ChatGPT

Bestehende Logik und Duplikate

Es existieren derzeit mehrere Implementierungen für die Whitelist-Erzeugung und Identitätsabfrage (Python-Skript *whitelist_blueprint*, das Go-Projekt *IdenaAuthGo* mit Indexer, Fetcher, etc.). Das Python-Skript aus dem *whitelist_blueprint* ruft die öffentlichen REST-Endpunkte der Idena-API auf (z.B. `/api/Block/{height}`, `/api/Epoch/Last`, `/api/Epoch/{epoch}/Identity/{addr}/ValidationSummary`), um anhand der letzten Validierungs-Session alle Teilnehmer zu sammeln und nach Status, Freigabe und Einsatzstärke zu filtern [1] [2]. Es verwendet die **dynamische DiscriminationStakeThreshold** für Humans und einen 10.000 IDNA-Schwellenwert für Verified/Newbie, sowie das Bad-Authors-Listing (`/Epoch/{epoch}/Authors/Bad`) zum Ausschluss von Flippern [2] [3].

Im Gegensatz dazu nutzt das Go-Backend (*IdenaAuthGo*) einen **rolling Indexer**, der per JSON-RPC (`dna_identities`, `dna_epochIdentities`) kontinuierlich alle Identitätszustände vom lokalen Knoten abruft. Die Indexer-API stellt Endpunkte wie `/identities/eligible` oder `/api/whitelist/current` bereit, listet aber **statisch alle Humans/Verified/Newbie mit ≥ 10.000 IDNA** als „eligible" – sie ignoriert den dynamischen Schwellenwert und prüft Strafstatus oder Flips erst später in der Whitelist-Berechnung [4] [5]. Die Haupt-API von IdenaAuthGo ruft dann diese Indizes ab und filtert in `buildEpochWhitelist` noch nach Strafstrafen („penalized") und Flip-Reports.

Duplikate: Daraus ergibt sich eine doppelte Funktionalität. Der Python-Blueprint scannt explizit Blöcke nach Transaktionen, während der Go-Indexer dies schon im Hintergrund erledigt. Beide Systeme führen ähnliche Filter (Status, Stake, Flips) jeweils auf eigene Weise aus [2] [4]. In der Praxis bedeutet das, dass – wie in der Analyse festgehalten – Adressen mit Human-Stake über dem dynamischen, aber unter 10.000 IDNA-Schwellenwert im Python-Skript bleiben, im Go-Ansatz jedoch fehlen [6]. Umgekehrt kann der Go-Indexer (vor Abzug der Strafen) Adressen führen, die das Python-Skript später ausschließt. Dies zeigt, dass dieselbe Whitelist-Logik **zweifach implementiert** ist (Python *vs.* Go), aber mit unterschiedlichen Kriterien.

Konsolidierungsempfehlungen

Um Inkonsistenzen zu vermeiden, sollten die Logikwege zusammengeführt werden: Entweder konsequent den dynamischen Schwellenwert verwenden oder klar dokumentieren, warum sich Ergebnisse unterscheiden. Praktisch könnte man **die Indexer-Logik erweitern**, etwa indem statt eines starren 10.000-IDNA-Limits die `discriminationStakeThreshold` für Humans genutzt wird (wie im Python-Skript). Im Go-Code findet man bereits die Funktion:

```
func isEligibleSnapshot(state string, stake float64, threshold float64) bool
{ … }
```

Die Übergabe von `threshold` an diese Funktion ist vorgesehen [7] [8], doch die Indexer-API selbst (`queryEligibleSnapshots`) verwendet bisher fix 10.000 und ignoriert `threshold` [8]. Hier sollte man konsolidieren und ggf. den dynamischen Threshold aus dem Node abfragen (z.B. mittels `/api/Epoch/Last`) und in der Eligibility-Abfrage berücksichtigen.

Zudem könnte die bestehende Python-Logik entweder in Go übernommen oder komplett ersetzt werden. Ist der Go-Indexer stabil, könnte man das Python-Skript (inkl. Block-Scanning) entfallen lassen und stattdessen nur noch auf den Indexer bzw. Idena-API-Aufrufe setzen. Da beide Ansätze aktuell parallel existieren, empfiehlt es sich, **doppelte Code-Teile zu entfernen** und gemeinsame Funktionen (z.B. API-Aufrufe, Filterbedingungen) nur einmal zu implementieren. Dabei hilft die PDF-Analyse (siehe *„Python vs Go Whitelist Logic“* [2] [4]) als Referenz für die zwingend nötigen Filter (Flips/Approved, dynamischer Threshold) und was derzeit fehlt.

Initiale Befüllung des Indexers (Bootstrapping)

Der Rolling-Indexer kann zu Beginn historische Daten von der öffentlichen API holen. Ist die Datenbank leer und `USE_PUBLIC_BOOTSTRAP=true`, wird in `main.go` (rolling_indexer) die Funktion `bootstrapHistory` aufgerufen [9] . Dort wird zunächst das aktuelle Epoch per JSON-RPC (`https://rpc.idena.io/api/Epoch/Last`) abgefragt und dann über die RPC-Methode `dna_epochIdentities` alle Adressen mit Status/Stake für die letzten *n* Epochen geholt. Diese Schnappschüsse werden mit zurückdatierten Zeitstempeln eingefügt, sodass etwa die letzten 3 Epochen abgedeckt werden (konfigurierbar über `BOOTSTRAP_EPOCHS`) [10] [9] .

Für die **aktuelle Epoche** kann man analog vorgehen: Entweder über den Node-RPC (`dna_epochIdentities` mit epoch-Parameter) oder über die offizielle REST-API (`api.idena.io`). Letztere bietet Endpunkte wie `/Epoch/Last` und vermutlich `/Epoch/{epoch}/Identity/...`. Beispiele aus dem Blueprint zeigen:

```
curl https://api.idena.io/api/Epoch/Last          # aktuelle Epoche und
Threshold
curl https://api.idena.io/api/Epoch/{epoch}/Identity/{addr}/ValidationSummary
```

Damit lässt sich in einer Initialisierungsschleife (oder Agent) der Whitelist-Status aller Adressen ermitteln [2] [3] . Da die *curl*-Befehle im Blueprint bereits dokumentiert sind, kann man sie nutzen, um etwa beim ersten Start eines Indexers alle teilnehmenden Adressen zu sammeln und in die SQLite-DB zu schreiben. Achten Sie dabei auf Paging/Tokens (wie in `fetch_bad_addresses`), damit keine Adressen verloren gehen.

Empfohlene Schritte (Konsolidieren & Fixen)

- **Vergleich durchführen:** Identifizieren Sie alle Stellen, wo dieselben Daten auf unterschiedliche Weise abgefragt werden. Prüfen Sie z.B. `cmd/strictbuilder/main.go` vs. `rolling_indexer/main.go` vs. Haupt-*main.go*, welche API-Calls bzw. RPC-Methoden verwendet werden [1] [5] . Entfernen Sie überflüssige Dubletten und verwenden Sie eine gemeinsame Funktion.
- **Threshold-Logik vereinheitlichen:** Entscheiden Sie sich für den dynamischen Discrimination-Stake für Humans (empfohlen) und implementieren Sie dies in der Go-Logik. Beispielsweise könnte der Indexer bei jedem Durchlauf mit `/api/Epoch/Last` den aktuellen Threshold holen und in `queryEligibleSnapshots` einfließen lassen. Die aktuell hart kodierten 10k in `isEligibleSnapshot` sollten überdacht werden [7] [8] .
- **Flips/Strafen prüfen:** Stellen Sie sicher, dass alle Flips (`Epoch/{epoch}/Authors/Bad`) und Straf-Flags entweder im Indexer-Datenbestand (über einen getrennten Flip-Report) oder beim

Whitelist-Build berücksichtigt werden. Der Python-Ansatz zieht die Bad-Authors-Liste explizit – im Go-Hauptcode geschieht das in `buildEpochWhitelist` mit `getPenaltyStatus/hasFlipReport` [11]. Dieser Check sollte zentral erfolgen (nicht einmal im Fetcher, einmal im Server).

- **Bootstrapping aktivieren:** Legen Sie in der Indexer-Konfiguration `USE_PUBLIC_BOOTSTRAP=true` und einen sinnvollen Wert für `BOOTSTRAP_EPOCHS` fest (z.B. 3) [10]. Prüfen Sie, ob die Initial-Datenbank leer ist und ob `bootstrapHistory` tatsächlich ausgeführt wird (Logmeldungen *„Bootstrapping from public API…"* [9]). Falls der Code hier abbricht, legen Sie geeignete Fallback-Aufrufe nach (z.B. erneuter Versuch oder Nutzung der REST-API statt RPC).
- **Offizielle API nutzen:** Nutzen Sie die **Swagger-Doku** (https://api.idena.io/api/swagger/doc.json), um REST-Endpunkte zu ergänzen. Zum Beispiel könnte beim Bootstrapping statt `dna_epochIdentities` auch der REST-Endpunkt für Identitäten eines Epochs abgefragt werden, falls vorhanden. Die Vorlage aus dem Python-Skript zeigt, welche REST-URLs gebraucht werden [2]. Schreiben Sie ggf. ein kleines Go-Agent-Tool (analog zum Python-Skript), das alle benötigten Endpunkte aufruft und die Ergebnisse in das Indexer-DB importiert.

Durch diese Maßnahmen werden redundante Implementierungen konsolidiert und der Indexer erhält zuverlässig alle Identitäten der aktuellen Epoche aus der offiziellen API. Letztendlich sollte **nur eine Whitelist-Generierung** (idealerweise im Go-Backend) aktiv sein und sich auf den Indexer stützen, der seinerseits sauber initialisiert wird.

Quellen: Analysen des Python-Skripts und des Go-Indexers aus dem Repository [1] [2] [9] [10]. These are excerpts from the provided code and documentation illustrating the overlap and suggested bootstrap workflow.

---

[1] [2] [3] [4] [6] [7] [8] Python vs Go Whitelist Logic.pdf
file://file-A1jAXU8zVZaGuNY2u7PZ6h

[5] [11] main.go
https://github.com/ubiubi18/IdenaAuthGo/blob/b57cfb5c5112336d0b207a61c6775a69def8a79c/main.go

[9] main.go
https://github.com/ubiubi18/IdenaAuthGo/blob/b57cfb5c5112336d0b207a61c6775a69def8a79c/rolling_indexer/main.go

[10] README.md
https://github.com/ubiubi18/IdenaAuthGo/blob/b57cfb5c5112336d0b207a61c6775a69def8a79c/README.md