

User Stories

Manual User Brainstorming

- **Direct Users:**
 - Screen time challengers, like 30 day detox enthusiasts etc
 - Digital minimalists, both crypto native or non-natives
- **Indirect Users:**
 - Affiliates, who make small percentage from both their referred user’s rewards and their slashed amounts
- **Moderators:**
 - Off chain verification component
 - Smart contract maintainer

AI-Assisted User Prioritization

- Screen Time Challengers (Direct Users)
- OffChain verification component
- Smart Contract Maintainers (Technical Moderators)

I agree with this prioritization list given by AI because:

- Screen time challengers are the primary participants in the system who are intrinsically motivated to reduce screen time and will directly experience the incentive-slash mechanism. Their behavior and feedback are crucial to proving whether the financial commitment + reward/slash mechanism actually motivates behavioral change.
- Without the off chain verification component it is impossible to validate user screen time usage proofs
- Smart contract maintainers make sure all the bug fixes, upgrades and edge cases are quickly addressed to maintain system stability and security

Core Function Mapping

Screen Time Challengers

- As a screen time challenger, I stake funds to join a challenge, so I can commit to reducing my phone usage.
- As a screen time challenger, I submit my daily screen time proof, so I can show whether I met my goal.
- As a screen time challenger, I receive a pass/fail result each day, so I know if I'm staying on track.
- As a screen time challenger, I earn rewards or get slashed based on my performance, so I stay financially accountable.
- As a screen time challenger, I track my progress during the challenge, so I stay motivated and aware of my performance.
- As a screen time challenger, I withdraw my remaining funds and rewards after the challenge ends, so I can reclaim what I've earned.

Off-Chain Verification Component

- As a verifier, I claim screen time submissions for review, so I can help validate user activity.
- As a verifier, I review and inspect user proofs, so I can determine if they met their daily goal.
- As a verifier, I submit a pass or fail judgment with optional notes, so my decision is recorded and transparent.
- As a verifier, I earn rewards for accurate and honest judgments, so I stay motivated to verify fairly.

Smart Contract Maintainers

- As a smart contract maintainer, I deploy and upgrade contracts, so the system runs securely and can adapt.
- As a smart contract maintainer, I manage protocol parameters, so the rules of the challenge remain balanced and effective.
- As a smart contract maintainer, I audit user and verifier activity, so I can detect abuse or irregularities.
- As a smart contract maintainer, I publish on-chain logs and events, so all actions are transparent and traceable.
- As a smart contract maintainer, I resolve disputes when needed, so the system stays fair and trusted.

Deriving Core PoC Requirement

Manual Task: 1-2 Core User Interactions

Screen Time Challengers

- As a screen time challenger, I stake funds to join a challenge, so I can commit to reducing my phone usage.
- As a screen time challenger, I submit my daily screen time proof, so I can show whether I met my goal.
- As a screen time challenger, I withdraw my remaining funds and rewards after the challenge ends, so I can reclaim what I've earned.

Off-Chain Verification Component

- As a verifier, I review and inspect user proofs, so I can determine if they met their daily goal.
- As a verifier, I submit a pass or fail judgment with optional notes, so my decision is recorded and transparent.

Requirements

Smart Contracts: Handle Staking, Challenge Tracking, and Rewards

Key Functions:

- `initialize_challenge(stake, duration, goal)` – User joins challenge with funds & target
- `submit_proof_uri(uri, day)` – User submits daily proof (e.g. IPFS link)
- `record_verdict(user, day, pass)` – Verifier submits pass/fail verdict
- `withdraw_rewards()` – User withdraws remaining stake & rewards

Challenge State (Per User):

- Stake balance
- Duration & goal
- Daily verdicts
- Rewards/slashes

Frontend (dApp)

User and verifier interface for interacting with contracts.

User Views:

- Join Challenge
- Upload Daily Proof
- Track Progress
- Withdraw Funds

Verifier Views:

- Review Queue
- Proof Viewer + Verdict Form
- Submission History

Off-Chain Backend

Simple server to route tasks and store files.

Components:

- Proof storage (IPFS, S3, etc.)
- Verifier task routing
- Verdict syncing to contract
- (Optional) Minimal DB for audit/trust tracking

Trust Assumptions

- Use whitelisted verifiers for PoC
- Only allow verdicts from trusted addresses
- Use testnet tokens (e.g. mock USDC)

Testnet Testing

- Deploy contracts to Solana devnet
- Run mock challenge cycles end-to-end
- Validate: proof submission, judgment flow, rewards, and withdrawal

Part B: Adversarial Analysis & Granularity Check

AI Critiques

- **Dispute Resolution:** You mention maintainers resolving disputes but no user story or technical flow. How do users appeal a verdict? How is that managed on-chain/off-chain?
- **Security & Fraud Prevention:**
 - How do you ensure proofs aren't forged or reused? Are there nonce/timestamp checks or hash commitments?
- **UX Edge Cases:**
 - What if a user misses submitting proof for a day?
 - Can proofs be updated or corrected?
 - How are partial rewards handled?
- **Scalability Considerations:**
 - For a PoC, manual verifier whitelisting is fine, but how will you scale verification? Could you add stories about delegation or crowdsourced verification?

Refined User Stories and Rationale Behind It

- As a screen time challenger, I can appeal if verification component submits incorrect proof.
- As a verifier, I get to resolve any user disputes

Additional Technical Requirements

Smart Contract

- `submit_appeal(user: Pubkey, day: u8, reason: String)`
- `resolve_appeal(maintainer: Pubkey, user: Pubkey, day: u8, decision: bool)`

Frontend

User:

- Raise appeal

Verifier:

- Resolve Appeal

Part C: Granularity & Clarity Refinement

Refer to `Part C: Refined Log of process` appendix for refinement rationale

Screen Time Challengers

- As a screen time challenger, I lock funds to join a challenge, so I can commit to reducing my phone usage.
- As a screen time challenger, I submit my daily screen time proof, so I can show whether I met my goal.
- As a screen time challenger, I can appeal if verification component submits incorrect proof.
- As a screen time challenger, I withdraw my remaining funds and rewards after the challenge ends, so I can reclaim what I've earned.

Off-Chain Verification Component

- As a verifier, I review and inspect user proofs, so I can determine if they met their daily goal.
- As a verifier, I submit a pass or fail judgment with optional notes, so my decision is recorded and transparent.
- As a verifier, I get to resolve any user disputes

Part D: Defining Potential On-Chain Requirements

Screen Time Challenger

User Story: "As a screen time challenger, I lock funds to join a challenge, so I can commit to reducing my phone usage"

Potential On-Chain Requirements:

- There must be different community challenges that users can choose to opt in, like 30min/day challenge or 40min/day challenge
- There must be a mechanism to get funds from the user as stake
- There must be a state that tracks the users stakes

User Story: "As a screen time challenger, I submit my daily screen time proof, so I can show whether I met my goal"

Requirements:

- Need a way to submit daily proof of screen usage
- Need to set users streak status according to the proof, like pass/fail
- If a user forgets to submit proof, the contract must automatically assume user failed the challenge for the day

User Story: I can appeal if verification was incorrect

- Need a way to let users raise an appeal (probably per day)
- Store appeal info on-chain like reason, status, etc.
- Pause reward/slash effects until appeal is handled
- Only trusted role, the verifier, can resolve it
- Should log appeal + decision for transparency

User Story: I withdraw my remaining funds and rewards after challenge ends

- Need withdraw function that becomes callable only after challenge ends
- Need to apply reward or slash logic
- Transfer remaining + earned tokens to the user

Verifier

User Story: I review and inspect user proofs

- System must authorize only verifier to judge user proofs
- User's daily streak must be updated accordingly

User Story: I submit a pass/fail judgment with notes

- Need function to record judgment (pass/fail + notes)
- Store verdict in user's challenge data
- Log every verdict with an event

User Story: I resolve user disputes

- Need to verify user appeals and recompute reward/slash accordingly
- Need to change the disputed status over user state to resolved after judgement

=====

Process Appendix

Process Appendix

AI Assisted User prioritization

Prompt

```
Prompt the AI with: "My project's value proposition is
[The core value proposition of the No Phone Addiction DAO is to create financial
accountability and gamified rewards for individuals aiming to reduce phone screen time. By
transforming digital well-being into a decentralized game of commitment and discipline, the
platform uniquely combines blockchain incentives with mental health goals.
].

Here is a brainstormed list of all potential user types:

[- direct users :
  - screen time challengers, like 30 day detox enthusiasts etc
  - digital minimalists, both crypto native or non-natives

- Indirect users:
  - Affiliates , who make small percentage from both their referred user's rewards and their slashed amounts

- Moderators:
  - Off chain verification component
  - Smart contract maintainer].

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 rati
```

Response

For the **initial Proof-of-Concept (PoC)** of **No Phone Addiction DAO**, the focus should be on a tightly scoped set of user types that are essential to validating the **core value**

1. Screen Time Challengers (Direct Users)

Rationale:

These users are the **primary participants** in the system. They are intrinsically motivated to reduce screen time and will directly experience the incentive-slash mechanism. Their

2. Moderators (Off-Chain Verifiers)

Rationale:

In the absence of a fully trustless screen-time oracle, early-stage verification must be **human-in-the-loop**. Moderators enable **proof validation**, forming the integrity backbone

3. Smart Contract Maintainers (Technical Moderators)

Rationale:

These users will maintain and update the protocol logic. For a PoC, having 1-2 maintainers ensures that bug fixes, upgrades, and edge cases in incentive design can be quickly addressed

4. Affiliates (Optional for PoC)

Rationale:

Although not essential for **core value validation**, including a few affiliates early on could test the **network growth model**. This helps evaluate whether the referral+commission

Recommended Focus for PoC:

Primary:

- * Screen Time Challengers
- * Off-Chain Moderators
- * Smart Contract Maintainers

Optional but valuable for PoC growth insights:

- * Affiliates (early evangelists)

This focused group balances **user behavior validation**, **protocol integrity**, and **technical maintainability**, giving the PoC the best chance to demonstrate that **financial**

Core Function Mapping

Prompt

For a project with this value proposition

[The core value proposition of the No Phone Addiction DAO is to create financial accountability and gamified rewards for individuals aiming to reduce phone screen time. By transforming digital well-being into a decentralized game of commitment and discipline, the platform uniquely combines blockchain incentives with mental health goals.

]

and focusing on these prioritized user types

[- Screen Time Challengers (Direct Users)

- OffChain verification componet

- Smart Contract Maintainers (Technical Moderators)]

, help map out the key functions or interactions each user would need to perform.

Response

Here's a clear mapping of the **key functions or interactions** required for each of the prioritized user types in the **No Phone Addiction DAO**, aligned with your value proposition

1. Screen Time Challengers (Direct Users)

These are the main participants in the protocol – users who stake funds and attempt to reduce screen time.

Core Interactions:

- * **Stake funds to enter a challenge**
 - Select challenge duration (e.g. 7, 14, 30 days), choose target daily screen time cap, and lock USDC as collateral.
- * **Submit daily screen time proof**
 - Upload screenshot from Screen Time app or synced report, ideally timestamped (manual or automated).
- * **Receive result (pass/fail)**

```
→ Based on validator judgment (initially off-chain), get confirmation of whether they met the goal that day.

* **Earn or lose rewards**
→ If compliant, receive gamified yield or NFT badge; if non-compliant, forfeit a portion of stake.

* **Track challenge progress**
→ View dashboard with stats: days completed, screen time logs, stake balance, rewards/slash status.

* **Withdraw remaining funds + rewards**
→ After challenge ends, withdraw retained stake and earned tokens, minus slashes.

---

## 📌 **2. Off-Chain Verification Component (Moderators/Validators)**

These actors verify the **authenticity and accuracy** of screen time proofs manually.

### 📌 **Core Interactions:**

* **Claim verification tasks**
→ Pick screen time submissions awaiting review, likely filtered by day, user, or randomness.

* **Review user submissions**
→ Check screenshot content, timestamp, image integrity, and whether goal was met.

* **Submit judgment (pass/fail + optional note)**
→ Mark submission as valid/invalid, optionally flag suspicious behavior or inconsistencies.

* **Earn rewards for accurate judging**
→ Get a fixed or performance-based reward for each correct + honest validation, possibly slashed for misbehavior (e.g. collusion).

* **Reputation or leaderboard system** *(optional for PoC)*
→ Moderators build trust score over time, useful for scaling and decentralization later.

---

## 📌 📌 **3. Smart Contract Maintainers (Technical Moderators)**

These are trusted devs managing the integrity and operations of the protocol smart contracts.

### 📌 **Core Interactions:**

* **Deploy and update smart contracts**
→ Initial deployment of staking, reward, slashing, and submission systems. Ability to patch bugs or upgrade logic.

* **Manage protocol parameters**
→ Adjust things like staking thresholds, reward ratios, challenge durations, slash rules during PoC phase.

* **Audit user and verifier activity**
→ Ensure fair execution and no contract abuse (e.g. suspicious slashing or reward farming).

* **Maintain transparency logs** *(via events or on-chain metadata)*
→ Emit logs for all state changes: challenge joins, proof submissions, verdicts, rewards, etc.

* **Settle disputes (if applicable)**
→ Temporarily act as appeals handler in case of malicious verification or edge cases.

---

## 📌 📌 Summary View:

| **User Type**          | **Core Functions**          |
| ----- | ----- |
| **Screen Time Challengers** | Stake funds, submit proofs, track progress, earn/lose rewards, withdraw funds |
| **Off-Chain Verifiers**    | Claim tasks, review submissions, submit verdicts, earn judging rewards |
| **Smart Contract Maintainers** | Deploy/upgrade contracts, adjust params, monitor execution, ensure system fairness |

---

## Deriving Core Poc requirements

**Prompt**

```

Based on these top 1-2 critical user interactions [- Screen time challengers - As a screen time challenger, I stake funds to join a challenge, so I can commit to reducing my phone usage. - As a screen time challenger, I submit my daily screen time proof, so I can show whether I met my goal. - As a screen time challenger, I withdraw my remaining funds and rewards after the challenge ends, so I can reclaim what I've earned.

- Off-Chain Verification Component
 - As a verifier, I review and inspect user proofs, so I can determine if they met their daily goal.
 - As a verifier, I submit a pass or fail judgment with optional notes, so my decision is recorded and transparent.]

, what are the key technical requirements needed to build a proof-of-concept?

```
**Response**

```

To build a **proof-of-concept (PoC)** that supports these top user interactions, you need to implement the **minimum viable technical components** that enable:

- Fund staking and rewards logic
- Proof submission and judgment
- Basic challenge state tracking

Here's a breakdown of the **key technical requirements**, grouped by functionality:

1. Smart Contracts (On-Chain Layer)

These handle the **staking logic**, **reward/slash accounting**, and **challenge lifecycle**.

Required Contracts & Functions:

- `initialize_challenge(stake_amount, duration, screen_time_goal)`
 - Stores challenge config and locks user stake.
- `submit_proof_uri(uri, day)`
 - Accepts a link (e.g. IPFS) or pointer to the user's screen time screenshot.
- `record_verdict(user, day, verdict: bool)`
 - Called by trusted verifier (off-chain or multisig) to record whether proof passed or failed.
- `withdraw_rewards()`
 - Lets user withdraw remaining stake and rewards after challenge ends.
- Challenge state (per user):
 - Stake balance
 - Start date / duration
 - Daily verdicts
 - Reward/slash amount

2. Frontend (Web App / dApp)

This enables **user interaction** with the protocol and verifier UI.

Key Screens:

- **Join Challenge** (form to stake funds, set goal & duration)
- **Daily Proof Upload** (upload screenshot or paste link)
- **Challenge Progress Dashboard** (shows days completed, verdicts, rewards)
- **Withdrawal Page** (trigger contract call to withdraw)

For Verifiers:

- **Review Queue** (list of pending user proofs)
- **Proof Viewer + Judgment Form** (view screenshot + mark pass/fail)
- **Submission history** (to verify fairness/trust)

3. Off-Chain Infrastructure

Since screen time proofs are manual, you'll need a basic off-chain backend to **handle uploads and route verification tasks**.

Off-Chain Components:

- **Proof Upload Storage** (e.g. IPFS, S3, or Firebase)
 - Users upload screenshots here
- **Verifier Assignment + UI Backend**
 - Routes unverified proofs to verifiers
 - Accepts verdicts and writes to smart contract
- **(Optional) Minimal database** to track:
 - Pending proofs
 - Verifier performance (for trust/reputation)
 - Audit trail / flagging suspicious behavior

4. Access Control & Trust Assumptions (PoC Simplified)

To reduce complexity for PoC:

- **Whitelist trusted verifiers** (manually approved by you)
- Let smart contracts **only accept verdicts** from these verifiers
- Use **testnet tokens** (e.g. USDC-mock) for staking

5. Testnet Deployment & Manual Testing

- Deploy smart contracts to Solana devnet/testnet
 - Use mock users + verifiers to run a few full challenge cycles
 - Observe:
 - Submission flow
 - Verdict accuracy
 - Correct reward/slash logic
 - Withdrawal flow at challenge end
-

Summary: PoC Feature Checklist

Feature	Required for PoC
User staking flow	Yes
Proof upload UI	Yes
Proof review + verdict UI	Yes
Smart contract: verdict + reward logic	Yes
Proof storage (IPFS or server)	Yes
Withdrawal function	Yes
Verifier whitelisting	Yes
Reputation or dispute logic	No (optional)
Oracle integration	No (future)

```
=====
Part B: Adversarial Analysis & Granularity Check
=====

**Prompt**
```

Review my core user functions/stories [- Screen time challengers - As a screen time challenger, I stake funds to join a challenge, so I can commit to reducing my phone usage. - As a screen time challenger, I submit my daily screen time proof, so I can show whether I met my goal. - As a screen time challenger, I receive a pass/fail result each day, so I know if I'm staying on track. - As a screen time challenger, I earn rewards or get slashed based on my performance, so I stay financially accountable. - As a screen time challenger, I track my progress during the challenge, so I stay motivated and aware of my performance. - As a screen time challenger, I withdraw my remaining funds and rewards after the challenge ends, so I can reclaim what I've earned.

- Off-Chain Verification Component
 - As a verifier, I claim screen time submissions for review, so I can help validate user activity.
 - As a verifier, I review and inspect user proofs, so I can determine if they met their daily goal.
 - As a verifier, I submit a pass or fail judgment with optional notes, so my decision is recorded and transparent.
 - As a verifier, I earn rewards for accurate and honest judgments, so I stay motivated to verify fairly.
- Smart Contract Maintainers
 - As a smart contract maintainer, I deploy and upgrade contracts, so the system runs securely and can adapt.
 - As a smart contract maintainer, I manage protocol parameters, so the rules of the challenge remain balanced and effective.
 - As a smart contract maintainer, I audit user and verifier activity, so I can detect abuse or irregularities.
 - As a smart contract maintainer, I publish on-chain logs and events, so all actions are transparent and traceable.
 - As a smart contract maintainer, I resolve disputes when needed, so the system stays fair and trusted.]

and requirements Requirements ::

Smart Contracts : Handle staking, challenge tracking, and rewards.

Key Functions:

- initialize_challenge(stake, duration, goal) – User joins challenge with funds & target
- submit_proof_uri(uri, day) – User submits daily proof (e.g. IPFS link)
- record_verdict(user, day, pass) – Verifier submits pass/fail verdict
- withdraw_rewards() – User withdraws remaining stake & rewards

Challenge State (per user):

- Stake balance
- Duration & goal
- Daily verdicts
- Rewards/slashes

2. Frontend (dApp)

User and verifier interface for interacting with contracts.

User Views:

- Join Challenge
- Upload Daily Proof
- Track Progress
- Withdraw Funds

Verifier Views:

- Review Queue
- Proof Viewer + Verdict Form
- Submission History

3. Off-Chain Backend

Simple server to route tasks and store files.

Components:

- Proof storage (IPFS, S3, etc.)
- Verifier task routing
- Verdict syncing to contract
- (Optional) Minimal DB for audit/trust tracking

4. Trust Assumptions

- Use whitelisted verifiers for PoC
- Only allow verdicts from trusted addresses
- Use testnet tokens (e.g. mock USDC)

5. Testnet Testing

- Deploy contracts to Solana devnet
- Run mock challenge cycles end-to-end
- Validate: proof submission, judgment flow, rewards, and withdrawal]

Considering my project's refined value proposition The core value proposition of the No Phone Addiction DAO is to create financial accountability and gamified rewards for individuals aiming to reduce phone screen time. By transforming digital well-being into a decentralized game of commitment and discipline, the platform uniquely combines blockchain incentives with mental health goals.]

, do these stories truly hit the mark? Are the requirements granular enough to map to specific technical components (e.g., database schemas, API endpoints, specific blockchain programs)? What's missing or unclear?"

****Response****

Your user stories and technical requirements align well with your project's core value proposition, but I'll give you a detailed review with suggestions for improvement and gaps to consider.

1. Alignment with Core Value Proposition

Your stories clearly focus on:

- **Financial accountability:** staking, rewards, slashing, withdrawal
- **Gamified rewards:** progress tracking, daily verdict feedback
- **Decentralized verification:** off-chain verifiers, transparent judgments
- **Mental health goals:** implicitly covered through behavior change workflows

Verdict: Your user stories hit the mark conceptually — they cover the core loop of commitment, proof submission, validation, and reward/slash outcome.

2. Granularity of Requirements vs Technical Components

You have defined:

- **Smart contract functions:** challenge initialization, proof submission, verdict recording, withdrawal
- **State elements:** stake balance, challenge parameters, daily verdicts, rewards/slashes
- **Frontend flows:** user join/upload/track/withdraw + verifier review/judgment
- **Off-chain backend:** storage, routing, verdict syncing, optional DB for audits
- **Trust assumptions & testnet plan**

These are a solid high-level foundation, but **to map precisely to technical components**, consider adding more detail on:

a. Data Model / Database Schema

- What is the exact schema for storing:
 - User challenges (ID, user address, stake amount, goal, start/end dates)
 - Daily submissions (proof URI, timestamp, status)
 - Verdicts (verifier ID, decision, notes, timestamp)
 - Verifier profiles (ID, reputation, whitelist status)
 - Rewards/slashes ledger

b. API Endpoints

- For frontend <-> backend communication (off-chain proof storage and verifier task routing), examples:
 - POST /challenge/join – user stakes & joins challenge
 - POST /proof/submit – upload proof metadata or URI
 - GET /proofs/pending – for verifier to fetch pending tasks
 - POST /verdict/submit – verifier submits judgment
 - GET /challenge/status – user fetches challenge progress and verdicts
 - POST /withdraw – user requests withdrawal transaction

c. Smart Contract Interfaces

- Define explicit instruction signatures with arguments, return values, and events:
 - InitializeChallenge {stake: u64, duration: u8, goal: u8}
 - SubmitProof {uri: String, day: u8}
 - RecordVerdict {user: Pubkey, day: u8, pass: bool}
 - WithdrawRewards {}
- Events for off-chain listening:

- ChallengeJoined(user, challengeId)
- ProofSubmitted(user, day)
- VerdictRecorded(user, day, pass)
- RewardsWithdrawn(user, amount)

3. What’s Missing or Could Be Clearer

- **Dispute Resolution:** You mention maintainers resolving disputes but no user story or technical flow. How do users appeal a verdict? How is that managed on-chain/off-chain?
- **Verifier Incentives & Reputation:**
 - User stories cover earning rewards for verifiers, but you might add explicit flows or logic to track verifier accuracy or penalize dishonest judgments.
 - This also affects how you whitelist verifiers and potentially automate trust in future iterations.
- **Security & Fraud Prevention:**
 - How do you ensure proofs aren't forged or reused? Are there nonce/timestamp checks or hash commitments?
- **UX Edge Cases:**
 - What if a user misses submitting proof for a day?
 - Can proofs be updated or corrected?
 - How are partial rewards handled?
- **Scalability Considerations:**
 - For a PoC, manual verifier whitelisting is fine, but how will you scale verification? Could you add stories about delegation or crowdsourced verification?

4. Summary

Aspect	Status	Recommendations
Core user stories	Well aligned	Maintain active voice, maybe add dispute story
Smart contract funcs	High-level solid	Define instruction signatures, event specs
Backend data & APIs	General outline	Draft detailed schema and endpoint definitions
Off-chain verification	Conceptualized	Add verifier reputation & incentive logic
Security & UX	Implicit but needs clarity	Address fraud, edge cases, proof validity
Scalability & growth	Mentioned in trust assumptions	Add future vision user stories

=====

Part C: Refinement Log

=====

Before Story :

- As a screen time challenger, I stake funds to join a challenge, so I can commit to reducing my phone usage.

After Story:

- As a screen time challenger, I lock funds to join a challenge, so I can commit to reducing my phone usage.

`Rationale:` removed "stake" with "lock" because not everyone knows "staking" means temporarily locking funds as commitment