Playkraft Intern Assignment, Wallet & Rewards

You're building a **cross-game wallet** that stores *Kraft Coins* (1 coin = 1 USD).
Games deposit periodic rewards (e.g., snake game awards every 12 hours). Users can top up via payment methods; all incoming USD converts to Kraft Coins.

Goal: build a **minimal in-memory backend** that correctly handles **concurrent modifications** to a user's wallet and **idempotent** requests. Keep the code minimal and runnable.

APIs (required)

1. `POST /wallet/topup`

   Body: `{ userId, amountUSD, idempotencyKey }`

   Converts `amountUSD` coins (1:1) and credits wallet.

2. `POST /game/reward`

   Body: `{ userId, amountCoins, rewardId, idempotencyKey }`

   — Credits reward coins to wallet.

3. `GET /wallet/:userId`

   — Returns current balance and recent operations (optional list OK; no full ledger required).

Must-have constraints

- **In-memory only**: do not use a database. Simple maps/objects are fine.
- **Atomic balance updates**: concurrent topup + reward must not cause lost updates.
- **Idempotency**: use `idempotencyKey` to prevent duplicate credits on retries.
- **Minimal code**: this is not an architecture essay, please produce runnable code (Node/Go/Python/Java - your choice) with clear run instructions.
- **Simple concurrency test**: provide a tiny client/test that fires a `topup` and a `reward` concurrently for the same user and asserts the final balance is correct.

What to explain (short)

In a short paragraph (3–6 sentences) include:

- The race condition you avoided and how your code prevents it.
- What your idempotency approach does and any edge cases you handled.

Deliverables

1. Source code (server + small test client).
2. Run instructions (how to start server and run test).
3. Single short explanation paragraph.
4. Deadline: **20th December 2025**.
5. Submission From: https://forms.gle/6jj93witMwXTSbTm6
6. Please don’t name your github repo with playkraft in it