from datetime import datetime

from decimal import Decimal

from typing import List, Optional

from pydantic import BaseModel

import uuid

import sqlite3

from fastapi import FastAPI, HTTPException, Depends

from fastapi.security import OAuth2PasswordBearer

from passlib.context import CryptContext

# Data Models

class User(BaseModel):

id: str

username: str

email: str

full\_name: str

role: str

created\_at: datetime

class Property(BaseModel):

id: str

title: str

address: str

country: str

price: Decimal

currency: str

status: str

owner\_id: Optional[str]

created\_at: datetime

class Transaction(BaseModel):

id: str

property\_id: str

buyer\_id: str

seller\_id: str

amount: Decimal

currency: str

status: str

created\_at: datetime

# Database Connection

def get\_db():

conn = sqlite3.connect('real\_estate\_banking.db')

try:

yield conn

finally:

conn.close()

# Initialize FastAPI app

app = FastAPI(title="International Real Estate Banking System")

pwd\_context = CryptContext(schemes=["bcrypt"], deprecated="auto")

oauth2\_scheme = OAuth2PasswordBearer(tokenUrl="token")

# Database initialization

def init\_db():

conn = next(get\_db())

cursor = conn.cursor()

# Create Users table

cursor.execute('''

CREATE TABLE IF NOT EXISTS users (

id TEXT PRIMARY KEY,

username TEXT UNIQUE,

email TEXT UNIQUE,

full\_name TEXT,

role TEXT,

password\_hash TEXT,

created\_at TIMESTAMP

)

''')

# Create Properties table

cursor.execute('''

CREATE TABLE IF NOT EXISTS properties (

id TEXT PRIMARY KEY,

title TEXT,

address TEXT,

country TEXT,

price DECIMAL,

currency TEXT,

status TEXT,

owner\_id TEXT,

created\_at TIMESTAMP,

FOREIGN KEY (owner\_id) REFERENCES users (id)

)

''')

# Create Transactions table

cursor.execute('''

CREATE TABLE IF NOT EXISTS transactions (

id TEXT PRIMARY KEY,

property\_id TEXT,

buyer\_id TEXT,

seller\_id TEXT,

amount DECIMAL,

currency TEXT,

status TEXT,

created\_at TIMESTAMP,

FOREIGN KEY (property\_id) REFERENCES properties (id),

FOREIGN KEY (buyer\_id) REFERENCES users (id),

FOREIGN KEY (seller\_id) REFERENCES users (id)

)

''')

conn.commit()

# Property Management

class PropertyService:

@staticmethod

async def list\_properties(

db: sqlite3.Connection,

country: Optional[str] = None,

max\_price: Optional[float] = None

) -> List[Property]:

cursor = db.cursor()

query = "SELECT \* FROM properties WHERE 1=1"

params = []

if country:

query += " AND country = ?"

params.append(country)

if max\_price:

query += " AND price <= ?"

params.append(max\_price)

cursor.execute(query, params)

return [Property(\*\*dict(zip(['id', 'title', 'address', 'country', 'price', 'currency', 'status', 'owner\_id', 'created\_at'], row)))

for row in cursor.fetchall()]

@staticmethod

async def create\_property(

db: sqlite3.Connection,

property\_data: dict

) -> Property:

property\_id = str(uuid.uuid4())

cursor = db.cursor()

cursor.execute('''

INSERT INTO properties (id, title, address, country, price, currency, status, owner\_id, created\_at)

VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?)

''', (

property\_id,

property\_data['title'],

property\_data['address'],

property\_data['country'],

property\_data['price'],

property\_data['currency'],

'available',

property\_data.get('owner\_id'),

datetime.utcnow()

))

db.commit()

return Property(id=property\_id, \*\*property\_data)

# Transaction Processing

class TransactionService:

@staticmethod

async def create\_transaction(

db: sqlite3.Connection,

property\_id: str,

buyer\_id: str,

amount: Decimal,

currency: str

) -> Transaction:

# Get property details

cursor = db.cursor()

cursor.execute("SELECT owner\_id, status FROM properties WHERE id = ?", (property\_id,))

property\_data = cursor.fetchone()

if not property\_data:

raise HTTPException(status\_code=404, detail="Property not found")

owner\_id, status = property\_data

if status != 'available':

raise HTTPException(status\_code=400, detail="Property is not available for purchase")

# Create transaction

transaction\_id = str(uuid.uuid4())

cursor.execute('''

INSERT INTO transactions (id, property\_id, buyer\_id, seller\_id, amount, currency, status, created\_at)

VALUES (?, ?, ?, ?, ?, ?, ?, ?)

''', (

transaction\_id,

property\_id,

buyer\_id,

owner\_id,

amount,

currency,

'pending',

datetime.utcnow()

))

# Update property status

cursor.execute('''

UPDATE properties

SET status = 'under\_contract'

WHERE id = ?

''', (property\_id,))

db.commit()

return Transaction(

id=transaction\_id,

property\_id=property\_id,

buyer\_id=buyer\_id,

seller\_id=owner\_id,

amount=amount,

currency=currency,

status='pending',

created\_at=datetime.utcnow()

)

# API Routes

@app.post("/properties/")

async def create\_property(

property\_data: dict,

db: sqlite3.Connection = Depends(get\_db)

):

return await PropertyService.create\_property(db, property\_data)

@app.get("/properties/")

async def list\_properties(

country: Optional[str] = None,

max\_price: Optional[float] = None,

db: sqlite3.Connection = Depends(get\_db)

):

return await PropertyService.list\_properties(db, country, max\_price)

@app.post("/transactions/")

async def create\_transaction(

property\_id: str,

buyer\_id: str,

amount: float,

currency: str,

db: sqlite3.Connection = Depends(get\_db)

):

return await TransactionService.create\_transaction(

db,

property\_id,

buyer\_id,

Decimal(str(amount)),

currency

)

# Initialize database on startup

@app.on\_event("startup")

async def startup\_event():

init\_db()