Solana Smart Contract Assignment

Develop a **Solana smart contract** that allows users to **earn rewards (SOL)** by selecting an activity from a predefined list. The **reward amount is not fixed**; instead, it fluctuates dynamically based on **the total number of users and available tasks** using a **demand-supply logic**.

Core Requirements

1. Smart Contract Implementation (Rust + Anchor)

- Implement a **PDA (Program Derived Address)** to store each user's reward history.
- Ensure **only valid reward transactions** go through:
  - Users must select an **eligible activity** from a predefined list.
  - The **reward amount dynamically adjusts** based on **demand-supply mechanics**:
    - **Increase rewards when the number of tasks is higher than the number of users** (High Demand, Low Supply).
    - **Decrease rewards when the number of users is higher than the number of tasks** (Low Demand, High Supply).
- Store the following **reward transaction details** in a user's PDA:
  - **User (`Pubkey`)**
  - **Reward Amount (`u64` in lamports)**
  - **Activity Type (`String`)**
  - **Timestamp (`i64`)**
- **Tasks can be flagged as "available" or "unavailable"** based on an RNG mechanism:
  - Every **10 seconds**, a **random function (RNG)** determines **task availability**.

2. Activity Types & Rewards

Each activity type is associated with a **base reward** that fluctuates based on the **total number of users and available tasks**.

🔹 **Basic Tasks (Starting at 0.01 SOL Reward)**

- ✅ **"Check-in" (Daily Login)**
- ✅ **"View Analytics" (e.g., viewing an on-chain dataset)**
- ✅ **"Vote in a Poll"**
- ✅ **"Subscribe to a Smart Contract"**
- ✅ **"Leave Feedback on a dApp"**
- ✅ **"Complete a Profile Setup"**

🔹 **Engagement Tasks (Starting at 0.05 SOL Reward)**

- ✅ **"Cast a Vote" (Governance Proposal)**
- ✅ **"Send a Message" (On-chain comment)**

- ✅ **"Refer a User" (If they claim at least one reward)**
- ✅ **"Complete a Tutorial on Solana Usage"**
- ✅ **"Test a Beta Feature on a dApp"**
- ✅ **"Review a Smart Contract's Code"**

🔹 **High-Impact Tasks (Starting at 0.1 SOL Reward)**

- ✅ **"Deploy a Sample Smart Contract"**
- ✅ **"Stake SOL for at Least 7 Days"**
- ✅ **"Mint and Transfer an NFT"**
- ✅ **"Provide Liquidity to a Protocol"**
- ✅ **"Run a Validator Node for 24 Hours"**
- ✅ **"Contribute Code to an Open-Source Project"**

3. Demand-Supply Based Reward System

Reward Calculation

1. **Increase Rewards When Tasks > Users (High Demand, Low Supply)**
   - If **there are more tasks than available users**, the **reward increases up to 20%**.
2. **Decrease Rewards When Users > Tasks (Low Demand, High Supply)**
   - If **there are more users than available tasks**, the **reward decreases up to 10%**.
3. **RNG-Based Task Availability**
   - Every **10 seconds**, an **RNG function** determines which tasks remain **available** or become **unavailable**.

4. User Cooldown

After completing a task, the user **must wait 5 seconds** before selecting another task.

5. Anti-Farming Mechanism

To **prevent users from exploiting the reward system**, introduce **progressive farming protection**:

- If a user **chooses the same task 3 times in a row**, their **reward is reduced by 50%** on the **third repetition**.
- If the user **does not switch tasks after the third time**, the **reward continues reducing by another 50%** on each additional repetition.
- If a user **switches tasks**, the reward resets to normal.

📌 **Example**

User Task History	Reward Given
"Vote in a Poll" (1st time)	0.05 SOL
"Vote in a Poll" (2nd time)	0.05 SOL

"Vote in a Poll" (3rd time)	**0.025 SOL (-50%)**
"Vote in a Poll" (4th time)	**0.0125 SOL (-50% again)**
"Vote in a Poll" (5th time)	**0.00625 SOL (-50% again)**
"Refer a User" (Switching Task)	**0.05 SOL (Normal)**

6. Leaderboard (Optional Enhancement)

- Implement a **leaderboard system** that tracks the **top 5 most rewarded users**.
- Display:
  - **User's Public Key**
  - **Total Earned Rewards**
  - **Total Tasks Completed**
  - **Most Frequently Chosen Task**
- Update the leaderboard dynamically based on rewards earned.

7. Testing Scenarios

More Tasks Than Users (High Demand, Low Supply)

- Simulate **more available tasks than users**.
- Verify that the reward **increases up to 20%** due to high demand.

More Users Than Tasks (Low Demand, High Supply)

- Simulate **more users than available tasks**.
- Verify that the reward **decreases up to 10%** due to low supply.

Balanced Demand-Supply (Equal Users and Tasks)

- Ensure that **when users and tasks are equal**, rewards remain **unchanged**.

Repeat the Above Three for Both Farming and Non-Farming Scenarios

- Ensure the **progressive farming penalty applies** when a user **repeats the same task continuously**.
- Ensure **normal behavior when users switch tasks**.

Perform the Above 4 Tests with RNG-Based Task and User Generation

- Simulate **random task availability changes every 10 seconds**.
- Simulate **random user entries and exits** to validate reward adjustments dynamically.

🚀 Good luck!