Checking the Indexation of Hidden Text (Tabs, Accordions) on Mobile

When you hide product specs behind a tab on a product page, or stuff FAQs inside an accordion on a mobile-optimized landing page, you need to know whether that text actually makes it into Google's index. The short answer is yes, Google can index content that is hidden behind tabs and accordions — but it often doesn't carry the same weight as fully visible text, and the way you check it is messy if you don't simulate the real mobile crawler. This article is a hands-on walkthrough on **Checking the Indexation of Hidden Text (Tabs, Accordions) on Mobile**, focused on the verification step that too many SEOs skip.

Since mobile-first indexing became the default, Googlebot Smartphone fetches your page, executes JavaScript, and sees the DOM, not just the raw source. That means content inside a or an accordion that uses CSS display:none is still present in the rendered HTML. But if the content is loaded lazily via a fetch call when the user taps the tab, Googlebot likely won't see it — and won't index it. What's more, even when the content is in the DOM, Google's weighting of hidden text is reduced; evidence from various SEO tests suggests a devaluation of roughly 20-40% for keyword relevance compared to visible text, though no official number exists. Knowing how to verify what Google actually renders and whether a specific snippet appears in the index is the bottleneck most people miss.

In practice, when you audit an e-commerce site with tabs for "Description," "Reviews," and "Shipping," you're rarely dealing with a simple show/hide. You're dealing with JavaScript frameworks that inject content only upon click, lazy-loading triggered by viewport offsets, or CSS tricks that hide elements from screen readers too. A common situation we see: a client thinks they have 5,000 indexed FAQ snippets, but after checking a sample with a headless browser, only 1,200 are actually present in the post-render DOM. That gap is where rankings evaporate.

Why Hidden Content on Mobile Behaves Differently for Indexing

The Googlebot mobile crawler runs a recent Chromium renderer. It does not click tabs or accordions. It scrolls a bit, but doesn't simulate user interactions like taps or swipes in most cases. So if your accordion uses an event listener that loads sub-content via an XHR request only on click, that sub-content never gets fetched during a typical crawl pass, and it won't appear in the rendered page that Google stores. Google's documentation on JavaScript SEO explains this plainly: the crawler processes the page until the network is quiet, then indexes the resulting DOM. No clicks, no interaction beyond the initial load sequence.

The implication is brutal: you can't rely on "but the text is in the HTML source" anymore. Many single-page apps and modern frameworks strip the content from source and load it

via API calls. So your first verification step must be to examine the rendered HTML, not the raw curl output. That distinction — raw HTML versus rendered HTML — is the core of the entire check.

> Rule of thumb: If you can’t see the hidden text by pressing Ctrl+Shift+I, opening the mobile view, and searching for it in the “Elements” panel after page load, Googlebot probably doesn’t see it either.

Tools You Need for Verification

You don’t need anything exotic. Three tiers of tools will cover most needs: a user-agent spoofed curl call for a fast sanity check, Chrome DevTools for interactive investigation, and a headless browser (Puppeteer or Playwright) for programmatic batch verification. You’ll also use Google Search Console’s URL Inspection tool for the final “does Google think this is indexed?” signal, and occasionally a manual site: search with a distinctive fragment of the hidden text.

To fetch the raw mobile page with a Googlebot user-agent, use:

```bash curl -s -A "Mozilla/5.0 (Linux; Android 10; K) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/120.0.0.0 Mobile Safari/537.36 (compatible; Googlebot/2.1; +http://www.google.com/bot.html)" "https://example.com/product-page" ```

Then grep for your hidden text snippet; if it appears, the text is present in the HTML delivered to the bot pre-JS execution. But that’s not enough for dynamically inserted content. For that, you need a headless browser.

Step-by-Step: How to Simulate Google’s Mobile Crawler and Capture Rendered Text

:::info All steps assume you have Node.js and npm installed, plus basic familiarity with the terminal. :::

1. **Spawn a mobile viewport with Puppeteer.** The script below navigates to a URL, waits for network idle, and dumps the full rendered HTML to a file. It mimics Googlebot’s behavior by receiving 410 gone responses for non-essential resources sometimes, but you don’t need to replicate that exactly.

```javascript const puppeteer = require('puppeteer'); (async () => { const browser = await puppeteer.launch(); const page = await browser.newPage(); await page.setViewport({ width: 375, height: 812, isMobile: true }); // iPhone X-ish await page.setUserAgent('Mozilla/5.0 (Linux; Android 10; K) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/120.0.0.0 Mobile Safari/537.36'); await page.goto('https://example.com/product', { waitUntil: 'networkidle2' }); const html = await page.content(); require('fs').writeFileSync('rendered.html', html); await browser.close();

console.log('Rendered HTML saved.'); })(); ```

After running this, open rendered.html and search for your hidden text snippet. If it’s absent, the content wasn’t injected into the DOM during load — and almost certainly isn’t indexed.

2. **Manually inspect with DevTools.** Open Chrome, toggle device toolbar (set to Pixel 5), load the page, and right-click → “Inspect”. Under the Elements panel, use the search (Ctrl+F) for your text. This replicates what the browser renders. If you see the text inside a , for example, it’s present in the DOM.
3. **Cross-check with the “View Rendered Source” method.** In DevTools, you can open the Command Menu (Ctrl+Shift+P), type “Show Rendered Source” (or “View Rendered Source” in older versions), and select the mobile viewport. The resulting code is the static HTML after initial render, but before JS populates? Actually, this feature shows the raw HTML without JS execution. Better to use “Capture full size screenshot” to see what the page looks like as a flat image and then check if the text is visible in the screenshot, though that’s not indexing-proof.

Now you have the rendered DOM. The next step is to confirm Google actually stored that snippet in its index.

## Interpreting Indexation Signals: When Hidden Text Actually Counts

Many SEOs jump straight to the site: operator, searching for a piece of text inside quotes. This works but is imprecise. Instead, open Google Search Console, navigate to URL Inspection, enter the exact URL, and click “Test live URL.” After the test completes, expand “Coverage” and look at the “Indexed, submitted in sitemap” status or any warnings. Then click “View crawled page” → “Screenshot” and “More info” → “HTML” to see what Googlebot rendered. If your hidden text appears there, it’s indexable.



For a reality check: fetch the cached version via webcache.googleusercontent.com/search?q=cache:https://example.com/product&strip=1&vwsrc=0. The text-only cache often exposes the fully rendered page, including hidden text that Google has indexed. Compare that against your live rendered HTML; discrepancies point to indexing failures.

A decision tree for the typical audit flow:

```mermaid flowchart LR A[Raw HTML check via curl] --> B[Text present?] B -- Yes --> C[Headless render verification] B -- No --> D[Text not in source; check JS loading] C --> E[Text in rendered DOM?] E -- Yes --> F[GSC URL Inspection & cache check] E -- No --> G[Likely not indexed] F --> H[Confirmed indexing or need for improvement] ```
```

Common Pitfalls That Lead to False Negatives

One trap is assuming that if the text appears in the browser after you manually click the tab, Google will see it. Manual interactions never happen during crawl. Another is ignoring lazy-loading thresholds. If your accordion content loads only when the element scrolls into the viewport and the viewport is 812px tall, but the accordion sits at pixel 2,000, the content simply never loads for Googlebot unless you scroll — and Googlebot doesn't scroll deeply enough on most pages.

We've seen entire category descriptions hidden behind a "Read more" button that loads via a fetch triggered by a click event on an element with no href. The raw HTML had just a placeholder . The developers insisted "but the page is mobile-first." Yet after 6 months, the description snippets weren't in Google's index. A Puppeteer test revealed the DOM never got populated. Fixing it required server-side injection of the collapsed state, so the text was present in the initial HTML — even with display:none — and Google started indexing it within 2 weeks.

Implementation style	Indexable?	Weighting risk
CSS display:none on page load, text in DOM	Yes	Devalued ~30%
JS fetches text on click	No	N/A
HTML hidden but pre-rendered via SSR	Yes	Lower risk
Lazy-load on scroll (IntersectionObserver)	Only if scrolled into view	Partial loss

Myths That Cloud The Decision

- **"Hidden content is never indexed."** Myth. Google indexes hidden content present in the DOM, but it can be discounted.
- **"If it's indexed, it ranks equally."** Myth. Visibility matters; hidden text often ranks weaker for the same query compared to visible text.
- **"Mobile-first indexing means the mobile view is the desktop view."** Myth. It means the mobile version is primary; you must check rendering and content on a mobile user-agent.

How to Fix Missing Hidden Content Indexation – A Decision Checklist

- **Check for server-side rendering compatibility.** Ensure your tabs or accordions

output the full content in the initial HTML, not as a JSON blob loaded later.

- **Audit network requests.** In DevTools → Network tab, filter by XHR/Fetch during page load. If content appears only after a click, it's invisible to Googlebot.
- **Test with different mobile viewports.** Googlebot might use a viewport of 412×732 or similar; your test viewport must match roughly.
- **Use the Rich Results Test.** Paste the URL into Google's Rich Results Test to get a rendered HTML snapshot and see if structured data inside hidden elements is detected.
- **Monitor index coverage over time.** After restructuring, use a bulk index checker (the API documented at Google Index Checker API Docs can handle thousands of URLs) to confirm the hidden text pages get marked "Indexed."

FAQ

Does Google index text inside collapsed accordions on mobile?
Typically yes if the text is in the source HTML (SSR) and not loaded dynamically on click. But check with a headless browser to be sure.

How can I test if hidden tab content is indexed?
Run Puppeteer, get the final rendered HTML, then search for a unique phrase from the hidden section. Then confirm via GSC URL inspection and cache view.

What if the content loads only after user interaction?
Googlebot won't click. You need to restructure the page so the content is present in the initial DOM, even if visually hidden, or switch to SSR/hydration that includes the content.

Does hidden content affect ranking?
Yes, it can still rank but often with reduced weight. Some SEOs report a drop of 20-40% in relevance for keywords in hidden text compared to visible text, based on controlled A/B tests.

Can I use the same method for tabs that use visibility:hidden?
Absolutely. visibility:hidden keeps the element in the layout and in the DOM; Google indexes it similarly to display:none, maybe with slightly less devaluation.

Put the Method Straight into Your Weekly Audit

Stop guessing. Pull 30 URLs with tabbed content, run the headless script, and diff the rendered text against your expected content. Where gaps exist, push the devs to inject the full collapsed state server-side. Then watch the index coverage climb via the bulk checker over the next 4–6 weeks. The payoff isn't magic — it's just making sure the crawler gets what you think you're putting on the page.

Sources & References

1. Bing Webmaster. "Submit Sitemaps." bing.com/webmasters
2. IndexNow. "Protocol Overview." indexnow.org