Intraverse FullStack Developer Task

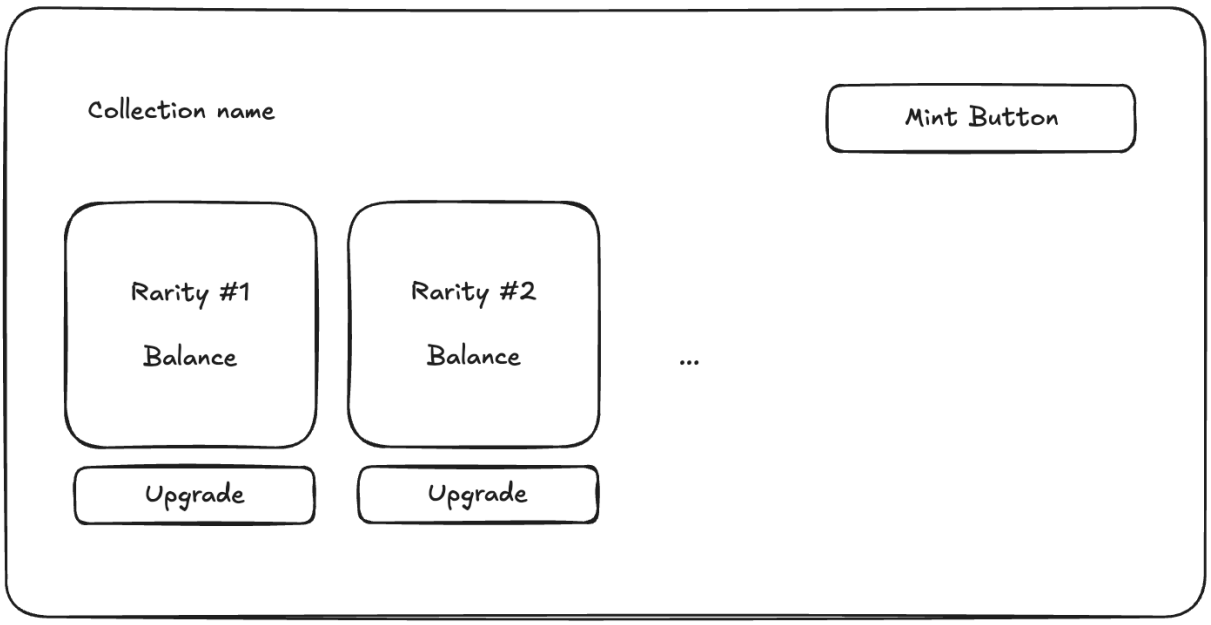
Build a small dapp with two routes:

- `/` — Mint & manage tokens
- `/analytics` — Simple on-chain analytics for one wallet vs. one contract

References

You might be able to use these addresses to have a first idea of what you want to build:

- https://play.intraverse.io/collections/ginch-loot



Smart Contract (given)

- **Contract address:** `0xC82E0CE02623972330164657e8C3e568d8f351FA` (IntraverseProtocolContract) - (Somnia Scan Link)
- **Functions you'll use**
  - `mint`
    - Params: `recipient (address)`, `signature (bytes)`
    - In the public phase: pass `0x0` for `signature`.
  - `upgradeTokenTo`
    - Upgrades to next rarity. Current config requires **2 tokens of the same rarity** per upgrade.
- **Network: Somnia**. (https://chainlist.org/chain/5031)
  Implement a **"Switch Network"** button that adds/switches the user's wallet to this chain. You may hardcode the chain configuration.

Requirements

1) Wallet & Network

- **Wallet connection (local)** using the browser wallet (e.g., MetaMask).
- **Switch Network** button that:
  - Adds Alethia (if missing) and switches to it.
  - Shows success/error states.

2) Mint & Manage (Home `/`)

- **Mint (public phase):**
  - Inputs: none (use connected account as `recipient`, `signature = 0x0`).
  - Button: "Mint".
  - Show transaction hash, pending/confirmed states, and errors.
- **Upgrade:**
  - Button: "Upgrade to next rarity" on each card (the collection has 12 cards)
  - Block the action unless the user holds at least **2 tokens of the same rarity**.
  - Show a clear explanation of the requirement.
- **Inventory widget:**
  - Display **the number of tokens the connected user owns** from this contract.

- **UX notes:**
    - Clear error messages (wrong network, no wallet, insufficient balance, user rejected, RPC error).
    - Loading/skeleton states for all async actions.

3) Analytics (`/analytics`)

To enable efficient transaction querying, it's standard practice to index on-chain transactions. Directly querying the blockchain can be cost-prohibitive or technically unfeasible.

- **Scope:**
    - **Wallet:** your own wallet or any other address (hardcode or env).
    - **Contract:** `0xC82E0CE02623972330164657e8C3e568d8f351FA`.
- **You must provide a minimal backend + DB** (your choice) that:
    - Indexes transactions were **from the wallet** AND **to contract** (or logs reference the contract).
      Stores at least: `hash, blockNumber, timestamp, from, to, method (if decoded), gasUsed, effectiveGasPrice`.

- **Queries to support (HTTP APIs + wired to the UI):**
    - `GET /api/analytics/summary`
        - Returns: total tx count, total gas used, total gas cost (native), first tx date, last tx date.
    - `GET /api/analytics/daily?from=YYYY-MM-DD&to=YYYY-MM-DD`
        - Returns: per-day tx count and gas used.
    - `GET /api/analytics/txs?limit=50&offset=0`
        - Returns: recent transactions table with decoded method names (`mint`, `upgradeTokenTo`).

- **Frontend (`/analytics`):**
    - KPI cards: total transactions, total gas used, total gas cost.
    - Simple line chart: daily tx count.
    - Table: recent transactions (hash → explorer link, method, date, gas used).

Instructions

1. **Technology Choices**

   Implement the solution using any technologies you prefer.

   Explain your technology choices in the `README.md`.

2. **Submission**

   Commit all code to a git repository.

   Push and provide a link to the repository or send us a .zip file of your git repo, make sure to archive it in a proper way so that we can see your commit history.

   You have at most **10 days** to complete the challenge after receiving the description.

Evaluation Criteria

- **Working Code**

  The solution must work without errors and should have an intuitive UI/UX.

  Provide an easy way to run the project (e.g., Docker and Docker Compose) or deploy it somewhere. (if you are using a db or service that you can't deploy it on free services there is absolutely no need to provide a deployed version, a description of how to run the project locally is enough.)

- **Project Design**

  Use appropriate design patterns and ensure component decoupling.

- **Code Readability**

  Follow best practices for code readability and maintainability.

- **Error Handling and Loading state**

  Implement robust error handling and loading system throughout the application. If you choose to use SSR technologies you should also handle loading states properly.

- **Smart usage of AI**

  You are encouraged to use AI tools, but we advise a strategic approach. Many projects are assigned to multiple candidates, and an over-reliance on AI without critical engagement will be apparent. While AI assistance is not inherently problematic, you must be prepared to articulate and defend your technical decisions, coding standards, logical reasoning, and how all components integrate effectively.

Additional Features (Bonus Points)

- **Authentication:** Implement a simple authentication integrated with web3 wallets.
- **Delivery Speed**: we value the speed of the development process in our company and if you deliver the task in 5 days we will consider it as a bonus point.

Any Questions?

Feel free to send an email on benyamin@intraverse.io or connect on discord and telegram with @belikeben id I'll reply as soon as possible.