ChatGPT

Architecture Overview

The rewritten service will call the Idena node's JSON-RPC endpoints to collect identity data for the current epoch, apply the Proof-of-Humanity eligibility rules, and expose both HTTP APIs and optional CLI commands. We will use Go modules for core logic (epoch/block fetching, identity lookups, filtering) and an HTTP web server (e.g. Echo or Gin) for the UI and REST endpoints. The Idena node's RPC key will be passed securely via environment variables (e.g. `IDENA_RPC_KEY` as shown in the original project docs [1]) and included in the JSON body of each RPC call.

1. Fetch Current Epoch and Start Block

First, call the `dna_epoch` RPC on the node to get the current epoch number (and related info). For example:

```
{ "method":"dna_epoch", "params":[], "id":1, "key":"YOUR_API_KEY",
"jsonrpc":"2.0" }
```

This returns the current epoch (and timestamps) [2]. If the node RPC provides the epoch start block (or an equivalent "snapshot block"), use it directly; otherwise, compute it via an indexer or other chain query. In practice, the rolling indexer or built-in Go service can record each epoch's startBlock in a local DB. For example, a "snapshot" process could query the indexer's SQLite database with:

```
SELECT address, state, stake FROM identity WHERE block_height = {startBlock};
```

(This matches the blueprint's approach of collecting all addresses active at the short-session epoch start [3].) The result is the raw list of identities at epoch snapshot.

2. Export Address List to File

Save the set of all identities at the epoch snapshot into a file (`addresses.txt` or `addresses.json`). For example, write one address per line (or as a JSON array). The whitelist_blueprint's Python prototype does this ("allAddresses.txt") before filtering [3]. Having this file allows the process to be rerun/reproduced. Similarly, fetch the network's current discrimination threshold via the `dna_globalState` RPC and write it to `discriminationStakeThreshold.txt`. (The blueprint saves the "current discriminationStakeThreshold" to a file [3].) This threshold defines the minimum stake required for *Human* identities.

3. Fetch Identity Details via RPC

For each address in the snapshot, call the node's `dna_identity` RPC method to get its current identity record. For example:

```
{ "method": "dna_identity", "params": ["0xABC..."], "id":1,
"key":"YOUR_API_KEY", "jsonrpc":"2.0" }
```

This returns a JSON with fields including `address`, `stake`, `state`, `penalty`, and `lastValidationFlags` (among others). (This usage is documented in the Idena RPC docs [4].) Extract the relevant fields (`state`, `stake`, `penalty`, `lastValidationFlags`) into a Go struct for each identity. This is equivalent to the original code's identity fetcher agents or indexer queries, but done directly via RPC calls in Go.

4. Determine Discrimination Stake Threshold

Call the `dna_globalState` RPC to retrieve the current network parameters; from its result extract the `discriminationStakeThreshold` (the minimum stake for Humans). Write this threshold value to `discriminationStakeThreshold.txt` for record. (The Python blueprint and Go system both expect this file [5].) This threshold (typically a fraction of the total stake) changes each epoch. If `dna_globalState` is not available, one can use the rolling indexer's data or a public API as fallback, but local node RPC is preferred.

5. Whitelist Eligibility Filtering

Apply the eligibility rules to each identity record:

- **State:** Only identities in state *Human*, *Newbie*, or *Verified* are eligible. (All other states – *Zombie*, *Suspended*, *Candidate*, *Killed*, *Undefined* – are excluded.)
- **Stake:** A *Human* must meet or exceed the discrimination stake threshold; a *Verified* or *Newbie* must have ≥10,000 iDNA. (These rules come from the Proof-of-Humanity criteria [6] [3].)
- **No Penalties:** The `penalty` field must be `"0"` (no active validation penalty).
- **No Bad Flip Reports:** The `lastValidationFlags` array must **not** contain `"AtLeastOneFlipReported"`.

For each identity, check these conditions. Ineligible addresses (e.g. humans below stake, "shitflippers", etc.) are filtered out [6] [3]. The remaining set is the epoch's whitelist.

6. Output Whitelist JSONL

Write the filtered identities to a JSON Lines (`.jsonl`) file named `idena_whitelist.jsonl`. Each line is a JSON object with at least `{address, stake, state, ...}` for an eligible identity. This matches the blueprint's output format (`idena_whitelist.jsonl`) [5]. JSONL is chosen so each entry is on its own line. Ensure the list is sorted (e.g. alphabetically by address) for determinism. (The indexer roadmap also recommends sorting before Merkle tree construction [7].) This file can be served or downloaded by the UI.

7. Web Interface and HTTP Endpoints

Integrate with the existing **proof-of-human.work** frontend (`static/index.html`). Add buttons/controls and new endpoints to trigger and display results:

- **Trigger Snapshot:** A new endpoint (e.g. `POST /snapshot`) starts the above process on demand. The web UI can have a "Start Snapshot" button that calls this API.
- **View Summary:** After snapshotting, return a summary (e.g. epoch number, block height, number of identities processed and whitelisted). An endpoint like `GET /whitelist/summary` or the response of `/snapshot` can show these stats.
- **Download Whitelist:** Add an endpoint (e.g. `GET /whitelist/download`) that serves `idena_whitelist.jsonl` as a downloadable file. The UI can provide a "Download Whitelist" link.

Likewise, retain or add the existing routes `/whitelist/current`, `/whitelist/epoch/{n}`, and `/whitelist/check` as described in the original docs [8]. These endpoints should return JSON arrays or objects of eligible addresses and details (for use by other services or frontends). You can also expose `/merkle_root` and `/merkle_proof?address=` if implementing Merkle proofs, as suggested in the indexer roadmap [9].

8. CLI Tools for Debugging

Implement equivalent Go command-line commands for each major step, for example via subcommands or flags. For instance, `idena-auth snapshot` could perform the data fetch & filtering, `idena-auth filter` could apply criteria to an existing file, etc. This matches the roadmap suggestion to provide CLI export tools [10]. Log verbose output so that one can run e.g. `go run main.go --snapshot` or a built binary to perform each action manually. This aids debugging and can be used in scripts (for example, wrapping the CLI in a cron job).

Key Implementation Notes

- **Modules/Structure:** Encapsulate node RPC calls in a Go module (e.g. `rpcclient`), filtering logic in a separate package, and file I/O in utilities. Keep web handlers (Echo/Gin) minimal, calling into these core modules.
- **Concurrency:** When fetching `dna_identity` for many addresses, use goroutines and rate limiting to parallelize RPC calls safely.
- **Secure Key Handling:** Store the node RPC key in an environment variable (e.g. `IDENA_RPC_KEY`) and pass it in the JSON body of each request as shown in the Idena docs [1]. Do **not** log or expose the key.
- **Indexing vs. Direct RPC:** The rolling indexer (SQLite DB) can optionally be used for faster snapshots (it already stores identity history [11]). However, the default should query the node directly as specified.
- **Reproducibility:** Include timestamps and epoch numbers in outputs so snapshots are fully auditable. The blueprint emphasizes a reproducible filter pipeline [3].

By following these steps and rules (as documented in IdenaAuthGo and the whitelist blueprint [6] [12]), the new Go-based implementation will generate the same whitelist output (in JSONL) as before. All core logic will live in Go modules for reuse, with both HTTP endpoints and CLI interfaces provided for operability. The discrimination threshold, identity filtering, and exclusion logic will match the existing criteria [6] [3].

Sources: The design closely follows the original IdenaAuthGo documentation and the reference Python blueprint [6] [12] [10] [4], adapting them into Go and adding the required web endpoints and JSONL output.

---

[1] [6] [8] README.md
https://github.com/ubiubi18/IdenaAuthGo/blob/5efb3a0e11ee6f49e879a4010a0606d2dca11067/README.md

[2] Idena node RPC | Idena documentation
https://docs.idena.io/docs/developer/node/node-rpc

[3] [5] [12] README.md
https://github.com/ubiubi18/whitelist_blueprint/blob/793441a12cf9bc89b52455f8605755b129a06313/README.md

[4] Introduction | idelse
https://idena.gitbook.io/idelse/idena-rpc/introduction

[7] [9] [10] [11] TODO-indexer-merkle.md
https://github.com/ubiubi18/IdenaAuthGo/blob/5efb3a0e11ee6f49e879a4010a0606d2dca11067/TODO-indexer-merkle.md