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

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

🐱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	`cargo test`	All modules individually
Integration Tests	`assert_cmd`, `tempdir`	Full CLI execution on sample repo states
WASM	`wasm-pack test`	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.