```
"""
エアドロップ追跡 API サーバー
起動: uvicorn main:app --reload --port 8000
"""
from fastapi import FastAPI, HTTPException
from fastapi.middleware.cors import CORSMiddleware
from pydantic import BaseModel
from typing import Optional
import sqlite3, json, os
from datetime import datetime

app = FastAPI(title="Airdrop Tracker API")

app.add_middleware(
    CORSMiddleware,
    allow_origins=["http://localhost:3000", "http://localhost:5173"],
    allow_methods=["*"],
    allow_headers=["*"],
)

DB_PATH = "airdrop.db"

# ---------- DB初期化 ----------
def init_db():
    con = sqlite3.connect(DB_PATH)
    cur = con.cursor()
    cur.executescript("""
        CREATE TABLE IF NOT EXISTS airdrops (
            id           INTEGER PRIMARY KEY AUTOINCREMENT,
            name         TEXT NOT NULL,
            symbol       TEXT,
            logo         TEXT,
            category     TEXT,
            chain        TEXT,
            status       TEXT,
            urgency      TEXT,
            deadline     TEXT,
            estimated_value TEXT,
            difficulty   INTEGER DEFAULT 1,
            description TEXT,
            twitter      TEXT,
            confirmed    INTEGER DEFAULT 0,
            created_at   TEXT DEFAULT (datetime('now')),
            updated_at   TEXT DEFAULT (datetime('now'))
        );

        CREATE TABLE IF NOT EXISTS tasks (
            id           INTEGER PRIMARY KEY AUTOINCREMENT,
            airdrop_id   INTEGER NOT NULL,
            label        TEXT NOT NULL,
            done         INTEGER DEFAULT 0,
            points       INTEGER DEFAULT 10,
            done_at      TEXT,
            FOREIGN KEY (airdrop_id) REFERENCES airdrops(id) ON DELETE CASCADE
        );

        CREATE TABLE IF NOT EXISTS notifications (
            id           INTEGER PRIMARY KEY AUTOINCREMENT,
            message      TEXT NOT NULL,
            type         TEXT DEFAULT 'info',
            read         INTEGER DEFAULT 0,
            created_at   TEXT DEFAULT (datetime('now'))
        );
    """)

    # 初期データ（初回のみ）
    count = cur.execute("SELECT COUNT(*) FROM airdrops").fetchone()[0]
    if count == 0:
        seed_data = [
            ("LayerZero","ZRO","🔵","インフラ","マルチチェーン","確認済み","高","2025-
            ("Scroll","SCR","📜","L2","Ethereum L2","未確認・有力","中","2025-04-01
            ("ZKsync","ZK","⚡","L2","Ethereum L2","配布済み（追加あり？）","低","未
            ("Hyperliquid","HYPE","🌊","DeFi","独自チェーン","配布済み・高額実績","低"
            ("Monad","MON","🟣","L1","新規L1","テストネット中","高","2025-05-01","未
            ("Berachain","BERA","🐻","L1","EVM L1","メインネット間近","高","2025-02-
        ]
        for d in seed_data:
            cur.execute("""INSERT INTO airdrops(name,symbol,logo,category,chain,s
                          VALUES(?,?,?,?,?,?,?,?,?,?,?,?,?)""", d)

        tasks_seed = {
            "LayerZero": [("ブリッジを5回以上実行",1,20),("3チェーン以上で操作",1,15),('
            "Scroll":    [("Scrollにブリッジ（ETH）",1,20),("DEXでスワップ1回以上",0,2
            "ZKsync":    [("ZKsyncでブリッジ",1,10),("SyncSwapでスワップ",1,20),("NF
            "Hyperliquid":[("パーペチュアル取引を実行",0,30),("累計取引量$10,000以上",0,
            "Monad":     [("テストネットに参加",0,20),("テストネットで取引",0,25),("Dis
            "Berachain": [("テストネット参加済み",1,15),("BGTをステーク",0,30),("BEX（
        }
        for name, tasks in tasks_seed.items():
            aid = cur.execute("SELECT id FROM airdrops WHERE name=?", (name,)).fe
            for label, done, pts in tasks:
                cur.execute("INSERT INTO tasks(airdrop_id,label,done,points) VALU

        # 初期通知
        notifs = [
            ("Berachain メインネット間近！タスク完了を急いで", "urgent"),
            ("LayerZero 締め切りまで24日", "warning"),
            ("Monad テストネット開始！参加可能になりました", "info"),
        ]
        for msg, t in notifs:
            cur.execute("INSERT INTO notifications(message,type) VALUES(?,?)", (m

    con.commit()
    con.close()

init_db()

# ---------- ヘルパー ----------
def get_con():
    con = sqlite3.connect(DB_PATH)
    con.row_factory = sqlite3.Row
    con.execute("PRAGMA foreign_keys = ON")
    return con

def row_to_dict(row):
    return dict(row) if row else None

# ---------- モデル ----------
class AirdropCreate(BaseModel):
    name: str
    symbol: Optional[str] = ""
    logo: Optional[str] = "🪂"
    category: Optional[str] = "DeFi"
    chain: Optional[str] = ""
    status: Optional[str] = "未確認"
    urgency: Optional[str] = "中"
    deadline: Optional[str] = "未定"
    estimated_value: Optional[str] = "未定"
    difficulty: Optional[int] = 1
    description: Optional[str] = ""
    twitter: Optional[str] = ""
    confirmed: Optional[bool] = False

class AirdropUpdate(AirdropCreate):
    pass

class TaskCreate(BaseModel):
    label: str
    points: Optional[int] = 10

class TaskUpdate(BaseModel):
    done: Optional[bool] = None
    label: Optional[str] = None
    points: Optional[int] = None

# ---------- エンドポイント ----------

@app.get("/")
def root():
    return {"status": "ok", "message": "Airdrop Tracker API"}

# -- Airdrops --
@app.get("/airdrops")
def list_airdrops():
    con = get_con()
    rows = con.execute("SELECT * FROM airdrops ORDER BY CASE urgency WHEN '高' TH
    result = []
    for row in rows:
        a = row_to_dict(row)
        tasks = con.execute("SELECT * FROM tasks WHERE airdrop_id=?", (a["id"],))
        a["tasks"] = [row_to_dict(t) for t in tasks]
        a["confirmed"] = bool(a["confirmed"])
        result.append(a)
    con.close()
    return result

@app.post("/airdrops", status_code=201)
def create_airdrop(data: AirdropCreate):
    con = get_con()
    cur = con.execute("""
        INSERT INTO airdrops(name,symbol,logo,category,chain,status,urgency,deadl
        VALUES(?,?,?,?,?,?,?,?,?,?,?,?,?,?)
    """, (data.name,data.symbol,data.logo,data.category,data.chain,data.status,da
          data.deadline,data.estimated_value,data.difficulty,data.description,dat
    con.commit()
    new_id = cur.lastrowid
    row = con.execute("SELECT * FROM airdrops WHERE id=?", (new_id,)).fetchone()
    result = row_to_dict(row)
    result["tasks"] = []
    con.close()
    return result

@app.put("/airdrops/{aid}")
def update_airdrop(aid: int, data: AirdropUpdate):
    con = get_con()
    con.execute("""
        UPDATE airdrops SET name=?,symbol=?,logo=?,category=?,chain=?,status=?,ur
        deadline=?,estimated_value=?,difficulty=?,description=?,twitter=?,confirm
        WHERE id=?
    """, (data.name,data.symbol,data.logo,data.category,data.chain,data.status,da
          data.deadline,data.estimated_value,data.difficulty,data.description,dat
    con.commit()
    con.close()
    return {"ok": True}

@app.delete("/airdrops/{aid}")
def delete_airdrop(aid: int):
    con = get_con()
    con.execute("DELETE FROM airdrops WHERE id=?", (aid,))
    con.commit()
    con.close()
    return {"ok": True}

# -- Tasks --
@app.post("/airdrops/{aid}/tasks", status_code=201)
def add_task(aid: int, data: TaskCreate):
    con = get_con()
    cur = con.execute("INSERT INTO tasks(airdrop_id,label,points) VALUES(?,?,?)",
    con.commit()
    row = con.execute("SELECT * FROM tasks WHERE id=?", (cur.lastrowid,)).fetchon
    con.close()
    return row_to_dict(row)

@app.patch("/tasks/{tid}")
def update_task(tid: int, data: TaskUpdate):
    con = get_con()
    if data.done is not None:
        done_at = datetime.now().isoformat() if data.done else None
        con.execute("UPDATE tasks SET done=?, done_at=? WHERE id=?", (int(data.do
    if data.label is not None:
        con.execute("UPDATE tasks SET label=? WHERE id=?", (data.label, tid))
    if data.points is not None:
        con.execute("UPDATE tasks SET points=? WHERE id=?", (data.points, tid))
    con.commit()
    row = con.execute("SELECT * FROM tasks WHERE id=?", (tid,)).fetchone()
    con.close()
    return row_to_dict(row)

@app.delete("/tasks/{tid}")
def delete_task(tid: int):
    con = get_con()
    con.execute("DELETE FROM tasks WHERE id=?", (tid,))
    con.commit()
    con.close()
    return {"ok": True}

# -- Notifications --
@app.get("/notifications")
def list_notifications():
    con = get_con()
    rows = con.execute("SELECT * FROM notifications ORDER BY created_at DESC LIMI
    con.close()
    return [row_to_dict(r) for r in rows]

@app.patch("/notifications/{nid}/read")
def mark_read(nid: int):
    con = get_con()
    con.execute("UPDATE notifications SET read=1 WHERE id=?", (nid,))
    con.commit()
    con.close()
    return {"ok": True}

# -- Stats --
@app.get("/stats")
def get_stats():
    con = get_con()
    total = con.execute("SELECT COUNT(*) FROM airdrops").fetchone()[0]
    active = con.execute("SELECT COUNT(*) FROM airdrops WHERE deadline NOT LIKE '
    urgent = con.execute("SELECT COUNT(*) FROM airdrops WHERE urgency='高'").fetc
    tasks_total = con.execute("SELECT COUNT(*) FROM tasks").fetchone()[0]
    tasks_done = con.execute("SELECT COUNT(*) FROM tasks WHERE done=1").fetchone(
    avg_progress = round(tasks_done / tasks_total * 100) if tasks_total > 0 else
    con.close()
    return {"total": total, "active": active, "urgent": urgent, "avg_progress": a
```