Header shows user email, profile button, logout, and settings.
- Settings and modals rendered portal-style within main layout.

Data flow highlights
- **Creating games:** Lobby â ‘createGame‘ writes Firestore â game screen subscribes to ‘games/{id}‘.
- **Joining games:** Transaction updates ‘player2‘ and status to ‘active‘, sets clocks.
- **Quick Match:** Firestore Cloud Function pairs ‘matchmaking_requests‘ and creates a game; client listens for new active game containing the user.
- **Tournaments:** Host starts â creates games for matches. Match completion flows into score updates and potential round advancement; Swiss-ish pairing sorted by scores for subsequent rounds.
- **WebRTC:** Signals stored on the game doc; only two participants supported.
- **Hardware:** Local game uses Firestore ‘hardware_moves‘ as an input feed; does not write moves back.

Extensibility notes
- Split App.jsx into feature modules (auth, lobby, tournaments, game play, video, hardware) for testability.

- Move shared Firestore types to a separate file and consider TypeScript for safer payloads.
- WebRTC for >2 peers or host/spectators would need a mesh/SFU design and different signaling structure.
- Hardware collections are openly writable by signed-in users per current rules; lock down for production.
- Tailwind is included via CDN; migrate to build-time Tailwind for theming/purge if needed.

Key entry points in code
- Firebase init and config: top of ‘src/App.jsx‘.
- Service helpers: ‘createGame‘, tournament helpers, WebRTC helper flows.
- Lobby UX: ‘GameSetup‘ component.
- Tournament UX: ‘TournamentLobby‘ + round/match helpers.
- Game UX and logic: ‘renderContent‘ branch for ‘view === ’game’‘ plus ‘makeMove‘, ‘onDrop‘, ‘GameActions‘, ‘GameOverDialog‘, ‘GameClocks‘, ‘VideoChat‘, ‘ChatBox‘.
- Navigation + deep links: ‘useEffect‘ blocks near the bottom of ‘App‘ for query params.