**THỰC HÀNH BUỔI 5**

# Mục tiêu

* Hiểu khái niệm cơ sở dữ liệu quan hệ (SQL) và ORM
* Thay thế Mock Data bằng SQLAlchemy models và PostgreSQL/SQLite
* Thiết lập database connection và session management
* Tạo và chạy migrations với Alembic
* Implement repository pattern với database operations
* Maintain tất cả API functionality với persistent data
* Hiểu quan hệ One-to-Many và Foreign Keys trong thực tế

# Kiến thức & Yêu cầu chuẩn bị

* Hoàn thành Buổi 4: FastAPI cơ bản và CRUD in-memory
* Python 3.8+ và virtual environment đã kích hoạt
* Cài đặt thêm các thư viện

pip install sqlalchemy alembic psycopg2-binary

* Có sẵn PostgreSQL hoặc SQLite local để kết nối

# Lý thuyết

## Tại sao cần Database?

Vấn đề của In-memory storage:

* Dữ liệu mất khi server restart
* Không thể scale với lượng data lớn
* Không có backup và recovery
* Không hỗ trợ concurrent access

Lợi ích của Database:

* Persistent storage: Dữ liệu bền vững
* ACID properties: Atomicity, Consistency, Isolation, Durability
* Query capabilities: Complex filtering, sorting, aggregation
* Concurrent access: Nhiều user cùng truy cập
* Backup và recovery: Bảo vệ dữ liệu

## SQLite Overview

Đặc điểm SQLite:

* Serverless: Không cần database server riêng
* File-based: Toàn bộ DB trong 1 file .db
* Zero-configuration: Không cần setup phức tạp
* Cross-platform: Chạy trên mọi hệ điều hành
* ACID compliant: Đảm bảo tính toàn vẹn dữ liệu

Khi nào dùng SQLite:

* Development và testing
* Small to medium applications (< 100GB data)
* Applications với concurrent reads, limited writes
* Embedded applications
* Prototyping và proof-of-concept

## ORM (Object-Relational Mapping)

ORM là gì:

* Kỹ thuật mapping giữa relational database và object-oriented programming
* Cho phép thao tác database bằng objects thay vì SQL
* Tự động convert giữa database records và Python objects

Kỹ thuật ánh xạ:

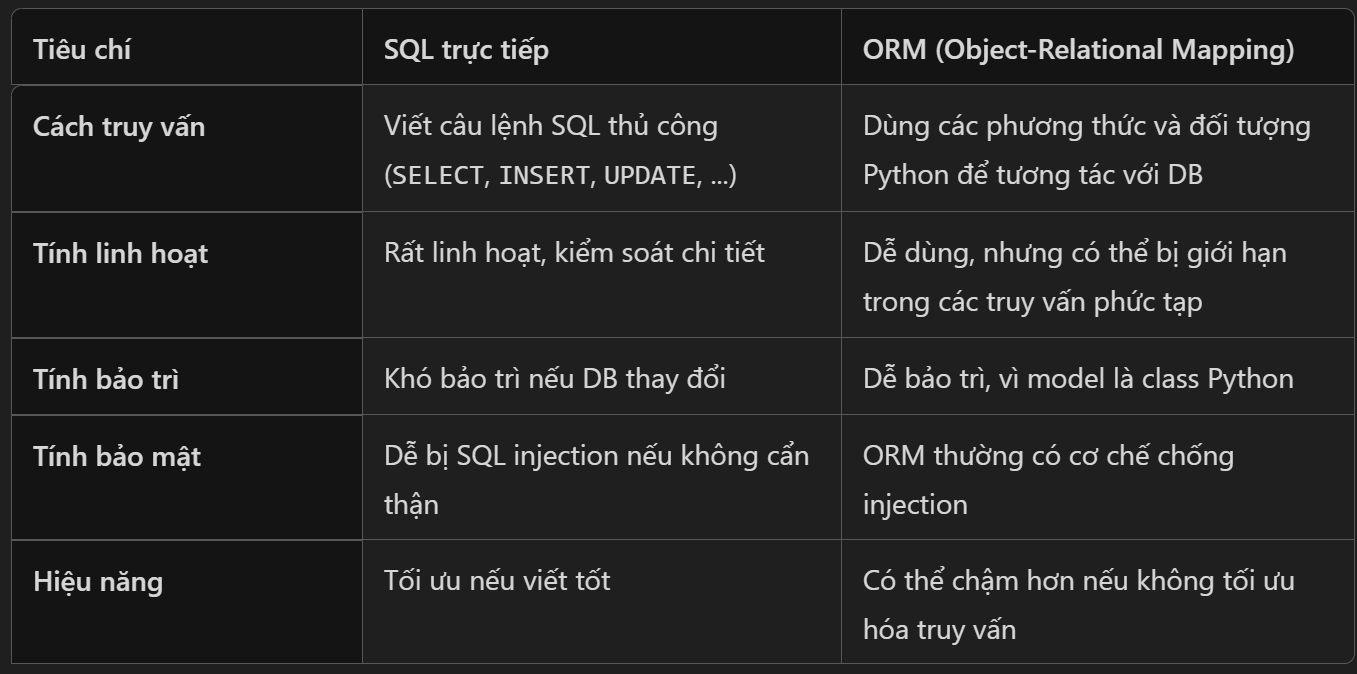
* + **Object** (Python classes) ↔ **Table** (Database tables)
  + **Instance** (Python objects) ↔ **Row** (Database records)
  + **Attribute** (Python properties) ↔ **Column** (Database fields)

Ưu điểm của ORM:

* Abstraction: Không cần viết SQL thuần
* Type safety: Python type hints và validation
* Database agnostic: Dễ chuyển đổi giữa các DB
* Security: Tự động prevent SQL injection
* Productivity: Code nhanh hơn, maintain dễ hơn

Nhược điểm của ORM:

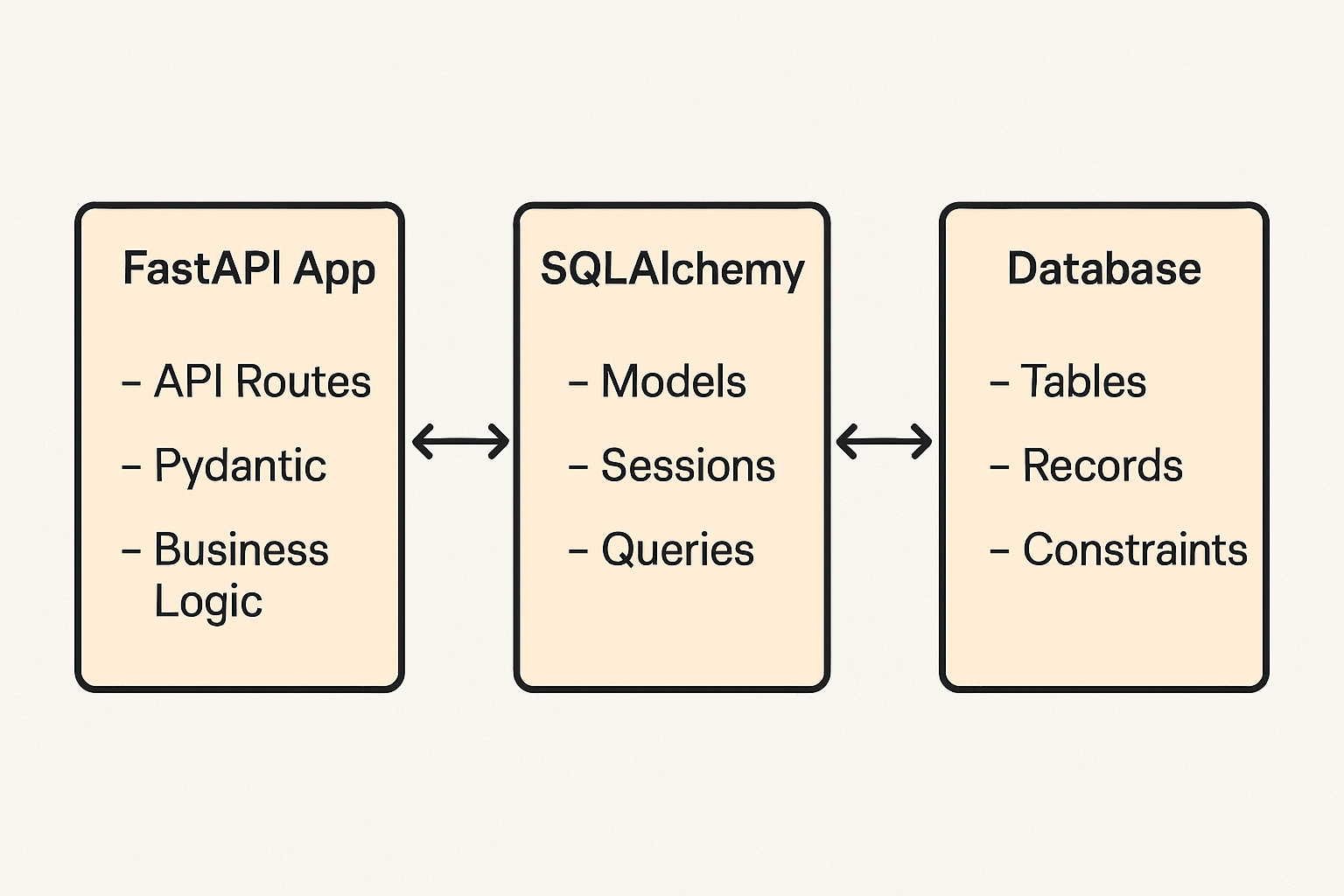
* Performance overhead: Slower than raw SQL
* Learning curve: Cần học ORM syntax
* Complex queries: Khó optimize với queries phức tạp



## SQLAlchemy Architecture

**SQLAlchemy** là một **ORM framework** và **SQL toolkit** cho Python, giúp bạn:

* Kết nối và tương tác với nhiều loại cơ sở dữ liệu như PostgreSQL, MySQL, SQLite, Oracle,...
* Viết truy vấn SQL một cách linh hoạt.
* Quản lý dữ liệu bằng cách ánh xạ các bảng trong database thành các class Python.



## Migration với Alembic

**Migration** là quá trình quản lý thay đổi database schema:

* **Version Control**: Tracking database schema changes
* **Reversible**: Có thể rollback về version cũ
* **Team Collaboration**: Đồng bộ schema giữa developers
* **Production Deployment**: Apply changes safely

# Hướng dẫn thực hành

**A diagram of a program

AI-generated content may be incorrect.**

**Bước 1:** Kết nối Ubuntu qua SSH trên VSCode (Xem file Thực hành buổi 1)

## Mở Terminal, Kích hoạt môi trường ảo

cd workspace/todo\_project

source venv/bin/activate

Cấu trúc database module

app/  
├── database/  
│ ├── \_\_init\_\_.py  
│ ├── connection.py # Database connection setup  
│ ├── models.py # SQLAlchemy models  
│ └── repository.py # Database operations (CRUD)  
├── migrations/ # Alembic migrations (sẽ tạo tự động)  
└── alembic.ini # Alembic configuration

**Bước 2.1. Cài đặt PostgreSQL**

sudo apt update

sudo apt install postgresql postgresql-contrib

**2.2. Kiểm tra trạng thái dịch vụ PostgreSQL**

sudo systemctl status postgresql

(Dịch vụ cần ở trạng thái active (running).)

**2.3. Tạo user, database cho project**

sudo -i -u postgres

#Kết nối vào postgres db

gõ: psql

**Tại psql shell, chạy:**

CREATE USER kanban\_user WITH PASSWORD 'yourpassword';

CREATE DATABASE kanban\_db OWNER kanban\_user;

GRANT ALL PRIVILEGES ON DATABASE kanban\_db TO kanban\_user;

\q

Gõ: exit

**2.4. Cài driver cho Python**

pip install psycopg2-binary

**2.5. Chuỗi kết nối SQLAlchemy**

**Thêm vào .env hoặc config:**

DATABASE\_URL=postgresql+psycopg2://kanban\_user:yourpassword@localhost:5432/kanban\_db

**Bước 3: Database Connection Setup**

**File: app/database/connection.py**

from sqlalchemy import create\_engine

from sqlalchemy.orm import sessionmaker, declarative\_base

from app.core.config import settings

print(settings.database\_url)

engine = create\_engine(

    settings.database\_url,

    echo=settings.database\_echo,

)

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

Base = declarative\_base()

def get\_db():

    db = SessionLocal()

    try:

        yield db

    finally:

        db.close()

def create\_tables():

    Base.metadata.create\_all(bind=engine)

**Environment Configuration**

**File: app/core/config.py** (tạo mới)

from pydantic\_settings import BaseSettings

from typing import Optional

class Settings(BaseSettings):

    database\_url: str

    database\_echo: bool = False

    # Application

    app\_name: str = "Kanban TODO API"

    debug: bool = True

    class Config:

        env\_file = ".env"

settings = Settings()

**Bước 4: Tạo SQLAlchemy Models**

**1. Database Models**

**File: app/database/models.py**

from sqlalchemy import Column, Integer, String, DateTime, ForeignKey, Boolean, Enum as SQLEnum

from sqlalchemy.orm import relationship

from datetime import datetime

from enum import Enum

from .connection import Base

# Enum classes

class StatusEnum(str, Enum):

    todo = "todo"

    in\_progress = "in\_progress"

    done = "done"

class PriorityEnum(str, Enum):

    low = "low"

    medium = "medium"

    high = "high"

#Class User theo phân tích buổi 3

class User(Base):

    \_\_tablename\_\_ = "users"

    id = Column(Integer, primary\_key=True, index=True)

    username = Column(String(50), unique=True, nullable=False, index=True)

    email = Column(String(100), unique=True, nullable=True)

    password = Column(String(255), nullable=False) #Sẽ hash ở buổi JWT

    full\_name = Column(String(100), nullable=True)

    is\_active = Column(Boolean, default=True, nullable=False)

    created\_at = Column(DateTime, default=datetime.utcnow, nullable=False)

    updated\_at = Column(DateTime, default=datetime.utcnow, onupdate=datetime.utcnow, nullable=False)

# Relationships

    boards = relationship("Board", back\_populates="owner", cascade="all, delete-orphan")

    assigned\_tasks = relationship("Task", back\_populates="assigned\_user")

#Class Board theo phân tích buổi 3

class Board(Base):

    \_\_tablename\_\_ = "boards"

    id = Column(Integer, primary\_key=True, index=True)

    name = Column(String(100), nullable=False)

    description = Column(String(500), nullable=True)

    is\_public = Column(Boolean, default=False, nullable=False)

    owner\_id = Column(Integer, ForeignKey("users.id"), nullable=False)

    created\_at = Column(DateTime, default=datetime.utcnow, nullable=False)

    updated\_at = Column(DateTime, default=datetime.utcnow, onupdate=datetime.utcnow, nullable=False)

    # Relationships

    owner = relationship("User", back\_populates="boards")

    tasks = relationship("Task", back\_populates="board", cascade="all, delete-orphan")

#Class Task theo phân tích buổi 3

class Task(Base):

    \_\_tablename\_\_ = "tasks"

    id = Column(Integer, primary\_key=True, index=True)

    title = Column(String(200), nullable=False)

    description = Column(String(1000), nullable=True)

    status = Column(SQLEnum(StatusEnum), default=StatusEnum.todo, nullable=False)

    priority = Column(SQLEnum(PriorityEnum), default=PriorityEnum.medium, nullable=False)

    position = Column(Integer, default=0, nullable=False)

    board\_id = Column(Integer, ForeignKey("boards.id"), nullable=False)

    assigned\_to = Column(Integer, ForeignKey("users.id"), nullable=True)

    due\_date = Column(DateTime, nullable=True)

    created\_at = Column(DateTime, default=datetime.utcnow, nullable=False)

    updated\_at = Column(DateTime, default=datetime.utcnow, onupdate=datetime.utcnow, nullable=False)

    # Relationships

    board = relationship("Board", back\_populates="tasks")

    assigned\_user = relationship("User", back\_populates="assigned\_tasks")

    def \_\_repr\_\_(self):

        return f"<Task(id={self.id}, title='{self.title}', status='{self.status}')>"

**Bước 5: Import models trong init.py**

**File: app/database/\_\_init\_\_.py**

from .connection import Base, engine, get\_db, create\_tables  
from .models import User, Board, Task, StatusEnum, PriorityEnum  
from .repository import UserRepository, BoardRepository, TaskRepository  
  
\_\_all\_\_ = [  
 "Base", "engine", "get\_db", "create\_tables",  
 "User", "Board", "Task", "StatusEnum", "PriorityEnum",   
 "UserRepository", "BoardRepository", "TaskRepository"  
]

**Bước 6: Setup Alembic Migrations**

**6.1 Khởi tạo Alembic**

# Từ thư mục root project

**6.2 Cấu hình Alembic**

**File: alembic.ini** (edit file existing)

[alembic]

interpolation = none

sqlalchemy.url =

script\_location = migrations

**File: migrations/env.py** (edit existing)

import sys

import os

from logging.config import fileConfig

from sqlalchemy import engine\_from\_config, pool

from alembic import context

# Add project root to sys.path

sys.path.append(os.path.abspath(os.path.join(os.path.dirname(\_\_file\_\_), '..')))

from app.core.config import settings

from app.database.connection import Base

config = context.config

fileConfig(config.config\_file\_name)

section = config.config\_ini\_section

# Override sqlalchemy.url in alembic.ini

url = settings.database\_url.replace('%', '%%')  # Escape % for ConfigParser

config.set\_main\_option("sqlalchemy.url", url)

target\_metadata = Base.metadata

def run\_migrations\_offline():

    url = config.get\_section(config.config\_ini\_section)['sqlalchemy.url']

    context.configure(

        url=url,

        target\_metadata=target\_metadata,

        literal\_binds=True,

        dialect\_opts={"paramstyle": "named"},

    )

    with context.begin\_transaction():

        context.run\_migrations()

def run\_migrations\_online():

    connectable = engine\_from\_config(

        config.get\_section(config.config\_ini\_section),

        prefix='sqlalchemy.',

        poolclass=pool.NullPool,

    )

    with connectable.connect() as connection:

        context.configure(

            connection=connection,

            target\_metadata=target\_metadata,

        )

        with context.begin\_transaction():

            context.run\_migrations()

if context.is\_offline\_mode():

    run\_migrations\_offline()

else:

    run\_migrations\_online()

**Bước 6: Repository Pattern - Database Operations**

**6.1 Base Repository**

**File: app/database/repository.py**

from sqlalchemy.orm import Session  
from typing import List, Optional, Generic, TypeVar, Type  
from .models import User, Board, Task, StatusEnum, PriorityEnum  
  
# Generic types  
ModelType = TypeVar("ModelType")  
CreateSchemaType = TypeVar("CreateSchemaType")  
UpdateSchemaType = TypeVar("UpdateSchemaType")  
  
class BaseRepository(Generic[ModelType, CreateSchemaType, UpdateSchemaType]):  
 def \_\_init\_\_(self, model: Type[ModelType]):  
 self.model = model  
   
 def get(self, db: Session, id: int) -> Optional[ModelType]:  
 return db.query(self.model).filter(self.model.id == id).first()  
   
 def get\_multi(self, db: Session, \*, skip: int = 0, limit: int = 100) -> List[ModelType]:  
 return db.query(self.model).offset(skip).limit(limit).all()  
   
 def create(self, db: Session, \*, obj\_in: CreateSchemaType) -> ModelType:  
 obj\_data = obj\_in.dict() if hasattr(obj\_in, 'dict') else obj\_in  
 db\_obj = self.model(\*\*obj\_data)  
 db.add(db\_obj)  
 db.commit()  
 db.refresh(db\_obj)  
 return db\_obj  
   
 def update(self, db: Session, \*, db\_obj: ModelType, obj\_in: UpdateSchemaType) -> ModelType:  
 obj\_data = obj\_in.dict(exclude\_unset=True) if hasattr(obj\_in, 'dict') else obj\_in  
 for field, value in obj\_data.items():  
 setattr(db\_obj, field, value)  
 db.commit()  
 db.refresh(db\_obj)  
 return db\_obj  
   
 def delete(self, db: Session, \*, id: int) -> ModelType:  
 obj = db.query(self.model).get(id)  
 db.delete(obj)  
 db.commit()  
 return obj

# Tạo User repository  
class UserRepository(BaseRepository[User, dict, dict]):  
 def \_\_init\_\_(self):  
 super().\_\_init\_\_(User)  
   
 def get\_by\_username(self, db: Session, username: str) -> Optional[User]:  
 return db.query(User).filter(User.username == username).first()  
   
 def get\_by\_email(self, db: Session, email: str) -> Optional[User]:  
 return db.query(User).filter(User.email == email).first()  
   
 def create\_user(self, db: Session, user\_data: dict) -> User:  
 # Trong thực tế sẽ hash password ở đây  
 db\_user = User(\*\*user\_data)  
 db.add(db\_user)  
 db.commit()  
 db.refresh(db\_user)  
 return db\_user  
# Tạo Board repository  
class BoardRepository(BaseRepository[Board, dict, dict]):  
 def \_\_init\_\_(self):  
 super().\_\_init\_\_(Board)  
   
 def get\_by\_owner(self, db: Session, owner\_id: int) -> List[Board]:  
 return db.query(Board).filter(Board.owner\_id == owner\_id).all()  
   
 def get\_public\_boards(self, db: Session) -> List[Board]:  
 return db.query(Board).filter(Board.is\_public == True).all()

# Tạo Task repository  
class TaskRepository(BaseRepository[Task, dict, dict]):  
 def \_\_init\_\_(self):  
 super().\_\_init\_\_(Task)  
   
 def get\_by\_board(self, db: Session, board\_id: int) -> List[Task]:  
 return db.query(Task).filter(Task.board\_id == board\_id).order\_by(Task.position).all()  
   
 def get\_by\_status(self, db: Session, board\_id: int, status: StatusEnum) -> List[Task]:  
 return db.query(Task).filter(  
 Task.board\_id == board\_id,  
 Task.status == status  
 ).order\_by(Task.position).all()  
   
 def get\_by\_assigned\_user(self, db: Session, user\_id: int) -> List[Task]:  
 return db.query(Task).filter(Task.assigned\_to == user\_id).all()  
   
 def search\_tasks(self, db: Session, query: str, board\_id: Optional[int] = None) -> List[Task]:  
 search\_query = db.query(Task).filter(  
 Task.title.contains(query) | Task.description.contains(query)  
 )  
   
 if board\_id:  
 search\_query = search\_query.filter(Task.board\_id == board\_id)  
   
 return search\_query.all()  
   
 def move\_task(self, db: Session, task\_id: int, new\_status: StatusEnum, new\_position: Optional[int] = None) -> Optional[Task]:  
 task = self.get(db, task\_id)  
 if not task:  
 return None  
   
 old\_status = task.status  
 task.status = new\_status  
   
# Tính position mới nếu không được specify  
 if new\_status != old\_status and new\_position is None:  
 tasks\_in\_new\_status = self.get\_by\_status(db, task.board\_id, new\_status)  
 new\_position = len(tasks\_in\_new\_status)  
   
 if new\_position is not None:  
 task.position = new\_position  
   
 db.commit()  
 db.refresh(task)  
 return task  
  
# Global instances  
user\_repository = UserRepository()  
board\_repository = BoardRepository()  
task\_repository = TaskRepository()

**Bước 7: Tạo và chạy migration đầu tiên**

# Tạo migration file tự động  
alembic revision --autogenerate -m "Create users, boards, and tasks tables"  
  
# Apply migration  
alembic upgrade head  
  
# Kiểm tra current migration  
alembic current  
  
# Xem history  
alembic history

**Bước 8: Update API Routers để sử dụng Database**

**8.1 Update User Router**

# User Management

POST /auth/register # Đăng ký

POST /auth/login # Đăng nhập

GET /users/me # Xem profile

PUT /users/me # Cập nhật profile

**File: app/routers/users.py** (replace existing)

from fastapi import APIRouter, HTTPException, status, Depends  
from sqlalchemy.orm import Session  
from typing import List  
  
from app.schemas.user import UserCreate, UserResponse, UserUpdate, PasswordChange  
from app.database import get\_db, user\_repository  
  
router = APIRouter(prefix="/users", tags=["users"])

# API users register  
@router.post("/", response\_model=UserResponse, status\_code=status.HTTP\_201\_CREATED)  
def create\_user(user\_data: UserCreate, db: Session = Depends(get\_db)):  
 """Tạo user mới"""  
 # Kiểm tra username đã tồn tại  
 if user\_repository.get\_by\_username(db, user\_data.username):  
 raise HTTPException(  
 status\_code=status.HTTP\_409\_CONFLICT,  
 detail="Username đã tồn tại"  
 )  
   
 # Kiểm tra email đã tồn tại (nếu có)  
 if user\_data.email and user\_repository.get\_by\_email(db, user\_data.email):  
 raise HTTPException(  
 status\_code=status.HTTP\_409\_CONFLICT,  
 detail="Email đã tồn tại"  
 )  
   
 user\_dict = user\_data.dict()  
 # Tạm thời lưu password plain text (sẽ hash ở buổi JWT)  
 db\_user = user\_repository.create\_user(db, user\_dict)  
 return UserResponse.from\_orm(db\_user)

@router.get("/", response\_model=List[UserResponse])  
def get\_users(skip: int = 0, limit: int = 100, db: Session = Depends(get\_db)):  
 """Lấy danh sách users"""  
 users = user\_repository.get\_multi(db, skip=skip, limit=limit)  
 return [UserResponse.from\_orm(user) for user in users]  
  
@router.get("/{user\_id}", response\_model=UserResponse)  
def get\_user(user\_id: int, db: Session = Depends(get\_db)):  
 """Lấy user theo ID"""  
 user = user\_repository.get(db, user\_id)  
 if not user:  
 raise HTTPException(  
 status\_code=status.HTTP\_404\_NOT\_FOUND,  
 detail="User không tồn tại"  
 )  
 return UserResponse.from\_orm(user)  
  
@router.put("/{user\_id}", response\_model=UserResponse)  
def update\_user(user\_id: int, user\_update: UserUpdate, db: Session = Depends(get\_db)):  
 """Cập nhật user"""  
 user = user\_repository.get(db, user\_id)  
 if not user:  
 raise HTTPException(  
 status\_code=status.HTTP\_404\_NOT\_FOUND,  
 detail="User không tồn tại"  
 )  
   
 # Kiểm tra email conflict nếu update email  
 if user\_update.email and user\_update.email != user.email:  
 existing\_user = user\_repository.get\_by\_email(db, user\_update.email)  
 if existing\_user and existing\_user.id != user\_id:  
 raise HTTPException(  
 status\_code=status.HTTP\_409\_CONFLICT,  
 detail="Email đã được sử dụng"  
 )  
   
 updated\_user = user\_repository.update(db, db\_obj=user, obj\_in=user\_update)  
 return UserResponse.from\_orm(updated\_user)  
  
@router.patch("/{user\_id}/password")  
def change\_password(user\_id: int, password\_change: PasswordChange, db: Session = Depends(get\_db)):  
 """Đổi mật khẩu user"""  
 user = user\_repository.get(db, user\_id)  
 if not user:  
 raise HTTPException(  
 status\_code=status.HTTP\_404\_NOT\_FOUND,  
 detail="User không tồn tại"  
 )  
   
 # Kiểm tra password hiện tại (tạm thời plain text)  
 if user.password != password\_change.current\_password:  
 raise HTTPException(  
 status\_code=status.HTTP\_400\_BAD\_REQUEST,  
 detail="Mật khẩu hiện tại không đúng"  
 )  
   
 # Cập nhật password mới  
 user\_repository.update(db, db\_obj=user, obj\_in={"password": password\_change.new\_password})  
 return {"message": "Đổi mật khẩu thành công"}

**9.2 Update Board Router**

# Board Management

GET /boards # Lấy danh sách boards

POST /boards # Tạo board mới

GET /boards/{id} # Lấy chi tiết board

PUT /boards/{id} # Cập nhật board

DELETE /boards/{id} # Xóa board

**File: app/routers/boards.py** (replace existing)

from fastapi import APIRouter, HTTPException, status, Query, Depends  
from sqlalchemy.orm import Session  
from typing import List, Optional  
  
from app.schemas.board import BoardCreate, BoardResponse, BoardUpdate, BoardWithTasks  
from app.schemas.task import TaskResponse  
from app.database import get\_db, board\_repository, task\_repository, user\_repository  
  
router = APIRouter(prefix="/boards", tags=["boards"])  
  
@router.get("/", response\_model=List[BoardResponse])  
def get\_boards(  
 owner\_id: Optional[int] = Query(None, description="Filter theo owner"),  
 skip: int = Query(0, ge=0),  
 limit: int = Query(100, ge=1, le=1000),  
 db: Session = Depends(get\_db)  
):  
 """Lấy danh sách boards"""  
 if owner\_id:  
 boards = board\_repository.get\_by\_owner(db, owner\_id)  
 else:  
 boards = board\_repository.get\_multi(db, skip=skip, limit=limit)  
   
 # Thêm tasks\_count  
 response\_boards = []  
 for board in boards:  
 board\_tasks = task\_repository.get\_by\_board(db, board.id)  
 board\_response = BoardResponse.from\_orm(board)  
 board\_response.tasks\_count = len(board\_tasks)  
 response\_boards.append(board\_response)  
   
 return response\_boards  
  
@router.post("/", response\_model=BoardResponse, status\_code=status.HTTP\_201\_CREATED)  
def create\_board(  
 board\_data: BoardCreate,  
 owner\_id: int = Query(..., description="ID của user tạo board"),  
 db: Session = Depends(get\_db)  
):  
 """Tạo board mới"""  
 # Kiểm tra user có tồn tại không  
 owner = user\_repository.get(db, owner\_id)  
 if not owner:  
 raise HTTPException(  
 status\_code=status.HTTP\_400\_BAD\_REQUEST,  
 detail="User không tồn tại"  
 )  
   
 board\_dict = board\_data.dict()  
 board\_dict["owner\_id"] = owner\_id  
   
 board = board\_repository.create(db, obj\_in=board\_dict)  
 board\_response = BoardResponse.from\_orm(board)  
 board\_response.tasks\_count = 0  
 return board\_response  
  
@router.get("/{board\_id}", response\_model=BoardWithTasks)  
def get\_board\_detail(board\_id: int, db: Session = Depends(get\_db)):  
 """Lấy chi tiết board kèm tasks"""  
 board = board\_repository.get(db, board\_id)  
 if not board:  
 raise HTTPException(  
 status\_code=status.HTTP\_404\_NOT\_FOUND,  
 detail="Board không tồn tại"  
 )  
   
 tasks = task\_repository.get\_by\_board(db, board\_id)  
 task\_responses = [TaskResponse.from\_orm(task) for task in tasks]  
   
 board\_response = BoardWithTasks.from\_orm(board)  
 board\_response.tasks = task\_responses  
 return board\_response  
  
@router.put("/{board\_id}", response\_model=BoardResponse)  
def update\_board(board\_id: int, board\_update: BoardUpdate, db: Session = Depends(get\_db)):  
 """Cập nhật board"""  
 board = board\_repository.get(db, board\_id)  
 if not board:  
 raise HTTPException(  
 status\_code=status.HTTP\_404\_NOT\_FOUND,  
 detail="Board không tồn tại"  
 )  
   
 updated\_board = board\_repository.update(db, db\_obj=board, obj\_in=board\_update)  
   
 # Thêm tasks\_count  
 tasks = task\_repository.get\_by\_board(db, board\_id)  
 board\_response = BoardResponse.from\_orm(updated\_board)  
 board\_response.tasks\_count = len(tasks)  
 return board\_response  
  
@router.delete("/{board\_id}")  
def delete\_board(board\_id: int, db: Session = Depends(get\_db)):  
 """Xóa board và tất cả tasks"""  
 board = board\_repository.get(db, board\_id)  
 if not board:  
 raise HTTPException(  
 status\_code=status.HTTP\_404\_NOT\_FOUND,  
 detail="Board không tồn tại"  
 )  
   
 # Đếm tasks sẽ bị xóa  
 tasks = task\_repository.get\_by\_board(db, board\_id)  
 deleted\_tasks\_count = len(tasks)  
   
 # Xóa board (cascade sẽ tự động xóa tasks)  
 board\_repository.delete(db, id=board\_id)  
   
 return {  
 "message": "Xóa board thành công",  
 "deleted\_tasks\_count": deleted\_tasks\_count  
 }

**9.3 Update Task Router**

# Task Management

GET /tasks?board\_id={id} # Lấy tasks theo board

POST /tasks # Tạo task mới

GET /tasks/{id} # Lấy chi tiết task

PUT /tasks/{id} # Cập nhật task

DELETE /tasks/{id} # Xóa task

PATCH /tasks/{id}/move # Di chuyển task (status/position)

**File: app/routers/tasks.py** (replace existing)

from fastapi import APIRouter, HTTPException, status, Query, Depends  
from sqlalchemy.orm import Session  
from typing import List, Optional  
  
from app.schemas.task import TaskCreate, TaskResponse, TaskUpdate, TaskMove, TaskAssign  
from app.database import get\_db, task\_repository, board\_repository, user\_repository  
from app.database.models import StatusEnum  
  
router = APIRouter(prefix="/tasks", tags=["tasks"])  
  
@router.get("/", response\_model=List[TaskResponse])  
def get\_tasks(  
 board\_id: int = Query(..., description="ID của board"),  
 status: Optional[str] = Query(None, description="Filter theo status"),  
 priority: Optional[str] = Query(None, description="Filter theo priority"),  
 assigned\_to: Optional[int] = Query(None, description="Filter theo assigned user"),  
 db: Session = Depends(get\_db)  
):  
 """Lấy tasks với filters"""  
 # Kiểm tra board tồn tại  
 board = board\_repository.get(db, board\_id)  
 if not board:  
 raise HTTPException(  
 status\_code=status.HTTP\_404\_NOT\_FOUND,  
 detail="Board không tồn tại"  
 )  
   
 # Get tasks với filters  
 if status:  
 try:  
 status\_enum = StatusEnum(status)  
 tasks = task\_repository.get\_by\_status(db, board\_id, status\_enum)  
 except ValueError:  
 raise HTTPException(  
 status\_code=status.HTTP\_400\_BAD\_REQUEST,  
 detail=f"Status không hợp lệ: {status}"  
 )  
 else:  
 tasks = task\_repository.get\_by\_board(db, board\_id)  
   
 # Apply additional filters  
 if priority:  
 tasks = [task for task in tasks if task.priority.value == priority]  
   
 if assigned\_to is not None:  
 tasks = [task for task in tasks if task.assigned\_to == assigned\_to]  
   
 return [TaskResponse.from\_orm(task) for task in tasks]  
  
@router.post("/", response\_model=TaskResponse, status\_code=status.HTTP\_201\_CREATED)  
def create\_task(task\_data: TaskCreate, db: Session = Depends(get\_db)):  
 """Tạo task mới"""  
 # Kiểm tra board tồn tại  
 board = board\_repository.get(db, task\_data.board\_id)  
 if not board:  
 raise HTTPException(  
 status\_code=status.HTTP\_400\_BAD\_REQUEST,  
 detail="Board không tồn tại"  
 )  
   
 # Tính position cho task mới  
 existing\_tasks = task\_repository.get\_by\_status(db, task\_data.board\_id, task\_data.status)  
 task\_dict = task\_data.dict()  
 task\_dict["position"] = len(existing\_tasks)  
   
 task = task\_repository.create(db, obj\_in=task\_dict)  
 return TaskResponse.from\_orm(task)  
  
@router.get("/{task\_id}", response\_model=TaskResponse)  
def get\_task(task\_id: int, db: Session = Depends(get\_db)):  
 """Lấy task theo ID"""  
 task = task\_repository.get(db, task\_id)  
 if not task:  
 raise HTTPException(  
 status\_code=status.HTTP\_404\_NOT\_FOUND,  
 detail="Task không tồn tại"  
 )  
 return TaskResponse.from\_orm(task)  
  
@router.put("/{task\_id}", response\_model=TaskResponse)  
def update\_task(task\_id: int, task\_update: TaskUpdate, db: Session = Depends(get\_db)):  
 """Cập nhật task"""  
 task = task\_repository.get(db, task\_id)  
 if not task:  
 raise HTTPException(  
 status\_code=status.HTTP\_404\_NOT\_FOUND,  
 detail="Task không tồn tại"  
 )  
   
 updated\_task = task\_repository.update(db, db\_obj=task, obj\_in=task\_update)  
 return TaskResponse.from\_orm(updated\_task)  
  
@router.patch("/{task\_id}/move", response\_model=TaskResponse)  
def move\_task(task\_id: int, task\_move: TaskMove, db: Session = Depends(get\_db)):  
 """Di chuyển task"""  
 moved\_task = task\_repository.move\_task(db, task\_id, task\_move.status, task\_move.position)  
 if not moved\_task:  
 raise HTTPException(  
 status\_code=status.HTTP\_404\_NOT\_FOUND,  
 detail="Task không tồn tại"  
 )  
 return TaskResponse.from\_orm(moved\_task)  
  
@router.patch("/{task\_id}/assign", response\_model=TaskResponse)  
def assign\_task(task\_id: int, task\_assign: TaskAssign, db: Session = Depends(get\_db)):  
 """Gán task cho user"""  
 task = task\_repository.get(db, task\_id)  
 if not task:  
 raise HTTPException(  
 status\_code=status.HTTP\_404\_NOT\_FOUND,  
 detail="Task không tồn tại"  
 )  
   
 # Kiểm tra user tồn tại nếu có assigned\_to  
 if task\_assign.assigned\_to:  
 user = user\_repository.get(db, task\_assign.assigned\_to)  
 if not user:  
 raise HTTPException(  
 status\_code=status.HTTP\_400\_BAD\_REQUEST,  
 detail="User được gán không tồn tại"  
 )  
   
 updated\_task = task\_repository.update(db, db\_obj=task, obj\_in={"assigned\_to": task\_assign.assigned\_to})  
 return TaskResponse.from\_orm(updated\_task)  
  
@router.delete("/{task\_id}")  
def delete\_task(task\_id: int, db: Session = Depends(get\_db)):  
 """Xóa task"""  
 task = task\_repository.get(db, task\_id)  
 if not task:  
 raise HTTPException(  
 status\_code=status.HTTP\_404\_NOT\_FOUND,  
 detail="Task không tồn tại"  
 )  
   
 task\_repository.delete(db, id=task\_id)  
 return {  
 "message": "Xóa task thành công",  
 "deleted\_task\_id": task\_id  
 }  
  
@router.get("/search/", response\_model=List[TaskResponse])  
def search\_tasks(  
 q: str = Query(..., min\_length=1, description="Từ khóa tìm kiếm"),  
 board\_id: Optional[int] = Query(None, description="Tìm trong board cụ thể"),  
 db: Session = Depends(get\_db)  
):  
 """Tìm kiếm tasks"""  
 if board\_id:  
 board = board\_repository.get(db, board\_id)  
 if not board:  
 raise HTTPException(  
 status\_code=status.HTTP\_404\_NOT\_FOUND,  
 detail="Board không tồn tại"  
 )  
   
 tasks = task\_repository.search\_tasks(db, q, board\_id)  
 return [TaskResponse.from\_orm(task) for task in tasks]

**Bước 9: Update Main Application**

**9.1 Update main.py**

**File: app/main.py** (replace existing)

from fastapi import FastAPI  
from fastapi.middleware.cors import CORSMiddleware  
  
from app.routers import users, boards, tasks  
from app.database import create\_tables  
from app.core.config import settings  
  
# Tạo tables khi khởi động (development only)  
create\_tables()  
  
# Tạo FastAPI app  
app = FastAPI(  
 title=settings.app\_name,  
 version="1.0.0",  
 description="Kanban TODO API với SQLAlchemy database integration",  
 docs\_url="/docs",  
 redoc\_url="/redoc"  
)  
  
# CORS middleware  
app.add\_middleware(  
 CORSMiddleware,  
 allow\_origins=["\*"],  
 allow\_credentials=True,  
 allow\_methods=["\*"],  
 allow\_headers=["\*"]  
)  
  
# Include routers  
app.include\_router(users.router)  
app.include\_router(boards.router)  
app.include\_router(tasks.router)  
  
@app.get("/")  
def read\_root():  
 return {  
 "message": f"Chào mừng đến với {settings.app\_name}!",  
 "version": "1.0.0",  
 "database": "SQLAlchemy integrated",  
 "docs\_url": "/docs"  
 }  
  
@app.get("/health")  
def health\_check():  
 return {  
 "status": "healthy",  
 "service": settings.app\_name,  
 "database": "connected"  
 }  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 import uvicorn  
 uvicorn.run("app.main:app", host="0.0.0.0", port=8000, reload=True)

**9.2 Update Pydantic Schemas để support SQLAlchemy**

**File: app/schemas/user.py** (add to existing)

# Thêm vào cuối file  
UserResponse.update\_forward\_refs()  
  
# Example để dùng from\_orm  
class UserResponse(UserBase):  
 id: int  
 is\_active: bool  
 created\_at: datetime  
 updated\_at: datetime  
  
 class Config:  
 orm\_mode = True  
 from\_attributes = True # For Pydantic v2

**Bước 10. Data Migration và Seeding**

**10.1 Seed Data Script**

**File: scripts/seed\_data.py** (tạo mới => Mock data)

import sys  
import os  
  
# Add parent directory to path  
sys.path.append(os.path.dirname(os.path.dirname(os.path.abspath(\_\_file\_\_))))  
  
from app.database import SessionLocal, User, Board, Task, StatusEnum, PriorityEnum  
from datetime import datetime, timedelta  
  
def seed\_data():  
 """Tạo dữ liệu mẫu trong database"""  
 db = SessionLocal()  
   
 try:  
 # Tạo users  
 users\_data = [  
 {  
 "username": "admin",  
 "email": "admin@example.com",   
 "password": "admin123",  
 "full\_name": "Administrator",  
 "is\_active": True  
 },  
 {  
 "username": "johndoe",  
 "email": "john@example.com",  
 "password": "password123",   
 "full\_name": "John Doe",  
 "is\_active": True  
 },  
 {  
 "username": "janesmith",   
 "email": "jane@example.com",  
 "password": "password123",  
 "full\_name": "Jane Smith",  
 "is\_active": True  
 }  
 ]  
   
 db\_users = []  
 for user\_data in users\_data:  
 # Kiểm tra user đã tồn tại chưa  
 existing\_user = db.query(User).filter(User.username == user\_data["username"]).first()  
 if not existing\_user:  
 db\_user = User(\*\*user\_data)  
 db.add(db\_user)  
 db.commit()  
 db.refresh(db\_user)  
 db\_users.append(db\_user)  
 print(f"Created user: {db\_user.username}")  
 else:  
 db\_users.append(existing\_user)  
 print(f"User already exists: {existing\_user.username}")  
   
 # Tạo boards  
 boards\_data = [  
 {  
 "name": "Personal Tasks",  
 "description": "Quản lý công việc cá nhân",  
 "is\_public": False,  
 "owner\_id": db\_users[1].id # johndoe  
 },  
 {  
 "name": "Work Project",  
 "description": "Dự án công ty",  
 "is\_public": True,  
 "owner\_id": db\_users[1].id # johndoe  
 },  
 {  
 "name": "Home Renovation",  
 "description": "Sửa chữa nhà cửa",  
 "is\_public": False,  
 "owner\_id": db\_users[2].id # janesmith  
 }  
 ]  
   
 db\_boards = []  
 for board\_data in boards\_data:  
 existing\_board = db.query(Board).filter(  
 Board.name == board\_data["name"],  
 Board.owner\_id == board\_data["owner\_id"]  
 ).first()  
   
 if not existing\_board:  
 db\_board = Board(\*\*board\_data)  
 db.add(db\_board)  
 db.commit()  
 db.refresh(db\_board)  
 db\_boards.append(db\_board)  
 print(f"Created board: {db\_board.name}")  
 else:  
 db\_boards.append(existing\_board)  
 print(f"Board already exists: {existing\_board.name}")  
   
 # Tạo tasks  
 tasks\_data = [  
 {  
 "title": "Setup development environment",  
 "description": "Cài đặt Python, FastAPI, PostgreSQL",  
 "status": StatusEnum.done,  
 "priority": PriorityEnum.high,  
 "position": 0,  
 "board\_id": db\_boards[0].id,  
 "assigned\_to": db\_users[1].id,  
 "due\_date": datetime.utcnow() - timedelta(days=1)  
 },  
 {  
 "title": "Design database schema",   
 "description": "Thiết kế ERD và relationships",  
 "status": StatusEnum.done,  
 "priority": PriorityEnum.high,  
 "position": 1,  
 "board\_id": db\_boards[0].id,  
 "assigned\_to": db\_users[1].id  
 },  
 {  
 "title": "Implement SQLAlchemy models",  
 "description": "Tạo User, Board, Task models",  
 "status": StatusEnum.in\_progress,  
 "priority": PriorityEnum.medium,  
 "position": 0,  
 "board\_id": db\_boards[0].id,  
 "assigned\_to": db\_users[1].id,  
 "due\_date": datetime.utcnow() + timedelta(days=2)  
 },  
 {  
 "title": "Write API tests",  
 "description": "Unit tests cho tất cả endpoints",  
 "status": StatusEnum.todo,  
 "priority": PriorityEnum.low,  
 "position": 0,  
 "board\_id": db\_boards[0].id  
 },  
 {  
 "title": "Setup CI/CD pipeline",  
 "description": "GitHub Actions for automated testing",  
 "status": StatusEnum.todo,  
 "priority": PriorityEnum.medium,  
 "position": 1,  
 "board\_id": db\_boards[1].id,  
 "assigned\_to": db\_users[2].id  
 }  
 ]  
   
 for task\_data in tasks\_data:  
 existing\_task = db.query(Task).filter(Task.title == task\_data["title"]).first()  
 if not existing\_task:  
 db\_task = Task(\*\*task\_data)  
 db.add(db\_task)  
 db.commit()  
 db.refresh(db\_task)  
 print(f"Created task: {db\_task.title}")  
 else:  
 print(f"Task already exists: {existing\_task.title}")  
   
 print("Seed data completed successfully!")  
   
 except Exception as e:  
 print(f"Error seeding data: {e}")  
 db.rollback()  
 finally:  
 db.close()  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 seed\_data()

**10.2 Chạy seed data**

python scripts/seed\_data.py

**Bước 11. Testing và Verification**

**11.1 Kiểm tra database**

# Kết nối tài khoản postgres kết nối database  
sudo -i -u postgres

#Kết nối vào database

psql

# Xem tables  
\dt  
  
# Xem data trong users table  
SELECT \* FROM users;  
  
# Xem data trong boards table   
SELECT \* FROM boards;  
  
# Xem data trong tasks table với join  
SELECT t.title, t.status, b.name as board\_name, u.username as assigned\_user  
FROM tasks t  
JOIN boards b ON t.board\_id = b.id  
LEFT JOIN users u ON t.assigned\_to = u.id;

**11.2 Test APIs với Swagger UI**

1. Khởi động server: **uvicorn app.main:app --reload**
2. Truy cập: <http://127.0.0.1:8000/docs>

Test từng API bằng Swagger để verify database integration

# Lỗi thường gặp

## Migration errors

# Reset migrations (development only)  
rm -rf migrations/  
alembic init migrations  
# Cấu hình lại env.py và alembic.ini  
alembic revision --autogenerate -m "Initial migration"  
alembic upgrade head

## Database connection errors

# Kiểm tra database URL  
from app.database.connection import engine  
print(engine.url)  
  
# Test connection  
try:  
 connection = engine.connect()  
 print("Database connected successfully!")  
 connection.close()  
except Exception as e:  
 print(f"Database connection failed: {e}")

**SQLAlchemy relationship errors**

* Đảm bảo back\_populates khớp nhau
* Import tất cả models trong migrations/env.py
* Sử dụng lazy="select" nếu có circular import

**13.2 Performance tips**

* Sử dụng joinedload() cho relationships cần thiết
* Add database indexes cho các cột thường query
* Implement pagination cho large datasets
* Sử dụng select() thay vì query() cho performance tốt hơn

**13.3 Development workflow**

# Sau mỗi thay đổi model:  
alembic revision --autogenerate -m "Description of change"  
alembic upgrade head  
  
# Backup database (SQLite)  
cp kanban\_todo.db kanban\_todo.db.backup  
  
# Reset database (development)  
rm kanban\_todo.db  
alembic upgrade head  
python scripts/seed\_data.py

**6. Bài tập thực hành**

**6.1 Bài tập cơ bản**

1. **Kiểm tra tất cả APIs**: Test CRUD operations qua Swagger UI
2. **Thêm dữ liệu**: Tạo thêm users, boards, tasks qua API
3. **Verify relationships**: Kiểm tra foreign key constraints hoạt động
4. **Test cascading**: Xóa board và verify tasks cũng bị xóa

**6.2 Bài tập nâng cao**

1. **Thêm indexes**:
   * Index cho username, email
   * Composite index cho (board\_id, status)
2. **Optimize queries**:
   * Sử dụng joinedload để load relationships
   * Implement efficient search với full-text search
3. **Advanced migrations**:
   * Thêm bảng TaskHistory để track changes

Thêm constraints và triggers