Source Code: AI Generated Report

Generated on: 2026-02-07 13:53:31

```
from fastapi import FastAPI, Depends, HTTPException, status

from fastapi.security import OAuth2PasswordBearer, OAuth2PasswordRequestForm

from pydantic import BaseModel

from sqlalchemy import create_engine, Column, Integer, String, Float, Date, ForeignKey

from sqlalchemy.orm import sessionmaker, DeclarativeBase, relationship

from sqlalchemy.ext.declarative import declarative_base

from sqlalchemy.orm import Session

from datetime import datetime

from fastapi.testclient import TestClient

import hashlib



DATABASE_URL = "sqlite:///:memory:"



engine = create_engine(DATABASE_URL, connect_args={"check_same_thread": False})

SessionLocal = sessionmaker(autocommit=False, autoflush=False, bind=engine)



class Base(DeclarativeBase):

    pass



class User(Base):

    __tablename__ = "users"

    id = Column(Integer, primary_key=True, index=True)

    username = Column(String, unique=True, index=True)
```

```python
    hashed_password = Column(String)


class Income(Base):

    __tablename__ = "income"

    id = Column(Integer, primary_key=True, index=True)

    user_id = Column(Integer, ForeignKey("users.id"))

    amount = Column(Float)

    source = Column(String)

    date = Column(Date)


class Expense(Base):

    __tablename__ = "expenses"

    id = Column(Integer, primary_key=True, index=True)

    user_id = Column(Integer, ForeignKey("users.id"))

    amount = Column(Float)

    category = Column(String)

    date = Column(Date)


class Budget(Base):

    __tablename__ = "budgets"

    id = Column(Integer, primary_key=True, index=True)

    user_id = Column(Integer, ForeignKey("users.id"))

    category = Column(String)

    limit = Column(Float)


Base.metadata.create_all(bind=engine)
```

```
app = FastAPI()

oauth2_scheme = OAuth2PasswordBearer(tokenUrl="token")


class UserCreate(BaseModel):

    username: str

    password: str


class UserInDB(UserCreate):

    hashed_password: str


class IncomeCreate(BaseModel):

    amount: float

    source: str

    date: datetime


class ExpenseCreate(BaseModel):

    amount: float

    category: str

    date: datetime


class BudgetCreate(BaseModel):

    category: str

    limit: float


def get_db():
    db = SessionLocal()

    try:

        yield db

    finally:

        db.close()



def hash_password(password: str):

    return hashlib.sha256(password.encode()).hexdigest()



@app.post("/register", response_model=UserCreate)

def register(user: UserCreate, db: Session = Depends(get_db)):

    user.hashed_password = hash_password(user.password)

    db_user = User(**user.dict())

    db.add(db_user)

    db.commit()

    db.refresh(db_user)

    return db_user



@app.post("/token")

def   login(form_data:   OAuth2PasswordRequestForm   =   Depends(),   db:   Session   =
Depends(get_db)):

    user = db.query(User).filter(User.username == form_data.username).first()

    if not user or user.hashed_password != hash_password(form_data.password):

            raise  HTTPException(status_code=status.HTTP_401_UNAUTHORIZED,  detail="Invalid
credentials")

    return {"access_token": user.username, "token_type": "bearer"}
```

```
@app.post("/income/", response_model=IncomeCreate)

def create_income(income: IncomeCreate, db: Session = Depends(get_db)):

    db_income = Income(**income.dict(), user_id=1)  # Mock user_id

    db.add(db_income)

    db.commit()

    db.refresh(db_income)

    return db_income


@app.post("/expenses/", response_model=ExpenseCreate)

def create_expense(expense: ExpenseCreate, db: Session = Depends(get_db)):

    db_expense = Expense(**expense.dict(), user_id=1)  # Mock user_id

    db.add(db_expense)

    db.commit()

    db.refresh(db_expense)

    return db_expense


@app.post("/budgets/", response_model=BudgetCreate)

def create_budget(budget: BudgetCreate, db: Session = Depends(get_db)):

    db_budget = Budget(**budget.dict(), user_id=1)  # Mock user_id

    db.add(db_budget)

    db.commit()

    db.refresh(db_budget)

    return db_budget


client = TestClient(app)
```

```
def test_register():
    response = client.post("/register", json={"username": "testuser", "password": "testpass"})
    assert response.status_code == 200
    data = response.json()
    if data:
        assert data["username"] == "testuser"


def test_create_income():
    response = client.post("/income/", json={"amount": 1000.0, "source": "Salary", "date": "2023-10-01"})
    assert response.status_code == 200
    data = response.json()
    if data:
        assert data["amount"] == 1000.0


def test_create_expense():
    response = client.post("/expenses/", json={"amount": 200.0, "category": "Food", "date": "2023-10-02"})
    assert response.status_code == 200
    data = response.json()
    if data:
        assert data["amount"] == 200.0


def test_create_budget():
    response = client.post("/budgets/", json={"category": "Food", "limit": 300.0})

    assert response.status_code == 200

    data = response.json()

    if data:

        assert data["limit"] == 300.0


test_register()

test_create_income()

test_create_expense()

test_create_budget()
```