ChatGPT

Idena Node vs Public API Endpoints

The **build_idena_identities_strict.py** script relies on Idena's public API (the indexer API) for several data points. To replicate this with a local node (idena-go), we must find the equivalent JSON-RPC calls or data sources. Key needs are: current epoch info (including the *DiscriminationStakeThreshold*), epoch parameters, block flags, block transactions, "bad authors" lists, and identity validation summaries.

- **Epoch and Threshold**: The Idena node's RPC provides `dna_epoch` (with and without parameters) to fetch epoch info. Calling `dna_epoch` (no args) returns the current epoch and includes `"discriminationStakeThreshold"` [1]. Likewise, `dna_epoch` with a specific epoch as parameter returns that epoch's info (epoch number and threshold) [2]. These correspond to the indexer API calls `/api/Epoch/Last` and `/api/Epoch/{N}` used by the script. For example, `getEpochAndThreshold()` in IdenaAuthGo invokes `dna_epoch` to get `epoch` and `threshold` [1], and `getEpochAndThresholdFor(N)` does `dna_epoch` with `N` [2].

- **First Block of Epoch**: In Idena, each epoch's *validation first block height* is essentially the start block of that epoch. The node RPC's `dna_epoch` response includes the start block (the code refers to `s.State.EpochBlock()` as *StartBlock* [3]). This value matches `validationFirstBlockHeight` in the indexer API. For example, the indexer's database `epochQuery` returns `ValidationFirstBlockHeight` for an epoch [4], while the node RPC returns `StartBlock` for an epoch [3].

- **Block Data and Flags**: The Idena node RPC can return a full block by height or hash. The Blockchain API's `BlockAt(height)` returns a `Block` object that includes the transaction hashes and flags [5]. Notably, the `Block` struct has fields `Transactions []common.Hash` and `Flags []string` [5], where flags can include `"ShortSessionStarted"`, `"IdentityUpdate"`, etc. (see `blockFlags` mapping in the indexer code [6]). Thus, calling `dna_blockAt(height)` (or equivalent) yields the block with its `flags` array and the list of transaction hashes. One can then fetch each transaction's details (via `dna_transaction(hash)`) to get sender/receiver data. The Transaction struct in the node RPC includes fields like `From`, `Type`, `Hash`, etc. [7], which lets us identify addresses that initiated flips or answers in that block.

- **Transactions of a Block**: Since `Block.Transactions` gives only hashes, the local indexer must fetch each transaction to get the sending address. The indexer API's `/Block/{height}/Txs` is effectively equivalent to iterating `BlockAt(height).Transactions` and calling the RPC transaction endpoint (e.g. `dna_transaction`) for details [7]. In practice, for each block we would do:

- `dna_blockAt(height)` → parse `Transactions`.
- For each tx hash, call `dna_transaction(hash)` to retrieve `"from"` (sender address) and other info.

- **Bad-Authors List**: The script calls `/api/Epoch/{epoch}/Authors/Bad` to get addresses who submitted bad flips. There is **no direct RPC method** for "bad authors"; this is computed in the indexer (the SQL `epochBadAuthorsQuery` [8]). To replicate this without the indexer DB, one would need to track flip submissions and evidence via low-level protocol data or the node's state machine (complex). A minimal approach might skip explicitly fetching bad-author lists and only rely on each identity's validation summary (see next bullet) which includes a `penalized` flag.

- **Identity Validation Summary**: The public API `/api/Epoch/{epoch}/Identity/{addr}/ValidationSummary` gives an identity's final validation result (approved/penalized, state, stake) for that epoch. The node RPC does not have a single summary call, but one can derive equivalent info: the node's state (via `dna_identity` or iterating identities) and the node's validation results storage. However, the Dna API (`dna_api.go`) does include methods for identity data. In particular, the RPC `dna_identity` and related methods can fetch an identity's state and flip stats. The **GlobalState** RPC (`dna_globalState`) includes the current discrimination threshold [9]. More directly, IdenaAuthGo's rolling indexer suggests using `dna_epochIdentities` to list identities and their states [10]. One could call `dna_epochIdentities(lastEpoch, 0)` with continuation to get each identity's summary (including fields like `Approved`, `Penalized`, `Stake`, and `State`). Indeed, the indexer's `EpochIdentityValidationSummary` SQL query [11] mirrors fields that the node could supply. In short, the local indexer can obtain each identity's state/stake from the node RPC (e.g. via `dna_identity` or `dna_epochIdentities`) and filter by whether they were penalized or approved.

- **Putting It Together**: In summary, a minimal local indexer can operate by calling the idena-go node's RPC instead of the public API. Key RPC calls are:

- `dna_epoch` (no args) → current epoch, discriminationStakeThreshold [1].

- `dna_epoch` (with epoch) → that epoch's start block and threshold [2].
- `dna_blockAt(height)` (or `dna_blockByHash`) → block with its `Flags` and `Transactions` [5].
- `dna_transaction(hash)` → details of a transaction, including `From` address [7].
- (Optionally) `dna_epochIdentities(epoch, token)` → list of all identities at epoch, including their state and stake [10].
- (Optionally) `dna_identity(address)` or similar → individual identity status.
- `dna_globalState` → current network size and discriminationStakeThreshold [9] (alternate to `dna_epoch`).

Using these, the new indexer can find the short-session blocks (by scanning `dna_blockAt` for `ShortSessionStarted` in flags), collect all senders of transactions in those blocks (using `dna_transaction`), and then for each address call the node to get state, stake, and whether it was penalized. This replicates what the script does via the public API, but entirely from the local node.

Implementation Notes

Given the requirements:

- **Minimal CLI**: A simple command-line tool (in Go or Python) is sufficient. It can sequentially make the above RPC calls and apply the same filters (non-approved identities, low-stake, penalized, bad flips) as the Python script. For example, the logic in *build_idena_identities_strict.py* (filtering by

state and stake vs thresholds) can be ported to Go/Python. The tool might not need a database – it could output directly the whitelist JSON or update in-memory data.

- **Dependencies**: No heavy external services are needed. It can use Go's `net/rpc/json` or Python's `requests` to hit the local node's RPC endpoint (usually `http://localhost:9009`). If using Go, one can even import the `idena-go/api` libraries (as in [53]) for structured data, but plain JSON-RPC calls are simpler and lighter. The only dependency is connectivity to an idena-go node with RPC enabled.

- **Integration with IdenaAuthGo**: The tool can be included in the IdenaAuthGo codebase (as suggested by the rolling indexer code [64]). Indeed, IdenaAuthGo's rolling indexer already sets up HTTP handlers for epoch info that use `dna_epoch` [1] [2]. A similar approach can be used: extend the indexer service to expose endpoints or directly embed the logic. Since the node is local, authentication keys (`--apikey`) may not be needed if RPC is open or secured locally.

- **Running Locally**: Ensure `idena-go` is running on the same server and RPC is reachable (default port 9009). Use the node's RPC URL and API key (if configured) in the tool. As shown in the rolling indexer code, one can call `dna_epoch` (and others) with a simple HTTP POST and parse the JSON response [1] [2].

References

- Idena node RPC (Go API code): epoch RPC returns epoch and *discriminationStakeThreshold* [1] [2].
- Blockchain API structs: `Block` includes `Transactions []common.Hash` and `Flags []string` [5]; `Transaction` includes `From`, `Type`, etc. [7].
- Idena-auth rolling indexer (for example usage of `dna_epoch`) [1] [2].
- Idena indexer DB queries (for context): *ValidationSummary* and *BadAuthors* are indexer concepts [11] [8] (no direct RPC).
- Idena docs/structures: `dna_globalState` includes the threshold [9], and `dna_epoch` RPC details [12] [3].

---

[1] [2] [10] main.go
https://github.com/ubiubi18/IdenaAuthGo/blob/6d37cf2a0554935d4e61c031c7bab576727f1981/rolling_indexer/main.go

[3] [9] dna_api.go
https://github.com/idena-network/idena-go/blob/8992ca07f961267f9d9848318bd2b158999c9f40/api/dna_api.go

[4] [8] epoch.go
https://github.com/idena-network/idena-indexer-api/blob/a018cebba8fc42a6b2eda3a11b303d26ea7abb5c/app/db/postgres/epoch.go

[5] [7] blockchain_api.go
https://github.com/idena-network/idena-go/blob/8992ca07f961267f9d9848318bd2b158999c9f40/api/blockchain_api.go

[6] indexer.go
https://github.com/idena-network/idena-indexer/blob/cb504737f58201e640cef076e5e814cd23e32359/indexer/indexer.go

[11] epoch_identity.go
https://github.com/idena-network/idena-indexer-api/blob/a018cebba8fc42a6b2eda3a11b303d26ea7abb5c/app/db/postgres/epoch_identity.go

12 Idena node RPC | Idena documentation
https://docs.idena.io/docs/developer/node/node-rpc