ChatGPT

PRD: AI-Enhanced Trading Signals Mobile App (FlutterFlow)

Overview and Objectives

This PRD outlines a FlutterFlow-built mobile application for **AI-enhanced trading signals**. The app delivers curated stock/crypto/option trading signals with real-time data, while an integrated AI assistant provides on-demand analysis via chat. Key objectives include a **freemium model** (limited teaser signals for free users, full access for premium subscribers) and a **token-based AI chat** feature that monetizes advanced insights. The design and UX draw inspiration from successful trading apps like *Option Signal – Flow Alert* (for signal feed and alert patterns) and utilize modern UI kits (Figma community mobile UI kit, FlutterFlow Premium UI components) for a polished, intuitive interface. The app's signal detail page is central – initially a static "signal card" (based on the provided HTML v13 design) that evolves into a **dynamic, interactive screen** when live data streaming and AI chat are enabled. This document defines each screen (Login, Onboarding, Dashboard, Signal List, Signal Detail + AI, Token Purchase, Profile) with UI components, data binding, and logic, ensuring coherence with the freemium + subscription + token economy strategy established in previous research.

Product Scope & Key Features:
- **Freemium Signals Feed:** Free users receive a limited number of teaser signals (e.g. a few per week) with partial information revealed [1] [2]. Paid subscribers get full, real-time signal access (e.g. several signals per day) [3]. Each signal includes entry/exit targets, stop-loss, and rationale (curated by backend analysts or algorithms). Free signals are deliberately partial/delayed to encourage upgrade – for example, showing that "ABC stock is up 5%" but hiding the full trade details behind a premium prompt.
- **Premium Subscription & Monetization:** The app offers monthly/annual subscriptions for premium access. Premium users unlock all signals (both stock and option alerts) and get real-time push notifications [4]. In-app purchases (via app store) handle the subscription. A **7-day free trial** may be offered to drive conversions [5]. The subscription gating is mirrored in UI: free users see lock icons or blurred content on premium features [6], along with frequent upgrade CTAs ("🔓 Unlock full signals") [7] [8]. Premium users see all content and a "Pro" badge or welcome message confirming their status [9].
- **Token-Based AI Assistant:** A standout feature is the **Signal AI** – an in-app chat assistant that provides deeper insight on signals and trading strategies. This AI mode is accessed via **consumable tokens** [10]. Users (subscriber or free) purchase token packs, and each AI query deducts tokens (pay-per-use model) [11]. The AI is fine-tuned for finance, using live market data and the context of each signal to give real-time, relevant answers (e.g. explaining strategy rationale, updating predictions) [12] [13]. The token system monetizes heavy AI usage while keeping subscription costs controlled. (For example, 20 free tokens might be gifted at signup [14], and users can buy more as needed via the Token Purchase screen.)
- **Dynamic Signal Detail Page:** Each signal's detail page transitions from static to dynamic when AI and live data are active. Initially, a signal card shows static info (from the time it was generated). With live feeds, the **price chart updates in real-time**, reflecting current quotes and any movement since signal issuance [13]. If the underlying asset's price hits certain thresholds (target reached, stop-loss hit, significant volatility), the app triggers an **AI update** – the assistant may proactively add a comment in the chat (e.g. "**AI:** The stock hit

the target price; this signal is now closed, locking in ~10% gain.”). Users can also manually request updates via the chat (“What’s the stock price now? Has the outlook changed?”), and the AI will recompute analysis using the latest data. The AI chat is **contextual to the signal’s lifecycle** – it knows the specific signal ID, content, and status [15], allowing focused, relevant dialogue tied to that trade. This contextual integration is a key differentiator from generic chatbots, ensuring the conversation stays on-topic and updates as the signal evolves [16].
- **UI Design & Patterns:** The app’s look-and-feel will use dark mode finance dashboards (as in the provided HTML v13, with dark background, gradient accents for highlights). We leverage UI components from the FlutterFlow marketplace kit (e.g. pre-built list tiles, card layouts, chat bubbles) and the Figma mobile kit for consistency. The **Option Signal – Flow Alert** app provides a reference for features like categorized signals, watchlists with mini charts, and in-app notifications [17] [18]. Similarly, common fintech UX patterns (clean typography, color-coded tags for strategies, clear call-to-action buttons) will be followed. The interface will be optimized for native performance in FlutterFlow, ensuring smooth transitions (e.g. swipeable onboarding slides, real-time chart animations) and compliance with app store guidelines.

Below, we break down the requirements **screen-by-screen**, detailing the UI components, data binding, and logic for each. User stories with acceptance criteria are provided for critical flows to guarantee the design meets the intended user experience.

Screen Specifications

1. Login & Registration Screen

Purpose: Allow users to securely sign up and sign in to the app, establishing an account (needed for saving preferences, subscription status, and token balance).

UI Layout: A clean form with the app’s logo at top, followed by input fields and action buttons. Components (FlutterFlow widgets) include:
- **Logo & Tagline:** A gradient-text logo (matching the app’s name, e.g. *SignalPro*) with a brief subtitle (e.g. “AI Trading Signals”). This mirrors the styling from the HTML header (bold gradient text) for branding consistency [19].
- **Email Text Field:** For user’s email (FlutterFlow TextField with validation).
- **Password Text Field:** Secure input for password. Possibly an eye-icon to toggle visibility.
- **Login Button:** Primary CTA button (“Log In”) styled in the app’s accent color (e.g. green/teal gradient).
- **Alternate Sign-Up Link:** Text/button toggling to a **Sign Up** flow (e.g. “New here? Create an Account”). If using separate screens for login vs. sign-up, tapping this navigates to a similar Registration screen where users input name, email, password (and possibly referral code). Alternatively, this could be a single form that switches modes.
- **OAuth options (optional):** Buttons for “Continue with Google/Apple” if FlutterFlow supports social login integration. These provide quicker onboarding on mobile.

Data & Integration:
- Leverage FlutterFlow’s Firebase Authentication integration for account creation and login. On “Log In”, call Firebase Auth signIn; on “Sign Up”, create user auth entry and a corresponding Firestore user profile document (storing any default settings, like initial token count, trial status, etc.).
- Password rules: minimum length and complexity as needed. Provide user feedback on invalid login (e.g. snackbar “Incorrect email or password”).

- If login is successful, navigate to next appropriate screen (Onboarding if first-time or Dashboard if returning user with profile set up).
- If using third-party logins, handle through FlutterFlow's auth actions and ensure to fetch/display user name/photo if available.

States & Logic:
- **First-Time User:** After sign-up, route directly to the **Onboarding Questionnaire** to collect preferences (acceptance: the onboarding must show only once for new accounts).
- **Returning User:** If a user is already authenticated (persisted login), bypass this screen on app launch and go to Dashboard directly. Otherwise, show Login.
- Error Handling: Show inline error messages for invalid fields (e.g. email format) and catch auth errors (like user not found, wrong password). Provide a "Forgot Password" link that navigates to a reset password screen or triggers Firebase password reset email.

User Story: *"As a new user, I want to create an account quickly so that I can access personalized trading signals and track my usage."*
Acceptance Criteria:
- *AC1:* When a user enters a valid email and password and taps "Log In", they are authenticated and taken to the next screen (onboarding or dashboard). Invalid credentials prompt an error message without navigation.
- *AC2:* The Sign-Up process requires a unique email and valid password; on success it creates a user profile and proceeds to onboarding. Duplicate emails or weak passwords are prevented (with clear messaging).
- *AC3:* If a logged-in session exists (e.g. token stored), the app skips the login screen on launch, providing a seamless return experience.
- *AC4:* Tapping "Continue with Google" (if implemented) authenticates via OAuth and then navigates forward. Tapping "Forgot Password" allows the user to reset their password via email.

2. Onboarding Questionnaire Screen(s)

Purpose: Gather user profile data (trading experience, interests, goals) and introduce core app features. This tailors the content (e.g. which signals to highlight) and improves engagement. Also, it may present the subscription offer if the user is not yet premium.

UI Layout: A multi-step onboarding flow, each step on its own sub-screen or using a PageView widget. The design is user-friendly with illustrations or icons and concise questions. For example:
- **Step 1: Experience Level** – Radio buttons or a segmented control: *"What's your trading experience?"* Options: **Beginner**, **Intermediate**, **Advanced**. This might adjust later UI complexity (e.g. beginners might see more tooltips, or simpler dashboards [20] [21]).
- **Step 2: Interests** – Multi-select chips or checkboxes: *"Which markets are you interested in?"* Options: **Stocks**, **Options**, **Crypto**, **Forex**, etc. Users can pick multiple. This influences what signals they'll see by default (e.g. if only crypto is chosen, the Signal List may filter to crypto category or send more crypto notifications).
- **Step 3: Trading Style** – A question like *"Preferred trading timeframe?"* with options: **Day Trading**, **Swing Trading**, **Long-Term Investing**, etc. Again used for personalization (e.g. what signal categories to emphasize).
- **Step 4: (Optional) Subscription Upsell** – If the user signed up without purchasing a subscription yet, this screen pitches the Premium plan. It might say *"Unlock full access: Get all signals and AI analysis"* with highlights of premium benefits (e.g. number of signals per day, real-time alerts, AI assistant). Show pricing

for Monthly and Annual, and a button to start a free trial or subscribe. If the user chooses to subscribe here, integrate in-app purchase flow (via app store). **If the user skips**, they remain a free user and can still proceed (the app will just operate in freemium mode).
- **Trial Note:** If offering a free trial, emphasize "7-day free, cancel anytime" [5] to reduce friction. The UI should clearly allow skipping or opting for later, to avoid user drop-off.
- **Final Step: Confirmation** – A "Let's Start" button after all info is gathered. Possibly display a summary: *"Welcome, [Name]! You're set as an [Intermediate] trader interested in [Stocks, Options]."* If the user subscribed in step 4, mention "Premium Unlocked!"; if not, encourage them to upgrade later via profile.

Each step will have a **Next** button (or **Skip** if appropriate), and perhaps a progress indicator (e.g. "Step 2 of 3"). Use FlutterFlow Widgets like CheckboxListTile, RadioListTile, or custom buttons for selections. The visual style remains consistent (dark background, white text, accent-colored next button). Include small **illustrations or icons** on each step for friendliness (e.g. a chart icon for experience, a bull vs bear for style, etc., possibly sourced from the UI kit or FlutterFlow's icon library).

Data & Integration:
- Bind the input fields to a local state or Form widget. On completion, save these preferences to the user's Firestore profile (e.g. fields: experienceLevel, interests array, tradingStyle). These values will be used in the Dashboard to filter content (e.g. if a user is beginner, the Dashboard might show a quick tutorial link or fewer complex metrics).
- The subscription step (if included) uses RevenueCat or StoreKit integration via FlutterFlow's in-app purchase actions: tapping "Subscribe" triggers the purchase. On success, update the user's profile (e.g. `isSubscribed=true`, `subPlan=Monthly`, `subExpiryDate=...`). Also, as part of welcome, grant some **free tokens** as a signup bonus – e.g. "20 tokens have been credited to your account to get you started" [14]. This will reflect in the Profile/Token balance.
- If the user declines subscription, mark them as `isSubscribed=false` (free tier). The app will later enforce feature gating accordingly.

Navigation: After final step, navigate to the **Dashboard (Home)**. Ensure that if a user goes back to onboarding from profile to change preferences later, it's handled (not common, but possibly allow editing interests in Profile rather than re-run onboarding).

User Story: *"As a user, I want to quickly set my experience and interests so the app can customize signals and not overwhelm me with irrelevant info."*
Acceptance Criteria:
- *AC1:* The onboarding flow presents questions in a logical sequence and saves the responses. For example, selecting "Beginner" and "Crypto" updates the user profile; the Dashboard then might show a simpler view with crypto signals prioritized.
- *AC2:* Users can skip non-mandatory steps (like the subscription offer) and still complete onboarding. Skipping the premium upsell keeps them on free tier, which the app reflects (locked features visible with upgrade prompts).
- *AC3:* If the user chooses a subscription during onboarding, the payment is processed and upon success, the user is marked premium and taken to the Dashboard with all features unlocked. A failure or cancellation returns them to the free path (with no features unlocked, but they can still continue).
- *AC4:* The UI should be welcoming and not tedious – minimal steps (3-4 max) with clear progress indication. Users should not be able to proceed without answering required questions (e.g. experience level), except for steps explicitly marked optional.

- *AC5:* Data binding works: the answers are stored in the backend. If the user logs out and back in, their preferences persist (Dashboard personalization does not ask again). If a returning user who already completed onboarding logs in, they skip directly to Dashboard [22] [23].

3. Dashboard (Home) Screen

Purpose: Serve as the central hub after login – providing a summary of content and quick access to core features. It surfaces relevant signals or alerts based on user preferences and highlights the AI assistant and subscription status. Essentially, the Dashboard greets the user and provides entry points into Signals and AI features.

Layout & Components: The Dashboard is a scrollable screen (FlutterFlow Column or ListView) with distinct sections. Key UI elements:
- **Header/Welcome Banner:** A top section greeting the user, e.g. "Good Morning, [Name]" (fetch from profile). If the user is premium, include a small "Premium " label next to their name or avatar for recognition [24]. If the user is free, the header might include an upgrade prompt (e.g. a highlighted banner "🔓 Unlock all signals & AI – Go Premium" that links to subscription screen [7]).
- **Token Balance & Quick AI Access:** Just below the greeting, show the user's token balance (for AI usage). For example: a pill or card: "Tokens: **{balance}**" with an icon. If tapped, navigate to Token Purchase screen. If the user is free (not subscribed), this section might be dimmed or have a tooltip like "Subscribe to use AI tokens" (since in our model only subscribers can buy/use tokens; however if allowing free token usage, then show it normally but their balance likely 0 until purchase). This constant visibility of tokens educates users on the AI feature availability [25]. Next to the balance, provide an **"Ask AI"** button – a one-tap entry into the global AI chat or the AI Signal generator. For instance, an icon of a chat bubble labeled "AI Assistant". Tapping it opens the **AI Assistant screen** (or modal) where users can ask general questions ("What's a good stock for 5 years?" etc.). This global AI chat is separate from signal-specific AI (which is accessed via Signal Detail), but it uses the same token system. *(Note: The PRD focuses on signal-specific AI, but including a general AI chat entry on the dashboard can be a value-add. If scope is limited to per-signal AI only, this button could instead prompt the user to select a signal first.)* [26] [27].
- **Latest Signals Overview:** A section showing a few recent or relevant signals. This could be a horizontally scrollable list of *signal cards* or a vertical list of 2-3 items. Each card shows brief info: Ticker, strategy type tag, maybe an icon indicating profit/loss if closed, or "New" if recently added. For example, "TSLA – *Call Spread* – Target +15%" with a small chart thumbnail. These cards are a subset of what's in the full Signal List, filtered based on user interests (e.g. if user chose Crypto, show latest crypto signals first). Tapping a card goes to **Signal Detail**. This section ensures users see content immediately on login. If the user is free, these cards should still appear but with limited info (e.g. a lock overlay or blurred target price), reinforcing the tease. A message like "Full details for premium members" can be overlayed on the card hover [8].
- **Watchlist Summary:** If the user has added any signals or tickers to their watchlist, display them here. For example, a small list or grid of watchlist items with current price and daily change. This mirrors the Option Signal app's watchlist feature, which allows unlimited watchlist items and shows a mini chart with timeframe toggles [17]. In our app, a watchlist item could be a stock or crypto the user manually added or marked from a signal. Each item shows ticker, current price, and maybe a 1-day sparkline chart. If none in watchlist, show an empty state (" Your watchlist is empty. Add signals or assets to track them here.") [28] [29].
- **Alerts/Notifications Teaser:** A section for any important alerts. For example, if a big market event happened or if one of the user's active signals has an update, show a highlighted card. E.g. " *AAPL hit its target!* – tap to view outcome." This can draw attention to significant changes. This might be dynamically

generated by the backend.
- **Educational or Tip (optional):** A small banner like "Tip: Chat with the Signal AI to understand trade rationale better [30]." – providing hints to engage with features.

Data Binding:
- The Dashboard aggregates data from various sources: user profile (name, token balance, subscription), signals (from Firestore or API feed), and watchlist data (could be stored in Firestore or fetched from an API for live prices). Use FlutterFlow Stream/Query widgets: e.g. a Firestore collection "signals" for latest signals, filtered by date and category. For watchlist, if watchlist items are stored in user document, fetch each item's price via an API or a combined Firestore query if such data is mirrored in the database.
- The token balance should update in real-time if the user purchases tokens or spends them. Possibly listen on the user document's `tokenBalance` field for changes (so after a chat query, the balance decrement is reflected).
- If implementing a global AI chat, ensure token availability is checked on tap: if balance is zero, direct user to purchase tokens first.

Navigation & Actions:
- Tapping a signal in Latest Signals -> opens **Signal Detail** for that signal.
- Tapping "View All Signals" (if there's a button or if user taps a section header) -> opens **Signal List** screen.
- Tapping a watchlist item -> could open either a simple price chart or the related signal if the item came from a signal. If it's just a ticker, perhaps open a basic detail or a webview to Yahoo Finance (or a future feature to show a mini detail). For now, this can route to Signal List filtered by that ticker if applicable.
- The "Ask AI" / "AI Assistant" button -> opens the **AI chat screen** (global context). In that screen, user can ask general questions not tied to a particular signal. *(If global AI chat is not in MVP scope, this button could be repurposed to something else like "New AI Signal" described next.)*
- **AI Signal Generation (On-Demand)**: The Dashboard may also feature a quick action to generate a new AI-driven signal idea. For instance, a button " New AI Signal" or specific options like "Live Signal: 📊 S&P500 Now". This follows the user flow described in the logic research, where a user requests a custom signal and the AI produces a strategy [31] [32]. If included, tapping it would prompt a dialog: "Use 20 tokens to generate a fresh signal idea?" [33]. On confirm, show a loading state then navigate to a Signal Result screen (or reuse Signal Detail layout) showing the AI-generated signal and a note of tokens deducted [32] [34]. This is an advanced feature; for MVP it might be skipped, but the design can accommodate it via the Dashboard quick actions area.

Conditional UI (Free vs Premium):
- If **Free user**: The Dashboard should prominently display upgrade inducements. For example, the Latest Signals might show only one teaser with "Upgrade for more signals". The AI sections (token balance, Ask AI) might be hidden or show a lock ("Premium Only"). Free users can still see their watchlist and any free signals provided. Also, consider showing a banner: "You are on Free tier – you're receiving up to 3 signals/week [35]. Upgrade for 3-5 signals/day and AI insights." This constant reminder can convert users.
- If **Premium user**: Show everything unlocked. They see all latest signals, their token balance, etc. The upgrade prompts are removed, possibly replaced with a subtle "Premium Member" tag. Ensure premium users never feel they're seeing a "lite" version – all content sections should be populated.

User Story: *"As a returning user, I want a home screen that highlights new trading signals and my status at a glance, so I can quickly decide what to do next (read signals, chat with AI, etc.)."*
Acceptance Criteria:

- *AC1:* Upon login, the Dashboard loads with the user's name and current status (premium or free) clearly indicated. Free users see an Upgrade CTA; premium users see their token count and no paywalls.
- *AC2:* The "Latest Signals" section shows at least the 1-3 most recent signals relevant to the user's interests. Each item correctly reflects the data (ticker, brief description, etc.). Clicking an item leads to the correct detail page. If a signal is locked for free users, tapping it prompts the upgrade flow instead of showing full details (e.g. an interstitial "Subscribe to view this signal" with a link to plans).
- *AC3:* The token balance indicator updates when tokens are added or spent. For example, if the user uses the AI chat and spends 1 token, returning to the Dashboard or viewing the balance should reflect the decrement (e.g. 19 -> 18 tokens).
- *AC4:* The "Ask AI" (global chat) or "New AI Signal" features respect token requirements: if user has enough tokens, they can proceed. If not, the app gives a friendly prompt to buy tokens. No action that costs tokens should silently fail – always inform the user of cost and confirm usage [33] [36].
- *AC5:* The UI is responsive and clean: important elements (like new signals or alerts) are visible without scrolling on a typical phone. Secondary info (like watchlist) is accessible by scrolling. Performance: The feed should load quickly (under 2 seconds) due to efficient queries (perhaps limit to ~5 items until navigating to full list).

4. Signal List Screen

Purpose: Display the full list of trading signals available to the user, with filtering by category and status. This is where users can browse all current opportunities and review past signals. It's essentially the "Signals" tab of the app, akin to the main feed in Option Signal app, showing multiple types of alerts (swing trades, options, crypto plays, etc.).

UI Layout: A vertically scrollable list of **Signal Cards**, each representing a trading signal. The screen also includes controls for filtering and segmentation:
- **Segmented Tabs (Active/Closed/Watchlist):** At the top, provide tabs or toggle buttons to switch between **Active Signals** (open, actionable trades), **Closed Signals** (completed or expired trades), and **My Watchlist**. Only one list is visible at a time. Using FlutterFlow, this can be a TabBar widget with TabView pages, or a custom toggle that shows/hides containers (e.g. by binding a variable to list visibility). The HTML v13 design indeed had separate containers for active vs. closed vs. watchlist signals [37] [28]. The default view is "Active Signals". Closed signals provide historical results (and possibly learnings), and Watchlist shows items the user manually marked.
- **Category Filter Pills:** A horizontal scroll of category chips that filter the list by strategy/type. Examples of categories: "All", "Stocks", "Options", "Crypto", "YOLO", "Swing", "Long Term", "Earnings Play", etc. These correlate with tags used on signals. In the design, these appeared as pills with counts [38] [39]. One pill can be active at a time, highlighting its selection (active pill has a different background and border color, e.g. glowing green outline). For example, tapping "Crypto" filters the list to only show crypto-play signals. A pill might also show the count of signals in that category [40]. The "All" or "Trending" pill could reset the filter to show everything. Scrolling is enabled for categories to handle overflow (with no visible scrollbar on mobile, using touch swipe) [38].
- **Signal Card Item:** Each signal is presented in a card container with a consistent layout:
- **Header Section:** Contains the Ticker symbol (e.g. **AAPL**), possibly the company name or a short descriptor, and a strategy badge. The strategy badge is a colored label indicating the type of play (e.g. "IPO Today", "YOLO Calls", "Swing Trade", "Earnings Beat" etc.) [41] [42]. These badges use distinctive background colors per strategy (as defined in the HTML CSS, e.g. IPO = red gradient, crypto = orange, etc. [43] [44]). The header might also display an icon for the market (e.g. a small Bitcoin icon for crypto, etc., or the app could rely on

text only).
- **Summary Stats:** Key numbers of the trade in a compact row or column. For instance: *Entry:* $X, *Target:* $Y, *Stop:* $Z, *Risk/Reward:* 1:3, etc. The HTML design shows various stats like Max Gain, If Fail, etc., likely customized per strategy [45] [46]. We will simplify: for each signal, at least entry price and target price are shown (if it's an entry signal), or if it's an informational alert (like "News event play"), maybe a different stat (e.g. current price move). For options signals, maybe strike and expiry are listed. These details can be formatted as small labels. If the signal is *closed* (in the Closed tab), the stats could show the result: e.g. "Result: +20%" or "Stopped: -5%", possibly with a green or red color and a checkmark or cross icon to indicate success/failure [47] [48].
- **Brief Description:** One or two sentences summarizing the opportunity or outcome. E.g. "20:1 split announced. Historical avg +15% to split date. Buy shares or Aug calls. Retail FOMO incoming. [49]" This text comes from the signal's rationale field. Keep it concise in the list view (maybe truncate after 100 chars). For closed signals, it might be a summary of what happened ("Sold at $103 three days later, 300% profit [50]").
- **Mini Chart or Visual Cue:** The static design included a mini chart canvas on some cards [51] [52]. In FlutterFlow, we might include a placeholder chart image or an icon indicating performance. To start, a simple approach: use an icon or emoji to denote trend (📈 for going up, 📉 for down) or a small sparkline image if available. Later, a custom widget or integration (e.g. ChartFlutter library) could render actual charts. In the UI, this chart would appear beneath or behind the text. For example, the card might have a background section with a faint chart line, or an inline small chart next to stats. (Ensuring performance: too many live charts could be heavy, so maybe only show live mini charts for watchlist items; for main list, static images updated periodically).
- **Decorative Highlights:** Certain special signals might have unique styling: e.g. a "YOLO" signal card had an animated neon border in HTML [53]. We can mimic this by adding a conditional decoration: if signal.riskLevel == "YOLO" or category == "YOLO", give the card a pulsing border (FlutterFlow can use animated containers or Lottie animations). IPO plays might have a colored border, etc., matching the design classes for `hot-ipo`, `yolo`, etc. [54] [55]. These touches make the list visually engaging, highlighting urgent or very risky plays.

- **Infinite Scroll/Pagination:** The list could be long, especially for closed signals. Implement lazy loading or pagination (FlutterFlow supports a ListView with a certain batch size, loading more on scroll). Alternatively, closed signals might be limited to the last N or separated by date sections. We ensure at least the active tab (which likely has fewer items) loads all active signals.

Behavior and Data:
- By default, when **Active tab** loads, query the "signals" collection for all signals where status == 'active/open'. Apply category filter if one is selected (e.g. a field `category: Crypto` or `type: Swing` in the data). If "All" is selected, show all active signals. Sorting: likely by time (newest first) or by an importance rank.
- On **Closed tab**, query signals where status == 'closed'. Possibly sort by closed date or by performance. Could also group by "Successful" vs "Failed" if desired (not required, but maybe use color-coding: e.g. green text for positive outcomes, red for negative).
- On **Watchlist tab**, if our watchlist consists of specific signals (user flagged them) or just tickers: - If it's signals, we can filter the signals collection where id in user.watchlist array. - If it's arbitrary tickers, we'd need to pull those from user's data and then fetch current prices separately. Perhaps simpler: allow watchlisting a signal which implicitly watchlists that ticker. Then in watchlist tab, list either the original signal (if still active/

closed) or just the ticker info if no active signal. For MVP, watchlist could just mirror active signals subset. - Provide a way on each Signal Detail to “Add to Watchlist” to populate this.

- Each card has an **onTap**: Navigate to **Signal Detail** screen, passing the signal ID (or full object).
- If the user is **Free**: enforce gating – in Active tab, free users should see only a limited subset. This can be done by backend (flagging some signals as free ones) or in UI by hiding some. A common approach: mark a few signals as “free_sample=true” in the data. The app can then show those to free users and hide the rest behind blurred cards or a message “Upgrade to see X more signals”. For example, show the first signal in each category, or the first 3 active signals, and then a big blurred card that says “+ 5 more signals locked – subscribe to view.” [56] [57]. On tap of a locked card or a signal that’s premium, show a dialog or route to subscription purchase. Similarly for closed signals, maybe free users can’t see the outcomes (or maybe they can, since it’s retrospective?). The PRD suggests likely locking full details for free even in closed.
- Free user detail view handling is described in that even if they tap, they get a partial view with prompt [58] [59]. So the list itself can allow tap but then detail is where upgrade is prompted. Alternatively, prevent tap by showing a lock icon on the card itself. Either approach is acceptable UX; perhaps letting them tap and then showing a paywall screen might be more persuasive (as they took the action).

User Story: *“As a user, I want to browse a list of all available trading signals and filter by type, so I can find opportunities that match my interests (e.g. only crypto signals).”*
Acceptance Criteria:
- *AC1:* The Signal List screen should default to showing all active signals. All signals are presented in a consistent card format with correct data fields. The design of each card (ticker, badge, short description, key stats) matches the signal’s attributes (e.g., a “Post-ER” badge for an earnings play) and uses the defined color-coding for that category [60] [61]. - *AC2:* Category filter pills work: tapping a category immediately refilters the list (e.g., selecting “Crypto” shows only crypto signals). The active pill is visually distinct [40] [62]. There is an “All” or equivalent to show everything. These filters do not require a full screen refresh (should ideally update via query or locally filter fetched data). - *AC3:* Switching to the Closed tab shows completed signals with outcome data. For each closed signal, its card clearly indicates result (profit or loss). If a signal has a special outcome (e.g., “Nailed the hot IPO – +300%” [50] or “Failed – -100%” [48]), that is shown prominently, perhaps in the card header or replacing the stats section, and maybe an icon like ✔ or ✖. This provides a historical record. - *AC4:* The Watchlist tab displays any signals or tickers the user saved. If none, an empty state is shown with guidance. If present, each item displays at least the ticker and current price (or last close price), updating periodically if live. - *AC5:* **Freemium gating:** Free users should see that some signals are locked. For example, after 1-2 free signals, subsequent cards might be blurred with a lock overlay and text “Premium Signal – Upgrade to view” [2] [8]. If a free user attempts to tap a locked signal, one of two things happens: (a) they are diverted to a **Paywall screen** detailing subscription benefits and pricing, or (b) a modal pops up saying “This is a premium signal. Subscribe to access full details and many more signals.” with options to subscribe or cancel. They should not be able to see the content unless they upgrade. (Whereas premium users obviously see all signals and can tap freely.) - *AC6:* Performance: The list should load within a couple seconds and scrolling should be smooth at 60fps. Use lazy loading for closed signals if needed to avoid huge loads. The UI should be responsive across different device sizes (FlutterFlow handles many adaptions, but test that cards display well on small vs large screens).

5. Signal Detail with AI Screen

Purpose: Show the full details of a selected trading signal and enable interactive AI analysis for that signal. This screen combines static trading information (the “signal card” details) with dynamic components: real-time updates (price movements) and an AI chat interface. It’s the heart of the AI-enhanced experience: users can understand the trade rationale, see live progress, and ask the AI questions about the trade.

Layout: This screen will likely use a Scrollable Column (possibly with sections pinned or collapsible). Major sections include:

- **Signal Info Header:** At the very top, show the signal’s identifier and summary. This includes:
- **Ticker & Company Name:** Large text for the ticker (e.g. *AAPL*) and the full name or brief descriptor (e.g. “Apple Inc.” or “Breakout Alert”). The ticker may be styled boldly. If the signal came from a particular category, optionally show an icon (e.g. a small emoji or icon for the category – 📈 for stocks, 🪙 for crypto, etc.).
- **Signal Type Badge:** A colored label showing the strategy type (like the badge on the list, e.g. *“Swing Trade”*, *“Earnings Play”*, *“YOLO”*). This badge reinforces the context. The badge color and style match the list (from the CSS classes we have, e.g. `.earnings-play` purple, `.yolo-play` flashy, etc. [60]).

- **Status/Live Indicator:** If the signal is active, perhaps show “Open” or a green dot indicating active. If closed, show “Closed” and maybe the result (P/L percentage). If active and being updated in real-time, could also show a blinking “Live” icon to indicate the feed.

- **Price & Performance Overview:** A prominent display of key numbers: Entry Price, Current Price (live), Target Price, Stop Loss, and perhaps % change since entry. Format example:

- Entry: $100.00
- Target: $115.00
- Stop: $95.00
- Current: $108.50 (+8.5%) – the current price updates with a small delay (maybe via WebSocket or polling). Green or red color based on movement relative to entry.

- Risk-Reward: 1:3 (or “Moderate Risk” label). These can be arranged in a grid or table-like layout (two columns, multiple rows), or as cards. E.g. FlutterFlow could use a Row of small Container cards for each metric. Important to highlight gain/loss. Use colors: if current price is above entry, show green and maybe a ↑ arrow; if below, red with ↓. If closed, show final outcome in place of current.
  For option signals, include contract details like “Strike $X, Expiry date”. If the signal has a time horizon, show “Timeframe: 3 days” or “Day Trade” etc.
  These stats mirror what an analyst would include on a signal card. They provide quick facts to the user without needing the AI.

- **Interactive Price Chart:** A central component is a chart showing price history and projections. Using Chart.js in the HTML indicates the intention for dynamic charts [51] [52] . In FlutterFlow, implementing a live chart might involve a custom widget or using Flutter packages like `syncfusion_flutter_charts` or `fl_chart`. For the PRD, we specify that a **line chart** will display: historical price leading up to the signal and potentially an indicated target/stop levels. For example, a vertical line or marker at the entry time, a horizontal line for target and stop. If the AI has any "prediction bands" (the HTML shows "← now | prediction →" or similar [51] [63]), we can illustrate an expected range beyond the current time. The chart should update in real-time as new price data comes in. For instance, if a stock is currently $108.50 and moves to $109, the chart line moves. Use an API or streaming service (like Alpaca or Finnhub) to fetch live quotes. The app might poll every few seconds or subscribe to a socket. (At least update on user prompt or when they open the screen, to fetch the latest price). If real-time updates are too advanced, update on important triggers (like crossing target). Include small labels on the chart: e.g. "IPO $31 → $103 peak" or "← now | prediction →" as in design [51], to contextualize events. But those specifics might be automated if AI contributes. At minimum, ensure the chart conveys current progress: perhaps shading the area between entry and current price, etc. This component is critical for visual traders. It ties into the "auto-update on price movement" requirement: as price changes, the chart is redrawn or the marker moves accordingly, without requiring user refresh.

- **Textual Analysis/Rationale:** A section with the original human analysis that came with the signal (if any). For example, the signal description: "**Rationale:** Company beat earnings and raised guidance; expecting a 5% continuation in next 3 days [64] [65]. Backtest shows 70% win rate on similar setups." This provides context. We can style this as a paragraph or bullet list. Possibly break it into sub-points: *Catalyst, Strategy, Risk*. This static content might overlap with what the AI would explain, but it's good to present it upfront. It also grounds the AI – the AI can elaborate on these points rather than just repeat. For free users viewing a premium signal, **this rationale or certain fields may be partially hidden** [8]. For instance, we could show: "Rationale: *** (upgrade to see analysis)" or show only the first sentence and blur the rest. And hide specific numbers like target/stop (maybe show as "$XX" placeholder). This is part of teaser UX, as described: *"free users get an alert... The signal page can tease some info and have an Upgrade CTA right there"* [2] [58]. The upgrade CTA can be a banner within the detail page for free users: e.g. inline after the first paragraph: "🔒 Upgrade to Premium to reveal full signal details and AI insights." Tapping it goes to subscription purchase. If the user is premium, of course all text is visible.

- **AI Chat Interface:** The bottom (and substantial) portion of this screen is the AI assistant chat, contextual to this signal. This component functions like a messaging interface: a scrollable chat history between the **user** and the **Signal AI**. Key elements:

- **Initial AI Comment:** When the user first opens a signal detail (and if they have access to AI), the AI can proactively provide an initial analysis of the signal. For example, it might say: "*AI:* I'm monitoring this trade. Currently, it's up 8% from entry. This signal was identified as a post-earnings play, which historically sees follow-through in 3 days [66] [67]. The risk-reward is favorable (1:3), meaning potential $15 gain vs $5 risk [68]." This initial insight gives value immediately and is derived from the signal data. This message could either appear automatically for premium users when they open the page, or after they tap a prompt like "Explain this signal". (Perhaps a UI choice: show a button " AI Analysis" that, when tapped, generates this message using 1 token. Or always show one free AI blurb if they have any token balance as a teaser). We need to decide if the first AI response costs a token or is complimentary; possibly it should cost since it uses the AI API, but maybe we can automatically deduct a token upon entering if the user agreed (but better to be explicit). Perhaps better: have a visible button "Ask AI for Analysis (1 token)" – user taps, then AI message appears and token deducts. After that, the chat is open for further questions.

- **User Input Field:** A text box where user can type questions or prompts to the AI, with a send button. For example, the user might ask: "What does an 'aggressive' risk level mean here?" or "If I already own shares of this stock, should I still take this trade?" [69] [70]. The input area should indicate the cost of asking. Perhaps a label like "Each question uses 1 token [71]." If the cost might vary by question complexity, we keep it simple for UI: assume 1 token per standard question (if certain heavy queries could cost more, the AI can inform before proceeding, but that's edge case).
- **Chat Messages:** Display the conversation. User messages on one side (e.g. right side bubbles), AI responses on the other (left side, perhaps with a different color bubble or a small "AI" avatar). The chat should maintain context for the session. The AI's responses should incorporate the signal's data plus any user-provided context. For instance, if the user asks a scenario ("how would this trade change if I hold 100 shares already?"), the AI can factor that in [69] [72] and answer accordingly (e.g., cautioning about exposure overlap) [73] [74]. Technically, the backend must handle keeping conversation state and feeding relevant data to the AI model. The UI just displays whatever the AI returns.
- **Auto-Updates via AI:** The AI may inject messages on its own when triggered by events. For example, if the stock hits a new high or drops significantly, and the user still has the detail page open, the app might push a message: "*AI:* Update – the price just fell below the stop loss. This signal would be considered failed. It might be wise to exit the position to limit loss." This would be displayed as an AI message without a user prompt. To the user, it appears the AI is monitoring proactively (which it is, with live data). These automated messages might or might not cost tokens; presumably they should not charge the user since they didn't ask (or it could be considered included in the subscription service to have AI monitoring). We will treat them as included alerts (no token deduction for push updates).
- **Token Consumption & UI Feedback:** Every time the user sends a question, immediately deduct the token(s) and maybe reflect that in the UI (e.g. token count in header decrements, or show a small "-1 token" animation). The chat might also display system messages or footnotes like "(1 token deducted)" under the AI's answer for transparency [75] [76]. If the user has 0 tokens, the input field should be disabled, showing "Out of tokens – purchase more to ask AI" and maybe a quick link to buy. If user is free (no subscription) and tries to open chat, either prevent entirely or show a message "AI chat is a premium feature. Subscribe and get tokens to use it." (Unless we allow free with tokens, but earlier logic suggests requiring subscription for AI usage to ensure they have full context access).
- **End Chat / Reset:** Possibly provide a "Reset conversation" or "Start new topic" option if needed, which would clear context. But since it's tied to a single signal, maybe not needed unless the conversation gets too long or off-track. The AI's domain focus will naturally keep it on this signal or related strategies [77] [78].

Data Flow:
- The signal's details come from Firestore/API (e.g. a "signals" collection document contains all fields needed: ticker, entry, targets, rationale text, category, timestamps, etc.). The chart data might be partly static (historical) plus dynamic (live). Historical prices around the signal time can be stored or fetched from an API on demand. Real-time price from a streaming API like Alpaca (the PNG files in user_files indicate Alpaca crypto API usage). We would integrate a cloud function or use FlutterFlow's API integration to get latest price repeatedly.
- The AI chat uses an external AI service (likely OpenAI GPT via an API that the backend orchestrates, given it needs the app's data). The user's question, along with context (signal info, prior chat messages) are sent to the API, which returns the answer to display. The token deduction logic ties into this call: before or after calling AI, subtract the appropriate token count from user's balance (ensuring not to go negative; if user

doesn't have enough, don't call AI and instead prompt purchase).
- The chat history might be stored temporarily (maybe not persisted between sessions for now, or maybe short-term if user leaves and returns to same signal within an hour, they see the conversation?). Could store last N messages in local state or not at all. Simpler: if user leaves detail and returns, they start fresh, with perhaps the initial AI analysis available again if they trigger it.

Free vs Premium Behavior on Detail:
- If a free user somehows navigates to a premium signal detail (perhaps via a teaser link), the detail page should show minimal info and encourage upgrade. For example: "This signal is for premium members. Upgrade to view entry and exit targets and unlock the AI analysis." Possibly show the chart with delayed data or no data. Essentially a tailored paywall page. In some cases, we might allow free users to see the static chart and the first line of rationale to show value, but hide the numbers (as described) [8] . The AI chat interface for free user will either be completely hidden or replaced with a big prompt to subscribe for AI. According to research, *both free and paid users can access AI mode with tokens* [10] , but if a free user doesn't have the actual signal data, the AI's utility is limited. We might allow it in a limited capacity (e.g. they could ask general questions about trading but not specifics of this signal since they can't see it fully). For simplicity, require subscription for using AI on signals. So free user sees a lock overlay on the chat input saying "Subscribe to ask AI". - Once the user upgrades (in the middle of viewing, if they tap upgrade), the detail page should seamlessly unlock: reveal the hidden fields and enable the chat.

User Story: *"As a premium user, I want to dive into each trading signal's full details and ask follow-up questions, so I can confidently understand and act on the trade."*
Acceptance Criteria:
- *AC1:* The Signal Detail screen displays all relevant information for the signal in an organized manner. Entry/exit/stop values are correct and clearly labeled. The current price updates to reflect live market data (within a reasonable interval, e.g. refresh every few seconds or on notable change) [13] . The chart shows the price movement and highlights key points (entry, target, etc.). If the price crosses a key level (target or stop), the UI indicates this (e.g. flashing or a note "Target hit!") and the signal status may change to closed.
- *AC2:* The original rationale or description text is present, enhancing the user's understanding. It should not just repeat raw data; it adds context (e.g. "historical average +5% after similar events" etc.). This matches how the AI might elaborate, providing a baseline. The text should be proofread and concise (no more than a short paragraph or a few bullet points).
- *AC3:* The AI chat interface is intuitive and responsive. When the user submits a question, an AI typing indicator could be shown (to signify thinking) and within a couple of seconds, a reply appears. The AI's response quality should reflect domain knowledge: it references the signal specifics (like "this trade is a post-earnings continuation; such signals often rely on momentum…") [66] [68] and possibly brings in extra data (if available, e.g. "current volume is high, which supports the breakout"). It **must not** hallucinate details that conflict with the signal card – ideally it uses the structured data provided [79] [80] . Acceptance is that answers are relevant and add value beyond the static info [81] [30] . - *AC4:* Token deduction occurs per question asked by the user, and the app reflects this properly. For instance, if the user had 10 tokens and asks one question, the balance becomes 9 without errors. If they have 0 tokens, the chat input is disabled or replaced by a "Buy tokens" prompt. Trying to send a question with insufficient tokens should not call the AI (and should prompt purchase flow). If the AI fails to respond (e.g. service down), handle gracefully: show "AI is currently unavailable, please try later" and **do not** deduct a token for that attempt [82] .
- *AC5:* The AI proactively providing an update (like in case of market movement) is handled correctly. For example, if the user leaves the detail screen open and the price crosses a threshold, within a short time an AI message appears noting the event (assuming backend triggers it). The message should be clearly from

AI and possibly prefaced with “Update:” to differentiate from a direct answer to user query. No duplicate or spammy updates – only significant ones. - *AC6:* If the signal is closed (either because time passed or target hit/stop hit), the detail screen should reflect that: perhaps overlay “Closed” on the chart, disable further AI questions on it (maybe allow questions but the context is final outcome?), and present the final outcome analysis. The AI can still answer retrospective questions (“What went wrong?” or “How could we have improved this trade?” – these could be interesting educational uses). Ensure the user experience cleanly transitions a signal from active to closed without confusion. - *AC7:* **Free user access:** If a free user somehow is viewing this screen for a locked signal, critical details (entry, target) are hidden and any attempt to reveal or use AI prompts an upgrade. Specifically, the acceptance is that free users cannot see exact trade levels [8] or full rationale, and the screen provides a clear call-to-action to upgrade for full access. For instance, a free user detail view might show: “Entry: $XXX, Target: $YYY (locked) [59] ” and a button “Upgrade to Premium for full signal details.” Pressing it goes to subscription flow. This effectively turns the detail page into a conversion driver for non-premium users.
- *AC8:* The UI is mobile-friendly: The content should fit without overly tiny text. The chat area especially should use available screen space well – possibly expanding to full screen height when active (with the keyboard up). The user should be able to scroll back up to see the chart while the chat is still accessible, etc. It might be useful to make the chart collapsible or the chat on a separate tab within detail if too crowded. But ideally, a user can scroll: top half info, bottom half chat. FlutterFlow’s responsive layout should allow scrolling the entire column, with perhaps the chat input pinned at bottom of the viewport when focused. Test on different device sizes to ensure usability.

6. Token Purchase Screen

Purpose: Enable users to buy token packs for AI usage. This screen is essentially a storefront for consumable in-app purchases, listing available token bundles and their prices, and showing the user’s current token balance for reference. It should also educate users on what tokens are and how they’re used (in case they access this screen out of curiosity).

Layout: A simple and clean purchase menu, likely a single page with list of products. Components:
- **Current Balance Display:** At top, show “You have **X tokens**” (big font). Possibly include a graphic (like a coin or token icon) for visual. If the user is not subscribed (and thus not eligible to use tokens in our model), this section might instead say “Tokens require Premium Access. Subscribe to enable AI tokens.” and show the subscribe button instead of packs. (Alternatively, as per earlier logic, maybe allow free to buy tokens but prompt them that full functionality is with subscription – but that’s complex; simpler: require subscription).
- **Token Packs List:** A vertical list of purchase options. Each option shows: **Pack Name**, **Number of Tokens**, and **Price**. For example: - Small Pack – 20 Tokens – $4.99
- Medium Pack – 50 Tokens – $9.99
- Large Pack – 120 Tokens – $19.99
- (If we have a subscription-plus model, maybe “Unlimited AI – included in Premium+” but currently not in scope; we stick to packs).
Each item has a “Buy” button. FlutterFlow can integrate with Apple/Google in-app purchases. We need the Product IDs configured. Tapping Buy triggers the purchase flow. - **Offer/Bonus Information:** Possibly mention if any bonus tokens or deals for bigger packs (e.g. “(Best Value)” tag on the large pack, or “Save 20%”). - **Usage Explanation:** Below or above the list, a small text block: “Tokens let you chat with the Signal AI. Each question or AI-generated signal uses tokens [11] [71]. Purchase more tokens as needed. Unused tokens roll over and never expire (but require an active subscription to use).” – This clarifies how the system works to the user and reinforces that tokens cost because the AI has a cost. - **Subscription Reminder (if**

applicable): If user is free and we do allow them here, perhaps a banner: “Premium subscription required to use tokens.” Or if user is premium, maybe a line: “Your subscription is active until [date]. Tokens are consumed only when you invoke AI; normal signal alerts do not use tokens.” - **Restore Purchases:** (For iOS) a small link to restore token purchases if needed (standard for consumables maybe not needed, but just in case). - Use a consistent style: dark background, cards or containers for each pack with slight gradient borders to stand out. Possibly incorporate the app’s gradient in the “Buy” buttons.

Logic:
- Only show packs available for the platform (some platforms might have different price tiers). These are configured in App Store/Play Console. FlutterFlow’s purchase actions will identify them.
- When a purchase succeeds, increment the user’s token balance in Firestore by the appropriate amount. Also, maybe log the purchase in a “transactions” subcollection for record. - Immediately reflect the new balance on this screen (and elsewhere via listeners). For example, if user had 5 tokens and buys 20, upon success the top should now read “25 tokens”. Possibly play a confetti animation or a success dialog: “Success! 20 tokens added to your balance ”. - If purchase fails or is cancelled, show an error or simply return to screen with no change, maybe a toast “Purchase cancelled”. - If user is free and tries to buy: We might intercept that action. Ideally, we would not even allow entering this screen if not subscribed; instead redirect them to subscribe first. Or allow it, but on attempt to buy tokens, check `isSubscribed`. If false, prompt “Subscribe to Premium to use AI tokens. Would you like to upgrade now?” as a modal. It could even offer to do both: first subscribe then purchase tokens. But that’s complicated flow and likely rare that a free user would try to buy tokens without subscribing if the UI makes it clear. So simpler: hide this screen behind subscription, or at least disable the buy buttons for free users with a note. - The **Profile screen** (later) will have a link to this Token Purchase screen. Also, from chat if tokens are low, tapping “Buy tokens” could deep link here. So ensure easy navigation back to where they came from (perhaps a back arrow to profile or detail).

User Story: *“As a premium user, I want to top-up my AI token balance easily so I can continue using the AI assistant without interruption.”*
Acceptance Criteria:
- *AC1:* The token store clearly lists the available packages and their prices. Prices match those set in the app stores. The UI is straightforward: tapping “Buy [pack]” triggers a native purchase prompt, and on confirmation, the user gets the correct number of tokens added. There should be no discrepancy (e.g. buying 50 actually gives 50 tokens in-app).
- *AC2:* The current balance updates immediately after purchase. If the user buys multiple packs in a row, those all accumulate correctly. If any purchase fails, the balance remains unchanged and an error is shown or logged.
- *AC3:* If the user is **not subscribed** and reaches this screen, the store should either: - Not allow a purchase: e.g. the packs are greyed out with a tooltip “Requires Premium subscription”, and a big button “Go Premium”. Or - Allow the purchase of tokens but warn them: e.g. if they tap buy, show “These tokens will be usable once you subscribe to Premium. Subscribe now to start using them.” This scenario might be confusing, so disallowing is cleaner. Acceptance: free users are appropriately guided to subscribe instead of buying tokens futilely. In any case, a free user should not be able to waste money on tokens they cannot use immediately.
- *AC4:* The explanatory text is present and correct – users understand what a token does and that one question = one token (or whatever the rule). No ambiguity like “maybe it costs more if...?”; keep it simple as per design: each prompt deducts tokens [71], and nothing is deducted without an actual answer delivered [83] [84]. The PRD mentions fairness: we explicitly do *not* charge tokens for signal delivery which is covered

by subscription [85], only for AI actions the user triggers. Make sure the UI messaging aligns with this (i.e. don't imply signals themselves cost tokens; they don't).
- *AC5:* Purchasing tokens requires an active internet and a logged-in user. If any state is not met (offline, no user), the screen should inform accordingly (though in normal use, they'd be online).
- *AC6:* The design of this screen should be consistent with the rest (use the same typography, etc.) even though it's a more static commerce page. Possibly include the app logo faintly or a token graphic to make it feel part of the app's brand.

7. Profile & Settings Screen

Purpose: Provide account-related functions – viewing or editing user info, managing subscription, seeing token balance, adjusting preferences, and accessing support or logout. It's the user's personal area.

Layout: A scrollable screen with sections for different settings. Common approach is a list of items (using ListTiles in FlutterFlow) grouped by category. For example:

- **Account Information:** Display the user's name, email, and membership status at the top. Perhaps use a Card with the user's avatar (could just be an initials circle if no photo) and name/email. Below that, a label for subscription:
- If premium: " Premium Member – Next billing: Aug 12, 2025" or "Plan: Premium (Annual) – Renews on 2025-12-01". Also possibly a "Manage Subscription" link (this could open the app store subscription management or our own screen if we implement it).

- If free: "Free Tier – Limited access" and a prominent **Upgrade to Premium** button here. (This is another spot to convert: the logic docs suggest having an unlock button visible for free users in profile [7] [86]). Tapping it goes to a subscription purchase flow (or shows plan options).

- **Token Balance & Store Link:** Show current token balance here as well (for convenience, redundant with dashboard but useful). Something like "AI Tokens: X remaining". If user is premium, next to it a button "Buy More" -> navigates to Token Purchase screen. If user is free, maybe this row is hidden or says "AI Tokens (Premium only) – Subscribe to enable". This duplicates the dashboard info but profile is a logical place users will look for anything account-related.

- **Preferences:**

- *Trading Interests/Filters:* Possibly list the preferences gathered in onboarding. E.g. "Interested Markets: Stocks, Crypto (Edit)" and "Experience Level: Intermediate (Edit)". Tapping these could reopen a simplified onboarding step or show a picker to update them. This allows the user to change their mind, say they want to see options signals now. (This editing could also be in a dedicated "Settings" submenu, but likely fine here.)
- *Notifications:* A submenu or toggle list: Types of push notifications they want. For example: " New Signals Alerts [On/Off]", "📊 Price Movement Alerts (for signals you follow) [On/Off]", " AI Tips and Updates [On/Off]", etc. [87] [88]. FlutterFlow can use SwitchListTile for each toggle and store preferences in Firestore (or local). We'll have those toggles set as per defaults (maybe all on by default for premium, minimal for free or as chosen in onboarding).

- *Watchlist Management:* Possibly a link “Manage Watchlist” that opens a list of tickers/signals the user is watching, allowing removal. If not, they can remove from each signal detail anyway, so this could be optional.
- *Appearance/Other:* If a dark mode toggle or language setting is relevant (likely not, we stick to dark mode only as design). Could skip.

- **Support & About:**

- Link to FAQ or Help (“**?** Help & Support”). If we have a web FAQ or if we integrate support chat/email. This could open a webview or mailto.
- “Privacy Policy” and “Terms of Service” links (as required for compliance, open external links).
- Possibly “About Us” or app version.

- **Danger Zone:**

- Logout button (“Log Out”) – simple action to sign out and return to Login screen.
- If needed, “Delete Account” which might prompt confirmation and then remove user data (if we implement).

The profile is likely accessible via a bottom nav item (e.g. an icon of a user, labeled “Profile”), or a menu icon on dashboard. We define it as its own screen accessible from main nav.

Data & Logic:
- Displaying subscription status requires checking if user is subscribed (from user profile or verifying with purchase receipts). We trust a field `isSubscribed` updated on purchase. Possibly also store subscription type and expiry for reference. If the app is not automatically validating receipts, we should have a reminder to user in case of renewal issues. (One could incorporate a restore function or a check on app start using RevenueCat or store API to update status). Out-of-scope details aside, profile should reflect the truth: if they canceled or expired, ideally it updates to show Free again and lock stuff (the logic doc mentions detecting lapse and reverting UI) [89] [90]. - The “Manage Subscription” could simply instruct: “To manage or cancel your subscription, use the App Store/Play Store settings” unless we integrate a direct link. On iOS, we can deep link to subscription management perhaps. On Android, maybe open Play Store account subscription page. Minimally, provide instructions to avoid user confusion.
- Upgrading from profile (for free users) goes through same subscription purchase process as onboarding. - Preferences toggles (notifications) would update fields in Firestore or OneSignal tags, etc. If user turns off “New Signals Alerts”, the backend/push logic should honor that. Similarly for “Token Low Balance alerts” as the logic doc example [87] (we might have a feature to alert user when tokens < say 5). We’ll include a toggle if relevant.
- When user logs out, ensure local data cleared (token balances etc. loaded fresh for next login).
- Ensure token balance shown in profile is consistent with everywhere else (should be, since we read same source).

User Story: *“As a user, I want to view and manage my account details (subscription, tokens, preferences) in one place, so I feel in control of my app experience.”*
Acceptance Criteria:
- *AC1:* The Profile screen correctly shows the user’s name and email. If the user has set a profile name, it

displays that; if not, maybe derive from email (before @). The membership status is clearly indicated: Premium users see their plan info and free users see an upgrade prompt [86] [6]. There must always be an obvious way for a free user to become premium from this screen (the “Upgrade” button or similar is present and functional).
- *AC2:* Premium users can see their token balance and have a way to purchase more tokens from here. Free users either do not see a token balance or see it disabled. For example, if a free user somehow got some free tokens (trial tokens), maybe we do show “Tokens: 20 (locked until Premium)” – but that’s an edge case. Simpler: only show token count if subscribed. If a free user taps “Upgrade to use AI tokens”, it triggers subscription flow.
- *AC3:* The user can adjust notification settings easily. Turning toggles on/off should persist (if they navigate away and back, the setting remains). If all notifications are off, the app should indeed not send them push alerts (we assume backend respects these flags). If the app uses Firebase Cloud Messaging or OneSignal, these toggles could call cloud functions to subscribe/unsubscribe from certain topics. This is beyond UI scope, but the UI must allow the user that control.
- *AC4:* If the user wants to change their initial preferences (interests, experience), they can. For instance, tapping “Interested Markets: Stocks, Crypto” might open a dialog with checkboxes or even reuse the onboarding screens for interests. Accept that changes get saved and subsequently influence the content (maybe immediately filter signals or by next login). At minimum it updates their profile in DB.
- *AC5:* All external links (FAQ, Privacy) open correctly (in a webview or external browser). The logout button indeed logs out (clears auth state, routes to login). If implementing delete account, it asks “Are you sure? This will erase all your data” and on confirm, removes their record and signs them out – but we need not get into those details unless required by compliance; it’s good to note though.
- *AC6:* The screen is properly laid out: on small devices, some items might be below the fold, but everything is reachable by scrolling. Use headings or dividers between sections to make it scannable (e.g. Account, Preferences, About). Possibly a sign that the UI kit design is applied (e.g. nice icons for each setting option). The style should match overall design (e.g. list tiles could have dark background and maybe an arrow icon for navigable items like “Help”).
- *AC7:* Real-time updates: If the user purchases a subscription or tokens in another part of the app and comes to profile, it should reflect the updated status. For example, if they were free and just subscribed via onboarding, when they open profile it should now show premium info (which likely it will, given state updated). Similarly, if they buy tokens then open profile, the balance is updated. No stale info.

User Stories and Acceptance Criteria Summary

To ensure the PRD covers crucial interactions, below are key user stories with acceptance criteria:

- **User Registration & Login:**
  Story: “As a new user, I want to easily sign up (or log in with existing account) so that I can start using the app.”
  Acceptance Criteria: The app must allow account creation with email/password (and ideally OAuth options). Login should validate credentials and navigate to the app if correct, or show errors if not. After first login, the user is taken through onboarding (if not done already). Returning users with saved sessions skip directly to content. The login screen should be simple and error-free (no crashes on wrong input). (See detailed AC in Login Screen above.)

- **Onboarding Questionnaire:**
  Story: “As a first-time user, I want to provide my trading experience and interests so the app can

personalize signals and not overwhelm me."
Acceptance Criteria: The onboarding flow must present at least two questions (experience level, interests) and record the answers. It should optionally present a subscription offer. Completion of onboarding leads to the main app. Users can navigate the onboarding easily (back/next) and cannot get stuck. Skipping the subscription step keeps them in free mode with appropriate limitations. On subsequent app launches, onboarding does not reappear (unless user profile is incomplete). The preferences collected must influence the app's behavior (e.g. filtering content).

- **Free vs. Paid Signal Gating:**
Story: "As a free user, I want to get a taste of the trading signals without paying, but I understand the best content is gated for premium users."
Acceptance Criteria: The app must clearly differentiate free vs premium content. Free users should receive a limited number of signals (e.g. a few per week) and see that more are available if they upgrade [35]. In the UI, any premium-only signal or feature should be indicated with a lock or blur. For example, in the signal list, after the free signals, further signals are not accessible [2]. In signal detail, free users see placeholders instead of actual trade levels [58] and an upgrade prompt. At all times, there should be an easy way to upgrade (CTA button). Premium users, once subscribed, should immediately see all content unlocked – no locked icons. The transition from free to paid (post-purchase) should update the UI in real-time (e.g. remove blurs, allow access). The gating should also apply to AI features: free users should not accidentally invoke AI without subscribing (or buying tokens if that was allowed). Essentially, free users can browse in a limited way, whereas paid users get full, real-time signals and AI. Both user types should remain engaged: free users get enough value to stay around (teasers, maybe educational content), while knowing what they miss out (FOMO) [56] [57].

- **Token Purchase & Usage Flow:**
Story: "As a premium user, I want to purchase and use AI tokens so that I can ask the AI questions and get additional insights."
Acceptance Criteria: The purchase flow for tokens must be smooth: selecting a pack, confirming payment, and receiving tokens without issues. The user's token balance should update immediately on purchase and be visible (dashboard/profile) [91] [25]. When the user uses the AI chat, each query deducts the correct number of tokens and this is reflected in their balance and possibly noted in chat [92] [75]. If the user runs low on tokens (below a threshold or zero), the app should prompt them to buy more – e.g. the input is disabled with a "Buy tokens" link, or even send a notification "You're out of tokens" if they try to use AI with none. Importantly, tokens cannot be used by free users unless they subscribe (or if allowed, the app must still restrict actual usage until subscription). The logic of token consumption should match the description: "everything has a cost in tokens" for AI usage [11], but **no** token is deducted for delivering regular static signals (that's subscription-based) [85]. The system should not double-charge. In summary, the acceptance is that token purchase is reliable and token spending is transparent and accurate. Users should feel in control: they know when a token is spent and how many remain.

- **In-Signal AI Chat & Auto-Updates:**
Story: "As a user viewing a signal, I want to chat with an AI to get explanations and updates, so I can better understand the trade and react to market changes in real time."
Acceptance Criteria: Within a signal's detail, the AI chat feature should be readily accessible (for premium w/ tokens). The AI should provide insightful answers specific to the signal [81] [30], not

generic advice. The AI must use up-to-date data – e.g., if the user asks something after the price moved, the AI accounts for that movement [12] [13]. If the market triggers an event (like stop-loss hit), the AI or app should notify the user in the chat or via push promptly. The auto-update logic means if significant price change or time event occurs, the signal detail reflects it (chart update, status change) and the AI can comment on it without user prompting (this might be implemented as an automated message in the chat, which the user should see within a short delay of the event). Each AI answer or update should clearly consume tokens only when it's a user-triggered query (auto AI alerts might be free or considered part of subscription). If the user prompts the AI for an update ("Give me the latest status"), that will use a token and the AI will retrieve fresh data and respond accordingly – e.g. "The stock is now $110, which is 2% below the target, momentum is still strong…" etc. The acceptance is that the AI chat feels like an **interactive analyst** that remembers the context of the signal and the conversation [15], and helps the user make informed decisions. It should not reset or give irrelevant answers mid-conversation. The system must gracefully handle errors (if AI times out or data feed is down, a friendly error is shown and no token lost). In addition, the UI should not allow free users to use the chat (they might see a teaser but not actually send messages) – ensuring only eligible users engage. When multiple inputs occur (user asks question, then follow-up), the AI maintains context across messages in that session (as long as they stay on that signal detail) [93] [94]. If user navigates away and back, it's okay if context resets (MVP-wise). Finally, any **auto-update** from AI should be clearly indicated as such and not overdone; users shouldn't be spammed. Ideally, give user control in settings if they want AI auto-commentary or not (maybe one of the notification toggles covers "AI auto updates in chat").

Supporting UX References: This PRD has been informed by prior research and UI references. The freemium gating and teaser content approach align with industry practices [35] [2] and apps like Option Signal (which limits free signals and requires subscription for full realtime access [18] [4]). The interactive AI design draws on the differentiation strategy outlined in our Signal AI considerations – leveraging real-time data and specialized training to offer insight beyond what a generic chatbot could [12] [81]. The token economy and usage model follow the blueprint of modern SaaS AI services, ensuring heavy usage is paid but also that users see clear value for each token spent [11] [71]. All user flows (login, onboarding, subscription, token purchase, signal viewing, AI chat) have been diagrammed and vetted for intuitiveness and completeness in the earlier logic flow analysis. The UI will be implemented in FlutterFlow with a focus on clean, readable layout (short paragraphs, bullet lists for stats, headings to separate sections) per the user's formatting guidelines, making the app not only powerful but also user-friendly.

This comprehensive design will guide the FlutterFlow build, screen by screen, ensuring that when the app is developed, it meets both the functional requirements and the high UX standards expected in 2025's fintech app market (clean design, fast performance, and AI-enhanced interactivity) [95] [96]. The result will be an innovative trading signals app where users can learn and act on market opportunities with the added confidence and insight provided by an integrated AI assistant.

---

[1] [2] [3] [8] [10] [35] [36] [56] [57] [58] [59] [71] [77] [78] [83] [84] [85] Trading Signals App_ Cost & Token Economy Design.pdf
file://file-HBkgXgnStexDVw3E1Yzkx1

4 17 18 Option Signal - Flow Alert on the App Store
https://apps.apple.com/sa/app/option-signal-flow-alert/id1575502688

5 6 7 9 11 14 20 21 22 23 24 25 26 27 31 32 33 34 75 76 82 86 87 88 89 90 91 92 93 94
UI Logic Flow for AI-Driven Trading Signals App.pdf
file://file-D67PGGaGRRttZCXPE8ywNe

12 13 15 16 30 66 67 68 69 70 72 73 74 79 80 81 Signal AI Design Considerations and Differentiation.pdf
file://file-YKHHHoTXBskxgPR6Y7EV9n

19 28 29 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 60 61 62 63 64 65
trading-opportunity-viz v13.html
file://file-FC2BY6V8trJDAKcWqCMDND

95 96 Optimizing FlutterFlow App- Guide for the U.S. App Market for 2025
https://www.flutterflowdevs.com/blog/optimizing-flutterflow-app--guide-for-the-u-s-app-market-for-2025