ChatGPT

Streamlit Layout and Structure

Use Streamlit's built-in layout primitives to build a clean, grid-like app. For example, switch to wide mode via `st.set_page_config(layout="wide")` to use the full page width [1]. Arrange content with `st.columns()` to create responsive columns (e.g. `col1, col2 = st.columns([2,1])`), and place Streamlit elements inside them. Columns automatically resize on different screens [2] [3]. Use `st.expander` for collapsible sections (e.g. long explanations or advanced filters) to maximize vertical space [4]. The example above hides secondary controls in an expander, keeping the main view uncluttered. Group related items with `st.container` (or use the `with` syntax) to create named panels. For example:

```
st.set_page_config(layout="wide")
col1, col2 = st.columns([3,1])
with col1:
    st.header("Main Dashboard")
    # charts, dataframes, etc.
with col2:
    st.header("Sidebar Controls")
    # sliders, selects, inputs
```

Sidebars (`with st.sidebar:`) are ideal for input widgets (filters, checkboxes, form fields), leaving the main area for outputs [5]. Organize the app so that top-level headers (via `st.title` or `st.header`) and containers define a clear hierarchy of sections. Use sub-headers (`st.subheader`) and dividers (`st.markdown("---")`) to break the app into logical chunks. This visual hierarchy – large titles and clear grouping – guides the user's eye down the page [6] [7].

- **Grid Layout:** Use `st.columns` (with width ratios if needed) to align charts and metrics side by side [8] [2].
- **Expanders & Containers:** Hide optional panels (help text, advanced filters) in `st.expander` [4]. Wrap complex sections in `with st.container(): ...` blocks for reuse.
- **Sidebar:** Put all user inputs in `st.sidebar` for a clean vertical flow [5]. The main content area then remains focused on outputs (charts, tables, etc.).
- **Spacing & Breaks:** Use markdown (`st.markdown("---")` or headings) or even empty `st.write("")` calls to separate major sections and avoid cramped layouts [9].

Typography and Sizing

Streamlit's default theme uses a simple sans-serif scale, but you control emphasis via heading commands. Larger text draws attention and creates hierarchy [6]: use `st.title` for the app title, `st.header` for major section headings, and `st.subheader` for sub-sections. For example:

```
st.title("Financial Dashboard")
st.markdown("A high-level overview of key metrics and projections.")
# brief description
```

```
st.header("Revenue Trends")
st.subheader("Last 12 Months")
```

For explanatory notes or less critical info (e.g. sources, footnotes), use `st.caption` [10]. This reserves smaller, lighter text that doesn't compete with main content.

For large numeric values (KPIs, totals), use `st.metric`, which displays the number in a big bold font. If the raw numbers are very large, shorten them (e.g. "1.2M" instead of "1,200,000") to improve readability [11]. For example:

```
from millify import millify
revenue = 12345678
st.metric("Total Revenue", millify(revenue))  # displays "12.3M"
```

Metrics can also show deltas with colored arrows (green up, red down). Use the `delta` and `delta_color` parameters of `st.metric` to indicate changes. Generally keep headings clear and concise, and accompany them with a brief paragraph (`st.write`) to explain the section's purpose.

Color, Spacing, and Visual Consistency

Adopt a consistent, professional color palette (often brand colors). Streamlit theming lets you set a light or dark base and primary accents via the config file (in Community Cloud, use `st.set_page_config` for icon/ title and custom CSS for deeper control). Ensure sufficient contrast: text should stand out sharply from its background [12]. For example, don't put dark text on a dark gray. Use light grays or whites as backgrounds, with one or two accent colors (e.g. for buttons or key chart lines). Streamlit's built-in theming applies a pleasing default palette to charts, but you can override it by supplying your own colors to Plotly or Altair if needed.

Maintain consistent spacing: give charts and inputs breathing room. In Streamlit you can wrap content in a `<div>` with padding or margins via HTML in markdown. For example:

```
st.markdown('<div style="padding:20px;"></div>', unsafe_allow_html=True)  # vertical space
```

Or insert a lightweight separator: `st.write("---")` or `st.write("#")` to add vertical gaps [9]. Avoid cramming widgets or text flush against each other—use columns and containers to add gutters and white space. For responsive grid layouts, adding `gap-4` or similar Tailwind classes (when using Tailwind) helps spacing. In summary: consistent padding between elements (e.g. 1rem to 2rem) gives a clean, airy feel.

Icons and Indicators

Rather than rely on Streamlit's default Material Icon font (which can fail to load), use inline SVG icon libraries. Popular choices include **Heroicons**, **Feather Icons**, **Lucide**, or **Tabler Icons**. These are open-source SVG sets (e.g. Heroicons is MIT-licensed [13]) that you can embed directly. For example, copy a Heroicon SVG snippet into an `st.markdown` or `components.html` call:

```
st.markdown("""
<svg xmlns="http://www.w3.org/2000/svg" class="h-6 w-6" fill="none"
viewBox="0 0 24 24" stroke="currentColor">
  <path stroke-linecap="round" stroke-linejoin="round" stroke-width="2"
        d="M9 5l7 7-7 7" />
</svg>
""", unsafe_allow_html=True)
```

This renders a right-arrow icon without external dependencies. You can adjust its size and color via attributes or parent CSS. Use these SVG icons for buttons, menu indicators, or stat labels. They are more reliable than icon fonts and work offline. For example, replace a “keyboard_arrow_right” expander indicator with a custom SVG to avoid fallback glitches. Feather and Tabler have similarly easy SVG sprites you can embed (their sites allow customizing size and color before copying the code).

Charts and Tables

Leverage Streamlit’s support for Plotly and Altair (both free). By default, Streamlit applies a unified theme to charts, so Altair/Plotly charts match your app’s style [14]. For example:

```
import altair as alt
chart = alt.Chart(data).mark_bar().encode(x="date", y="revenue")
st.altair_chart(chart, theme="streamlit", use_container_width=True)
```

The `theme="streamlit"` makes the chart use Streamlit’s blue accent and fonts [15]. You can override colors if needed by specifying your color scales. Plotly charts also inherit the theme by default. For a consistent look, limit yourself to the theme’s color palette and font.

For tables, prefer `st.dataframe` (interactive) over `st.table` (static) when possible. You can pass a pandas `Styler` to `st.dataframe` to color cells or format values [16]. For example:

```
st.dataframe(df.style.format({"Amount": "${:,.
2f}"}).highlight_max(color="lightgreen"))
```

This allows conditional formatting. For advanced features (sorting, filtering), use a community component like streamlit-aggrid. It renders a full-featured grid with Bootstrap styling. Otherwise, as a hack you can create an HTML table (or embed a small web widget) via `st.components.html(...)`, though built-in dataframes are usually sufficient for financial dashboards.

CSS and Styling Pitfalls

Streamlit’s layout uses generated CSS classes (often random names) and limited pseudo-classes. Don’t use unsupported selectors (e.g. `:contains()` in CSS) or rely on internal `data-testid` attributes—those can change between versions. In general, **custom CSS is not officially supported**, and widget class names can change after an upgrade [17]. If you inject CSS, do it defensively: for example, use

broad containers or the official `.stBlock` or `.stMetric` classes documented by Streamlit rather than obscure names.

To safely inject styles, use `st.markdown` with `unsafe_allow_html=True` or the `st.components.v1.html()` function. For example:

```
st.markdown("""
<style>
  .stButton>button {background-color: #4A90E2; border-radius: 8px;}
</style>
""", unsafe_allow_html=True)
```

This CSS snippet is wrapped in a `<style>` tag inside markdown. Be aware that after a Streamlit update you might need to adjust selectors. Avoid heavy hacking like removing spans via `nth-child` selectors on unknown classes. Also avoid inline JavaScript or fonts that need additional requests, since Community Cloud may block some resources.

Finally, steer clear of internal identifiers: e.g. do not parse or use Streamlit's `data-testid` values for styling or logic, as those are intended for testing and may be unstable.

Reusable Components

Encapsulate repeated UI patterns as Python functions or HTML templates. For example, you can build a **card** by writing a small HTML block with borders and padding via `st.components.html` or `st.markdown`. For instance:

```
import streamlit.components.v1 as components

def kpi_card(label, value, help_text=""):
    components.html(f"""
      <div style='padding: 12px; border-radius: 6px; box-shadow: 0 1px 3px
rgba(0,0,0,0.1);
                  background: white; margin: 5px; display:inline-block;'>
        <strong style='font-size:24px;'>{label}</strong><br/>
        <span style='font-size:20px;'>{value}</span><br/>
        <small style='color:gray;'>{help_text}</small>
      </div>
    """)
# Usage
kpi_card("ROI", "8.5%", "Return on Investment")
```

Above, `components.html` renders a standalone HTML fragment for each card. Alternatively, you can use `st.html` (introduced in newer Streamlit versions) to inject HTML without an iframe [18]. This way you can define "card", "badge", or "alert" templates once and reuse them throughout.

In pure Streamlit, you can also use `st.container` or `st.columns` along with markdown to simulate cards. For instance, using the st_tailwind wrapper, one can write:

```
import st_tailwind as tw
tw.initialize_tailwind()
with tw.container(classes="w-48 p-4 bg-white rounded-lg shadow"):
    st.metric(label="Accuracy", value="92%")
```

This Tailwind-based example shows how to create a reusable KPI card in code [19]. In general, group each reusable section (e.g. side-panel, info box) into a function that takes parameters and outputs Streamlit elements.

Design Testing and Maintenance

Finally, treat your app's UI as code that needs testing. Use visual regression testing to catch layout breakages. For example, the SeleniumBase framework can capture a "baseline" screenshot of your app and compare it on each test run [20]. Automating this means you'll get alerts if a code change (or library update) shifts your charts or text unexpectedly. As noted by Streamlit's own testing guide: visual regression tests "help detect when the layout or content of an app changes, without requiring the developer to manually inspect" [20]. In practice, add a CI step that runs your app (e.g. via Selenium) and checks that key elements (titles, metrics) are still present and that screenshots match the baseline. This ensures long-term consistency [21].

For daily debugging, isolate layout issues by simplifying the page. Temporarily remove sections or use dummy data to focus on one component. Use the browser's developer tools (inspect the DOM) to diagnose padding/margin quirks. Try your layout at different widths to ensure it doesn't break on smaller screens (Streamlit columns stack on narrow viewports automatically, but embedded HTML might not). Keep your Streamlit version pinned in requirements to avoid surprises, and re-test after any Streamlit upgrade.

By organizing your app with these principles—clear grid layout, consistent styling, reusable card components, and automated visual checks—you'll create a polished, investor-ready financial dashboard. Each decision (from colors to fonts to layout flow) should reinforce clarity and trust in the data being presented.

Sources: Streamlit documentation and tutorials [1] [2] [4] [22] [11] [14] [16] [17] [23] [20] [21] (official blog posts and API guides).

---

[1] [5] [6] [7] [8] [10] [12] [22] App Layout & Style Tips | Designing Apps for User (Part II)
https://blog.streamlit.io/designing-streamlit-apps-for-the-user-part-ii/

[2] [3] [4] App Layout Primitives: Columns, Containers & Expanders
https://blog.streamlit.io/introducing-new-layout-options-for-streamlit/

[9] Create empty space to separate portions of the app - Using Streamlit - Streamlit
https://discuss.streamlit.io/t/create-empty-space-to-separate-portions-of-the-app/8689

[11] st.metric - Streamlit Docs
https://docs.streamlit.io/develop/api-reference/data/st.metric

[13] Heroicons
https://heroicons.com/

14 15 A new Streamlit theme for Altair and Plotly charts
https://blog.streamlit.io/a-new-streamlit-theme-for-altair-and-plotly/

16 st.dataframe - Streamlit Docs
https://docs.streamlit.io/develop/api-reference/data/st.dataframe

17 Css customization on streamlit - Using Streamlit - Streamlit
https://discuss.streamlit.io/t/css-customization-on-streamlit/60145

18 23 st.components.v1.html - Streamlit Docs
https://docs.streamlit.io/develop/api-reference/custom-components/st.components.v1.html

19 New Component: streamlit-tailwind, simplify the process of creating user interfaces - Custom Components - Streamlit
https://discuss.streamlit.io/t/new-component-streamlit-tailwind-simplify-the-process-of-creating-user-interfaces/79832

20 21 How to Create Automated Visual Tests [SeleniumBase Tutorial]
https://blog.streamlit.io/testing-streamlit-apps-using-seleniumbase/