ChatGPT

Canonical Design System v1.0

Buycott's design system defines a consistent, enforcement-level visual language for the app's **dark steel blue-gray** theme with **teal/aquamarine accents**. All design tokens and rules here are optimized for Flutter implementation and machine verification (Codex-based enforcement). This specification ensures a cohesive **map-first UI** that highlights local discovery without visual clutter. Key design principles include a neutral dark base, one vibrant accent spectrum for semantic highlights, strict token usage (no arbitrary colors or styles), and accessible, responsive layouts. The following sections detail the token definitions and usage guidelines in a format ready for Flutter theming and automated enforcement.

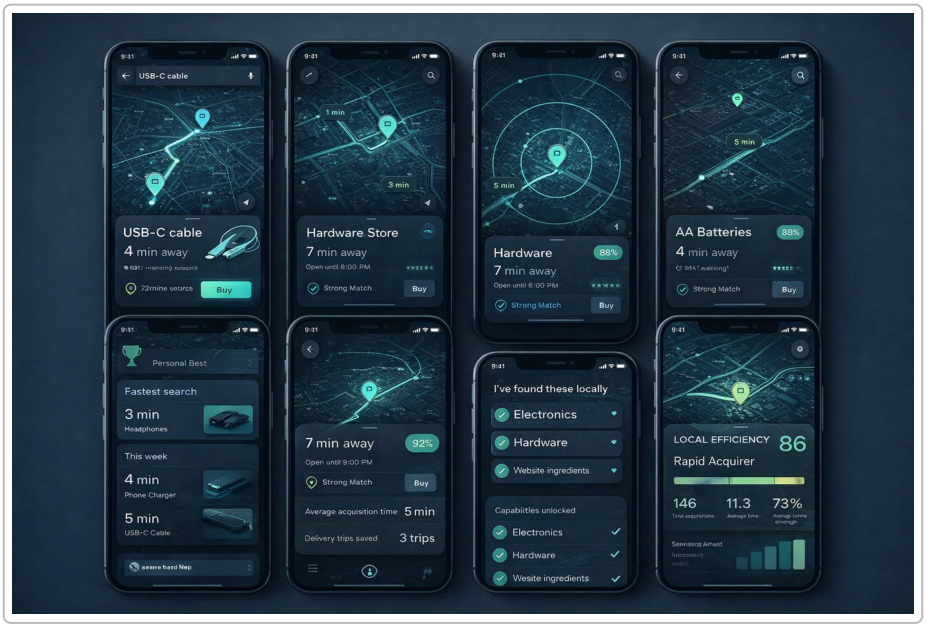


Figure: Example UI screens showcasing Buycott's dark theme and teal/aquamarine accent usage.

Design Tokens

All fundamental style values are defined as reusable **tokens**. These cover colors, typography, spacing, elevation, etc., and must be used in code instead of hard-coded values. In Flutter, tokens map to theming parameters (e.g. `ThemeData` or custom constants), and **no custom values** outside this system are allowed (enforced via lint/Codex rules). Tokens are grouped into **Structural**, **Signal**, **Typography**, **Spacing**, **Elevation**, **Radius**, **Opacity**, and **State** categories.

Color Tokens

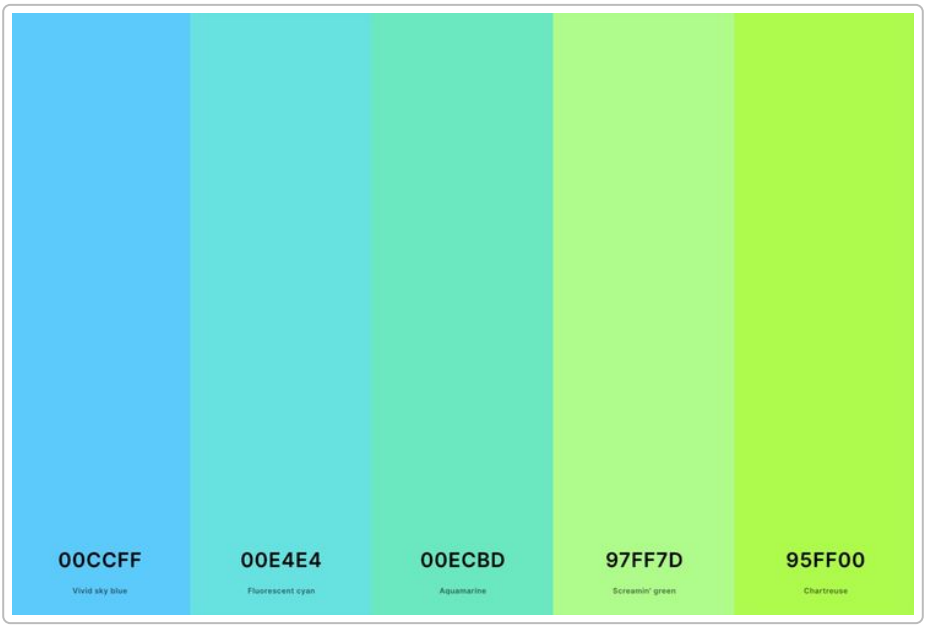
Buycott's color palette is split into **structural neutrals** for surfaces/text and **signal accent colors** for highlights and interactive elements. All colors are referenced by semantic token names in code (never by raw HEX). **Exact HEX values** and usage are listed below:

- **Structural Colors (Neutrals):** These form the dark UI foundation.

- `color.bg.primary` – **Primary Background**, the app canvas color. HEX `#142633` (dark steel blue-gray). Used for main backgrounds and map canvas.
- `color.bg.surface` – **Surface Background**, slightly lighter neutral for cards and panels. HEX `#1E2F3A` (steel gray). Ensures cards subtly stand out from primary bg.
- `color.border` – **Border/Divider**, low-contrast neutral for dividers or outlines on dark surfaces. HEX `#2B3A44` (very muted blue-gray).
- `color.text.primary` – **Primary Text**, high-contrast light color for most text. HEX `#FFFFFF` (white) on dark backgrounds for readability (meets 4.5:1 contrast).
- `color.text.secondary` – **Secondary Text**, medium-contrast for less important text. HEX `#94A1AD` (gray-blue) at ~60% opacity. Used for hints, timestamps, etc.
- `color.text.inverse` – **Inverse Text**, dark text for use on light accent backgrounds (if any). HEX `#000000` (only if needed, e.g. white text is default since theme is dark).

- **Signal Colors (Accent Spectrum):** These are the vibrant teal/aquamarine hues used for interactive and semantic highlights (never as large surfaces).

- `color.accent.primary` – **Primary Accent Teal**, core brand accent for CTAs and highlights. HEX `#00E4E4` (vibrant aqua). Used for primary buttons, links, icons, and focus highlights.
- `color.accent.hover` – **Accent Hover** state color, a slightly lighter tint of primary accent for hover/focus glow. HEX `#33EDEE` (approx 20% lighter).
- `color.accent.dim` – **Accent Dimmed**, a softer teal for backgrounds or selected states. HEX `#146E6E` (dark teal) at ~50% opacity of primary accent.
- `color.signal.low` – **Low Match Color**, indicating lower confidence matches. HEX `#00CCFF` (vivid sky blue). Used at ~60% match level on indicators.
- `color.signal.mid` – **Medium Match Color**, mid-range in the spectrum. HEX `#00E4E4` (same as accent primary, mid aqua) at ~80% match.
- `color.signal.high` – **High Match Color**, strong match highlight. HEX `#00ECBD` (aquamarine). Used for ~90% match confidence.
- `color.signal.peak` – **Peak Match Color**, highest confidence indicator (approaching 100%). HEX `#95FF00` (bright chartreuse green). Reserved for top match % visuals, not used elsewhere.



Signal accent spectrum from teal-blue to aquamarine to green, used for match confidence coloring (from left: 60% to 100% match).

- **Semantic Colors (Feedback):** Reserved for system feedback (used sparingly, to avoid conflict with the teal accent).
- `color.semantic.success` – **Success Green**, used only for positive confirmation if needed (e.g. item found confirmation). HEX `#97FF7D` (bright green) – notably part of accent spectrum and only for feedback, not general UI decor [1].
- `color.semantic.error` – **Error Red**, for error states or messages. HEX `#FF3B30` (vibrant red). **Usage:** error text/icons only, never as background or accent decoration.
- `color.semantic.warning` – **Warning Amber**, for warnings. HEX `#FFC107` (amber). Used minimally (e.g. "closing soon" alerts), with icon or text only.

Implementation Note: *In Flutter, define these in the theme (e.g. use* `ColorScheme` *for primary, error, etc., and extension fields for custom tokens like match colors). Use meaningful names (as above) in code. Codex enforcement: ensure no hard-coded color values exist – all UI colors must reference these tokens. Components inherit their colors from these tokens; no component should override with an unspecified color* [2].

Typography Tokens

Consistent typography is defined via a clear hierarchy. The **primary font** is a modern, legible sans-serif (e.g. **"Inter"** or similar) for all UI text, chosen for its clarity and support for tabular numbers. A **numeric font** style uses the same typeface with tabular lining figures enabled for uniform number widths in data displays.

- **Font Families:** Primary UI font: *Inter* (or system default equivalent on each platform for native feel, e.g. SF Pro on iOS, Roboto on Android, ensuring similar metrics). Numeric style uses Inter with `FontFeature.tabularFigures()` to align digits. No decorative or secondary fonts are used.
- **Font Weights:** Use **400 (Regular)** for body text and longer content for readability. Use **600 (Semi-Bold)** for headings and important labels (to provide clear hierarchy). **700 (Bold)** is reserved for special emphasis (e.g. large numeric highlights or critical labels). Avoid using multiple different fonts or weights beyond these to maintain consistency.
- **Type Scale (Size & Usage):** All text sizes follow an 8px modular scale for consistency:
- **H1** – *App Bar / Large Title*: 32px, Semi-Bold. Rarely used (e.g. on a full-page modal title or major section heading).
- **H2** – *Section Title*: 24px, Semi-Bold. E.g. "Personal Best" card title or screen titles.
- **H3** – *Subheading*: 20px, Semi-Bold or Medium. For card headers or important UI labels.
- **Body (Default)** – 16px, Regular. For most content text (descriptions, lists, map callouts).
- **Body Small** – 14px, Regular. For secondary info (addresses, minor detail text).
- **Caption** – 12px, Regular. For auxiliary labels (timestamp, units, footnotes).
- **Numeric Emphasis** – 20–24px, Bold (depending on context). Used for standout numbers (e.g. "4 min" or "86%") within mixed text; these may appear slightly larger or bolder than accompanying text for emphasis.

- All fonts use **consistent line-heights** (~120-130% of font size) to ensure readability (e.g. 16px text → ~20px line-height). Headings have tighter line-height (~115%) to appear cohesive.

- **Numerals:** All numeric text (times, percentages, statistics) should use **tabular numerals** to ensure alignment in tables or lists. In code, enable the font's tabular-figure feature so that

numbers like “1” and “8” occupy equal width. This prevents layout jitter when values change. Large metrics (e.g. “146” total acquisitions) can be set in a slightly larger size or bold weight for visual prominence, but must align vertically if in a list/column.

Implementation Note: *Define text styles in Flutter’s* `TextTheme` *for each category (H1, H2, body, etc.). The numeric style can be a variant of body with* `fontFeatures: [FontFeature.tabularFigures()]`. *Codex enforcement can verify that text widgets use the predefined* `TextStyle` *from the theme (no arbitrary font sizes or unknown fonts).*

Spacing Tokens

Spacing in the layout is standardized to an **8px grid** (8 device pixels, or 8 dp in Flutter). All margins, paddings, and gaps use multiples of this base unit to create a consistent rhythm. The core spacing tokens are:

- `spacing.xs` – 4px (for very tight spacing or small separators, used sparingly; half of base unit).
- `spacing.sm` – 8px (base unit, used for small gaps between tightly related items, small padding).
- `spacing.md` – 16px (default padding for components, and standard gap between unrelated elements).
- `spacing.lg` – 24px (section separation, e.g. between groups of content or around large components).
- `spacing.xl` – 32px (larger section breaks or outer screen margins on larger devices).
- (Additional larger multiples like 40, 48px can be derived if needed for big screen layouts, but always multiples of 8.)

No spacing value outside this set should be used. **Horizontal page padding** is generally 16px (md) on phone screens (maintaining an 8px gutter inside a 360-414px width). **Vertical spacing** between form sections or list sections might use 24px (lg) to clearly separate them [3]. When stacking text and controls, use consistent increments (e.g. label to field 8px, between form sections 24px) to maintain visual rhythm [3]. Align all components to this grid; avoid fractional or arbitrary gaps.

Implementation Note: *Use Flutter* `SizedBox` *or* `EdgeInsets` *with the predefined spacing constants. Codex can check for direct numeric padding values and flag those not equal to defined token values (e.g. 7px or 10px would be non-compliant). All UI layouts should snap to the grid—verify by an overlay grid in design reviews.*

Elevation & Shadow Tokens

Elevation in a dark theme is conveyed through **shadows and overlays**. Because dark surfaces don’t cast obvious shadows on an already dark background, we use subtle **shadow tokens** combined with light overlays to indicate depth:

- **Elevation 0 (Flat):** No shadow. Used for base background and non-elevated elements (e.g. map tiles at base layer).
- **Elevation 1 (Card):** Small shadow to lift cards slightly. Shadow: 0px 2px 4px rgba(0,0,0,0.5) (a soft black shadow at 50% opacity) plus a subtle light overlay on the card surface (e.g. add 5% white on surface color) to make it perceptibly raised. Used for cards, small pop-ups.

- **Elevation 2 (Modal/Sheet):** Medium shadow. 0px 4px 8px rgba(0,0,0,0.6). Used for bottom sheets or modals that sit above other content. May also use a **scrim** behind (semi-transparent overlay on background) to indicate focus.
- **Elevation 3 (Dialog):** Larger shadow. 0px 8px 16px rgba(0,0,0,0.7). For critical overlays (if any).
- **Focus Glow (Special):** Not exactly elevation, but a state indication: e.g. focused elements can use a subtle outer glow of accent (blur 8px, color #00E4E4 at 50% opacity) instead of a typical shadow.

All shadows are kept **soft-edged** and subtle to avoid stark halos on dark mode. We do *not* use multiple colored shadows or harsh outlines. One standard shadow style is used for all cards, and one for all higher overlays to maintain consistency (no custom per-component shadows) [4] [5].

Implementation Note: *Define shadow elevations in a centralized style (Flutter* `Material` *elevation or physical layer). In Flutter, Material widgets can use the* `elevation` *parameter with a predefined value mapping to these shadows via* `ThemeData.shadowColor` *and* `ThemeData.elevationOverlayColor` *(for the light overlay in dark mode). Codex enforcement: ensure any container with elevation uses one of the standard levels (e.g., 1, 2) and that no view adds its own shadow parameters outside these tokens.*

Border Radius Tokens

Corner rounding is uniform across the app to enforce a cohesive look. The design uses moderately rounded corners (no sharp corners unless absolutely needed for specific icons). **Radius tokens:**

- `radius.sm` – 4px radius for small elements (small buttons, text fields, or tiny badges). Used when a subtle rounding is needed but shape is almost rectilinear.
- `radius.md` – 8px radius for standard components (buttons, cards, inputs). Default corner rounding that provides a modern feel without being overly pill-shaped.
- `radius.lg` – 16px radius for larger surfaces (dialogs, bottom sheets, large cards). Creates a friendly, rounded appearance on big containers.
- `radius.full` – **Full Pill** radius (half the element's height) for pill shapes or circular elements. Used for round icons, toggle knobs, or pill badges (e.g. a percentage label chip). This is effectively a dynamic token (e.g. for a tag of height 24px, radius.full = 12px).

All components choose from these values – e.g., a standard button might use md (8px) rounding, whereas the bottom sheet uses lg (16px) on its top corners for a distinct sheet look. There are **no custom radius values** outside these tokens to prevent drift.

Implementation Note: *Use consistent* `BorderRadius.circular()` *values in Flutter from a central source. Codex can detect literal radius values and enforce they match one of the token constants.*

Opacity & State Tokens

To handle component states (hover, pressed, disabled, etc.) and layering, we define standard opacity levels and state styles:

- **Hover (Pointer Over):** *Hover state* (applicable on web/desktop or button focus on mobile) is indicated by a lightening of the element. Use an overlay of white at 8% opacity on the element's color (or equivalently, increase brightness by ~5-10%). Token: `opacity.hoverOverlay = 0.08`. This ensures a subtle highlight that meets contrast needs for focus indicator [6].

- **Pressed/Active:** *Active state* (on tap press) is indicated by a slightly stronger effect than hover. Use a white overlay at ~16% (`opacity.activeOverlay = 0.16`) on dark surfaces (lightening them), or a black overlay at 20% on accent-colored buttons (darkening the bright accent slightly). The result is a clear "depressed" look when a button or card is being pressed.
- **Selected:** *Selected state* (for toggles, selected list items, or the active map pin) uses the accent color in a controlled way. For list items or cards: apply a subtle teal **tint** to the background (e.g. `color.accent.dim` at 10% opacity) or a **1-2px accent outline** around the element. This draws attention without filling the item with full accent. For map pins, the selected pin is fully accent-colored (teal) while non-selected pins are neutral (see Map Styling). Any selection highlight must maintain a 3:1 contrast against both the component and background [6] (e.g. an accent outline on a dark card is clearly visible).
- **Disabled:** Disabled elements are visibly muted. Apply 50% opacity to the entire element (token `opacity.disabled = 0.5`) relative to its normal state [6]. For text, use `color.text.primary` at 50% (appears mid-grey on dark). For icons or buttons, either render them with the same color at 50% opacity or swap to a designated disabled color token (e.g. `#94A1AD`, the secondary text color). The goal is a unified disabled appearance – no interactive accent should show at full strength if the control is disabled. Disabled elements have no hover or active effects.
- **Focus:** Focus (for keyboard navigation or accessibility) is indicated with a clear outline or glow. Use a **focus ring**: 2px outline in `color.accent.primary` (teal) if around a dark surface, or in white if around an accent surface, ensuring ≥3:1 contrast [6]. Optionally, a soft outer glow of teal (8px blur, 50% opacity) can accompany the outline to improve visibility on very dark or busy backgrounds. This is only for the element currently focused (e.g. a text field or button).
- **Transitions:** All state changes (hover in/out, press, selection, focus appearance) should be **animated** with a short duration token: `motion.duration.short = 100ms` (for simple hovers) to `motion.duration.medium = 200ms` (for selection changes). Easing function should be standard (ease-in-out for hover/focus so it's gentle, maybe linear for press to keep it snappy). This prevents abrupt jarring changes and provides feedback to the user that feels smooth.

Implementation Note: *Encapsulate these state rules in widgets (e.g., Flutter's* `InkWell` *for splash/hover with a configured splashColor using the overlay opacities). Use theme highlightColor/overlayColor to globally set the 8%/16% overlays. Codex enforcement: ensure that any custom component uses these standard opacity values for hover/pressed (for instance, if a developer manually changes a button highlight, flag if it's not 0.08 or 0.16). Disabled state should be implemented by setting* `Opacity` *widget or* `ButtonStyle.foregroundColor` *with opacity from theme.*

Usage Guidelines and Constraints

This section details **how and where to use** the above tokens. Adhering to these constraints ensures a coherent look-and-feel. **Vibrant signal colors must be used judiciously** – neutrals dominate the UI, and accent appears only for emphasis [7]. The system enforces a strict visual hierarchy: one primary accent at a time, neutral backgrounds, and no unauthorized stylistic deviations.

Signal Color Usage

The teal/aquamarine **signal colors** are reserved for specific semantic purposes: - **Primary Accent (Teal)** is used for interactive highlights: the primary action button on a screen (e.g. a "Buy" or search button), selected map marker, toggles in "on" state, links, and key actionable icons. This draws user attention to interactive elements or important info. - **Match Strength Indicators** use the accent spectrum (teal to green) *only* to visualize semantic meaning (the strength of the query-to-store match). They are **not**

decorative; they always convey data (e.g. a higher percentage match is greener). - **Positive Feedback** (success state) may use the green end of the spectrum (e.g. a checkmark icon or success toast could use `color.semantic.success`), but such usage is infrequent and always paired with a message. - **Avoid using accent colors for large text blocks or backgrounds.** Do not set body text or entire card backgrounds in teal or green. Signal colors are meant to accent, not to fill large areas (which would reduce contrast and overwhelm the dark theme). - **Icons**: Important icons (e.g. a navigation icon for current location or a category icon) can use accent *if* it denotes an active state or important status. Otherwise, default icons should be neutral or white. For example, a solid teal icon might indicate "this is your selected item", whereas general icons in lists remain white or gray. - Do **not use accent colors for purely decorative elements** or to highlight non-interactive text. The accent must correspond to an interactive element or a data-driven indicator. For instance, don't randomly color a heading or underline in teal unless it signifies something (like a filter chip active). **No accent text** in long paragraphs – reserve color for short labels or values that need emphasis.

In short, **neutrals carry the bulk of UI**, and **teal accent appears sparingly for actions or data highlights** [7]. This ensures that when a teal element is on screen, it immediately signals "this is important or clickable."

Surface & Background Usage

All primary surfaces (screens, cards, sheets) use the **structural neutral colors**. This maintains high contrast and reduces eye strain in the dark theme: - The app background and map background use `color.bg.primary` (dark steel blue-gray). Large panels (drawers, headers) also use this or a very close variant. This consistent dark canvas provides a backdrop that lets accent colors pop. - **Cards and Panels**: use `color.bg.surface`, a slightly lighter neutral, to distinguish them from the background. The contrast is subtle (e.g., surface maybe 5-10% lighter than bg), just enough to see the card edges. This differentiation helps group content without using additional borders. - **Bottom sheets and modals**: also use the neutral surface color. Even though they overlay other content, they should not use a bright or accent background. The consistent neutral surfaces prevent any one panel from stealing focus simply by background color. - **No pure black** surfaces: While the theme is dark, avoid using true black (#000000) as a background except for possibly the map tiles if needed. The base neutrals (like #142633) have a hint of color (blue-gray) which gives the design warmth and avoids stark contrast of pure black vs white text. Similarly, no bright white backgrounds are used in dark mode (white is reserved for text and small elements). - **Scroll backgrounds**: If content scrolls over the map or another surface (e.g. a semi-transparent panel), use a translucent neutral (e.g., black at 80% opacity) – still maintaining the overall dark tone. - **Surface Contrast**: Ensure text on any surface meets contrast guidelines (e.g. white text on `color.bg.surface` should be ≥4.5:1 contrast; our chosen neutrals ensure this). If using secondary text (gray) on a slightly lighter card, confirm it's still readable (≥3:1 for larger text). Increase contrast by using a lighter text color rather than making surfaces brighter – we prefer adjusting text color to maintain dark UI integrity.

Accent Hierarchy & One-Accent-Per-Screen

To avoid visual conflict, Buycott enforces a **single accent hue at a time** on any screen [7]: - The **primary accent color (teal)** is the default and only accent used throughout the app's normal state. We do not introduce a different competing accent on top of it. For example, we wouldn't have a **teal** button alongside a totally different colored icon (like bright orange) on the same screen. - If a secondary accent is absolutely needed (for instance, to denote something distinct like an alert), it must not appear simultaneously with the primary accent in the same context. Practically, this means the UI should largely stick to the teal/aquamarine family. For error states (red) or warning (amber), these should appear in isolation and only in the moment of that message. They should not persist as part of the

regular screen UI. - **Hierarchy of Accents:** Within the teal spectrum, treat the mid-range teal (`#00E4E4`) as the **default accent** for interactive elements. The greener tones (aquamarine to chartreuse) appear only in data visualizations (match indicators) or success feedback, which are typically small UI elements (percentage text, small bars). Thus, even though technically two shades of accent might co-occur (e.g. a teal button and a slightly greener 92% match icon), they are close in hue and part of one family – this is acceptable. What is not allowed is introducing a completely different hue (e.g. a red or orange accent) concurrently as a design choice – **one accent family per screen**. - **Avoid Accent Competition:** Never use accent color in such a way that two elements compete for attention. E.g., if you have a teal CTA button as primary action, do not make another less important button teal as well on the same view. Instead, secondary actions should be neutral (text button or outlined). Similarly, if the map route is drawn in teal (accent), the pin icon itself might be a neutral icon with a teal outline, rather than fully solid teal, to ensure the route (primary highlight) remains the main accent element at that moment. - **Token Enforcement:** Design tokens help enforce this rule by only providing one set of accent values. Developers/designers should not add another brand color. If there's a temptation to use an additional color to differentiate something, consider using different **values** (e.g. lighter/darker teal) or neutrals instead of adding a new hue.

By adhering to one accent, the interface stays **focused and uncluttered**, and users learn to associate that accent with interactive or important elements across the app [7]. Neutrals and subtle variations handle everything else.

Map Styling Rules

The map is central to Buycott's UI. The map's visual style integrates with the app's dark theme, using neutral tones for base map features and **accent colors for highlights**. All map styling is token-driven and consistent with the UI theme:

- **Base Map & Structural Tones:** The map uses a **dark base** so that plotted results and routes stand out. Land areas and general background: use a very dark bluish charcoal (`#0F1A20` roughly, matching or slightly darker than `color.bg.primary`). Water bodies: slightly differentiated dark tone – e.g. `#142633` (the base blue-gray) for oceans/rivers, to give a subtle blue cast to water. The contrast between land and water is low (in dark mode, we avoid large bright blues for water) but just enough to discern coastlines. Park areas or forests: very dark desaturated green (almost black with a hint of green) – they should not draw attention, just be mildly distinct from urban land. Overall, the basemap colors are **muted and close to each other in value**. This ensures the map reads as a background layer. Administrative boundaries or minor grid lines, if shown, use the `color.border` neutral (very low intensity).
- **Road Hierarchy:** Roads are styled by relative importance through brightness (not hue). Minor roads and local streets: drawn in a dark neutral slightly lighter than land (e.g. a dark gray-blue line at 20% brightness) so they are barely visible, enough for context but not distraction. Major roads (highways, arterials): a bit brighter line (maybe 40–50% brightness neutral) or with a subtle **outline**. For example, a highway might be a dark gray line with a faint lighter casing to distinguish it. **No saturated road colors** (e.g., no bright yellow highways as in some maps). Everything remains in the dark gray/blue palette. This preserves the focus on user-specific overlays (our results and routes). Road labels: use `color.text.secondary` (desaturated gray) at small size. Labels for major places (city names, etc.) might use `color.text.primary` at a low opacity (e.g. white at 50%) – still legible but not overpowering the UI.
- **Route Styling:** When a user selects a result, the route from the user's location to that business is drawn prominently:
- **Route Color:** Use the **primary accent teal** for the route line (e.g. a bright aqua #00E4E4). This creates immediate contrast against the dark map [8].

- **Route Stroke Specs:** The route line should be thick enough to be visible on mobile screens: ~4dp width (with slight scaling at different zooms if needed). The route line can include a subtle outer border or glow for contrast – e.g. a 1px outer outline in `#1E2F3A` (dark) or a soft shadow – to ensure it's visible over varying backgrounds. This is especially useful if the route passes over a lighter element like a city label; the dark outline prevents color blending.
- If turn arrows or directional markers are shown on the route, they should use the same teal or white – but keep them small. The primary impression of the route should be a clean teal path.
- Only one route is shown at a time (the one to the selected location) to avoid multiple accent-colored lines; any alternate routes (if provided) would be in a neutral dashed line until selected.
- **Pin Icons & Markers:** Map pins follow a strict convention for clarity:
- **User Location Pin:** Represent the user with a distinct but not too flashy marker. Typically a small circle with an outline: for example, a white dot with a teal/aqua outline or glow. It should pulsate softly (using the accent color at ~50% opacity) to draw attention to "you are here." The pulsation (if implemented) uses the **glow** accent but at limited radius (no more than 10dp) and a slow pulse (~2s interval) to avoid distraction.
- **Search Result Pins:** All result locations are marked with a consistent icon (e.g., a drop-pin or location marker shape). Use a **neutral base with accent highlight** to avoid too many solid teal markers overwhelming the map. For instance, the pin could be a dark gray or steel blue shape with a small teal top or dot. However, the **selected result's pin** will be highlighted: the selected pin can be a solid accent teal icon with maybe a white icon glyph in it (if using an icon inside) or a larger size. Additionally, the selected pin might have an animated ring or a bounce animation when it's selected to catch the eye.
- **Pin Labels/Callouts:** When a pin is tapped, a callout card appears (either above the pin or as a bottom sheet). The pin itself can remain accent-colored to indicate "this one is active." Non-selected pins might be slightly smaller or a different shape (e.g., outline only) to recede.

- **Other Map POIs:** The map may show other points of interest (like landmarks, if any). These should use **standard map style** (often a simple icon and label in neutral colors) and not the accent, to avoid confusion with our results. Our app-specific pins take precedence in visual hierarchy by using the accent highlight.

- **Match Confidence Gradient:** A unique design aspect is the **match confidence color-coding**. When showing how strong a match is (e.g. 72% vs 90%), we use a **color gradient** from teal to green to visually indicate confidence:

- The gradient mapping: Lower similarity scores use the bluish end (e.g. around 60% = `color.signal.low` at #00CCFF, a blue-cyan). As the score increases, the color shifts through aqua (`color.accent.primary` at ~75-80%) toward green (`color.signal.high` and `peak` near 90-100%). At the top end, a near-chartreuse (#95FF00) indicates a very strong match. This gradient provides an at-a-glance cue: "greener" means a better match.
- **Usage in UI:** The match percentage might be displayed as colored text or a small filled graphic:
    - On map pins or callouts, a ring around the pin could be colored according to the score (for example, a circular outline whose color = match color). So a pin with a 90% match might have a vivid green outline, whereas 70% has teal.
    - In list items or the bottom sheet, the percentage number itself could be colored (e.g., "88%" in a font colored #00ECBD), **and accompanied by a descriptor** ("Strong Match" for high values, "Moderate Match" for mid) to ensure clarity.
    - Alternatively, a small bar indicator or radial progress graphic can be shown: for example, a horizontal bar under the store name that is filled X% and colored accordingly. Or a semi-circular gauge icon behind the percentage.

- **Do not use this gradient for any other purpose.** It is strictly tied to match confidence. No decorative gradients are used in the app, and no other metric uses this teal-green spectrum. This avoids confusion—users learn that colored bars or numbers always relate to match strength.
- **Accessibility:** Since color alone conveys match strength, always include the numeric percent or a text label for color-blind users. The color choices are also chosen to be distinguishable (we avoid reds in the gradient to prevent confusion with errors, and the difference between teal and chartreuse is perceivable even in various color blindness scenarios, though labels ensure no ambiguity).
- The gradient steps (as defined in Color Tokens) can be implemented via interpolation in code: e.g., given a percentage p, the color = interpolate between defined stops. For enforcement, ensure the colors used for any given value are within the set range – e.g., if a developer tries to use a completely different color for a certain percentage, that's not allowed.

Implementation Note: *Many map styles can be achieved via Mapbox or Google Maps style JSON. Ensure the JSON uses these token values. For example, set all POI icons to neutral grays, water fill to #142633, etc., and road stroke colors as specified. Codex enforcement could include a check on the map style configuration file to confirm the color values match tokens (no stray colors). For runtime drawing (like route lines or pins drawn in Flutter), use the color constants from the design tokens (e.g., route line uses* `Colors.tealAccent` *from the defined palette if it maps). Validate on screens that multiple accent hues are not appearing incorrectly (the system should warn if, say, an error red icon is shown persistently alongside teal elements).*

Component Design Rules

This section defines standards for common UI components: how they look, behave, and respond to interaction. Components are built using the above tokens and follow consistent layouts. **No component should diverge from the design system in styling** – e.g. a new card must use the same padding, radius, etc., as defined here. This ensures a cohesive experience.

Cards

Cards present information in contained boxes, such as a search result summary or a stat overview ("Personal Best"). Card design guidelines: - **Layout & Appearance:** Cards use `color.bg.surface` as the background. They have a border radius of 8px (`radius.md`) for standard cards (giving gently rounded corners). Padding inside a card is uniform: typically 16px on all sides (or 16px horizontal and 12-16px vertical). This provides comfortable spacing for content without making the card too bulky. - **Elevation:** Cards are elevated at **Elevation 1**, casting the standard small shadow on the background. In dark mode, this shadow is subtle; the card might also appear slightly lighter than the background as noted. This distinguishes it from the canvas. - **Content Alignment:** Follow the 8px grid for internal layout. If a card has a title and subtitle, the gap between them might be 4px or 8px depending on visual grouping (likely 4px if title and subtitle are closely related). Multiple pieces of info stack vertically with consistent spacing. For example, in a search result card: Title (store name) may be bold, below it a secondary line (distance or hours) 4px below, and maybe a match indicator aligned to the right. - **Behavior:** Cards can be static or interactive. - *Interactive cards* (like a search result that can be tapped) should have a hover and pressed state as defined (slight lighten on hover, etc.). They might also have an **ink ripple** or highlight on tap (Flutter's InkWell can provide this) – use the accent or a neutral for ripple? Usually on dark, a light ripple or subtle teal ripple works (but not too opaque to avoid big color blotch). - If tapping a card triggers navigation (e.g., opening details in a bottom sheet), the card should briefly highlight (pressed state) to confirm the interaction. - **Information Density:** Cards should not overflow with too much info. Use typography hierarchy to keep it scannable: e.g., one line for title, one for key info, maybe small icon + text elements (like a pin icon + distance "4 min away"). If more content is

needed, prefer expanding the card or using a bottom sheet rather than cramming. - **Visual Consistency:** All cards share similar styling. For example, the “Personal Best” stats card and a search result card might have different content, but they should look like they belong to the same family: same background color, similar padding and corner rounding. The difference might be in layout (the stats card might center content or use a graph, whereas result card is horizontal layout). But things like the header style, section dividers (if any), etc., should reuse token values (e.g., a divider line in a card uses `color.border`). - **Accent within Cards:** As per accent rules, avoid flooding a card with accent. If a card needs to draw attention to a particular value (say 86% match), do so with a small accent element (colored text or a tiny bar) rather than making the whole card teal. In a stat card, you might use accent for an icon or the numeric value, but the rest of the card (title, background) stays neutral.

Bottom Sheets

Bottom sheets (sliding up panels from bottom) are used for details like selecting a specific result or showing more options. They follow Material-esque behavior but with our styling: - **Shape & Color:** Bottom sheets use the same surface color as cards (`color.bg.surface`) to maintain continuity when a card expands into a sheet. The top corners are rounded at 16px (`radius.lg`), while bottom corners meet the screen edge (unless it’s a floating sheet, but usually it’s anchored to bottom). A small **grip bar** at the top (for drag indication) may be present: a 36px wide, 4px tall bar, colored in a muted gray (e.g. `color.border` or `color.text.secondary` at 50% opacity) and with full-pill radius. This grip helps indicate the sheet is draggable. - **Elevation:** Sheets appear above other content, using **Elevation 2**. This will cast a moderate shadow and typically also dim the content behind (apply a scrim behind the sheet: e.g., a semi-transparent black overlay on the map when sheet is open, so the user’s focus is on the sheet). - **Behavior:** The sheet can be swiped up to expand or down to dismiss (if design allows). It should move smoothly, with standard animation timing (~300ms ease-out for full expansion). When tapping outside the sheet (on the scrim), if the sheet is dismissible, it should close, following Material guidelines. - **Content Layout:** Inside, use a **vertical stack** with consistent spacing. If the sheet is a detailed view of a location, it might include: - A header (maybe a short top bar or a line indicating the sheet can be dragged). - The location name as H2, left-aligned, with maybe a close button or down-arrow on the right. - Content sections like address, hours, “Average acquisition time”, etc. These can be separated by 8px–16px gaps or thin dividers (1px line using `color.border` if needed). - Key metrics (like distance, match%) can be presented in a horizontal row with equal spacing. - If there is a call-to-action inside the sheet (e.g. a “Open in Maps” button), it should be at least 16px from the bottom (to not hug the screen edge) and full-width or appropriately sized. - **Scrolling:** If content exceeds the sheet height, the sheet content scrolls internally. The background remains `color.bg.surface`. Use a **scroll indicator fade** or shadow at top of content if needed to hint at more content. - **No Separate Accent**: The bottom sheet itself should not have a different colored header or accent stripe. Keep it simple – the content within will use accent for highlights as needed (e.g. a phone icon in teal for a call action, or a green text for “Open until 9:00 PM” if we decided to color open status). But the container stays neutral.

Buttons

Buttons trigger actions and thus use the accent color prominently to stand out. We define a clear hierarchy of button styles: - **Primary Button:** This is the main call-to-action (CTA) on a screen. It uses **filled accent** style: - Background: `color.accent.primary` (teal #00E4E4). - Text: `color.text.inverse` (which is basically black or white depending on contrast; in this case, white, to contrast the teal). - Border: none (or if a border, maybe a subtle 1px in teal slightly darker, but generally filled buttons don’t need borders). - Corner radius: 8px (`radius.md`). - Padding: 16px horizontal, 12px vertical typically (to make a comfortably tappable touch target ~48px tall). - Font: Body or button font (16px, Semi-Bold perhaps, since it’s a short label like “Buy” or “Search”). - **States:** On hover/focus, lighten the background (we add the 8% white overlay, making it a bit brighter aqua). On press, slightly darken

(20% black overlay) so it looks pressed. Disabled: background becomes a gray (we can apply 50% opacity to the teal which yields a dull grayish teal, automatically handled by `ButtonStyle` using opacity). The text in disabled state also becomes semi-transparent. - Only one primary button per screen ideally (the main action). Others should defer to secondary style. - **Secondary Button:** Used for less prominent actions or where multiple actions are needed (e.g. a "Cancel" next to a "Save"): - Style: **Outline or Text** on dark theme. An outline style works well to keep it subtle: - Background: transparent (or `color.bg.primary` if we want it to look like just text on background). - Text: `color.accent.primary` (teal text) to indicate it's clickable yet not filled. - Border: 2px solid teal for outline variant, with 8px radius. This gives a defined shape but visually less weight than a filled button. - Alternatively, a text-button style: no border, just text in accent color. Use when even less emphasis is needed and when it's clearly distinct (e.g., in a dialog "CANCEL" as a text button vs "OK" as filled). - States: On hover, if outline – fill its background with 8% teal (very light) or 8% white? Possibly white 8% to lightly highlight the area. On press, increase to 16% fill. For text-only buttons, underline them or slightly brighten text on hover to show it's interactive (since background is none). - These secondary buttons should not outshine the primary. Ensure the accent text is not too loud – perhaps we use a slightly lighter teal for text (like the accent hover color #33EDEE for the normal state text) to differentiate from a fully saturated primary button. - **Tertiary/Button Icon:** If icon buttons are used (like a circle icon button to refresh or locate), style them consistently: - Icon-only buttons on dark background can be shown as circular with `color.bg.surface` background and an icon in accent or white. - Example: A "locate me" floating action button on the map – could be a circle with 56dp diameter, background `#263645` (a bit lighter than main bg), and a white GPS icon. On hover/press, background can turn slightly lighter or icon can turn teal if not already. - If an icon button is meant to represent an action in the accent color (like a floating action "scan" button), then fill it with accent teal and put a white icon, same as primary button style but round. - **Destructive Button:** Though rare (like "Delete account"), if it exists, use the error color (red) as the fill or text, following the same pattern (filled red with white text for a primary destructive action). Use only in contexts where that action is primary and destructive. Otherwise, avoid using red buttons in the interface at all (to keep within one accent rule, these would appear alone in confirmation dialogs). - **Button Alignment & Spacing:** Buttons inside layouts should span full width if alone in a row (e.g., a single CTA at bottom fills parent width minus margins), or if two side by side, each take half or use appropriate spacing (maybe 8px gap). They should align to the 8px grid (e.g. left margin 16, button height 48 aligns). - **Iconography:** If a button includes an icon (e.g. a right arrow in a "Navigate" button), the icon should be tinted the same as the text (white for primary filled, teal for outline), and placed with 8px padding from text.

Match Indicators

Match indicators communicate how well a store matches the search query. They are a crucial UI element and have a consistent format: - **Presentation:** Typically a numeric percentage (e.g. 88%) accompanied by a qualitative label (like "Strong Match" or "Moderate Match"). The percentage is usually bold or larger than surrounding text to catch attention. It may be enclosed in a small pill or badge for emphasis. - **Color Coding:** As described in the map styling, the color corresponds to the confidence: - Low match (below some threshold, say <50%) might not be shown or could be labeled "Weak" in gray to indicate not a strong find. If shown, maybe use a very desaturated teal or just leave as white text to avoid implying a strong connection. - Moderate (around 60–80%) – use the mid accent color (teal). The "Moderate Match" text could be in a neutral color with only the percentage in teal, or the whole badge in teal with white text for the percentage. However, avoid full teal fill if it's not a strong match – better to use just text colored teal. - Strong (80%+) – use the brighter aquamarine/green. For top matches, you could display a small circular icon filled with green and a white checkmark, next to the text "Strong Match" – but that might conflict with green meaning success. Alternatively, simply color the text or a star icon in that green. The percentage number could be green to visually reinforce it. - **Badge Style:** One approach is to use a **capsule badge**: e.g., `[ 88% match ]` with a rounded rect background. If

doing so: - Use a dark background with a colored border or colored text: e.g., background = `color.bg.primary` (transparent look) and border = colored line in accent, text white. Or background tinted with accent (maybe accent at 20% opacity) and text white. Either way, keep it subtle enough not to draw more attention than the primary CTA. - Size the badge small (maybe 24px height, padding 4px sides) so it doesn't dominate. - **Inline Text:** In contexts like a list, you might simply append "(88%)" after the store name, colored appropriately. Make sure there's a space or parentheses to visually separate it. The text color can be accent and perhaps semi-bold, whereas the store name stays white. This way the match is visible but the name (content) is still primary in hierarchy. - **Icons:** Optionally, use a small icon to represent match quality. For example, some systems use signal bars or a "circle filled 3/4" icon. If we were to, we'd ensure the icon is also colored in the accent range. But since a numeric percentage is straightforward, an icon is not strictly needed. - **Placement:** Always place match indicators in a consistent spot for easy scanning. In a search result card, perhaps right-aligned on the same line as the distance or store name, so as you scroll you see a column of match % on the right. In a detail sheet, the match could be in the header next to the business name (like a subtitle "87% match"). Visual consistency helps users know where to look. - **Do not confuse with other badges:** Only use the teal/green badge for semantic match strength. If you have other badges (like "Open Now" or "New"), style those differently (e.g., grey background, white text) to avoid them looking like match info. - **Dynamic Updates:** If match percentages could update (probably not in real-time in UI, but if re-sorted), any change in the indicator color should animate (e.g., cross-fade from one color to another over 300ms) rather than snapping, to draw attention without a harsh flash.

Numeric Emphasis & Metrics

Certain numeric information (time durations, percentages, counts) are key to Buycott's experience (e.g., "4 min away", "5 min avg. acquisition time", "146 total acquisitions"). The design treats these numbers with special emphasis: - **Bold or Differentiated Weight:** In a block of text that includes numbers, the numbers can be rendered in a slightly heavier weight or different style to stand out. For example, in "4 min away", the "4" could be Semi-Bold while "min away" remains Regular weight. This helps users scanning the card to pick out the numeric values quickly. - **Slight Size Difference:** For critical stats (like a main stat on a personal dashboard card), the number may be set larger than the label. E.g., "146" could be 24px while "total acquisitions" is 14px below it. Use this sparingly and logically (commonly on personal summary or highlight cards). The typography tokens include a notion of a larger numeric style if needed. - **Alignment:** When displaying multiple numeric metrics (like a series of stats), align them in a way to facilitate comparison. Typically right-align numbers if in a table column, or use tabular figures monospaced as specified. For instance: - If showing `11.3` avg time and `73%` coverage strength stacked, align the decimal points or use same width for each so they look neat. - In a horizontal row of metrics, give each metric its own mini-block with consistent width rather than mixing text flows. - **Units and Labels:** Always accompany numbers with context (units or label) in a smaller or lighter style. E.g., display "5 **min**" rather than just "5" to avoid ambiguity. The unit "min" can be in lower-case, slightly smaller font or lighter color (secondary text color). For percentage, appending "%" is clear, or writing "match" after it. We don't rely solely on color or size to explain a number. - **Color Usage for Numbers:** In general, keep numeric text white or neutral, except when indicating an especially positive or negative connotation: - If a number is itself a status (e.g., 0 results found might be shown in a muted or red context), but normally, "4 min" doesn't need to be teal. We use teal for interactive or match semantics. Time and distance remain neutral (white) because they are factual, not indicating good/bad. - However, in charts or special highlight, a number might take on an accent: e.g., if "3 min (personal best)" is exceptionally good, you might color it teal or green to celebrate. Or if it's beating a goal, etc. But this would be explicitly defined; generally, times and counts remain neutral text so that the match percentage or any interactive element can use the accent color. - **Real-time changes:** If any number updates live (e.g., a timer or count), ensure the styling remains consistent and the layout doesn't shift (hence the tabular nums). If highlighting an increase or decrease, do so with icons (▲▼) or subtle color

(green up arrow, red down arrow perhaps) rather than changing the number's color itself extensively. But in this app context, live changes are probably minimal.

Data Visualization

The app may include small data visualizations, such as bar charts (e.g., personal efficiency score, coverage strength over time, etc.). All data viz elements should align with the design system colors and typography: - **Color Palette:** Use the approved color tokens for any chart elements. For instance, a bar graph comparing the user vs average might use **teal** for the user's bar and a neutral gray for the average bar. Avoid introducing new colors for chart series. If multiple categories are needed and must be distinguished: - Prefer shades of the accent for positive metrics, or shades of neutral for background metrics. - Example: A stacked bar could use teal and aquamarine as two parts (both in the accent family), instead of teal and say purple which is off-brand. - If more than two series, use neutral tones for less important series and only one accent for the key series. Or differentiate with patterns (striped vs solid) rather than colors alone. - **Background & Gridlines:** Charts should have a transparent or neutral background (e.g., if a chart is on a card, it should use the card's surface color). Gridlines or axes can use `color.border` or be very subtle lines (maybe 1px, 50% opacity of secondary text). They shouldn't stand out; they are for reference only. - **Text in Charts:** All labels, axis text, and values use the Typography tokens. Keep them small but legible (maybe 12px for axis labels, 14px for important value labels). They should use neutral colors (white or secondary text color) depending on contrast with the background. Critical values might be bold. - If labeling a data point directly, use white text on a teal data point (if the teal is dark enough, or consider putting a dark outline or drop shadow behind text for contrast). - Ensure any text over colored bars meets contrast (if bar is teal, white text is usually fine). - **Visual Emphasis:** Use accent color to draw attention to certain data: - If one number is the focus (e.g. an achievement score), you might show a large number in the center in accent color. - If showing a trend line, you might color the line teal and fill the area under it with a light teal translucent fill. - Data that is less critical can be gray. For example, a baseline or goal line on a chart can be a dashed gray line. - **Consistency:** Maintain consistent styling across all charts. All bar charts should look alike (same corner rounding on bar ends, same spacing), all pie charts if any should use the same stroke width etc. However, note: pie or donut charts in dark theme can be tricky with our palette (big areas of teal/green might be loud). Possibly prefer bar/line charts. - **Numeric formatting:** Use the same numeric rules for any numbers on charts (tabular if stacked vertically, units included, etc.). If a chart has a y-axis in minutes, label like "5 min" not just "5". - **Interaction:** If charts have interactive elements (like hover to see value), the tooltip should follow our design too: likely a small neutral tooltip with text, maybe slight transparency. If highlighting a bar or section on tap, consider using the accent outline or a glow on that segment. - **Motion:** Animate data visualization entrances in a subtle way (e.g., bars grow from 0 to value on load, or lines draw from left to right). Use our motion timing tokens (maybe 300ms for simple charts, up to 800ms for more complex) to make it smooth but not sluggish. Easing out is nice so it settles gently.

In summary, data visualizations should feel like an extension of the UI, not a separate graphic. They use the same dark backgrounds, the same accent for highlighting, and same fonts for text.

Layout & Responsive Design

The layout system ensures the app looks balanced on various screen sizes and orientations. Key principles include the 8px grid usage, consistent padding, and scalable components: - **8px Grid:** *All* UI elements align to an 8dp grid (as mentioned in spacing tokens). This means: - Left/right padding of main content from screen edges is 16px (which is 2 * 8). - Icons and text baselines align on multiples of 8 whenever possible. (For instance, an icon of 24px height is placed in a 32px container with 4px padding top/bottom). - Vertical rhythm: stack text and components using 8px or 16px gaps. Uneven

spacing (like 10px) is not used because it would misalign with other content and break the rhythm [3]. - **Padding & Gutters:** Standard padding inside containers: - Cards: ~16px padding (or 12px vertical, 16px horizontal). - Bottom sheets: 16px horizontal padding inside, and maybe a bit more top padding if there is a header (to differentiate from the drag handle area). - Screen-level (page) padding: 16px from edges, but if the screen has a full-bleed map behind, the content (like search bar or floating cards) should still obey a margin from edges (usually at least 8-16px from edge to avoid touching screen bezel). - Lists: if we use a list of cards or items that already have internal padding, the list container can have 0 padding and rely on item spacing. If not, give the list a padding of 8px top/bottom so items aren't cut off at extremes. - Touch targets: ensure at least 48px of space for any tappable element (which often means at least 8px padding around icon or text in a button). This is partly spacing and partly interactive guideline. - **Section and Stack Separation:** Visually group related items by smaller gaps, and separate sections by larger gaps [9]. For example: - In a bottom sheet with multiple info sections (address, hours, stats), use an 8px gap between items within a section, but maybe a 24px gap between the address section and the stats section title. This way, sections are clearly distinct. - If using headings within a page, put extra spacing above a heading to differentiate it from the previous content (e.g., a section title might have 16px above it and 8px below it if followed by content). - Use consistent values: e.g., always 24px above a major section heading, never 18px one place and 30px in another. - Avoid double-spacing by mistake; e.g., if a card already has 16px margin bottom and the next section has 16px top, realize that yields 32px gap – ensure intended or adjust to not inadvertently create off-grid gaps. - **Responsive Scaling:** While the primary design is for mobile portrait, the system should scale to different screen sizes: - On smaller devices (narrow width), content can tighten: e.g., maybe reduce horizontal padding to 8px if screen is extremely small (320px wide), to utilize space. Font sizes remain mostly the same (we don't want to go below legibility), but be mindful of truncation. Use flex and maybe vertical stacking rather than shrinking text. - On larger devices or landscape mode, maintain the grid. Possibly increase margins on the sides (e.g., on a tablet, you might keep content to a max width for readability, using margins). The one-accent rule etc. still applies. - On web or desktop (if Flutter web), consider breaking longer lists into columns or revealing more content in one view. But any such changes should still adhere to the spacing scale (e.g., if using a two-column layout on wide screen, ensure a 32px gutter between columns). - **Responsive Typography:** If needed, the type scale can step up slightly on larger screens (e.g., H1 might be 36px on tablet vs 32px on mobile), but this should be done sparingly. More often, the same sizes can be used; the spacious screen will naturally make them look smaller relative to screen, which is usually fine. If anything, just don't let text lines get too long in width (optimal reading width ~60 characters). This can be handled by introducing max-widths for text blocks or columns on wide layouts. - **Scroll and Overflow:** Because of the map-first nature, some screens have floating overlays. Ensure that when content is scrolled (lists inside sheets, etc.), padding is preserved and content doesn't abut edges suddenly. e.g., inside a scrolling bottom sheet, maintain 16px padding at bottom so last element isn't flush with sheet bottom. - **Consistent Alignment:** Use alignment to grid and between elements to avoid a jarring look. For instance, if you have icons and text in a row, align their centers on the same horizontal line. If multiple cards are side by side (maybe in a carousel), ensure they all have equal width and align to the same baseline vertically. - **Baseline grid:** Although not always explicit, try to align text baselines every 4px or 8px as well, for a harmonious vertical rhythm (this is more subtle, but it means line-heights should ideally line up such that baselines are on the grid or half-grid).

Adhering to these layout rules results in an interface that feels balanced and intuitive, with no odd gaps or crowding. It also simplifies development (since everything is an increment of 8, layout calculations are predictable).

Typography Hierarchy and Usage

We've defined the typography tokens, but here we clarify their *usage and enforcement* in the hierarchy of content:

- **App Title / Logo:** The app's logotype (if displayed in a splash or nav) would likely use a custom style, but within the UI, large titles should use H1 style (32px, Semi-Bold). For example, a welcome screen or a major section like "Buycott Stats".
- **Page Titles (H2):** Each screen's title or prominent section title uses H2 (24px Semi-Bold). E.g., the top of the "Personal Best" section might have "Personal Best" in this style. These should be unique per screen to avoid confusion and placed with enough padding above to stand out.
- **Subheaders (H3):** For sub-sections or card titles. For example, within a bottom sheet, you might have a subheader like "Store Details" before listing address/hours, styled in H3 (20px, Medium). Use H3 also for things like form section headers or group titles.
- **Body Text:** The majority of descriptive text, instructions, or longer form info (like an explanation of semantic matching, or a paragraph in a help section) should use the body style (16px Regular). It should be left-aligned for readability (never justified, to avoid odd spacing). Line length should be capped (as mentioned in layout) to keep it readable. On dark background, use primary text color (white) for Body. Avoid using body text in accent color; keep it neutral for maximum contrast and legibility.
- **Secondary Text / Caption:** Use the smaller text styles (14px or 12px) for any ancillary information. For example, on a card, the distance "7 min away" could be 14px, especially if the title is 16px – this differentiates it subtly. Caption (12px) is good for things like time stamps ("Updated 2h ago") or footnotes.
- Be cautious with 12px on mobile; ensure the color is high contrast (we typically use it in white or light gray because small text + low contrast would fail legibility).
- Use Secondary color (gray) when the info is not critical (like a timestamp). But if even small text conveys important info (like "Open until 9:00 PM"), you may keep it white to ensure it's read.
- **Lists and Tables:** If presenting data in a tabular format (maybe a comparison of options or a list with multiple columns like the example use cases table in the README), maintain consistent font sizes across the row to avoid misalignment. Usually Body small or Caption is used for tabular data if space is tight. Use tabular figures as specified so numbers align in columns.
- **Highlight text:** If a piece of text needs special emphasis in a body (like a key term), prefer using Semi-Bold or Italic rather than a different font size or color. This keeps it accessible (screen readers understand etc.). Color is used only if it's a link or interactive (teal for links).
- **Links:** In this app, links might be minimal (perhaps a link to an external map or website). If so, style links as inline accent color text (teal), underlined on hover. Use the same accent token as primary accent. Ensure link text is discernible from regular text (color and maybe an underline or differing weight).
- **Uppercase Labels:** If any labels are all-caps (like some UIs have buttons or tab labels uppercase), ensure letterspacing is provided (usually +0.5px tracking for readability at small all-caps text). But prefer to avoid all-caps except perhaps very small container labels or icon annotations. None of the example content suggests heavy use of all-caps, so likely skip except maybe an acronym or "AM/PM" etc.
- **Numerals:** (Restating but emphasizing) Always use tabular and lining figures. Also ensure numerals use the same typeface so they don't look out of place. Do not switch to a gimmicky "digital" font for numbers, for instance. Consistency is key.
- **Dynamic text sizing:** On different device pixel ratios, fonts might appear slightly different – test that these sizes remain legible. Possibly allow the user's accessibility font scaling: the app should handle if user chooses larger text – our spacing should accommodate one or two size steps

larger without breaking. Keep that in mind (e.g., allow text to wrap or containers to expand as needed).

- **Enforcement:** Developers must use the predefined text styles. For example, in Flutter, use `Theme.of(context).textTheme.bodyText1` (mapped to our 16px style) rather than setting a Text widget to 16 manually each time. This central control means if we adjust the font or size in the theme, it propagates everywhere. Codex rules could flag usage of `TextStyle(fontSize: ...)` in code if it's a value equal to one of our token sizes but not referencing the theme (which might mean a dev hard-coded it).
- **Accessibility:** Maintain at least AA contrast for all text. E.g. if we have any text on teal background (like a chip), it must be white text (teal on dark is fine, white on teal is fine, but gray on teal might fail). Already covered but reiterating in context of typography: contrast checking is mandatory for any new text/background combination [10] [11].

By following this hierarchy, the content across the app will have a clear structure: Users will quickly distinguish headings from body text, primary info from secondary, and interactive text from static text.

Interaction & Motion System

Interactivity in Buycott should feel responsive and refined. All interactive feedback (animations, state changes) are designed to reinforce the user's actions without being distracting or inconsistent. We outline the **motion timing, focus/glow rules, and state transitions** below:

Motion Timing & Easing

Consistent timing for animations creates an intuitive sense of how the app responds: - **Standard Animation Speeds:** Use a base duration of ~200ms for most UI transitions. Examples: - Button press ripple/fade: ~100-150ms for the initial press feedback (fairly quick). - Navigation transitions (one screen to another, or sheet slide): ~300ms, giving a smooth movement that's noticeable but not sluggish. - Micro-animations (icon rotations, state toggles): 200ms is a sweet spot. - **Easing Curves:** Prefer **ease-out** or **ease-in-out** for movement: - Ease-out (fast start, slow end) for things like a bottom sheet appearing (it should decelerate as it comes to rest, feeling natural). - Ease-in-out for symmetrical transitions (something fading in while something fades out). - Avoid linear for position changes (feels mechanical), but linear might be fine for color or small opacity changes if under 150ms. - **Sequential vs Parallel:** If multiple elements animate, consider staggering if it aids comprehension. E.g., selecting a card: first the card elevates and highlights (100ms), then the bottom sheet comes up (300ms). Slight delay (50ms) between them can make it feel orchestrated. However, don't make the user wait unnecessarily – parallel animations are fine if they don't conflict visually. - **Motion Purpose:** Animations should always serve a purpose (feedback or focus). No gratuitous bouncing or spinning just for ornamentation. For instance, when you tap a result, the map camera moves to center on that location (duration maybe 500ms, easing out), the pin might bounce slightly upon landing (a quick spring effect, one bounce within 300ms). These give a cue that "here is the item you selected". - **Frame Rate & Performance:** Keep animations smooth (60fps target). This is more a dev note: avoid heavy animations that could jank. The design's role is to ensure we don't demand something like animating a blur radius that might be expensive. Most of our animations are position, opacity, or color changes which are fine.

Focus and Glow Effects

Focus (especially for keyboard navigation or accessibility focus on elements) and **glow** effects draw user attention or indicate interactivity: - **Focus Rings:** As mentioned, use a 2px outline in accent for focus. This should appear when an element is focused via non-touch input (e.g., tabbing through links/buttons, or screen reader focus). - The outline should have a slight rounding matching the element's

shape (e.g., if a button has 8px radius, focus outline also rounded 8px). - If the focused element is already teal (like a primary button), consider using a white outline (2px) instead, to stand out against the teal fill (ensuring contrast). - The focus outline can fade in/out (animate its opacity or scale from 0 to 1) in ~100ms so it's noticeable but not popping harshly. - **Glow Usage Limitations:** Glow or shadow effects are used sparingly: - **Pulsing Glow (Attention):** For example, to highlight the user's location or a new result, a gentle pulsing ring can be used. It should be subtle: e.g., a ring of teal around the user dot that expands from 0 to e.g. 24px radius and fades out, over 2 seconds, repeating. This only on the user's own location or on an item that requires immediate user attention (like a newly appeared recommended result, if such exists). Don't put pulsing glows around every pin or button. - **Hover Glow:** We do not use heavy outer glows on hover of buttons or cards (some designs do a neon glow on hover – we avoid that to keep it sleek). Instead, we rely on the color/opacity change. The only slight exception: a focus state on a card might drop a very faint glow if needed to differentiate from hover. But generally, stick to outline rather than fuzzy glow. - **Text Glows:** No text shadows or glows for readability on dark – instead, choose appropriate text color. We avoid the old-school "neon text glow" look entirely (fits the aesthetic, but harms legibility and appears inconsistent with flat design). - **Elevation Shadow as Glow:** In dark mode, sometimes a component's shadow can look like a glow (light edges). We have defined subtle shadows. Do not add additional blue glows to accentuate elevation beyond the specified tokens. - **When Glow is Prohibited:** Glow should not be used as a general accent indicator. For example, do not make icons glow on hover or toggle; instead change their color or add a simple underline for text. Glow is reserved for: 1. Map location pulses (user or maybe a search highlight). 2. Possibly an error indication – e.g., an error dialog might shake and glow red briefly (though we might just shake or outline red). 3. Focus outlines (as an alternative style). If developers attempt to add any other glow (like "this panel has a constant glow border"), that's against the design spec. - **Visual Comfort:** If any glow is used, ensure it's low opacity and large radius so it's soft. Hard, bright glows cause eye fatigue against dark backgrounds. For instance, a teal glow around a pin should be maybe 20% opacity at most. A little goes a long way in dark UI.

State Transitions & Feedback

For each interactive state, we ensure a smooth transition: - **Hover to Press:** On pointer input, when an element goes from hover to pressed, the visual change (e.g., overlay getting stronger) should be immediate (no delay on press feedback) and then when released, the state should animate back to hover or normal. Use the defined `motion.duration.short` (100ms) to transition out, so it doesn't just snap off. This avoids a "flash" feeling. - **Selection Change:** When an item becomes selected (e.g., selecting a filter, toggling a switch): - If it results in a UI change like turning the toggle knob teal or showing a checkmark, animate those. A toggle sliding could take ~150ms with ease-out. A checkmark icon can fade in within 100ms. - If selecting an item triggers navigation (like selecting a search result opens bottom sheet), provide a subtle continuity: maybe keep the selected card highlighted until the sheet fully opens, then the highlight can settle. This ties the two UI pieces together mentally. - **Disabled State Changes:** Typically, enabled/disabled is not animated; it's a sudden state (e.g., a form button becomes enabled when fields are filled). We can simply switch the style (from disabled grey to active teal) without a fancy animation, or with a very quick fade (100ms fade from grey to teal) to make it not jarring. But immediate is acceptable since it's system feedback that an action is now available. - **Page Transitions:** If navigating between pages or major panels, keep transitions consistent (if using Flutter's navigation, likely a standard push transition or maybe a custom one since map context). - Possibly we want a subtle transition for the map screen itself if content overlays come/go. For example, launching the app could fade in the map and slide up the search UI elements. - The transition style (slide, fade) should remain uniform across similar actions. E.g., all full-screen dialogs might fade in on top. - **Feedback Animations:** On successful actions, we might incorporate small pleasant animations: - e.g., after successfully finding an item and tapping "Mark as acquired" (if such feature exists), maybe the checkmark icon does a brief scale-up and down (like a pop) in green. - On error, maybe the error icon or

the border of a field shakes slightly (horizontal shake 3px left-right, 2 cycles, over 300ms) – a standard form error indication. - These little feedback animations should follow the tone: not goofy or excessive, just enough to acknowledge the event. - **No Abrupt Changes:** A principle: avoid hard cut changes in UI without transition, except when absolutely necessary (e.g. real-time map updates which might just appear). For anything in our control, prefer a fade or slide that lasts at least 50-100ms to cue the eye. Even something like a list resorting after applying a filter could fade the list out and in or items slide to new spots rather than a sudden resort jump. - **Motion Consistency:** The user should perceive the system as having consistent physics. If one sheet slides up quickly, another similar sheet should not slide up slowly – they should match. Document typical times: e.g. *Sheets slide: 300ms ease-out*, *Tooltips fade: 150ms ease-in-out*, *Loading spinners appear/disappear: fade in 200ms, no fade out if replacing content maybe*, etc. Developers should use these standard timings from a central place if possible (like having duration constants). - **Platform Adaptation:** If targeting iOS/Android, consider adapting to platform conventions subtly: e.g., perhaps use Cupertino-style page transition on iOS (slight slide) vs Material on Android. But if using Flutter's Material design as baseline, it might be okay. The main thing is, it should feel at home on device (e.g., maybe disable overscroll glow in dark mode – that blue glow that Android shows on overscroll might clash, consider customizing it to teal or neutral).

By adhering to these interaction guidelines, the app will feel **polished and intentional**. Every hover, click, and navigation has a predefined response, eliminating guesswork and preventing any odd or inconsistent behavior from creeping in.

Strict Anti-Drift Rules

To maintain the integrity of the design, we set forth **anti-drift rules** – hard constraints to prevent gradual deviation over time. These rules are to be enforced both in design reviews and via automated checks:

- **Prohibited Off-Brand Colors:** No colors outside the defined palette may be introduced without design approval. This means no random hex values in code or design comps. **Developers must not use system default widget colors** (which might be blue, etc.) unless mapped to our tokens. For example, no bright reds or oranges for emphasis (aside from the semantic error/warning tokens). Even within the teal family, do not adjust hues arbitrarily – stick to the provided HEX codes for consistency [7]. If a new UI element seems to require a new color, it should be discussed and a new token officially added rather than a one-off usage.
- **No Decorative Gradients:** Gradients are **not** part of Buycott's standard UI, except the match confidence use-case. Do not use gradients for backgrounds, buttons, or randomly behind text. The dark steel blue-gray background is **flat** (or at most a subtle radial shadow in map, but not a color gradient) to keep the look clean. Accent areas should be flat color fills. This avoids a cluttered or inconsistent look. If a designer feels a subtle gradient would enhance a background, it must be extremely subtle and still only use our existing colors (for example, a slight top-to-bottom shift from #142633 to #1B2E3C – both within our neutral range – could be allowed if it adds depth, but it should be discussed). Under no circumstance should multi-color or high-contrast gradients be introduced [12] [13]. The LinkedIn-inspired theme gradient (steel blue to teal) informed our palette but is not used as a literal background in the app's UI without explicit design call.
- **No Bright/Light Surfaces in Dark Mode:** All surfaces remain dark. Do not suddenly use a white or light gray card in the dark theme – that would be jarring (the only exception might be an image or map tile which contains lighter tones by necessity, but that's content, not UI chrome). Also avoid large solid areas of accent color as surfaces – e.g., a teal notification banner spanning the width would be too bright; instead use a dark background with a left border or

icon in teal to convey the message. Bright surfaces ruin the dark mode aesthetic and can cause eye strain with high contrast transitions. Keep the overall luminance low.

- **Avoid Excessive Accent Usage (Accent Stacking):** As discussed, do not place multiple accent elements together in a way that they compete. For example, a screen that already has a teal header and teal primary button should not also have teal used for secondary icons or texts – those should be toned down to neutral. Only the primary interactive or highlighted elements get the accent. This prevents an oversaturation where nothing stands out. In components, similarly, if a component (like a card) has an accent-colored element (say a percentage badge), the rest of the card (title, icon) should not also all be in accent. One accent highlight per component. If a developer finds multiple accent-colored widgets adjacent, they should reconsider the hierarchy and likely change some to neutral. Automated linting could catch if more than X instances of accent color appear in a single screen widget tree, though that's complex; at least code review should.
- **No Custom Fonts or Sizes:** Don't use unapproved font families or arbitrary font sizes. All text should conform to the typographic scale. For instance, no random 15px or 17px text – it's either 16 or 14 as defined. No usage of fancy display fonts or icon fonts that aren't part of the design system. If an engineer needs an icon, they should use the official icon set (or SF Symbols or Material Icons, styled to our color) but not use, say, an emoji or a random glyph with a different style.
- **Preserve Spacing Consistency:** Do not introduce odd spacing to "make something fit." If an element doesn't fit within the grid, re-think the layout rather than use 5px margins. Every element added should respect the spacing tokens around it. A common drift is adding tiny padding to nudge something – this accumulates as inconsistency. Instead, adjust the layout using multiples of 8. The design system should be seen as a constraint system; violating it undermines the consistency.
- **Enforce with Tools:** Use tools to keep things in check – style linting to catch hard-coded colors, automated UI tests to compare against approved designs (visual regression). For example, if a pull request includes a color code not in our palette, it should be flagged automatically (Codex rule: search for "#" hex patterns in code and compare against allowed list). Similarly, if a text style is created with a font size not in {12,14,16,20,24,32}, flag it. Over time, these rules prevent unintended drift.

In essence, these anti-drift rules boil down to **"follow the system, not personal preference."** Any deviation must go through the design system maintainers. By being strict, we ensure that six months from now, Buycott's UI will still look like one coherent product, not a patchwork of styles. Each rule above is non-negotiable for developers and designers working on the project.

Implementation & Enforcement Notes

Finally, to ensure this design system is effectively realized in the product, here are guidelines for implementation in Flutter and for automated enforcement:

- **Flutter Implementation:** Leverage Flutter's theming capabilities to encode these tokens. Use a custom `ThemeData` where:
- `ThemeData.brightness = Brightness.dark` and set `primaryColor`, `accentColor` (if using Material legacy) or better, define a `ColorScheme` with primary = our teal, secondary = perhaps a variant if needed, surface = surface color, background = primary bg, error = error red, etc. This integrates our colors at the core.
- Extend ThemeData with custom fields if needed (Flutter allows extension via `Theme.of(context).extension<YourThemeExt>()`) for things like match colors or custom radius values if not directly supported.

- Define TextTheme with our font sizes/weights. E.g., `headline4` for H1 32px, `headline5` for H2 24px, etc., mapping appropriately. Use GoogleFonts (or include the font asset) to get Inter, or if using system fonts, specify fallback family names.
- Global styles: set `buttonTheme`, `inputDecorationTheme`, etc. to use our colors and radius (for instance, input fields should have 8px radius and use accent for focused border).
- For spacing, you might create constants in a Dimensions class, since Flutter doesn't have built-in spacing tokens concept.
- **Component reuse:** Build widgets that encapsulate these designs. For example, make a `PrimaryButton` widget that automatically uses the correct styling so that developers don't inadvertently style each button differently. Same for Cards, perhaps a `BuycottCard` that has the correct padding and corner by default.
- Use Flutter's Material 3 capabilities if possible, as it allows specifying many tokens (like outline color, surface tint for elevation overlay, etc.) that we can align with our system.
- **Codex/Automation Enforcement:** We aim to use automated checks (maybe even AI assistants) to ensure compliance:
- **Color checks:** A script can parse Dart code for any `Color(` or hex strings and verify they belong to the approved set. Only exceptions would be color definitions in the theme setup. This catches any stray usage.
- **Text style checks:** Search for TextStyle usages where fontSize or fontWeight is set. Ensure they match token values or use theme references. For example, if `TextStyle(fontSize: 15)` appears, that's not allowed. Similarly, if `fontFamily` is set to something not "Inter" or system default, flag it.
- **Widget usage:** If we provide base widgets (like the PrimaryButton class), encourage usage of those. Codex could review that no raw `ElevatedButton.styleFrom` is used with custom colors – instead, `Theme.of(context).colorScheme.primary` should be referenced. Possibly enforce by code review that design system widgets are used or theme is leveraged.
- **Layout checks:** Some drift is harder to catch with static analysis (e.g., 10px padding). But we can set guidelines: use only multiples of 8 in `EdgeInsets`. Perhaps a lint rule could be written that checks the literal padding values in code for %8==0. While not foolproof (EdgeInsets.symmetric(vertical:10,horizontal:8) might slip one value), it's something to aim for. Developers can be educated to use `EdgeInsets.all(16)` from our constants instead of magic numbers.
- **Design tokens centralization:** All token values (colors, sizes, etc.) should be defined in one place (maybe a `design_system.dart`). This file can be the single source of truth. Codex can ensure any changes to it are reviewed carefully and that usage throughout references it.
- **Testing UI**: We can create screenshot tests of key screens and use them to catch unintended changes. If someone accidentally changes a color or font, the test will show a difference.
- **Documentation and Training:** This spec should be available to all team members. Do regular design reviews for new features strictly comparing against these rules. If something new can't be achieved with current tokens, that signals a need to officially extend the system (with new token) rather than a one-off solution.

By marrying the design system with Flutter's theming and using automated checks, we ensure the design is not just aspirational but actually enforced in the product. Every element on screen can be traced back to these specifications, guaranteeing a **pixel-perfect**, brand-consistent, and accessible experience across the app.

---

1 2 3 4 5 6 7 8 9 10 11 19 Web Design Principles for User-Centric Sites
https://uxpilot.ai/blogs/web-design-principles

12 13 18+ The Best Gradient Background Examples for Modern Landing Pages
https://www.linkedin.com/pulse/18-best-gradient-background-examples-modern-landing-pages-rahman--dfa9c