perplexity

Yes

Here's a concrete first pass of the SQL schema plus TypeScript-style API handlers for **buy**, **tip**, and **cashout** built around the economics you defined.[1]

SQL schema (Postgres-style)

```
-- USERS / STREAMERS

CREATE TABLE users (
  id              BIGSERIAL PRIMARY KEY,
  email           TEXT UNIQUE NOT NULL,
  username        TEXT UNIQUE NOT NULL,
  is_streamer     BOOLEAN NOT NULL DEFAULT FALSE,
  created_at      TIMESTAMPTZ NOT NULL DEFAULT NOW()
);

-- TOKEN WALLET

CREATE TABLE token_wallets (
  id              BIGSERIAL PRIMARY KEY,
  user_id         BIGINT NOT NULL REFERENCES users(id) UNIQUE,
  balance_tokens  NUMERIC(18, 4) NOT NULL DEFAULT 0,
  created_at      TIMESTAMPTZ NOT NULL DEFAULT NOW(),
  updated_at      TIMESTAMPTZ NOT NULL DEFAULT NOW()
);

-- TOKEN PURCHASE (BUY NUGGIEZ)

CREATE TABLE token_purchases (
  id                     BIGSERIAL PRIMARY KEY,
  user_id                BIGINT NOT NULL REFERENCES users(id),
  tokens_purchased       NUMERIC(18, 4) NOT NULL,
  unit_price_usd         NUMERIC(10, 4) NOT NULL, -- e.g. 2.30
  total_usd              NUMERIC(12, 4) NOT NULL,
  payment_processor_fee   NUMERIC(12, 4) DEFAULT 0,
  external_payment_id     TEXT,
  created_at             TIMESTAMPTZ NOT NULL DEFAULT NOW()
);

-- TIPS

CREATE TABLE tips (
  id               BIGSERIAL PRIMARY KEY,
  tipper_id         BIGINT NOT NULL REFERENCES users(id),
  streamer_id       BIGINT NOT NULL REFERENCES users(id),
  amount_tokens     NUMERIC(18, 4) NOT NULL,
```

```
  amount_usd_value  NUMERIC(12, 4) NOT NULL, -- amount_tokens * 2.00
  created_at        TIMESTAMPTZ NOT NULL DEFAULT NOW()
);

CREATE INDEX idx_tips_streamer_id ON tips (streamer_id);
CREATE INDEX idx_tips_tipper_id   ON tips (tipper_id);

-- STREAMER BALANCE (USD VALUE OF TIPS)

CREATE TABLE streamer_balances (
  id              BIGSERIAL PRIMARY KEY,
  streamer_id     BIGINT NOT NULL REFERENCES users(id) UNIQUE,
  available_usd   NUMERIC(12, 4) NOT NULL DEFAULT 0,
  created_at      TIMESTAMPTZ NOT NULL DEFAULT NOW(),
  updated_at      TIMESTAMPTZ NOT NULL DEFAULT NOW()
);

-- CASHOUT REQUESTS

CREATE TABLE cashout_requests (
  id                BIGSERIAL PRIMARY KEY,
  streamer_id       BIGINT NOT NULL REFERENCES users(id),
  gross_amount_usd  NUMERIC(12, 4) NOT NULL,
  fee_percent       NUMERIC(5, 2)  NOT NULL, -- 18.50
  fee_amount_usd    NUMERIC(12, 4) NOT NULL,
  net_payout_usd    NUMERIC(12, 4) NOT NULL,
  status            TEXT NOT NULL, -- 'pending','approved','paid','rejected'
  payout_method     TEXT,          -- 'ach','paypal',etc.
  payout_reference  TEXT,          -- transaction id from bank/processor
  created_at        TIMESTAMPTZ NOT NULL DEFAULT NOW(),
  processed_at      TIMESTAMPTZ
);

CREATE INDEX idx_cashout_requests_streamer_id ON cashout_requests (streamer_id);

-- OPTIONAL: GENERIC TRANSACTIONS TABLE FOR REPORTING

CREATE TYPE transaction_type AS ENUM (
  'token_purchase',
  'tip',
  'cashout_fee',
  'cashout_payout',
  'adjustment'
);

CREATE TABLE nuggiez_transactions (
  id                BIGSERIAL PRIMARY KEY,
  user_id           BIGINT REFERENCES users(id),
  streamer_id       BIGINT REFERENCES users(id),
  type              transaction_type NOT NULL,
  amount_tokens     NUMERIC(18, 4),
  amount_usd        NUMERIC(12, 4),
  related_id        BIGINT, -- can reference token_purchases.id, tips.id, cashout_reques
  metadata          JSONB,
  created_at        TIMESTAMPTZ NOT NULL DEFAULT NOW()
);
```

This gives you: per-user wallet, per-streamer USD balance, and explicit tables for purchases, tips, and cashouts that your admin UI can filter by type “Nuggiez”.[2]

Constants

```
const TOKEN_PURCHASE_PRICE_USD = 2.30; // user pays this
const TOKEN_VALUE_USD = 2.00;          // value to streamer balance
const CASHOUT_FEE_PERCENT = 18.5;      // streamer cashout fee
```

TypeScript API – Buy Nuggiez

`POST /api/nuggiez/buy`

Payload:

```
{ "quantity": 50 }
```

Handler (Express-style, using a generic `db` with transactions):[1]

```
import { Request, Response } from 'express';

export const buyNuggiez = async (req: Request, res: Response) => {
  const userId = req.user.id; // from auth middleware
  const { quantity } = req.body as { quantity: number };

  if (!Number.isFinite(quantity) || quantity <= 0) {
    return res.status(400).json({ error: 'Invalid quantity' });
  }

  const tokens = quantity;
  const totalUsd = tokens * TOKEN_PURCHASE_PRICE_USD;

  await db.tx(async (t) => {
    // 1. Charge card (pseudo - replace with real PSP call)
    const paymentResult = await chargeCard({
      userId,
      amountUsd: totalUsd,
    });

    if (!paymentResult.success) {
      throw new Error('Payment failed');
    }

    // 2. Insert purchase record
    const purchase = await t.one(
      `
      INSERT INTO token_purchases
        (user_id, tokens_purchased, unit_price_usd, total_usd,
         payment_processor_fee, external_payment_id)
      VALUES ($1, $2, $3, $4, $5, $6)
      RETURNING id
      `,
```

```
      [
        userId,
        tokens,
        TOKEN_PURCHASE_PRICE_USD,
        totalUsd,
        paymentResult.feeUsd ?? 0,
        paymentResult.externalId,
      ],
    );

    // 3. Upsert wallet
    await t.none(
      `
      INSERT INTO token_wallets (user_id, balance_tokens)
      VALUES ($1, $2)
      ON CONFLICT (user_id)
      DO UPDATE SET balance_tokens = token_wallets.balance_tokens + EXCLUDED.balance_toker
                    updated_at = NOW()
      `,
      [userId, tokens],
    );

    // 4. Optional transaction record
    await t.none(
      `
      INSERT INTO nuggiez_transactions
        (user_id, type, amount_tokens, amount_usd, related_id, metadata)
      VALUES ($1, 'token_purchase', $2, $3, $4, $5)
      `,
      [
        userId,
        tokens,
        totalUsd,
        purchase.id,
        { externalPaymentId: paymentResult.externalId },
      ],
    );
  });

  return res.status(201).json({
    success: true,
    tokensPurchased: tokens,
    totalUsd,
  });
};
```

TypeScript API – Tip Streamer

`POST /api/nuggiez/tip`

Payload:

```
{ "streamerId": 123, "tokens": 10 }
```

Handler:

```
export const tipStreamer = async (req: Request, res: Response) => {
  const tipperId = req.user.id;
  const { streamerId, tokens } = req.body as {
    streamerId: number;
    tokens: number;
  };

  if (!Number.isFinite(tokens) || tokens <= 0) {
    return res.status(400).json({ error: 'Invalid token amount' });
  }

  if (tipperId === streamerId) {
    return res.status(400).json({ error: 'Cannot tip yourself' });
  }

  await db.tx(async (t) => {
    // 1. Check wallet balance
    const wallet = await t.oneOrNone(
      `SELECT balance_tokens FROM token_wallets WHERE user_id = $1 FOR UPDATE`,
      [tipperId],
    );

    if (!wallet || Number(wallet.balance_tokens) < tokens) {
      throw new Error('Insufficient Nuggiez balance');
    }

    const usdValue = tokens * TOKEN_VALUE_USD;

    // 2. Deduct tokens from tipper wallet
    await t.none(
      `
      UPDATE token_wallets
      SET balance_tokens = balance_tokens - $1,
          updated_at = NOW()
      WHERE user_id = $2
      `,
      [tokens, tipperId],
    );

    // 3. Create tip record
    const tip = await t.one(
      `
      INSERT INTO tips
        (tipper_id, streamer_id, amount_tokens, amount_usd_value)
      VALUES ($1, $2, $3, $4)
      RETURNING id
      `,
      [tipperId, streamerId, tokens, usdValue],
    );

    // 4. Upsert streamer balance (in USD)
    await t.none(
      `
      INSERT INTO streamer_balances (streamer_id, available_usd)
```

```
      VALUES ($1, $2)
      ON CONFLICT (streamer_id)
      DO UPDATE SET available_usd = streamer_balances.available_usd + EXCLUDED.available_
                    updated_at = NOW()
      `,
      [streamerId, usdValue],
    );

    // 5. Optional transaction record
    await t.none(
      `
      INSERT INTO nuggiez_transactions
        (user_id, streamer_id, type, amount_tokens, amount_usd, related_id)
      VALUES ($1, $2, 'tip', $3, $4, $5)
      `,
      [tipperId, streamerId, tokens, usdValue, tip.id],
    );
  });

  return res.status(201).json({
    success: true,
    streamerId,
    tokens,
    usdValue,
  });
};
```

TypeScript API – Cashout for Streamer

`POST /api/nuggiez/cashout`

Assume full-balance cashout each time for now.

```
export const requestCashout = async (req: Request, res: Response) => {
  const streamerId = req.user.id;

  await db.tx(async (t) => {
    // 1. Lock current balance
    const balanceRow = await t.oneOrNone(
      `SELECT available_usd FROM streamer_balances WHERE streamer_id = $1 FOR UPDATE`,
      [streamerId],
    );

    const availableUsd = balanceRow ? Number(balanceRow.available_usd) : 0;

    if (availableUsd <= 0) {
      throw new Error('No available balance to cash out');
    }

    const gross = availableUsd;
    const fee = +(gross * (CASHOUT_FEE_PERCENT / 100)).toFixed(2);
    const net = +(gross - fee).toFixed(2);

    // 2. Zero out available balance (locking this amount)
    await t.none(
```

```
      `
      UPDATE streamer_balances
      SET available_usd = 0,
          updated_at = NOW()
      WHERE streamer_id = $1
      `,
      [streamerId],
    );

    // 3. Create cashout request
    const cashout = await t.one(
      `
      INSERT INTO cashout_requests
        (streamer_id, gross_amount_usd, fee_percent,
         fee_amount_usd, net_payout_usd, status)
      VALUES ($1, $2, $3, $4, $5, 'pending')
      RETURNING id
      `,
      [streamerId, gross, CASHOUT_FEE_PERCENT, fee, net],
    );

    // 4. Transaction records
    await t.none(
      `
      INSERT INTO nuggiez_transactions
        (streamer_id, type, amount_usd, related_id)
      VALUES ($1, 'cashout_fee', $2, $3)
      `,
      [streamerId, fee, cashout.id],
    );

    await t.none(
      `
      INSERT INTO nuggiez_transactions
        (streamer_id, type, amount_usd, related_id)
      VALUES ($1, 'cashout_payout', $2, $3)
      `,
      [streamerId, net, cashout.id],
    );

    // 5. Send notification (pseudo)
    await sendNotification(streamerId, {
      kind: 'cashout_requested',
      gross,
      feePercent: CASHOUT_FEE_PERCENT,
      fee,
      net,
    });
  });

  return res.status(201).json({
    success: true,
    message: 'Cashout requested. You will receive a payout after review.',
  });
};
```

Quick economics sanity check

- Buy 1 Nuggiez: user pays 2.30 USD.
- Tip 1 Nuggiez: streamer balance +2.00 USD.
- Cashout: streamer receives $2.00 \times (1 - 0.185) = 1.63$ USD.
- Platform net (before PSP / payout costs): $2.30 - 1.63 = 0.67$ USD per fully-cycled token.[3]

Do you want to support **partial** cashouts (streamer chooses amount) or should we keep it full-balance only and simpler for now?

⁂

1. http://nodesource.com/blog/scalable-api-with-node.js-and-typescript/
2. https://docs.pega.com/bundle/platform/page/platform/system-administration/database-schema-optimizing.html
3. https://branchapp.com/blog/from-cash-tips-to-digital-wallets-the-evolution-of-tipping
4. https://www.youtube.com/watch?v=G5z0V5Pod1U
5. https://patents.google.com/patent/US9436935B2/en
6. https://files.radiatorsoftware.com/radiator/ref/AuthBySQLDIGIPASS.html
7. https://github.com/orgs/community/discussions/162011
8. https://developer.squareup.com/docs/in-app-payments-sdk/how-it-works
9. https://docs.getdbt.com/blog/sql-comprehension-technologies
10. https://www.reddit.com/r/node/comments/1kv07ji/i_built_a_nodejs_typescript_api_starter_with_full/
11. https://schemasmith.com/documentation/mssql/enterprise/script-tokens.html
12. https://docs.confluent.io/platform/current/schema-registry/develop/using.html
13. https://www.tigerdata.com/learn/text-to-sql-a-developers-zero-to-hero-guide
14. https://github.com/membership/membership.db
15. https://www.mssqltips.com/sqlservertip/3510/using-token-and-tokencount-in-sql-server-integration-services/
16. https://stackoverflow.com/questions/38493568/sending-handling-get-requests-in-typescript-using-express-request-and-node-js