Zen Working Group
Request for Discussion and Implementation Record

ZDS 0016
Implementation Specification

The zenfmt Server: REST, Streaming, and the Administered Service

STATE
committed

CREATED
2026-08-10

LAST UPDATED
2026-08-11

INTENDED STATUS
Committed

AUTHORS
Vikrant Rathore
Ronak Rathore (assistance)

DISCUSSION

The zenfmt server: reusable CLI and service libraries, REST and streaming conversion, two-role administration on embedded zaxonlite, observability, a themed Zig WebAssembly interface, and reproducible Tika and Docling comparisons

LABELS
server, api, security, web, benchmark

Status of This Memo

This document is an internal Zen discussion record authored in Typst and tracked in git. It intentionally follows RFC-style structure so the design scope, rationale, trade-offs, and operational constraints remain explicit. Documents that still use the placeholder number XXXXX are provisional drafts. Numbered ZDS documents are part of the project record.

Contents

165. Abstract

zenfmt converts documents through a bounded Zig engine, a command-line tool, a typed Python library (ZDS 0014), and a browser WebAssembly build (ZDS 0015). What it cannot yet do is sit on a network and accept documents from other machines. Teams that today run Apache Tika as a sidecar service have no zenfmt equivalent: a process that starts in one command, answers `PUT /api/v1/convert` with Markdown and structured reports. When the deployment warrants

it, the same service authenticates users, records an audit trail, and serves a small administration interface.

This record specifies the zenfmt server: a subproject of this monorepo, started with `zenfmt serve`, that exposes the engine over HTTP with REST and streaming responses. The server is built almost entirely on the Zig standard library. Its core facilities are `std.http.Server`, `std.Io.net`, `std.crypto`, and `std.Io`. The implementation organizes these facilities into two new reusable libraries. `zencli` extracts the comptime flag-table front end that `src/cli.zig` already contains, so the umbrella CLI and the `serve` subcommand share one grammar engine. `zenserve` is a deliberately small service kernel: a bounded HTTP listener, a comptime router, structured logging, a static metrics registry with Prometheus text exposition, health endpoints, and an authentication core with exactly two roles, `administrator` and `user`.

Like Tika, the server runs in two postures. Open mode is one command with no state: bind the loopback interface, convert whatever arrives, expose the converter page. Secure mode adds a data directory: accounts, sessions, API keys, and the audit log live in zaxonlite. This embedded replicated SQLite library comes from the paxos-zig monorepo and is consumed as a pinned package in single-node embedded mode with TLS transport compiled out. The bundled web interface follows the architecture Zig uses to serve its own standard-library documentation (`zig std`): a static HTML shell, one fixed JavaScript glue file, and a WebAssembly interface module compiled from Zig that renders every page, styled with vendored daisyUI 5 components. No CDN, no npm build, no client-side framework, and no interface logic outside Zig.

This record is the normative implementation blueprint for that server: the libraries, the HTTP surface, the storage schema, the observability contract, the interface, the build integration, the tests, and the delivery phases. Release 0.3.0 implements this record. The acceptance gates below describe the shipped contract and distinguish the few service extensions that remain future work.

166. Introduction

A document converter earns a network interface the moment more than one program wants it. Upload pipelines, search indexers, chat systems, and retrieval stores all need the same operation. They send bytes of an office format and receive Markdown plus a manifest. None of them want to link a native library or shell out to a CLI from inside a request handler. Apache Tika recognized this years ago: `tika-server` is often deployed not because Tika the library is hard to embed but because a converter behind a port is easier to operate, meter, and upgrade than a converter inside every application.

The zenfmt server holds itself to the standard the rest of the monorepo has set:

- the shortest invocation is complete: `zenfmt serve` and the service is useful, exactly as `zenfmt report.docx` and `zenfmt.convert(...)` are;
- every failure is an Elm-style report that states what happened, where it happened, what the server did instead, and what the caller can do. The same report renders as structured JSON for API callers and as readable panels in the interface;
- every resource is bounded before the first request is accepted: connections, request bytes, concurrent conversions, queue depths, log line length, metric cardinality;
- the engine's guarantees survive the transport: one arena per conversion, `convert` never returns an error union, the manifest is canonical, and the server never invents a report the engine did not produce.

The server is also the first zenfmt consumer with persistent multi-user state. Rather than adopting a general database dependency, it uses zaxonlite, the insan.ai embedded SQLite library

already hardened in the paxos-zig monorepo and published at `github.com/insanai/zaxonlite`. In embedded single-node mode zaxonlite is an ordinary durable SQLite store with a serialized writer and pooled readers; the same storage code carries an upgrade path to a replicated multi-node deployment that this record deliberately leaves for a future ZDS.

166.1. Relationship to existing records

- ZDS 0002 remains authoritative for the engine, the CLI's conversion grammar, limits, and the TigerBeetle-derived coding standard. The server changes no conversion semantics.
- ZDS 0013 remains authoritative for the layered IR and writer lowering.
- ZDS 0014 established the pattern this record reuses twice: a language surface in its own subdirectory, orchestrated by the root `zig build` graph, returning the artifact ensemble from memory. The server's JSON result envelope mirrors the Python `Conversion` model deliberately.
- ZDS 0015 established the browser profile, the no-framework web posture, and the accessibility gates; the server interface inherits its progressive-enhancement stance and its ban on third-party runtime assets.

When this record and the engine records disagree about conversion semantics, the engine records win; this record decides how HTTP represents them.

166.2. Normative language and completion

Requirements written with **must** or **required**, and declarative contract sentences such as "the listener rejects," are binding for the first server release. **May** marks a permitted choice; **future** marks work that requires another record or an amendment. The implementation is complete when every deliverable and acceptance gate in this record is satisfied, not when a demo server answers one request on a developer machine.

167. Terminology and Scope

- **server**: the long-running process started by `zenfmt serve`;
- **open mode**: the stateless posture with no accounts, data directory, or persistence; comparable to a default `tika-server`;
- **secure mode**: the administered posture with accounts, sessions, API keys, audit, and settings persisted in zaxonlite;
- **zencli**: the new reusable command-line library extracted from `src/cli.zig`: comptime flag tables, help generation, and exit-code policy shared by every zenfmt front end;
- **zenserve**: the new reusable service library: HTTP kernel, router, middleware, streaming helpers, logging, metrics, health, and the authentication core. zenserve never imports zenfmt;
- **store**: the zaxonlite data directory owned by one server process in secure mode;
- **role**: one of exactly two authorization levels, `administrator` and `user`;
- **principal**: the authenticated identity of a request, represented by a session, an API key, or the implicit anonymous user of open mode;
- **ui module**: the Zig WebAssembly application (`zenfmt_server_ui`) that renders and drives the web interface;
- **glue**: the fixed JavaScript adapter between browser APIs and the ui module. It is the only JavaScript in the product;
- **command protocol**: the versioned message contract between the ui module and the glue;
- **conversion request**: one HTTP request that carries document bytes and returns an artifact, reports, and a manifest;
- **envelope**: the JSON response object carrying status, artifact, resources, manifest, reports, and exit class;

- **streaming response**: chunked transfer of artifact bytes, an NDJSON record stream, or a `text/event-stream` channel;
- **report**: zenfmt's structured diagnostic value; server-origin reports use the `server.` code prefix and the same shape.

In scope:

- the complete first server release described by this record;
- the `zencli` extraction and the `serve` subcommand grammar;
- the `zenserve` library and its bounded HTTP kernel;
- the REST and streaming API, its envelope, and its error contract;
- open and secure modes, the two-role model, and session/API-key auth;
- the zaxonlite storage schema, migrations, and operational lifecycle;
- logging, metrics, health, audit, and the admin event stream;
- the embedded web interface, its vendored assets, and its accessibility gates;
- root build integration, tests, benchmarks, documentation, and release gates.

Out of scope:

- TLS termination inside the process (the standard library ships no TLS server; the deployment section defines the reverse-proxy contract, and native TLS is future work);
- multi-node zaxonlite replication and high availability;
- an asynchronous job queue, callbacks, or webhooks for long conversions;
- per-user quotas, billing, or metering beyond the metrics catalog;
- OpenID Connect, LDAP, or any external identity provider;
- Python or browser client libraries for the server API;
- changing any reader mapping, writer lowering, or engine limit.

167.1. Deliverable contract

Deliverable	Required implementation outcome
`cli/zencli/`	The extracted comptime command/flag library, adopted by the existing conversion grammar without behavior change, plus the `serve` subcommand registration.
`server/zenserve/`	The bounded HTTP kernel, router, middleware, streaming helpers, observability core, and auth core, free of any zenfmt import and unit-tested in isolation.
`server/src/`	The zenfmt server application: modes, REST surface, storage, interface handlers, and the `run` entry point invoked by the CLI.
storage	The zaxonlite schema, migrations, exactly-once bootstrap, query limits, backup path, and retention jobs specified here.
interface	The embedded interface in the autodoc pattern: static shell, fixed glue, generated daisyUI stylesheet, and the `zenfmt_server_ui` wasm module with its command protocol, size budget, CSP, and accessibility gates.
build graph	`build/server.zig`, the new root steps, `fmt_paths` and package `.paths` entries, the pinned zaxonlite dependency with `tls = false`, the `-Dserver` build option, and single executable release packaging with no runtime sidecars.

Deliverable	Required implementation outcome
tests	zenserve unit tests, end-to-end loopback API tests, representative authorization and protocol tests, restart persistence tests, and the browser smoke test.
documentation	The book's server chapter, the report-code reference additions, and the deployment guide.
benchmark	Two reproducible comparisons that reuse `benchmarks/corpus.json`: a loopback server benchmark against Apache Tika Server and an extension of the existing native converter benchmark that adds Docling beside AnyDoc and Pandoc.

168. Problem Statement

Every existing zenfmt surface is in-process. A team that wants conversion as a service today must wrap the CLI in their own server, and that wrapper inherits the same problems ZDS 0014 catalogued for Python `subprocess` wrappers, plus HTTP-specific ones:

- process-per-request startup cost, or a hand-rolled pool;
- text and exit codes instead of typed reports; the wrapper either parses stderr or discards diagnostics;
- no shared enforcement point for size limits, rate limits, or authentication. Every wrapper reinvents them, usually late and differently;
- no capability discovery: clients hard-code the format list and drift from the deployed release;
- no observability: the operator of the wrapper cannot answer "how many conversions failed this hour and why" without building logging and metrics themselves.

The alternative teams reach for is Apache Tika's server, which answers the operational questions but changes the answer's content: a JVM dependency, different format coverage, no manifest, no loss pricing, and diagnostics that do not survive the trip. The gap is a first-party server that speaks the engine's own contracts over HTTP.

There is a second, structural gap. The monorepo has no reusable server substrate and no reusable CLI substrate: `src/cli.zig` owns a good comptime flag table that nothing else can use, and nothing owns HTTP at all. Writing the server as one monolithic application would leave the next service to start from zero again. insan.ai has more services planned. The libraries are as much the deliverable as the server.

169. Goals and Non-Goals

169.1. Goals

- Make `zenfmt serve` a complete Tika-class deployment: one command, no configuration file, loopback bind, converter UI, REST API, health, and metrics.
- Make secure mode a deliberate, explicit upgrade: `--secure --data-dir` turns on accounts, sessions, API keys, audit, and the admin interface, with a printed one-time bootstrap credential.
- Keep the dependency surface at exactly one new Zig package: `zaxonlite` (pinned release, `tls = false`); everything HTTP is the standard library, and everything else is this repository's own modules.
- Preserve engine semantics exactly: the same bytes convert to the same artifact, manifest, and reports through the server, the CLI, and the Python library on the same release.

- Apply the Elm-style diagnostic contract to every server-origin failure with stable `server.*` codes, each asserted in tests.
- Bound every resource statically: connection slots, header bytes, body bytes, concurrent conversions, password verification, rate buckets, and audit retention. No unbounded queue exists anywhere in the design.
- Stream where streaming matters: chunked artifact responses and NDJSON batch results over `std.http.Server.respondStreaming`.
- Ship observability that an operator can consume with stock tools: logfmt or JSON lines on stderr, Prometheus text exposition on `/metrics`, liveness and readiness endpoints, and an audit trail in the store.
- Serve the interface from the binary: the static shell, the fixed glue, the generated daisyUI stylesheet, and the compiled ui module, all embedded at compile time; the server runs offline and never references a CDN.
- Preserve single executable distribution. A released `zenfmt` with server support contains the engine, HTTP service, zaxonlite and SQLite code, migrations, shell, glue, stylesheet, and ui wasm. It requires no adjacent bundle, Java runtime, Python environment, model directory, npm tree, or repository checkout. The `-Dserver=false` release remains a smaller single executable converter.
- Keep the interface in Zig: every page, fragment, and state transition lives in the ui module, unit-testable on the host; the glue contains infrastructure only and no product logic.
- Keep zenserve and zencli reusable: no zenfmt import, no server-specific assumption, unit tests that run without a network.
- Follow the TigerBeetle-derived standard of ZDS 0002 throughout: bounded loops, no recursion, files under 1000 lines, functions under 70 lines, assertions on every state transition.

169.2. Non-Goals

- Terminating TLS in-process. The record defines the reverse-proxy deployment contract instead and reserves native TLS for a future record.
- Replicated or highly available deployments. The storage schema is written so a future multi-node record changes the `open` call, not the schema.
- A plugin or filter marketplace over HTTP. The server compiles the same default bundle as the CLI; filters remain a build-time decision (ZDS 0002).
- Persisting document content. Uploaded bytes and converted artifacts never touch the store or the filesystem; conversions are memory-only.
- A JavaScript build pipeline. If an asset is not a committed file or a product of `zig build`, it is not used.
- Fine-grained permissions. Two roles are a feature: the matrix stays small enough to test exhaustively.

170. Design Overview

The server is three layers with strict dependency direction:

```
┌──────────────────────────────────────────────────────┐
│ server/src: the zenfmt server application            │
│   modes, routes, storage schema, interface pages     │
│   imports: zenfmt, zenfmt_capabilities, zenserve, zencli,  │
│            zaxonlite                                 │
├──────────────────────────────────────────────────────┤
│ server/zenserve: the service kernel (reusable)       │
│   listener, connections, router, middleware, streaming,    │
```

```
|   logging, metrics, health, auth core                    |
|   imports: std only                                      |
├──────────────────────────────────────────────────────────┤
| cli/zencli: the command-line kernel (reusable)           |
|   comptime command and flag tables, help, exit codes     |
|   imports: std only                                      |
└──────────────────────────────────────────────────────────┘
```

The application layer is the only place where zenfmt, zaxonlite, and zenserve meet. zenserve knows nothing about documents; zencli knows nothing about HTTP; the engine knows nothing about either. This is the same discipline that keeps `core/` free of per-format knowledge (ZDS 0002), applied one level up.

One request travels the system as follows. A service task owns the connection and parses the head with `std.http.Server`. Middleware assigns a request id, records the start time, resolves the principal, and checks the route's required role. The conversion handler reads the body into the request arena under the body limit, builds `zenfmt.ConvertOptions` with `OutputSpec.memory`, and calls `zenfmt.convert` under the conversion semaphore. The engine returns a `Conversion` whose arena owns artifact, resources, manifest, and reports; the handler serializes the negotiated representation, the middleware records metrics and the log line, and one `deinit` releases everything. A failed conversion follows the identical path with a failure envelope; the handler has no error-union branch because `convert` does not return one.

170.1. Modes

Aspect	Open mode	Secure mode
start	`zenfmt serve`	`zenfmt serve --secure --data-dir PATH`
bind default	`127.0.0.1:8998`	`127.0.0.1:8998`
principal	implicit anonymous `user`	session, API key, or rejected
store	none; nothing touches disk	zaxonlite directory, `0o700`
interface	converter page only	login, converter, account, admin
audit	none	written to the store
operational plane	open	open (`/healthz`, `/readyz`, `/metrics`)
conversion API	open	authenticated
admin API	absent (404)	`administrator` only

Open mode is not a degraded secure mode; it is the deliberate small deployment, and it must stay as capable as a default Tika server. Secure mode is not a flag sprinkled through handlers; the route table declares a required role per route, and open mode satisfies `user` routes with the anonymous principal. The admin plane is compiled in but unrouted in open mode, so the role matrix is the single authorization mechanism in both postures.

Binding a non-loopback address in open mode is permitted but loud: the server emits the `server.open-network-bind` warning report at startup, stating that every network peer can convert documents anonymously and directing the operator to `--secure` or a firewall. The default port 8998 is deliberately adjacent to Tika's 9998 so side-by-side migration needs no port juggling, while remaining distinct.

171. The zencli Library

`src/cli.zig` already contains a disciplined front end: a comptime flag table that generates parsing, validation, and `--help` text from one declaration, and the four-value exit-code policy (`0` success, `1` conversion, `2` usage, `3` limit). It is currently trapped inside the conversion grammar. zencli extracts the machinery without changing the behavior of a single existing invocation.

171.1. Shape

zencli lives at `cli/zencli/src/root.zig` as module `zencli`, importing only std. It owns:

- `Flag`: the existing declaration shape with a long name, short name, value arity, and help line, moved verbatim;
- `Command`: a name, a flag table, a positional contract, and a help epilogue; commands compose into a comptime command table;
- `parse`: bounded argv iteration producing a typed result or a usage diagnostic; parsing never allocates beyond the caller's arena and never recurses;
- help generation: `--help` output assembled at comptime, as today;
- exit-code policy: the shared `exit_ok`, `exit_conversion`, `exit_usage`, `exit_limit` constants and the mapping from `report.ExitClass`.

`src/cli.zig` becomes zencli's first consumer: its flag table and positional grammar re-declare through zencli types, its parser calls zencli's, and its output is byte-identical. The repository has no CLI behavior suite today, so the extraction's first deliverable is that suite: help-text snapshots, parse edge cases, exit codes, and usage-report rendering, written against the current `src/cli.zig` and wired into `zig build test`. The suite then runs unchanged across the extraction and gates it: any help-text or exit-code drift is a test failure, not a review discussion.

171.2. The subcommand rule

The umbrella CLI today is single-verb: `zenfmt [options] INPUT`. This record introduces the first subcommand, and the grammar must stay unambiguous for existing users:

- if the first positional argument is exactly `serve`, the invocation is the serve subcommand and the remaining arguments parse against the serve flag table;
- every other invocation parses exactly as today, including inputs that happen to be files named `serve`. Those are written `./serve`, and the usage diagnostic for a bare `serve` with no serve flags says so explicitly;
- `zenfmt serve --help` documents the server; `zenfmt --help` gains one line pointing at it.

Future subcommands (none are promised) would join the same comptime command table; zencli rejects a command table with a name that collides with a flag alias at compile time.

172. The zenserve Library

zenserve is the service kernel: everything a bounded Zig service needs to answer HTTP, observe itself, and authenticate principals. It contains nothing about documents. It lives at `server/zenserve/src/` as module `zenserve`, imports only std, and is unit-tested without sockets wherever the design allows (parser-level tests construct `std.http.Server` over in-memory reader/writer pairs).

172.1. HTTP kernel

The kernel is a classic bounded threaded design, chosen over an event loop because conversions are CPU-bound and arena-heavy; a service task per active connection with a hard cap is simpler to reason about, and the cap is the point. A connection's task spends most of its life parked in a

read, so the task count tracks the connection slots; the processor bound is a separate conversion semaphore. Kernel concurrency uses `std.Io` tasks in one bounded group rather than raw threads, because only Io tasks can be cancelled portably (the acceptor blocked in `accept` is unblocked by cancelling its task; a connection blocked in a read is unblocked by shutting its socket down).

- One acceptor task owns the `std.Io.net` listener and hands each accepted connection to a service task.
- One service task per active connection, capped by the connection slots; the CPU-bound work is bounded separately by the conversion semaphore (default: logical CPU count, flag-capped), so parked tasks are cheap and conversions never oversubscribe the processor.
- Connection state is a static array of `max_connections` slots (default 128) allocated once at startup. The slot itself requires no heap allocation per connection. A connection beyond capacity receives `503` with `Retry-After` and a `server.busy` report envelope, then closes.
- Each slot owns fixed receive and send buffers (16 KiB each) handed to `std.http.Server.init`; a request head that does not fit is a `431` with `server.head-too-large`.
- Keep-alive is supported with a bounded request count per connection and an idle deadline; header read and body read carry deadlines so a slow-loris peer costs one slot for a bounded time, never forever. The standard library exposes no per-read socket timeout, so deadlines are enforced by a watchdog task: each slot publishes its current deadline in an atomic before parking in a read, and the watchdog scans the slot array once per second and shuts down (`Stream.shutdown`) any expired socket, which wakes the parked read with end-of-stream. Deadlines therefore land within one watchdog tick of nominal.
- Request bodies stream through `readerExpectContinue`, honoring `Expect: 100-continue` so oversized uploads are refused from the `Content-Length` before the client sends a byte when possible, and at the byte cap otherwise (`413`, `server.body-too-large`).
- Every request executes inside a per-request arena reset after the response flushes; handlers allocate only from it.

172.2. Router and middleware

The route table is comptime data, exactly like the flag table and the plugin descriptor tables:

```
pub const Route = struct {
    method: std.http.Method,
    path: []const u8,          // literal segments and one optional {param}
    role: Role,                // .anonymous, .user, .administrator
    handler: *const fn (*Context) HandlerError!void,
};
```

Matching is a bounded segment walk with no regular expressions, dynamic registration, or allocation. A route table with duplicate method/path pairs fails to compile. The middleware chain is likewise a fixed comptime list, executed in order with no ability to re-enter: request id, timing, principal resolution, role check, rate check, handler, metrics, log. Each element is a plain function; there is no dynamic middleware interface to misuse.

`Context` carries the request, the arena, the resolved principal, the request id, and the response helpers, including:

- `respondJson` serializes envelopes through `std.json`.
- `respondStream` uses chunked transfer with an explicit flush contract.
- `respondNdjson` writes one JSON document per line and flushes each record.

172.3. Observability core

Logging. zenserve installs the process `std.log` function and emits one structured line per record to stderr: logfmt by default, JSON lines with `--log-format json`. Fields are a closed set (time, level, event, request_id, method, path, status, duration_ms, bytes_in, bytes_out, principal, and event-specific pairs from a bounded list). Values are escaped and truncated at a fixed length. A log line can never contain document content, credentials, or token material. The API for emitting a line takes typed fields rather than preformatted strings, which makes that sentence enforceable.

Metrics. The registry is a comptime-declared set of counters, gauges, and fixed-bucket histograms with atomic cells. Label sets are comptime enums (route name, method, status class, reader format, writer format, outcome), so cardinality is bounded by construction and a scrape allocates nothing. Exposition is the Prometheus text format, rendered directly to the response writer.

Health. `/healthz` answers `200` whenever the process can answer at all (liveness). `/readyz` consults registered readiness checks. These include the kernel's task group and, in secure mode, a store ping. The endpoint answers `503` with the failing check names until all pass (readiness). Both endpoints are unauthenticated in both modes; they expose no data beyond check names. Release 0.3.0 serves the operational plane on the main listener. A separate operational listener requires another record because a parsed but ignored address flag would be worse than one explicit port contract.

172.4. Authentication core

zenserve owns the mechanism; the application owns the storage. The library defines:

- `Role`: `anonymous`, `user`, `administrator`. The values are ordered, so a route check is one comparison;
- password hashing with `std.crypto.pwhash.argon2` (Argon2id), PHC-string encoded so parameters travel with the hash and can be raised later without a migration; first-release parameters are the OWASP interactive profile (19 MiB memory, 2 iterations, parallelism 1), which is exactly the standard library's `Params.owasp_2id` constant, cited in one place with the rationale. Because each verification costs 19 MiB, concurrent verifications are bounded by a static counter with a default of 2. An arrival beyond the bound promptly receives `server.busy`;
- opaque tokens: 256-bit values from the Io interface's cryptographically secure generator (`Io.random`; `std.crypto.random` no longer exists in Zig 0.16), transmitted once, stored only as SHA-256 digests; comparison is constant-time over the digests;
- credential presentation on the wire: a session travels only in the session cookie; an API key travels only as `Authorization: Bearer zfk_<id>.<secret>`, where the public id half selects the row and the secret half is digest-compared in constant time. No other presentation is accepted;
- session and API-key descriptors as plain structs plus a `Store` interface (vtable) that the application implements over zaxonlite. The operations are lookup by digest, insert, touch, and revoke. This boundary keeps zenserve storage free;
- fixed-size rate buckets keyed by principal or peer address with LRU eviction over a static array; the login route and the conversion routes attach separate bucket policies.

zenserve deliberately does not implement JWTs, OAuth, or password reset email. Two roles and opaque revocable tokens cover the server's stated scope, and everything in this list is testable without a network.

173. The REST and Streaming API

All application routes live under `/api/v1`; the operational plane (`/healthz`, `/readyz`, `/metrics`), the OpenAPI document, and the interface live at the root. Versioning is by path segment; `v1` semantics are frozen once the record commits, and breaking changes require `v2` alongside `v1`.

173.1. Route table

Method and path	Role	Mode	Contract
`POST /api/v1/convert`	user	both	Convert one document from the request body; `PUT` is accepted as an alias for Tika muscle memory.
`POST /api/v1/convert/batch`	user	both	Multipart batch in, NDJSON envelope stream out, bounded part count.
`GET /api/v1/formats`	user	both	Capability document: readers, writers, extensions, limits, version.
`GET /api/v1/status`	user	both	Version, revision, mode, uptime, bounded counters; store health in secure mode.
`POST /api/v1/session`	anonymous	secure	Login with name and password; sets the session cookie and returns the session document.
`DELETE /api/v1/session`	user	secure	Logout; revokes the presented session.
`GET /api/v1/session`	user	secure	The current principal, role, and expiry.
`POST /api/v1/session/password`	user	secure	Change own password; revokes every other session of the account.
`GET/POST /api/v1/keys`	user	secure	List and create own API keys; the secret appears exactly once in the create response.
`DELETE /api/v1/keys/{id}`	user	secure	Revoke an owned key; administrators may revoke any key.
`GET/POST /api/v1/users`	administrator	secure	List at most 500 accounts in name order, or create an account with name, role, and a one-time password. The interface filters this bounded result locally.
`PATCH /api/v1/users/{name}`	administrator	secure	Change role, disable or enable, or reset the password to a new one-time value.
`DELETE /api/v1/users/{name}`	administrator	secure	Delete an account and revoke its sessions and keys; the last ad-

Method and path	Role	Mode	Contract
			ministrator cannot be deleted or demoted.
`GET /api/v1/audit`	administrator	secure	The newest 100 audit records, newest first.
`GET /healthz`, `GET /readyz`	anonymous	both	Liveness and readiness as defined above.
`GET /metrics`	anonymous	both	Prometheus text exposition on the main listener.
`GET /openapi.json`	anonymous	both	The embedded OpenAPI 3.1 contract. It is available without a session and describes secure mode authentication where it applies.
interface routes	per page	both	`GET /`, `/login`, `/account`, `/docs`, `/admin/...`, and `/assets/{name}`. Asset names carry content hashes and are immutable.

173.2. The conversion request

The request body is the document. Metadata rides in headers and query parameters, so the body needs no wrapping and `curl` stays one line:

```
curl -s -T report.docx \
  "http://127.0.0.1:8998/api/v1/convert?to=markdown" \
  -H "Accept: text/markdown"
```

- `?to=` selects the writer (default: the bundle's default output format); `?from=` overrides detection; `?strict=` selects the strictness grade with the same spellings as the CLI.
- The input name is used for detection exactly as `InputSpec.bytes` requires. It comes from, in priority order, the `filename` of a multipart part, the `X-Zenfmt-Name` header, a `Content-Disposition` filename, or the fixed name `upload` plus an extension guessed by content sniffing (`detect.sniff`, exposed from the core as a behavior-neutral public export for the server). Detection failure is the engine's `usage` report, not a server invention.
- `multipart/form-data` with a single `file` part is accepted equivalently, because HTML forms and the interface's upload path produce it.
- Limit overrides (`?limit=NAME=VALUE`, repeatable, same grammar as the CLI's `--limit`) are accepted from administrators only; other principals receive `server.limit-override-forbidden`. Server-side caps (body size, conversion concurrency) are never overridable per request.

Content negotiation on `Accept`:

Accept	Successful response
`text/markdown` (default for text-emission writers)	The artifact bytes alone, chunked-streamed, with `X-Zenfmt-Report-Count` and `X-Zenfmt-Exit-Class` summary headers. Callers who want the manifest and reports ask for the envelope instead; nothing is smuggled through headers.

Accept	Successful response
`application/json`	The envelope: `status`, `artifact` (UTF-8 string for `utf8_text` emission, base64 with `"encoding": "base64"` for binary), `artifact_name`, `resources` (name, media type, base64 bytes, digest), `manifest` (the canonical manifest document, embedded verbatim), `reports`, `exit_class`, `source_format`, `output_format`. Field names and semantics mirror the Python `Conversion` model of ZDS 0014.
`application/x-ndjson` (batch route only)	One envelope per line per part, in part order, flushed as each conversion completes.

A failed conversion returns the failure envelope with `status: "failed"`, `reports`, `exit_class`, and no artifact. The HTTP status maps from the exit class: `usage` ⟶ `400`, `limit` ⟶ `413`, `conversion` ⟶ `422`. Report objects are produced by the engine's own `writeJsonOptions` with `include_exit_class`; the server never rewrites a report.

173.3. Tika compatibility boundary

The server is comparable to Tika Server as an operational document extraction service, but version 1 is not a wire compatible replacement. The accepted `PUT` method helps existing operators and scripts, but zenfmt does not expose Tika paths such as `/tika`, `/rmeta`, `/meta`, or `/unpack`. It also does not copy Tika response metadata or error bodies. A migration changes the request path and must choose either zenfmt Markdown or the JSON envelope explicitly. The book must include a short Tika migration table that maps the common content extraction request to `/api/v1/convert` and marks every unsupported Tika endpoint without implying compatibility. Benchmark adapters call each server through its own documented endpoint. They never claim that matching HTTP methods make the protocols equivalent.

173.4. Streaming semantics

- **Chunked artifact.** `respondStreaming` with no `Content-Length`; the handler writes the artifact in bounded slices from the arena and flushes. The engine currently produces the complete artifact before the response begins; true pipelined emission through `OutputSpec.writer` is an optimization the envelope contract already permits (future, no new record needed).
- **NDJSON batch.** At most `max_batch_parts` (default 16) parts per request; parts convert sequentially under one request arena that resets per part, and each envelope line flushes before the next part begins, so a client sees progress without polling.

173.5. Server report codes

Server-origin failures reuse the engine's report struct with the `server.` prefix. Every code below ships with a test that provokes it and asserts the code, the HTTP status, and a non-empty direction, in the manner of the engine's stable-code suite.

Code	Status	Meaning
`server.head-too-large`	431	Request head exceeded the slot's fixed buffer.

Code	Status	Meaning
`server.body-too-large`	413	Body exceeded `--max-body`; refused at `Content-Length` when declared, at the cap otherwise.
`server.unsupported-media`	415	Multipart shape or content type the route does not accept.
`server.missing-input`	400	Empty body where a document was required.
`server.invalid-query`	400	A query parameter carries a value the route does not recognize: an unknown parameter name, or a bad `?strict=` grade.
`server.invalid-request`	400	A JSON body is malformed or contains an unsupported account, credential, role, or length value.
`server.unknown-route`	404	No route matches; the admin plane in open mode answers this identically to a truly absent path.
`server.method-not-allowed`	405	Path exists, method does not; `Allow` header enumerates the methods.
`server.unauthorized`	401	No usable principal on an authenticated route; `WWW-Authenticate: Bearer` is set.
`server.invalid-credentials`	401	Login failed; identical timing and body whether the account exists or not.
`server.forbidden`	403	Principal role below the route's requirement, or ownership rule violated.
`server.password-change-required`	403	Account carries a one-time password; only the password-change route is permitted.
`server.last-administrator`	409	Refused deletion, demotion, or disabling of the final administrator.
`server.rate-limited`	429	Bucket exhausted; `Retry-After` is set.
`server.busy`	503	Connection slots or conversion semaphore exhausted; `Retry-After` is set.
`server.limit-override-forbidden`	403	`?limit=` from a non-administrator principal.
`server.store-unavailable`	503	zaxonlite write or read failed or timed out; the log carries the store detail, the envelope does not.
`server.shutting-down`	503	Request arrived after drain began.
`server.open-network-bind`	not applicable	Startup warning, not an HTTP response: open mode bound a non-loopback address.

Code	Status	Meaning
`server.out-of-memory`	500	The reserved static report, mirroring `core.out-of-memory`: preallocated, allocation-free to emit.

174. Storage on zaxonlite

174.1. Dependency and lifecycle

The server consumes zaxonlite as a pinned package from the published repository. The setup command is `zig fetch --save https://github.com/insanai/zaxonlite/archive/refs/tags/v0.3.0.tar.gz`. The dependency is built with `.tls = false`, which compiles out the OpenSSL mTLS transport entirely: the embedded single-node path has no network listener and no OpenSSL dependency. The release tarball pins its own `paxos`, `sqlite` (3.50.4, `SQLITE_THREADSAFE=1`, FTS5, no load-extension), and `sqlite-vec` dependencies by content hash; zenfmt's `build.zig.zon` gains exactly one new entry.

Lifecycle at startup, in order, all before the listener binds:

- `durability.setSyncMode(.full)` is explicit, so Linux gets the same fsync barrier that Darwin uses by default. It is set once before any store I/O.
- `Node.open(gpa, io, .{ .directory = data_dir })` uses empty `members`. This yields the single-member configuration: commit and apply complete before `exec` returns, no peers, no listener. The directory is created `0o700`; the `LOCK` file enforces one process per store.
- Migrations run single-threaded on the `Node` as described in the next section.
- Bootstrap runs if the store is empty.
- `SharedNode.adopt(gpa, node, .{})` wraps the node for the serving phase: one serialized writer with the default queue depth of 32 and a five-second deadline, four pooled read-only connections with a two-second deadline. Handler writes use `execPrepared` / `execTransaction`; reads use `queryPreparedTyped` with `QueryLimits` set (`max_rows`, `max_bytes`, `max_vm_steps` all non-zero), because a network host must never run an unbounded query. Deadline expiry maps to `server.store-unavailable`, never to a hung request.

Graceful shutdown closes in reverse: drain HTTP, `SharedNode.close`, which closes the node and releases the lock.

174.2. Schema

Version 1, applied by numbered migrations recorded in `meta`; each migration is one `execTransaction` on the pre-adopt `Node`, and a store with a newer schema version than the binary refuses to start with a direction to upgrade the binary.

```
CREATE TABLE meta (
  key   TEXT PRIMARY KEY,
  value TEXT NOT NULL
) STRICT;                       /* schema_version, server_instance_id */

CREATE TABLE users (
  id                   INTEGER PRIMARY KEY,
  name                 TEXT NOT NULL UNIQUE,
  role                 TEXT NOT NULL
                         CHECK (role IN ('administrator', 'user')),
  password_phc         TEXT NOT NULL,
  must_change_password INTEGER NOT NULL DEFAULT 0,
```

```
  disabled             INTEGER NOT NULL DEFAULT 0,
  created_at           INTEGER NOT NULL,
  updated_at           INTEGER NOT NULL
) STRICT;

CREATE TABLE sessions (
  token_sha256    BLOB PRIMARY KEY,
  user_id         INTEGER NOT NULL
                    REFERENCES users(id) ON DELETE CASCADE,
  created_at      INTEGER NOT NULL,
  absolute_expiry INTEGER NOT NULL,   /* created_at + 24h */
  idle_expiry     INTEGER NOT NULL,   /* last use + 2h, touched at most
                                         every 5 minutes */
  peer            TEXT NOT NULL
) STRICT;

CREATE TABLE api_keys (
  id            TEXT PRIMARY KEY,     /* zfk_<base32>, public half */
  secret_sha256 BLOB NOT NULL,
  user_id       INTEGER NOT NULL
                  REFERENCES users(id) ON DELETE CASCADE,
  label         TEXT NOT NULL,
  disabled      INTEGER NOT NULL DEFAULT 0,
  created_at    INTEGER NOT NULL,
  last_used_at  INTEGER               /* coarse: updated at most hourly */
) STRICT;

CREATE TABLE audit (
  id      INTEGER PRIMARY KEY,
  at      INTEGER NOT NULL,
  actor   TEXT NOT NULL,              /* user name, key id, or 'system' */
  action  TEXT NOT NULL,              /* closed verb set, see below */
  subject TEXT NOT NULL,
  detail  TEXT NOT NULL               /* JSON, bounded, never document data */
) STRICT;
CREATE INDEX audit_at ON audit(at);
```

The audit verb set is closed and enumerated in code: `user.create`, `user.role`, `user.disable`, `user.enable`, `user.delete`, `user.password-reset`, `session.login`, `session.login-failed`, `session.logout`, `session.password-change`, `key.create`, `key.revoke`, `server.start`, `server.stop`, `server.bootstrap`. Conversion requests are metered in metrics and logs. They are not audited per request. The audit table records administrative fact, and bounded retention (default: 90 days or 100 000 rows, whichever first, pruned at startup and daily) keeps it from becoming an unbounded log.

174.3. Bootstrap

First start in secure mode, store empty: the server creates `admin` with role `administrator`, a generated 24-character one-time password, and `must_change_password = 1`; prints the credential to stderr exactly once, framed so it cannot be mistaken for a log line; writes the `server.bootstrap` audit record; and proceeds. Until the password is changed, the account can reach only the password-change route (`server.password-change-required` elsewhere). Bootstrap uses the pre-adopt `Node` and is therefore trivially exactly-once under the process lock; no idempotency session is needed for the first release, and the record notes zaxonlite's `execIdempotent` sessions as the tool a future multi-node record would reach for.

175. Observability Specification

175.1. Log events

One line per event, closed field set, logfmt or JSON lines. The event vocabulary is bounded and asserted in tests:

- `http.request`: request_id, method, route (the route pattern, not the raw path, so cardinality is the route table), status, duration_ms, bytes_in, bytes_out, principal;
- `convert.done`: request_id, source_format, output_format, exit_class, report_count, input_bytes, artifact_bytes, duration_ms;
- `auth.login`, `auth.login-failed`, `auth.logout`, `auth.key-used` (first use per key per hour);
- `store.slow` (deadline warnings), `store.error`;
- `server.start` (version, revision, mode, address), `server.stop`, `server.drain`.

Raw request paths, query strings, header values, document names, and document content never appear in logs. The peer address appears only in `auth.*` events.

175.2. Metric catalog

All metrics carry the `zenfmt_` prefix; label values come from comptime enums only.

Metric	Type	Labels
`zenfmt_build_info`	gauge (=1)	`version`, `revision`
`zenfmt_http_requests_total`	counter	`route`, `method`, `status_class`
`zenfmt_http_request_duration_seconds`	histogram	`route`
`zenfmt_http_connections_active`	gauge	none
`zenfmt_http_rejected_total`	counter	`reason` (busy, head, body, rate)
`zenfmt_conversions_total`	counter	`source_format`, `output_format`, `exit_class`
`zenfmt_conversion_duration_seconds`	histogram	none
`zenfmt_conversion_input_bytes`	histogram	none
`zenfmt_conversion_artifact_bytes`	histogram	none
`zenfmt_conversions_active`	gauge	none
`zenfmt_auth_failures_total`	counter	`kind` (password, token, key)

`source_format` and `output_format` label values are the compiled bundle's format identifiers plus `unknown`. This is a closed set at compile time, which is what permits them as labels at all.

176. The Web Interface

176.1. The autodoc pattern

The interface is a Zig program. It follows the architecture Zig itself uses to serve its standard-library documentation (`zig std`; `lib/docs` and `lib/compiler/std-docs.zig` in the Zig distribution): a minimal static HTML shell, one fixed JavaScript glue file, and a WebAssembly module compiled from Zig. The module owns every page, every fragment of markup, and all interface state. The server embeds the shell, glue, two stylesheets, OpenAPI document, and WebAssembly module via `@embedFile`. It serves browser assets from memory at immutable paths that contain content hashes. Nothing at request time reads a filesystem or a CDN.

Artifact	Kind	Authority
`shell/index.html`	committed, static	One root element, the stylesheet and glue references, and the `<noscript>` page. Never grows application markup.
`glue/ui.js`	committed, fixed	Instantiates the module, forwards browser events in, executes the command list out. Infrastructure only: no markup, no route names, no form logic, no zenfmt knowledge.
`assets/daisyui-5.0.45.css`	committed, vendored	The pinned daisyUI component and theme foundation.
`assets/layout.css`	committed, first party	The Material style surfaces, interaction states, responsive layout, and editorial typography.
`openapi.json`	committed, static	The OpenAPI 3.1 contract served at `/openapi.json` and linked from the interface.
`zenfmt_server_ui.wasm`	built by the graph	Everything else: routes, state, rendering, validation, interaction.

The division of authority is the point. Zig's documentation viewer proves the shape: its glue manipulates the DOM and forwards events, while the Zig-compiled module (`html_render.zig`, `Walk.zig`) renders every piece of HTML and holds every piece of state. This record adopts that split verbatim. The glue is written once, reviewed like zenserve code, bounded (the budget is 500 lines), and expected to change only when the command protocol changes. Every product decision, including what a page shows, how a report renders, and what a button does, is Zig code that can be unit tested on the host without a browser.

176.2. The ui module

`server/ui/src/` compiles to `wasm32-freestanding` at `ReleaseSmall` as module `zenfmt_server_ui`, mirroring the `bindings/wasm` discipline: a versioned ABI constant, explicit exports, `String` slices into linear memory, and no host capability beyond the glue's import table. The module imports std only. It renders reports and manifests from the envelope JSON the API already serves, so it needs no engine modules and stays small. The release gate budgets the compiled module at 300 KiB (the engine's browser wasm is 1.7 MiB because it carries nineteen readers; the ui module carries none).

Inside, the module is an ordinary bounded application in the ZDS 0002 style:

- a route enum (converter, API docs, login, account, admin users, admin audit, admin status) with path based navigation: the server serves the same shell at every page path, the glue reports `location.pathname` at startup and on history traversal, and the module's `navigate` command pushes a new path onto the history without a reload;
- one state struct for the route machine; no recursion, and every collection is bounded by a fixed API result limit;

- an HTML builder in the manner of `lib/docs/wasm/html_render.zig`: interpolations are escaped by construction, and daisyUI class names are comptime constants gathered in one theme file;
- the Elm-style report renderer, shared by conversion reports and API failure envelopes, so the interface never paraphrases a report.

The ui state also owns a three-value theme preference: `system`, `light`, or `dark`. The initial value is `system`. The glue reports both the stored preference and the current `prefers-color-scheme` result in the `init` event. It also forwards later media-query changes. The module resolves those inputs, renders the theme selector, and emits the command that applies the matching daisyUI theme. Only the preference is persisted. The server never receives theme state, and no document name, content, conversion result, account field, or navigation history enters browser storage.

176.3. The command protocol

The module and the glue exchange length-prefixed JSON documents in linear memory under a comptime schema with an ABI version checked at instantiation. A mismatch is a load failure rather than a guess (the `bindings/python` loader rule).

Direction	Messages
module → glue (commands)	`patch` (element id, HTML), `title`, `focus` (element id), `navigate` (path, via `history.pushState`), `fetch` (request id, method, path, bounded header set, body view), `download` (name, media type, bytes view), `dialog_open` / `dialog_close`, `clipboard` (text), `theme_apply` (`light` or `dark`), `preference_store` (theme value).
glue → module (events)	`init` (route, mode, stored theme, system color scheme), `route_change`, `color_scheme_change`, `action` (the `data-action` name of the clicked or submitted element plus its serialized form fields), `file` (name, size, bytes copied into module memory), `fetch_done` (request id, status, content type, body).

The network stays in the glue because a freestanding wasm module has no sockets: the module decides **what** to request, the glue performs the fetch with same-origin credentials and hands the response back as an event. The interface is therefore client number one of the public REST API. It calls exactly the routes this record specifies, with no private endpoints, which keeps the API honest and the interface replaceable.

176.4. daisyUI without a framework

daisyUI 5 is a CSS artifact with no JavaScript of its own. It styles markup from the WebAssembly module without knowing who wrote the HTML. The vendor stylesheet and the first party visual system are committed separately. `server/ui/THEME.md` records the exact source, versions, and digests. Regeneration is a documented maintainer action. CI never runs npm or a CSS toolchain. Components that need behavior map to native elements plus one command each: modals are `<dialog>` driven by `dialog_open`/`dialog_close`, dropdowns are `<details>`, tabs are radio inputs, toasts are timed `patch` commands.

The visual language combines Material style surfaces, elevation, visible state, and interaction sizes with bold editorial typography and generous spacing. Routine conversion presents one clear sequence: choose a document, keep the useful defaults, and convert. Detailed controls remain

available but visually secondary. Account changes and destructive administration use explicit labels, consequence text, and confirmation. This keeps familiar work immediate while giving higher risk decisions room for deliberation.

A compact selector in the navigation offers System, Light, and Dark in that order. System is the default and follows `prefers-color-scheme` without a reload. An explicit choice persists in local storage under one versioned key. The glue applies a validated stored choice before it loads the wasm module, which prevents a flash of the wrong theme. If storage is unavailable or corrupt, the glue reports `system` and does not fail interface startup. One accent color and a small layout file complete the visual language. The result must feel modern and calm rather than resemble a dense monitoring dashboard.

The `/docs` page is public in both modes and links the machine readable OpenAPI document. In secure mode, a visitor without a session may read these docs. A visit to a protected account or administration page goes to sign in. The docs explain session and bearer authentication without exposing account data.

176.5. JavaScript requirement

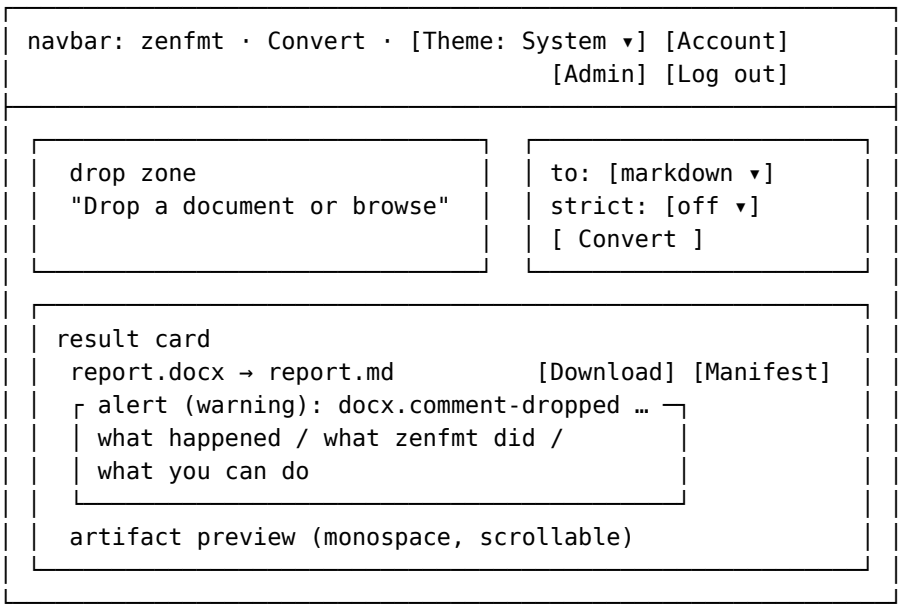
This design requires JavaScript and WebAssembly in the browser. The shell's `<noscript>` page says so and shows the copy-pasteable `curl` line for the conversion API. The API, rather than the interface, is the no-script path. This is a deliberate departure from ZDS 0015's progressive-enhancement stance for the project site, and it is justified differently here: the server interface is an application in front of an API that remains fully usable from anything speaking HTTP, whereas the site's pages are documents. Accessibility is not relaxed. The module renders semantic HTML with landmarks, labels, table headers, live status regions, and native dialogs. The glue places focus inside each opened dialog, and the browser gate exercises the labeled controls through their accessible roles.

176.6. Pages

Page	Role	Content
`/`	user	The converter: drop zone and file picker, output format select (from `/api/v1/formats`), strictness select, convert button; result panel with artifact preview, report panels, and download buttons for artifact and resources. The common navigation includes the System, Light, and Dark theme selector.
`/login`	anonymous	Name, password, error panel; redirects to the password-change form when the account carries a one-time password. Secure mode only; open mode has no login surface.
`/account`	user	Password change; API key list, create (secret shown once in a copy-to-clipboard field), revoke.
`/admin/users`	administrator	Account table with search, status and role filters over the bounded account result, role badges, and a visible last administrator invariant. Create, role change, disable, password reset, and delete use focused dialogs with explicit confirmation.
`/admin/audit`	administrator	The bounded audit table with actor, action, subject, and time.

Page	Role	Content
`/admin/status`	administrator	Version, mode, uptime, active connections, and conversion slot status from `/api/v1/status`.

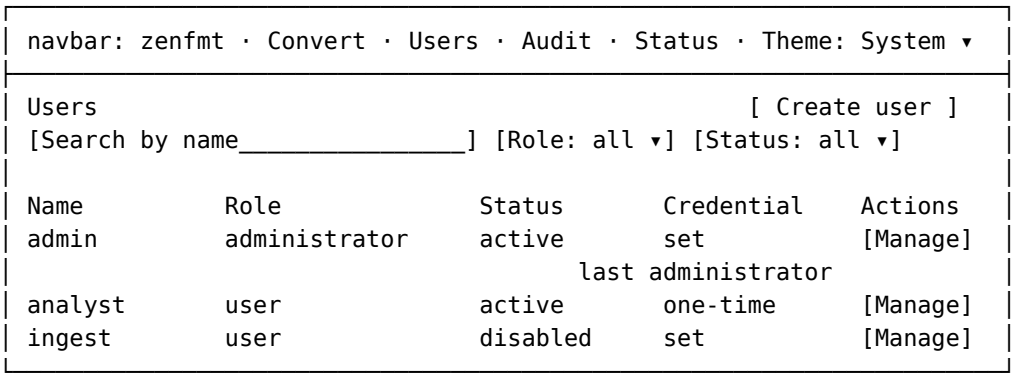
176.7. Converter wireframe



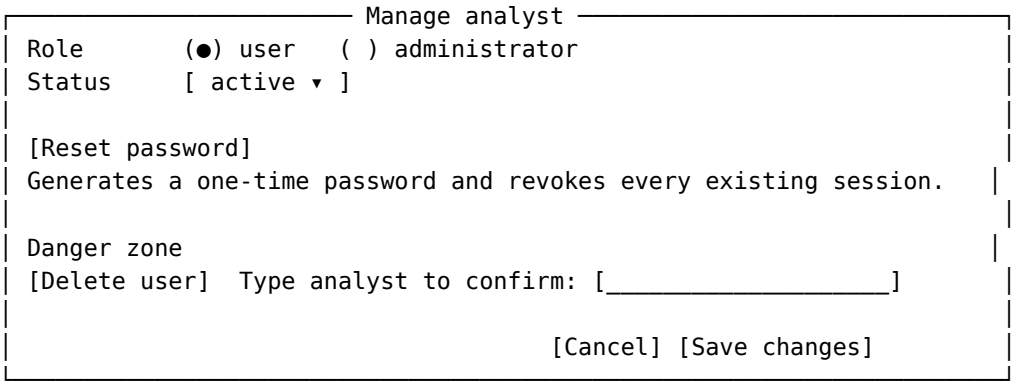
The module renders the form; the glue reads the picked or dropped file through the File API and copies the bytes into module memory as a `file` event; the module issues a `fetch` command posting `multipart/form-data` to `/api/v1/convert` with `Accept: application/json` and renders a progress indicator until `fetch_done` delivers the envelope, which becomes the result card. Downloads are `download` commands over the envelope's artifact and resource bytes. Report panels render the Elm-style sections verbatim. The interface never paraphrases a report.

176.8. User management wireframes

The primary administration view keeps discovery, account state, and guarded actions visible without opening a dialog:



Create and manage actions share one dialog frame. The create form requires a name and role, explains that the generated password is shown once, and places focus on the name field. The manage form makes status and role changes explicit. Destructive actions remain visually separate.



After create or reset, a replacement dialog shows the one-time password once with Copy and Done actions. Closing that dialog discards the plaintext from ui state. The page refreshes from `GET /api/v1/users` after each mutation and announces the result through an `aria-live` status region. The interface disables an impossible last administrator action for guidance, but the API remains authoritative and must still return `server.last-administrator` for stale or concurrent requests. Narrow layouts retain horizontal access to the complete table without hiding account state or actions.

176.9. Session security in the browser

- Cookie `__Host-zenfmt_session` (falling back to `zenfmt_session` without the prefix when serving plain HTTP on loopback, where `__Host-` is invalid): `HttpOnly`, `SameSite=Strict`, `Secure` whenever the deployment is proxied over TLS (`--behind-proxy` asserts this), `Path=/`.
- State-changing interface requests carry a per-session CSRF token in the `X-Zenfmt-Csrf` header. `GET /api/v1/session` returns the token to the ui module after login. The module retains it only in memory and includes it in the bounded header set of each relevant `fetch` command. The glue forwards the header without understanding it. The server requires the token for every cookie-authenticated request whose method is not `GET` or `HEAD`. Bearer and API-key requests are CSRF-exempt by construction.
- CSP on every page: `default-src 'self'; script-src 'self' 'wasm-unsafe-eval'; style-src 'self'; img-src 'self' data:; connect-src 'self'; frame-ancestors 'none'`. The `'wasm-unsafe-eval'` source is the price of instantiating WebAssembly under CSP. It permits wasm compilation only and does not permit JavaScript `eval`. The only script and module the page can load remain the embedded same-origin bytes.
- Login answers in constant time against a fixed dummy hash when the account does not exist; the login route has the strictest rate bucket (10 attempts per minute per peer).

177. The CLI Surface

```
zenfmt serve [options]

  --address ADDR         bind address (default 127.0.0.1)
  --port N               bind port (default 8998)
  --secure               enable accounts, sessions, keys, audit
  --data-dir PATH        zaxonlite store directory (required with --secure)
  --behind-proxy         assert TLS termination and require Secure cookies
  --allow-insecure-network
                         allow cleartext secure mode on a non-loopback bind
  --max-body BYTES       request body cap (default 64 MiB)
  --connections N        connection slots (default 128)
```

```
  --conversions N        concurrent conversion cap (default: logical CPUs)
  --limit NAME=VALUE     engine limit override, repeatable (same grammar
                         as the conversion CLI)
  --log-format FMT       text | json (default text)
  --log-level LEVEL      err | warn | info | debug (default info)
  --no-ui                serve the API and operational plane only
  --drain-seconds N      graceful shutdown deadline (default 30)
  --help                 serve-specific help
```

Flags follow zencli's table grammar; `--secure` without `--data-dir` is a usage error with a direction, not a default path. There is no configuration file in the first release: the flag set is legible, and a config file would create a second source of truth (rejected below). The server exits with zencli's shared codes; startup failures (bind, lock, migration) render Elm-style to stderr.

Examples the documentation must keep runnable:

```
zenfmt serve
zenfmt serve --port 9000 --max-body 256MiB
zenfmt serve --secure --data-dir /var/lib/zenfmt --behind-proxy
```

178. Concurrency and Resource Model

Resource	Default bound	On exhaustion
connection slots	128, static array	`503 server.busy`, `Retry-After`
concurrent password verifications	2, atomic admission counter	`503 server.busy`
service tasks	**179. connection slots**	fixed at startup
concurrent conversions	logical CPUs, semaphore	`503 server.busy`
request head	16 KiB per slot	`431 server.head-too-large`
request body	64 MiB	`413 server.body-too-large`
batch parts	16	`400` usage report
keep-alive requests per connection	1024	connection close
header / body read deadlines	10 s / 120 s	connection close, logged with timeout status
rate buckets	static arrays, LRU	`429 server.rate-limited`
store writes	queue 32, 5 s deadline	`503 server.store-unavailable`
store reads	4 pooled, 2 s deadline	`503 server.store-unavailable`
audit rows	100 000 / 90 days	oldest pruned
log line	4 KiB	truncated with marker

Memory per conversion is governed by the engine's own limits; the server's contribution is `max-body` plus the envelope serialization, both arena scoped. Worst case resident memory is therefore a product of bounded factors. The formula is conversions × (max body + engine budget). The book's server chapter must state the formula and its defaults.

Shutdown: on SIGINT or SIGTERM the acceptor closes, in-flight requests get `--drain-seconds` to finish (new requests receive `server.shutting-down`), `SharedNode.close` runs, and the process exits with the conventional 128-plus-signal code. A second signal exits immediately.

The kernel honors the standard's mechanics: no recursion anywhere (the router is a bounded walk, multipart parsing is an explicit state machine), every loop carries a bound from the table above, every state transition asserts, files stay under 1000 lines and functions under 70.

180. Project and Build Implementation

180.1. Repository layout

```
zenfmt/
├── build.zig                  # + addServer between addCli and addTests
├── build.zig.zon              # + zaxonlite dependency; + "server" path
├── build/
│   └── server.zig             # server build graph (new)
├── cli/
│   ├── src/main.zig           # unchanged thin main
│   └── zencli/
│       └── src/root.zig       # module zencli (new)
├── src/cli.zig                # re-based onto zencli, output-identical
├── server/
│   ├── src/
│   │   ├── root.zig           # module zenfmt_server: Options, run()
│   │   ├── serve_command.zig  # zencli grammar for `zenfmt serve`
│   │   ├── app.zig            # startup order, mode wiring, shutdown
│   │   ├── api/               # convert, batch, formats, status,
│   │   │                      # session, keys, users, audit
│   │   ├── store/             # schema, migrations, users, sessions,
│   │   │                      # keys, audit (zenserve Store impl)
│   │   └── ui/                # pages, fragments, asset routes
│   ├── zenserve/
│   │   └── src/               # module zenserve: http, router,
│   │                          # middleware, stream, log, metrics,
│   │                          # health, auth, ratelimit
│   ├── ui/
│   │   ├── src/               # module zenfmt_server_ui (wasm): routes,
│   │   │                      # state, html builder, command protocol
│   │   ├── shell/index.html   # static shell with the noscript page
│   │   ├── glue/ui.js         # the fixed glue; no product logic
│   │   ├── assets/            # generated daisyUI/Tailwind stylesheet
│   │   ├── MANIFEST.md        # committed-artifact digests
│   │   └── THEME.md           # css regeneration procedure
│   └── tests/                 # end-to-end loopback suites
└── docs/…                     # book server chapter, this record
```

180.2. Module graph

- `zencli`: root `cli/zencli/src/root.zig`, imports std;
- `zenserve`: root `server/zenserve/src/root.zig`, imports std;

- `zenfmt_server_ui`: root `server/ui/src/main.zig`, imports std; compiled twice: to `wasm32-freestanding ReleaseSmall` for the embedded artifact (reusing the target conventions of `build/wasm.zig`, though it shares no code with the engine's browser module), and natively for its unit tests;
- `zenfmt_server`: root `server/src/root.zig`, imports `zenfmt`, `zenfmt_core` (reports), `zenfmt_capabilities`, `zenfmt_build` (version and revision), `zencli`, `zenserve`, `zaxonlite`, and embeds the compiled ui module, shell, glue, and stylesheet as bytes;
- `zenfmt_cli`: gains imports `zencli` and, when the server is compiled in, `zenfmt_server` for dispatching the subcommand.

The entry point the CLI calls is one function:

```
// server/src/root.zig
pub const Options = struct {
    address: []const u8 = "127.0.0.1",
    port: u16 = 8998,
    secure: bool = false,
    data_dir: ?[]const u8 = null,
    behind_proxy: bool = false,
    allow_insecure_network: bool = false,
    max_body_bytes: u64 = 64 * 1024 * 1024,
    connections: u32 = 128,
    conversions: ?u32 = null,    // null: logical CPU count
    limits: zenfmt.Limits = .{},
    log_format: LogFormat = .text,
    log_level: std.log.Level = .info,
    ui: bool = true,
    drain_seconds: u32 = 30,
};

/// Runs until SIGINT/SIGTERM; returns the process exit code.
/// Startup failures render Elm-style reports to `err_out` first.
pub fn run(
    gpa: std.mem.Allocator,
    io: std.Io,
    options: Options,
    err_out: *std.Io.Writer,
) u8;
```

The engine's wasm module graph is untouched. It never sees zencli, zenserve, or the server, just as it never sees `zenfmt_cli` today. The ui module is a second, deliberately tiny wasm root with no engine imports; unlike `zig std`, which JIT-compiles its wasm per request, the server embeds bytes built by the ordinary graph, so releases stay deterministic and offline. The build option `-Dserver=true|false` (default true) controls whether the umbrella binary carries the serve subcommand and the zaxonlite link; `-Dserver=false` produces the pure converter binary for size-sensitive targets, and CI builds both.

180.3. Root build steps

Command	Contract
`zig build serve`	Build and run `zenfmt serve` in open mode (`zig build serve -- --secure --data-dir …` passes flags through).

Command	Contract
`zig build server-test`	zencli, zenserve, and native ui-module unit tests plus the server end-to-end loopback suite.
`zig build server-ui`	Compile the ui module to wasm and report its size against the 300 KiB budget without building the server.
`zig build server-browser-test`	The interface smoke test through the existing browser-test harness (uv-driven, as `site-browser-test` is today).
`zig build benchmark`	Extend the existing native converter lens with the pinned Docling adapter beside AnyDoc and Pandoc.
`zig build benchmark-server`	Run the required loopback zenfmt and Apache Tika Server profiles and write `benchmarks/results/server.json`.

Aggregates: `zig build test` depends on `server-test`; `fmt` and `fmt-check` cover `server` and `cli/zencli` via `fmt_paths`. The release gate adds `server-browser-test`, validates the provenance of checked in benchmark baselines, and does not execute a benchmark. A new reference benchmark run requires both pinned competitors. Local development runs record an unavailable Tika or Docling environment explicitly.

180.4. Integration obligations

- `build.zig.zon`: `.paths` gains `"server"`; dependencies gain the pinned zaxonlite release; `minimum_zig_version` stays 0.16.0, which zaxonlite matches.
- `fmt_paths` in `build.zig` gains `"server"` (zencli sits under the existing `"cli"` entry).
- The docs-drift gate: every `server.*` report code and every serve flag appears in the book's reference chapter; `tests/docs_sync.zig` extends to scan `server/src`.
- The allocation-failure suite's pinned facts (nineteen readers, one static OOM report) are untouched; the server adds its own pinned fact: the static `server.out-of-memory` report and the route count.
- CI (`ci.yml`) needs no new job for the API suite (it rides `zig build test`); the browser smoke test joins the existing browser-test job.
- `benchmarks/benchmark.zig`, `build/benchmark.zig`, and the site aggregator gain the Docling row as one coordinated schema change. A new server runner writes `benchmarks/results/server.json` and reuses the existing corpus verification code rather than implementing another downloader.

180.5. Self contained distribution

The normal release artifact is one executable named `zenfmt`. zaxonlite, SQLite, the schema migrations, and every interface asset link or embed into that executable at build time. Starting open mode from a directory that contains only the executable must serve conversions and the complete web interface. Starting secure mode may create only the operator selected data directory and its durable database files. It must not unpack a runtime bundle, search for adjacent assets, download a component, or invoke another language runtime.

Tika, Docling, AnyDoc, Pandoc, Java, Python, model assets, and LibreOffice are benchmark setup dependencies only. None may enter the release archive or the runtime search path. Tailwind and daisyUI are build inputs whose generated CSS is embedded. The release archive may include license, checksum, signature, and documentation files for humans, but deleting those files after extraction must not affect execution.

The release suite copies each built executable into a fresh empty temporary directory with a minimal `PATH`. For the default build it checks version and revision identity, one direct conversion, open mode health, one API conversion, and every embedded interface asset. It then bootstraps secure mode in a new data directory, restarts against that directory, and logs in. CI also builds the `-Dserver=false` configuration on Linux, macOS, and Windows. No runtime check may refer to the checkout after the executable is copied.

181. Security Considerations

181.1. Threat model

The server accepts untrusted documents from, in open mode, anyone who can reach the port. It executes the format readers, which are the code most influenced by an attacker, against those documents. The engine was built for exactly this (ZDS 0002's archive, XML, and decompression limits), and the server's job is to not undermine it: no server-side format dispatch, no temporary files, no shell, no network egress. The server adds three new assets to protect: credentials and tokens, the audit trail's integrity, and the availability of the process itself.

181.2. Input and process containment

- Uploaded bytes live only in the request arena; no spill files, no buffering to disk, no persistence. `--max-body` bounds them before the engine's `max_input_bytes` sees them.
- The engine runs in-process with its limits enforced per conversion; limit overrides require the administrator role. The conversion semaphore bounds concurrent engine arenas.
- The server makes no outbound connections of any kind; readers already never follow references (ZDS 0002), and the server has no HTTP client.
- Asset and interface routes serve only embedded bytes by comptime name. No filesystem reads occur at request time, so path traversal has no target.

The first release does not isolate each conversion in a child process. Engine limits bound expected work, but they do not contain a panic, a memory safety defect, or a process level out of memory event. Any of those failures can end the server and all requests in flight. The deployment guide must state this directly and require a service supervisor plus operating system or container CPU and memory limits for workloads that accept documents outside a trusted boundary. Readiness must remain false until startup recovery completes. The Tika benchmark records Tika's child process policy because its stronger crash containment is a product difference, not free performance overhead that the report may hide.

181.3. Credential handling

- Passwords: Argon2id PHC strings only; parameters upgradeable in place because verification reads them from the hash. No password ever appears in a log, an envelope, or an audit detail.
- Tokens and keys: 256-bit random, stored as SHA-256 digests, constant-time comparison; the plaintext exists only in the create response. Key ids (`zfk_…`) are public identifiers.
- Sessions: absolute and idle expiry, revoked on password change, revocable by administrators via user disable or delete; enumeration answers are uniform.
- The bootstrap credential prints once and is invalid after first use by construction (`must_change_password`).

181.4. Transport

The standard library provides no TLS server, and linking OpenSSL into the core process contradicts the dependency goal, so in-process TLS is explicitly out of scope. The supported deployments are loopback only (the default bind), a private network with `--secure`, or a TLS-

terminating reverse proxy with `--behind-proxy`. Release 0.3.0 ignores every forwarded origin and peer header, including `Forwarded` and `X-Forwarded-For`. The immediate socket peer remains authoritative for logs and rate limits. The proxy flag is an operator assertion that TLS terminates in front of zenfmt, and it marks cookies `Secure`. Binding a non-loopback address in secure mode without `--behind-proxy` requires the explicit `--allow-insecure-network` flag, whose startup warning report names the exposure: credentials and tokens crossing the network in clear text. The deployment chapter ships nginx and Caddy fragments.

181.5. Denial of service

The resource table is the defense: static slots, deadlines on every read, admission control before work, rate buckets on authentication and conversion, and the engine's own decompression-ratio caps against archive bombs. The explicit worst case is conversions × (max body + engine budget). It is documented so operators size containers deliberately rather than discovering the product in an OOM kill.

181.6. Storage

The zaxonlite directory is plaintext on disk (its documentation says so plainly); the server's contribution is to store nothing secret in recoverable form. It stores only Argon2id strings and SHA-256 digests and sets `0o700`. Full-disk or filesystem encryption is the deployment's job and the guide says so. The `LOCK` file prevents the two-process WAL corruption case. Backup is an operator action through zaxonlite's own tooling against the data directory; v1 exposes no backup endpoint over HTTP (open question 4).

181.7. Supply chain

The interface ships zero third-party executable code: the ui module is compiled from this repository's Zig, and the glue is first-party, fixed, and small enough to review whole. The only third-party interface artifact is the generated stylesheet, pinned by digest in `MANIFEST.md` and verified by a unit test that hashes the embedded bytes. A regeneration is a reviewed diff of file plus digest. The zaxonlite dependency is hash-pinned by the Zig package manager; `-Dtls=false` removes the OpenSSL surface from the build entirely.

182. Testing and Quality Gates

182.1. Unit

- zencli: grammar tables, parse edge cases, help snapshots, exit codes; the conversion CLI's existing tests gate the extraction bit-for-bit.
- zenserve tests router matching, principal roles, bounded rate buckets, Argon2id round trips, token parsing and digesting, metric exposition, log escaping and truncation, multipart limits, and the reusable event ring. These tests are networkless.
- The native ui tests cover event to command rendering, HTML escaping, capability parsing, failure panels, route state, theme resolution, and malformed theme preference fallback.
- Server tests cover envelope serialization, report catalog invariants, the store against a temporary directory, migration refusal for a newer schema, restart, and exactly once bootstrap.

182.2. End-to-end (loopback)

`server/tests/` starts the real server on an ephemeral port and drives it with `std.http.Client`:

- conversion parity: for the corpus sample, byte-identical artifact and manifest versus a direct engine call on the same release;
- content negotiation, chunked conversion, and NDJSON batch ordering;

- anonymous, user, administrator, and one-time password authorization at representative public, conversion, session, and administration routes;
- login, API key creation and use, password change, last administrator refusal, restart, and bootstrap against an empty store;
- oversized heads and bodies, malformed request lines, multipart limits, CSRF rejection, and clean rebinding after shutdown.

182.3. Interface and release gates

The browser smoke test drives the real interface headlessly: convert a fixture through the wasm module, assert the report panel text, exercise login, create a user, change its role, reset its password, disable it, and delete it in secure mode. It verifies local search state, one-time secret dialogs, status announcements, and the last administrator guidance. It also tests explicit theme persistence, System mode, report rendering, and the `<noscript>` page. Release requires: all suites green under `zig build test`; `fmt-check` clean; the docs-drift gate green with the server chapter present; the `-Dserver=false` build green; every native release target passing the empty-directory CLI and server smoke test; and every acceptance example in this record executing unchanged.

183. Benchmark Specification

The benchmark work answers two separate questions. The native converter lens asks how zenfmt compares with Docling, AnyDoc, and Pandoc when each tool turns a local document into Markdown. The server lens asks how the long running zenfmt HTTP service compares with Apache Tika Server over loopback. Results from these lenses must never be merged into one ranking because process startup, HTTP transfer, parser initialization, and service isolation are different costs.

183.1. One corpus and one provenance contract

Both lenses use the exact files named by `benchmarks/corpus.json`. The fetch and digest verification workflow already defined by ZDS 0015 remains authoritative. A server benchmark must not copy files into a second manifest, change a source document, or silently reduce the corpus. Every result records the corpus manifest digest, selected file names, zenfmt revision, tool and runtime versions, configuration, host identity, operating system, architecture, CPU, memory, power policy, date, warmup count, sample count, and raw samples. Missing support and failed conversion remain visible per file. Only successful files shared by a pair contribute to that pair's ratio.

The setup phase may download the corpus, a Tika distribution, Docling wheels, and format specific parser dependencies. It must not download Docling model assets. Timing begins only after setup and digest verification finish. Timed runs have network egress disabled. Any dependency that attempts a download during measurement fails the sample and the publishable run.

183.2. Native converter lens with Docling

The existing `zig build benchmark` harness gains a pinned Docling adapter and keeps its current wall time, CPU time, peak RSS, warmup, median, support matrix, and geometric mean rules. The adapter converts to Markdown through a documented Docling API, consumes the complete result, and discards it only after validation. Its environment records the exact Docling and Python versions, wheel digests, installed extras, the verified absence of model assets, accelerator policy, OCR policy, pipeline, and every external converter version. The reference profile uses CPU execution on the same modest benchmark host used for zenfmt. It permits only Docling backends that produce a `DoclingDocument` directly through `SimplePipeline`, or another pinned path that the adapter proves loads no model. OCR, the standard PDF pipeline, VLM and ASR pipelines,

layout models, table structure models, formula or code enrichment, picture classification, picture description, chart extraction, remote services, external plugins, GPU access, and accelerator specific code are prohibited. The environment contains no model directory or model cache. Network egress is denied, and an attempted model lookup or download fails the entire Docling benchmark.

The pinned adapter owns an explicit model free format allowlist. A corpus file outside that allowlist is recorded as `supported=false` for Docling. In particular, a PDF or image that requires `StandardPdfPipeline`, OCR, or any model remains unsupported instead of changing the profile. This choice makes the comparison narrower than Docling's full product capability, but it keeps the workload appropriate for the simple CPU and memory constrained machines that zenfmt targets. The report labels the row `Docling parser only` and states that it does not measure Docling AI features.

The cold Docling row starts a fresh interpreter for each sample, just as the existing AnyDoc and Pandoc rows include their documented process startup. A second warm Docling profile may keep one `DocumentConverter` alive, but it must appear beside the existing warm in process zenfmt and Python profiles. It must not replace the cold row or appear in a cold process chart. Legacy Office files that require LibreOffice record that dependency and its version. Files that need an unavailable optional backend remain `supported=false` rather than falling back silently.

This record amends the native tool inventory from ZDS 0015. Stable display order becomes zenfmt, Docling, AnyDoc, and Pandoc. The installed zenfmt wheel remains an additional zenfmt surface rather than a fifth independent product. The harness enum, argument parser, support table, head to head pairs, `benchmarks/results/latest.json`, generated Markdown report, book chapter, and site aggregation must all gain Docling together. A checked in result must never be edited by hand.

183.3. Server lens against Apache Tika Server

`zig build benchmark-server` starts zenfmt in open mode on an ephemeral loopback port and starts a pinned Apache Tika Server distribution on a second ephemeral loopback port. The initial reference runner pins Apache Tika 4.0.0-beta-1 because that release provides a documented Markdown handler. A later stable 4.x release may replace it only through an intentional manifest change and a complete result regeneration. The Tika archive or container digest, Java vendor and version, parser package, endpoint, handler, process isolation configuration, limits, and all server flags belong in the result. The runner waits for each readiness signal and verifies a probe before timing.

Each server receives identical source bytes already resident in the runner. The zenfmt adapter uses `PUT /api/v1/convert?to=markdown` with `Accept: text/markdown`. The Tika adapter uses its documented Markdown content handler. Request creation and source file reading occur before the timer. The timed region starts immediately before the client writes request bytes and ends only after it has read and validated the complete response. Connections use the same keep alive policy. Output is consumed rather than aborted, so backpressure and response bytes remain part of the service cost.

The 0.3.0 server runner records these profiles:

- Startup readiness measures process start through the first successful readiness probe. It reports five independent starts and their median.
- Sequential warm latency first warms both services, then sends one discarded request and eight measured requests per corpus file on one keep alive client. It reports the median and p95.

- Concurrent throughput uses concurrency 1, 2, 4, and 8. Each level sends a fixed 24 requests divided across its workers using one small document both services accept. It reports documents per second. This is a short throughput sample, not a saturation or capacity study.
- Memory samples the resident size of each parent and its direct parser children after file and throughput profiles. It reports the largest sampled total. The 0.3.0 runner does not record CPU, byte throughput, rejection latency, deeper descendants, or recovery after overload.

The runner uses the documented product defaults plus the flags recorded in the source. It does not claim matched JVM and native resource caps. JVM heap policy, Tika child process isolation, parser coverage, and diagnostic payload size therefore remain limitations when reading the comparison. A later record may add saturation and matched-cap profiles after their schemas and controls are implemented.

183.4. Correctness before timing

A response qualifies for the 0.3.0 server timing when its status is successful and its body is nonempty. The runner does not require UTF-8 because Tika emits ISO-8859-1 for some accepted documents. It does not yet run the tool neutral fixture oracle or compare zenfmt server bytes with a direct conversion. Those quality and parity checks remain in the native and browser lenses. This is a limitation of the server lens and the site must describe it rather than imply that useful output was proven by the timing gate.

Coverage, quality, latency, throughput, and memory are separate outputs. The report must not collapse them into a universal score or declare one product a winner. Every ratio names its numerator, denominator, shared file count, and whether a larger or smaller value is better.

183.5. Result artifacts and gates

`benchmarks/results/server.json` stores the version and revision, Tika version, startup medians, sampled resident peaks, file medians and p95 values, and throughput samples. `benchmarks/results/latest.json` stores the extended native lens with Docling. `benchmarks/results/results.md`, the book benchmark chapter, and the site dashboard are generated views of those files. The initial schema does not record command lines, archive digests, machine identity, or individual raw samples, so the result is a reference from one machine rather than a fully reproducible laboratory record.

A local development run may proceed with an unavailable competitor and must write an explicit `not_benchmarked` record with the reason. A complete release aggregate requires the pinned Tika and Docling environments, every implemented profile, and the measured revision. Each lens keeps its own version and revision. If one lens completes while another does not, the site may publish the completed current lens beside an earlier reference lens. It must identify each lens independently and state that the full release benchmark is incomplete. It must never relabel an earlier measurement as a current one. The designated host has no discrete GPU and is the same modest machine used for the existing zenfmt reference results. The runner itself does not yet record physical core count, memory capacity, or the absence of model assets. CI validates checked in identities on ordinary changes. It does not perform the expensive reference run.

184. Operational Considerations

The server is one binary, one optional directory, and one port; the deployment guide covers systemd (with `DynamicUser=yes` and `ReadWritePaths=` on the data directory), container images built from the release binaries, reverse-proxy fragments, Prometheus scrape configuration, and the backup procedure (zaxonlite's `backup` produces a plain single-file SQLite database and is safe against a live store because it runs through the store's own API). Log rotation is the supervisor's

job. The server writes stderr and never opens log files. Upgrades are stop, replace binary, start; migrations run forward automatically and never backward, and the refuse-newer-schema rule makes a rollback with an upgraded store a clear error instead of corruption.

The monitoring runbook in the book's server chapter maps the shipped signals to metrics. These signals cover error rate through `zenfmt_http_requests_total`, saturation through `zenfmt_http_rejected_total` and `zenfmt_conversions_active`, and authentication pressure through `zenfmt_auth_failures_total`.

185. Delivery Plan

185.1. Phase 1: kernels

The zencli extraction with bit-identical CLI behavior; the zenserve HTTP kernel, router, middleware, logging, metrics, and health; open mode serving `/api/v1/convert`, `/formats`, `/status`, the operational plane, and the error contract; the adversarial protocol tests. Exit: a Tika-class anonymous converter service passes the end-to-end suite.

185.2. Phase 2: interface

The command protocol and glue, the shell and noscript page, the generated stylesheet with digest tests, the ui module with the converter page, report panels, downloads, and the System, Light, and Dark theme selector; the native golden suite and the browser smoke test. Exit: open mode is complete for demonstration offline, theme behavior passes every first paint and persistence check, and the ui module's size budget holds.

185.3. Phase 3: secure mode

zaxonlite integration (open → migrate → bootstrap → adopt), the auth core wired to the store, sessions, keys, users, audit, CSRF, rate buckets, and the account and admin pages. The role matrix and lifecycle suites. The users page, manage dialog, one-time credential dialog, narrow layout, and complete administration browser flow are required. Exit: secure mode passes every gate in this record's testing section.

185.4. Phase 4: polish and release

NDJSON batch, `--behind-proxy` semantics with the proxy fragments, the native Docling comparison, the server benchmark and its Tika comparison, the book's server chapter, the docs-drift extension, the `-Dserver=false` build wiring, and release-gate integration. The implementation commit moves this record to `committed`. Each benchmark lens remains a separate measured artifact. A completed server result may therefore be published without presenting an incomplete native comparison as a current result.

186. Alternatives Considered

186.1. A third-party Zig HTTP framework

`zap` wraps facil.io (C), and `http.zig` and `jetzig` bring their own allocators, routers, and release cadences. The server needs a small, bounded subset of HTTP; `std.http.Server` plus a kernel the repository fully owns beats an external framework that the coding standard cannot be imposed on. Rejected.

186.2. An event loop instead of threads

io_uring and kqueue single-threaded designs shine at high-concurrency I/O multiplexing. This workload is the opposite shape: few connections, each carrying a CPU-bound conversion.

Threads with static caps match both the workload and `SharedNode`'s blocking API. Rejected for the first release; the listener/service-task seam is where a future record would swap this.

186.3. A supervised conversion process pool

A fixed pool of child processes would contain parser panics, address space faults, and many out of memory failures. Apache Tika 4 uses child parsing for its principal extraction endpoints, so the benefit is concrete. The cost is a second bounded protocol, artifact transfer between processes, worker restart policy, platform specific process limits, and process tree observability. The first release instead relies on engine limits, admission control, an external supervisor, and operating system containment. This is an accepted availability and isolation gap. It is not an assertion that in process conversion is equivalent. A future record should prefer a warm fixed worker pool over one new process per request. The server benchmark in this record preserves the evidence needed to decide when that work is justified.

186.4. In-process TLS

std has a TLS client only. A TLS server would mean linking OpenSSL, which is a dependency the design goals exclude, or hand rolling TLS, which is irresponsible. The reverse-proxy contract is the industry-standard deployment anyway. Deferred, not rejected: a future record can adopt a std TLS server if one lands.

186.5. JWTs instead of opaque tokens

Stateless tokens cannot be revoked without a denylist, which is a store lookup. At that point, the store lookup may as well be the token. The store is embedded and reads are pooled; the latency argument for JWTs does not apply here. Rejected.

186.6. SQLite directly (or flat files) instead of zaxonlite

Linking sqlite3 directly would shed the Paxos journal a single node does not strictly need. But zaxonlite is already the organization's audited SQLite packaging. It provides a pinned amalgamation, WAL discipline, `SharedNode` concurrency, backup, and integrity tooling. It also carries the only credible path to a replicated server without a storage rewrite. The journal's cost at this write volume (administrative writes and session touches) is negligible. Flat files would mean reinventing transactions and losing SQL for the audit queries. Rejected.

186.7. A configuration file

The flag set fits in a systemd unit and a README line; a configuration file creates precedence questions and a parser with its own failure modes. Revisit only if the flag count grows past legibility. Rejected for the first release.

186.8. A separate `zenfmt-server` binary

A second binary doubles release artifacts and splits the user's mental model; Tika's own lesson is that the server is a mode of the tool. The `-Dserver=false` build option covers embedded-minded consumers who want the smaller binary. Rejected.

186.9. Server-rendered HTML with htmx

An earlier draft of this record specified server rendered fragments with vendored htmx driving partial page swaps. It is a respectable design because it degrades to plain forms without JavaScript and needs no CSP relaxation. However, it splits the interface across two authorities: page logic in Zig handlers and interaction semantics in a third-party JavaScript runtime's attribute language, executing over authenticated pages. The autodoc pattern keeps every interface decision in Zig, ships zero third-party executable code, and gains a native golden-test suite no template-over-

HTTP design can match. The lost no-script page is priced openly in the interface section: the API is the no-script path. Rejected in favor of the ui module.

186.10. A SPA framework interface

React, Vue, or Svelte would bring npm into the build, a second language ecosystem into CI, and client side state in a language the coding standard does not govern. The ui module **is** a client side application, but it is written in Zig under this repository's rules. Its state machines are tested natively. Rejected, consistent with ZDS 0015.

186.11. Reusing the ZDS 0015 playground wasm as the interface

The browser playground already converts documents in wasm. That work occurs on the client with the 1.7 MiB engine module, outside the server's limits, authentication, and audit. The server interface must exercise the authoritative server conversion, so it embeds a small ui-only module and calls the REST API. A future hybrid (offline preview via the engine module beside authoritative server conversion) remains possible without changing this design. Rejected for the first release.

186.12. Compiling the interface wasm per request, as `zig std` does

`zig std` rebuilds `main.wasm` on every request so contributors see edits on refresh. This development loop feature requires a compiler dependency at runtime. The server embeds release bytes built by the ordinary graph; the development loop is `zig build serve`, which rebuilds the embedded artifacts like any other dependency. Rejected.

186.13. A live administration event channel

WebSockets and Server Sent Events both add a long lived channel that the first release does not need. Status and audit pages fetch bounded snapshots. A later record may add a live channel after an operational requirement justifies its connection budget and authorization surface. Deferred.

186.14. Storing conversion history

A history table would make the server a document store. It would introduce retention, privacy, and size questions that the design avoids on principle. Operators who need history put a real archive in front of the API. Rejected.

186.15. An asynchronous job API for large documents

Job queues introduce persistence of request state, polling endpoints, and cleanup. Those features form a second product. The synchronous API with generous deadlines and streaming responses covers the engine's actual conversion times, which the benchmark record documents. Deferred until a real workload demands it.

187. Resolved Questions

The discussion phase settled the record's open questions as follows (2026-08-10):

- The default port stays `8998`. Adjacency to Tika's `9998` keeps migration legible while side-by-side operation (including the loopback benchmark) needs no remapping.
- The reverse-proxy fragments are verified through a manual checklist in the deployment guide; CI stays container-free.
- Health, readiness, and metrics remain on the main listener in the first release. A separate operational listener requires a future record.

- No backup endpoint ships in the first release. Operators back up through zaxonlite's own tooling against the data directory. A future endpoint would want `SharedNode` to expose the streaming backup handle that today exists only on `Node`.
- The first release's interface is English. Reports render verbatim through the shared renderer, so a future engine localization story requires no interface rework.

188. Acknowledgements

The shape of this record follows ZDS 0014 and ZDS 0015. The storage design leans on the zaxonlite documentation of embedded mode, `SharedNode`, and durability in the paxos-zig monorepo.

189. References

- ZDS 0002: zenfmt architecture, limits, diagnostics, coding standard
- ZDS 0013: layered IR and writer lowering
- ZDS 0014: the Python library, memory ensemble, and envelope precedent
- ZDS 0015: WASM and the project site, asset, and accessibility posture
- `src/cli.zig`: the comptime flag table zencli extracts
- `bindings/shared/capabilities.zig`: the capability JSON generator
- zaxonlite: `https://github.com/insanai/zaxonlite`, v0.3.0, `Node`, `SharedNode`, durability, and data-directory documentation
- paxos-zig: `https://github.com/insanai/paxos-zig`
- Apache Tika Server 4.0.0 beta 1 documentation (`https://tika.apache.org/4.0.0-beta-1/`): the operational baseline and Markdown handler that the initial server runner measures
- Docling CLI and `DocumentConverter` references (`https://docling-project.github.io/docling/reference/cli/` and `https://docling-project.github.io/docling/reference/document_converter/`): the model free `SimplePipeline` boundary and the native comparison adapter
- Zig autodoc (`lib/docs/` and `lib/compiler/std-docs.zig` in the Zig 0.16 distribution): the shell/glue/wasm interface architecture this record adopts
- daisyUI 5 (`daisyui.com`) over Tailwind 4: the generated stylesheet
- OWASP Password Storage Cheat Sheet: the Argon2id parameter profile
- Prometheus text exposition format