

The Blueprint

Most documentation stacks make you choose: author in a simple, structured format and get a static site, or hand-write MDX/HTML and get interactivity. This project is a reference architecture for **not choosing** — MyST Markdown stays the single source of truth, and a purpose-built [Kotlin Multiplatform transpiler, `myst2mdx`](#), turns its resolved AST into native Starlight pages, interactivity included.

The page you’re reading is the proof, not a pitch: everything below was authored in MyST, transpiled to MDX by `myst2mdx`, and built by Astro Starlight — the pipeline demonstrating itself with its own output.

Download this page as a PDF ↓ — the same MyST source, exported through `mystmd`’s Typst pipeline; no separate content to maintain. Every page has its own PDF (the topbar link above always points at the one for whatever page you’re on).

1. WHAT “NATIVE” BUYS YOU

The transpiler doesn’t drop down to raw HTML unless it has to. MyST admonitions become Starlight’s own `<Aside>` component — same theme, same dark-mode behavior, no separate CSS to maintain:

Note

This admonition is authored as a MyST directive (`::: {note}`). The transpiler rewrites it to Starlight’s native `<Aside>`, so it inherits Starlight’s styling with no shim.

Danger

MyST has more admonition kinds than Starlight has aside types. They collapse through a table the tool owns — danger and error both map to danger. See [the tool page](#) for the full mapping, demonstrated live.

2. INTERACTIVITY, WITHOUT LEAVING MYST

A plain code fence stays a plain code fence — Expressive Code, same frame and copy button as any other Starlight site. But MyST’s `js-eval` fences are special-cased: the transpiler swaps them for a `<CodeMirrorEval>` island, a sandboxed Web Worker running underneath a real editor. No React, nothing handwritten in the page source below the fence.

```
const fib = n => n < 2 ? n : fib(n - 1) + fib(n - 2);
console.log([...Array(10)].map((_, i) => fib(i)).join(', '));
```

On the live site, that’s a real editor: edit the code and press Run to evaluate it in-browser, off the main thread. In the PDF you’re reading now, it’s necessarily just the

source — Typst has no JavaScript engine to run it against, so the interactivity above is the thing being demonstrated, not something a static export can reproduce.

```
// A normal fenced block goes through Expressive Code, exactly like the
rest of
// the site — same frame, same copy button, same theme.
export const answer = 42;
```

3. NEXT

[Read how the transpiler works arrow.r](#) — the native-vs-fallback contract, the four escaping channels that keep MDX generation from breaking, and why the tool compiles to both a JVM CLI and a JS playground from one shared core.