```markdown
# 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/)
```

```markdown
- **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 $$P(X \ge t) \;=\; 1 - \sum_{k=0}^{\lceil t\rceil-1} \frac{e^{-\lambda}\lambda^{k}}{k!}$$
   where **`t`** is the user‑selected points threshold
- **Interpretation:** Purely distribution‑based likelihood a player scores **over** the line given their season‑long mean

---
```

```markdown
### 🎲 Monte Carlo Probability (`monteCarloProbability`)
- **Data window:** Up to **60** most-recent games (regular *and* playoff)
- **Stats used:** sample mean `µ` & standard deviation `σ`
- **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 ≥ 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 ≥ 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 **σ** (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`	Flask API gateway	• Boots the Flask server, loads a service-account JSON, and initializes the Firebase Admin SDK (Firestore client).• **Public health check** at `"/"` simply returns `"API is running"` so Cloud Run + uptime checks can verify liveness.• Keeps existing user-facing endpoints (e.g., `POST /api/parse_screenshot`, `POST /api/player`) **unchanged**.• **New admin analytics suite**: - `GET /api/admin/overview` – counts total users, active bets, processed players, total winnings, mock API-hits. - `GET /api/admin/users` – aggregates engagement stats (active users, new sign-ups, avg. session time). - `GET /api/admin/bets` – returns bet totals, win-rate, ROI, 30-day trend. - `GET /api/admin/players` – league-wide hit-rate, most-analyzed players, profit leaders.

		`GET /api/admin/system` – live ops metrics (API latency, DB performance, CPU %, memory, service status). • Each admin route wraps Firestore queries in a `try/except`, returning JSON `{…, "status": "success"}` or an `{"error": …}` message with proper HTTP status codes.• Runs in local debug mode when executed directly (`python app.py`), but Cloud Run deploy uses Gunicorn (`app:app`).
`backtester.py`	Historical profit-and-loss simulator	• Scans prior `processedPlayers/*` docs, applies a simple bet-settlement rule, and builds a P&L Series.• Useful for validating the model or generating performance charts in notebooks.
`chatgpt_bet_explainer.py`	Natural-language "Why this bet?" generator	• 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.
`injury_report.py`	Live injury-status scraper	• 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 `player_analyzer.py` to adjust probabilities.
`main.py`	Cloud-Run entry-point + cron helpers	• Exposes `app` for Gunicorn **and** contains scheduled logic that: – polls recent box scores; – moves finished picks from *active* → *concluded*; – updates user bet history.
`monte_carlo.py`	Python wrapper around native Monte-Carlo engine	• Fetches ≤ 60 recent games → computes μ, σ → runs **100 000** sims.• Prefers the ultra-fast shared lib `libmontecarlo.so` (see below) but can fall back to NumPy.
`player_analyzer.py`	Master data wrangler & feature builder	• Queries `nba_api` for season stats, last-5 logs, playoff data, opponent strength, etc.• Calls `injury_report`, `volatility.forecast_volatility`, `prediction_analyzer.poisson_over_prob`, and `monte_carlo.monte_carlo_probability`.• Bundles everything into a dict that the front-end cards expect.

`prediction_analyzer.py`	Math helpers (Poisson & misc.)	• Implements closed-form Poisson “≥ threshold” calculation.• Provides thin wrappers invoked by `player_analyzer` and feeds into `chatgpt_bet_explainer`.
`volatility.py`	GARCH(1,1) volatility forecaster	• Builds a 50-game (or playoff-only) series of point “returns”, fits `arch_model`, and returns 1-step-ahead σ.• Output stored as `volatilityForecast` / `volatilityPlayOffsForecast`.
`screenshot_parser.py`	OCR extractor	• Accepts base-64 images, calls OpenAI Vision, parses **player / threshold** pairs, and returns them to `app.py`.
`requirements.txt`	Python dependency list	• Flask, `nba_api`, `arch`, `firebase-admin`, `openai`, etc.—installed in Stage 2 of the Docker build.
`mc_stub.c` & **`montecarlo.ml`**	Native speed layer for Monte-Carlo	• `montecarlo.ml` → OCaml routine that performs the random draws.• `mc_stub.c` bridges Python ↔ OCaml via `ctypes`, producing **`libmontecarlo.so`** during the Docker build.
`Dockerfile`	Two-stage container build	**Stage 1 (OCaml):** 1. Starts from `ocaml/opam`, installs OCaml + `ctypes`. 2. Compiles `montecarlo.ml` into a PIC object, compiles `mc_stub.c`, links both into **`libmontecarlo.so`**. **Stage 2 (Python runtime):** 1. `python:3.9-slim`, installs `libffi` and Python deps from `requirements.txt`. 2. Copies the compiled `.so` and all back-end source files. 3. Launches Gunicorn (`CMD gunicorn app:app --bind 0.0.0.0:${PORT:-8080}` …).

frontEnd/

File	Purpose in one glance	Key responsibilities / notable details

`tailwind.config.js`	**Design-system config for Tailwind CSS**	**• Specifies content scan globs (`index.html`, all files under `src/`) so unused classes are purged from production builds.• Extends the default theme with custom breakpoints (`xs` 475 px to `2xl` 1536 px), extra spacing steps (18, 88, 128), a granular font-scale (`xs` → `4xl`), and utilities like `minHeight.touch` (44 px iOS tap target) and `maxWidth.mobile` (100 vw).• Adds reusable animations & keyframes – slide-in/out, fade-in/out, spin, pulse – referenced by class names such as `animate-slide-in-right`.• 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	• Declares a restrictive `<meta name="viewport" …>` 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/

File	Purpose in one glance	Key responsibilities / notable details
`App.css`	**Global styling + mobile-first overrides**	**• Sets the global shell: `#root` max-width 1280 px, centered with zero padding, left-aligned text.• Keeps Vite starter styles: logo hover/ spin animation, `.card` padding, `.read-the-docs` gray text.• Removes the previously bloated mobile overrides—Tailwind utility classes in the new components now handle responsiveness.• Adds a slim mobile-only block (`max-width: 768px`) that just enforces 44 px minimum touch targets and `font-size: 16px` on form controls to prevent iOS zoom.**
`App.jsx`	**Top-level React router**	**• Uses React Router v6 to declare the full routing map: `/` → `SignIn`, `/dashboard` → `DashboardPage`, `/processed-players` → `ProcessedPlayersPage`, `/previous-bets` → `PreviousBetsPage`, `/alerts` → `AlertsPage`.• Includes a legacy redirect by pointing `/HomePage` to `DashboardPage`, preserving old bookmarks.• Stateless functional wrapper; exported as default so `main.jsx` can mount it at the root.**

`firebase.js`	Front-end Firebase initializer	• Reads API keys & IDs from Vite env variables and calls `initializeApp()`.• Exports Firestore instance `db` and, in browser environments only, `analytics`—guarded so SSR or Node tests don't break.
`index.css`	Tailwind layer injection	• Simply imports `@tailwind base`, `components`, and `utilities`; actual custom styles live in individual component `.css` files.
`main.jsx`	React entry point rendered by Vite	• Boots the app via `createRoot().render(<StrictMode><App /></StrictMode>)`.• Pulls in `index.css` so Tailwind styles apply globally before any component mounts.

frontEnd/src/pages

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

`AlertsPage.jsx`	Notifications / alerts hub	• Uses `AppLayout`.• Thin wrapper that mounts `Notifications`; future iterations will hydrate from user-specific subs.• Route: `/alerts`.
`SignIn.jsx`	Simple username + password login screen	• Stylized form with show-password toggle.• Verifies credentials via `getUserByUsername`; seeds new users; stores `currentUser` in `sessionStorage`.• Redirects to `/dashboard` on success.
`AdminLogin.jsx`	Admin portal login page	• Gradient "Admin Portal" screen with Lucide icons; username + password fields with show/hide toggle.• Calls `getAdminCredentials` & `verifyAdminPassword` (from `firebaseService`) to authenticate.• On success sets `sessionStorage.isAdmin = true` & `adminUser`, then `navigate("/admin/dashboard")`.• Shows red error banner on failure & loading spinner while verifying.• Route: `/admin`.
`AdminDashboard.jsx`	Auth-protected admin SPA (analytics + monitoring)	• Wrapped in `AdminLayout` (admin header/sidebar).• On mount checks `sessionStorage.isAdmin`; if missing, redirects to `/admin`.• 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: `/admin/dashboard`.

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	• Shows threshold, computed probability and Poisson probability.• Colors the Recommendation banner green / yellow / red based on `prediction.category` ("Almost guaranteed", "Neutral", "Riskey").
PreviousBets.jsx	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 `expandedBets` state tracks which items are unfolded.
processed-players.css *(updated)*	Hover, touch & animation helpers for ProcessedPlayers grid	• Scales card to 1.02 × on hover, 0.98 × on touch-press; adds deeper shadow.• Re-usable `.pulse` key-frame (green glow) for "Added to Picks"-style confirmations.• Mobile tweaks: bigger touch targets (`min-height: 400 px`), 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	• Fetches docs via `getProcessedPlayers()` → 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 ✅ now pulses for 3 s; per-player added state stored in `addedPlayers` map.• Handles loading spinner, graceful error banner, empty-state, and responsive grid (1 × → 3 ×).• Clicking a card opens PlayerAnalysisModal.
RecommendationCard.jsx	Lightweight "quick verdict" box used inside the analysis modal	• 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” opens paginated modal of up to 15 games.
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 while awaiting AI response	• 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”	• 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.
ImageWithFallback.jsx	Re-usable `<img>` 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 & 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 `/api/player/{id}/more_games`.
PlayerAnalysisModal.jsx *(updated)*	Full-screen modal deep-dive (mobile-first)	• 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 `id`.
PlayerAnalysisSearch.jsx	Smart search bar that drives the analysis flow	• Autocomplete form with recent-search drop-down (localStorage), search-tips panel, and live validation.• Saves top 5 searches; emits `onSearch(player, threshold)`; parent supplies loading / error props.

PlayerCard.jsx	Simple summary card for list views	• 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	• 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	• Renders advanced-metrics grid plus career-season table.• Includes an info call-out explaining each stat.
ApiTest.jsx	Connectivity checker for the Flask API	• Calls `testAPI()` 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	• 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?”)	• 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	• Lets user pick platform, bet type, amount, choose picks, and computes potential winnings; fires `onConfirm` with formatted picks.
chatgpt-thinking.css	Dot-pulse & logo-glow animation for loaders	• Defines `.chatgpt-thinking-dot` key-frames and `.logo-pulse` halo.
ChatGptThinking.jsx	Loader component that uses the above CSS	• Shows ChatGPT logo + three pulsing dots and optional status text.
DailyPicks.jsx	“Today’s picks” & performance-tracker panel	• 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	• 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	• Desktop header with nav links; mobile header + slide‑in menu using Lucide icons and `mobileMenuOpen` state.• Highlights active route via `useLocation()`, 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	• Navigation buttons now use React Router’s `navigate()` instead of local state.• Sticky mobile header with hamburger/close icon toggles drawer; desktop sidebar fixed 64 px wide.• Wraps `{children}` 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)	• Shows Your Picks panel with lock‑in, remove and live count (picks.length / 6).• 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	• 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/components/admin/

File	Purpose in one glance	Key responsibilities / notable UI behavior
`SystemOverview.jsx`	Snapshot of overall platform health	• Fetches system KPIs via `getSystemOverview()` 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).
`UserAnalytics.jsx`	Insights into user engagement & behavior	• Time-range selector (24h, 7d, 30d, 90d) → `getUserAnalytics()`.• Metrics tiles (active users, avg session, top performer, new sign-ups).• Recent-activity table & engagement bar chart.
`BetPerformance.jsx`	Platform-wide bet outcome analytics	• Time-range selector; fetches via `getBetPerformance()`.• Metrics tiles (total bets, win-rate, winnings, ROI), win/loss distro bars, most-profitable picks list, 30-day performance trend chart.
`PlayerAnalytics.jsx`	Aggregate view of player-level stats	• Fetches via `getPlayerAnalytics()`; sort dropdown (hit-rate, analyzed, profit, popularity).• Metrics tiles, sortable ranking table, threshold distribution & team-frequency charts.
`FinancialMetrics.jsx`	Revenue and financial health dashboard	• Time-range selector; fetches via `getFinancialMetrics()`.• Metrics tiles (revenue, user winnings, platform ROI, avg bet size), revenue breakdown bars, top-earning users list, revenue trend chart.
`SystemMonitoring.jsx`	Live operational monitoring panel	• Polls `getSystemHealth()` 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

`mobile-viewport.js`	Forces proper mobile scaling when the main app is loaded from `/public/mobile-viewport.html`	• Immediately-invoked function checks whether a `<meta name="viewport">` already exists; if not, it injects one with `initial-scale=1`, `maximum-scale=1`, and `user-scalable=no` to lock the zoom level.
`use-mobile-detector.js`	Lightweight React hook to tell components "are we on a phone?"	• Keeps an `isMobile` state that is `true` when `window.innerWidth < 768 px`.• Listens for `resize` events so the flag updates dynamically if the user rotates or resizes the window.
`initDatabase.js`	One-time Firestore seeder for local/dev demos	• When executed, creates a demo user `bryanram` (with a basic profile), admin site-wide stats, and an admin user list document if they don't already exist.• Uses `serverTimestamp()` 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
`api.js`	Thin wrapper around your *Flask* back-end	• Single exported helper `analyzePlayer(playerName, threshold)` that POSTs to `/api/player`. • Cleans / type-casts the threshold, maps legacy field names (`nba_player_id → playerId`, 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
`onPlayerStatusChange`	A document under `processedPlayers/players/active/{docId}` is updated	• If the field `gameStatus` flips to `"Concluded"` the function moves that document into `processedPlayers/players/concluded/{docId}` and deletes it from *active*—keeping the two sub-collections mutually exclusive.
`onActiveBetWrite`	A user's bet is written at `users/{userId}/activeBets/{betId}` (create / update / delete)	• When the bet is deleted or its `status` transitions to a terminal state (`Concluded`, `Completed`, `Won`, `Lost`) the pick is copied into `users/{userId}/betHistory/{betId}` with a `settledAt` server timestamp, then removed from `activeBets`.

`onUserPicksUpdate`	The top-level user doc `users/{userId}` is updated	• Compares the previous `picks` array to the new one; if any picks now have `gameStatus: "Concluded"` 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
`full_injury_report.py`	*Deep* PDF scraper that converts the NBA's official daily-injury PDF into clean JSON	• 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 `pdfplumber`, 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 *Reason* column. • Exposes two helpers: • `get_full_injury_report()` → list ⊂ {gameDate, gameTime, team, player, status, reason} for every entry. • `get_player_status(name)` → 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.