

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
│       │   ├── AppLayout.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/
│   ├── DashboardPage.jsx
│   ├── ProcessedPlayersPage.jsx
│   ├── PreviousBetsPage.jsx
│   ├── AlertsPage.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
<code>app.py</code>	Flask API gateway	<ul style="list-style-type: none"> Boots the Flask server, enables CORS, and initializes Firebase Admin. Hosts routes such as <code>/api/parse_screenshot</code> (OCR) and <code>/api/player</code> (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>screenshot_parser.py</code>	OCR extractor	<ul style="list-style-type: none"> Accepts base-64 images, calls OpenAI Vision, parses player / threshold pairs, and returns them to <code>app.py</code>.
<code>requirements.txt</code>	Python dependency list	<ul style="list-style-type: none"> Flask, <code>nba_api</code>, <code>arch</code>, <code>firebase-admin</code>, <code>openai</code>, etc.—installed in Stage 2 of the Docker build.
<code>mc_stub.c</code> & <code>montecarlo.ml</code>	Native speed layer for Monte-Carlo	<ul style="list-style-type: none"> <code>montecarlo.ml</code> → OCaml routine that performs the random draws. • <code>mc_stub.c</code> bridges Python ↔ OCaml via <code>ctypes</code>, producing <code>libmontecarlo.so</code> during the Docker build.
<code>Dockerfile</code>	Two-stage container build	<p>Stage 1 (OCaml): 1. Starts from <code>ocaml/opam</code>, installs OCaml + <code>ctypes</code>. 2. Compiles <code>montecarlo.ml</code> into a PIC object, compiles <code>mc_stub.c</code>, links both into libmontecarlo.so.</p> <p>Stage 2 (Python runtime): 1. <code>python:3.9-slim</code>, installs <code>libffi</code> and Python deps from <code>requirements.txt</code>. 2. Copies the compiled <code>.so</code> and all back-end source files. 3. Launches Gunicorn (CMD <code>gunicorn app:app --bind 0.0.0.0:\${PORT:-8080} ...</code>).</p>

frontEnd/

File	Purpose in one glance	Key responsibilities / notable details
<code>tailwind.config.js</code>	Design-system config for Tailwind CSS	<ul style="list-style-type: none"> Specifies content scan globs (<code>index.html</code>, all files under <code>src/</code>) so unused classes are purged from production builds. • Extends the default theme with custom breakpoints (<code>xs</code> 475 px to <code>2xl</code> 1536 px), extra spacing steps (18, 88, 128), a granular font-scale (<code>xs</code> → <code>4xl</code>), and utilities like <code>minHeight.touch</code> (44 px iOS tap target) and <code>maxWidth.mobile</code> (100 vw). • Adds reusable animations & keyframes – slide-in/out, fade-in/out, spin, pulse – referenced by class names such as <code>animate-slide-in-right</code>. • No additional plugins loaded; relies solely on Tailwind core.

frontEnd/public/

File	Purpose in one glance	Key details
------	-----------------------	-------------

mobile-viewport.html	Mini HTML shim that forces mobile-friendly scaling and blocks pinch/double-tap zoom	<ul style="list-style-type: none"> • Declares a restrictive <code><meta name="viewport" ...></code> so the SPA renders at 100 % width on phones. • Inline IIFE listens for touchstart (multi-touch) and touchend events to preventDefault()—stopping iOS pinch-zoom and the 300 ms double-tap zoom gesture. • Contains no UI markup; it simply injects these behaviors before the React bundle mounts.
-----------------------------	-------------------------------------------------------------------------------------	--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------




frontEnd/src/pages

File	Purpose in one glance	Key responsibilities / notable elements
DashboardPage.jsx	Main dashboard SPA (picks + search + upload)	<ul style="list-style-type: none"> • Wrapped in AppLayout so header, nav, earnings banner, and warning banner are always visible. • Hosts PlayerAnalysisSearch, ScreenshotUploader, current-picks panel, DailyPicks, active-bet section, BetSlip, BetConfirmation, and modal stack (PlayerStatsModal, EditBetModal). • Loads user profile, active bets, bet history, and legacy picks from Firestore; moves completed bets to history on mount. • All navigation is handled via useNavigate(); page is mounted at /dashboard and is the post-login landing route.
ProcessedPlayersPage.jsx	Dedicated gallery of server-processed players	<ul style="list-style-type: none"> • Uses AppLayout. • Renders ProcessedPlayers component and passes onAddToPicks to let users queue cards directly into their pick list. • Loads any legacy picks for de-duplication, enforces 6-pick max, and persists additions with addUserPick. • Route: /processed-players.
PreviousBetsPage.jsx	Bet history & active wagers center	<ul style="list-style-type: none"> • Uses AppLayout. • Displays ActiveBet list (editable / cancellable) followed by PreviousBets accordion built from Firestore bet history. • On mount, moves completed bets from <i>active</i> → <i>history</i> and refreshes both lists; provides EditBetModal, PlayerStatsModal for deeper actions. • Route: /previous-bets.

<code>AlertsPage.jsx</code>	Notifications / alerts hub	<ul style="list-style-type: none"> • Uses <code>AppLayout</code>. • Thin wrapper that mounts <code>Notifications</code>; future iterations will hydrate from user-specific subscriptions. • Route: <code>/alerts</code>.
<code>SignIn.jsx</code>	Simple username + password login screen	<ul style="list-style-type: none"> • Stylized form with show-password toggle. • On submit, verifies credentials via <code>getUserByUsername</code>, runs bootstrap (<code>initializeUser, initializeDatabase</code>) if needed, stores <code>currentUser</code> in <code>sessionStorage</code>, then redirects to <code>/dashboard</code> with <code>useNavigate()</code>. • Still seeds the demo user on first mount.
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"> • Sets the global shell: <code>#root</code> max-width 1280 px, centered with zero padding, left-aligned text. • Keeps Vite starter styles: logo hover/ spin animation, <code>.card</code> padding, <code>.read-the-docs</code> gray text. • Removes the previously bloated mobile overrides—Tailwind utility classes in the new components now handle responsiveness. • Adds a slim mobile-only block (<code>max-width: 768px</code>) that just enforces 44 px minimum touch targets and <code>font-size: 16px</code> on form controls to prevent iOS zoom.
<code>App.jsx</code>	Top-level React router	<ul style="list-style-type: none"> • Uses React Router v6 to declare the full routing map: <code>/ → SignIn, /dashboard → DashboardPage, /processed-players → ProcessedPlayersPage, /previous-bets → PreviousBetsPage, /alerts → AlertsPage</code>. • Includes a legacy redirect by pointing <code>/HomePage</code> to <code>DashboardPage</code>, preserving old bookmarks. • Stateless functional wrapper; exported as default so <code>main.jsx</code> can mount it at the root.
<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	<ul style="list-style-type: none"> • 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	<ul style="list-style-type: none"> 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
PredictionCard.jsx	Mini card that surfaces the model's score prediction for <i>one</i> player	<ul style="list-style-type: none"> 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").
PreviousBets.jsx	Accordion list of a user's completed and in-flight wagers	<ul style="list-style-type: none"> 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.
processed-players.css	Hover & animation helpers for <i>ProcessedPlayers</i> grid	<ul style="list-style-type: none"> 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.
ProcessedPlayers.jsx	Searchable / filterable gallery of players already analyzed server-side	<ul style="list-style-type: none"> Fetches docs via <code>getProcessedPlayers()</code> then lets users search, filter by team, and filter by AI recommendation from three drop-downs. Confirmation  flashes for 3 s when <i>Add to Picks</i> fires; uses internal <code>addedPlayers</code> map to prevent dupes. Clicking a card opens <code>PlayerAnalysisModal</code>.
RecommendationCard.jsx	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).

ScreenshotUploader.jsx

Drag-and-drop widget that parses PrizePicks images, then chains player analysis

- Supports multi-file drag-and-drop and click-to-browse; shows image previews with type & size badges and per-file remove/X.
- Simulates progress to 95 %, calls `/api/parse_screenshot`, then sequentially POSTs each parsed `{player, threshold}` to `/api/player` while updating row status chips (spinner →  .
- Handles error banners, success banners, *Clear All*, and clears previews after processing.

StatsCard.jsx

Detailed stat panel inside the analysis modal

- 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.

thinking-animation.css

Re-usable pulse animation for ChatGPT “thinking” loader

- `.thinking-dot` staggered dot pulse and `.thinking-ring` breathing ring key-frames.

ThinkingAnimation.jsx

Centered loader component shown while awaiting AI response

- Uses the above CSS to render a gradient ring, three pulsing dots, and explanatory blurb (“Gathering player statistics...”).
- Accepts optional `text` prop (default “Analyzing”).

TrendingPicks.jsx

Static demo card of “most popular picks”

- Currently hard-codes an array of three players with popularity %, threshold and recommendation; renders with icons & team info.

FavoritePlayers.jsx

Empty-state panel for future “starred” players

- 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 photo, logos, matchup info, AI recommendation chip, threshold, Poisson and Monte-Carlo %s.
- Key-stats tiles (season avg, last 5, vs-opponent, home/away), volatility tiles, playoff tiles.
- Expandable sections for all-season encounters, recent games, playoff log; “Load more games” fetches via `/api/player/{id}/more_games`.
- Formats numbers/percents, color-codes vs threshold, and exposes Add to Picks.

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"> • Autocomplete-style form with recent-search dropdown (<code>localStorage</code>), search-tips panel, and live validation. • Saves top 5 searches, includes info & history icons, and shows error/loader states under the form. • Emits <code>onSearch(player, threshold)</code>; parent supplies <code>loading/error</code> props.
<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 ChatGptThinking 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>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>AppLayout.jsx</code>	Shared top-level layout wrapping all main pages	<ul style="list-style-type: none"> • Desktop header with nav links; mobile header + slide-in menu using Lucide icons and stateful <code>mobileMenuOpen</code>. • Highlights active route via <code>useLocation()</code>, displays warning banner (“Play at your own risk”) and earnings banner. • Fetches user profile (avatar, display name, earnings) on mount; handles sign-out and auth guard redirection.
<code>MobileLayout.jsx</code>	Responsive shell that swaps between a desktop sidebar and a collapsible mobile drawer	<ul style="list-style-type: none"> • Navigation buttons now call <code>navigate()</code> (React Router) instead of local state; highlights current route. • Sticky mobile header with hamburger / close icon toggles drawer; desktop sidebar fixed 64 px wide. • Wraps <code>{children}</code> so any page can inherit the layout.

MobileOptimizedDashboard.jsx

All-in-one mobile dashboard workflow (pick list → search → stats)

- Shows Your Picks panel with lock-in, remove and live count (`picks.length / 6`).
- Responsive analysis form (player + threshold) and analyze button.
- Renders `PlayerCard`, `StatsCard`, `RecommendationCard`, 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

- Header shows pulsing alert icon; Edit and Cancel buttons invoke prop callbacks.
- Clicking header toggles expanded view showing metadata & pick list; pick clicks surface deeper player info.
- Status color: Final = green, Live = yellow, default = gray.

frontEnd/src/scripts/

File	Purpose in one glance	Key responsibilities / notable elements
<code>mobile-viewport.js</code>	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.
<code>use-mobile-detector.js</code>	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.
<code>initDatabase.js</code>	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

- Reads the old flat user document, rolls username/password/email into a nested `profile` object, and writes it back via `updateDoc`.
- Splits historic `bets` array into `activeBets` and month-bucketed `betHistory` 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	<p>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.</p>

`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 <code>Reason</code> column. Exposes two helpers: <ul style="list-style-type: none"> <code>get_full_injury_report()</code> → <code>list</code> \subset {gameDate, gameTime, team, player, status, reason} for every entry. <code>get_player_status(name)</code> → quick lookup returning {status, reason} (or "NOT YET SUBMITTED" if a team hasn't filed).

`main.py`

Cloud-Function handler that keeps Firestore documents in sync with real-time game status

- Utility `fetch_game_status()` hits `nba_api`'s `ScoreboardV2` & `BoxScoreTraditionalV2` to see whether a game has started, is live, or has finished—and, if finished, grabs the player's final points.
- `check_active_players()` walks the `processedPlayers/players/active` collection; if `fetch_game_status()` returns new info, it calls `update_doc()` to patch the Firestore doc (e.g., set `"gameStatus": "Concluded"` and `"finalPoints": n`).
- Stubs for `check_user_picks()` and `check_active_bets()` illustrate the same pattern for user bet docs.
- `check_games_handler(request)` is the HTTP entry-point wired to Cloud Scheduler (runs every hour); it invokes `check_active_players()` and responds 200 OK or 500 on error.