🏗️

undertext — Tech Stack & Architecture (Shareable)

📌 This is a shareable, non-sensitive overview of undertext 2.0's architecture and deployment approach. It intentionally omits credentials, passwords, account identifiers, and other operational secrets.

the one-line version

> **one core, two faces, every Apple surface.** a contemplative original-language Bible reader — Hebrew, Greek, and the Aramaic Peshitta held one word at a time — built as a single web core, wrapped natively where it needs chrome, and rewritten natively where the screen is small or ambient.

---

the architecture: one core, two faces

the whole system rests on a single decision: **the scripture data is the interface.** every surface — web, the WKWebView apps, and the pure-native watch/TV apps

— reads the same JSON contract from the same origin. nothing embeds its own copy of the text.

🖥️ **Face 1 — big screens (web core in a shell)**
iPhone, iPad, Mac (Designed for iPad), and other large-screen Apple contexts. the existing Vite + React + TypeScript reader, run inside a SwiftUI `WKWebView`.

⌚ **Face 2 — small / ambient screens (pure native)**
Apple Watch and Apple TV. native SwiftUI, no web view, reading the SAME `undertext.org/data` JSON through a shared Swift package.

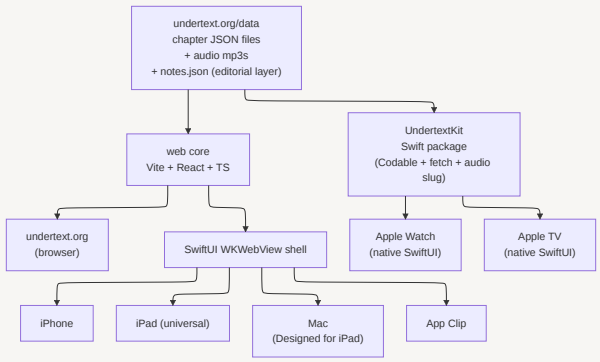
a shared Swift package (**UndertextKit**) centralizes the Codable models + fetch + audio-slug logic so the native surfaces remain consistent with the web reader. **the JSON contract is the interface.**

```
flowchart TD
  DATA["undertext.org/data<br>chapter JSON files<br>+ audio mp3s<br>+ notes.json (editorial layer)"]
  WEB["web core<br>Vite + React + TS"]
  KIT["UndertextKit<br>Swift package<br>(Codable + fetch + audio slug)"]

  DATA --> WEB
  DATA --> KIT

  WEB --> SITE["undertext.org<br>(browser)"]
  WEB --> SHELL["SwiftUI WKWebView shell"]
  SHELL --> IPHONE["iPhone"]
  SHELL --> IPAD["iPad (universal)"]
  SHELL --> MAC["Mac<br>(Designed for iPad)"]
  SHELL --> CLIP["App Clip"]

  KIT --> WATCH["Apple Watch<br>(native SwiftUI)"]
  KIT --> TV["Apple TV<br>(native SwiftUI)"]
```



the tech stack, layer by layer

the web core (`www/`)

- **Vite + React + TypeScript**, ships as static files. the reader **lazy-loads one chapter of scripture JSON at a time**.
- **hash routing** (`#/Book/Ch/Verse`) so every HTTP request is effectively `/`.
- **two search indexes** as runtime assets: an English full-text index and an original-language/transliteration index (lazy-loaded).
- **recorded-voice mp3s** served as static files; tap a word, hear it.
- **accessibility-first** (contrast, Dynamic Type, touch target sizing, correct `lang` tags for scripts).
- an **editorial notes layer** (`notes.json`) that is **tier-gated** (e.g., adopted/review/caution) so alternates are never silently substituted.

the native shell (`ios/`)

- **SwiftUI +** `WKWebView` wrapping the built web core.
- a custom scheme handler serves the embedded web bundle and **proxies + caches** `undertext.org/data/*` to keep requests same-origin (and avoid `file://` CORS issues).
- genuinely-native features layered on top: notifications, immersive reading modes, settings, offline cache, native share sheet.

- build workflow embeds a **built copy** of the web app into the iOS target (generated; not hand-edited).

the shared native core (`UndertextKit`)

- a **Swift package** shared across native targets: models, verse fetch service, audio-slug logic, reference parsing.

---

the surfaces (what 2.0 ships)

> ⚠️ **"all platforms" still means multiple App Store submissions.** iOS/iPadOS (and embedded watch + clip) ship together; tvOS ships as its own app submission.

Surface	How it ships
iPhone / iPad	Universal iOS submission (single binary)
App Clip	Embedded in the iOS submission
Apple Watch	Embedded companion (installed via iPhone)
Apple TV	Standalone tvOS submission
Mac	Runs via "Designed for iPad" compatibility (not Catalyst)

---

how it deploys (high level)

release flow is automated from the repo root (dry-run supported). at a high level:

1. build the web bundle
2. sync static web assets to object storage + CDN
3. embed the built web bundle into the iOS shell
4. build native targets in Xcode

key deployment traits:

- static assets are versioned/cache-optimized; the app shell is configured to update quickly.

- deploy tooling includes guards to prevent credential leakage into built assets.

---

how it’s gated in the cloud (high level)

- **prod web**: object storage + CDN with an origin-access model (bucket not publicly readable; CDN is the reader).
- **dev AI proxy (if enabled)**: server-side only; API keys live in server env vars and never in client bundles.
- **stats dashboard**: access-restricted (auth at the edge / gateway) and backed by server-side queries over CDN logs.
- **analytics philosophy**: no client tracking scripts; server logs are used for aggregate, privacy-preserving insight.

---

force-update floor (per app)

a small JSON file at a stable URL can set:

- **minimum build** (hard floor)
- **latest build** (optional nudge)

this is **per-app** (keyed by bundle id / app identity) so one platform’s update prompt can’t block another.

---

app clip: short-url correctness

App Clip invocation uses a short canonical URL form (`/c/<ref>`) to fit the App Clip Code URL length constraints; long query-string forms are retained for contexts without that limit.

store-facing safety note: anything experimental (e.g., AI companion features) must remain strictly build-gated so review sees only the shipped, static experience.