

PRIZEPICKS_PREDICTIONWEBSITE/

```
├── backEnd/
│   ├── app.py
│   ├── backtester.py
│   ├── chatgpt_bet_explainer.py
│   ├── injury_report.py
│   ├── main.py
│   ├── monte_carlo.py
│   ├── player_analyzer.py
│   ├── prediction_analyzer.py
│   ├── requirements.txt
│   ├── screenshot_parser.py
│   └── volatility.py
├── frontEnd/
│   ├── public/
│   │   └── mobile-viewport.html
│   ├── src/
│   │   ├── components/
│   │   │   ├── ActiveBet.jsx
│   │   │   ├── AdvancedMetricsCard.jsx
│   │   │   ├── ApiTest.jsx
│   │   │   ├── BetConfirmation.jsx
│   │   │   ├── BetExplanationCard.jsx
│   │   │   ├── BetSlip.jsx
│   │   │   ├── chatgpt-thinking.css
│   │   │   ├── ChatGptThinking.jsx
│   │   │   ├── DailyPicks.jsx
│   │   │   ├── Dashboard.jsx
│   │   │   ├── EditBetModal.jsx
│   │   │   ├── FavoritePlayers.jsx
│   │   │   ├── ImageWithFallback.jsx
│   │   │   ├── InjuryStatusCard.jsx
│   │   │   ├── MobileLayout.jsx
│   │   │   ├── MobileOptimizedDashboard.jsx
│   │   │   ├── MobilePlayerCard.jsx
│   │   │   ├── MonteCarloCard.jsx
│   │   │   ├── Notifications.jsx
│   │   │   ├── PlayerAnalysisDashboard.jsx
│   │   │   ├── PlayerAnalysisModal.jsx
│   │   │   ├── PlayerAnalysisSearch.jsx
│   │   │   ├── PlayerCard.jsx
│   │   │   ├── PlayerStatsModal.jsx
│   │   │   ├── PredictionCard.jsx
│   │   │   ├── PreviousBets.jsx
│   │   │   ├── processed-players.css
│   │   │   ├── ProcessedPlayers.jsx
│   │   │   ├── RecommendationCard.jsx
│   │   │   └── ScreenshotUploader.jsx
```

```

├── StatsCard.jsx
├── thinking-animation.css
├── ThinkingAnimation.jsx
├── TrendingPicks.jsx
├── pages/
│   ├── HomePage.jsx
│   └── SignIn.jsx
├── scripts/
│   ├── initDatabase.js
│   ├── migrateData.js
│   ├── mobile-viewport.js
│   └── use-mobile-detector.js
├── services/
│   ├── api.js
│   └── firebaseService.js
├── App.css
├── App.jsx
├── firebase.js
├── index.css
├── main.jsx
├── functions/
│   └── index.js
├── injury_report_fn/
│   ├── index.css
│   └── main.jsx
├── .firebaserc
├── firebase.json
└── README.md

```

More information on every File:

backEnd/

File	Purpose in one glance	Key responsibilities / notes
app.py	Flask API gateway	<ul style="list-style-type: none"> Boots the Flask server, enables CORS, and initializes Firebase Admin. Hosts routes such as /api/parse_screenshot (OCR) and /api/player (full analysis). Persists results to Firestore and returns JSON to the front-end.

<code>backtester.py</code>	Historical profit-and-loss simulator	<ul style="list-style-type: none"> Scans prior <code>processedPlayers/*</code> docs, applies a simple bet-settlement rule, and builds a P&L Series. Useful for validating the model or generating performance charts in notebooks.
<code>chatgpt_bet_explainer.py</code>	Natural-language "Why this bet?" generator	<ul style="list-style-type: none"> Crafts a prompt with player stats + probabilities, calls the OpenAI ChatGPT API, and returns a concise explanation. Cached in Firestore so each pick is explained only once.
<code>injury_report.py</code>	Live injury-status scraper	<ul style="list-style-type: none"> Pulls the NBA's official daily injury feed (or a mirrored JSON). Normalizes status (Out, Q, P) and injury details, returning a clean dict keyed by NBA player ID. Consumed by <code>player_analyzer.py</code> to adjust probabilities.
<code>main.py</code>	Cloud-Run entry-point + cron helpers	<ul style="list-style-type: none"> Exposes <code>app</code> for Gunicorn and contains scheduled logic that: <ul style="list-style-type: none"> polls recent box scores; moves finished picks from <i>active</i> → <i>concluded</i>; updates user bet history.
<code>monte_carlo.py</code>	Python wrapper around native Monte-Carlo engine	<ul style="list-style-type: none"> Fetches ≤ 60 recent games → computes μ, σ → runs 100 000 sims. Prefers the ultra-fast shared lib <code>libmontecarlo.so</code> (see below) but can fall back to NumPy.
<code>player_analyzer.py</code>	Master data wrangler & feature builder	<ul style="list-style-type: none"> Queries <code>nba_api</code> for season stats, last-5 logs, playoff data, opponent strength, etc. Calls <code>injury_report</code>, <code>volatility.forecast_volatility</code>, <code>prediction_analyzer.poisson_over_prob</code>, and <code>monte_carlo.monte_carlo_probability</code>. Bundles everything into a dict that the front-end cards expect.
<code>prediction_analyzer.py</code>	Math helpers (Poisson & misc.)	<ul style="list-style-type: none"> Implements closed-form Poisson "\geq threshold" calculation. Provides thin wrappers invoked by <code>player_analyzer</code> and feeds into <code>chatgpt_bet_explainer</code>.
<code>volatility.py</code>	GARCH(1,1) volatility forecaster	<ul style="list-style-type: none"> Builds a 50-game (or playoff-only) series of point "returns", fits <code>arch_model</code>, and returns 1-step-ahead σ. Output stored as <code>volatilityForecast</code> / <code>volatilityPlayOffsForecast</code>.

<code>HomePage.jsx</code>	Main authenticated dashboard SPA	<ul style="list-style-type: none"> • Orchestrates every major front-end feature once a user is signed-in. • Pulls user profile, active bets, historical bets and legacy picks from Firestore via <code>firebaseService</code> and keeps them in React state. • Hosts the tabbed layout (Dashboard ▶ Processed Players ▶ Previous Bets ▶ Notifications) and renders dozens of child components—e.g. <code>PlayerAnalysisSearch</code>, <code>ScreenshotUploader</code>, <code>ActiveBet</code>, <code>BetSlip</code>, <code>PlayerAnalysisDashboard</code>, etc. • Implements client-side bet creation / editing / cancel flows, plus logic for moving completed bets to history. • Acts as a central router-less “controller” page that wires together fetch calls (<code>/api/player</code>) and UI, handling loading & error states.
<code>SignIn.jsx</code>	Simple username + password login screen	<ul style="list-style-type: none"> • Displays a stylized sign-in form (username / password with show-password toggle). • On submit, looks up the user in Firestore (<code>getUserByUsername</code>), verifies password, runs any first-time <code>initializeUser</code> / <code>initializeDatabase</code> migration, then stashes the username in <code>sessionStorage</code>. • Redirects successful logins to <code>/HomePage</code> via React Router’s <code>useNavigate</code>. • Runs a one-time database bootstrap (<code>initializeDatabase("bryanram")</code>) on mount so demo users can sign in without manual setup.
frontEnd/src/		
File	Purpose in one glance	Key responsibilities / notable details
<code>App.css</code>	Global styling + mobile-first overrides	<ul style="list-style-type: none"> • Provides core layout rules for <code>#root</code>, logo hover effects, and the default Vite card styles. • A max-width 768 px media query force-stacks flex layouts, makes tables horizontally scrollable, converts modals to full-screen, and ensures every element obeys responsive sizing—turning the desktop SPA into a fluid mobile experience.
<code>App.jsx</code>	Top-level React router	<ul style="list-style-type: none"> • Uses React Router v6 to map <code>/</code> → <code>SignIn</code> and <code>/HomePage</code> → <code>HomePage</code>. • Stateless functional component; exported as default so <code>main.jsx</code> can mount it.
<code>firebase.js</code>	Front-end Firebase initializer	<ul style="list-style-type: none"> • Reads API keys & IDs from Vite env variables and calls <code>initializeApp()</code>. • Exports Firestore instance <code>db</code> and, in browser environments only, <code>analytics</code>—guarded so SSR or Node tests don’t break.

<code>index.css</code>	Tailwind layer injection	• Simply imports <code>@tailwind base, components, and utilities</code> ; actual custom styles live in individual component <code>.css</code> files.
<code>main.jsx</code>	React entry point rendered by Vite	• Boots the app via <code>createRoot().render(<StrictMode><App /></StrictMode>)</code> . • Pulls in <code>index.css</code> so Tailwind styles apply globally before any component mounts.

frontEnd/src/components/

File	Purpose in one glance	Key responsibilities / notable UI behavior
<code>PredictionCard.jsx</code>	Mini card that surfaces the model's score prediction for <i>one</i> player	• Shows threshold, computed probability and Poisson probability. • Colors the <i>Recommendation</i> banner green / yellow / red based on <code>prediction.category</code> ("Almost guaranteed", "Neutral", "Risky").
<code>PreviousBets.jsx</code>	Accordion list of a user's completed and in-flight wagers	• Splits view into Active Bets and Completed Bets sections. • Click to expand → reveals pick details, result hit/ miss chips, and P&L. • Local <code>expandedBets</code> state tracks which items are unfolded.
<code>processed-players.css</code>	Hover & animation helpers for <i>ProcessedPlayers</i> grid	• Adds scale-up shadow on <code>.player-card: hover</code> and a reusable <code>.pulse</code> key-frame for success confirmations. • General transition rules applied to buttons & expandable sections.
<code>ProcessedPlayers.jsx</code>	Searchable / filterable gallery of players already analyzed server-side	• Fetches Firestore docs via <code>getProcessedPlayers()</code> and displays responsive card grid. • Live filters: team dropdown, AI-recommendation dropdown, free-text search. • Each card opens <code>PlayerAnalysisModal</code> or emits <code>onAddToPicks</code> . • Maintains <code>addedPlayers</code> map to flash a <input checked="" type="checkbox"/> for 3 s after a pick is queued.

<code>RecommendationCard.jsx</code>	Lightweight “quick verdict” box used inside the analysis modal	<ul style="list-style-type: none"> • Derives OVER/UNDER & confidence (High / Medium) from basic averages vs threshold. • Shows Poisson probability and icon-based sentiment (🟢 green / 🟡 red).
<code>ScreenshotUploader.jsx</code>	Drag-and-drop widget that parses PrizePicks images, then chains player analysis	<ul style="list-style-type: none"> • Lets users drop multiple screenshots, previews them, and tracks upload % with a fake progress bar. • Calls <code>/api/parse_screenshot</code> → receives <code>{ parsedPlayers[] }</code> → sequentially POSTs each to <code>/api/player</code> while painting per-row status chips (spinner / ✅ / ❌). • Auto-clears previews when done; shows animated success & error toasts.
<code>StatsCard.jsx</code>	Detailed stat panel inside the analysis modal	<ul style="list-style-type: none"> • Displays season avg, last-5 avg, vs-opponent avg, home/away avg. • Inline table of last 5 games, color-coded vs threshold. • “See More” button opens modal paginated (10 per page) over up to 15 games, with lazy “Load more games” pagination.
<code>thinking-animation.css</code>	Re-usable pulse animation for ChatGPT “thinking” loader	<ul style="list-style-type: none"> • <code>.thinking-dot</code> staggered dot pulse and <code>.thinking-ring</code> breathing ring key-frames.
<code>ThinkingAnimation.jsx</code>	Centered loader component shown while awaiting AI response	<ul style="list-style-type: none"> • Uses the above CSS to render a gradient ring, three pulsing dots, and explanatory blurb (“Gathering player statistics...”). • Accepts optional <code>text</code> prop (default “Analyzing”).
<code>TrendingPicks.jsx</code>	Static demo card of “most popular picks”	<ul style="list-style-type: none"> • Currently hard-codes an array of three players with popularity %, threshold and recommendation; renders with icons & team info.
<code>FavoritePlayers.jsx</code>	Empty-state panel for future “starred” players	<ul style="list-style-type: none"> • Renders a grid of favorite player tiles once data exists; for now shows a big prompt and “Add Players” CTA button using Lucide icons.

`ImageWithFallback.jsx`

Re-usable `` wrapper that never breaks

- Attempts primary `src`; on `onError` swaps to `fallbackSrc` (or `/placeholder.svg`) so broken images don't wreck the layout.

`InjuryStatusCard.jsx`

Rich card that visualizes a player's latest injury report

- Three states: no data, found on report, healthy.
- Maps status → color (red = Out, yellow = Questionable, green = Probable/Available).
- Shows reason, game date/time, and matchup when available.

`MonteCarloCard.jsx`

Explains the Monte-Carlo simulation result for a threshold

- Displays probability (green / yellow / red), distribution type, and an info blurb with a chart icon.
- Parses string or numeric inputs and formats to `##.## %`.

`Notifications.jsx`

Placeholder settings panel for future alerts

- Static copy describing upcoming features (game-start, performance, result alerts).
- Disabled toggle switches communicate “coming soon.”

`PlayerAnalysisDashboard.jsx`

In-page deep-dive dashboard shown after a search

- Hero banner with player info, logos, threshold, AI recommendation and Poisson + Monte-Carlo probs.
- Key-stats tiles, volatility tiles, expandable tables (all-season encounters, playoffs log, recent games).
- Fetches extra games on demand, color-codes vs-threshold, and exposes `onAddToPicks`.

`PlayerAnalysisModal.jsx`

Full-screen modal version of the above dashboard

- Same data logic but scrolls inside a modal; close button, add-to-picks icon, expandable sections.
- Accepts `playerData`, `onClose`, `onAddToPicks` props; fetches extra games lazily.

<code>PlayerAnalysisSearch.jsx</code>	Smart search bar that drives the analysis flow	<ul style="list-style-type: none"> • Handles player name + threshold inputs, keeps a <i>recent searches</i> dropdown in <code>localStorage</code>, shows helpful tips pane, and forwards calls via <code>onSearch</code>.• While back-end is working, displays <code>ChatGptThinking</code> loader.
<code>PlayerCard.jsx</code>	Simple summary card for list views	<ul style="list-style-type: none"> • Shows photo, team/opponent logos, ranks, next-game info—used in processed players & search suggestions.
<code>PlayerStatsModal.jsx</code>	Lightweight modal for viewing bet details from <i>Previous Bets</i>	<ul style="list-style-type: none"> • On mount, tries to hydrate from Firestore (<code>getProcessedPlayer</code>) for richer stats; falls back to passed prop.• Presents threshold, recommendation, timings, and (if available) season / last-5 / vs-opponent averages and actual result.
<code>AdvancedMetricsCard.jsx</code>	Shows eFG%, 3-pt shot share, FT-rate & splits	<ul style="list-style-type: none"> • Renders advanced metrics grid plus career-season table.• Includes an info call-out explaining each stat.
<code>ApiTest.jsx</code>	Connectivity checker for the Flask API	<ul style="list-style-type: none"> • Calls <code>testAPI()</code> on mount, shows <code>ChatGptThinking</code> loader until response, then prints success or error with a Retry button.
<code>BetConfirmation.jsx</code>	Post-submission modal summarizing a locked bet	<ul style="list-style-type: none"> • Displays platform logo, bet amount, potential winnings, and a scrollable list of selected picks; closes on Done or ✕.
<code>BetExplanationCard.jsx</code>	AI narrative block (“Why this bet?”)	<ul style="list-style-type: none"> • Chooses up/down/warning icon & colors from recommendation, shows ChatGPT text plus Poisson & Monte-Carlo %s, with fallback No Recommendation state.
<code>BetSlip.jsx</code>	Full-screen wizard to assemble & confirm a wager	<ul style="list-style-type: none"> • Lets user pick platform, bet type, amount, choose which picks to include, and computes potential winnings; fires <code>onConfirm</code> with formatted picks.

<code>chatgpt-thinking.css</code>	Dot-pulse & logo-glow animation for loaders	<ul style="list-style-type: none"> • Defines <code>.chatgpt-thinking-dot</code> key-frames and <code>.logo-pulse</code> halo.
<code>ChatGptThinking.jsx</code>	Loader component that uses the above CSS	<ul style="list-style-type: none"> • Shows ChatGPT logo + 3 pulsing dots and optional status text.
<code>DailyPicks.jsx</code>	“Today’s picks” & performance tracker panel	<ul style="list-style-type: none"> • If no picks: friendly empty state. • Otherwise lists each locked pick, game details, and a mini KPI row (count / winnings / bet amount).
<code>Dashboard.jsx</code>	Self-contained demo dashboard for manual testing	<ul style="list-style-type: none"> • Combines pick list, search form, and stat display without Firestore; supports add/remove/lock-in flows and renders <code>PlayerCard</code>, <code>StatsCard</code>, <code>RecommendationCard</code>.
<code>EditBetModal.jsx</code>	Modal to tweak an existing bet before settlement	<ul style="list-style-type: none"> • Lets user change amount, platform, bet type & which picks are included; recalculates winnings; validates at least one pick selected.
<code>MobileLayout.jsx</code>	Responsive shell that swaps between a desktop sidebar and a collapsible mobile drawer	<ul style="list-style-type: none"> • Sticky mobile header with hamburger / close icon toggling <code>sidebarOpen</code> state. • Desktop: fixed 64-px-wide sidebar listing Home, Processed Players, Previous Bets, Alerts. • Mobile: full-height overlay that closes on outside click. • Wraps <code>{children}</code> so any page can inherit the layout.
<code>MobileOptimizedDashboard.jsx</code>	All-in-one mobile dashboard workflow (pick list → search → stats)	<ul style="list-style-type: none"> • Shows Your Picks panel with lock-in, remove and live count (<code>picks.length / 6</code>). • Responsive analysis form (player + threshold) and analyze button. • Renders <code>PlayerCard</code>, <code>StatsCard</code>, <code>RecommendationCard</code>, last-5 games table, and Add to Picks CTA after a search. • Uses props for all handlers so state lives one level up.

MobilePlayerCard.jsx

Compact player-info tile for small screens

- Displays photo (with fallback), name, team, position, team / opponent playoff ranks, and next-game info.
- Fully responsive flex layout that stacks on narrow widths.

ActiveBet.jsx

Expandable card summarizing an in-progress wager

- Lists each active bet; click header toggles expanded view via `expandedBets` state.
- Top bar shows Active Bet title, pulsing alert icon, and Edit / Cancel buttons (prop-driven).
- Expanded: two-column grid with bet metadata (amount, platform, status) and pick list; each pick clickable for deeper info.
- Color-codes pick status (Final = green, Live = yellow).

frontEnd/src/scripts/

File	Purpose in one glance	Key responsibilities / notable elements
mobile-viewport.js	Forces proper mobile scaling when the main app is loaded from <code>/public/mobile-viewport.html</code>	<ul style="list-style-type: none">• Immediately-invoked function checks whether a <code><meta name="viewport"></code> already exists; if not, it injects one with <code>initial-scale=1</code>, <code>maximum-scale=1</code>, and <code>user-scalable=no</code> to lock the zoom level.
use-mobile-detector.js	Lightweight React hook to tell components “are we on a phone?”	<ul style="list-style-type: none">• Keeps an <code>isMobile</code> state that is <code>true</code> when <code>window.innerWidth < 768 px</code>.• Listens for <code>resize</code> events so the flag updates dynamically if the user rotates or resizes the window.
initDatabase.js	One-time Firestore seeder for local/dev demos	<ul style="list-style-type: none">• When executed, creates a demo user <code>bryanram</code> (with a basic profile), admin site-wide stats, and an admin user list document if they don’t already exist.• Uses <code>serverTimestamp()</code> so created/last-login times are server-authoritative.
migrateData.js	Script to reorganize legacy user docs into the new schema	<ul style="list-style-type: none">• Reads the old flat user document, rolls username/password/email into a nested <code>profile</code> object, and writes it back via <code>updateDoc</code>.• Splits historic <code>bets</code> array into <code>activeBets</code> and month-bucketed <code>betHistory</code> sub-collections, transforming each bet to the new field names along the way.

frontEnd/src/services/

File	Purpose in one glance	Key responsibilities / notable elements
<code>api.js</code>	Thin wrapper around your <i>Flask</i> back-end	<ul style="list-style-type: none"> • Single exported helper <code>analyzePlayer(playerName, threshold)</code> that POSTs to <code>/api/player</code>. • Cleans / type-casts the threshold, maps legacy field names (<code>nba_player_id</code> → <code>playerId</code>, etc.) and inserts fall-back images so the UI never breaks. • Ensures dates are ISO-formatted and provides default “Unknown Team/Opponent” strings when the back-end response is incomplete.
<code>firebaseService.js</code>	Firestore data-access layer	Auth / profile helpers – <code>getUserByUsername</code> , <code>verifyUserPassword</code> , <code>getUserProfile</code> , <code>updateUserProfile</code> . Stats & picks – <code>updateUserStats</code> , <code>addUserPick</code> , <code>removeUserPick</code> , <code>getUserPicks</code> , <code>clearUserPicks</code> . Bet management – <code>createBet</code> , <code>getActiveBets</code> , <code>cancelActiveBet</code> , <code>updateActiveBet</code> , <code>getBetHistory</code> , <code>getAllBetHistory</code> . Schema utilities – <code>initializeDatabase</code> and <code>initializeUser</code> set up or migrate user docs to the new profile-first structure; <code>migrateData.js</code> relies on these. Processed players – <code>getProcessedPlayers</code> & <code>getProcessedPlayer</code> pull the shared analysis docs for dashboard views.

`functions/index.js` – Firebase Cloud Functions triggers

Export	Fires when...	What it does
<code>onPlayerStatusChange</code>	A document under <code>processedPlayers/players/active/{docId}</code> is updated	<ul style="list-style-type: none"> • If the field <code>gameStatus</code> flips to “Concluded” the function moves that document into <code>processedPlayers/players/concluded/{docId}</code> and deletes it from <code>active</code>—keeping the two sub-collections mutually exclusive.
<code>onActiveBetWrite</code>	A user's bet is written at <code>users/{userId}/activeBets/{betId}</code> (create / update / delete)	<ul style="list-style-type: none"> • When the bet is deleted or its <code>status</code> transitions to a terminal state (<code>Concluded</code>, <code>Completed</code>, <code>Won</code>, <code>Lost</code>) the pick is copied into <code>users/{userId}/betHistory/{betId}</code> with a <code>settledAt</code> server timestamp, then removed from <code>activeBets</code>.

<code>onUserPicksUpdate</code>	The top-level user doc <code>users/{userId}</code> is updated	<ul style="list-style-type: none"> • Compares the previous <code>picks</code> array to the new one; if any picks now have <code>gameStatus: "Concluded"</code> they're filtered out so the array only contains still-live picks—preventing stale cards from showing up in the UI.
--------------------------------	---	--

injury_report_fn/

File	Purpose in one glance	Key responsibilities / notable elements
<code>full_injury_report.py</code>	Deep PDF scraper that converts the NBA's official daily-injury PDF into clean JSON	<ul style="list-style-type: none"> • Re-creates the NBA's PDF URL based on the most recent "eastern-time" release window (reports drop 12 p.m./5 p.m./8 p.m. ET). • Downloads the PDF, opens it with <code>pdfplumber</code>, and uses explicit x-coordinates to pull a column-perfect table. • Normalizes team names, splits camel-cased headings, swaps "Lastname, Firstname" → "Firstname Lastname," and strips line-breaks from the <i>Reason</i> column. • Exposes two helpers: <ul style="list-style-type: none"> • <code>get_full_injury_report()</code> → <code>list < {gameDate, gameTime, team, player, status, reason}</code> for every entry. • <code>get_player_status(name)</code> → quick lookup returning <code>{status, reason}</code> (or "NOT YET SUBMITTED" if a team hasn't filed).
<code>main.py</code>	Cloud-Function handler that keeps Firestore documents in sync with real-time game status	<ul style="list-style-type: none"> • Utility <code>fetch_game_status()</code> hits <code>nba_api</code>'s <code>ScoreboardV2</code> & <code>BoxScoreTraditionalV2</code> to see whether a game has started, is live, or has finished—and, if finished, grabs the player's final points. • <code>check_active_players()</code> walks the <code>processedPlayers/players/active</code> collection; if <code>fetch_game_status()</code> returns new info, it calls <code>update_doc()</code> to patch the Firestore doc (e.g., set <code>"gameStatus": "Concluded"</code> and <code>"finalPoints": n</code>). • Stubs for <code>check_user_picks()</code> and <code>check_active_bets()</code> illustrate the same pattern for user bet docs. • <code>check_games_handler(request)</code> is the HTTP entry-point wired to Cloud Scheduler (runs every hour); it invokes <code>check_active_players()</code> and responds 200 OK or 500 on error.