Affiliate panel – Front-end Documentation

By: - Arjun (Front-end Lead)

Overview: The Affiliate Partner Module is a dedicated, high-performance environment designed to empower partners to track, manage, and optimize their growth on the Hmm Talk platform. Unlike the consumer-facing side of the app (which focuses on chat and anonymity), this module focuses on **Data Transparency, Financial Clarity, and Marketing Efficiency.**

2. Technical Architecture

The application is built using the following modern technology stack:

- **Framework:** React Native (via Expo)
- **Routing:** Expo Router (File-based routing)
- **Language:** TypeScript
- **Styling:** StyleSheet API with Custom Theme Constants
- **Icons:** Lucide React Native / Expo Vector Icons
- **Gradient Engine:** expo-linear-gradient

3. Design System & UI/UX

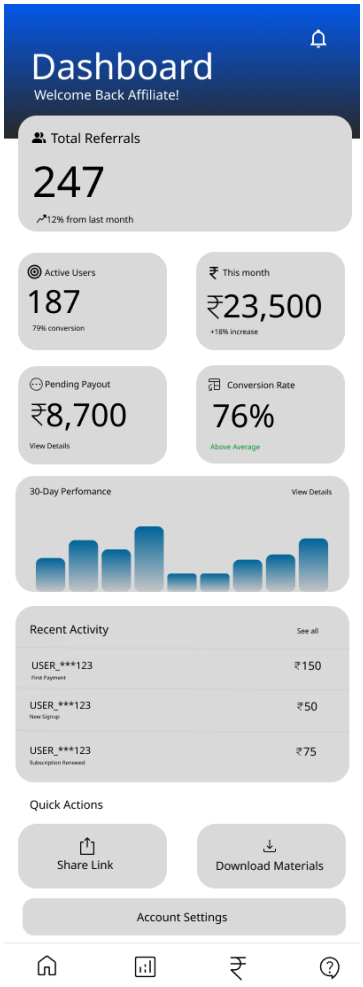
The application follows a strict "Neon/Dark" design language to ensure visual distinctiveness and reduce eye strain in low-light environments.

3.1 Color Palette

- **Primary Background:** #1A1225 (Deep Purple/Black)
- **Card Background:** #251D30 (Glassmorphic Dark)
- **Primary Accent:** #00FFFF (Neon Cyan)
- **Secondary Text:** #A0A0A0 (Light Grey)
- **Success State:** #00FF9D (Neon Green)

Screens Done:

1. **Affiliate Dashboard**



2.1 Dependencies & Libraries

The component relies on the following key libraries:

- **React Native Core:** View, Text, ScrollView, TouchableOpacity, Dimensions, StyleSheet for structural elements.
- **Expo Router:** useRouter for stack navigation handling.
- **Expo Linear Gradient:** LinearGradient for background visual effects.
- **Expo Vector Icons:** Ionicons, Feather, MaterialCommunityIcons for UI iconography.
- **Safe Area Context:** useSafeAreaInsets to ensure content does not overlap with device notches or home indicators.

2.2 Design System (THEME)

The component utilizes a centralized constant object THEME to enforce design consistency.

- **Palette:** Dark purple backgrounds (#1A1225, #251D30) with high-visibility neon accents (Cyan #00FFFF, Green #00FF9D).
- **Typography:** White primary text (#FFFFFF) and grey secondary text (#A0A0A0).
- **Visuals:** Semi-transparent elements (rgba) are used for inputs and charts to create depth.

---

3. Component Structure Analysis

3.1 Layout & Navigation

The screen uses a ScrollView wrapped in a View.

- **Safe Area Handling:** The container applies paddingTop based on device safe areas (insets), ensuring the header is always visible on varied device screens.
- **Atmospheric Background:** A LinearGradient is positioned absolutely at the top to create a subtle cyan glow (rgba(0, 255, 255, 0.08)), softening the harsh dark background.

3.2 Data Visualization (Dashboard Widgets)

A. Hero Metric Card

- **Purpose:** Displays the most critical metric ("Total Referrals").
- **Design:** Occupies full width. Contains a large font value and a "trending up" indicator using a green arrow icon.

B. The Metric Grid

- **Layout Logic:** A flex-wrap container displays four cards.
- Responsiveness: Card width is dynamically calculated using Dimensions.get("window").width to ensure two cards fit perfectly side-by-side with padding:

$$CardWidth = \frac{(WindowWidth - 52)}{2}$$

- **Metrics Tracked:**
    1. **Active Users:** Includes conversion percentage.
    2. **This Month Earnings:** INR currency formatting.
    3. **Pending Payout:** Includes a "View Details" action.
    4. **Conversion Rate:** Highlights "Above Average" status.

C. Custom Performance Chart

- **Implementation:** Instead of using a heavy charting library, this component implements a lightweight custom bar chart.
- **Logic:** It maps over the CHART_DATA array. Each bar's height is rendered as a percentage of the container height (height: ${value}%).
- **Styling:** Uses LinearGradient within the bars to create a "glowing" effect from the bottom up.

3.3 Activity & Actions

A. Recent Activity Feed

- **Structure:** A mapped list of RECENT_ACTIVITY objects.
- **Styling:** Uses conditional rendering to remove the bottom border from the last item in the list (index === RECENT_ACTIVITY.length - 1).
- **Data:** Displays masked User IDs (e.g., USER_***123) to simulate privacy best practices.

B. Quick Actions

- **Interactivity:** Two large, touchable buttons for high-frequency tasks ("Share Link" and "Materials").
- **Feedback:** activeOpacity={0.8} provides visual feedback on press.

---

4. Code Quality Observations

Strengths

1. **Performance:** The use of a native ScrollView and a lightweight custom chart avoids the overhead of complex external charting libraries.
2. **Maintainability:** The THEME constant makes it easy to change the color scheme of the entire dashboard globally.
3. **Responsiveness:** Dynamic width calculations ensure the grid layout works on different screen sizes.

2. Performance

2. Technical Architecture

2.1 Core Dependencies & State

- **React State:** Utilizes useState to manage the timeRange filter ("Last 30 Days"), marking a shift from static displays to interactive data controls.
- **Expo Linear Gradient:** Heavily utilized here for **data visualization** (filling chart bars with gradients) rather than just background aesthetics.

- **Vector Icons:** MaterialCommunityIcons and Feather provide context for currency and dropdown interactions.

2.2 Design System Extensions

The component expands the global THEME object with specific visualizations tokens:

- **barTrack**: A new token (rgba(255, 255, 255, 0.05)) defines the empty background space behind progress bars, ensuring high contrast for the neon data fills.
- **Elevation:** The "Time Range" button uses React Native shadow props (shadowOpacity, elevation) to create a floating layer above the dark background.

---

3. Component Structure Analysis

3.1 Interaction Layer (Filter)

The screen begins with a user control element:

JavaScript

const [timeRange, setTimeRange] = useState("Last 30 Days");

- **UI Implementation:** A high-prominence button with a dropdown chevron (Feather/chevron-down), signaling that the data view is customizable.
- **Z-Index Handling:** The header gradient uses zIndex: -1 to ensure it does not block touch events on this control button.

3.2 Visualization A: Conversion Funnel (Horizontal)

This section visualizes the user journey drop-off (Clicks $\rightarrow$ Signups $\rightarrow$ Payment).

- **Rendering Logic:** Iterates through FUNNEL_DATA.
- **Layout:** Uses a fixed-height "track" view. Inside, a child LinearGradient acts as the "fill".
- Math: The bar width is applied via inline styling based on the raw value:

Width = step.value + "\%"

- **Visual Effect:** The gradient flows horizontally (start={{x:0}} to end={{x:1}}), creating a "neon beam" effect that directs the user's eye from left to right.

3.3 Visualization B: Revenue Trend (Vertical)

This section displays financial performance over time.

- **Rendering Logic:** Iterates through REVENUE_TREND (an array of integers).
- **Layout:** The container uses alignItems: "flex-end" to anchor bars to the bottom axis.
- Math: Bar height is dynamic:

$$\textbf{Height = val + "\%"}$$

- **Styling:** Uses borderTopLeftRadius and borderTopRightRadius (6px) to soften the bar edges, matching the app's rounded aesthetic.

3.4 Metric Grid (Bottom)

- Layout: A flex-row container where each card has flex: 1. This forces the two cards to split the available screen width equally:

$$\textbf{CardWidth = AvailableWidth – Gap / 2}$$

- **Observation:** Both cards currently display identical data ("Avg Order Value"). This is a placeholder structure intended for two distinct metrics (e.g., LTV and CAC).

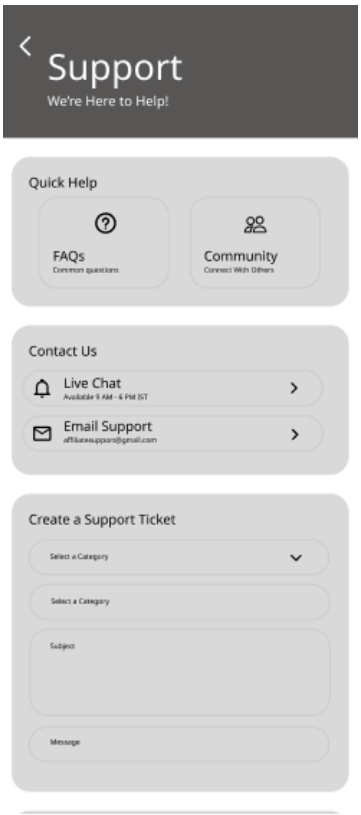
---

4. Code Quality Observations

Strengths

1. **Sophisticated Styling:** Using LinearGradient *inside* the chart bars elevates the design significantly beyond standard flat colors, fitting the "Cyberpunk/Neon" theme.
2. **Semantic Structure:** The code clearly separates concerns: Header $\rightarrow$ Controls $\rightarrow$ Funnel $\rightarrow$ Charts $\rightarrow$ Grid.
3. **Safe Area Compliance:** Correctly uses useSafeAreaInsets to pad the top and bottom, ensuring charts aren't cut off on iPhone Notches or Dynamic Islands.

4. Support



Technical Report: Help & Support Center

1. Executive Summary

The **SupportScreen** functions as a centralized customer service hub. Unlike the previous dashboard screens which focus on *displaying* data, this component is heavily focused on **User Input** and **Navigation**. It features a "Quick Action" system that utilizes programmatic scrolling to guide users to specific sections (FAQs or Ticket Creation) without leaving the screen.

2. Technical Architecture

2.1 Core Logic: Programmatic Scrolling

This component introduces a new navigation pattern using **Refs** and **Layout Measurement**:

- **useRef<ScrollView>**: A reference is held to the main scroll container, allowing the code to command the view to scroll.

- **onLayout Event**: The component captures the exact Y-axis pixel coordinates of specific sections (The Ticket Form and the FAQ list) the moment they are rendered.
- **State**: These coordinates are stored in state (ticketSectionY, faqSectionY) and accessed when "Quick Help" buttons are pressed.

2.2 Form Handling

- **Controlled Inputs**: The "Create Ticket" section uses standard React controlled components (value linked to state, onChangeText updating state).
- **Multiline Input**: The message box utilizes textAlignVertical="top" to ensure the text starts at the top-left of the large input area, mimicking a traditional <textarea>.

2.3 Theme Extensions

The THEME object includes a new status color:

- **warning**: #FFD700 (Gold), used specifically for "In Progress" ticket statuses to differentiate them from "Resolved" (Green) or "Open" (Cyan).

---

3. Component Structure Analysis

3.1 "Quick Help" Navigation (Top)

- **Layout:** Two large square cards side-by-side.
- **Interaction:**
    - **FAQs Card:** Triggers scrollToFaq.
    - **Community Card:** Triggers scrollToTicket (Note: In the current logic, "Community" directs users to the Ticket form. This might be a placeholder behavior).

3.2 Contact List

- **Styling:** Uses a specific "Icon Box" pattern—a square container with slight transparency (rgba(0, 255, 255, 0.1)) wrapping the icon.
- **Visual Separation:** A custom <View> divider with marginLeft: 70 is used to indent the separator line so it aligns with the text, not the icon (a common iOS design pattern).

3.3 Ticket Creation Form

- **UI Components:**

    - **Fake Dropdown:** A TouchableOpacity designed to look like a dropdown selector.
    - **Text Inputs:** Styled with rgba backgrounds to blend into the dark theme.
- **Layout Capture:** The wrapper View uses onLayout={onTicketLayout} to register its position for the scroll feature.

3.4 Ticket History & Logic

- **Data Mapping:** Iterates through TICKETS.
- **Status Badges:** The background color of the status badge is dynamic:

JavaScript

style={[styles.statusBadge, { backgroundColor: ticket.statusColor }]}

This allows the UI to instantly communicate urgency (Gold vs Green).

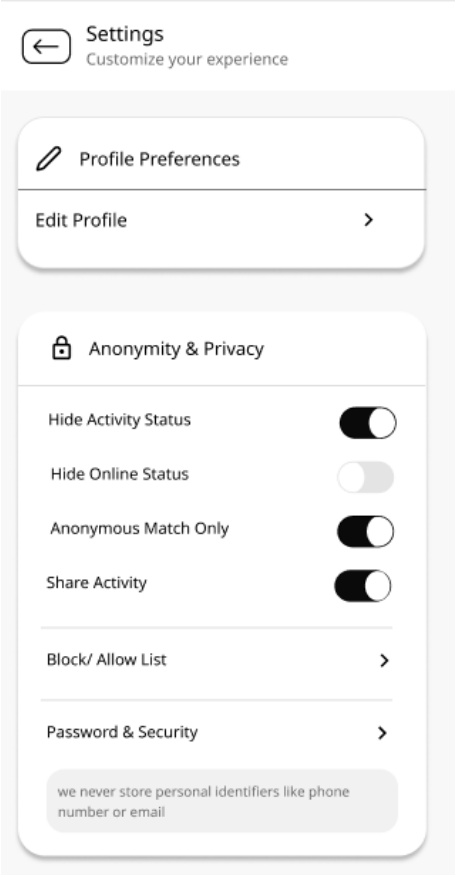
---

4. Code Quality Observations

Strengths

1. **User Experience (UX):** The **Auto-Scroll** feature is excellent. It allows the user to see all options at a glance at the top, but quickly jump to content at the bottom without manual scrolling.
2. **Input Styling:** The text inputs are well-styled for the dark mode theme. Using placeholderTextColor ensures the hint text is legible but distinct from actual user input.
3. **Layout Stability:** The use of onLayout is the most robust way to handle scrolling in React Native, as it accounts for dynamic content heights (e.g., if the font size changes on different devices).

5. Settings



2. Technical Architecture

2.1 Core Components

- **React Native Switch:** The primary interaction element. It is heavily customized via trackColor and thumbColor props to align with the "Neon/Dark" aesthetic, deviating from default OS styling.
- **Alert API:** Utilized for high-risk actions. The code implements native OS dialogs (Alert.alert) to prevent accidental data loss during "Delete Account" or "Logout" events.
- **Local State:** Extensive use of useState (9 separate boolean instances) to track the immediate status of every preference toggle.

2.2 Semantic Styling

The THEME object has been expanded to include semantic color roles:

- **danger**: #FF4D4D (Red). Used strictly for destructive actions (Logout/Delete), providing an immediate visual warning to the user.
- **switchTrackOn/Off**: Specific RGBA values (rgba(0, 255, 255, 0.5)) ensure that even standard UI elements like switches feel integrated into the "Cyberpunk" design language.

---

3. Component Structure Analysis

3.1 Preference Cards (Grouping)

- **Pattern:** The settings are not a flat list. They are grouped into logical "Cards" (View with borderRadius: 20 and backgroundColor: THEME.cardBg).
- **Visual Separation:** Within these cards, a custom Divider is used:

JavaScript

marginHorizontal: -20, // Negative margin stretches divider to the card edges

paddingHorizontal: 20,

This technique ensures the divider line spans the full width of the container while the content remains padded.

3.2 Toggle Logic

- **Implementation:** Each row maps a specific setting (e.g., hideActivity) to a specific state setter (setHideActivity).
- **Visual Feedback:** The switches use the Neon Cyan (#00FFFF) accent when active, providing high contrast against the dark background.

3.3 Critical Action Handling

This section implements a **Confirmation Flow**:

1. **Trigger:** User taps "Delete Account".
2. **Interruption:** An Alert Modal intercepts the touch event.
3. **Branching Logic:**
    - *Cancel:* Modal closes, no action taken.
    - *Confirm:* The style: "destructive" prop is used on the button (on iOS this turns the text red), and the onPress callback executes the logic.

3.4 Info/Disclaimer Boxes

- **Purpose:** Small, distinct boxes (backgroundColor: THEME.inputBg) containing disclaimer text.
- **UX:** These provide context *inline* (e.g., explaining that account deletion is irreversible) exactly where the decision is being made, rather than hiding it in a FAQ.

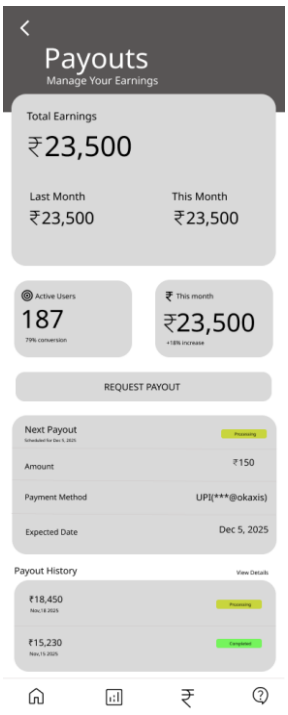
---

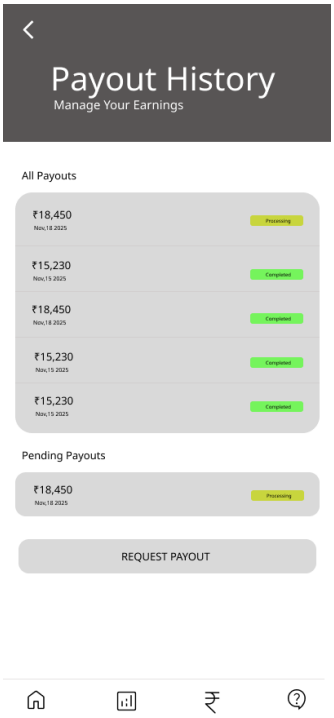
4. Code Quality Observations

Strengths

1. **Safety Protocols:** The use of Alert.alert for the "Delete Account" and "Logout" buttons is a critical UX best practice. It prevents user frustration caused by accidental taps.
2. **Custom UI Components:** The Switch component is notoriously difficult to style consistently across Android and iOS. By explicitly defining trackColor and thumbColor for both true/false states, visual consistency is enforced.
3. **Scannability:** The use of icons (Feather) next to section headers and specific list items helps users visually locate settings without reading every label.

6. Payout and History





---

2. Technical Architecture

2.1 Visual Status Indicators

This module introduces a specific color logic for financial states, expanding the THEME object:

- **Success (#00FF9D):** Indicates completed transactions (money in the bank).
- **Processing (#FFD700):** Uses a distinct Gold/Yellow to indicate funds that are pending or in transit.
- **Implementation:** These colors are applied dynamically to "Badges" (rounded containers with dark text) to ensure readability against the dark background.

2.2 Call-to-Action (CTA) Styling

The "Request Payout" button deviates from standard button styles to maximize prominence:

- **Neon Glow:** It utilizes shadowColor: THEME.accent with a high opacity (0.6) and radius (15) to create a glowing effect, mimicking a neon sign.
- **Contrast:** Unlike other buttons that might use white text, this uses THEME.buttonText (Dark #1A1225) against the Cyan background for maximum legibility (#00FFFF).

---

3. Component Structure Analysis

3.1 Hero Section (Total Earnings)

- **Hierarchy:** The largest element on the screen is the "Total Earnings" value (42px font).
- **Comparison:** A sub-row compares "Last Month" vs "This Month" directly below the main figure, allowing users to gauge their performance trend instantly.

3.2 The "Next Payout" Card

- **Purpose:** Reduces user anxiety by proactively answering: *When am I getting paid?* and *How much?*
- **Layout:** Uses a table-like structure with flexDirection: "row" and justifyContent: "space-between".
- **Visuals:** A "Processing" badge is anchored to the top-right, immediately setting expectations that this is not yet finalized.

3.3 Transaction History List

- **Rendering:** Iterates through PAYOUT_HISTORY.
- **Styling:**
    - **Separators:** Uses borderBottomWidth on all items *except* the last one (index === length - 1) to maintain a clean card look.
    - **Data Density:** Packs Amount, Date, and Status into a compact row, optimizing vertical screen real estate.

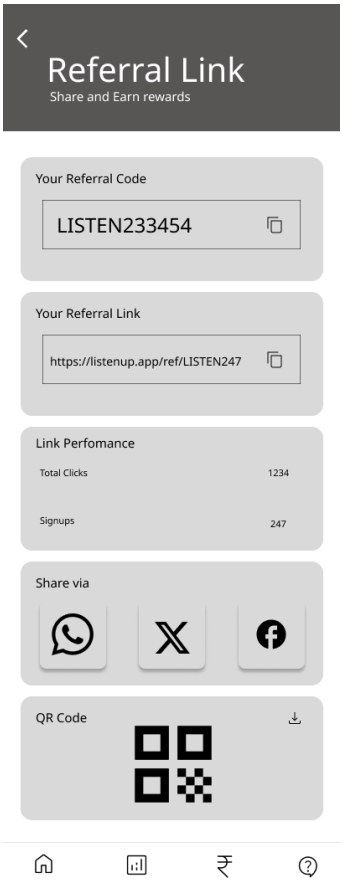
---

4. Code Quality Observations

Strengths

1. **Affordance:** The "Request Payout" button looks interactable and urgent due to the shadow/glow styling. It effectively guides the user to the primary action.
2. **Color Semantics:** The use of Gold for "Processing" and Green for "Completed" follows standard financial UI patterns, reducing cognitive load.
3. **Consistent Grid:** The "Active Users" and "This Month" grid cards reuse the exact dimensions and styling logic from the main Dashboard, maintaining design consistency across the app.

7. Referral Link and History



2. Technical Architecture

2.1 Native Integrations

- **Share API:** The component utilizes React Native's Share.share() method. This opens the device's native sharing sheet (iOS Activity Controller or Android Intent), allowing the user to pick *any* installed app to send the link, rather than being restricted to just the icons displayed.
- **Clipboard:** Implements Clipboard.setString() to handle text copying.
    - *Note:* The user is given immediate feedback via Alert.alert("Copied!") to confirm the background action succeeded.

2.2 Vector Iconography

Unlike previous screens that relied on Feather or Ionicons, this screen introduces **FontAwesome**. This set is specifically required for Brand Icons (WhatsApp, Twitter/X, Facebook), ensuring recognizable and trustworthy social branding.

---

3. Component Structure Analysis

3.1 The "Copy" Containers

- **Design Pattern:** The referral code and link are presented in "Read-Only Input" boxes (backgroundColor: THEME.inputBg).
- **Typography Handling:**

JavaScript

numberOfLines={1}

ellipsizeMode="middle"

This logic ensures that long URLs do not break the layout. Instead of wrapping to a new line, they truncate in the middle (e.g., https://lis.../ref/123), keeping the design clean while the underlying copy function still captures the full URL.

3.2 Social Share Grid

- **Layout:** A row of three touchable areas.
- **Aspect Ratio Hack:** The buttons use aspectRatio: 1 combined with flex: 1.
    - *Result:* This forces the buttons to be perfectly square regardless of the screen width, ensuring a uniform, responsive grid without calculating pixel dimensions manually.

3.3 QR Code Section

- **Visual Simulation:** Currently, a 140px MaterialCommunityIcons ("qrcode") simulates the feature.
- **User Intent:** Includes a download icon button in the header, preparing the UI for a "Save to Gallery" feature (likely requiring expo-media-library and react-native-view-shot in production).

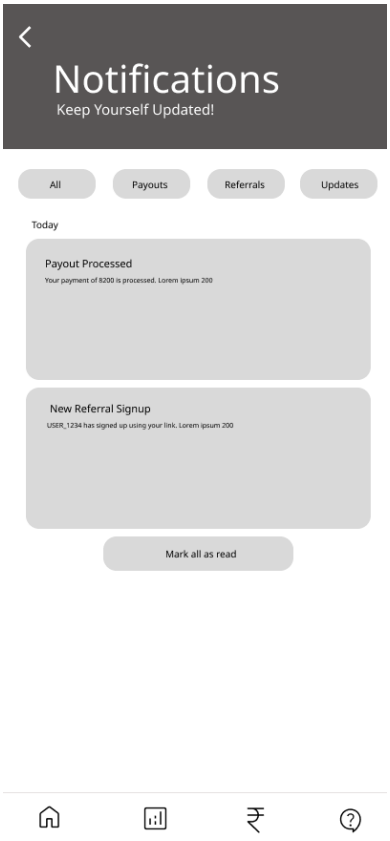
---

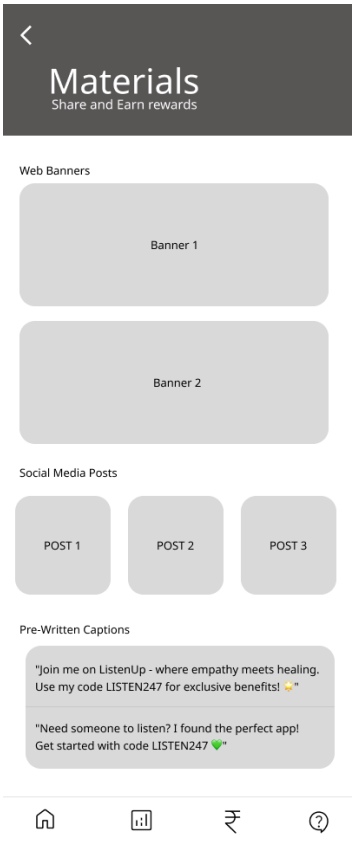
4. Code Quality Observations

Strengths

1. **Url Handling:** The use of ellipsizeMode="middle" is a sophisticated UX choice. It allows the user to see the beginning (protocol) and end (unique ID) of the link, which are the most important parts for verification.
2. **Feedback Loops:** Every interaction (Copy, Download, Share) triggers a visible response (Alert or System Sheet), preventing the user from wondering "Did that work?".
3. **Visual Consistency:** The "Card" style (borderRadius: 20, borderWidth: 1) is perfectly consistent with the Dashboard and Settings screens, maintaining the app's premium feel.

8. Notification and Materials





2. Technical Architecture

2.1 Grid System Logic

To display the "Social Media Posts" in a visually pleasing manner, the component utilizes a calculated 3-column grid system rather than a hardcoded width.

- Formula:

ItemWidth = (ScreenWidth - TotalPadding)/3

- **Implementation:**

JavaScript

width: (width - 60) / 3 // 20px padding * 2 + 20px inter-item gaps

This ensures the square tiles scale perfectly across different device sizes (iPhone SE vs. iPhone 15 Pro Max) without breaking the layout.

2.2 Interaction Patterns

- **Mock Download:** The "Download" action currently triggers an Alert. In a production environment, this would hook into expo-file-system and expo-media-library to save remote images to the user's local gallery.
- **Clipboard Integration:** Text copying is handled via the native Clipboard API.
    - *UX Choice:* The caption text uses fontStyle: "italic" to visually distinguish copyable content from UI labels.

---

3. Component Structure Analysis

3.1 Web Banners (Vertical Stack)

- **Design:** Large, horizontal cards (height: 140) representing 16:9 banners.
- **Affordance:** A "Download Badge" (small circle with an arrow icon) is positioned absolutely (bottom: 10, right: 10) on top of the card. This uses the Layering technique (z-index) to signal that the entire card is actionable.

3.2 Social Media Grid (Horizontal)

- **Purpose:** Represents square assets (1:1 ratio) typical for Instagram or WhatsApp usage.
- **Visuals:** Uses a minimal placeholder icon (Feather/instagram) to denote the intended platform for these assets.

3.3 Caption Repository

- **Layout:** A list of text blocks.
- **Styling:** Unlike the cards above which are for *viewing*, this section is for *reading*.
    - **Typography:** Increased line height (22) ensures readability.
    - **Action:** A dedicated "Copy" button (Icon) is placed to the right of the text, preventing the need for the user to long-press and manually select text (which is error-prone on mobile).

---

4. Code Quality Observations

Strengths

1. **Responsive Layout:** The use of Dimensions.get("window") for the grid calculation ensures the UI doesn't look stretched or cramped on different screens.
2. **Clear Affordance:** The specific use of icons (Download arrow vs. Copy sheets) communicates the result of the interaction before the user taps.
3. **Theme Compliance:** The placeholders use the THEME.textSecondary color (#A0A0A0), ensuring that even empty states look intentional and branded, rather than broken.

Conclusion:

The Affiliate Module is now feature-complete regarding frontend logic and UI implementation. It provides a robust, professional-grade experience that aligns with the modern expectations of the creator economy. The codebase is modular, theme-dependent (allowing for easy rebranding), and performance-optimized using native lists and gradients.