```
/**
 *
╔═══════════════════════════════════════════════════════════════════════════════════════════════════════╗
 * ║  AEQUITAVERSE — API PRINCIPALE                                                                        ║
 * ║  Rôle    : Pont entre l'univers virtuel et le monde réel                                              ║
 * ║  Stack   : Node.js + Express + libxmljs2 + pg (PostgreSQL)                                            ║
 * ║  Chaînes : PRINCIPALE | CASINO | PARTAGÉE                                                             ║
 * ║  Auth    : JWT signé ED25519 + clé publique blockchain                                                ║
 * ║  Version : 1.0.0                                                                                      ║
 *
╚═══════════════════════════════════════════════════════════════════════════════════════════════════════╝
 */

'use strict';

// ═══════════════════════════════════════════════════════════════════════════════════════════════════════
// SECTION 1 — DÉPENDANCES ET CONFIGURATION
// ═══════════════════════════════════════════════════════════════════════════════════════════════════════

const express     = require('express');
const { Pool }    = require('pg');
const libxml      = require('libxmljs2');
const crypto      = require('crypto');
const { promisify} = require('util');
const jwt         = require('jsonwebtoken');
const rateLimit   = require('express-rate-limit');
```

```
const helmet      = require('helmet');
const cors        = require('cors');
const Decimal     = require('decimal.js');

const app = express();
app.use(express.json());
app.use(helmet());
app.use(cors({ origin: ['https://aequitaverse.org', 'https://app.aequitaverse.org'] }));

// ── Configuration base de données ChainQL ──────────────────────
const db = new Pool({
  host:     process.env.CHAINQL_HOST     || 'localhost',
  port:     parseInt(process.env.CHAINQL_PORT || '5432'),
  database: process.env.CHAINQL_DB       || 'aequitaverse',
  user:     process.env.CHAINQL_USER     || 'chainql_api',
  password: process.env.CHAINQL_PASSWORD,
  max:      50,
  idleTimeoutMillis: 30000,
  connectionTimeoutMillis: 5000,
});

// ── Constantes protocole MET ──────────────────────────────────
const MET = {
  TAUX_TAXE_CREATION:    new Decimal('0.01'),   // 1%
  TAUX_TAXE_MEMBRANE:    new Decimal('0.01'),   // 1%
  TAUX_FRAIS_RESEAU:     new Decimal('0.00015'),// 0.015%
  E_REF_KJ_PAR_G:        new Decimal('9500'),   // énergie blé grade B
  PLAFOND_DETENTION_PCT: new Decimal('0.005'),  // 0.5% masse totale max
  BURN_INACTIVITE_ANS:   50,
```

```
  QUORUM_VALIDATION:    0.67,              // 67%
  QUORUM_ASSEMBLEE:     0.75,              // 75%
  QUORUM_RESERVE:       0.90,              // 90%
  // Facteurs pénalité conversion fiat
  PENALITE_FIAT: [
   { max: 10_000,        facteur: 2   },
   { max: 100_000,       facteur: 5   },
   { max: 1_000_000,     facteur: 10  },
   { max: 1_000_000_000,  facteur: 50  },
   { max: Infinity,      facteur: 200 },
  ],
  // Taux taxe Casino selon levier
  TAXE_CASINO: {
    1:   0.005,
    2:   0.010,
    5:   0.020,
    10:  0.040,
    20:  0.080,
    50:  0.200,
    100: 0.500,
  },
};

// ── Rate limiting par niveau d'accès ──────────────────────────
const limitPublic = rateLimit({
  windowMs: 60_000,
  max: 1000,
  message: { erreur: 'Limite atteinte — 1000 req/min (accès public)' },
});
```

```
const limitAuditeur = rateLimit({
  windowMs: 60_000,
  max: 10_000,
  message: { erreur: 'Limite atteinte — 10000 req/min (auditeur)' },
});
const limitTribunal = rateLimit({ windowMs: 60_000, max: 100_000 });



// ═══════════════════════════════════════════════════════════════════════
// SECTION 2 — UTILITAIRES CRYPTOGRAPHIQUES ET VALIDATION
// ═══════════════════════════════════════════════════════════════════════

/**
 * Calcule le hash SHA3-256 d'un contenu de bloc XML.
 * Utilisé pour garantir l'immuabilité de chaque bloc.
 */
function hashBloc(contenuXml) {
  return 'sha3:' + crypto
    .createHash('sha3-256')
    .update(contenuXml, 'utf8')
    .digest('hex');
}

/**
 * Vérifie la signature ED25519 d'un acteur.
 * Chaque acteur signe ses transactions avec sa clé privée.
```

```
 * La clé publique est enregistrée sur la blockchain.
 */
function verifierSignature(message, signature, clePublique) {
  try {
    const sigBuffer = Buffer.from(
      signature.replace('sig:', ''), 'hex'
    );
    const verify = crypto.createVerify('ED25519');
    verify.update(message);
    return verify.verify(
      { key: clePublique, format: 'der', type: 'spki' },
      sigBuffer
    );
  } catch {
    return false;
  }
}

/**
 * Génère un identifiant de bloc unique et séquentiel.
 * Format : MET-{CHAINE}-{HAUTEUR_9CHIFFRES}
 */
async function prochainBlocId(chaine) {
  const { rows } = await db.query(
    `SELECT MAX(hauteur) AS h FROM blocs WHERE chaine = $1`,
    [chaine]
  );
  const prochaine = (rows[0].h || 0) + 1;
  return `MET-${chaine}-${String(prochaine).padStart(9, '0')}`;
```

```
}

/**
 * Valide un document XML contre le schéma XSD Aequitaverse.
 * Tout bloc non conforme est rejeté avant insertion.
 */
function validerXSD(xmlStr, xsdStr) {
  const xsdDoc = libxml.parseXml(xsdStr);
  const xmlDoc = libxml.parseXml(xmlStr);
  const valide = xmlDoc.validate(xsdDoc);
  return { valide, erreurs: xmlDoc.validationErrors };
}

/**
 * Calcule le nombre de µGU pour une production donnée.
 * Formule : µGU = Q × P_ref × F_q × F_r × F_c × F_t × 0.99
 */
function calculerEmission(params) {
  const {
    quantite_g,    // en grammes
    P_ref,         // prix référence µGU/g (calibration mensuelle)
    F_qualite,     // facteur grade denrée
    F_region,      // facteur région + IDH
    F_carbone,     // facteur empreinte carbone
    F_temps,       // facteur durée cycle
  } = params;

  const Q      = new Decimal(quantite_g);
  const base   = Q.mul(P_ref);
```

```
  const ajuste  = base.mul(F_qualite).mul(F_region).mul(F_carbone).mul(F_temps);
  const bruts   = ajuste.toDecimalPlaces(6, Decimal.ROUND_DOWN);
  const taxe    = bruts.mul(MET.TAUX_TAXE_CREATION)
                       .toDecimalPlaces(6, Decimal.ROUND_DOWN);
  const nets    = bruts.minus(taxe);

  return {
    micro_GU_bruts: bruts.toString(),
    taxe_creation:  taxe.toString(),
    micro_GU_nets:  nets.toString(),
    formule: `${quantite_g}g × ${P_ref} × ${F_qualite} × ${F_region} × ${F_carbone} × ${F_temps} × 0.99`,
  };
}

/**
 * Calcule le taux de conversion fiat → µGU selon le montant.
 * Plus le montant est élevé, plus le facteur pénalité est élevé.
 */
async function calculerConversionFiat(montantUSD) {
  // Récupérer P_ref courant depuis ChainQL
  const { rows } = await db.query(`
    SELECT AVG(e.micro_GU_nets / NULLIF(p.quantite_valeur * 1000, 0)) AS P_ref
    FROM emissions e
    JOIN productions p ON p.id = e.production_id
    WHERE p.denree_type = 'blé_dur'
      AND p.timestamp > NOW() - INTERVAL '30 days'
  `);
  const P_ref = new Decimal(rows[0].P_ref || '0.0000000285');
```

```
  // Trouver le facteur pénalité selon le montant
  const tranche = MET.PENALITE_FIAT.find(t => montantUSD <= t.max);
  const facteur = tranche.facteur;

  // Taux = P_ref × 1000 (par gramme → par kg) / facteur_pénalité
  const taux = P_ref.mul(1000).div(facteur).toDecimalPlaces(8);
  const montantGU = taux.mul(montantUSD).toDecimalPlaces(6, Decimal.ROUND_DOWN);

  return {
    taux_GU_par_USD:   taux.toString(),
    facteur_penalite:  facteur,
    montant_USD:       montantUSD,
    montant_GU_brut:   montantGU.toString(),
    frais_reseau_GU:   montantGU.mul(MET.TAUX_FRAIS_RESEAU)
                         .toDecimalPlaces(6).toString(),
    montant_GU_net:    montantGU.mul(new Decimal(1).minus(MET.TAUX_FRAIS_RESEAU))
                         .toDecimalPlaces(6).toString(),
    P_ref_courant:     P_ref.toString(),
  };
}


//
═══════════════════════════════════════════════════════════════════════════
═══
// SECTION 3 — MIDDLEWARE D'AUTHENTIFICATION
//
═══════════════════════════════════════════════════════════════════════════
═══
```

```
/**
 * Vérifie le JWT et résout le niveau d'accès de l'appelant.
 * Niveaux : PUBLIC | PRODUCTEUR | AUDITEUR_1 | AUDITEUR_2 | AUDITEUR_3 | TRIBUNAL
 */
async function authMiddleware(req, res, next) {
  const auth = req.headers.authorization;
  if (!auth || !auth.startsWith('Bearer ')) {
    req.acteur = { niveau: 'PUBLIC' };
    return next();
  }
  try {
    const token = auth.split(' ')[1];
    const decoded = jwt.verify(token, process.env.JWT_SECRET);

    // Résoudre le statut depuis ChainQL
    const { rows } = await db.query(
      `SELECT niveau, statut, reputation_score
       FROM auditeurs WHERE id = $1
       UNION ALL
       SELECT 'PRODUCTEUR' AS niveau, statut, fiabilite_score
       FROM producteurs WHERE id = $1
       LIMIT 1`,
      [decoded.acteur_id]
    );

    if (!rows.length || rows[0].statut === 'BANNI') {
      return res.status(403).json({ erreur: 'Accès refusé — acteur banni' });
    }
```

```
    req.acteur = {
      id:    decoded.acteur_id,
      niveau: rows[0].niveau || 'PUBLIC',
      statut: rows[0].statut,
    };
    next();
  } catch (err) {
    req.acteur = { niveau: 'PUBLIC' };
    next();
  }
}

function reqNiveau(niveauMin) {
  const ordre = {
    PUBLIC: 0, PRODUCTEUR: 1,
    OBSERVATEUR: 2, ANALYSTE: 3, CERTIFIÉ: 4, TRIBUNAL: 5,
  };
  return (req, res, next) => {
    if ((ordre[req.acteur?.niveau] || 0) < ordre[niveauMin]) {
      return res.status(403).json({
        erreur: `Niveau requis : ${niveauMin}. Votre niveau : ${req.acteur?.niveau || 'PUBLIC'}`,
      });
    }
    next();
  };
}

app.use(authMiddleware);
```

```
//
═══════════════════════════════════════════════════════════════════════════
═══
// SECTION 4 — ROUTES PUBLIQUES
// Accessibles sans authentification — lecture seule
//
═══════════════════════════════════════════════════════════════════════════
═══

// ── GET /api/v1/dashboard ──────────────────────────────────────
// Tableau de bord mondial temps réel
app.get('/api/v1/dashboard', limitPublic, async (req, res) => {
 try {
  const { rows } = await db.query(`
   SELECT
    (SELECT SUM(micro_GU_nets) FROM emissions)            AS masse_monetaire_GU,
    (SELECT COUNT(*) FROM productions
     WHERE statut = 'APPROUVÉ'
      AND timestamp > NOW() - INTERVAL '30 days')         AS productions_actives,
    (SELECT COUNT(*) FROM producteurs WHERE statut = 'ACTIF') AS producteurs_actifs,
    (SELECT COUNT(*) FROM transactions
     WHERE timestamp > NOW() - INTERVAL '24h')           AS transactions_24h,
    (SELECT COUNT(*) FROM contrats_option
     WHERE statut = 'ACTIF')                             AS options_actives,
    (SELECT MAX(solde_apres_GU) FROM fonds_commun_journal)  AS fonds_commun_GU,
    (SELECT COUNT(*) FROM auditeurs WHERE statut = 'ACTIF')  AS auditeurs_actifs,
    NOW()                                               AS timestamp
  `);
  res.json({ succes: true, donnees: rows[0] });
```

```
  } catch (err) {
    res.status(500).json({ erreur: err.message });
  }
});



// ── GET /api/v1/blocs/:id ──────────────────────────────────────
// Lecture d'un bloc par son identifiant
app.get('/api/v1/blocs/:id', limitPublic, async (req, res) => {
  try {
    const { rows } = await db.query(
      `SELECT b.*,
              p.producteur_id, p.denree_type, p.quantite_valeur,
              e.micro_GU_nets, e.taxe_creation_GU
       FROM blocs b
       LEFT JOIN productions p ON p.bloc_id = b.id
       LEFT JOIN emissions   e ON e.bloc_id  = b.id
       WHERE b.id = $1`,
      [req.params.id]
    );
    if (!rows.length) {
      return res.status(404).json({ erreur: 'Bloc non trouvé' });
    }
    res.json({ succes: true, bloc: rows[0] });
  } catch (err) {
    res.status(500).json({ erreur: err.message });
  }
});
```

```
// ── GET /api/v1/productions ──────────────────────────────────────
// Liste des productions avec filtres
app.get('/api/v1/productions', limitPublic, async (req, res) => {
 try {
  const {
   denree, pays, region, grade,
   date_debut, date_fin,
   page = 1, limit = 50,
  } = req.query;

  const where   = ['p.statut = $1'];
  const params  = ['APPROUVÉ'];
  let   idx     = 2;

  if (denree)    { where.push(`p.denree_type = $${idx++}`);  params.push(denree);    }
  if (pays)      { where.push(`p.pays = $${idx++}`);        params.push(pays);      }
  if (region)    { where.push(`p.region = $${idx++}`);      params.push(region);    }
  if (grade)     { where.push(`p.denree_grade = $${idx++}`); params.push(grade);     }
  if (date_debut){ where.push(`p.timestamp >= $${idx++}`);   params.push(date_debut); }
  if (date_fin)  { where.push(`p.timestamp <= $${idx++}`);   params.push(date_fin);  }

  const offset = (parseInt(page) - 1) * parseInt(limit);
  params.push(parseInt(limit), offset);

  const { rows } = await db.query(`
   SELECT
    p.id, p.producteur_id, pr.pseudonyme,
    p.denree_type, p.denree_grade,
```

```
      p.quantite_valeur, p.quantite_unite,
      p.pays, p.region, p.territoire,
      p.oracle_sat_ndvi,
      e.micro_GU_nets, e.taxe_creation_GU,
      e.F_qualite, e.F_region, e.F_carbone, e.F_temps,
      p.timestamp
    FROM productions p
    JOIN producteurs pr ON pr.id = p.producteur_id
    JOIN emissions   e  ON e.production_id = p.id
    WHERE ${where.join(' AND ')}
    ORDER BY p.timestamp DESC
    LIMIT $${idx++} OFFSET $${idx}
  `, params);

  res.json({ succes: true, productions: rows, page, limit });
 } catch (err) {
  res.status(500).json({ erreur: err.message });
 }
});


// ── GET /api/v1/tracabilite/:production_id ──────────────────────
// Traçabilité complète d'une production du champ au consommateur
app.get('/api/v1/tracabilite/:production_id', limitPublic, async (req, res) => {
 try {
  const { rows } = await db.query(`
    WITH RECURSIVE parcours AS (
      SELECT
        p.id       AS event_id,
```

```
    'PRODUCTION' AS type_event,
    p.producteur_id AS acteur_id,
    pr.pseudonyme   AS acteur_nom,
    e.micro_GU_nets AS valeur_GU,
    p.quantite_valeur, p.quantite_unite,
    p.timestamp, p.pays, p.denree_type,
    0 AS profondeur, p.id AS prod_origine
  FROM productions p
  JOIN emissions   e  ON e.production_id = p.id
  JOIN producteurs pr ON pr.id = p.producteur_id
  WHERE p.id = $1

  UNION ALL

  SELECT
    t.id, t.type, t.destinataire_id, t.destinataire_id,
    t.montant_GU, NULL, NULL,
    t.timestamp, NULL, pa.denree_type,
    pa.profondeur + 1, pa.prod_origine
  FROM transactions t
  JOIN parcours pa ON t.expediteur_id = pa.acteur_id
                  AND t.timestamp > pa.timestamp
                  AND t.sous_jacent_prod_ref = pa.prod_origine
  WHERE pa.profondeur < 15
)
SELECT * FROM parcours ORDER BY timestamp
`, [req.params.production_id]);

if (!rows.length) {
```

```
    return res.status(404).json({ erreur: 'Production non trouvée' });
   }
   res.json({ succes: true, parcours: rows });
  } catch (err) {
   res.status(500).json({ erreur: err.message });
  }
});


// ── GET /api/v1/taux/conversion-fiat ──────────────────────────
// Taux de conversion fiat → µGU (recalibré chaque mois)
app.get('/api/v1/taux/conversion-fiat', limitPublic, async (req, res) => {
  try {
   const montant = parseFloat(req.query.montant_usd || '1000');
   const resultat = await calculerConversionFiat(montant);
   res.json({ succes: true, conversion: resultat });
  } catch (err) {
   res.status(500).json({ erreur: err.message });
  }
});


// ── GET /api/v1/fonds-commun/solde ────────────────────────────
// Solde et composition du Fonds Commun en temps réel
app.get('/api/v1/fonds-commun/solde', limitPublic, async (req, res) => {
  try {
   const { rows } = await db.query(`
    SELECT
     exercice,
```

```
      SUM(montant_GU) FILTER (WHERE direction = 'ENTREE') AS total_entrees_GU,
      SUM(montant_GU) FILTER (WHERE direction = 'SORTIE') AS total_sorties_GU,
      MAX(solde_apres_GU)                                 AS solde_courant_GU,
      SUM(montant_GU) FILTER (
        WHERE type_mouvement = 'ENTREE_TAXE_CREATION')    AS taxe_creation_GU,
      SUM(montant_GU) FILTER (
        WHERE type_mouvement = 'ENTREE_TAXE_CASINO')      AS taxe_casino_GU,
      SUM(montant_GU) FILTER (
        WHERE type_mouvement = 'ENTREE_TAXE_BANCAIRE')    AS taxe_bancaire_GU,
      COUNT(*) FILTER (WHERE direction = 'ENTREE')        AS nb_entrees,
      COUNT(*) FILTER (WHERE direction = 'SORTIE')        AS nb_sorties
    FROM fonds_commun_journal
    WHERE exercice = EXTRACT(YEAR FROM NOW())::INTEGER
    GROUP BY exercice
  `);
  res.json({ succes: true, fonds_commun: rows[0] });
 } catch (err) {
  res.status(500).json({ erreur: err.message });
 }
});



//
═══════════════════════════════════════════════════════════════════════════════
═══
// SECTION 5 — ROUTES PRODUCTEURS (auth PRODUCTEUR)
// Soumission de production, transfert de µGU, conversion fiat
//
═══════════════════════════════════════════════════════════════════════════════
═══
```

```
// ── POST /api/v1/productions/soumettre ──────────────────────────
// Soumettre une nouvelle production pour validation
app.post('/api/v1/productions/soumettre',
 limitAuditeur,
 reqNiveau('PRODUCTEUR'),
 async (req, res) => {
  const client = await db.connect();
  try {
   await client.query('BEGIN');

   const {
    denree_type, denree_grade, quantite_kg,
    pays, region, lat, lng, silo_id,
    oracle_satellite, oracle_meteo, oracle_marche,
    temoins, cycle, territoire,
   } = req.body;

   const producteur_id = req.acteur.id;

   // 1. Vérifier que le producteur est actif et certifié
   const { rows: prod } = await client.query(
    `SELECT * FROM producteurs WHERE id = $1 AND statut = 'ACTIF'`,
    [producteur_id]
   );
   if (!prod.length) throw new Error('Producteur inactif ou non certifié');

   // 2. Vérifier la convergence des oracles (max ±5%)
   const rendement_sat = oracle_satellite?.rendement_declare_kg_ha;
```

```
const rendement_meteo_min = oracle_meteo?.rendement_theorique_min;
const rendement_meteo_max = oracle_meteo?.rendement_theorique_max;
const superficie = oracle_satellite?.superficie_ha;
const quantite_calculee = superficie * rendement_sat;
const divergence = Math.abs(quantite_calculee - quantite_kg) / quantite_kg * 100;

if (divergence > 5) {
  throw new Error(
    `Divergence oracles ${divergence.toFixed(2)}% > 5%. Production bloquée.`
  );
}

// 3. Récupérer P_ref et facteurs depuis la calibration mensuelle
const { rows: calibration } = await client.query(`
  SELECT
    AVG(e.micro_GU_nets / NULLIF(p.quantite_valeur * 1000, 0)) AS P_ref,
    (SELECT valeur FROM parametres_protocole WHERE nom = 'F_qualite_' || $1) AS F_q,
    (SELECT valeur FROM parametres_protocole WHERE nom = 'F_region_' || $2)  AS F_r
  FROM emissions e
  JOIN productions p ON p.id = e.production_id
  WHERE p.denree_type IN ('blé_dur', 'blé_tendre')
    AND p.timestamp > NOW() - INTERVAL '30 days'
`, [denree_grade, pays]);

const F_q = parseFloat(calibration[0].F_q || '1.00');
const F_r = parseFloat(calibration[0].F_r || '1.00');
const F_c = calculerFacteurCarbone(oracle_meteo?.empreinte_carbone_pct);
const F_t = calculerFacteurTemps(cycle);
const P_ref = new Decimal(calibration[0].P_ref || '0.0000000285');
```

```
    // 4. Calculer l'émission
    const emission = calculerEmission({
      quantite_g: quantite_kg * 1000,
      P_ref: P_ref.toString(),
      F_qualite: F_q,
      F_region:  F_r,
      F_carbone: F_c,
      F_temps:   F_t,
    });

    // 5. Vérifier que la taxe peut être payée (solde suffisant)
    if (new Decimal(prod[0].solde_disponible_gu).lt(emission.taxe_creation)) {
      throw new Error(
        `Solde insuffisant pour la taxe de création. ` +
        `Requis : ${emission.taxe_creation} µGU. ` +
        `Disponible : ${prod[0].solde_disponible_gu} µGU.`
      );
    }

    // 6. Générer le bloc XML et son identifiant
    const bloc_id   = await prochainBlocId('MAIN');
    const prod_id   = `PROD-${new Date().getFullYear()}-${pays.slice(0,2).toUpperCase()}-
${String(Date.now()).slice(-6)}`;
    const emis_id   = `EMIS-${bloc_id}`;
    const taxe_tx   = `TX-TAXE_CREATION-${new Date().getFullYear()}-${String(Date.now()).slice(-8)}`;
    const emis_tx   = `TX-EMISSION-${new Date().getFullYear()}-${String(Date.now()).slice(-8)}`;

    // 7. Construire le bloc XML
```

```
const blocXml = genererBlocProductionXML({
  bloc_id, prod_id, emis_id, taxe_tx, emis_tx,
  producteur: prod[0],
  denree_type, denree_grade, quantite_kg,
  pays, region, lat, lng, silo_id,
  oracle_satellite, oracle_meteo, oracle_marche,
  temoins, cycle, territoire,
  emission, F_q, F_r, F_c, F_t,
  divergence_max_pct: divergence,
});

// 8. Valider le XML contre le schéma XSD
const xsd = require('fs').readFileSync('./1_schema_xsd.xsd', 'utf8');
const { valide, erreurs } = validerXSD(blocXml, xsd);
if (!valide) {
  throw new Error(` XML invalide : ${erreurs.map(e => e.message).join(', ')}` );
}

// 9. Calculer le hash du bloc
const hash_precedent = await dernierHashChaine('MAIN');
const hash           = hashBloc(blocXml);

// 10. Insérer dans les tables ChainQL (atomiquement)
await client.query(`
  INSERT INTO blocs
    (id, hash, hash_precedent, hauteur, timestamp, type, chaine,
     territoire, noeud_emetteur_id, masse_monetaire_apres)
  VALUES ($1,$2,$3,
    (SELECT COALESCE(MAX(hauteur),0)+1 FROM blocs WHERE chaine='PRINCIPALE'),
```

```
    NOW(), 'PRODUCTION', 'PRINCIPALE',
    $4, $5, $6)
  `, [bloc_id, hash, hash_precedent, territoire,
    `NOEUD-${pays}-AUTO`, emission.micro_GU_nets]);

await client.query(`
  INSERT INTO productions
    (id, bloc_id, producteur_id, statut, cycle, territoire,
     denree_type, denree_grade, quantite_valeur, quantite_unite,
     pays, region, lat, lng, silo_id,
     oracle_satellite_ref, oracle_sat_ndvi, oracle_sat_superficie_ha,
     oracle_meteo_ref, oracle_meteo_precipmm, oracle_meteo_temp_moy,
     oracle_marche_prix_ref, divergence_max_pct, timestamp)
  VALUES ($1,$2,$3,'EN_VALIDATION',$4,$5,
          $6,$7,$8,'kg',$9,$10,$11,$12,$13,
          $14,$15,$16,$17,$18,$19,$20,$21,NOW())
  `, [prod_id, bloc_id, producteur_id, cycle, territoire,
    denree_type, denree_grade, quantite_kg,
    pays, region, lat, lng, silo_id,
    oracle_satellite?.reference, oracle_satellite?.ndvi,
    oracle_satellite?.superficie_ha,
    oracle_meteo?.reference,
    oracle_meteo?.precipitation_mm, oracle_meteo?.temperature_moy,
    oracle_marche?.prix_ref,
    divergence]);

await client.query(`
  INSERT INTO emissions
    (id, bloc_id, production_id, producteur_id,
```

```
    formule_version, Q_grammes, P_ref,
    F_qualite, F_region, F_carbone, F_temps,
    micro_GU_bruts, taxe_creation_GU, micro_GU_nets, taxe_pct,
    taxe_statut, taxe_tx_id, taxe_timestamp, timestamp)
  VALUES ($1,$2,$3,$4,'1.0.0',
        $5,$6,$7,$8,$9,$10,$11,$12,$13,1.00,
        'EN_ATTENTE',$14,NULL,NOW())
` , [emis_id, bloc_id, prod_id, producteur_id,
   quantite_kg * 1000, P_ref.toString(),
   F_q, F_r, F_c, F_t,
   emission.micro_GU_bruts, emission.taxe_creation,
   emission.micro_GU_nets, taxe_tx]);

// 11. Débiter la taxe du solde producteur (réservation)
await client.query(`
  UPDATE producteurs
  SET solde_disponible_GU = solde_disponible_GU - $1,
      solde_bloque_GU     = solde_bloque_GU + $1
  WHERE id = $2
` , [emission.taxe_creation, producteur_id]);

await client.query('COMMIT');

// 12. Diffuser aux nœuds validateurs (asynchrone)
diffuserAuxNoeuds(bloc_id, blocXml).catch(console.error);

res.status(201).json({
  succes: true,
  bloc_id,
```

```
      production_id: prod_id,
      emission: {
        micro_GU_bruts: emission.micro_GU_bruts,
        taxe_creation:  emission.taxe_creation,
        micro_GU_nets:  emission.micro_GU_nets,
      },
      statut: 'EN_VALIDATION',
      message: `Production soumise. Validation en cours (quorum 67% requis).`,
    });

  } catch (err) {
    await client.query('ROLLBACK');
    res.status(400).json({ erreur: err.message });
  } finally {
    client.release();
  }
 }
);



// ── POST /api/v1/transactions/transfert ──────────────────────────
// Transférer des µGU entre producteurs (vente de denrée)
app.post('/api/v1/transactions/transfert',
 limitAuditeur,
 reqNiveau('PRODUCTEUR'),
 async (req, res) => {
  const client = await db.connect();
  try {
    await client.query('BEGIN');
```

```
const {
  destinataire_id, montant_GU,
  sous_jacent_desc, production_ref,
  bon_livraison, livraison_prevue,
  signature,
} = req.body;

const expediteur_id = req.acteur.id;
const montant = new Decimal(montant_GU);

// Vérifier le solde
const { rows: exp } = await client.query(
  `SELECT solde_disponible_GU FROM producteurs WHERE id = $1`,
  [expediteur_id]
);
if (!exp.length || new Decimal(exp[0].solde_disponible_gu).lt(montant)) {
  throw new Error(`Solde insuffisant. Disponible : ${exp[0]?.solde_disponible_gu} µGU`);
}

// Calculer les frais réseau
const frais = montant.mul(MET.TAUX_FRAIS_RESEAU)
                .toDecimalPlaces(6, Decimal.ROUND_UP);
const montant_total = montant.plus(frais);

if (new Decimal(exp[0].solde_disponible_gu).lt(montant_total)) {
  throw new Error(`Solde insuffisant pour les frais réseau. Total requis : ${montant_total} µGU`);
}
```

```
// Vérifier la signature de l'expéditeur
const message = `${expediteur_id}:${destinataire_id}:${montant_GU}:${Date.now()}`;
// (Vérification signature désactivée en développement)

const bloc_id = await prochainBlocId('MAIN');
const tx_id   = `TX-TRANSFERT-${new Date().getFullYear()}-${String(Date.now()).slice(-8)}`;

// Insérer le bloc et la transaction
const hash_prec = await dernierHashChaine('MAIN');
const hash     = hashBloc(tx_id + montant_GU + expediteur_id);

await client.query(`
  INSERT INTO blocs
    (id, hash, hash_precedent, hauteur, timestamp,
     type, chaine, noeud_emetteur_id)
  VALUES ($1,$2,$3,
    (SELECT COALESCE(MAX(hauteur),0)+1 FROM blocs WHERE chaine='PRINCIPALE'),
    NOW(),'TRANSACTION','PRINCIPALE','NOEUD-AUTO-TX')
`, [bloc_id, hash, hash_prec]);

await client.query(`
  INSERT INTO transactions
    (id, bloc_id, type, chaine, expediteur_id, destinataire_id,
     montant_GU, frais_reseau_GU,
     sous_jacent_desc, sous_jacent_prod_ref, bon_livraison, livraison_prevue,
     solde_expediteur_avant, solde_expediteur_apres,
     type_economique, timestamp, timestamp_confirmation)
  VALUES ($1,$2,'TRANSFERT','PRINCIPALE',$3,$4,
         $5,$6,$7,$8,$9,$10,$11,
```

```
        $11::numeric - $5, 'COMMERCIAL', NOW(), NOW())
    `, [tx_id, bloc_id, expediteur_id, destinataire_id,
      montant.toString(), frais.toString(),
      sous_jacent_desc, production_ref, bon_livraison, livraison_prevue,
      exp[0].solde_disponible_gu]);

  // Mettre à jour les soldes
  await client.query(`
    UPDATE producteurs
    SET solde_disponible_GU = solde_disponible_GU - $1
    WHERE id = $2
  `, [montant_total.toString(), expediteur_id]);

  await client.query(`
    UPDATE producteurs
    SET solde_disponible_GU = solde_disponible_GU + $1
    WHERE id = $2
  `, [montant.toString(), destinataire_id]);

  // Verser les frais au Fonds Commun
  await crediterFondsCommun(client, {
    type: 'ENTREE_FRAIS_RESEAU',
    montant: frais.toString(),
    source_tx_id: tx_id,
    note: `Frais réseau TX ${tx_id}`,
  });

  await client.query('COMMIT');
```

```
      res.json({
        succes: true,
        tx_id,
        montant_GU:    montant.toString(),
        frais_GU:      frais.toString(),
        expediteur_id,
        destinataire_id,
        timestamp:     new Date().toISOString(),
      });

    } catch (err) {
      await client.query('ROLLBACK');
      res.status(400).json({ erreur: err.message });
    } finally {
      client.release();
    }
  }
);



// ── POST /api/v1/conversion/fiat-vers-GU ──────────────────────
// Convertir des USD (ou autre fiat) en µGU
app.post('/api/v1/conversion/fiat-vers-GU',
  limitAuditeur,
  reqNiveau('PRODUCTEUR'),
  async (req, res) => {
    const client = await db.connect();
    try {
      await client.query('BEGIN');
```

```
const { montant_USD, devise = 'USD' } = req.body;
const acteur_id = req.acteur.id;

if (montant_USD <= 0) throw new Error('Montant invalide');

// Convertir en USD si autre devise (via oracle de change)
const montant_en_USD = devise === 'USD'
  ? montant_USD
  : await convertirVersUSD(montant_USD, devise);

// Calculer la conversion
const conv = await calculerConversionFiat(montant_en_USD);

const bloc_id = await prochainBlocId('MAIN');
const tx_id   = `TX-CONVERSION_FIAT-${new Date().getFullYear()}-${String(Date.now()).slice(-8)}`;
const hash    = hashBloc(tx_id + montant_USD + acteur_id);
const hash_prec = await dernierHashChaine('MAIN');

await client.query(`
  INSERT INTO blocs
    (id, hash, hash_precedent, hauteur, timestamp,
     type, chaine, territoire, noeud_emetteur_id)
  VALUES ($1,$2,$3,
    (SELECT COALESCE(MAX(hauteur),0)+1 FROM blocs WHERE chaine='PRINCIPALE'),
    NOW(),'TRANSACTION','PRINCIPALE','NEXUS','NOEUD-NEXUS-CONV-001')
`, [bloc_id, hash, hash_prec]);

await client.query(`
```

```
  INSERT INTO transactions
    (id, bloc_id, type, chaine, expediteur_id, destinataire_id,
     montant_GU, frais_reseau_GU,
     montant_fiat_valeur, montant_fiat_devise,
     taux_conversion, facteur_penalite,
     solde_expediteur_avant, solde_expediteur_apres,
     type_economique, multiplicateur_taxe,
     timestamp, timestamp_confirmation)
  SELECT $1,$2,'CONVERSION_FIAT','PRINCIPALE',$3,$3,
         $4,$5,$6,$7,$8,$9,
         solde_disponible_GU,
         solde_disponible_GU + $4::numeric,
         'COMMERCIAL', 1.00, NOW(), NOW()
  FROM producteurs WHERE id = $3
` , [tx_id, bloc_id, acteur_id,
    conv.montant_GU_net, conv.frais_reseau_GU,
    montant_en_USD, devise,
    conv.taux_GU_par_USD, conv.facteur_penalite]);

// Créditer le compte du producteur
await client.query(`
  UPDATE producteurs
  SET solde_disponible_GU = solde_disponible_GU + $1
  WHERE id = $2
` , [conv.montant_GU_net, acteur_id]);

// Frais réseau au Fonds Commun
await crediterFondsCommun(client, {
  type: 'ENTREE_FRAIS_RESEAU',
```

```
      montant: conv.frais_reseau_GU,
      source_tx_id: tx_id,
      note: `Conversion fiat ${montant_USD} ${devise} → µGU`,
    });

    await client.query('COMMIT');

    res.json({
      succes: true,
      tx_id,
      conversion: conv,
      message: `${montant_USD} ${devise} → ${conv.montant_GU_net} µGU crédités`,
    });

  } catch (err) {
    await client.query('ROLLBACK');
    res.status(400).json({ erreur: err.message });
  } finally {
    client.release();
  }
 }
);



//
═══════════════════════════════════════════════════════════════════════
═══

// SECTION 6 — ROUTES CASINO (Territoire NEXUS)
```

```
//
═══════════════════════════════════════════════════════════════════════════
═══

// ── POST /api/v1/casino/options/emettre ──────────────────────────
// Émettre un nouveau contrat d'option dans NEXUS
app.post('/api/v1/casino/options/emettre',
 limitAuditeur,
 reqNiveau('PRODUCTEUR'),
 async (req, res) => {
  const client = await db.connect();
  try {
   await client.query('BEGIN');

   const {
    type_option,       // CALL | PUT
    commodite,
    quantite_kg,
    production_ref,    // sous-jacent réel obligatoire
    prix_exercice_GU,
    echeance,
    prime_GU,
    levier,
    vendeur_pseudonyme,
   } = req.body;

   const acheteur_id = req.acteur.id;

   // 1. Vérifier que la commodité est autorisée dans le Casino
```

```
const commoditesAutorisees = [
  'blé_dur','blé_tendre','riz','maïs','soja','orge',
  'café','cacao','energie_solaire','energie_eolienne',
  'eau_douce_certifiee','poisson_durable',
];
if (!commoditesAutorisees.includes(commodite)) {
  throw new Error(` Commodité "${commodite}" non autorisée dans le Casino `);
}

// 2. Vérifier que le sous-jacent réel existe
if (!production_ref) throw new Error('Sous-jacent réel obligatoire');
const { rows: prod } = await client.query(
  ` SELECT quantite_valeur FROM productions WHERE id = $1 AND statut = 'APPROUVÉ' `,
  [production_ref]
);
if (!prod.length) throw new Error('Production de référence non trouvée ou non approuvée');
if (quantite_kg > prod[0].quantite_valeur) {
  throw new Error(
    ` Quantité contractée (${quantite_kg} kg) > production réelle (${prod[0].quantite_valeur} kg) `
  );
}

// 3. Vérifier que l'acheteur a le solde Casino suffisant
const prime = new Decimal(prime_GU);
const levier_val = parseInt(levier);
const taux_taxe = MET.TAXE_CASINO[levier_val];
if (!taux_taxe) throw new Error(` Levier ${levier} non supporté `);

const taxe = prime.mul(taux_taxe).toDecimalPlaces(6, Decimal.ROUND_UP);
```

```
    const montant_net = prime.minus(taxe);

    // 4. Vérifier que position < 0.1% du marché (anti-manipulation)
    const { rows: mktotal } = await client.query(
      `SELECT SUM(prime_payee_GU * levier_valeur) AS total FROM contrats_option WHERE statut = 'ACTIF'`
    );
    const total_marche = new Decimal(mktotal[0].total || '0');
    const pct = total_marche.gt(0)
      ? prime.mul(levier_val).div(total_marche).mul(100)
      : new Decimal(0);
    if (pct.gt(0.1)) {
      throw new Error(
        `Position ${pct.toFixed(4)}% > 0.1% du marché total. Identification obligatoire.`
      );
    }

    // 5. Générer le contrat
    const bloc_id = await prochainBlocId('CASINO');
    const opt_id  = `OPT-${new Date().getFullYear()}-${commodite.slice(0,3).toUpperCase()}-${String(Date.now()).slice(-8)}`;
    const taxe_tx = `TX-TAXE_CASINO-${new Date().getFullYear()}-${String(Date.now()).slice(-8)}`;
    const hash_prec = await dernierHashChaine('CASINO');
    const hash = hashBloc(opt_id + prime_GU + acheteur_id);

    await client.query(`
      INSERT INTO blocs
        (id, hash, hash_precedent, hauteur, timestamp,
         type, chaine, territoire, noeud_emetteur_id)
```

```
  VALUES ($1,$2,$3,
    (SELECT COALESCE(MAX(hauteur),0)+1 FROM blocs WHERE chaine='CASINO'),
    NOW(),'OPTION','CASINO','NEXUS','NOEUD-NEXUS-MARKET-001')
` , [bloc_id, hash, hash_prec]);

await client.query(`
  INSERT INTO contrats_option
    (id, bloc_id, type_option, statut,
     acheteur_pseudonyme, acheteur_adresse, acheteur_pct_marche,
     vendeur_pseudonyme, vendeur_prod_ref,
     commodite, quantite_kg, pct_production_couverte,
     prix_exercice_GU, echeance, prime_payee_GU,
     levier, levier_valeur, taux_taxe_pct,
     taxe_montant_GU, taxe_statut, taxe_tx_id,
     mouvement_net_GU, timestamp_emission)
  VALUES ($1,$2,$3,'ACTIF',
         $4,$5,$6,$7,$8,
         $9,$10,$11,$12,$13,$14,
         $15,$16,$17,$18,'PAYÉE',$19,$20,NOW())
` , [opt_id, bloc_id, type_option,
    `Acheteur_${acheteur_id.slice(-6)}` ,
    `met-casino1:${acheteur_id.slice(-40)}` ,
    pct.toFixed(5),
    vendeur_pseudonyme, production_ref,
    commodite, quantite_kg,
    (quantite_kg / prod[0].quantite_valeur * 100).toFixed(2),
    prix_exercice_GU, echeance, prime_GU,
    levier, levier_val, (taux_taxe * 100).toFixed(1),
    taxe.toString(), taxe_tx, montant_net.toString()]);
```

```
  // Verser la taxe Casino au Fonds Commun
  await crediterFondsCommun(client, {
    type: 'ENTREE_TAXE_CASINO',
    montant: taxe.toString(),
    source_tx_id: taxe_tx,
    note: `Taxe Casino levier ${levier} — contrat ${opt_id}`,
  });

  await client.query('COMMIT');

  res.status(201).json({
    succes: true,
    contrat_id: opt_id,
    type_option,
    commodite,
    prime_GU,
    taxe_GU:       taxe.toString(),
    montant_net_GU: montant_net.toString(),
    taux_taxe_pct:  (taux_taxe * 100).toFixed(1),
    levier,
    echeance,
    statut: 'ACTIF',
  });

} catch (err) {
  await client.query('ROLLBACK');
  res.status(400).json({ erreur: err.message });
} finally {
```

```
    client.release();
   }
  }
);


// ── POST /api/v1/casino/options/:id/exercer ──────────────────────
// Exercer un contrat d'option + traversée de la membrane
app.post('/api/v1/casino/options/:id/exercer',
 limitAuditeur,
 reqNiveau('PRODUCTEUR'),
 async (req, res) => {
  const client = await db.connect();
  try {
   await client.query('BEGIN');

   const { livraison_physique = false } = req.body;
   const opt_id     = req.params.id;
   const acteur_id  = req.acteur.id;

   // Récupérer le contrat
   const { rows: opts } = await client.query(
    `SELECT * FROM contrats_option WHERE id = $1 AND statut = 'ACTIF'`,
    [opt_id]
   );
   if (!opts.length) throw new Error('Contrat non trouvé ou déjà clôturé');
   const opt = opts[0];

   if (new Date() > new Date(opt.echeance)) {
```

```
  throw new Error('Contrat expiré — ne peut plus être exercé');
}

// Prix marché actuel
const { rows: prix } = await client.query(`
  SELECT AVG(t.montant_GU / NULLIF(p.quantite_valeur, 0)) AS prix_spot
  FROM transactions t
  JOIN productions p ON p.id = t.sous_jacent_prod_ref
  WHERE p.denree_type = $1
    AND t.timestamp > NOW() - INTERVAL '1 hour'
`, [opt.commodite]);
const prix_spot = new Decimal(prix[0].prix_spot || opt.prix_exercice_gu);

// Calculer le gain
let gain_brut = new Decimal(0);
if (opt.type_option === 'CALL') {
  gain_brut = prix_spot.minus(opt.prix_exercice_gu)
                .mul(opt.quantite_kg)
                .toDecimalPlaces(6, Decimal.ROUND_DOWN);
} else {
  gain_brut = new Decimal(opt.prix_exercice_gu)
                .minus(prix_spot)
                .mul(opt.quantite_kg)
                .toDecimalPlaces(6, Decimal.ROUND_DOWN);
}

if (gain_brut.lte(0)) {
  throw new Error('Option hors de la monnaie — exercice non rentable');
}
```

```
// Taxe de membrane (1% pour passer Casino → Économie réelle)
const taxe_membrane = gain_brut.mul(MET.TAUX_TAXE_MEMBRANE)
                      .toDecimalPlaces(6, Decimal.ROUND_UP);
const montant_net   = gain_brut.minus(taxe_membrane);

// Enregistrer l'exercice dans la chaîne Casino
const bloc_casino_id = await prochainBlocId('CASINO');
const hash_prec_c    = await dernierHashChaine('CASINO');
const hash_c         = hashBloc(opt_id + 'EXERCICE' + acteur_id);

await client.query(`
  INSERT INTO blocs
    (id, hash, hash_precedent, hauteur, timestamp,
     type, chaine, territoire, noeud_emetteur_id)
  VALUES ($1,$2,$3,
    (SELECT COALESCE(MAX(hauteur),0)+1 FROM blocs WHERE chaine='CASINO'),
    NOW(),'EXERCICE','CASINO','NEXUS','NOEUD-NEXUS-MARKET-001')
`, [bloc_casino_id, hash_c, hash_prec_c]);

// Générer le bloc correspondant dans la chaîne principale (membrane)
const bloc_principal_id = await prochainBlocId('MAIN');
const membrane_id       = `MEM-${bloc_casino_id}`;
const hash_prec_p       = await dernierHashChaine('MAIN');
const hash_p            = hashBloc(membrane_id + montant_net.toString());

await client.query(`
  INSERT INTO blocs
    (id, hash, hash_precedent, hauteur, timestamp,
```

```
    type, chaine, territoire, noeud_emetteur_id)
  VALUES ($1,$2,$3,
    (SELECT COALESCE(MAX(hauteur),0)+1 FROM blocs WHERE chaine='PRINCIPALE'),
    NOW(),'MEMBRANE','PRINCIPALE','NEXUS','NOEUD-NEXUS-MARKET-001')
`, [bloc_principal_id, hash_p, hash_prec_p]);

// Enregistrer le passage membrane
await client.query(`
  INSERT INTO passages_membrane
    (id, bloc_casino_id, bloc_principal_genere_id, contrat_ref,
     acteur_pseudonyme, montant_transfere_GU, taxe_membrane_GU,
     montant_net_GU, livraison_physique_possible, silo_id, quantite_kg,
     timestamp)
  VALUES ($1,$2,$3,$4,$5,$6,$7,$8,$9,$10,$11,NOW())
`, [membrane_id, bloc_casino_id, bloc_principal_id, opt_id,
    `Acheteur_${acteur_id.slice(-6)}`,
    gain_brut.toString(), taxe_membrane.toString(), montant_net.toString(),
    livraison_physique,
    livraison_physique ? opt.vendeur_prod_ref : null,
    livraison_physique ? opt.quantite_kg : null]);

// Créditer le compte réel de l'acheteur
await client.query(`
  UPDATE producteurs
  SET solde_disponible_GU = solde_disponible_GU + $1
  WHERE id = $2
`, [montant_net.toString(), acteur_id]);

// Verser la taxe membrane au Fonds Commun
```

```
await crediterFondsCommun(client, {
  type: 'ENTREE_TAXE_MEMBRANE',
  montant: taxe_membrane.toString(),
  source_tx_id: membrane_id,
  note: `Taxe membrane exercice ${opt_id}`,
});

// Clôturer le contrat
await client.query(`
  UPDATE contrats_option
  SET statut = 'EXERCÉ',
      prix_marche_echeance_GU = $1,
      gain_acheteur_GU = $2,
      timestamp_cloture = NOW()
  WHERE id = $3
`, [prix_spot.toString(), gain_brut.toString(), opt_id]);

await client.query('COMMIT');

res.json({
  succes: true,
  exercice: {
    contrat_id:     opt_id,
    prix_spot_GU:    prix_spot.toString(),
    prix_exercice_GU: opt.prix_exercice_gu,
    gain_brut_GU:    gain_brut.toString(),
    taxe_membrane_GU: taxe_membrane.toString(),
    montant_net_GU:  montant_net.toString(),
    bloc_casino_id,
```

```
        bloc_principal_id,
        livraison_physique_disponible: livraison_physique,
      },
    });

  } catch (err) {
    await client.query('ROLLBACK');
    res.status(400).json({ erreur: err.message });
  } finally {
    client.release();
  }
 }
);



// ═══════════════════════════════════════════════════════════════════════════
// SECTION 7 — ROUTES AUDITEURS (Guilde des Gardiens)
// ═══════════════════════════════════════════════════════════════════════════

// ── GET /api/v1/audit/scan ──────────────────────────────────────────────
// Scanner les productions récentes pour détecter les anomalies
app.get('/api/v1/audit/scan',
 limitAuditeur,
 reqNiveau('ANALYSTE'),
 async (req, res) => {
  try {
```

```
const { rows } = await db.query(`
  WITH historique AS (
    SELECT
      p.producteur_id, p.denree_type,
      AVG(e.micro_GU_nets / NULLIF(p.quantite_valeur, 0)) AS ratio_moy,
      STDDEV(e.micro_GU_nets / NULLIF(p.quantite_valeur, 0)) AS ratio_std,
      COUNT(*) AS nb_obs
    FROM productions p
    JOIN emissions e ON e.production_id = p.id
    WHERE p.timestamp BETWEEN NOW() - INTERVAL '90 days' AND NOW() - INTERVAL '1 day'
      AND p.statut = 'APPROUVÉ'
    GROUP BY p.producteur_id, p.denree_type
    HAVING COUNT(*) >= 3
  )
  SELECT
    p.id AS production_id, p.producteur_id, pr.pseudonyme,
    p.denree_type, p.pays, p.region,
    p.oracle_sat_ndvi,
    ROUND(ABS(e.micro_GU_nets / NULLIF(p.quantite_valeur,0) - h.ratio_moy)
        / NULLIF(h.ratio_std, 0.0001), 2) AS z_score,
    CASE
      WHEN p.oracle_sat_ndvi < 0.1 THEN '🔴 GHOST_PRODUCTION'
      WHEN ABS(e.micro_GU_nets / NULLIF(p.quantite_valeur,0) - h.ratio_moy)
          / NULLIF(h.ratio_std, 0.0001) > 4 THEN '🔴 ANOMALIE_CRITIQUE'
      WHEN ABS(e.micro_GU_nets / NULLIF(p.quantite_valeur,0) - h.ratio_moy)
          / NULLIF(h.ratio_std, 0.0001) > 3 THEN '🟠 ANOMALIE_ÉLEVÉE'
      WHEN ABS(e.micro_GU_nets / NULLIF(p.quantite_valeur,0) - h.ratio_moy)
          / NULLIF(h.ratio_std, 0.0001) > 2 THEN '🟡 ANOMALIE_MODÉRÉE'
```

```
        ELSE '🟢 NORMAL'
      END AS niveau_alerte,
      p.timestamp
    FROM productions p
    JOIN emissions   e  ON e.production_id = p.id
    JOIN producteurs pr ON pr.id = p.producteur_id
    LEFT JOIN historique h ON h.producteur_id = p.producteur_id
                          AND h.denree_type = p.denree_type
    WHERE p.timestamp > NOW() - INTERVAL '24h'
    ORDER BY z_score DESC NULLS LAST
  `);

  res.json({ succes: true, anomalies: rows, scan_timestamp: new Date().toISOString() });
 } catch (err) {
  res.status(500).json({ erreur: err.message });
 }
 }
);


// ── POST /api/v1/audit/signalements ──────────────────────────
// Soumettre un signalement de fraude
app.post('/api/v1/audit/signalements',
 limitAuditeur,
 reqNiveau('ANALYSTE'),
 async (req, res) => {
  try {
   const {
    production_id, type_fraude,
```

```
      z_score_energie, z_score_travail,
      pattern, preuve_chainql,
    } = req.body;

    await db.query(
      `CALL soumettre_signalement($1,$2,$3,$4,$5,$6,$7,$8)`,
      [req.acteur.id, production_id, type_fraude,
       z_score_energie, z_score_travail, pattern,
       preuve_chainql, 'sig:pending']
    );

    res.status(201).json({
      succes: true,
      message: 'Signalement soumis. Le Tribunal examinera dans 30 jours max.',
    });
  } catch (err) {
    res.status(400).json({ erreur: err.message });
  }
 }
);



//
═══════════════════════════════════════════════════════════════════════════
═══
// SECTION 8 — ROUTES TRIBUNAL
//
═══════════════════════════════════════════════════════════════════════════
═══
```

```
// ── POST /api/v1/tribunal/verdicts ──────────────────────────────
// Rendre un verdict sur un signalement
app.post('/api/v1/tribunal/verdicts',
 limitTribunal,
 reqNiveau('TRIBUNAL'),
 async (req, res) => {
  try {
   const {
    signalement_id, verdict,
    montant_fraude_GU, niveau_sanction,
    motivation,
   } = req.body;

   await db.query(
    `CALL cloturer_signalement($1,$2,$3,$4,$5,$6)`,
    [signalement_id, verdict, montant_fraude_GU,
     niveau_sanction, req.acteur.id, 'sig:tribunal']
   );

   // Enregistrer la motivation publiquement
   await db.query(`
    INSERT INTO motivations_tribunal
     (signalement_id, arbitre_id, motivation, timestamp)
    VALUES ($1, $2, $3, NOW())
   `, [signalement_id, req.acteur.id, motivation]);

   res.json({ succes: true, verdict, signalement_id });
  } catch (err) {
   res.status(400).json({ erreur: err.message });
```

```
  }
 }
);



//
═══════════════════════════════════════════════════════════════════════════
═══
// SECTION 9 — FONCTIONS AUXILIAIRES
//
═══════════════════════════════════════════════════════════════════════════
═══

/** Calcule le facteur carbone selon l'empreinte relative */
function calculerFacteurCarbone(pct_vs_mondiale) {
 if (!pct_vs_mondiale) return 1.00;
 if (pct_vs_mondiale < 50)  return 1.15;
 if (pct_vs_mondiale < 80)  return 1.05;
 if (pct_vs_mondiale < 120) return 1.00;
 if (pct_vs_mondiale < 200) return 0.90;
 return 0.75;
}

/** Calcule le facteur temps selon la durée du cycle de production */
function calculerFacteurTemps(cycle) {
 const cycles = {
  'JOURNALIER':   1.00,
  'HEBDOMADAIRE':  1.00,
  'MENSUEL':     1.00,
  'PRINTEMPS':    1.00,  // ~90 jours
```

```
    'AUTOMNE':      1.00,
    'ANNUEL':       1.10,
    'BISANNUEL':    1.25,
    'TRIENNAL':     1.25,
    'QUINQUENNAL':  1.45,
    'DÉCENNAL':     1.70,
  };
  return cycles[cycle?.toUpperCase()] || 1.00;
}

/** Récupère le hash du dernier bloc d'une chaîne */
async function dernierHashChaine(chaine) {
  const { rows } = await db.query(
    `SELECT hash FROM blocs WHERE chaine = $1 ORDER BY hauteur DESC LIMIT 1`,
    [chaine]
  );
  return rows[0]?.hash ||
    'sha3:0000000000000000000000000000000000000000000000000000000000000000';
}

/** Crédite le Fonds Commun et enregistre dans le journal */
async function crediterFondsCommun(client, { type, montant, source_tx_id, note }) {
  const { rows } = await client.query(
    `SELECT MAX(solde_apres_GU) AS solde FROM fonds_commun_journal`
  );
  const solde_avant = new Decimal(rows[0]?.solde || '0');
  const solde_apres = solde_avant.plus(montant);
  const fc_id = `FC-${source_tx_id}`;
```

```
  await client.query(`
    INSERT INTO fonds_commun_journal
      (id, type_mouvement, direction, montant_GU,
       source_tx_id, solde_avant_GU, solde_apres_GU,
       exercice, timestamp, note)
    VALUES ($1,$2,'ENTREE',$3,$4,$5,$6,
            EXTRACT(YEAR FROM NOW())::INTEGER,NOW(),$7)
  `, [fc_id, type, montant, source_tx_id,
      solde_avant.toString(), solde_apres.toString(), note]);
}

/** Génère le XML d'un bloc de production */
function genererBlocProductionXML(params) {
  const { bloc_id, prod_id, producteur, emission,
          denree_type, denree_grade, quantite_kg,
          pays, region, lat, lng, cycle, territoire,
          oracle_satellite, oracle_meteo, oracle_marche,
          F_q, F_r, F_c, F_t } = params;

  return `<?xml version="1.0" encoding="UTF-8"?>
<bloc_production
  xmlns="https://protocol.aequitaverse.org/v1"
  id="${bloc_id}"
  hash="sha3:pending"
  hash_precedent="sha3:pending"
  hauteur="0"
  timestamp="${new Date().toISOString()}"
  type="PRODUCTION"
  chaine="PRINCIPALE">
```

```
<entete>
  <version_protocole>1.0.0</version_protocole>
  <hauteur>0</hauteur>
  <chaine>PRINCIPALE</chaine>
  <territoire>${territoire}</territoire>
  <masse_monetaire_totale>0</masse_monetaire_totale>
  <noeud_emetteur id="NOEUD-AUTO" categorie="A" region="${pays}"/>
</entete>
<producteur id="${producteur.id}" type="${producteur.type}"
          pseudonyme="${producteur.pseudonyme}"
          membre_depuis="${producteur.membre_depuis}">
  <certification>
    <organisme>${producteur.certification_organisme}</organisme>
    <numero>${producteur.certification_numero}</numero>
    <validite>${producteur.certification_validite}</validite>
    <specialite>${denree_type}</specialite>
  </certification>
  <historique_fiabilite>
    <score>${producteur.fiabilite_score}</score>
    <productions_validees>${producteur.productions_validees}</productions_validees>
    <incidents>${producteur.incidents}</incidents>
    <derniere_fraude>N/A</derniere_fraude>
  </historique_fiabilite>
  <caution_deposee>
    <montant>${producteur.caution_montant_gu}</montant>
    <verrouillage_jusqua>${producteur.caution_validite || 'N/A'}</verrouillage_jusqua>
    <taux_applicable>${producteur.caution_taux_pct}%</taux_applicable>
  </caution_deposee>
  <portefeuille>
```

```
    <adresse>${producteur.portefeuille_adresse}</adresse>
    <solde_disponible>${producteur.solde_disponible_gu}</solde_disponible>
    <solde_bloque>${producteur.solde_bloque_gu}</solde_bloque>
  </portefeuille>
</producteur>
<production id="${prod_id}" statut="EN_VALIDATION" cycle="${cycle}">
  <denree>
    <type>${denree_type}</type>
    <grade>${denree_grade}</grade>
    <quantite unite="kg">${quantite_kg}</quantite>
  </denree>
  <localisation>
    <pays>${pays}</pays>
    <region>${region}</region>
    <coordinates lat="${lat}" lng="${lng}"/>
  </localisation>
  <preuves_externes>
    <oracle type="satellite">
      <source>${oracle_satellite?.source || 'ESA-Sentinel-2'}</source>
      <reference>${oracle_satellite?.reference || 'AUTO'}</reference>
      <signature>sig:${crypto.randomBytes(32).toString('hex')}</signature>
    </oracle>
    <oracle type="meteo">
      <source>${oracle_meteo?.source || 'WMO'}</source>
      <reference>${oracle_meteo?.reference || 'AUTO'}</reference>
      <signature>sig:${crypto.randomBytes(32).toString('hex')}</signature>
    </oracle>
  </preuves_externes>
</production>
```

```
<calcul_emission>
 <formule_version>1.0.0</formule_version>
 <variables>
  <var nom="F_qualite" valeur="${F_q}"/>
  <var nom="F_region"  valeur="${F_r}"/>
  <var nom="F_carbone" valeur="${F_c}"/>
  <var nom="F_temps"   valeur="${F_t}"/>
 </variables>
 <etapes>
  <etape ordre="1">${emission.formule}</etape>
 </etapes>
 <micro_GU_bruts>${emission.micro_GU_bruts}</micro_GU_bruts>
</calcul_emission>
<taxe_creation>
 <taux_applicable>1%</taux_applicable>
 <montant_du>${emission.taxe_creation}</montant_du>
 <paiement>
  <statut>EN_ATTENTE</statut>
  <timestamp>${new Date().toISOString()}</timestamp>
  <tx_id>TX-TAXE-PENDING</tx_id>
  <destination>FONDS-COMMUN-MET-GLOBAL</destination>
 </paiement>
 <validation_conditionnelle>
  <condition>taxe_creation.paiement.statut = 'PAYÉE'</condition>
  <si_vrai>PROCÉDER À L'ÉMISSION</si_vrai>
  <si_faux>REJETER — BLOC INVALIDE</si_faux>
 </validation_conditionnelle>
</taxe_creation>
<emission statut="EN_ATTENTE">
```

```
  <micro_GU_bruts>${emission.micro_GU_bruts}</micro_GU_bruts>
  <taxe_deduite>${emission.taxe_creation}</taxe_deduite>
  <micro_GU_nets>${emission.micro_GU_nets}</micro_GU_nets>
  <distribution>
   <part destinataire="${producteur.id}" pct="100"
       tx_id="TX-EMISSION-PENDING">${emission.micro_GU_nets}</part>
  </distribution>
  <masse_monetaire_apres>0</masse_monetaire_apres>
 </emission>
</bloc_production>`;
}

/** Diffuse un bloc aux nœuds validateurs du réseau */
async function diffuserAuxNoeuds(bloc_id, blocXml) {
 const { rows: noeuds } = await db.query(
  `SELECT endpoint FROM noeuds WHERE statut = 'ACTIF' LIMIT 20`
 );
 const fetch = require('node-fetch');
 await Promise.allSettled(
  noeuds.map(n =>
   fetch(`${n.endpoint}/api/v1/validation/recevoir`, {
    method: 'POST',
    headers: { 'Content-Type': 'application/xml' },
    body: blocXml,
    timeout: 5000,
   })
  )
 );
}
```

```
/** Placeholder conversion de devise via oracle externe */
async function convertirVersUSD(montant, devise) {
  // En production : appel à un oracle de taux de change certifié
  const taux = { EUR: 1.08, GBP: 1.27, JPY: 0.0067, CAD: 0.74 };
  return montant * (taux[devise] || 1);
}



//
═══════════════════════════════════════════════════════════════════════════════
═══
// SECTION 10 — GESTION D'ERREURS ET DÉMARRAGE
//
═══════════════════════════════════════════════════════════════════════════════
═══


// Handler 404
app.use((req, res) => {
  res.status(404).json({
    erreur: 'Route non trouvée',
    routes_disponibles: [
      'GET  /api/v1/dashboard',
      'GET  /api/v1/blocs/:id',
      'GET  /api/v1/productions',
      'GET  /api/v1/tracabilite/:id',
      'GET  /api/v1/taux/conversion-fiat',
      'GET  /api/v1/fonds-commun/solde',
      'POST /api/v1/productions/soumettre',
      'POST /api/v1/transactions/transfert',
```

```
      'POST /api/v1/conversion/fiat-vers-GU',
      'POST /api/v1/casino/options/emettre',
      'POST /api/v1/casino/options/:id/exercer',
      'GET  /api/v1/audit/scan',
      'POST /api/v1/audit/signalements',
      'POST /api/v1/tribunal/verdicts',
    ],
  });
});

// Handler erreurs globales
app.use((err, req, res, next) => {
  console.error('[AEQUITAVERSE API]', err.stack);
  res.status(500).json({ erreur: 'Erreur interne du serveur MET' });
});

// Démarrage
const PORT = process.env.PORT || 3000;
app.listen(PORT, () => {
  console.log(`
  ╔═══════════════════════════════════════════════╗
  ║  AEQUITAVERSE API — Port ${PORT}        ║
  ║  Chaînes : PRINCIPALE | CASINO | FC   ║
  ║  ChainQL : PostgreSQL connecté        ║
  ║  Schéma  : XSD v1.0.0 chargé          ║
  ╚═══════════════════════════════════════════════╝
  `);
});
```

```
module.exports = app;
```