

How It Was Built

THE SYSTEM ARCHITECTURE OF THE BEREAN DASHBOARD

*“They received the word with all readiness of mind, and searched
the scriptures daily, whether those things were so.”*

— Acts 17:11

V60 EDITION · SMITH FOR CHRIST

The whole system on one page: [the system map →](#) — every arrow labelled with what moves along it. Or watch it: [How Berean Works — a narrated tour →](#), six minutes, in Jarnathan's voice. This paper is the long form. A **web edition** — written as text, not photographed from an older build. Every number on this page was measured from a shipping file.

01 What we actually built

FOUR ENGINES, ONE PAGE, NO ACCOUNT

The Berean Dashboard is a Bible study notebook that runs entirely in a browser tab. There is no account to create, no password to lose, no server that can read your notes, and nothing to install. You open a web address and you are working.

That description sounds like a limitation, and for the first week of the build it was treated as one. It turned out to be the whole design. A tool a student can only reach by logging in is a tool a student can be locked out of. A tool that keeps notes on someone else's computer is a tool whose promises depend on that someone staying in business. The Dashboard makes a narrower promise and keeps it completely: *your study is yours, it lives on your device, and nothing here charges you*. Since v39 it can also live, sealed, on a server — section 10 says exactly what that means and what it cost.

Underneath the single page there are four engines, and every feature in the app is one of them wearing different clothes.

1 · THE READER

Fetches chapters, shows them side by side across translations, and lets a student mark words in six colors — God's character, commands, repetition, sin and warning, promises, time and place.

It reads aloud, it paces itself around verse numbers, and it can be widened into a comfort text for eyes that need one. Since v37 it has a second shape for a phone, and since v38 that shape can be asked for on a Mac or an iPad — section 09.

2 · THE NOTEBOOK

Observation, Interpretation, Application — three fields per passage, saved as you type, exportable to Markdown, PDF and Word.

One storage key, on your device. The app never sends a note anywhere, and the export that comes back in is the one it wrote.

3 · THE ORIGINALS

Hebrew and Greek beside the English, tagged with Strong's numbers, tappable into a lexicon, markable like any other word.

Since v34 the Greek column is a choice, not a default, and since v33 the Variant Lens shows where two Greek editions actually differ.

4 · THE COUNTER

Word and phrase analytics that measure the real text rather than remembering a number — by word, by Strong's number, by sequence of numbers, by family.

When it cannot measure something exactly it prints a dash and says why.

Figure 1 — the four engines. Every panel in the app is one of these four, dressed for a different job.

A fifth thing sits beside them and is deliberately *not* an engine: the Study Partner, an AI coach that asks questions rather than answering them. Section 05 explains the leash it wears and why it was built with one.

02 One file, and what ships beside it

THE ENTIRE APPLICATION, MEASURED

The single-file architecture is not a slogan. It is a constraint that has been enforced through thirty-eight versions, and it has killed features that would otherwise have shipped. Here is the whole application as it stands today.

It has also been *tested* rather than merely restated. Version 35 added two things that would ordinarily have arrived as dependencies, and neither did. A mind-map export needs a ZIP file, because the `.xmind` format is a ZIP; the obvious answer is a hundred kilobytes of library, and the answer taken instead was eighty lines that write the archive directly — a checksum, a header per file, a directory at the end. Pictures in a note need somewhere to live that is not the notes; the answer was the browser's own database, which costs nothing to add. Neither needed a build step. The rule held under pressure, which is worth more than the rule asserted a fourth time.

<code>index.html</code>	1.5 MB	The application. Markup, styles and script in one file — the reader, the notebook, the originals, the counter, the panels, the service-worker registration.
<code>_worker.js</code>	80 KB	The only backend: a Cloudflare Pages Advanced-Mode worker that answers <code>POST /api/chat</code> and, since v39, <code>GET</code> and <code>PUT /api/notes</code> — a sealed vault in and out of a key-value store — and, since v50, <code>GET /api/tts</code> , a narrator that is inert until it is given a voice — and, since v53, <code>/api/votd/subscribe</code> and <code>/api/votd/send</code> , the morning verse — and, since v57, <code>/api/share</code> , a sealed study it cannot open, kept thirty days by id — and, since v58, <code>/api/house</code> , one sealed document per household that each member's device rewrites — and hands every other request straight to the static assets.
<code>sw.js</code>	7.3 KB	The service worker — offline shell, cache versioning, a deliberately narrow caching policy, and, since v53, the one notification a morning.
<code>manifest.json</code>	826 B	What makes it installable to a home screen.

No build step. No bundler. No framework. No package manifest to install from, no `node_modules`, nothing that has to be compiled before it can be read. You can open `index.html` in a text editor and see the entire program. That was chosen on purpose, and the reason is uncomfortable to say out loud: this app was built for my family, and it needs to still work when nobody is maintaining it. A build pipeline is a dependency on somebody continuing to run the build.

Two more single files stand beside the application and read the same shipped corpora: `patterns.html`, the Patterns Lab, and — added between v36 and v37 — `explorer.html`, the ELS Explorer — the letter stream searched at any skip, with a shuffled control computed in the same run beside every result. Neither has a dependency either.

Twelve data files ship alongside the application. They total roughly twenty-five megabytes, and **not one of them is loaded when the app starts**. Each is fetched the first time a feature needs it and cached by the browser and the edge from then on.

text-KJV.json	4.9 MB	The full 66-book text of three translations, exactly as the app displays them — so phrase counting can be exact, local and uncapped instead of paging a search server.
text-WEB.json	4.4 MB	
text-YLT.json	4.5 MB	
seq-ot.json	1.7 MB	Per-verse Strong's sequences in original word order for the Hebrew Old Testament, from the OpenScriptures tagged text.
seq-nt.json	766 KB	The same for the Greek New Testament. Numbers never inflect, so one sequence sweeps every spelling and every rendering.
wordfreq-KJV.json	286 KB	Whole-word occurrence and verse counts per testament, tokenized exactly the way the app tokenizes, so the summary line and the results agree.
wordfreq-WEB.json	264 KB	
wordfreq-YLT.json	262 KB	
kjv-hard.json	32 KB	The difficult-word dictionary — 185 King James headwords whose sense has moved, 139 of them words still in daily use that meant something else. Lazy-loaded on first use, like xrefs.json, never bundled into index.html. Built and re-verified by scripts/build-hard-words.mjs, which refuses to write the file if any of its 212 verse citations does not hold against the shipped King James text.
eng2strong-KJV.json	191 KB	English word to Strong's numbers, measured from how the tagged text actually renders each number — not from a dictionary's opinion.
text-WLCC.json	3.1 MB	The consonantal Westminster Leningrad Codex, from the OpenScriptures <i>morphhb</i> edition, whose text is public domain. Added at v36 so the Patterns Lab counts Hebrew letters from a file that ships with the app rather than fetching 929 chapters from bolls.life one at a time.
text-NTGT.json	1.7 MB	Tischendorf's 8th edition Greek New Testament, from CrossWire's public-domain SWORD module. The same reason, and the same v36 release — but it is a <i>different digitisation</i> from the one the Lab read before, so it is an edition change rather than a correction, and the Lab says so on the page. The Textus Receptus is still read live: the editions that match it are not freely licensed.

xrefs.json	3.2 MB	344,756 cross references, verse to verse, from the openbible.info table (CC BY 4.0), keyed by the verse a reader is on. Added at v39; fetched the first time Deeper opens on a verse, never at start-up. The verse text each reference shows comes from the Bible already on the device, so a reference costs no second request.
syntax/1–66.json	48 MB	The syntax tree of every sentence in the Bible — clause and phrase structure, the role of each part, a gloss and a Strong’s number on every word — from the MACULA Greek datasets (Clear Bible, CC BY 4.0), Nestle 1904, for 40–66 (12.5 MB), and the MACULA Hebrew datasets (Clear Bible, CC BY 4.0), Westminster Leningrad Codex, for 1–39 (36 MB). One file per book, the same shape for both testaments; Examine fetches a book the first time its block diagram is asked for, never at start-up. Derived by <code>scripts/build-syntax.mjs</code> and <code>scripts/build-syntax-hebrew.mjs</code> , and the licences ship beside the files.
geo/places.json, geo/land.json	304 KB	The map behind Examine’s <i>Where</i> : every identifiable place in the Bible with its coordinates and every verse that names it, from openbible.info’s Bible Geocoding Data (CC BY 4.0); and Natural Earth land (public domain), clipped to the Bible lands and simplified to 5,818 points. Projected and drawn as SVG on the device — no map service, no key. Both fetched the first time the map is opened, never at start-up; derived by <code>scripts/build-geo.mjs</code> .

Why ship the text at all?

Every one of those counts could be asked of a search server instead. Three things argued against it. A remote count is capped, paged and slow, so the app would have to either lie about totals or make a student wait. A remote count also fails when the network does, and a student on a phone in a church basement is exactly the reader this was built for. And a remote count is a request somebody else pays to serve — sixteen megabytes shipped once and cached is kinder to a small server than a thousand queries a week.

Each of those files carries a `note` field describing its own provenance in plain English — where the text came from, how it was tokenized, what was filtered out. A data file that cannot explain itself is a data file nobody can check.

03 Around the world on no budget

WHOSE SERVERS, AND WHAT WE OWE THEM

The Dashboard costs almost nothing to run, and that is not because it is clever. It is because other people are generous, and because the app was built to lean on them lightly.

THE READER'S DEVICE

The browser holds the notes, the marks, the settings and the cached chapters. This is where the study lives.

CLOUDFLARE PAGES — THE EDGE

Serves the four application files and the twelve data files from wherever the reader is. Free tier. One Advanced-Mode worker handles the two API routes, and one key-value namespace holds the sealed note vaults — blobs the worker stores and returns and cannot read.

BOLLS.LIFE — THE SCRIPTURE API

Translations, tagged Hebrew and Greek, lexicon entries, and the commentary seats. Maintained by Bohuslav Pavlisinec in Ukraine, given away free.

HELLOAO.ORG — A SECOND SCRIPTURE SOURCE

Additional translations, fetched only when a reader selects one.

AN AI PROVIDER — ONLY FOR THE STUDY PARTNER

Optional, and the only part of the system that costs money. If no key is configured the app is fully usable; the Partner simply declines.

Figure 2 — the whole stack. Nothing above the reader's device holds a note it can read.

Being a good guest

Leaning on a free API run by one person carries obligations, and the code takes them seriously. Requests go out sequentially rather than in a burst. A rate-limit response produces a backoff and one patient retry, not a hammering loop. Paged scans stop at a bounded number of pages and the interface says so — a capped count is printed with a $\geq$ rather than dressed up as a total.

Where a page size is configurable it is set to 128, which is not a guess: it is the number Bohuslav himself named when asked what load he preferred. And wherever a static dump could replace a live harvest, the dump won. That is the real reason those sixteen megabytes

ship — they are counts that would otherwise be scraped, one query at a time, off somebody else's free server.

The six seats

The Deeper tab holds commentary seats: Jamieson-Fausset-Brown, John Gill, Matthew Henry, Tyndale, Adam Clarke, and Keil-Delitzsch. All six are public-domain works served by the scripture API.

There were seven. David Guzik's Enduring Word held one, and it was removed in July 2026 when it became clear that Enduring Word's terms permit linking but not hosting. The app now links to Guzik's commentary on the open chapter and never fetches, caches or excerpts a word of it — the service worker explicitly refuses to cache that domain, because caching would be hosting. His seat was not handed to another voice. Six seats and an honest link is better than seven seats and a quiet violation, and a test suite of thirty checks now exists for the sole purpose of making sure nobody ever "improves" that back.

04 Anatomy of the notebook

WHERE THE STUDY LIVES, AND WHAT WATCHES OVER IT

A student opens a chapter, marks it, writes about it, and closes the tab. Here is everything that happens to that work.

1 · FETCH

The chapter is requested from the scripture API in the chosen translations. If it was read before and the translation has not been updated since, the service worker answers from cache and the network is never touched.



2 · RENDER

Verses are laid out in columns, verse numbers made tappable, Strong's tags turned into lexicon handles, and any existing marks re-applied to the words they belong to.



3 · MARK AND WRITE

A tap paints a word in one of the six colors. The three note fields accept Observation, Interpretation and Application. Both are captured as they happen.



4 · SAVE

Everything — notes, marks, settings, seat choices, translation picks — is written to a single browser storage key named `berean-dashboard`. One key. No request. No copy anywhere else.



5 · KEEP

Once there is anything worth keeping, the app asks the browser to mark that storage persistent, so routine housekeeping will not evict it. Exports to Markdown, PDF and Word are one tap, and the Markdown export can be read back in.

Figure 3 — the notebook pipeline, from a chapter request to work that survives the tab closing.

The flag that watches the key

Browser storage can fail. Private browsing modes throttle it, some configurations disable it, and a full disk will refuse a write. Silent failure here would be the worst bug this app could have: a student would study for an hour and lose all of it with no warning.

So every read and every write of that one key sets a flag, and the moment a write throws, a banner appears telling the student their work is not being saved and offering the export. Nothing about that path is decorative. Any change touching it is treated as safety-critical and tested against storage that refuses to cooperate.

There is one more guard. If a student navigates away with unsaved work in a field, the browser is asked to confirm. It is a small thing that has saved real study.

The exports deserve a note of their own. Markdown is the only format that can be imported back — it is the format of record. PDF and Word are for sharing and for print, and both are generated from the same builder, so the two can never drift apart and start telling different stories about the same session. There is also an auto-save-to-file option for browsers that support writing directly to a chosen file, which turns the notebook into something with a home on disk.

05 The Study Partner

A COACH ON A SHORT LEASH

The Study Partner is the only part of this application that talks back, and it was the hardest part to build honestly. An AI that answers Bible questions is easy. An AI that *refuses* to answer them, and hands the work back to the student, is the thing worth building.

1 · GATHER

The app assembles what the student is actually looking at: the passage, the marks, the notes so far, the commentary excerpt on screen with its author named, any of the student's own published writing that touches the passage, and — critically — any count the app has just measured.



2 · BOUND

Each piece is truncated to a fixed budget before it is sent: the passage to 2,600 characters, the commentary to 1,600, the notes to 1,000, a measured count to 900, and the last ten turns of conversation to 4,000 each. Nothing grows without limit.



3 · INSTRUCT

The system prompt sets the method (Observe, then Interpret, then Apply), the theological frame, the teachers who may be drawn on and credited by name, and eleven numbered rules of role and boundary.



4 · ANSWER

The worker calls the configured AI provider, streaming when the client asks for it. If no provider is configured it returns a plain error rather than pretending. The reply goes to the screen and is never stored.

Figure 4 — the Study Partner loop. Everything it knows arrives with the question and leaves with the answer.

The leash

The rules that matter most are the ones that make it less useful in the short term. It defaults to questions rather than answers, and only gives a straight answer when the student says they are stuck or asks outright. It refuses to leave its lane: politics, medicine, money and general assistance all get one warm sentence and a return to the open passage. If a student shares something heavy it stops coaching entirely and urges them toward a pastor or a professional, because a study tool is not a shepherd. It will not invent scripture, will not pro-

nounce on anyone’s salvation, will not set prophetic dates, and says plainly what it is when asked.

Between v36 and v37 the Partner gained a second seat. **Coach** is the leash above, and it is the seat every boot lands on: the stored preference is deliberately not restored, so nobody opens the app into a long exposition because someone else chose it last night. **Teacher** is opt-in and expounds the passage at length, borrowing a named method — the Companion, the Expositor, the Shepherd, the Watchman, the Architect — as a lens that changes emphasis and order and never the doctrine. The two prompts are separate, and a test pins a hash of the Coach’s so that work on the Teacher cannot widen the Coach by a byte.

And one rule governs the whole relationship between the Partner and the counter:

Counts are measurements, not memories.

If a student asks how many times a word appears and the app has not just measured it, the Partner is instructed to say it cannot verify a number from where it stands and to point at the button that counts the real text. Concordance numbers recalled from memory are how false Bible trivia spreads, and this app will not be a source of it.

Cost of the app	Nothing. No account, no fee, no donation taken.
Cost of the Partner	Pennies per conversation, paid by the author, on a small fast model with a 700-token ceiling per reply.
If no key is set	Everything else works. The Partner returns a clear message saying it is not configured.
Fallback path	A free platform model, if one is bound. Two paths, so the feature can survive a budget it never had.
What is stored	Nothing. No transcript, no log, no history of what anyone asked.

The system prompt itself is not served publicly. It sits in the worker, not in the page, which is the one genuine asymmetry in an otherwise fully readable application — and it is there because a coach whose instructions can be read can be talked out of them.

o6 The Originals, and what “original” means

V33 AND V34 — TWO ROOMS ADDED TO THE READER

For most of its life the app showed one Hebrew text and one Greek text and called them the originals. That was fine until it wasn't. The Hebrew is not seriously contested — the Masoretic text stands. The Greek is a live question, and pretending otherwise was the last real dishonesty left in the reader.

Choosing the Greek

Since v34 the Greek column is a choice among three editions: the Textus Receptus behind the King James, Tischendorf's critical edition, and a modern critical Greek New Testament. The Hebrew column remains the Westminster Leningrad Codex, because there is no comparable live dispute to represent.

What the picker refuses

Two texts a reader might expect to find here are deliberately absent from the originals picker: the Septuagint, and Delitzsch's Hebrew New Testament. Both are valuable. Both are *translations* — a Greek rendering of Hebrew scripture, and a Hebrew rendering of Greek scripture. Putting either in a column labeled “original” would teach a student something false about what they were looking at. They are welcome in the translation columns, where they belong, and refused in the originals column, where they do not.

The borrow

Not every Greek edition on offer arrives tagged with Strong's numbers. Two of the three are plain text. Rather than drop word-study marking whenever a reader chooses an untagged edition, the app borrows: it fetches the tagged edition of the same testament, builds a ledger of which numbers belong to which words, and pins those numbers onto the chosen text so the lexicon handles still work.

The borrow is recorded rather than hidden. A reader who has borrowed tags can see that the numbers came from a neighbor. Silent borrowing would be a small lie told a thousand times.

The Variant Lens

With two Greek columns on screen, v33 added a lens that shows where they differ. It walks both verses as sequences of normalized words, finds the longest run they share, and marks every word outside that run with a dotted underline. Where one edition has a verse the other does not, it says so with a badge rather than leaving a confusing blank. A summary line reports what it found.

The lens carries one vow, written into the code beside it:

The lens describes; it never judges — the reader sees, the reader weighs.

It will never label a reading “original,” never mark one edition correct, never put a thumb on the scale. It shows a student that a difference exists and exactly where it falls, and then it gets out of the way. That is the entire feature, and the restraint is the feature.

07 The engines that count

FOUR WAYS TO ASK, AND ONE STANDARD OF PROOF

Word study is where a Bible tool either earns trust or quietly forfeits it. A number on a screen looks equally authoritative whether it was measured or remembered. The Dashboard has four counting engines, and every one of them measures.

WORD AND PHRASE

Exact whole-word counting against the shipped text, case-aware, uncapped in the three translations whose full text ships.

It reports occurrences and verses separately, because a word appearing three times in one verse is not the same finding as three verses.

BY STRONG'S NUMBER

Every occurrence of a number across the canon — the roadmap's oldest promise, kept in v30. A bare number typed into the search box is understood as a number.

Numbers do not inflect, so one query sweeps every spelling and every rendering the translators chose.

BY SEQUENCE

Two to six numbers in order, matched against the original word order within one testament, with a mode that tolerates a couple of intervening words.

This is phrase search that survives translation entirely.

WORD TO NUMBER

Type an English word, get the numbers it actually renders — measured from the tagged text, not from a dictionary.

It *offers*. It never silently substitutes a number for the word you typed.

Figure 5 — the four counting engines. Each answers a different question about the same text.

Why four and not one

Because English collapses distinctions the original languages keep, and a single engine hides that. The worked example that drove the design: searching the English words “fear not” in the King James turns up 187 verses — but the list includes Genesis 20:11, where Abraham says “the fear of God is not in this place.” That is not the phrase anyone was looking for. Change the capitalization and the total changes again, because “Fear not” at the head of a sentence is a different string.

Ask the same question by the numbers instead — the Hebrew negative particle followed by the verb for fear — and 74 Old Testament verses stand up, cleanly, including ones the English search missed entirely because the translators rendered them another way. Four en-

gines exist because the fourth one finds what the first three cannot, and because a student deserves to know which question they actually asked.

Honesty artifacts

Some counts genuinely cannot be exact. A translation whose full text does not ship can only be counted through a paged remote search, and a paged search that hits its ceiling knows only that there are *at least* so many. In those cases the table prints $\geq$ before the number. Where a count is impossible — a form the data cannot distinguish, or an edition without tags — it prints a dash and a short reason rather than a zero. A zero is a claim; a dash is an admission.

| *Counts are measurements, not memories.*

That sentence appears in three places in this system: in the counter's design, in the Study Partner's instructions, and here. It is the closest thing this application has to a creed.

o8 The pipeline that ships it

HOW A CHANGE BECOMES A LIVE SITE

There is no build step, but there is a gate. A hosting platform that replaces the entire site on every deploy is unforgiving: forget one file and it is not stale, it is *gone*. So the pipeline is built to fail closed.

1 · BRANCH

Work happens on a branch. The main line is never edited directly and never deployed by hand — not through the command line, not through a hosting dashboard.



2 · THE BUNDLE GUARD

Before anything else, the workflow checks that all forty-eight required files are present — the four application files, the two icons, the twelve data files, the cross-reference table, `robots.txt`, and, since v40, every shipping page of the library, so a document the About page links can never be missing from a deploy, and, since v42, the seven files of the OCR reader in `ocr/`, and, since v46, the difficult-word dictionary and the page and opening narration of each of the two films. Only the PDF editions are optional. A missing file stops the deploy. This list has only ever grown; it has never been shortened.



3 · SYNTAX AND SANITY

Every inline script is extracted and syntax-checked, the worker is parsed as a module, and the whole tree is rejected if any file contains a local `file://` path that escaped from someone's machine.



4 · THE SUITES

One hundred and six test suites run. They are plain scripts with no test framework: they read the shipping files and assert on what is actually in them. An empty test directory is itself a failure — a suite that cannot run must never look like a suite that passed.



5 · ASSEMBLE AND VERIFY

The required files are copied into a clean directory along with the optional documents, the version string in the service worker is checked against the version in the app, the bundle is published to the edge, and the live site is fetched back to confirm it answers.

Figure 6 — the deploy pipeline. Five gates, any one of which can stop a bad release.

What the suites actually test

Most are wiring checks rather than unit tests, and that is the right shape for a single-file app with no framework — but not all of them are any more. Where a piece of logic is pure, it is wrapped in a marked block inside the shipping file, and a suite lifts that block straight out of `index.html` and runs it. Undo, the method engine, the note search, the ZIP writer and the mind-map builder are all tested that way: the code under test is the code that ships, not a copy of it. It is how an application with no build step still gets real unit tests. They assert that every document the index page links exists in the repository *and* is named in the deploy bundle, because a working link to a file the bundle forgot is a 404 nobody notices. They assert that the Enduring Word policy still holds. They assert that the seat list is six long and heals a stored seven. They assert that the version strings agree. They assert that the counting tables and the search paths still behave as documented.

One honest admission

The workflow runs on pushes to the main line, not on pull requests. That means a proposed change gets no automatic signal before it is merged; the suites are run locally instead, and the person merging is the one accountable for having run them. It is a real gap, it is known, and stating it here is cheaper than pretending the gate is tighter than it is.

The service worker

The offline layer is intentionally narrow. The shell is cached so the app opens without a network. Same-origin requests go to the network first and fall back to cache. Scripture requests go to cache first and refresh in the background — except the translation catalog, which is always fetched fresh, because the app reads each translation's update timestamp to decide which cached chapters have gone stale. The chat route, video and captions are never cached at all. And one domain is excluded by name, with a comment explaining why: caching Enduring Word would be hosting it.

One more detail, small and hard-won: when the worker cannot serve a request it always returns a real response. Returning nothing produces a browser error message that tells a student nothing useful, and on some browsers it is the only thing they will see.

09 One grid, three screens

V37 AND V38 — THE PHONE, AND THE PHONE EMULATED

Until v37 the phone got the desktop folded in half: four columns stacked into one, so a single verse was the King James, then the NASB, then Young's, then the Greek, and a chapter of Acts stood twenty-two thousand pixels tall. The obvious fix was a second renderer for small screens. It was refused, because every feature in section 04 — marks, highlights, the Variant Lens, the Strong's borrow, scroll-to-verse, the analytics' walk over the cells — addresses one grid, and a second grid is a second place for each of them to be wrong. There is still exactly one grid. What changed is which of its cells are allowed to be seen.

1 · ONE GRID, ALWAYS

The Reader builds the same grid on every device: one cell per verse per version, in verse order, each cell carrying its translation as a data attribute. Nothing about the build knows the width of the screen.



2 · A CLASS, NOT A MEDIA QUERY

At 640 pixels and below the body gains one class, `pocket`, and every phone rule in the stylesheet lives behind it. A media query cannot be switched on by a person; a class can, which is what makes the opt-in in the next step possible.



3 · ONE RULE SHOWS ONE VERSION

A single generated CSS rule — `body.pocket #grid .cell:not([data-tr="KJV"]){display:none}` — hides every cell but the version being read. Switching versions rewrites that one rule and fetches nothing. Tapping a verse number clones that verse's hidden cells, originals first, into a sheet. Acts 17 went from 22,962 pixels to 4,701.



4 · THE FRAME (V38)

On a wide screen with pocket mode ticked, the body itself becomes the phone: a centred 430-pixel column no taller than a phone, carrying a CSS transform. A transform makes an element the containing block for its fixed-position descendants — so every bottom sheet, bar and pill that says `left:0`; `right:0` now spans the phone and not the window. Nothing inside pocket mode learned that a frame exists.

Figure 7 — *pocket mode*. The same grid, one class, one rule, and on a Mac or an iPad a body that is the phone.

The discipline this bought is easy to state and was worth the whole release: the phone and the desktop cannot disagree about a verse, a mark or a count, because they are reading the

same cells. The suites hold the line from the other side — the phone-reading tests assert that the desktop’s rules are unchanged *by absence*, checking that no pocket rule exists outside the class, and a fifty-five-check suite pins each pocket rule by reading the shipping file.

What the phone got that the desktop did not

Seventeen-pixel type; Strong’s numbers folded to five-pixel dots that keep a 44-pixel hit area and the same click handler; the interlinear line opt-in; a word sheet that fetches one line of gloss from the same dictionary call the Lexicon makes; a reference picker; Settings that only forward to the desktop controls’ own handlers, so there is one implementation of every setting; and the documents opening inside the app, in an iframe sheet, in the app’s own type. One sanitizer fix reached the desktop too: the source’s stray King James marginal notes are stripped, and the space it writes before punctuation after a tagged word is closed.

The tablet, and the desktop, audited (v39)

v37 made the phone the best-served screen, and a September audit measured the other two the same way — live, at four widths, with screenshots — and found them the weaker ones. Nine things were fixed in one stack, and none of them added a control: the welcome card became a dialog with a close button and an Escape key; the top bar breaks at a chosen width into a navigation row and a tools row rather than wrapping ragged; on an iPad held upright the fourth column, which had been 208 pixels off the right edge with nothing to say so, folds instead, and one chip at the foot of the text names what folded and offers it back; the six hundred and seventy-eight speaker glyphs under a chapter’s Greek show only on hover or on the focused verse; the pencil sits at the corner of the text area; the Notes pane folds its rare toolbar half behind one `⋮` so the writing box gets the height; the Lexicon and Partner voice controls fold shut on every screen; and on a wide monitor a verse tap opens the commentary in a column beside the Bible instead of over it — a changed default, and the reader’s last choice wins after that. Then a second pass, on the tablet alone: every primary control in the side panel is 44 pixels on a coarse pointer, chips and fold summaries 36, the palette’s dots 30 with a halo whose edges meet the next dot’s and never cross it — all of it fenced to `pointer:coarse` and outside pocket mode, so the desktop keeps its density and the phone keeps its own sizes. The price, measured and written on the board rather than hidden: 69 to 83 pixels more chrome above the writing box on an iPad.

The rule from section 09 held through all of it. There is still one grid; the tablet’s fold hides cells, it does not render new ones.

10 The honest delta, and the road ahead

WHERE THE BIG TOOLS STILL WIN

Blue Letter Bible is a fine tool with two decades of work in it. This is a comparison written by the author of the smaller one, so read it accordingly — but the rows where they win are listed first, and honestly.

Interlinear parsing	They win. A full morphological parse — tense, voice, mood, person, number — word by word. The Dashboard shows Strong's numbers and lemmas, not parsing codes.
Breadth of resources	They win. Dozens of commentaries, dictionaries, maps, timelines and media libraries. This app has six public-domain seats, a link, and — since v39 — the cross references with each verse shown in place, which is the one part of that shelf they do not do.
Translations offered	They win on licensed modern translations. Licensing costs money this app does not have and will not take donations to get.
Your notes	Berean. Notes are first-class here, on your device, with no account — and, since v39, on your other devices too, through a copy the server cannot read. Look-up sites are not built to be written in.
Counting the text	Berean. Exact, uncapped, local counting by word, by number, by sequence — with a printed admission whenever a count is capped or impossible.
Showing the variants	Berean. Two Greek editions side by side with the differences marked, and no editorial verdict attached to them.
Offline and free	Berean. Installs to a home screen, opens without a network, costs nothing, asks for nothing.

The positioning follows from the table: **Blue Letter Bible is where you look things up. Berean is where you do the work.**

On the parsing gap, precisely

An earlier edition of this document named the tagged original-language corpora as the next mountain to climb. Part of that climb has happened — the Old Testament sequence data shipping with this app comes from the OpenScriptures tagged Hebrew text, which is a proper scholarly corpus rather than a scraped approximation.

But it must be said plainly, because it would be easy to overclaim: *a Strong's number is not a parsing code*. Knowing a word is number 3372 is not knowing it is a Qal imperfect second-person masculine singular. The gap is narrower than it was and it has not closed. It remains a data-source ceiling rather than a coding problem, and the standing rule is that no parse will ever be manufactured from data that does not contain one. A guessed parse presented confidently would do more damage than an honest blank.

The notebook, and the second store

Version 35 did most of its work in the notebook, and one part of it is genuinely architectural rather than cosmetic.

Notes live in one browser key, and now pictures do not. Every note this app has ever taken is saved into a single entry in the browser's local storage — deliberately, because it is simple, synchronous and has no server anywhere near it. That store is small: a few megabytes for the whole site. A photograph off a phone is three to five. Put one anywhere near that key and the write that fails is not the picture's write, it is the *notes'* write, and a pasted photo could take a morning's writing with it. So pictures go to the browser's database instead — a different store, with a quota measured in hundreds of megabytes — and the notes did not move at all. The rule that fell out of it is worth stating plainly: **the whole application writes to local storage in exactly one place**. A test fails the build if a second writer ever appears, because the promise that a picture cannot cost you a sentence is only as good as that count.

A note is a bag of named fields, and always was. For thirty-four versions a note had three boxes — Observation, Interpretation, Application — and it was assumed that offering other study methods would mean converting every note ever written. It did not. The save has always been *write this text under this name*; nothing underneath ever knew that only three names were in use. SOAP simply writes two more beside them. Nothing was converted, because there was nothing to convert.

The real risk sat one level up, and it was invisible. Because switching method *hides* fields rather than deleting them — which is what makes switching safe and reversible — a note can hold writing in a field the current method does not display. Twenty places in the app walked a fixed list of those three names: three exports, the search, the Partner's context, the note manager, and the check that decides whether an export is offered at all. Every one of them would have quietly left that writing out of an archive while it sat perfectly safe in storage. All twenty now ask the note what it contains. A test fails the build if a fixed list ever reappears, because all twenty looked correct: each was locally reasonable and globally wrong.

A verse note is one more name in the bag (v39). The first feature every serious competitor has and this app did not was a note anchored to a verse rather than a chapter. It arrived without a storage change, for the reason above: a note on verse 11 is the field `v11` in the chapter's bag, beside `O`, `I` and `A`. Every surface that already asked the note what it contained — the three exports, the search, the Partner's context, the manager — carried it from the first minute. The mark beside the verse number, the pin on the palette and the fold at the foot of the pane are the whole of the new code.

A mark on the whole verse is a mark with no phrase (v40). A phrase mark is `{ b, c, v, phrase, color, style }`, and every reader of marks — the painter, the word analytics, the every-occurrence colouring, the four printers — walks the phrase. The whole-verse mark is the same record with `whole: true` and no phrase, which means the analytics and the phrase painter skip it without being taught to: there is nothing to walk. One new painter tints the cell and draws the line down its margin, in every column, because the mark is anchored to the verse and not to any translation's words. The four places that print a mark say *the whole verse* through one function. No new store; one new key in the shape sync carries.

The study log is a fact about the device, not the study (v40). `state.sessions` sits in the one `localStorage` key beside the studies — thirty records at most, each a counted evening: chapters opened, marks by colour, words written since a chapter was first seen that evening, lookups, commentators read, cross references, questions, searches. Nine one-line hooks feed it, every one fail-soft: a report can never cost a mark or a word. It is deliberately outside the sync payload, which still carries only `{ studies, gone }` and is pinned to that shape; an evening on the iPad belongs to the iPad. And it is measured, never narrated: the one sentence it prints is built from counts, and nothing in the app writes a word of prose about your work.

The peek adds no store and no control (v40). On a fine pointer, outside pocket mode, resting on a Strong's number, an original word, a marked phrase or the mark beside a verse number shows a card of what is already on the screen — the same dictionary call the Lexicon makes, cached per query; the legend's meaning for the colour; the verse note. It is fenced twice, in CSS by `(hover: hover)` and `(pointer: fine)` and `body: not(.pocket)` and in the listener by the same test, and it is `pointer-events: none`, so a finger never sees it and it can never take a click. The phone taught the app a peek before a commitment in v37; the desktop got the same idea in v40 without a single new button.

Landing on a verse from its words is an index, not a query (v41). The first half of the camera-on-a-printed-page goal needed no camera: the three shipped canons are on the device, so a pasted sentence is reduced to its content words, stemmed just enough that *be-*

lieveth and *believes* agree, and matched against an inverted index built once per canon from the same `text-*.json` the counter reads — 31,207 KJV verses in under two hundred milliseconds, then any lookup in under ten. The three rarest words pick the candidates; the score is four-fifths the words carried and one-fifth their order. Nothing is sent anywhere, and the reader's own translation is tried before the other two, so the common case costs one file. The second half — the camera itself, and an OCR sidecar of some fifteen megabytes, loaded only when asked for — waits on whether the first half is used.

The sync route now counts (v41). The only cheap finding of the security review was that `/api/notes` checked everything about a backup except how often one address sent one, and the free tier allows the whole site a thousand writes a day. It now allows thirty writes per address per ten minutes, counted in the isolate's memory after every cheaper refusal — best-effort by design, and the block says so — and answers the thirty-first with a 429 the client turns into a sentence. It still cannot read what it stores.

The camera is a sidecar, not a service (v42). Reading a printed page could have been an API call — a photograph up, a verse back — and it would have broken the promise on the first tap: a photograph of a Bible on a kitchen table is a photograph of a kitchen table. So the frame is drawn to a canvas and read where it is: by the browser's own `TextDetector` where it exists, and otherwise by Tesseract compiled to WebAssembly and shipped in `ocr/` — nineteen megabytes in the repository, about six fetched the first time the shutter is pressed, none at boot, and cached by the service worker like any other same-origin file, so the second use works with no signal. The browser asks for the camera when the sheet opens and never before; `getUserMedia` is called once in the file and a suite counts it. The words then go to v41's landing index, and the still stays on screen with the verse outlined on the page and the original language beneath — the same `origWords` the Originals column uses, because a second renderer would have been a second truth. The seven files of the reader are in the deploy guard; a bundle without them is not the app.

Live is a loop with a conscience (v43). Hovering instead of pressing could have meant reading every frame, and a phone reading every frame is a phone that is hot in a minute. So the loop reads a 32×18 grey thumbnail, cheaply, and asks one question — is the hand still? — and only when it has been still for two samples reads the guide's strip of the frame, never the whole picture, never two readings at once. What comes back passes a verdict before it changes the screen: a sure verse switches at once, a guess must be seen twice, a miss never removes what is there. The outline is drawn over the live video through the same letterbox arithmetic the still uses, and it fades while the hand moves, because an outline that is wrong for half a second costs more trust than one that is briefly absent. Freezing reads the whole page once and lands every segment — the cuts at printed numbers and the windows, since

the small numerals of a real Bible are the first thing a reader loses — one box set per verse, tappable. None of this is tracking; the outline is locked to the frame it was read from, and gliding it with the page is the plan's next step, not this version's.

The door opens where a camera is, not where a finger is (v44). v43 hid the camera button on every desktop, reasoning that a Mac cannot be held over a page. That was the wrong test. The right one is whether a camera exists, and the browser can answer it without asking anyone — `enumerateDevices` lists video inputs before permission is granted, with their labels blank — so the button now paints itself from that list and repaints when the list changes, which is exactly what happens when an iPhone wakes up beside a Mac and Continuity Camera offers it. The choice among cameras is one pure function: the remembered device if it is still present, else the first whose label says *iPhone* or *Continuity*, else the browser's default; labels are readable only once a stream has been granted, so the picker is painted after the camera opens, and hidden when there is only one. The Greek chips under the verse write the same state. origNT the settings gear writes and repaint the strip from the cached chapter — no request, no second code path. The rig can prove all of this with one synthetic camera and no more: Chromium's fake device is a single device, so the switch between two is a hardware check, marked as such.

The third store: a vault the server cannot read

For thirty-eight versions this document said, in figure 2's caption, that nothing above the reader's device holds a single note, and the repository's own rules said there would be no server copy by design. v39 changed that rule, in writing, and the change is worth stating as exactly as the rule was.

What did not move. Local storage is still the source of truth, still written from exactly one place, still counted by the same test. A reader who never opens the sync fold runs the v38 architecture unchanged.

What was added. A secret that never leaves the device. Until v52 the reader chose it, and the cost was written down honestly here: a design with no accounts has no per-user salt to hide behind, so the secret itself had to carry the strength, and twelve characters of human-chosen text was the weakest part of an otherwise strong system. The browser now generates **256 bits** from the platform's own randomness instead. From it the browser derives two things by a deliberately slow hash — six hundred thousand rounds: a vault identifier the server can index by, and a separate encryption key the server never sees. The studies, notes, marks and legends are serialised, sealed with that key using the browser's own cryptography, and put to the worker's second route, which stores the blob in a key-value namespace under

the identifier and hands it back on request. The worker cannot decrypt what it holds, cannot tell two readers apart except by opaque identifiers, and refuses to run at all if its namespace is not bound — a missing binding is a plain *not set up* to the reader, never a silent no-op. Pictures stay in the browser's database and do not travel; each device's own settings stay put.

There is no reset, on purpose. An account would give the reader a recovery path, and every recovery path is a second key held by somebody else. The trade was made the other way: the generated secret is the only key, and a lost one leaves a blob nobody can open. What v52 added is not a back door but a way to write the key down. The same 256 bits are shown once as **24 words** from the BIP-39 English list — verified here against the published fingerprint of that list rather than trusted — whose final word carries a checksum, so a phrase mistyped on restore is refused rather than quietly deriving a different, empty vault. The reader writes them down once and is never asked for them again unless a device is lost. That is the same promise as before, with one more device in it and a way back.

A second device pairs by looking at the first. The showing device renders the secret as a QR code — the raw bytes in base64url behind a scheme prefix, 57 characters, which at error-correction level M is a 33×33 code that reads in one try off a screen at arm's length. The reading device decodes it with the platform's own barcode reader where one exists and with a vendored decoder where none does, which on iOS is always. Nothing is typed and nothing is shown to the reader. The code is the key in plain sight, so it is on screen for sixty seconds with a countdown and is removed from the document on expiry, and the scanner accepts that one prefix so a code on a parcel or a menu cannot be mistaken for a vault. Both libraries load only when the pairing panel is opened; neither is in `index.html`.

When two devices disagree, both are kept. Each device remembers a small base — a hash of every field as it stood at the last sync — and a sync is a three-way merge, field by field: changed on one side only, take that side; changed on both, keep both, the other device's words joined under a line that says so, counted and named in the message after the sync. A journal that settles a conflict by timestamp will one day discard the paragraph that mattered and never say so. Deletions are tombstones with a time on them, and a deletion outranks only the edits that came before it, so a study deleted and recreated under its own name survives.

What the build learned. The first draft of the merge reused two function names the app already owned for merging notes between two windows of one browser. Every one of eighty-six suites passed, because each lifts one block and tests it alone; hoisting meant the running app took the new functions for the old ones and every save would have corrupted the notes. A rig driving two real devices caught it in one line. Every sync name is now prefixed, and a

new suite holds that no top-level name in the application is declared twice — the cheapest test in the repository, and the one that would have caught the most expensive bug.

Three things added as data (v50)

Between v47 and v50 the application grew by about a third, and almost none of that growth is machinery. It is three tables. Each was the alternative to a dependency, and each is held to account by a suite that reads the shipping file.

Paths is a table, not a feature. A path is a walk through Scripture a verse at a time — the Romans Road, the twelve steps of a recovery programme, the seven *I am* sayings of John — and every one of them is a row: an id, a name, an icon, a line of description, and a list of references written as four integers at most, in the app's own numbering of the sixty-six books. One hundred and four rows; 1,243 passages; 2,571 verses. The table is not stored in display order. A row names its group, a section names the groups it holds, and one pure function derives the list the reader sees, so moving a card is one word on its row and no path's id — which is the key its walked steps are stored under — ever changes.

What is measured: **every one of those 2,571 verses is checked against text-KJV.json, the very text the app displays**, so a wrong chapter or verse fails the build and never reaches the reader. That suite runs 277 checks. It also holds a ceiling: no two topic paths may share a third of their verses — there were fifteen such pairs when it was written, the worst at sixteen verses of eighteen — with three exemptions written down beside their reasons. What it cannot measure is whether a verse that exists is the *right* verse. That was done by reading: the newest paths had each reference printed beside the verse's own words before it went in, and one failed — a psalm about the city of God that had been chosen for a wife. The read-back is kept in the repository for the same reason the suites are: so the claim can be checked by somebody else.

The icons are one hidden drawing. Nine controls had been a glyph with no word, two of them the same open book, and emoji are drawn by the operating system — differently on Apple, Android and Windows, which is part of why one reader reported the site as malformed. They are now fifty-six symbols in a single inline SVG sprite at the top of the page: twenty-four units square, one stroke weight, no colour of their own, so they take the colour of the text around them in either theme. No icon font, no library, no second request. A suite compares every symbol's path to the approved drawing character by character, and keeps a list of the glyphs that were deliberately left as glyphs, so a new emoji cannot arrive unnoticed.

The narrator is a route that does nothing until it is told to. `GET /api/tts` will read a verse aloud in a recorded voice. It takes a translation and a reference and *never the words*: the worker fetches the text itself, so the key behind it cannot be spent by anyone who can type a URL. Only public-domain texts are on its list; a copyrighted translation is refused by name. Each verse is voiced once and kept in an object store, served from there for ever after; a miss is metered against a daily character budget, and over the budget the route says so and the reader hears the device's own voice instead. Without the store or the secret it answers 503 and names what is missing — the same shape as the notes vault.

What was measured, and what it said

The route has a health check that costs nothing. On the day this edition was cut it answered *key rejected* on the live site — so the narrator was switched off for every reader, and the fallback was doing all of the work. The code had seventy-six green checks the whole time. A suite can prove a route is built correctly; only asking the running site can prove it is on.

One diagram, one map, two hosts (v53)

The block diagram and the map were built for the Examine stage, which wants a landscape screen. A phone had neither, and the first design for the phone would have been a second renderer — smaller, simpler, and from the first day a little different from the one on the iPad. It is not built that way. Each panel is split into a pure half and a wired half, and the split is the architecture. `ckBlockLayout(sentence)` turns a MACULA tree into lines with a depth, a role and a connective, and `ckBlockHtml(lines, ...)` turns the lines into markup; neither reads the page. `ckGeoBuild(data, {book, chapter, verse, sel, view})` returns the map's markup and the frame it chose, and `ckGeoWire(box, state)` attaches drag, pinch, zoom and the place list to whatever box it is given, reading and writing only the state object it was handed — `view`, `sel`, and three callbacks for repaint, place and verse. The stage passes an adapter over its own globals (`ckGeoCtx`); the phone's verse sheet passes a small object of its own (`pkGeo`). There is one diagram and one map in this application, with two hosts, and a suite holds the sheet to calling the stage's functions rather than copying them.

The same rule put the panels' manners in one place. A panel wears a head (`ckPanelHead`: its name, the verse, the English in one line, the way back), a legend before it, and a why-note that is written from the tree (`ckBlockWhy`: the parent line, the role, the connective) — so the explanation can never disagree with the drawing, because it is computed from the same lines. Labels on the map are sized in screen pixels from the SVG's rendered size and placed

by a four-position collision pass (`ckGeoLabels`) that treats the ringed dots as obstacles, which is why a name never covers a neighbouring place on a four-inch screen.

A verse each morning, with nothing kept but an address (v53)

The morning notification is Web Push, written out rather than imported: the worker signs a JWT with the site's VAPID key (RFC 8292) and seals each message to the device's own public key with ECDH, HKDF and AES-128-GCM (RFC 8291, `aes128gcm`), all through WebCrypto, and a suite plays the browser's half — mints the device's keys, decrypts what the worker sealed, verifies the signature. What the server keeps is one record per device under the existing NOTES binding, `votd:sub:<hash of the endpoint>`: the endpoint and its two keys, the device's clock offset, and the running plan. No account, no address, nothing that names a person; a plan change re-sends the record, the bell's off deletes it, and a push service answering 404 or 410 deletes it too. The send is a GitHub Actions cron at 12:00 UTC that presents a bearer secret to `POST /api/votd/send`, which handles one page of ten devices per call and hands back a cursor, so no call grows with the list. The verse texts it needs are a 52 KB file built beside the plans, never the canon. Until the three secrets exist the routes answer 503 naming what is missing, and the card and the plans work regardless — the subscription is a courtesy on top of a feature, not the feature.

The plans are the same shape as the paths: tables, generated once by a script and held by a suite to what they promise — every whole-Bible plan covers all 1,189 chapters exactly once, a weekday plan has its 260 or 261 days, no day is empty. The reader's place in a plan rides in the one state object through a restore sanitiser, like everything else that is the reader's.

Five things a Berean asked for, built the same way (v57)

The share is the vault's shape turned outward: a study's chapter — marks, legend, a question per step, never the notes — sealed on the leader's device under a key minted for it and never stored, kept thirty days by the worker under `share:<id>`, the key only in the link's fragment, which no browser sends. The receiving device opens it, runs it through a restore sanitiser (the six legend keys, marks inside the named chapter, the two styles, tags stripped) and takes it into a study of its own; a second share of the same chapter refreshes the shared marks and leaves the reader's notes and other chapters alone. No list of recipients exists anywhere.

What did I miss? is the Partner with a frame put first in its prompt that allows only observation, and a parser on the client that holds the reply to a form — a verse, two to six exact

words of it, a tool — and drops what does not anchor, so a wandering answer costs an empty box and never a wrong line. The memory room is one table in the state object and one pure function over six intervals; the morning note learns its due dates and nothing else. Drive mode is a face over the one reader (`readChapter`) and a gate between its verses that listens, for a second and a half, for four words; nothing listens while the Bible is being read. The journeys are a table built by a script that refuses an unresolved name, drawn by the one map builder for both hosts. Each is held by its own suite; a hundred and sixty-five pass.

A handout, a legend, and a household (v58)

The household is the share's shape made to last: one sealed document under `house:<id>`, the key only in the link's fragment, and — the one difference — every member's device rewrites it whole. A device reads the document, merges its own line (a first name the family typed, a plan's id, a count of days read, a count of verses solid, and nothing else), drops any line not heard from in a hundred and twenty days, keeps the freshest eight, seals it and puts it back; the worker renews a hundred-and-eighty-day life on every write and never learns a name. Two devices writing in the same second can lose one line for one tick, and the next tick restores it. The document is a convenience, and it was not given a merge protocol it does not need.

The handout's marked passage is drawn from the chapter the app already holds and the marks in the study: the longest phrase first so that a shorter one never splits it, each wrapped in its colour with its shape as a glyph before it, because a copier keeps a shape and loses a colour. The diagram on the handout is the stage's own layout and markup with a print stylesheet over it — there is one diagram in this app, and it now has three hosts. The legend's presets are two tables of six strings. And the notebook's prompts were given a rule in place of an accident: a heading goes where the writer is, or to the end, and never into the middle of a line.

Examine • Lampstand (v59)

v59 carries the ideas of Gabe's Lampstand reader into Examine, and every one of them was built to the rules already here. **Elsewhere is links and only links**, the discipline the Guzik link set: seven addresses built from the book, chapter and verse, every pattern checked against the live sites for all sixty-six books, nothing fetched, framed or copied; the two the app already linked keep their own door and their named windows. **Teachers is a list kept by hand** — two tables in the one file, the teachers in Ed's order and the lessons with their passages — because an open search returns anything at all, and a list is a judge-

ment someone can be asked about. A lesson's player is YouTube's privacy-enhanced embed, and it is not built until the reader taps: nothing from YouTube loads on a verse nobody asked to watch. **The art stays off the first-visit path.** Gabe's lamp in the header is inlined at a kilobyte; his full picture is its own file, named once in the stylesheet and drawn as a background, so it loads only where it is shown and the one file stays under its gzipped ceiling. The bar along the foot is the old AI strip given six neighbours and plain names; the strip kept its id and its handlers, so the rail it raises did not change.

v60 carried it to the phone the same way. The verse sheet's Teachers and Elsewhere tabs call the very functions the stage calls — a lesson cannot appear on one and not the other — and the lamp in the sheet's header is the HUD's one inlined copy, read back rather than inlined twice. /examine on a phone now walks the hand-off card's own door, and still fetches nothing first. The cost was three kilobytes gzipped, and the first-visit ceiling was raised to 488 in writing rather than quietly.

What is queued

Search honesty defaults, so that quoted text triggers exact matching automatically and a typed phrase gets the same offer treatment a typed word gets. A path from a search result straight into the notebook, because finding something and recording it should not be two separate acts of will. A Septuagint seat in the translation columns, where it belongs, and the research to let the Variant Lens handle it correctly. And the Study Partner learning to narrate what the counter measured, so a number becomes an observation instead of a fact.

None of this is impressive engineering. It is one HTML file, some JSON, a small worker and a stubborn refusal to claim more than can be measured. It was built for my family, so that we would search the Scriptures diligently, daily, together — some of us for whom most Bible software is a wall of small dense text — and every design decision in it was tested against a single question: does this help us, or does it just look capable?

Five loaves and two fishes were not impressive either. They were what was on hand, and they were handed over.

Ed Smith · Smith For Christ
berean-dashboard.pages.dev
Soli Deo Gloria