from fastapi import FastAPI, HTTPException

from fastapi.middleware.cors import CORSMiddleware

from pydantic import BaseModel, EmailStr, Field

from typing import List, Optional, Literal, Dict, Any

import uuid

from datetime import datetime

app = FastAPI(title="Scam Likely Mini API", version="0.3")

app.add\_middleware(

CORSMiddleware,

allow\_origins=["\*"], # tighten later

allow\_credentials=True,

allow\_methods=["\*"],

allow\_headers=["\*"],

)

# ---------------- In-memory storage (for demo only) ----------------

ENTITIES: Dict[str, Dict[str, Any]] = {}

REPORTS: Dict[str, Dict[str, Any]] = {}

DISPUTES: Dict[str, Dict[str, Any]] = {} # key = dispute\_id

# seed one entity + published report

if not ENTITIES:

ent\_id = "demo-1"

ENTITIES[ent\_id] = {

"id": ent\_id,

"display\_name": "ABC Web Solutions",

"type": "business",

"top\_identifier": "+1 (443) 555-0123",

"phones": ["+14435550123"],

"emails": ["owner@abcweb.co"],

"state": "MD",

"city": "Baltimore",

"report\_count": 1,

}

seed\_rid = str(uuid.uuid4())

REPORTS[seed\_rid] = {

"id": seed\_rid,

"entity\_id": ent\_id,

"category": "Non-delivery",

"narrative": "Paid $500 for a website; after 6 weeks nothing delivered.",

"amount\_cents": 50000,

"incident\_date": "2025-08-20",

"incident\_mode": "digital",

"incident\_location": {"scammer\_origin": {"city": "Baltimore", "state": "MD", "country": "US"}},

"reporter\_public\_anonymous": True,

"status": "published",

"created\_at": datetime.utcnow().isoformat() + "Z",

}

# ---------------- Schemas ----------------

class Identifiers(BaseModel):

gov\_name: Optional[str] = None

business\_name: Optional[str] = None

phones: List[str] = []

emails: List[EmailStr] = []

handles: List[str] = []

websites: List[str] = []

class ReportCreate(BaseModel):

identifiers: Identifiers

category: str

narrative: str = Field(min\_length=50)

amount\_cents: Optional[int] = None

currency: str = "USD"

incident\_date: Optional[str] = None

incident\_mode: Literal["digital", "in\_person"]

incident\_location: Dict[str, Any]

reporter\_public\_anonymous: bool = True

reporter\_email: Optional[EmailStr] = None # demo only

class DisputeCreate(BaseModel):

contact\_email: EmailStr

text: str = Field(min\_length=20, max\_length=2000)

public\_anonymous: bool = True # if true, show as "Official reply"

# ---------------- Health ----------------

@app.get("/healthz")

def health():

return {"ok": True}

# ---------------- Home list (only published/disputed) ----------------

@app.get("/v1/recent-entities")

def recent\_entities():

items = []

for e in ENTITIES.values():

count = sum(1 for r in REPORTS.values()

if r["entity\_id"] == e["id"] and r["status"] in ("published", "disputed"))

if count:

items.append({

"id": e["id"],

"display\_name": e["display\_name"],

"top\_identifier": e.get("top\_identifier") or (e.get("phones") or e.get("emails") or [""])[0],

"state": e.get("state"),

"city": e.get("city"),

"report\_count": count,

"status": "published",

})

return {"items": items}

# ---------------- Search (very simple) ----------------

@app.get("/v1/search")

def search(q: str, state: Optional[str] = None, city: Optional[str] = None):

q\_lower = q.lower()

results = []

for e in ENTITIES.values():

hay = " ".join([e.get("display\_name",""),

" ".join(e.get("phones", [])),

" ".join(e.get("emails", []))]).lower()

if q\_lower in hay:

if state and e.get("state") != state:

continue

if city and e.get("city") != city:

continue

count = sum(1 for r in REPORTS.values()

if r["entity\_id"] == e["id"] and r["status"] in ("published", "disputed"))

results.append({

"id": e["id"],

"display\_name": e["display\_name"],

"top\_identifier": e.get("top\_identifier") or (e.get("phones") or e.get("emails") or [""])[0],

"state": e.get("state"),

"city": e.get("city"),

"report\_count": count or 0

})

return {"items": results, "next\_page": None}

# ---------------- Submit Report ----------------

@app.post("/v1/reports")

def create\_report(payload: ReportCreate):

# Validate location rules

if payload.incident\_mode == "digital":

origin = payload.incident\_location.get("scammer\_origin")

unknown = payload.incident\_location.get("unknown")

if not origin and not unknown:

raise HTTPException(400, detail="For digital incidents, include incident\_location.scammer\_origin or set unknown=true.")

elif payload.incident\_mode == "in\_person":

loc = payload.incident\_location

required = all([loc.get("city"), loc.get("state"), loc.get("country")])

if not required:

raise HTTPException(400, detail="For in-person incidents, city/state/country are required.")

else:

raise HTTPException(400, detail="incident\_mode must be 'digital' or 'in\_person'.")

# Find/create entity (naive demo logic)

display\_name = payload.identifiers.business\_name or payload.identifiers.gov\_name or "Unknown"

match\_id = None

for e in ENTITIES.values():

if any(p in (e.get("phones") or []) for p in payload.identifiers.phones) or \

any(str(m).lower() in [em.lower() for em in (e.get("emails") or [])] for m in payload.identifiers.emails):

match\_id = e["id"]; break

if not match\_id:

eid = str(uuid.uuid4())

ENTITIES[eid] = {

"id": eid,

"display\_name": display\_name,

"type": "business" if payload.identifiers.business\_name else "person",

"phones": payload.identifiers.phones,

"emails": [str(e) for e in payload.identifiers.emails],

"state": payload.incident\_location.get("state"),

"city": payload.incident\_location.get("city"),

}

match\_id = eid

rid = str(uuid.uuid4())

REPORTS[rid] = {

"id": rid,

"entity\_id": match\_id,

"category": payload.category,

"narrative": payload.narrative,

"amount\_cents": payload.amount\_cents,

"incident\_date": payload.incident\_date,

"incident\_mode": payload.incident\_mode,

"incident\_location": payload.incident\_location,

"reporter\_public\_anonymous": payload.reporter\_public\_anonymous,

"status": "pending", # moderation required

"reporter\_email": str(payload.reporter\_email) if payload.reporter\_email else None,

"created\_at": datetime.utcnow().isoformat() + "Z",

}

return {"id": rid, "status": "pending", "entity\_id": match\_id}

# ---------------- My Reports (demo) ----------------

@app.get("/v1/reports/mine")

def my\_reports(email: EmailStr):

items = [r for r in REPORTS.values()

if r.get("reporter\_email") and r["reporter\_email"].lower() == str(email).lower()]

return {"items": items}

# ---------------- NEW: Publish a report (demo moderation) ----------------

@app.post("/v1/reports/{report\_id}/publish")

def publish\_report(report\_id: str):

r = REPORTS.get(report\_id)

if not r:

raise HTTPException(404, detail="report not found")

r["status"] = "published"

# bump entity report\_count

e = ENTITIES.get(r["entity\_id"])

if e:

e["report\_count"] = sum(1 for rr in REPORTS.values()

if rr["entity\_id"] == e["id"] and rr["status"] == "published")

return {"id": r["id"], "status": r["status"]}

# ---------------- NEW: Entity detail with reports ----------------

@app.get("/v1/entities/{entity\_id}")

def entity\_detail(entity\_id: str):

e = ENTITIES.get(entity\_id)

if not e:

raise HTTPException(404, detail="entity not found")

reports = [r for r in REPORTS.values()

if r["entity\_id"] == entity\_id and r["status"] in ("published", "disputed")]

return {

"id": e["id"],

"display\_name": e.get("display\_name"),

"type": e.get("type"),

"phones": e.get("phones", []),

"emails": e.get("emails", []),

"state": e.get("state"),

"city": e.get("city"),

"reports": reports

}

# --------------- Disputes / Right-to-Reply ---------------

# Create a dispute / reply for a report

@app.post("/v1/reports/{report\_id}/dispute")

def create\_dispute(report\_id: str, payload: DisputeCreate):

r = REPORTS.get(report\_id)

if not r:

raise HTTPException(404, detail="report not found")

did = str(uuid.uuid4())

DISPUTES[did] = {

"id": did,

"report\_id": report\_id,

"entity\_id": r["entity\_id"],

"contact\_email": str(payload.contact\_email),

"text": payload.text.strip(),

"public\_anonymous": bool(payload.public\_anonymous),

"status": "open", # open | under\_review | resolved\_unlisted | resolved\_published

"created\_at": datetime.utcnow().isoformat() + "Z",

}

# mark the report as disputed so UI can badge it

r["status"] = "disputed"

return {"id": did, "report\_id": report\_id, "status": "open"}

# List disputes for a single report (public)

@app.get("/v1/reports/{report\_id}/disputes")

def list\_disputes(report\_id: str):

if report\_id not in [rr["id"] for rr in REPORTS.values()]:

raise HTTPException(404, detail="report not found")

items = [d for d in DISPUTES.values() if d["report\_id"] == report\_id]

items.sort(key=lambda d: d["created\_at"], reverse=True)

return {"items": items}

# Optional: report detail with disputes bundled

@app.get("/v1/reports/{report\_id}")

def get\_report(report\_id: str):

r = REPORTS.get(report\_id)

if not r:

raise HTTPException(404, detail="report not found")

disputes = [d for d in DISPUTES.values() if d["report\_id"] == report\_id]

return {"report": r, "disputes": disputes}