

PrizePicks Prediction Website

Lead Developer | Feb 2025 - Present

A full-stack, AI-powered platform that automatically analyzes NBA "Over/Under" picks on PrizePicks. From OCR'ing screenshots to running Poisson, Monte Carlo & GARCH volatility forecasts (including playoff games), and generating natural-language bet explanations via ChatGPT, this site manages the entire pipeline end-to-end-hosted on Firebase Hosting + Cloud Run with CI/CD.

🚀 Project Overview

- ****Objective:**** Predict NBA player point performances ("Over/Under" picks) using statistical models (Poisson, Monte Carlo, GARCH volatility) and AI-driven explanations.
- ****Live Outcome:**** Turned \$10 into \$3,279+ on PrizePicks (29,900% ROI) with an 11/14 lineup win rate.
- ****Core Features:****
 - ****Screenshot Parsing (OCR):**** Upload PrizePicks cards, extract player & threshold pairs.
 - ****Player Pipeline:****
 - Season & last-5 game averages
 - Poisson probability
 - Monte Carlo simulation
 - GARCH volatility forecast (regular season & playoffs)
 - Injury report scraping
 - ChatGPT-powered bet explanation
 - ****Playoff Support:**** Automatically switches to playoff stats after ≥ 5 postseason games.
 - ****Real-Time Updates:**** Background Cloud Functions mark "Concluded" games and settle bets.
 - ****CI/CD & Hosting:**** React + Vite on Firebase Hosting, Flask + Docker on Cloud Run, GitHub Actions auto-deploy.

📸 Pre Flight Website Access

[Website Link] (<https://prizepicksproject-15337.web.app/>)

- ****Currently, the project is still in development as more features will be integrated along with bug fixes****

- If you would like access to the website despite it's early development phase, please feel free to reach out to bryanram2024@gmail.com

🎥 Demo Video

[Watch on
GitHub] (<https://github.com/user-attachments/assets/ec796b28-824e-4374-8d9a-beedc7a0ed4e>)

🖼️ Screenshots

Home Page

![] (<https://github.com/user-attachments/assets/39f4e1e9-add3-415b-95ca-03cb9c5b3129>)
Greeted by Earnings, Active Bets & Live Picks.

Player Analysis Panel

![] (<https://github.com/user-attachments/assets/8d960312-30c7-47f6-9004-ed82facc348b>)
Input a player + threshold → see probability forecasts & AI explanation.

Processed Players Dashboard

![] (<https://github.com/user-attachments/assets/3f9c727b-b315-4688-bd57-0a12a55820dc>)
Aggregated player cards across all users.

📦 Tech Stack

Front-End

- ****React + Vite**** - SPA framework
- ****Tailwind CSS**** - Utility-first styling
- ****Lucide React**** - Icon library
- ****Recharts**** - Charts & graphs

Back-End

- ****Python 3.9+****
- ****OCaml**** - Monte Carlo

- **Flask** - REST API
- **gunicorn** - WSGI server (Cloud Run)
- **firebase-admin** - Firestore & Auth
- **openai** - ChatGPT o4-mini integration

📈 Data & Analytics

- **Poisson & Monte Carlo** - Probability pipelines
- **GARCH (arch-model)** - Volatility forecasting
- **pandas, NumPy** - Data wrangling
- **NBA API** - Stats & box scores
- **OCR (screenshot_parser.py)** - Image data extraction
- **Requests** - Web scraping (NBA Injury Report)
- **!!!Coming Soon!!!** - ML Algorithm trained off of data stored in Firestore

🏗️ Infrastructure & Deployment

- **Firebase Hosting** - Front-end CDN & SSL
- **Cloud Run** - Containerized Flask API
- **Firebase Cloud Functions** - Background jobs & data migration
- **GitHub Actions** - CI/CD (build → deploy Hosting & Cloud Run)
- **Docker** - Back-end container

📊 Probability & Forecasting Methods

Below is a quick reference on how each analytical value is produced inside the player documents.

📊 Poisson Probability (`poissonProbability`)

- **Data window:** *All* regular-season games from the current season
- **Library:** Native Python `math` (no external deps)
- **Computation:**
 - Calculate the season scoring average λ
 - Evaluate $P(X \geq t) = 1 - \sum_{k=0}^{\lceil t \rceil - 1} \frac{e^{-\lambda} \lambda^k}{k!}$

where **t** is the user-selected points threshold
- **Interpretation:** Purely distribution-based likelihood a player scores **over** the line given their season-long mean

```

### 🎲 Monte Carlo Probability (`monteCarloProbability`)
- Data window: Up to 60 most-recent games (regular and playoff)
- Stats used: sample mean  $\mu$  & standard deviation  $\sigma$ 
- Simulations: 100 000 random seasons per query
- Engine priority:
  1. OCaml routine exposed through a C shared library (`mc_stub.c`) for speed
  2. Fallback to NumPy's np.random.normal() if the native lib isn't available
- Output: Fraction of simulations where the random score  $\geq$  user threshold
- Why Monte Carlo? Captures hot/cold streaks and non-Gaussian tails better than a single closed-form model

```

```

### 📈 GARCH Volatility Forecast (`volatilityForecast`, `volatilityPlayOffsForecast`)
- Data window: Last 50 games (or all playoff games once  $\geq 5$  exist)
- Library: [arch] (https://github.com/bashtage/arch) - fits a GARCH(1,1) model
- Pipeline:
  1. Convert the points series to "returns" via first differences
  2. Fit GARCH(1,1) on those returns
  3. Return the 1-step-ahead forecasted  $\sigma$  (square-root of the predicted variance)
- Interpretation: Forward-looking volatility that reflects clustering of high-variance performances

```

Together, these three metrics give a balanced outlook:

Metric	Scope	Strength
Poisson	Season-long	Fast analytical baseline
Monte Carlo	Last ≤ 60 games	Empirical tail-risk capture
GARCH σ	Last 50 games	Short-run variance / streak detection

Project Scheme

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
├── admin/
│   ├── SystemOverview.jsx
│   ├── UserAnalytics.jsx
│   ├── BetPerformance.jsx
│   ├── PlayerAnalytics.jsx
│   ├── FinancialMetrics.jsx
│   └── SystemMonitoring.jsx
├── pages/
│   ├── SignIn.jsx
│   ├── DashboardPage.jsx
│   ├── ProcessedPlayersPage.jsx
│   ├── PreviousBetsPage.jsx
│   ├── AlertsPage.jsx
│   ├── SignIn.jsx
│   ├── AdminLogin.jsx
│   └── AdminLayout.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

```

Firestore Database Scheme

firestore/

```

├── processedPlayers/ (collection)
│   ├── active/ (document)
│   │   └── {first_last_threshold_YYYYMMDD (e.g. aaron_gordon_11.5_20250511)}/
│   │       (document)
│   │           └── name: string (e.g. Aaron Wiggins)

```

```

└─ playerId: string (e.g. 1630598)
└─ team: string (e.g. Oklahoma City Thunder)
└─ position: string (e.g. G)
└─ opponent: string (e.g. Minnesota Timberwolves)
└─ photoUrl: string
└─ teamLogo: string
└─ opponentLogo: string
└─ gameDate: Timestamp
└─ gameTime: string
└─ gameType: string
└─ teamPlayoffRank: number
└─ opponentPlayoffRank: number
└─ seasonAvgPoints: number
└─ last5RegularGamesAvg: number
└─ seasonAvgVsOpponent: number
└─ homeAwayAvg: number
└─ last5RegularGames: array<map>
  └─ [{ date, points, opponent, opponentFullName, ... }, ...]
└─ advancedPerformance: map
└─ careerSeasonStats: array<map>
└─ injuryReport: map
└─ betExplanation: map
└─ poissonProbability: number
└─ monteCarloProbability: number
└─ volatilityForecast: number
└─ season_games_agst_opp: array<map>
└─ num_playoff_games: number
└─ playoffAvg: number
└─ playoff_games: array<map>
  └─ [{ date, points, opponent, ..., gameType: "Playoffs" }, ...]
└─ volatilityPlayOffsForecast: number
└─ concluded/ (document)
  └─ {first_last_threshold_YYYYMMDD}/
    └─ (same fields as active/)
└─ injury_report/ (document)
  └─ {team_name (e.g. indiana_pacers)}/ (document)
    └─ lastUpdated: timestamp
    └─ players: array<map>
      └─ gameDate: string
      └─ gameTime: string
      └─ reason: string
    └─ team: string

```

```

└─ users/{userId}/ (collection)
|   └─ activeBets/{YYYYMMDDTHHMMSSZ}
|       └─ { betAmount, potentialWinnings, picks: [ [0] player_Document_References,
[i]... ] }
|   └─ betHistory/{YYYYMMDDTHHMMSSZ}
|       └─ { betAmount, potentialWinnings, betResult, picks: [ [0]
player_Document_References (+ points, minutes added), [i]... ] ] }
|   └─ picks: picks: [ [0] player_Document_References, [i]... ]
|       └─ profileData
└─ admin/
    └─ profile/
        └─ username: string
        └─ password: string
    └─ analytics/
        └─ daily_stats/
        └─ user_metrics/
        └─ system_health/
    └─ monitoring/
        └─ api_performance/
        └─ error_logs/
    └─ reports/
        └─ bet_performance/
        └─ player_analytics/

```

More information on every File:

backEnd/

File	Purpose in one glance	Key responsibilities / notes
app.py	Central Flask API gateway + admin analytics hub	<ul style="list-style-type: none"> Boots Flask, enables CORS for localhost + Firebase-hosted domains, and auto-auths with the Cloud Run service account → initializes Firestore. Utility helpers pkey() + thr_doc_ref() standardise document IDs for “player + threshold” records. /api/player (POST) – cache-or-compute pipeline: <ul style="list-style-type: none"> Checks processedPlayers/players/active/{player_threshold}; returns doc if it exists. Otherwise runs full analysis stack: <ul style="list-style-type: none"> player_analyzer season + L5 stats real-time injury status (injury_report) Poisson & Monte Carlo win odds GARCH volatility (reg-season and playoffs if ≥5 PO games) ChatGPT natural-language bet explanation Persists the result back to Firestore (pick_id field added). /api/parse_screenshot (POST) – accepts one or more

		<p>images, OCRs each with <code>screenshot_parser</code>, POSTs every detected player + threshold pair to <code>/api/player</code>, and returns a flat list of what it parsed.</p> <ul style="list-style-type: none"> • <code>/api/player/<id>/more_games</code> (GET) – lazy-loads all extra regular-season games for infinite-scroll tables. • Admin analytics suite now pulls live counts instead of pure mocks: <ul style="list-style-type: none"> – <code>/api/admin/overview</code> totals users, active bets, processed players, winnings, plus a placeholder <code>apiRequests</code> counter. – <code>/api/admin/users</code> returns per-user engagement stats (earnings, win-rate, last login, etc.). – <code>/api/admin/bets</code> aggregates bet volume/performance (still mostly sample data). – <code>/api/admin/players</code> builds league-wide hit-rate & “most-analyzed” tables on the fly. – <code>/api/admin/system</code> surfaces health metrics (CPU, memory, latency, uptime). • <code>check_games</code> cron endpoint wired in via <code>main.check_games_handler</code> for scheduled status sweeps. • Runs <code>app.run()</code> in local debug; Cloud Run uses Gunicorn (<code>app:app</code>).
<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 “≥ 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 <code>libmontecarlo.so</code>.</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>

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
<code>mobile-viewport.html</code>	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 <code>touchstart</code> (multi-touch) and <code>touchend</code> events to <code>preventDefault()</code>—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/

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: <ul style="list-style-type: none"> <code>/</code> → <code>SignIn</code>, <code>/dashboard</code> → <code>DashboardPage</code>, <code>/processed-players</code> → <code>ProcessedPlayersPage</code>, <code>/previous-bets</code> → <code>PreviousBetsPage</code>, <code>/alerts</code> → <code>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 `main.jsx` can mount it at the root.

<code>firebase.js</code>	Front-end Firebase initializer	• 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/pages

File	Purpose in one glance	Key responsibilities / notable elements
<code>DashboardPage.jsx</code>	Main dashboard SPA (picks + search + upload)	• Wrapped in <code>AppLayout</code> (header, nav, earnings & warning banners). • Hosts <code>PlayerAnalysisSearch</code> , <code>ScreenshotUploader</code> , picks panel, <code>DailyPicks</code> , <code>ActiveBet</code> , <code>BetSlip</code> , <code>BetConfirmation</code> , and modal stack (<code>PlayerStatsModal</code> , <code>EditBetModal</code>). • Loads user profile, active bets, history, legacy picks; on mount moves completed bets to history. • Route: <code>/dashboard</code> (post-login landing).
<code>ProcessedPlayersPage.jsx</code>	Dedicated gallery of server-processed players	• Uses <code>AppLayout</code> . • Renders <code>ProcessedPlayers</code> , passes <code>onAddToPicks</code> ; enforces 6-pick cap; de-dupes legacy picks. • Route: <code>/processed-players</code> .
<code>PreviousBetsPage.jsx</code>	Bet history & active wagers center	• Uses <code>AppLayout</code> . • Shows <code>ActiveBet</code> list (editable/cancellable) + <code>PreviousBets</code> accordion from Firestore history. • On mount migrates completed bets; supplies <code>EditBetModal</code> , <code>PlayerStatsModal</code> . • Route: <code>/previous-bets</code> .

<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 subs. • 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. • Verifies credentials via <code>getUserByUsername</code>; seeds new users; stores <code>currentUser</code> in <code>sessionStorage</code>. • Redirects to <code>/dashboard</code> on success.
<code>AdminLogin.jsx</code>	Admin portal login page	<ul style="list-style-type: none"> • Gradient “Admin Portal” screen with Lucide icons; username + password fields with show/hide toggle. • Calls <code>getAdminCredentials</code> & <code>verifyAdminPassword</code> (from <code>firebaseService</code>) to authenticate. • On success sets <code>sessionStorage.isAdmin = true</code> & <code>adminUser</code>, then <code>navigate("/admin/dashboard")</code>. • Shows red error banner on failure & loading spinner while verifying. • Route: <code>/admin</code>.
<code>AdminDashboard.jsx</code>	Auth-protected admin SPA (analytics + monitoring)	<ul style="list-style-type: none"> • Wrapped in <code>AdminLayout</code> (admin header/sidebar). • On mount checks <code>sessionStorage.isAdmin</code>; if missing, redirects to <code>/admin</code>. • Tab navigation: System Overview, User Analytics, Bet Performance, Player Analytics, Financial Metrics, System Monitoring. • Dynamically renders the active tab component; polls Firestore via service helpers. • Route: <code>/admin/dashboard</code>.

`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 one player	<ul style="list-style-type: none"> Shows threshold, computed probability and Poisson probability. Colors the Recommendation 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 (updated)	Hover, touch & animation helpers for ProcessedPlayers grid	<ul style="list-style-type: none"> Scales card to 1.02 × on hover, 0.98 × on touch-press; adds deeper shadow. Re-usable <code>.pulse</code> key-frame (green glow) for "Added to Picks"-style confirmations. Mobile tweaks: bigger touch targets (<code>min-height: 400 px</code>), tighter grid gaps, larger font, and button sizing. Utility spin & fade-in key-frames used by loading spinners / card entrance.
ProcessedPlayers.jsx (updated)	Searchable / filterable gallery of players already analyzed server-side	<ul style="list-style-type: none"> Fetches docs via <code>getProcessedPlayers()</code> → builds team and AI-recommendation drop-downs (100 % YES / 90–100 % YES / 80–90 % possible). Mobile-first card redesign with gradient headers, team/opponent logos, and threshold/probability panels. "Add to Picks" confirmation <input checked="" type="checkbox"/> now pulses for 3 s; per-player added state stored in <code>addedPlayers</code> map. Handles loading spinner, graceful error banner, empty-state, and responsive grid (1 × → 3 ×). 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 (<input type="checkbox"/> green / <input type="checkbox"/> red).

ScreenshotUploader.jsx	Drag-and-drop widget that parses PrizePicks images, then chains player analysis	<ul style="list-style-type: none"> • 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 <code>/api/parse_screenshot</code>, then sequentially POSTs each parsed {player, threshold} to <code>/api/player</code> 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	<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” opens paginated modal of up to 15 games.
thinking-animation.css	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.
ThinkingAnimation.jsx	Centered loader component 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...”).
TrendingPicks.jsx	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.
FavoritePlayers.jsx	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.
ImageWithFallback.jsx	Re-usable <code></code> wrapper that never breaks	<ul style="list-style-type: none"> • Attempts primary <code>src</code>; on <code>onError</code> swaps to <code>fallbackSrc</code> (or <code>/placeholder.svg</code>) so broken images don’t wreck the layout.

InjuryStatusCard.jsx	Rich card that visualizes a player's latest injury report	<ul style="list-style-type: none"> • 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	<ul style="list-style-type: none"> • 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	<ul style="list-style-type: none"> • 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	<ul style="list-style-type: none"> • Hero banner with photo, logos, matchup info, AI recommendation chip, threshold, Poisson & Monte-Carlo % s. • Key-stats tiles, volatility tiles, playoff tiles. • Expandable sections for all-season encounters, recent games, playoff log; “Load more games” fetches via <code>/api/player/{id}/more_games</code>.
PlayerAnalysisModal.jsx (updated)	Full-screen modal deep-dive (mobile-first)	<ul style="list-style-type: none"> • Gradient hero with photo, dual-logo overlay, threshold & AI recommendation chip. • Key-stats grid, volatility tiles (regular & playoffs), and mobile-optimized expandable sections (all-season encounters, more-games fetch). • Displays Poisson & Monte-Carlo % s, advanced metrics, injury report snippet, plus free-text AI bet explanation. • “Add to Picks” button closes modal and returns enriched pick object with deterministic <code>id</code>.
PlayerAnalysisSearch.jsx	Smart search bar that drives the analysis flow	<ul style="list-style-type: none"> • Autocomplete form with recent-search drop-down (localStorage), search-tips panel, and live validation. • Saves top 5 searches; emits <code>onSearch(player, threshold)</code>; parent supplies loading / error props.

PlayerCard.jsx	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.
PlayerStatsModal.jsx	Lightweight modal for viewing bet details from Previous Bets	<ul style="list-style-type: none"> Hydrates from Firestore when possible for richer stats; falls back to passed prop. Presents threshold, recommendation, timings, season / last-5 / vs-opponent averages and actual result.
AdvancedMetricsCard.jsx	Shows eFG %, 3-pt 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.
ApiTest.jsx	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.
BetConfirmation.jsx	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 ✖.
BetExplanationCard.jsx	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.
BetSlip.jsx	Full-screen wizard to assemble & confirm a wager	<ul style="list-style-type: none"> Lets user pick platform, bet type, amount, choose picks, and computes potential winnings; fires <code>onConfirm</code> with formatted picks.
chatgpt-thinking.css	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.
ChatGptThinking.jsx	Loader component that uses the above CSS	<ul style="list-style-type: none"> Shows ChatGPT logo + three pulsing dots and optional status text.
DailyPicks.jsx	“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).

EditBetModal.jsx	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.
AppLayout.jsx	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 <code>mobileMenuOpen</code> state. • Highlights active route via <code>useLocation()</code>, displays “Play at your own risk” banner + cumulative-earnings banner. • Fetches user profile (avatar, display name, earnings) on mount; handles sign-out and auth-guard redirection.
MobileLayout.jsx (updated)	Responsive shell that swaps between a desktop sidebar and a collapsible mobile drawer	<ul style="list-style-type: none"> • Navigation buttons now use React Router’s <code>navigate()</code> instead of local state. • 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—and delegates global banners & auth guard to AppLayout.
MobileOptimizedDashboard.jsx	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 and “Analyze” button. • Renders PlayerCard, StatsCard, RecommendationCard, last-5 games table, and Add to Picks CTA after a search.
MobilePlayerCard.jsx	Compact player-info tile for small screens	<ul style="list-style-type: none"> • 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	<ul style="list-style-type: none"> • 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/components/admin/

File	Purpose in one glance	Key responsibilities / notable UI behavior
<code>SystemOverview.jsx</code>	Snapshot of overall platform health	<ul style="list-style-type: none">Fetches system KPIs via <code>getSystemOverview()</code> every 30 s.Stats grid (users, bets, processed players, winnings, uptime, API, error-rate, response time).Recent activity feed & quick-action buttons (manage users, DB backup, alerts).
<code>UserAnalytics.jsx</code>	Insights into user engagement & behavior	<ul style="list-style-type: none">Time-range selector (24h, 7d, 30d, 90d) → <code>getUserAnalytics()</code>.Metrics tiles (active users, avg session, top performer, new sign-ups).Recent-activity table & engagement bar chart.
<code>BetPerformance.jsx</code>	Platform-wide bet outcome analytics	<ul style="list-style-type: none">Time-range selector; fetches via <code>getBetPerformance()</code>.Metrics tiles (total bets, win-rate, winnings, ROI), win/loss distro bars, most-profitable picks list, 30-day performance trend chart.
<code>PlayerAnalytics.jsx</code>	Aggregate view of player-level stats	<ul style="list-style-type: none">Fetches via <code>getPlayerAnalytics()</code>; sort dropdown (hit-rate, analyzed, profit, popularity).Metrics tiles, sortable ranking table, threshold distribution & team-frequency charts.
<code>FinancialMetrics.jsx</code>	Revenue and financial health dashboard	<ul style="list-style-type: none">Time-range selector; fetches via <code>getFinancialMetrics()</code>.Metrics tiles (revenue, user winnings, platform ROI, avg bet size), revenue breakdown bars, top-earning users list, revenue trend chart.
<code>SystemMonitoring.jsx</code>	Live operational monitoring panel	<ul style="list-style-type: none">Polls <code>getSystemHealth()</code> every 10 s.Metrics grid (API response, DB perf, CPU/memory, latency, error rate) colored by status.System alerts feed, 24 h charts for response time & error rate, service-status grid with uptime dots.

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.
<code>migrateData.js</code>	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.

firebaseService.js

Firestore
data-access
layer

- Unified document-reference architecture
- Helper creators / resolvers (`createPlayerDocumentReference`, `resolveDocumentReference(s)`) let the front-end store just *Firestore document refs* in user picks & bets instead of full JSON objects.
- Migration helpers (`migrateUserPicksToReferences`, `migrateActiveBetsToReferences`, `migrateBetHistoryToReferences`, `migrateUserToReferences`) auto-convert legacy arrays to the new reference model.
- Full CRUD for picks & bets
- `addUserPick`, `removeUserPick`, `getUserPicks` now read/write arrays of document references and immediately resolve them back to full objects for UI use.
- New sub-collection workflows for `activeBets` & `betHistory`, plus helpers to cancel/update bets while keeping a legacy fallback for the old flat `bets[]` array.
- User bootstrap & profile upgrades
- `initializeUser` and `initializeDatabase` now seed users in the new profile-object format but can also migrate existing flat-field users in place.
- `updateUserStats` increments nested `profile.*` counters when present, otherwise patches legacy fields.
- Admin analytics suite (powers the React *admin* dashboard)
- Credential helpers: `getAdminCredentials`, `verifyAdminPassword`.
- Data aggregators: `getSystemOverview`, `getUserAnalytics`, `getBetPerformance`, `getPlayerAnalytics`, `getFinancialMetrics`, `getSystemHealth` – each queries Firestore (or returns mock placeholders) and shapes the response expected by the new admin components.
- Overall, the service now bridges three generations of data shapes (legacy full objects → reference arrays → sub-collections) while exposing a clean, Promise-based API for both user and admin UIs.

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.