Source Code: AI Generated Report

Generated on: 2026-02-07 13:37:25

```
from fastapi import FastAPI, Depends, HTTPException, status
from fastapi.security import OAuth2PasswordBearer, OAuth2PasswordRequestForm
from sqlalchemy import create_engine, Column, Integer, String, Float, Date
from sqlalchemy.ext.declarative import declarative_base
from sqlalchemy.orm import sessionmaker, Session
import hashlib
import jwt
from datetime import datetime, timedelta
from pydantic import BaseModel
from typing import List

DATABASE_URL = "sqlite:///:memory:"
engine = create_engine(DATABASE_URL)
SessionLocal = sessionmaker(autocommit=False, autoflush=False, bind=engine)
Base = declarative_base()

oauth2_scheme = OAuth2PasswordBearer(tokenUrl="token")

class User(Base):
    __tablename__ = "users"
    id = Column(Integer, primary_key=True, index=True)
    username = Column(String, unique=True, index=True)
    hashed_password = Column(String)
```

```
class Transaction(Base):
    __tablename__ = "transactions"
    id = Column(Integer, primary_key=True, index=True)
    amount = Column(Float)
    type = Column(String)  # income or expense
    category = Column(String)
    date = Column(Date)
    notes = Column(String)

Base.metadata.create_all(bind=engine)

class UserCreate(BaseModel):
    username: str
    password: str

class UserInDB(UserCreate):
    hashed_password: str

class TransactionCreate(BaseModel):
    amount: float
    type: str
    category: str
    date: str
    notes: str
```

```
class Token(BaseModel):
    access_token: str
    token_type: str


class TokenData(BaseModel):
    username: str


app = FastAPI()


def create_hashed_password(password: str) -> str:
    return hashlib.sha256(password.encode()).hexdigest()


def verify_password(plain_password: str, hashed_password: str) -> bool:
    return create_hashed_password(plain_password) == hashed_password


def get_user(db: Session, username: str):
    return db.query(User).filter(User.username == username).first()


def create_user(db: Session, user: UserCreate):
                                    db_user         =         User(username=user.username,
hashed_password=create_hashed_password(user.password))
    db.add(db_user)
    db.commit()
    db.refresh(db_user)
    return db_user
```

```
def create_access_token(data: dict, expires_delta: timedelta = None):
    to_encode = data.copy()
    if expires_delta:
        expire = datetime.utcnow() + expires_delta
    else:
        expire = datetime.utcnow() + timedelta(minutes=15)
    to_encode.update({"exp": expire})
    return jwt.encode(to_encode, "secret", algorithm="HS256")


@app.post("/token", response_model=Token)
async def login(form_data: OAuth2PasswordRequestForm = Depends()):
    db = SessionLocal()
    user = get_user(db, form_data.username)
    if not user or not verify_password(form_data.password, user.hashed_password):
          raise HTTPException(status_code=status.HTTP_401_UNAUTHORIZED, detail="Incorrect
username or password", headers={"WWW-Authenticate": "Bearer"})
    access_token = create_access_token(data={"sub": user.username})
    return {"access_token": access_token, "token_type": "bearer"}


@app.post("/transactions/", response_model=TransactionCreate)
def     create_transaction(transaction:     TransactionCreate,     db:     Session     =
Depends(SessionLocal)):
    db_transaction = Transaction(**transaction.dict())
    db.add(db_transaction)
    db.commit()
    db.refresh(db_transaction)
```

```
    return db_transaction


@app.get("/transactions/", response_model=List[TransactionCreate])
def read_transactions(skip: int = 0, limit: int = 10, db: Session =
Depends(SessionLocal)):
    transactions = db.query(Transaction).offset(skip).limit(limit).all()
    return transactions


if __name__ == '__main__':
    from fastapi.testclient import TestClient
    client = TestClient(app)
        response = client.post("/token", data={"username": "testuser", "password":
"testpass"})
    print("Login Response:", response.json())
     response = client.post("/transactions/", json={"amount": 100.0, "type": "income",
"category": "salary", "date": "2023-10-01", "notes": "Monthly salary"})
    print("Transaction Creation Response:", response.json())
    response = client.get("/transactions/")
    print("Transactions:", response.json())
```