Source Code: AI Generated Report

Generated on: 2026-02-07 13:50:33

```
from fastapi import FastAPI, HTTPException, Depends
from fastapi.security import OAuth2PasswordBearer, OAuth2PasswordRequestForm
from sqlalchemy import create_engine, Column, Integer, String, Float, DateTime
from sqlalchemy.ext.declarative import declarative_base
from sqlalchemy.orm import sessionmaker, Session
from pydantic import BaseModel
from datetime import datetime
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)
Base = declarative_base()

app = FastAPI()
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)
    user_id = Column(Integer)
    amount = Column(Float)
    description = Column(String)
    created_at = Column(DateTime, default=datetime.utcnow)

class Budget(Base):
    __tablename__ = "budgets"
    id = Column(Integer, primary_key=True, index=True)
    user_id = Column(Integer)
    amount = Column(Float)
    category = Column(String)

Base.metadata.create_all(bind=engine)

class TransactionCreate(BaseModel):
    user_id: int
    amount: float
    description: str

class BudgetCreate(BaseModel):
    user_id: int
    amount: float
```

```
    category: str


@app.post("/transactions/")
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/")
def   read_transactions(skip:   int   =   0,   limit:   int   =   10,   db:   Session   =
Depends(SessionLocal)):
    transactions = db.query(Transaction).offset(skip).limit(limit).all()
    return transactions


@app.put("/transactions/{transaction_id}")
def update_transaction(transaction_id: int, transaction: TransactionCreate, db: Session
= Depends(SessionLocal)):
                db_transaction    =    db.query(Transaction).filter(Transaction.id    ==
transaction_id).first()
    if db_transaction is None:
        raise HTTPException(status_code=404, detail="Transaction not found")
    for key, value in transaction.dict().items():
        setattr(db_transaction, key, value)
```

```
    db.commit()
    return db_transaction

@app.delete("/transactions/{transaction_id}")
def delete_transaction(transaction_id: int, db: Session = Depends(SessionLocal)):
                db_transaction    =    db.query(Transaction).filter(Transaction.id    ==
transaction_id).first()
    if db_transaction is None:
        raise HTTPException(status_code=404, detail="Transaction not found")
    db.delete(db_transaction)
    db.commit()
    return {"detail": "Transaction deleted"}

@app.post("/budgets/")
def create_budget(budget: BudgetCreate, db: Session = Depends(SessionLocal)):
    db_budget = Budget(**budget.dict())
    db.add(db_budget)
    db.commit()
    db.refresh(db_budget)
    return db_budget

@app.get("/budgets/")
def read_budgets(user_id: int, db: Session = Depends(SessionLocal)):
    budgets = db.query(Budget).filter(Budget.user_id == user_id).all()
    return budgets
```

```
@app.put("/budgets/{budget_id}")
def update_budget(budget_id: int, budget: BudgetCreate, db: Session = Depends(SessionLocal)):
    db_budget = db.query(Budget).filter(Budget.id == budget_id).first()
    if db_budget is None:
        raise HTTPException(status_code=404, detail="Budget not found")
    for key, value in budget.dict().items():
        setattr(db_budget, key, value)
    db.commit()
    return db_budget

@app.delete("/budgets/{budget_id}")
def delete_budget(budget_id: int, db: Session = Depends(SessionLocal)):
    db_budget = db.query(Budget).filter(Budget.id == budget_id).first()
    if db_budget is None:
        raise HTTPException(status_code=404, detail="Budget not found")
    db.delete(db_budget)
    db.commit()
    return {"detail": "Budget deleted"}

# Mock data for testing
from fastapi.testclient import TestClient

client = TestClient(app)

def test_app():
```

```
    # Create a transaction
        response = client.post("/transactions/", json={"user_id": 1, "amount": 100.0,
"description": "Salary"})
    print("Create Transaction Response:", response.json())

    # Read transactions
    response = client.get("/transactions/")
    print("Read Transactions Response:", response.json())

    # Update a transaction
    transaction_id = response.json()[0]['id']
        response = client.put(f"/transactions/{transaction_id}", json={"user_id": 1,
"amount": 150.0, "description": "Updated Salary"})
    print("Update Transaction Response:", response.json())

    # Delete a transaction
    response = client.delete(f"/transactions/{transaction_id}")
    print("Delete Transaction Response:", response.json())

    # Create a budget
    response = client.post("/budgets/", json={"user_id": 1, "amount": 500.0, "category":
"Monthly Expenses"})
    print("Create Budget Response:", response.json())

    # Read budgets
    response = client.get("/budgets/?user_id=1")
```

```
    print("Read Budgets Response:", response.json())


    # Update a budget
    budget_id = response.json()[0]['id']
     response = client.put(f"/budgets/{budget_id}", json={"user_id": 1, "amount": 600.0,
"category": "Updated Monthly Expenses"})
    print("Update Budget Response:", response.json())


    # Delete a budget
    response = client.delete(f"/budgets/{budget_id}")
    print("Delete Budget Response:", response.json())


test_app()
```