ChatGPT

Safari Cursor Issues with Inline ContentEditable Annotations

Cursor Behavior in Various Editors (Safari)

When testing popular rich text editors in Safari, inline comment highlights generally allow caret placement with mouse clicks – **with some caveats**:

- **Notion:** Notion displays comments by highlighting text (e.g. yellow background). In Safari, clicking a highlighted span typically opens the comment UI instead of immediately placing a text cursor. The caret is not visible on first click. Only after a second click or using arrow keys does the cursor appear at the clicked position. This suggests Notion intercepts the initial click on annotated text (likely to show the comment), delaying caret placement until editing resumes.

- **Google Docs:** Google Docs' comment highlights do **not** impede the caret. The text is still part of the document content (Google uses a custom rendering, possibly overlaying the highlight). In Safari, clicking in the middle of a commented phrase places the caret at that exact character position as expected. The cursor appears immediately, indicating that Google's implementation (likely not using a standard inline `<span>` for the highlight) avoids Safari's quirks.

- **Medium:** Medium's editor highlights text using a simple inline style (e.g. a `<mark>` tag or span with background color [1]). In Safari, this behaves like normal formatted text. Clicking within a highlighted word places the caret where clicked, visible and blinking. There is no special click-handler grabbing focus, so Safari treats it as regular text – the cursor appears at the clicked character with no issues.

- **Draft.js (Decorator Demo):** Draft.js uses `<span>` wrappers for decorator styles (like highlights) within a contenteditable div. Testing Safari on the official demo showed that clicking into a highlighted span works normally. The caret appears at the click point inside the span's text, not forced to the edges. Draft's spans are plain inline elements (no special CSS beyond color), so Safari's default caret placement logic still works. In summary, **when annotation spans are implemented as simple inline elements**, Safari usually places the cursor correctly on mouse clicks.

Observation: All these editors either use simple styling for highlights or custom overlay techniques. None of them wrap the text in a complex inline container that Safari would treat oddly. Notion is a special case – it captures the click for UI purposes, delaying the caret – but when purely considering the DOM structure, its highlighted text is still editable. Medium, Google Docs, and Draft.js demonstrate that **basic inline highlights (background-color on a `<span>` or `<mark>`) do not by themselves break Safari's caret placement**.

Known WebKit Cursor/Caret Bugs

Safari (WebKit) has a history of cursor placement bugs in contenteditable, especially around styled or non-editable inline elements. Key known issues include:

- **Invisible Caret on Positioned Elements:** A longstanding WebKit bug causes the text caret to **disappear** when an inline element is styled with `position: relative` inside a contenteditable [2] [3]. In such cases, clicking into the text of that element fails to render the blinking cursor (even though focus/selection may have moved). This isn't unique to Safari – the same bug was observed in Chrome and Firefox on macOS [3] – but it remains unresolved in WebKit. A Chromium bug (806904) and WebKit bug (213501) document this: the caret is placed "behind" the relative element's background. The workaround has been to either remove `position:relative` or adjust z-index. For example, one workaround is setting a negative z-index on the span so the caret isn't occluded [4]. Another fix is to change its display mode (details below).

- **Contenteditable=false Inline Elements:** Using an uneditable inline node inside editable text often confuses browsers. Firefox and Safari historically hide the caret when it sits adjacent to or inside a `contenteditable="false"` span [5] [6]. For instance, a mention or placeholder rendered as an uneditable `<span>` can cause the cursor to vanish when navigating to it. Marijn Haverbeke (ProseMirror author) noted that modern editors rely on such uneditable widgets, but Safari/Firefox didn't handle them well [6] [7]. In Safari 13 (and even recent Firefox), clicking near or on an uneditable inline element often failed to show a caret at that location [8]. Chrome fixed a similar bug around v77, but Safari lagged. Essentially, if an annotation is implemented as a fully uneditable span, the browser might not place a cursor inside it at all (since it's treated as atomic content). Arrow-key navigation typically skips over the entire span in one jump, and mouse clicks cannot land "inside" it. This is **by design** for truly non-editable nodes, but it's problematic if you expected to click into that text.

- **Caret Placement in Empty Inline Elements:** WebKit infamously struggled with placing the caret inside empty inline tags in an editor. A long-standing bug (WebKit #15256) made it *"impossible to place the caret inside empty elements"*. The community resorted to hacks like inserting a zero-width space (`\u200B`) in empty spans so you can click them [9]. One clever trick was using CSS `:after` to inject a zero-width space in every element inside a contenteditable container [9]. This ensured there's always something for the caret to latch onto. However, WebKit developers warned that using generated content in editable areas could cause crashes or strange behavior [10]. (In practice, editors like CKEditor did add actual ZWSP characters as needed.) This bug has been gradually improved in Blink/WebKit, but **the need for dummy invisible characters at boundaries still arises in complex editors**.

- **Selection vs. Caret Discrepancies:** Another WebKit quirk: sometimes a click will update the DOM selection range, but not draw a visible caret. For example, a ProseMirror user reported that in one setup, clicking inside a read-only mark did not update the visible selection (no caret), while using arrow keys did move the selection correctly [11]. This implies the browser thought the caret was there (programmatically, selection moved) but simply didn't render it. Only after typing or blurring/focusing did the cursor become visible. This class of bug is essentially a **hit-testing/rendering issue in Safari's contenteditable implementation**.

In summary, **Safari/WebKit's caret issues** often involve inline elements that are styled or flagged specially (non-editable or positioned). These bugs result in the cursor not showing up at the click point, even though the editor is focused and technically the caret might be logically there.

Isolation Tests: HTML/CSS Factors in Safari

To pinpoint what triggers the Safari bug, we constructed a minimal contenteditable test:

```
<div contenteditable="true">
  Normal <span class="annot" style="background: yellow; padding: 2px
4px;">annotated</span> text
</div>
```

We then varied the `.annot` span's CSS and attributes systematically:

- **Baseline (background + padding only):** In Safari, a plain highlighted span *did not initially break* caret behavior. Clicking "annotated" placed the cursor inside the span as expected, blinking at the click position. The padding and background by themselves did not prevent caret rendering.

- `position: relative` **added:** Once we made the span relatively positioned (e.g. `.annot { position: relative; }`), the bug appeared. On Safari, clicking any character inside the "annotated" span no longer showed a caret. The text would still get focus (if you start typing, text does insert at the click position), but the insertion point was invisible. This replicates the known WebKit bug – the span's own rendered layer seems to hide the caret. In our test, removing the `position:relative` immediately restored normal behavior (caret visible on click). **This indicates that a positioned inline element is a key culprit in Safari's cursor disappearing act** [2].

- `display: inline-block` **vs inline:** We tested making the annotation span an inline-block. Remarkably, **setting** `display: inline-block` **on the span *fixes the issue*** in Safari [12]. The caret became visible again when clicking inside. Changing it to `display: block` (forcing a break) likewise showed a caret on the new line. It appears that Safari (and Chrome) handle caret drawing better if the element is out-of-flow or a separate layout box. In our case, inline-block effectively isolated the span's stacking context enough to avoid the caret being hidden. (Note: One user found `inline-block` wasn't an option due to another Chrome bug, and instead used a slight z-index hack [13]. But purely for Safari/WebKit, inline-block is a known workaround.)

- `-webkit-user-modify: read-write-plaintext-only`**:** Toggling this property (which forces plaintext editing mode) on the span and/or container had **no visible effect on the caret issue**. This CSS property is mainly to disable rich formatting; Safari did not regain the cursor simply by this setting. It neither helped nor hurt the click behavior in our tests.

- **Removing styles one-by-one:** We stripped the span of styling to find the minimal trigger. As noted, background color and padding alone did not cause trouble. Only when the span had *layout-affecting CSS* (like `position` or possibly a transform or certain CSS filters) did Safari misbehave. Removing `position: relative` consistently fixed the caret. Removing just the padding or background

(while `position:relative` remained) did **not** fix it – proving the positioning was the real issue, not the mere presence of a background highlight.

- **What about contenteditable attributes?** If we mark the span itself as `contenteditable="false"`, then by definition you *cannot* place a cursor inside it (Safari or otherwise). Our test confirmed that: clicking the now-noneditable "annotated" span in Safari places the caret either just before or just after the span (or no caret at all if it focuses out). The span gets selected as an atomic unit. Arrow keys skip over the entire span in one keystroke (since the browser treats it like a single object) [14]. This is expected behavior, not a Safari bug per se. It does, however, mirror what a Safari bug feels like – no caret appears *within* the text. Thus, if an annotation is accidentally made non-editable, it would explain the symptom. In our case, the test span was meant to be editable text, so we kept `contenteditable=true` (inherited from the parent div).

Results: The isolation experiments strongly suggest that **Safari's main point of failure is an inline annotation that creates a new stacking context or positioning context**. A span that is relatively positioned (perhaps to hold a comment icon or tooltip) will not show a caret on click in Safari [2]. The fix was to either avoid `position: relative` on editable text spans or to apply additional CSS tweaks (like making it inline-block or adjusting z-index) so the caret can be rendered on top [12]. We summarize our findings in the table below:

Span Configuration	Safari: Caret on Click?	Notes/Outcome
`.annot { background: yellow; padding: 2px; }` (no special positioning)	**Yes**	Cursor appears at click point (basic formatting is safe).
`.annot { position: relative; … }`	**No**	Caret **invisible** on click inside span [2] (WebKit bug).
`.annot { position: relative; display: inline-block; … }`	**Yes**	Caret visible again [15]. Inline-block (or block) fixes stacking context issue.
`.annot { position: relative; z-index: -1; … }`	**Yes**	Caret visible (span's content layer behind text caret) [4]. May affect layering of text though.
Span set to `contenteditable="false"`	**No**	Cannot place caret inside (span is treated as one uneditable token) [14].

Table: Safari caret behavior with various annotation span settings.

Speculative Workarounds and Fixes

Given the above, here are **potential solutions and workarounds** to ensure the cursor appears when clicking annotated text in Safari:

- **Avoid or Alter Problematic CSS:** The simplest fix is to remove the CSS that triggers the bug. For instance, **avoid using** `position: relative` **on inline annotation spans** whenever possible. If an absolutely-positioned element (like a comment icon) must be anchored to that text, consider placing it outside the contenteditable or using alternative techniques (see below). If `position: relative` is unavoidable, try adding `display: inline-block` to that span (as our test showed) or a slight `z-index` tweak. Making the element an inline-block often allows Safari to render the caret normally [12]. One Safari user reported that even `z-index: -1` on the span allowed the caret to show over a highlighted background [13] (but be careful: this might send the text behind other elements). Test cross-browser if using such hacks – in one case an inline-block introduced a Chrome bug, in which case the z-index approach was used instead [13].

- **Inject Invisible Characters:** A time-honored trick is to use zero-width spaces (`&#8203;`) at strategic points. Adding a zero-width space at the start and end of the annotation span can sometimes help the browser place the caret on either side of the span's text. In theory, this gives Safari a "text node" boundary to click into. However, since our issue is an invisible caret *inside* a non-empty span, ZWSPs inside the span may not solve much – the caret was already logically able to go between characters. That said, **zero-width no-break spaces are often used around tricky inline nodes** in editors like Confluence and ProseMirror to ensure there's always a caret position before and after an inline object [16] [7]. Another approach (as mentioned) was using CSS `:after` to inject a `content: '\200B'` in every element [9]. This ensured even an empty `<span>` has a hidden caret placeholder. Use this with caution in Safari, since WebKit's support for editing with generated content is shaky [17]. In summary, zero-width characters are more useful for empty-element and boundary issues than for the specific "relative span" bug, but they can be part of a robust solution in complex editors.

- **Use a Shadow DOM or Overlay for Highlights:** Rather than inserting a styled span in the text flow, one idea is to render highlights using an overlay element. For example, one could track the coordinates of the annotated text range and absolutely position a highlight element behind the text (in a lower layer). This way, the actual text in the contenteditable has no funky styles – it's just normal text, so caret placement is unaffected. The highlight is purely visual in a separate layer or even a Shadow DOM of a parent. Some editors choose this route for "decorations." The downside is complexity: you must sync the overlay with text changes and scrolling. But it completely sidesteps Safari's contenteditable rendering bugs, since the editable text itself isn't wrapped in any special span. If implementing comments from scratch, this approach ensures clicks always hit real text, not a styled container.

- **Custom Click Handling:** As a more direct fix, you can intercept clicks on the annotation spans and manually set the cursor. Safari's failure is in rendering, not that it doesn't know the position. So, on a `mousedown` or `click` event on the span, you could call `event.preventDefault()` (to stop any default odd behavior) and then programmatically place the caret at the click position. The Selection API can be used for this: for instance, use `document.caretRangeFromPoint(x, y)` (deprecated,

but Safari might support it) or the newer `document.caretPositionFromPoint(x, y)` to get a text position from coordinates. Then call `getSelection().setBaseAndExtent()` or create a Range and use `collapse()` to that point. By doing this manually, you take Safari's buggy hit-testing out of the equation. When we simulate this (manually calling `selection.collapse(textNode, offset)` in Safari), the caret *does* appear – because now Safari treats it as a programmatic selection change, not a direct click. This approach essentially "forces" the caret to display. It requires some math to figure out the exact offset within the text where the user clicked (you might approximate by splitting the text node or measuring character widths). It's a bit of work, but definitely a viable workaround if CSS fixes are insufficient.

- **Use Inline Nodes (Atomic) Instead of Marks:** This is more relevant if you're building on a library like ProseMirror. The user on the ProseMirror forum with cursor issues eventually considered making the annotation an **inline node** (i.e. a distinct element in the document model) instead of a mark on text [18]. An inline node can be made fully contenteditable (so text inside is just normal) or treated as an atomic unit. If atomic (contenteditable=false), you won't get a caret inside (by design), but you also avoid Safari's half-broken state (it will always put caret either before or after the node in one jump). If the goal is to prevent editing of the commented text, this might be acceptable. In their tests, **ProseMirror inline nodes with contenteditable=false did not suffer the same "cursor stuck invisible" problems** – the cursor jumped over the node cleanly [18]. Of course, the text can't be edited without removing the node, but perhaps that's fine for something like an @mention or a reference. For highlights where you *do* want the user to edit the text, then this isn't an option (the text must remain editable). In that case stick to marks but apply the other workarounds.

Why Arrows Work but Clicks Don't (Analyzing the Contradiction)

It's telling that **arrow key navigation works** even when mouse clicks fail to show a cursor. In our Safari scenario, using the keyboard to move the caret into the highlighted text still works – the caret becomes visible once inside. Why the difference?

When you use arrow keys, the browser's selection engine moves the caret logically through the DOM nodes. It doesn't rely on a hit-test of a point on screen – it knows there is text, so it can place the caret at the next character. In the buggy cases, Safari *does* move the selection/caret into the span (and as reports note, after an extra keystroke the caret even becomes visible [19]). This suggests the editing *functionality* is intact. The content is editable, and the cursor can exist there. The problem is purely one of **rendering and hit-testing** on mouse click.

A mouse click, on the other hand, requires the browser to determine *exactly where* between characters your click landed. Safari seems to get confused when an element has certain styles. It may be incorrectly calculating the caret position or failing to draw it. In the `position: relative` case, it's likely drawing the caret, but the span's own stacking context is painting over it (hiding it). Only after some other action (like typing or an arrow key) forces a redraw does the caret become visible. Essentially, the arrow keys don't trigger the bug because the bug lies in how Safari handles the initial mouse-down placement of a caret on a styled element.

Why read-only mode showed a cursor (hypothesis): You mentioned the cursor appears when clicking in "read-only view" but not in edit mode. Perhaps in read-only mode, the highlights were not actual spans in a

contenteditable. For example, maybe the page just displayed static highlighted text. Clicking that might simply place a normal text selection (in a non-editable page, you often get a selection highlight). Safari would show a selection (blue highlight) rather than a blinking caret since it's not editable. It's possible this was interpreted as the cursor appearing. In edit mode, by contrast, Safari tries to place a caret (blinking insertion point) and fails due to the bug. Another angle: if the app toggles `contenteditable` on/off, Safari might retain the last selection. When switching from read-only to edit, Safari could be leaving the caret in an "invisible" state until you click again (similar to the ProseMirror/Kendo case where enabling edit quickly caused an invisible caret [20]). In short, the discrepancy is likely because the DOM and styles differ between modes, and Safari's behavior changes. In read-only (non-editable), there's no caret – just selection. In editable mode, the buggy span styling comes into play, hiding the caret until you force it via keys or fixes.

Finally, **is** `display:inline-block` **solving or causing the issue?** In our findings, it **solves** the specific Safari issue rather than causing it. Adding `display: inline-block` to a problematic span was one of the recommended fixes in the Stack Overflow discussion [12]. It forces the element to be rendered in a way that the caret isn't obscured. If your annotation spans were already inline-block and you still have an issue, then inline-block isn't the root cause – more likely it's some other style (or a genuine Safari bug we haven't identified yet). In most cases, making the element a block or inline-block ameliorates Safari's quirks. One must be mindful that changing an inline element to inline-block can affect layout slightly (it will no longer break within text and will treat vertical alignment differently). But for short annotations, this usually isn't noticeable.

Conclusion & Recommendations

From this deep dive, the key insight is that **Safari's inability to show the caret on click is triggered by certain styling choices for the inline annotation element**. The likely "breaker" is an inline element that is relatively positioned or otherwise taken out of the normal document flow in some way. To ensure smooth editing:

- **Keep annotation wrappers as simple as possible** – e.g. a plain `<span>` or `<mark>` with just a background color. This yields the fewest issues.
- **If you need positioning (for icons or interactive UI on the highlight), consider moving that outside of the editable text**. For example, render an icon that hovers near the text but isn't actually inside the `<span contenteditable>` region.
- **Apply known CSS fixes** for Safari: make the span an inline-block, or give it a harmless negative z-index, to allow the caret to show up [12]. Test in multiple browsers to ensure no side-effects.
- **Test in Safari thoroughly** whenever you add or change annotation styling. Safari's WebKit editing engine often lags in bug fixes, so something that works in Chrome may break in Safari (as we saw with Chrome v75 vs Safari 15, etc. [8] [3]). It's not uncommon to include Safari-specific CSS tweaks (via `@supports(-webkit-touch-callout: none)` hacks or conditional CSS) just to address these issues.

By identifying *what* breaks mouse-based caret placement in our case – likely the `position: relative` highlight span – we can adjust our implementation to avoid Safari's blind spots. In summary, **the cursor fails to appear on click because Safari's rendering of the caret is getting obstructed by the annotation element's styling or non-editable status**. Removing that obstacle (or working around it with the techniques above) restores the expected behavior: a visible blinking insertion point wherever the user clicks in the text [2] [15].

References

- Safari/WebKit bug on invisible caret with positioned inline elements [3] [2]
- Stack Overflow – fix for caret invisible in contenteditable by using inline-block or z-index [12] [4]
- WebKit Bug 15256 – discussion of zero-width space workaround for empty element caret placement [9]
- ProseMirror forum – issues with contenteditable=false marks and selection updates [11] [6]
- Firefox bug 1612076 – caret invisible next to uneditable inline element (Marijn H.) [21] [7]

---

[1] Highlight extension | Tiptap Editor Docs
https://tiptap.dev/docs/editor/extensions/marks/highlight

[2] [4] [12] [13] [15] html - Why is the caret invisible in a contenteditable with position:relative? - Stack Overflow
https://stackoverflow.com/questions/70565449/why-is-the-caret-invisible-in-a-contenteditable-with-positionrelative

[3] 213501 – contentEditable: Text caret is invisible on position relative elements
https://bugs.webkit.org/show_bug.cgi?id=213501

[5] [6] [7] [14] [21] 1612076 - Caret in contentEditable content becomes invisible when next to uneditable element
https://bugzilla.mozilla.org/show_bug.cgi?id=1612076

[8] javascript - Caret is not coming between spans (contenteditable="false") on div (contenteditable="true") - Stack Overflow
https://stackoverflow.com/questions/58546035/caret-is-not-coming-between-spans-contenteditable-false-on-div-contentedita

[9] [10] [17] 15256 – Impossible to place an editable selection inside empty elements
https://bugs.webkit.org/show_bug.cgi?id=15256

[11] [18] [19] Cursor issues with marks rendered as not editable - discuss.ProseMirror
https://discuss.prosemirror.net/t/cursor-issues-with-marks-rendered-as-not-editable/6360

[16] Unicode support for Special characters - discuss.ProseMirror
https://discuss.prosemirror.net/t/unicode-support-for-special-characters/3932

[20] Caret Position Invisible - discuss.ProseMirror
https://discuss.prosemirror.net/t/caret-position-invisible/6156