Cél

- Egy **következetes pénzkezelés**: minden coinmozgás naplózva, idempotens, visszakövethető.
- **Egyszerű kliens**, minden írás **Cloud Functions-on** keresztül, tranzakcióval.
- **Egyetlen igazságforrás**: `wallets/{uid}.balance` + `wallets/{uid}/ledger/*`.

Adatmodell

1) Wallet dokumentum

`wallets/{uid}`

- `balance`: **int** (érme darabszám; **soha ne double**)
- `pending`: int (opcionális; foglalásokhoz, ha használjuk a “reserve” modellt)
- `updatedAt`: timestamp
- `version`: int (optimista zároláshoz – opcionális)

2) Ledger bejegyzés

`wallets/{uid}/ledger/{entryId}`

- `amount`: **int** (pozitív = jóváírás, negatív = terhelés)
- `currency`: `"TippCoin"`
- `type`: `"bet_stake" | "bet_win" | "bet_refund" | "bonus_daily" | "admin_adjust" | ..."`
- `refType`: `"ticket" | "system" | "bonus" | ..."`
- `refId`: string (pl. ticketId)
- `status`: `"committed" | "reversed" | "pending"` (ha foglalást használsz)
- `idempotencyKey`: string (kötelező minden CF hívásnál)
- `meta`: map (bookmakerId, odds, marketKey, stb.)
- `createdAt`: timestamp

Invariáns: `balance` = Σ(ledger.amount where status=="committed")

Műveleti modell

0) Általános elvek

- **Minden írás CF-ben, tranzakcióval** (`runTransaction`).
- **Idempotencia**: minden művelet **idempotencyKey**-t kap (pl. `functionExecId` vagy `ticketId:win`), és tranzakcióban ellenőrizzük, létezik-e már.
- **Integer pénzegység** (coin): nincs lebegőpontos hiba.
- **Olvasás kliensen**: csak `wallet` doc és a `ledger` „saját” listázása.

1) Fogadás indítása (ticket létrehozás)

Döntés: két korrekt opció – válassz.

A. Azonnali levonás (egyszerűbb)

- CF `placeBet(uid, ticketId, stake)`:
  1. Tranzakcióban:
     - check: `balance >= stake`
     - write ledger: `amount = -stake`, `type="bet_stake"`, `refId=ticketId`, `idempotencyKey="ticket:{ticketId}:stake"`
     - update wallet.balance: `balance - stake`
  2. Ticket `status="pending"`
- Előny: nincs „pending” mező.
- Következmény: vesztes szelvénynél nincs további tranzakció; nyertesnél **csak a nyereményt** írjuk jóvá (stake vissza + profit egyben).

B. Foglalás (komplexebb)

- CF `placeBetReserve(uid, ticketId, stake)`:
  1. Tranzakció: `pending += stake`, ledger `status="pending"`
- CF `finalizeBet(...)`:
  1. win: `pending -= stake`; ledger stake → committed(-stake) + win(+payout)
  2. lose: `pending -= stake`; stake ledger committed(-stake)
  3. cancel: `pending -= stake`; refund ledger committed(+stake)
- Előny: visszamondásig nem „tűnik el” a coin.
- Hátrány: bonyolultabb, több edge case.

Javaslat: **A. Azonnali levonás** – kevesebb hiba, jobb UX (pontos egyenleg azonnal).

2) Szelvény lezárás (match_finalizer)

- Input: `ticketId`, `uid`, `outcome="win"|"lose"|"cancel"`, `payout` (int; win esetén stake+profit, vagy csak profit – döntsd el, lásd alább)
- Tranzakcióban:
  - **Idempotencia**: keresd a ledgerben `idempotencyKey="ticket:{ticketId}:{outcome}"`. Ha létezik → **no-op**.
  - `lose`: nincs új bejegyzés (ha stake már levonva), **vagy** írhatsz 0-s „settle” logot csak nyomkövetésre.
  - `win`: ledger `+payout`, update `balance += payout`
  - `cancel`: ledger `+stake`, update `balance += stake`
- **Döntés – payout számítás**
  - **Klasszik**: `win_credit = round(stake * odds)` és **stake már levonva**, így a `win_credit` tartalmazza a stake-et is.
  - **Alternatíva**: csak profitot jóváírni → `profit = round(stake * odds) - stake`.

Javaslat: **klasszik** (payout teljes összeg), így a ledgerben **két egyszerű lépés** látszik:

- `-stake` (tét)
- `+payout` (nyeremény teljes összege)

3) Napi bónusz

- CF `claimDailyBonus(uid)`:
  - Ellenőrizd ledgerben: van-e `type="bonus_daily"` utolsó 24 órában (**index kell**: `type, createdAt`).
  - Ha nincs: tranzakció → ledger `+bonus`, update `balance += bonus`.
  - Idempotency: `idempotencyKey="bonus:{yyyy-mm-dd}"`.

4) Admin korrekció

- CF `adminAdjust(uid, amount, reason)` csak **admin auth**-ra, audit loggal. Ledger bejegyzés + balance módosítás.

Biztonsági szabályok (Firestore Rules – vázlat)

```
rules_version = '2';
service cloud.firestore {
  match /databases/{db}/documents {

    function isAuth() { return request.auth != null; }
    function isOwner(uid) { return isAuth() && request.auth.uid == uid; }
```

```
    // Wallet
    match /wallets/{uid} {
      allow read: if isOwner(uid);
      allow write: if false; // soha ne közvetlenül kliensről
      match /ledger/{entryId} {
        allow read: if isOwner(uid);
        allow write: if false;
      }
    }

    // Tickets – olvasás sajátokra, írás CF-ből
    match /tickets/{ticketId} {
      allow read: if isAuth();
      allow write: if false; // ticketet is CF írja (vagy szigorú validációval)
    }

    // Notifications, relations stb…: hasonló minta (read self, write CF)
  }
}
```

Indexek (ajánlott)

- **Ledger időrend**:
  - `collectionGroup("ledger")` by `wallets/{uid}/ledger` → `orderBy(createdAt desc)`
- **Idempotencia**:
  - Egyedi index **nem kell**, de `where(idempotencyKey == X)` queryhez sima mezőindex kell (automatikus szokott lenni).
- **Bónusz check**:
  - `where(type == "bonus_daily").orderBy(createdAt desc).limit(1)`

Tranzakciós séma (pszeudó)

```
async function applyLedger(uid, entry) {
  const wRef = db.doc(`wallets/${uid}`);
  const eRef = wRef.collection('ledger').doc(); // or deterministic id

  await db.runTransaction(async tx => {
    const [wSnap] = await Promise.all([tx.get(wRef)]);

    // idempotencia
    const dup = await db.collectionGroup('ledger')
      .where('idempotencyKey', '==', entry.idempotencyKey)
      .limit(1).get();
    if (!dup.empty) return; // already applied

    const balance = (wSnap.exists ? wSnap.get('balance') : 0) as number;
    const newBalance = balance + entry.amount; // entry.amount lehet -stake vagy
+payout
```

```
    tx.set(eRef, {
      ...entry,
      createdAt: FieldValue.serverTimestamp(),
      status: 'committed',
      currency: 'TippCoin',
    });
    tx.set(wRef, {
      balance: newBalance,
      updatedAt: FieldValue.serverTimestamp(),
    }, { merge: true });
  });
}
```

Migráció a jelenlegi állapotról

Jelenleg: vegyes használat (`users/{uid}.coins`, néhol `users/{uid}.balance`).
Cél: `wallets/{uid}.balance` + `ledger`.

Lépések

1. **CF togglek & feature flag**: vezess be egy `USE_WALLET=true` env flaget.
2. **Wallet init script (egyszeri CF / admin job)**:
    - Végigmegy a `users`-en:
        - read `coins || balance || 0`
        - write `wallets/{uid}.balance = value`
        - write `wallets/{uid}/ledger/{init}`: `amount=value`, `type="admin_adjust"`, `refType="migration"`, `idempotencyKey="wallet:init"`
3. **Kódfrissítés**:
    - `daily_bonus`, `coin_trx`, `match_finalizer`, `placeBet` → **wallet API** hívások.
    - `users.{coins|balance}` olvasás **eltávolít** → helyette `wallets/{uid}.balance`.
4. **Read model a kliensen**:
    - Home/Profil: `wallets/{uid}` doc snapshot.
    - Történet: paginált `ledger` query `orderBy(createdAt, desc)`.
5. **Régi mezők kivezetése**:
    - Ha minden live, egy batch script törölheti `users/{uid}.coins|balance` mezőket (opcionális).

API (belső szolgáltatásréteg)

```
// WalletService (CF oldalon)
credit(uid, { amount, type, refType, refId, idempotencyKey, meta })
debit(uid,  { amount, type, refType, refId, idempotencyKey, meta })
// idempotens, tranzakciós applyLedger-re épít

// Place bet
placeBet(uid, ticketId, stake) // -> debit -stake, create ticket pending

// Finalize
settleWin(uid, ticketId, payout)  // -> credit +payout
settleCancel(uid, ticketId, stake) // -> credit +stake
settleLose(uid, ticketId) // -> (no-op vagy 0-ás log)

// Bonus
claimDaily(uid, bonusAmount) // -> credit +bonus (idempotencia napi kulccsal)
```

Hibakezelés & megfigyelhetőség

- **Szigorú hibakódok** CF válaszban: `INSUFFICIENT_FUNDS`, `IDEMPOTENT_REPLAY`, `LEDGER_WRITE_FAILED`.
- **Log**: minden sikeres/hibás tranzakció struktúrált log (userId, idempotencyKey, refId, amount, balance_before/after).
- **Alert**: anomáliák (pl. új `balance` ≠ Σ ledger) – háttér ellenőrző job napi futással.

Edge case-ek

- **Dupla futás** (retry): idempotencia kulcs miatt **no-op**.
- **Versenyhelyzet** (két fogadás egyszerre): tranzakció **sorba állít**, consistency megmarad.
- **Ticket törlés elhelyezés előtt**: ha már levontad a stake-et, **cancel** úton visszaírod.
- **Résznyeremény / több tippes szelvény**: `refId=ticketId:legIndex` – több ledger sorral is mehet, ugyanazzal a fő `ticketId`-val.

Mit lát a user?

- **Balance**: real-time (`wallets/{uid}`).
- **Történet**: végtelen scroll a `ledger`-ből (szűrők: típus, dátum).
- **Ticket állapot**: külön a tickets UI kezeli, de **pénzmozgás mindig csak a ledgeren át**.

Rövid döntési fa

- Gyors indulás, kevés rizikó? → **Azonnali levonás** modell.
- Precíz pénzügyi audit (foglalások)? → **Reserve/pending** modell (`pending` mező + kétlépcsős commit).