Project: Decentralized Scraping Hub (Solana)

Part A — Initial User & Function Mapping

A.1 Manual user brainstorming (comprehensive list)

Direct users

- **Task Creator (Data Consumer):** posts scraping jobs, funds rewards, reviews results.
- **Scraper Node / Operator:** runs scrapers, uploads results to IPFS, submits proofs on-chain.
- **Task Verifier / Creator-as-validator:** reviews disputed results; finalizes tasks when needed.
- **Platform Admin / Developer:** deploys and maintains the system during POC.

Indirect beneficiaries

- **AI/ML engineers:** consumers of curated datasets.
- **Market researchers / e-commerce analysts:** frequent data consumers.
- **Independent developers / freelancers:** build data-driven features using scraped outputs.

Moderation / governance

- **Community moderators / DAO stewards:** dispute oversight, policy decisions (future).
- **Token holders / stakers:** economic interest and governance (future).

Stakeholders / infrastructure

- **IPFS / storage providers** (Web3.Storage, Pinata).
- **RPC / node providers** (public RPC / QuickNode etc.).
- **Enterprise early adopters** (potential paying customers).

A.2 AI-assisted user prioritization (POC focus)

Summary of approach: I used an AI assistant to help prioritize which users to focus on for the POC. I pasted the brainstorm list and asked which 2–5 types are most essential to prove the core value.

Final prioritized list (2–4 users) — with rationale

1. **Task Creator (Data Consumer)** — primary source of demand; POC must let creators post tasks and retrieve verified data.
2. **Scraper Node / Operator** — supply side; POC must show nodes can fetch tasks, run scrapers, upload to IPFS, and submit proofs.
3. **Task Verifier (Creator-as-validator + auto-majority)** — verification is core value; POC must implement majority verification and an explicit creator override.
4. **Platform Admin (Developer)** — required for deployment, monitoring, and manual interventions during the POC.

Why these four: They form the minimal closed loop: task creation → execution → verification → payout. Other roles (token holders, enterprise buyers, moderators) are important later but not required to demonstrate the POC.

A.3 Core function mapping (for prioritized users)

Task Creator

- Submit a task: provide target (URL or keyword), output schema/instructions, and reward amount.
- Fund the task (deposit tokens into escrow).
- View task status, submitted CIDs, and transaction history.
- Manually finalize a task when automatic verification is inconclusive.

Scraper Node / Operator

- Discover open tasks (public task pool).
- Accept/claim a task or pick from the pool.
- Execute scraping locally and normalize output to the specified format.
- Upload raw+parsed output to IPFS and obtain CID.
- Submit a signed on-chain `submit_result(task, cid)` transaction.
- Receive payout if their CID is accepted.

Task Verifier (Creator-as-validator & auto verify)

- Automatic verification: count matching CIDs and accept the majority if present.
- If no majority, creator inspects IPFS results and calls finalize with chosen CID.
- Trigger payout on-chain after finalization.

Platform Admin / Developer

- Deploy the Anchor program to Devnet.
- Monitor tasks and intervene for stuck states (manual finalize for testing).
- Provide the scraper sample/template for early nodes (optional for POC).

A.4 Deriving Core POC Requirements

Top 2 critical user interactions

1. **Happy path (end-to-end):** Creator posts a task → funds escrow → nodes scrape → upload to IPFS → submit CIDs → on-chain majority verifies → escrow releases reward(s).
2. **Dispute fallback:** Creator posts task → nodes submit different CIDs (no majority) → creator manually reviews IPFS outputs → creator finalizes → escrow releases reward(s).

Key technical requirements (for POC)

- **On-chain instructions:** `create_task`, `submit_result`, `finalize_task`.
- **Accounts:** `Config` (reward mint), `Task` (PDA), `ScrapeResult` (PDA per task+node), escrow token ATA (owned by Task PDA).
- **Escrow mechanism:** task-funded token ATA under program control, CPI transfers via `anchor_spl::token`.
- **Verification:** on-chain majority counting of CIDs; creator override when no majority.
- **Off-chain:** Node CLI (Playwright/Puppeteer or Scrapy) that normalizes output, uploads to IPFS (Web3.Storage), then calls `submit_result`.
- **Frontend:** Next.js app to create tasks, list tasks + CIDs, and allow creator finalize.
- **Anti-spam:** `min_reward` and optional `min_confirmations` to avoid trivial auto-finalize.
- **Auditability:** store raw + normalized data on IPFS; show transaction/CID history in UI.

Part B — Adversarial Analysis & Granularity Check

B.1 AI critique (summary) and adjustments

AI critique highlights

- Break composite stories into atomic actions.

- Specify exact on-chain data shapes (account fields, PDAs, signer roles).
- Add anti-spam and timestamp fields.
- Ensure uniqueness per `[task, node]` for submissions.

My adjustments

- Split “create task” into separate actions (metadata registration, escrow funding, open state).
- Added fields to `Task` and `ScrapeResult` (timestamps, `min_confirmations`, `expected_schema` or `extractor_hash`).
- Enforce `ScrapeResult` PDA uniqueness using seed `[b"result", task, node].`
- Add `min_confirmations` optional parameter to let tasks auto-finalize only with enough submissions.

Part C — Granularity & Clarity Refinement

C.1 Final atomic user stories (clean, non-technical wording)

C1 — Register Task Metadata

- *User signs and submits task metadata (URL, instruction, expected output format).*

C2 — Fund Escrow for Task

- *User deposits reward tokens into an on-chain escrow associated with the task.*

C3 — Mark Task Open

- *System marks task as open for nodes once escrow is confirmed.*

C4 — Node Discovers Task

- *Node lists available tasks and selects one to attempt.*

C5 — Node Runs Scraper Locally

- *Node executes the scraping job, normalizes output to the specified format, and prepares files.*

C6 — Node Uploads to IPFS

- *Node uploads raw + normalized outputs to IPFS and receives a CID.*

C7 — Node Submits CID On-Chain

- *Node creates a signed transaction to record the CID for the task (one submission per node).*

C8 — Automatic Verification

- *When enough results exist, anyone can call finalize: the contract checks if a CID has a strict majority (>50%) and marks it as winner.*

C9 — Creator Manual Finalize

- *If no majority, the creator reviews IPFS outputs and calls finalize with their chosen CID.*

C10 — Payout Distribution

- *Program transfers escrowed reward to the node(s) who submitted the winning CID, splitting evenly if multiple nodes submitted identical winning outputs.*

C.2 Refinement log (examples)

- **Before:** “User creates a task.”
  After: Split into C1 (register metadata), C2 (fund escrow), C3 (mark open).
  Rationale: Each maps to distinct on-chain actions (account init, token CPI, state change).

- **Before:** “Nodes submit results.”
  After: C5–C7 split into run scraper, upload to IPFS, submit CID on-chain.
  Rationale: Separates compute, storage, and on-chain proof steps so verification is reproducible.

- **Before:** “Verification and payout.”
  After: C8–C10 split auto verify, manual finalize, and payout.
  Rationale: Makes the decision and payment steps explicit and auditable.

Part D — Potential On-Chain Requirements (per story)

> *Notes: PDAs use seeds like `["task", creator_pubkey, task_counter]` and `["result", task_pubkey, node_pubkey]`. Token transfers use `anchor_spl::token` CPIs. Use `Clock` sysvar for timestamps.*

C1: Register Task Metadata

- Instruction: `create_task_metadata(url, instruction, expected_schema, min_confirmations)`
- On-chain needs:

- Create `Task` PDA account and store: `creator: Pubkey`, `created_at: i64`, `url: String`, `instruction: String`, `expected_schema: String`, `min_confirmations: u8`, `reward_amount: u64 (0 until funded)`, `finalized: bool`, `final_cid: Option<String>`, `bump: u8`.
- Emit `TaskCreated(task_pubkey)` event.

C2: Fund Escrow for Task

- Instruction: `fund_task(reward_amount)` (or combined with create in POC)
- On-chain needs:
  - Create escrow token account ATA with mint = `Config.reward_mint` and owner = `task_pda`.
  - Transfer tokens from user ATA to escrow ATA via `token::transfer` CPI.
  - Set `task.reward_amount = reward_amount` and `task.funded = true`.

C3: Mark Task Open

- Implicit after `fund_task` success. Require `task.funded == true` before nodes can submit.

C4: Node Discovers Task

- Off-chain behavior: client uses `program.account.task.all()` to list tasks. (No on-chain change required.)

C5: Node Runs Scraper Locally

- Off-chain behavior: node uses Playwright/Puppeteer or Scrapy, normalizes output per `expected_schema`. (No direct on-chain requirement.)

C6: Node Uploads to IPFS

- Off-chain behavior: upload raw+parsed artifacts to Web3.Storage; get CID. (Record extractor version/hash in metadata.)

C7: Node Submits CID On-Chain

- Instruction: `submit_result(task_pubkey, cid, extractor_hash)`
- On-chain needs:
  - Create `ScrapeResult` PDA with seed `[b"result", task_pubkey, node_pubkey]`. Store: `task_pubkey`, `node_pubkey`, `cid`, `submitted_at: i64`, `extractor_hash`.
  - Check `task.funded == true && task.finalized == false`.
  - Increment `task.total_submissions` (or rely on counting `remaining_accounts` at finalize).

- Prevent duplicate submissions by same node (PDA existence check).
- Emit `ResultSubmitted(task, node, cid)` event.

C8: Automatic Verification

- Instruction: `finalize_task_auto(task_pubkey, results_accounts...)` or a single `finalize_task` that inspects passed `ScrapeResult` accounts.
- On-chain needs:
  - Iterate `remaining_accounts` (the `ScrapeResult` accounts) and build `HashMap<CID, Vec<Pubkey>>`.
  - Let `total = total submissions`. If exists `cid` with `count > total/2` *or* `count >= task.min_confirmations` → accept that CID.
  - Move to payout step.

C9: Creator Manual Finalize

- Instruction: `finalize_task_manual(task_pubkey, chosen_cid)`
- On-chain needs:
  - Require `signer == task.creator`. If no majority, accept `chosen_cid` only if it exists among submitted CIDs.
  - Proceed to payout step.

C10: Payout Distribution

- On-chain needs:
  - Identify `winners = nodes_that_submitted(winning_cid)`. For each winner, verify they have an associated token account (`winner_ata`).
  - Transfer `reward_amount / winners.len()` from `escrow_ata` to each `winner_ata` via `token::transfer` (CPI), using PDA signer seeds to authorize.
  - Mark `task.finalized = true` and `task.final_cid = Some(winning_cid).`
  - Emit `TaskFinalized(task_pubkey, final_cid, winners).`

Cross-cutting on-chain requirements

- `Config` account storing `reward_mint`, `min_reward`, protocol fee.

- Error codes: `AlreadyFinalized`, `NotFunded`, `DuplicateSubmission`, `NoResults`, `NotCreator`, `TooFewConfirmations`.
- Events for indexing: `TaskCreated`, `ResultSubmitted`, `TaskFinalized`.
- Use `Clock` sysvar for timestamps.

Process Appendix — Prompts & AI Outputs

> *Required by the assignment: retain a record of prompts, AI outputs, and notes on how you used them.*

Prompt 1 (user prioritization):

> *My project's value proposition is a Decentralized Scraping Hub on Solana: users post scraping tasks and stake token rewards; distributed nodes perform scraping, upload results to IPFS, and the contract handles payouts after verification. Here is a brainstormed list of all potential user types: [list]. Based on the value proposition, which 2-5 of these user types are the most critical to focus on for an initial Proof-of-Concept? For each user you recommend, provide a brief rationale explaining why they are essential for proving the project's core value.*

AI Output 1 (summary used):

> *Recommended priority: Task Creators, Scraper Nodes, Task Verifiers (creator-as-validator), Platform Admin — because these four close the loop: create → execute → verify → payout.*

Prompt 2 (function mapping):

> *For a project with this value proposition [summary] and focusing on these prioritized users [Task Creator, Scraper Node, Task Verifier, Platform Admin], map out the key functions or interactions each user would need to perform.*

AI Output 2 (summary used):

> *Functions for each user were enumerated: create task & fund escrow; discover task & run scrapers & upload to IPFS; verify via majority or creator override; and admin/deploy/monitor.*

Prompt 3 (technical requirements for core interactions):

> *Based on the top interactions (end-to-end happy path and dispute fallback), what technical requirements are needed to build a POC?*

AI Output 3 (summary used):

> *Key technical requirements: on-chain instructions `create_task`, `submit_result`, `finalize_task`, PDAs for `Task` and `ScrapeResult`, escrow token ATA, IPFS off-chain storage, node CLI, and a frontend.*

Prompt 4 (adversarial critique):

> *Review my core user functions/stories and requirements. Are they granular enough and do they map to technical components? What's missing?*

AI Output 4 (summary used):

> *Critique: split stories into atomic steps, add timestamps and account schemas, ensure per-node uniqueness, add anti-spam and min_confirmations, and recommend storing extractor versions for reproducibility.*
