Case Study: Casino Wallet Transaction Handler

Background

You are building part of a **casino aggregation platform**.

External casino providers send us **game events** for processing — two types:

- `bet` → when the player places a bet
- `result` → when the game round completes, and the outcome (win amount) is known

Your service must **process these events**, update player balances, and store transaction records.

Example Event Data

➤ Bet Event


```
{
  "amount": "10",
  "currency": "INR",
  "game_code": "ntn_aloha",
  "player_id": "67fff5c",
  "wallet_id": "684038d",
  "req_id": "42af0cb0-a016-4b55-a02f",
  "round_id": "3bb83e87-e540-4dd1-a3e8",
  "session_id": "37717449-fa2f-4f4f-9f03",
```

```
  "type": "bet"
}
```


➤ **Result Event**


```
{
  "amount": "8",
  "currency": "INR",
  "game_code": "ntn_aloha",
  "player_id": "67fff5c",
  "wallet_id": "684038d",
  "req_id": "c300ff2d-f7d1-43dc-b054",
  "round_id": "3bb83e87-e540-4dd1-a3e8",
  "session_id": "37717449-fa2f-4f4f-9f03",
  "type": "result"
}
```

Definitions

- **player_id** → unique player identifier
- **wallet_id** → wallet identifier (1:1 with player_id in this example)
- **req_id** → unique request identifier — every event (bet or result) has its own req_id

- **round_id** → **round identifier —** used to **match result to corresponding bet**

- **session_id** → session identifier **— tracks all bets and results while player is active in 1 game session**
  → A player may play **multiple rounds in 1 session** (session_id remains the same)
  → Session can have **N bets and N results** over time

Mapping Logic

- Every `"result"` event **must map to an earlier `"bet"`** → using `round_id`

- If a `"result"` arrives for a `round_id` not seen in any prior `"bet"` → the system must reject the result

- The system should store full transaction history (with `session_id` and `round_id`) → for audit & reconciliation

---

Casino Integration Flow

Typical flow from casino provider:

1 `GET /wallet/{player_id}` → to display player balance before allowing bet
2`POST /event` → `"bet"`
3 `POST /event` → `"result"`

Your TASK

Implement an **API endpoint** to receive these events (both `bet` and `result`)
Implement a simple **wallet system** to:

Deduct **bet amount** on `"bet"`
Add **win amount** on `"result"`
Store all transactions (both `"bet"` and `"result"`) for auditing
Wallet balance must remain **consistent with transaction history**

Implement:

`GET /wallet/{player_id}` → returns current balance (required for casino flow)
`GET /players` → returns a list of players and their balances (for testing)

POST /event Response Expectations

Your `POST /event` must return **clear status**:

On success (event accepted and processed):

http

```
HTTP 200 OK
{
  "status": "success"
}
```

On rejection (example reasons):

- *Reason of rejection* on `"bet"`
- `"result"` without matching prior `"bet"`

Example response:

```
HTTP 400 Bad Request
{
  "status": "rejected",
  "reason": "just i wished :)"
```

```
}
```

Or:

```
HTTP 400 Bad Request

{
  "status": "rejected",
  "reason": "I find another reason to reject…"
}
```

Expected Load

Casino providers typically send **50 - 500 requests per second** — this load can vary based on player activity.
It is common for **multiple events for the same player/round** to arrive very close together.

Additional Requirements

- The system should support **multiple players** — minimum **10 players** must be created and usable for testing.
- Provide a `GET /players` endpoint to list players and balances — this will be used for stress testing.

Submission Instructions

Please submit:

- GitHub link (preferred) or a zip file of your code
- A `README.md` with:
    - Tech stack used
    - How to run locally
    - Any assumptions or design notes
    - Notes about potential improvements