

Refactored Design for `adrscan` – Ensuring Requirements Compliance

This refactored design ensures complete coverage of the functional and non-functional requirements.

CLI Architecture Overview

Core Modules

Module	Purpose
<code>init.rs</code>	Initializes ADR folder, template, and initial ADR
<code>inventory.rs</code>	Walks the repository, hashes file contents, stores inventory snapshot
<code>diff.rs</code>	Compares current inventory to previous, outputs diff JSON
<code>propose.rs</code>	Generates ADR draft from diff, applies template
<code>index.rs</code>	Parses ADRs and regenerates <code>adr/index.md</code>
<code>validate.rs</code>	Lints ADR metadata and format, checks YAML or JSON frontmatter
<code>format.rs</code>	Handles output format conversion (JSON, Markdown, Table)
<code>wasm.rs</code>	Exposes CLI functions as WebAssembly bindings using <code>wasm-bindgen</code>

Command Routing (Clap-based)

```
adrscan <command> [options]
```

Subcommands:

- `init`
 - `inventory`
 - `diff`
 - `propose`
 - `index`
 - `validate`
 - `config` *(optional future: caching, policy, templates)*
-

👉 Functional Requirements Mapping

Requirement	Implementation Notes
Initialize ADR environment with boilerplate and templates	✓ <code>init.rs</code> creates <code>adr/</code> , <code>adr-template.md</code> , and <code>ADR-0000</code>
Generate and store an inventory hash of the repository content	✓ <code>inventory.rs</code> walks files (uses <code>walkdir</code> , <code>ignore</code>), hashes to JSON snapshot
Compare current state to previous snapshot to detect drift	✓ <code>diff.rs</code> loads latest inventory and compares against current state
Generate delta-based ADR proposals from detected drift	✓ <code>propose.rs</code> consumes diff and template, emits markdown ADR draft
Build and update <code>index.md</code> referencing all known ADRs	✓ <code>index.rs</code> scans <code>adr/*.md</code> , parses metadata, updates TOC in <code>index.md</code>
Lint and validate ADR files for metadata and formatting	✓ <code>validate.rs</code> checks metadata schema, structure, frontmatter
Output all command results in JSON by default	✓ All modules use <code>serde_json</code> ; <code>--format</code> flag handled by <code>format.rs</code>
Provide Node.js-compatible WASM bindings	✓ <code>wasm.rs</code> exports core functions via <code>wasm-bindgen</code> , bundled as <code>npx create-adr</code>
Integrate with GitHub Actions workflows	✓ YAML template and binary support already documented
Support markdown and table output formatting via flags	✓ Handled centrally by <code>format.rs</code>

🐱 Rust Crate and Build Integration

```
# Cargo.toml
[lib]
crate-type = ["cdylib", "rlib"]

[dependencies]
clap = { version = "4", features = ["derive"] }
serde = { version = "1", features = ["derive"] }
serde_json = "1"
walkdir = "2"
ignore = "0.4"
anyhow = "1"
```

```
wasm-bindgen = "0.2"
rayon = "1" # for parallel hash walk
```

🐱 Testing Strategy

Type	Tooling	Coverage
Unit Tests	<code>cargo test</code>	All modules individually
Integration Tests	<code>assert_cmd</code> , <code>tempdir</code>	Full CLI execution on sample repo states
WASM	<code>wasm-pack test</code>	Validate Node.js bindings
CI/CD	GitHub Actions	Lint, test, build matrix (Linux/macOS/Windows)

👋 Caching and Offline Mode (Planned)

- `~/.adrscan/cache.json` will store previous snapshot hashes
- Future `adrscan scan --fast` skips rehash if no git/mtime change

🤖 WASM Interface Sample (Node.js)

```
import init, { initWorkspace, runInventory, runDiff, proposeADR } from 'adrscan-wasm';
await init();

await initWorkspace();
const inventory = await runInventory();
const diff = await runDiff(inventory);
const proposal = await proposeADR(diff);
```

👉 Output Format Strategy

All commands return `serde_json::Value` and route to formatter:

```
match output_format {
  Format::Json => serde_json::to_writer_pretty(stdout(), &result),
  Format::Markdown => render_markdown(&result),
```

```
Format::Table => render_table(&result),  
}
```

Distribution

- CLI: `cargo install adrscan`
 - WASM: `npm install -g create-adr-module`
 - GitHub: Prebuilt binaries + Releases + CI validation
-

Licensing & Metadata

- Dual licensed: MIT + Apache 2.0
 - Cargo.toml: `license = "MIT OR Apache-2.0"`
 - SPDX headers in all source files
-

Let me know if you want this turned into an implementation plan, contributor guide, or GitHub repo starter kit.