ChatGPT

Carbon Design System Guide for AI Agents

2x Grid Layout Principles

The Carbon Design System uses the **2x Grid** to ensure consistent, responsive layouts. This grid is based on a **16-column** structure at large breakpoints (≥1056px), with smaller breakpoints using 8 or 4 columns [1]. In practice, this means designing layouts that align to a 16-column grid on desktop, 8-column on tablet, and 4-column on mobile. All content should sit within these columns, with defined gutters between them, to prevent arbitrary placement and overlap. The base **"mini unit"** for the grid is **8px**, which is the foundation for all sizing and spacing in Carbon [2] [3]. By using multiples of this base unit (8px, 16px, 24px, etc.), the UI will naturally align to the grid and maintain visual harmony.

Guidelines: - **Use Carbon's Grid System:** Leverage Carbon's layout components or CSS classes instead of custom CSS for overall page structure. For example, Carbon provides `<Grid>` containers with `<Row>` and `<Column>` elements (in React and web components; Svelte can utilize the same CSS classes or equivalent components) to create responsive layouts [4]. Always structure pages with these grid patterns to ensure consistency. If an official Svelte `<Grid>` isn't available, you can still apply Carbon's grid classes (like `bx--row`, `bx--col-lg-16`, etc.) or copy the vanilla HTML approach from Carbon's web components tutorial for Svelte integration. - **Responsive Columns:** Design your layout to **reflow at standard breakpoints**. At the large (lg) breakpoint and above, use up to 16 columns; at medium (md) widths use up to 8 columns; at small (sm) widths use 4 columns [1]. This ensures the interface adapts without breaking. For example, a sidebar might span 4 of 16 columns on desktop, but on small screens it should span the full width (4 of 4 columns) below the main content. - **Consistent Gutters and Alignment:** The space between columns (gutters) in Carbon's default grid is 16px on narrow and 32px on wide layouts [5] [6]. Make sure to include appropriate side padding or margins in containers so content isn't glued to the viewport edge. Typically, content is centered or left-aligned within a max width (1584px at max breakpoint) with consistent margins [7]. Adhering to these grid constraints will prevent elements from overlapping or appearing misaligned.

Spacing and Layout Consistency

Proper **spacing** is crucial for a clean, usable UI. Carbon provides a **spacing scale** that complements the 2x Grid, using increments of 2px, 4px, and primarily 8px as the base [8]. Rather than using arbitrary CSS values, use Carbon's **design tokens** for spacing to keep everything aligned to the 8px rhythm. Each token (e.g. `$spacing-03`, `$spacing-05`, etc. or their CSS variable equivalents like `--cds-spacing-03`) corresponds to a specific pixel value. For example, $spacing-01 = 2px, $spacing-02 = 4px, $spacing-03 = 8px, $spacing-04 = 12px, $spacing-05 = 16px, $spacing-06 = 24px, and so on up to large values [9] [10]. These allow you to apply uniform padding and margins that scale consistently.

Guidelines: - **Use Spacing Tokens:** Avoid hard-coding magic numbers for margins or padding. Instead, use Carbon spacing tokens or the CSS custom properties provided by the Carbon theme. For instance, in your Svelte styles you can use `padding: var(--cds-spacing-05)` for a standard 16px padding, or `gap: var(--cds-spacing-03)` for an 8px gap. This ensures all components use consistent spacing values drawn from the design system. Carbon's 8px base means most spacing will be in multiples of 8px (or sometimes 4px for tight elements) [11]. - **Maintain Vertical Rhythm:** When stacking sections vertically (e.g. a toolbar below a header, or cards in a column), use consistent spacing between them – typically one of the spacing token values (like 16px or 24px, depending on the design density). This creates a regular vertical rhythm and prevents components from colliding or overlapping. According to Carbon guidelines, all spacing between elements should be defined by the base unit multiples (1x, 2x, 3x, 4x, etc.) [3] . For example, using a **2x (16px)** or **3x (24px)** spacing consistently between sections will visually group related content and separate different sections clearly. - **Avoid Random or Inconsistent Gaps:** Audit the UI for any unusual padding or margin values that don't fit the 8px grid (such as 5px, 10px, 20px, etc., which are off-scale). These should be replaced with the nearest Carbon token value. Consistency in spacing not only looks better but also makes the interface more predictable. It helps users understand the hierarchy and grouping of elements through uniform white space [8] [12] .

Prevent Overlap and Ensure Proper Layout

In a Carbon-based layout, components should flow naturally within the grid without overlapping each other. **Overlapping elements** usually indicate a layout issue – often caused by absolute positioning, negative margins, or not accounting for dynamic content size. To fix this, rely on the grid and flexbox for positioning rather than manual offsets.

Guidelines: - **Use Standard Layout Containers:** Wrap groups of elements in `<div class="bx--row">` and `<div class="bx--col">` (or the Svelte `<Row>`/`<Column>` components) instead of layering elements with absolute positioning. This way, each component occupies its own grid cell or flex item and won't intrude on others. For example, the CHUCC-SQUI toolbar and query editor should be in a structured container that provides the necessary space between them (e.g., margin-bottom on the toolbar or padding in the container) rather than relying on fixed heights. - **Avoid Fixed Heights (Unless Necessary):** If a container's height is fixed or if overflow is hidden incorrectly, content can overlap (e.g., results overlapping a footer). Let containers size to their content or use flexbox with `flex: 1` to distribute space (as done with the SplitPane). Only use position absolute for overlays or truly floating elements, and when you do, ensure a proper z-index and that they are closed/dismissed to not permanently cover other UI. - **Test at Different Resolutions:** Check the UI at various screen sizes to catch any overlap or wrapping issues. For instance, on a narrow screen, the toolbar items might wrap; ensure there is sufficient vertical space so that if (for example) the endpoint selector moves below the button, it doesn't collide with other content. Using the recommended grid breakpoints and spacing will normally handle this. If something overlaps when the window is small, consider adding a media query (or letting Carbon's grid stack the columns) to rearrange the layout (as is already done with the toolbar's responsive CSS in your project). - **Consistent Alignment:** Make sure that components align to the same vertical and horizontal grid lines. Misalignment can happen if you manually add padding that isn't symmetric or if you don't use the grid offsets. Leverage Carbon's grid for alignment – for example, using an empty Column to create a spacer or the `offset` classes to push content – rather than nudging with CSS that might not scale. Consistent keylines and alignments greatly improve the professionalism of the UI [13] [14] .

Leveraging Carbon Design Tokens and Theming

Carbon provides a comprehensive set of **design tokens** (CSS variables) for colors, typography, and components, in addition to spacing. In CHUCC-SQUI, the theming (colors and themes like g90, g100) is already applied, which is great. Continue to use tokens for things like text color (`--cds-text-primary`, etc.), backgrounds (`--cds-layer-01` for base layer, etc.), and interactive elements. This ensures your app will automatically update if the design system updates those values, and keeps dark mode and light mode working consistently.

For typography and iconography: use Carbon’s recommended font sizes and weights (which correspond to tokens like `--cds-productive-heading-01`, etc.) or the defaults provided by Carbon components. Avoid mixing in custom fonts or sizes unnecessarily. The Carbon type scale is designed to pair with the 2x grid and spacing scale, maintaining appropriate line-heights and hierarchy. If you use Carbon’s components (like `TextInput`, `Button`, `Tag`, etc.), they come with the correct font and spacing built-in. Just ensure any custom elements you style follow suit (for example, a custom `<h5>` in your Svelte code can use the class `bx--type-beta` or simply use similar CSS as Carbon’s heading of that level).

Key Point: The aim is to have **uniformity**. Every padding, margin, and position should feel deliberate and consistent with Carbon’s rules. By using the grid and spacing tokens, you eliminate guesswork and avoid UI issues such as elements overlapping or having inconsistent gaps.

Task: Fix Carbon Design Implementation Issues in CHUCC-SQUI

Status: *Open* (High Priority)
Summary: The CHUCC-SQUI interface needs an audit and updates to fully align with Carbon Design System guidelines. Several layout and spacing inconsistencies have been identified – such as improper grid usage, overlapping elements, and irregular padding/margins – which should be corrected to improve UI consistency and responsiveness.

Problem Outline

- **Inconsistent Grid Layout:** The app is not strictly following Carbon’s 2x Grid structure. Some pages or components aren’t contained within the recommended column grid, leading to elements that stretch or misalign on large screens. There may be a missing max-width or gutter, causing the layout to look different from Carbon’s standards.
- **Irregular Spacing:** Margins and padding between components use non-standard values in places (not adhering to the 8px base increment). This results in uneven whitespace and a visual impression that some sections are cramped while others are too spaced out.
- **Overlap and Alignment Issues:** In certain scenarios, components overlap or don’t align properly (for example, a floating element might cover another, or items aren’t lining up on the same baseline). This is likely due to ad-hoc CSS (absolute positioning or negative margins) compensating for missing grid structure or spacing.

Requirements (Must Fix)

- **Adopt Carbon Grid in Layout:** Refactor the overall page structure to use Carbon’s grid approach. Wrap the main content areas in a container that has the appropriate padding and max width (e.g., centered 1584px max at desktop). Use a responsive grid (16/8/4 columns) for major regions. For example, the toolbar, editor, results pane, and side panels should all align to the grid columns. This may involve adding a parent `<div class="bx--grid">` and inside it, structuring content in `<div class="bx--row">` and `<div class="bx--col">` elements of specified widths. Ensure that at smaller breakpoints, these columns stack or resize as per Carbon guidelines.
- **Standardize Spacing and Padding:** Go through all components’ CSS and replace hard-coded spacing with Carbon spacing tokens or values from the spacing scale. For instance:
- Use `margin-bottom: var(--cds-spacing-05)` (16px) or similar on sections like the toolbar, to separate it from content below in a consistent way.

- Ensure equal padding inside containers (e.g., cards, modals) using the same token values (8px, 16px, 24px as appropriate). Remove any unusual values (like 13px or 20px) in favor of the nearest 8px-based value.
- Maintain consistent **vertical spacing** between stacked elements and **horizontal spacing** in inline layouts using the design tokens (the project already uses `gap: var(--cds-spacing-04)` in places – extend this usage throughout).
- **Eliminate Overlap:** Identify any UI element that overlaps another or extends outside its intended boundary. Remove the CSS causing this – for example, if a component was absolutely positioned to achieve a certain layout, refactor it to be in normal flow within the grid or flex container. All interactive elements (dropdowns, dialogs) that do overlay others should be deliberately handled (with proper z-index and backdrop if needed), and not as a result of layout issues. After refactoring the grid and spacing, there should be no unintended overlaps. Test scenarios like: long query results, many tabs open, window resized to small, etc., to confirm nothing sits on top of something else incorrectly.
- **Align and Size Components Consistently:** Ensure that similar elements use consistent styling. Headings and labels should use the same font sizes and weights defined by Carbon (e.g., all section titles could be `<h5>` with the same class or size). Buttons and form inputs should be the default Carbon sizes (field heights, etc.). If any custom CSS adjusted these, revisit and rely on Carbon defaults for uniformity. Check that icons and text within a line are vertically centered (Carbon's components usually handle this). Use the Carbon **tokens for typography** and **icon sizes** if available, or match the values from Carbon's theme.
- **Test Theming and Contrast:** Since theming is already in place (using `Theme` component and CSS variables), verify that after layout fixes, the **theming still works** (e.g., in g90 dark mode, the backgrounds and text still use the `--cds-*` variables). Also, after spacing adjustments, confirm that contrast is sufficient – e.g., no text gets too close to other elements or edges. All text should have the recommended padding around it for readability (at least 16px on dialogs, etc., per Carbon) [15] [16].

Implementation Plan (Suggestions)

1. **Introduce a Grid Container:** In the Svelte root component (e.g. `SparqlQueryUI.svelte` or a parent wrapper), add a Carbon grid container. For example:

```
<div class="bx--grid">
  <div class="bx--row">
    <!-- your content, possibly multiple columns -->
    <div class="bx--col-lg-16 bx--col-md-8 bx--col-sm-4">
      ... main content ...
    </div>
  </div>
</div>
```

   This will ensure a padded, centered layout. Alternatively, if not using Carbon's CSS classes directly, manually apply equivalent styles: e.g., `max-width: 99rem (1584px); margin: 0 auto; padding: 0 2rem;` at large breakpoints [7].
2. **Audit CSS for Spacing:** Go through the `.svelte` component style blocks and replace any non-multiple-of-8 values. Many can likely be replaced by existing CSS variables (the project already uses `--cds-spacing-03`, etc.). Make use of tokens like `spacing-05 (16px)` for standard gaps. A quick strategy: search for `px` values in the code and verify they match 4, 8, 16, 24, 32... If not, adjust them. Also check any `margin` or `padding` using rem units (0.125rem, 0.25rem,

0.5rem, 0.75rem, etc., correspond to Carbon tokens). Align those to the nearest token if they aren't already exact.

3. **Remove Hacky Positioning:** If any component was given position absolute or fixed to achieve a certain positioning (aside from purposeful overlays like modals or tooltips), remove those styles. Instead, use Carbon's layout utilities. For example, if two components needed to sit side by side, wrap them in a `bx--row` rather than absolutely positioning one next to the other. This step might involve minor HTML structure changes to introduce container divs that apply `display: flex` or the Carbon grid classes.
4. **Verify with Storybook/Examples:** If you have Storybook stories or can run the app, review the interface after changes. Pay attention to the **Toolbar and Tabs** area (should have proper padding so it's not clinging to the top or sides of the container), the **Results table section** (should scroll without overlapping other panels), and **side panels or dialogs** (should overlay correctly with backdrop and not push content underneath unexpectedly). Use Carbon's examples as a reference – e.g., compare spacing around a Carbon `Modal` or `DataTable` in official docs to your implementation and adjust accordingly.
5. **Regression Testing:** After making these changes, test all major features in both light and dark themes. Ensure that all text is legible (spacing fixes can sometimes reveal if a text was accidentally using the wrong color token, etc., but since you're using Carbon tokens for text, this should be fine). Also test on different screen sizes to confirm the responsiveness holds up – the grid should handle most of it, but check edge cases like a very long endpoint URL in the endpoint selector (should truncate with ellipsis as it does, without breaking layout) or extremely wide result tables (should scroll within their container, not overflow out).

Acceptance Criteria

- The application layout uses Carbon's 2x Grid conventions: content is properly contained and aligned in a responsive 16/8/4 column grid. No content stretches full-width on large screens without margins, and overall page width is capped for readability (except in high-density full-width modes if deliberately chosen).
- All spacing between UI elements is consistent and based on the Carbon spacing scale. There are no visual anomalies of extra-large or small gaps; padding inside components and margin between them follow a logical rhythm (8px, 16px, 24px... as appropriate).
- No unintended overlaps occur at any screen size. Every component has sufficient breathing room and is layered correctly. For example, notifications or dropdowns that overlay are rendered via Carbon components (which handle z-index), and they do not cause other content to reflow incorrectly or hide permanent content.
- The interface **feels "Carbon":** by using the Carbon grid and spacing, the UI should now resemble the design system's style – clean lines, appropriate whitespace, aligned elements, and a consistent theme. This will improve not just aesthetics but also usability and accessibility (spacing impacts how easily users can click targets and scan content).
- Documentation is updated if needed (if any README or docs sections prescribe certain layout or theming usage, update them to reflect these changes). Developers should be informed (via code comments or docs) to continue using the Carbon grid and tokens for any new UI work.

References

- Carbon Design System – **2x Grid** guidelines [3] [1]
- Carbon Design System – **Spacing** guidelines [8] [9]
- *Carbon Components (Svelte)* – Grid and theme usage examples [4] [11]
- Internal UI/UX Agent notes – emphasis on using Carbon spacing tokens and layout best practices [17] [18]

[1] Understanding how the Grid System works · Issue #930 - GitHub
https://github.com/carbon-design-system/carbon-components-svelte/issues/930

[2] [3] IBM Design Language – 2x Grid
https://design-language-website.netlify.app/design/language/2x-grid/

[4] [11] Building Modern Web Applications with Next.js and Carbon Design System | by Basukori | Nov, 2025 | Medium
https://medium.com/@basukori8463/building-modern-web-applications-with-next-js-and-carbon-design-system-9468bb3c6cd0

[5] [6] [7] [13] [14] info icon
https://carbondesignsystem.com/elements/2x-grid/usage/

[8] [9] [10] [12] carbondesignsystem.com
https://carbondesignsystem.com/elements/spacing/overview/

[15] [16] [18] ResultsPlaceholder.svelte
https://github.com/arne-bdt/CHUCC-SQUI/blob/ed4c667d04863f1668d5077f36a3aa3186ebbdd7/src/lib/components/Results/ResultsPlaceholder.svelte

[17] ui-ux.md
https://github.com/arne-bdt/CHUCC-SQUI/blob/ed4c667d04863f1668d5077f36a3aa3186ebbdd7/.claude/agents/ui-ux.md