**THỰC HÀNH BUỔI 4**

# Mục tiêu

✅ Thiết lập quy trình làm việc Git với chiến lược phân nhánh

✅Hiểu và thực hành xây dựng backend API từ con số 0

✅Setup project FastAPI với cấu trúc thư mục chuẩn

✅Tạo Pydantic schemas để validate dữ liệu

✅Xây dựng mock data system thay thế database

✅Implement CRUD operations cho User, Board, Task

✅Test API qua Swagger UI

✅Chuẩn bị nền tảng cho database integration và authentication

# Yêu cầu chuẩn bị

* Khởi động môi trường ảo và chạy pip install requirements.txt (nếu chưa)
* Tạo 1 tài khoản Github bằng tài khoản sinh viên (Để đăng kí dạng Github Education (Github Copilot)
* Cài đặt Postman để test API

# Lý thuyết

## Quản lý Mã nguồn với Git

Git là gì?

* Git là hệ thống kiểm soát phiên bản phân tán (Distributed Version Control System)
* Được phát triển bởi Linus Torvalds (tác giả của Linux) năm 2005
* Cho phép theo dõi thay đổi trong code, hợp tác nhóm hiệu quả
* Là backbone của hầu hết các dự án phần mềm mã nguồn mở

## Tại sao cần Git?

* Backup an toàn: Không lo mất code khi máy hỏng
* Theo dõi thay đổi: Biết ai đã sửa gì, khi nào, tại sao
* Làm việc nhóm: Nhiều người cùng code không bị conflict
* Rollback: Quay lại phiên bản cũ khi có lỗi
* Branching: Phát triển tính năng mới mà không ảnh hưởng code chính

## GitHub/GitLab vs Git

**Git**: Công cụ dòng lệnh để quản lý version

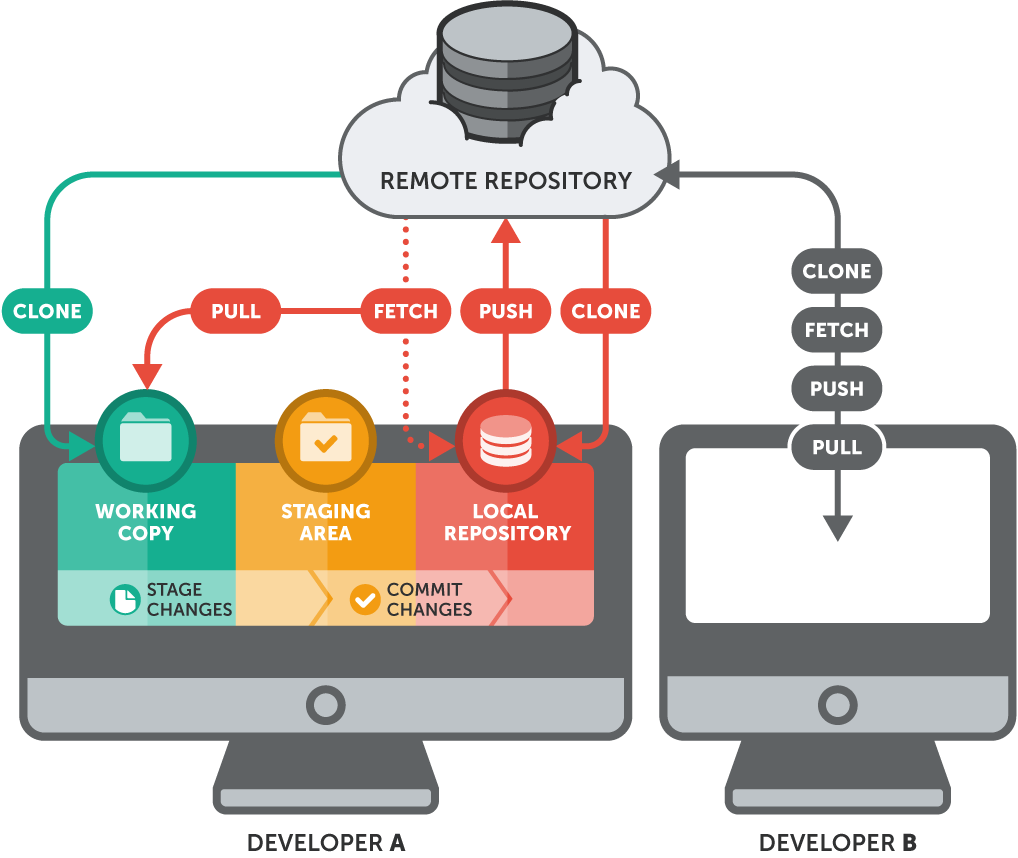
**GitHub/GitLab**: Dịch vụ cloud hosting Git repositories với giao diện web

**GitHub/GitLab** cung cấp thêm: Issues tracking, Pull Requests, Actions (CI/CD),…

## Quy trình Git cơ bản (Git Workflow)

Git Repository Lifecycle:

* Working Directory: Nơi bạn chỉnh sửa file
* Staging Area: Nơi chuẩn bị file để commit
* Repository: Nơi lưu trữ lịch sử commit



**Các lệnh Git quan trọng:**

* git init: Khởi tạo repository
* git add: Thêm file vào staging area
* git commit: Lưu snapshot vào repository
* git push: Đẩy code lên remote (GitHub)
* git pull: Kéo code mới từ remote về
* git branch: Quản lý nhánh phát triển
* git merge: Gộp nhánh

A diagram of a local and local reload

AI-generated content may be incorrect.

## Teamwork và Cộng tác trong Mã nguồn mở

*Xem thêm tại:* [*https://docs.github.com/en/organizations/organizing-members-into-teams/about-teams*](https://docs.github.com/en/organizations/organizing-members-into-teams/about-teams)

**Open Source Development Model:**

* **Fork & Pull Request**: Tạo bản sao, sửa đổi, đề xuất merge
* **Issues**: Báo cáo lỗi, đề xuất tính năng
* **Code Review**: Kiểm tra code trước khi merge
* **Documentation**: README, Wiki, inline comments

**Best Practices cho Teamwork:**

* **Meaningful commit messages**: Mô tả rõ ràng thay đổi
* **Small, focused commits**: Mỗi commit làm một việc cụ thể
* **Regular pulls**: Cập nhật code từ team thường xuyên
* **Branch naming**: Tên nhánh mô tả tính năng (feature/user-auth)

A diagram of a software system

AI-generated content may be incorrect.

Quy tắc quản lý Team management role

*Xem chi tiết tại* [*https://docs.github.com/en/organizations/managing-peoples-access-to-your-organization-with-roles/roles-in-an-organization*](https://docs.github.com/en/organizations/managing-peoples-access-to-your-organization-with-roles/roles-in-an-organization)

A diagram of a system

AI-generated content may be incorrect.

## Backend Development là gì?

Backend là phần "sau" của ứng dụng web, chịu trách nhiệm:

* Xử lý logic nghiệp vụ (business logic)
* Quản lý và lưu trữ dữ liệu
* Xác thực người dùng và phân quyền
* Cung cấp API cho frontend sử dụng

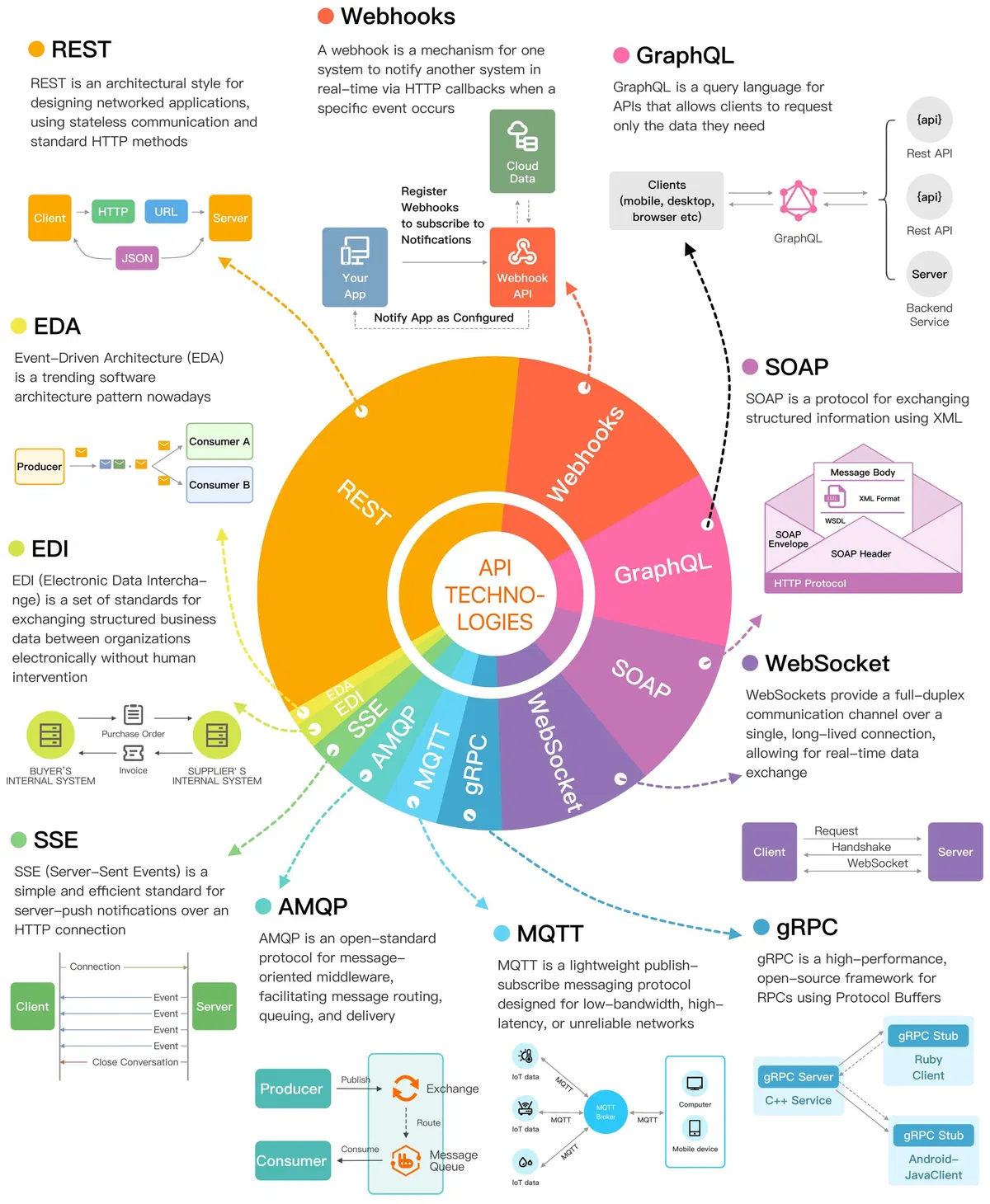
So sánh Frontend vs Backend:

A black screen with white text

AI-generated content may be incorrect.

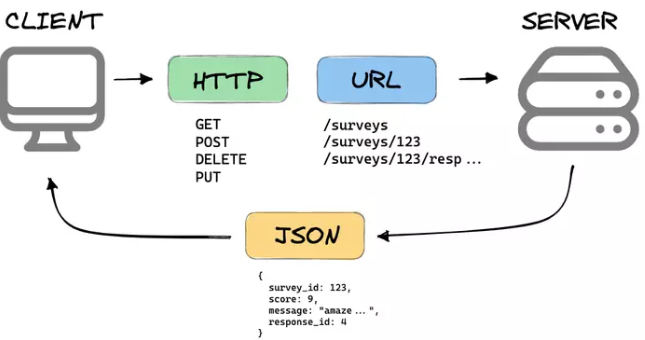
## API - RESTful API và HTTP Methods

**API (Application Programming Interface)** là cách để các ứng dụng giao tiếp với nhau thông qua các endpoint.



**REST API (Representational State Transfer):**

* Kiến trúc để thiết kế web services
* Sử dụng **HTTP methods** để thao tác với resources
* Stateless: Mỗi request độc lập, không phụ thuộc session
* Uniform Interface: URL có cấu trúc nhất quán, dễ hiểu



**HTTP Methods chính:**

* **GET**: Lấy dữ liệu (Read)
* **POST**: Tạo dữ liệu mới (Create)
* **PUT**: Cập nhật toàn bộ resource (Update)
* **PATCH**: Cập nhật một phần resource (Partial Update)
* **DELETE**: Xóa resource (Delete)

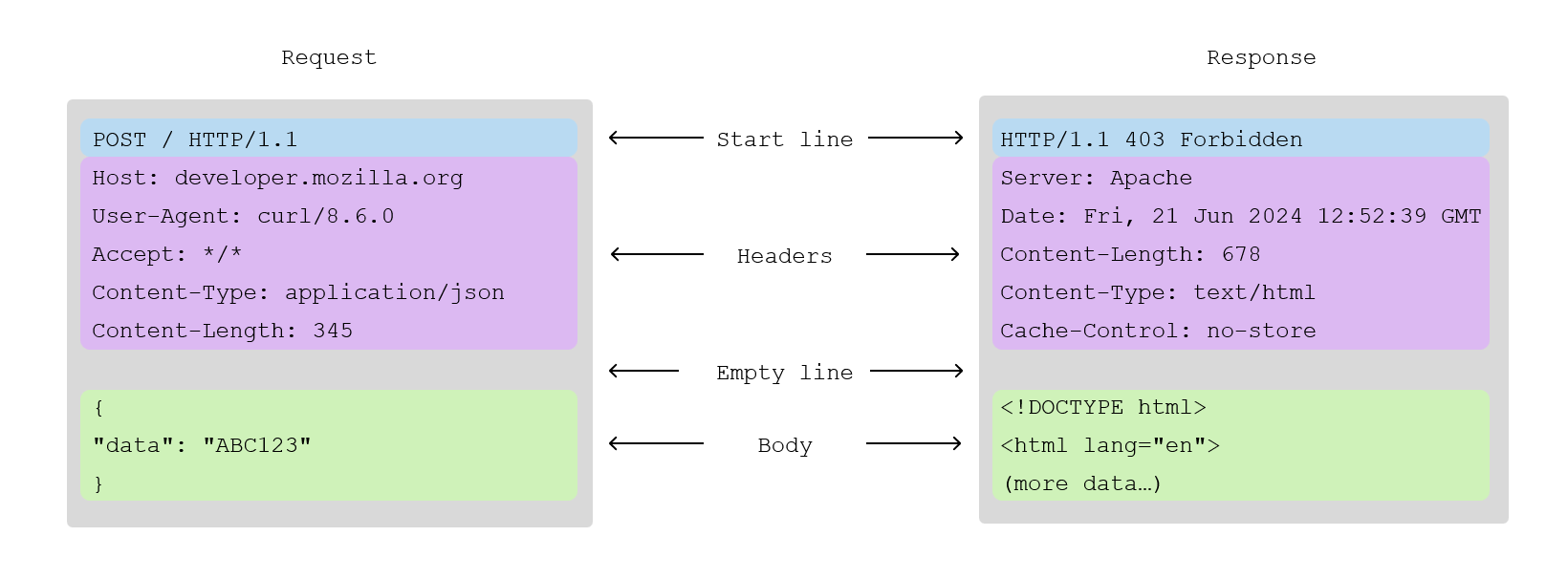
**HTTP Status Codes:**

*Xem thêm tại:* [*https://httpstatuses.com/*](https://httpstatuses.com/)

* **2xx Success**: 200 (OK), 201 (Created), 204 (No Content)
* **4xx Client Error**: 400 (Bad Request), 404 (Not Found), 401 (Unauthorized)
* **5xx Server Error**: 500 (Internal Server Error)

**A diagram of a webpage

AI-generated content may be incorrect.**



**RESTful URL Design:**

* **Good**: /api/tasks, /api/tasks/123, /api/users/456/tasks
* **Bad**: /api/getTasks, /api/task?action=delete&id=123

## API Documentation và OpenAPI

Xem thêm tại: <https://swagger.io/docs/>

A screenshot of a computer program

AI-generated content may be incorrect.

**OpenAPI Specification (fka Swagger):**

* Tiêu chuẩn mô tả REST API
* FastAPI tự động tạo OpenAPI spec từ code
* Sinh ra interactive documentation

**Lợi ích của Auto Documentation:**

* **Testing**: Test API ngay trong browser
* **Client Generation**: Tự động tạo SDK cho nhiều ngôn ngữ
* **Team Communication**: Spec rõ ràng cho frontend/mobile dev

## FastAPI - Framework Web hiện đại

A close up of a logo

AI-generated content may be incorrect.

*Xem thêm tại:* [*https://fastapi.tiangolo.com/tutorial/*](https://fastapi.tiangolo.com/tutorial/)

**FastAPI là gì?**

* Framework Python để xây dựng API nhanh chóng và hiệu quả
* Được tạo bởi Sebastián Ramirez (tiangolo) năm 2018
* Dựa trên các tiêu chuẩn mở: OpenAPI, JSON Schema
* Hỗ trợ async/await nên có hiệu suất cao

### Ưu điểm của FastAPI

* **Performance cao**: Nhanh ngang NodeJS, Go
* **Easy to learn**: Cú pháp Python đơn giản
* **Auto documentation**: Tự động tạo API docs (Swagger UI)
* **Type hints**: Kiểm tra kiểu dữ liệu tự động
* **Modern standards**: Hỗ trợ async, WebSocket, GraphQL
* **Production ready**: Dùng trong nhiều công ty lớn

### So sánh với các framework khác

* **Django**: Nặng, full-featured, tốt cho web app phức tạp
* **Flask**: Nhẹ, linh hoạt, cần setup nhiều
* **FastAPI**: Cân bằng giữa đơn giản và mạnh mẽ, focus vào API

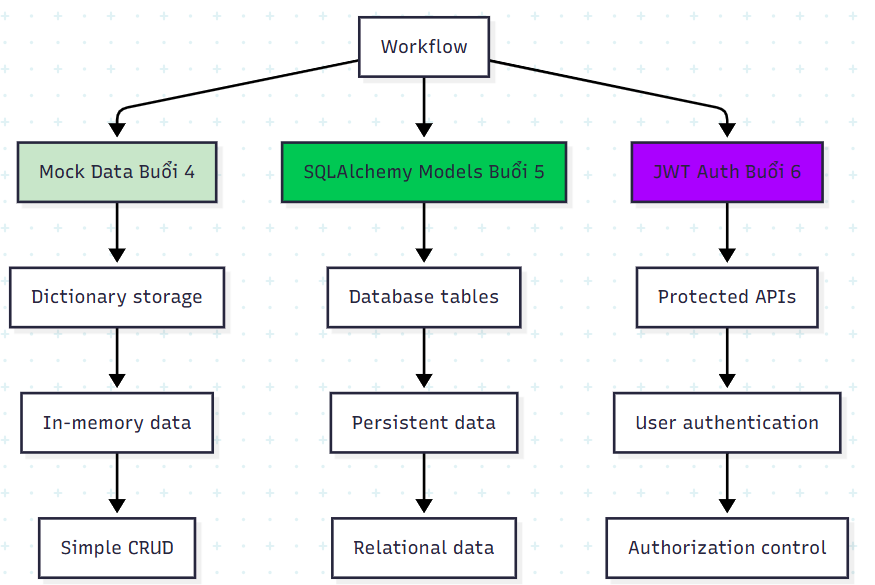
## Pydantic Models

*Xem thêm tại:* [*https://pydantic-docs.helpmanual.io/*](https://pydantic-docs.helpmanual.io/)

* Định nghĩa cấu trúc dữ liệu với type hints
* Tự động validation và serialization
* Tích hợp với FastAPI để tạo OpenAPI schema

**Hoạt động**

* FastAPI sử dụng **Pydantic** để xác thực dữ liệu đầu vào/ra.
* Các route được định nghĩa bằng decorators như @app.get(), @app.post(), v.v.
* **Không cần setup database system**, dữ liệu thường **được lưu tạm trong bộ nhớ (RAM) hoặc trả về dữ liệu mẫu (mock)**



# Hướng dẫn thực hành

## 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

## Cài đặt FastAPI framework

Các thư viện đã được cài đặt theo file requirements.txt (Xem lại file **Thực hành buổi 1**) nếu chưa thực hiện như sau:

Cài đặt thư viện chính:

pip install fastapi uvicorn python-multipart python-jose[cryptography] passlib[bcrypt] python-decouple

## Git cơ bản và tạo repository

***Khởi tạo dự án***

chmod +x setup-project.sh

./setup-project.sh

***Bước 1: Cấu hình Git (tài khoản kết nối với Github):***

git config --global user.name "Tên của bạn"

git config --global user.email [email@example.com](mailto:email@example.com)

git remote add origin git@github.com:[username]/kanban-todo-project.git

# Kiểm tra cấu hình

git config --list

***Bước 2: Khởi tạo Git repository***

git init

git status

Nội dung file .gitignore:

# Python

\_\_pycache\_\_/

\*.py[cod]

\*$py.class

\*.so

.Python

env/

venv/

ENV/

\*.egg-info/

dist/

build/

# IDE

.vscode/

.idea/

\*.swp

\*.swo

\*~

# Database

\*.db

\*.sqlite3

# Logs

\*.log

# OS

.DS\_Store

Thumbs.db

# Environment variables

.env

.env.local

# FastAPI

.pytest\_cache/

***Bước 4: Tạo README.md***

# My TODO App

Ứng dụng quản lý công việc với FastAPI

## Tính năng

- ✅ CRUD operations cho tasks

- 🔐 Xác thực người dùng cơ bản

- 📊 Thống kê tasks

- 📚 Auto-generated API documentation

## Công nghệ sử dụng

- \*\*Backend\*\*: FastAPI, Python 3.8+

- \*\*Database\*\*: SQLite (development)

- \*\*Documentation\*\*: Swagger UI, ReDoc

## Cài đặt và chạy

### Requirements

pip install fastapi uvicorn python-multipart

Chạy development server

uvicorn main:app --reload

Truy cập

API: http://localhost:8000

Swagger UI: http://localhost:8000/docs

ReDoc: <http://localhost:8000/redoc>

***Bước 4: Commit lên Github***

git add .

git status

git commit -m "Initial commit: Setup project, API structure and README"

## Tạo GitHub repository và kết nối

**Bước 1: Tạo repository trên GitHub**

1. Đăng nhập GitHub.com
2. Click "New repository"
3. Đặt tên: todo-project
4. Chọn Public/Private
5. KHÔNG tick "Initialize with README" (vì đã có local)
6. Click "Create repository"

**Bước 2: Các bạn sinh viên khác kết nối local với GitHub**

# Thêm remote origin (thay YOUR\_USERNAME bằng username GitHub của bạn)

git remote add origin https://github.com/YOUR\_USERNAME/todo-project.git

# Kiểm tra remote

git remote -v

# Push code lên GitHub

git branch -M main

git push -u origin main

## Cấu hình cơ bản

## Viết ứng dụng FastAPI đầu tiên

Tạo File: app/main.py

from fastapi import FastAPI

# Tạo instance FastAPI

app = FastAPI(

title="Kanban TODO API",

description="API đơn giản để quản lý công việc theo mô hình Kanban",

version="1.0.0"

)

# Route đầu tiên - GET /

@app.get("/")

async def root():

return {

"message": "Chào mừng đến với Kanban TODO API!",

"version": "1.0.0",

"status": "running"

}

# Route thứ hai - GET /health

@app.get("/health")

async def health\_check():

return {

"status": "healthy",

"service": "Kanban TODO API"

}

***Giải thích code***

# 1. Import FastAPI class

from fastapi import FastAPI

# 2. Tạo instance app với metadata

app = FastAPI(

title="Tên API", # Tên hiển thị trong docs

description="Mô tả API", # Mô tả chi tiết

version="1.0.0" # Phiên bản API

)

# 3. Decorator định nghĩa endpoint

@app.get("/") # HTTP Method + Path

async def function\_name(): # Hàm xử lý (async recommended)

return {"key": "value"} # Response dạng JSON

**Chạy FastAPI development server**

# Chạy từ thư mục gốc dự án

uvicorn main:app --reload

# Hoặc chạy với options chi tiết

uvicorn main:app --reload --host 127.0.0.1 --port 8000

Giải thích lệnh:

* app.main:app: Module main.py trong folder app, object app
* --reload: Tự động restart khi code thay đổi
* --host: IP address (127.0.0.1 = localhost)
* --port: Cổng (default 8000)

**Kiểm tra ứng dụng**

Mở trình duyệt và truy cập:

* **API endpoint**: <http://127.0.0.1:8000/>
* **Health check**: <http://127.0.0.1:8000/health>
* **API docs**: <http://127.0.0.1:8000/docs>
* **ReDoc**: <http://127.0.0.1:8000/redoc>

**RESTful API Planning** **(ở buổi 3)**

# User Management

POST /auth/register # Đăng ký

POST /auth/login # Đăng nhập

GET /users/me # Xem profile

PUT /users/me # Cập nhật profile

# 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

# 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)

## Tạo Pydantic Schemas

**File: app/schemas/user.py**

from pydantic import BaseModel, validator  
from datetime import datetime  
from typing import Optional  
  
class UserBase(BaseModel):  
 username: str  
 email: Optional[str] = None  
 full\_name: Optional[str] = None  
  
 @validator('username')  
 def username\_must\_be\_valid(cls, v):  
 if not v or len(v.strip()) < 3:  
 raise ValueError('Username phải có ít nhất 3 ký tự')  
 if len(v) > 20:  
 raise ValueError('Username không được quá 20 ký tự')  
 return v.strip().lower()  
  
class UserCreate(UserBase):  
 password: str  
  
 @validator('password')  
 def password\_must\_be\_strong(cls, v):  
 if len(v) < 6:  
 raise ValueError('Mật khẩu phải có ít nhất 6 ký tự')  
 return v  
  
class UserResponse(UserBase):  
 id: int  
 is\_active: bool  
 created\_at: datetime  
 updated\_at: datetime  
  
 class Config:  
 # Cho phép chuyển đổi từ dict sang Pydantic model  
 orm\_mode = True  
  
class UserUpdate(BaseModel):  
 email: Optional[str] = None  
 full\_name: Optional[str] = None  
  
class PasswordChange(BaseModel):  
 current\_password: str  
 new\_password: str  
  
 @validator('new\_password')  
 def new\_password\_must\_be\_strong(cls, v):  
 if len(v) < 6:  
 raise ValueError('Mật khẩu mới phải có ít nhất 6 ký tự')  
 return v

**5.3 Board schemas**

**File: app/schemas/board.py**

from pydantic import BaseModel, validator

from datetime import datetime

from typing import Optional, List

class BoardBase(BaseModel):

    name: str

    description: Optional[str] = None

    is\_public: bool = False

    @validator('name')

    def name\_must\_not\_be\_empty(cls, v):

        if not v or len(v.strip()) == 0:

            raise ValueError('Tên board không được để trống')

        if len(v.strip()) > 100:

            raise ValueError('Tên board không được quá 100 ký tự')

        return v.strip()

class BoardCreate(BoardBase):

    pass

class BoardUpdate(BaseModel):

    name: Optional[str] = None

    description: Optional[str] = None

    is\_public: Optional[bool] = None

    @validator('name')

    def name\_validator(cls, v):

        if v is not None and (not v or len(v.strip()) == 0):

            raise ValueError('Tên board không được để trống')

        return v.strip() if v else v

class BoardResponse(BoardBase):

    id: int

    owner\_id: int

    created\_at: datetime

    updated\_at: datetime

    tasks\_count: Optional[int] = 0

    class Config:

        orm\_mode = True

class BoardWithTasks(BoardResponse):

    tasks: List['TaskResponse'] = []

# Try to resolve forward references (TaskResponse is defined in app.schemas.task)

try:

    # Importing here avoids circular import at module import time

    from app.schemas.task import TaskResponse  # noqa: F401

    # Rebuild model to let Pydantic resolve the forward ref

    BoardWithTasks.model\_rebuild()

except Exception:

    # If rebuild fails during import time, FastAPI/Pydantic will attempt resolution later.

    pass

**5.4 Task schemas**

**File: app/schemas/task.py**

from pydantic import BaseModel, validator  
from datetime import datetime  
from typing import Optional  
from enum import Enum  
  
class StatusEnum(str, Enum):  
 todo = "todo"  
 in\_progress = "in\_progress"  
 done = "done"  
  
class PriorityEnum(str, Enum):  
 low = "low"  
 medium = "medium"  
 high = "high"  
  
class TaskBase(BaseModel):  
 title: str  
 description: Optional[str] = None  
 priority: PriorityEnum = PriorityEnum.medium  
 status: StatusEnum = StatusEnum.todo  
  
 @validator('title')  
 def title\_must\_not\_be\_empty(cls, v):  
 if not v or len(v.strip()) == 0:  
 raise ValueError('Tiêu đề task không được để trống')  
 if len(v.strip()) > 200:  
 raise ValueError('Tiêu đề task không được quá 200 ký tự')  
 return v.strip()  
  
class TaskCreate(TaskBase):  
 board\_id: int  
  
class TaskUpdate(BaseModel):  
 title: Optional[str] = None  
 description: Optional[str] = None  
 priority: Optional[PriorityEnum] = None  
  
 @validator('title')  
 def title\_validator(cls, v):  
 if v is not None and (not v or len(v.strip()) == 0):  
 raise ValueError('Tiêu đề task không được để trống')  
 return v.strip() if v else v  
  
class TaskMove(BaseModel):  
 status: StatusEnum  
 position: Optional[int] = None  
  
class TaskAssign(BaseModel):  
 assigned\_to: Optional[int] = None  
  
class TaskResponse(TaskBase):  
 id: int  
 board\_id: int  
 position: int  
 assigned\_to: Optional[int] = None  
 due\_date: Optional[datetime] = None  
 created\_at: datetime  
 updated\_at: datetime  
  
 class Config:  
 orm\_mode = True

**6. Tạo Mock Data System**

**6.1 Mock Data là gì?**

* **Mock Data**: Dữ liệu giả lập thay thế database
* **Lưu trữ trong memory**: Dữ liệu mất khi restart server
* **Mục đích**: Test API logic trước khi tích hợp database thật

**Ở đây chúng ta sẽ tạo giả lập dữ liệu như sau**:

* **2 user**: admin và user
* **2 board**: Personal tasks, Work project
* **3 task**: ToDo, In-process, Done

**File: app/models/mock\_data.py**

from datetime import datetime, timedelta  
from typing import Dict, List, Optional  
  
# Mock databases - dictionary lưu trữ dữ liệu trong memory  
users\_db: Dict[int, dict] = {}  
boards\_db: Dict[int, dict] = {}  
tasks\_db: Dict[int, dict] = {}  
  
# Auto-increment IDs  
next\_user\_id = 1  
next\_board\_id = 1  
next\_task\_id = 1  
  
def init\_sample\_data():  
 """Khởi tạo dữ liệu mẫu để test"""  
 global next\_user\_id, next\_board\_id, next\_task\_id  
   
 # Tạo sample users  
 sample\_users = [  
 {  
 "id": 1,  
 "username": "admin",  
 "email": "admin@example.com",  
 "password": "admin123", # Thực tế sẽ hash password  
 "full\_name": "Administrator",  
 "is\_active": True,  
 "created\_at": datetime.utcnow(),  
 "updated\_at": datetime.utcnow()  
 },  
 {  
 "id": 2,  
 "username": "johndoe",   
 "email": "john@example.com",  
 "password": "password123",  
 "full\_name": "John Doe",  
 "is\_active": True,  
 "created\_at": datetime.utcnow(),  
 "updated\_at": datetime.utcnow()  
 }  
 ]  
   
 for user in sample\_users:  
 users\_db[user["id"]] = user  
 next\_user\_id = 3  
   
# Tạo sample boards  
 sample\_boards = [  
 {  
 "id": 1,  
 "name": "Personal Tasks",  
 "description": "Quản lý công việc cá nhân",  
 "owner\_id": 2,  
 "is\_public": False,  
 "created\_at": datetime.utcnow(),  
 "updated\_at": datetime.utcnow()  
 },  
 {  
 "id": 2,  
 "name": "Work Project",  
 "description": "Dự án công ty",  
 "owner\_id": 2,  
 "is\_public": True,  
 "created\_at": datetime.utcnow(),  
 "updated\_at": datetime.utcnow()  
 }  
 ]  
   
 for board in sample\_boards:  
 boards\_db[board["id"]] = board  
 next\_board\_id = 3  
   
# Tạo sample tasks  
 sample\_tasks = [  
 {  
 "id": 1,  
 "title": "Setup development environment",  
 "description": "Cài đặt Python, FastAPI, và các công cụ cần thiết",  
 "status": "todo",  
 "priority": "high",  
 "position": 0,  
 "board\_id": 1,  
 "assigned\_to": 2,  
 "due\_date": datetime.utcnow() + timedelta(days=3),  
 "created\_at": datetime.utcnow(),  
 "updated\_at": datetime.utcnow()  
 },  
 {  
 "id": 2,  
 "title": "Thiết kế API endpoints",  
 "description": "Tạo specification cho tất cả API",  
 "status": "done",  
 "priority": "high",   
 "position": 0,  
 "board\_id": 1,  
 "assigned\_to": 2,  
 "due\_date": None,  
 "created\_at": datetime.utcnow() - timedelta(days=2),  
 "updated\_at": datetime.utcnow() - timedelta(days=1)  
 },  
 {  
 "id": 3,  
 "title": "Implement FastAPI",  
 "description": "Viết code cho các API endpoints",  
 "status": "in\_progress",  
 "priority": "medium",  
 "position": 0,  
 "board\_id": 2,  
 "assigned\_to": 2,  
 "due\_date": datetime.utcnow() + timedelta(days=5),  
 "created\_at": datetime.utcnow(),  
 "updated\_at": datetime.utcnow()  
 }  
 ]  
   
 for task in sample\_tasks:  
 tasks\_db[task["id"]] = task  
 next\_task\_id = 4  
  
# Helper functions cho User operations  
def get\_user\_by\_username(username: str) -> Optional[dict]:  
 """Tìm user theo username"""  
 for user in users\_db.values():  
 if user["username"] == username:  
 return user  
 return None  
  
def get\_user\_by\_id(user\_id: int) -> Optional[dict]:  
 """Tìm user theo ID"""  
 return users\_db.get(user\_id)  
  
def create\_user(user\_data: dict) -> dict:  
 """Tạo user mới"""  
 global next\_user\_id  
 user = {  
 "id": next\_user\_id,  
 "is\_active": True,  
 "created\_at": datetime.utcnow(),  
 "updated\_at": datetime.utcnow(),  
 \*\*user\_data # Spread operator - giải nén tất cả key-value từ user\_data  
 }  
 users\_db[next\_user\_id] = user  
 next\_user\_id += 1  
 return user  
  
def update\_user(user\_id: int, updates: dict) -> Optional[dict]:  
 """Cập nhật thông tin user"""  
 if user\_id in users\_db:  
 users\_db[user\_id].update(updates)  
 users\_db[user\_id]["updated\_at"] = datetime.utcnow()  
 return users\_db[user\_id]  
 return None  
  
# Helper functions cho Board operations   
def get\_all\_boards() -> List[dict]:  
 """Lấy tất cả boards"""  
 return list(boards\_db.values())  
  
def get\_board\_by\_id(board\_id: int) -> Optional[dict]:  
 """Tìm board theo ID"""  
 return boards\_db.get(board\_id)  
  
def create\_board(board\_data: dict) -> dict:  
 """Tạo board mới"""  
 global next\_board\_id  
 board = {  
 "id": next\_board\_id,  
 "created\_at": datetime.utcnow(),  
 "updated\_at": datetime.utcnow(),  
 \*\*board\_data  
 }  
 boards\_db[next\_board\_id] = board  
 next\_board\_id += 1  
 return board  
  
def update\_board(board\_id: int, updates: dict) -> Optional[dict]:  
 """Cập nhật board"""  
 if board\_id in boards\_db:  
 boards\_db[board\_id].update(updates)  
 boards\_db[board\_id]["updated\_at"] = datetime.utcnow()  
 return boards\_db[board\_id]  
 return None  
  
def delete\_board(board\_id: int) -> bool:  
 """Xóa board và tất cả tasks trong board đó"""  
 if board\_id in boards\_db:  
 # Xóa tất cả tasks thuộc board này  
 task\_ids\_to\_delete = [  
 task\_id for task\_id, task in tasks\_db.items()   
 if task["board\_id"] == board\_id  
 ]  
 for task\_id in task\_ids\_to\_delete:  
 del tasks\_db[task\_id]  
   
 # Xóa board  
 del boards\_db[board\_id]  
 return True  
 return False  
  
# Helper functions cho Task operations  
def get\_tasks\_by\_board(  
 board\_id: int,   
 status: Optional[str] = None,   
 priority: Optional[str] = None  
) -> List[dict]:  
 """Lấy tasks theo board, có thể filter theo status và priority"""  
 tasks = [task for task in tasks\_db.values() if task["board\_id"] == board\_id]  
   
 if status:  
 tasks = [task for task in tasks if task["status"] == status]  
   
 if priority:  
 tasks = [task for task in tasks if task["priority"] == priority]  
   
 # Sắp xếp theo position, sau đó theo created\_at  
 tasks.sort(key=lambda x: (x["position"], x["created\_at"]))  
 return tasks  
  
def get\_task\_by\_id(task\_id: int) -> Optional[dict]:  
 """Tìm task theo ID"""  
 return tasks\_db.get(task\_id)  
  
def create\_task(task\_data: dict) -> dict:  
 """Tạo task mới"""  
 global next\_task\_id  
   
 # Tính position tiếp theo cho status này  
 board\_tasks = get\_tasks\_by\_board(task\_data["board\_id"], task\_data["status"])  
 next\_position = len(board\_tasks)  
   
 task = {  
 "id": next\_task\_id,  
 "position": next\_position,  
 "assigned\_to": None,  
 "due\_date": None,  
 "created\_at": datetime.utcnow(),  
 "updated\_at": datetime.utcnow(),  
 \*\*task\_data  
 }  
 tasks\_db[next\_task\_id] = task  
 next\_task\_id += 1  
 return task  
  
def update\_task(task\_id: int, updates: dict) -> Optional[dict]:  
 """Cập nhật task"""  
 if task\_id in tasks\_db:  
 tasks\_db[task\_id].update(updates)  
 tasks\_db[task\_id]["updated\_at"] = datetime.utcnow()  
 return tasks\_db[task\_id]  
 return None  
  
def delete\_task(task\_id: int) -> bool:  
 """Xóa task"""  
 if task\_id in tasks\_db:  
 del tasks\_db[task\_id]  
 return True  
 return False  
  
def move\_task(task\_id: int, new\_status: str, new\_position: Optional[int] = None) -> Optional[dict]:  
 """Di chuyển task sang status mới và/hoặc position mới"""  
 if task\_id not in tasks\_db:  
 return None  
   
 task = tasks\_db[task\_id]  
 old\_status = task["status"]  
   
 # Nếu chuyển sang status khác, tính position mới  
 if new\_status != old\_status:  
 if new\_position is None:  
 # Chuyển xuống cuối danh sách status mới  
 same\_status\_tasks = get\_tasks\_by\_board(task["board\_id"], new\_status)  
 new\_position = len(same\_status\_tasks)  
   
 updates = {"status": new\_status}  
 if new\_position is not None:  
 updates["position"] = new\_position  
   
 return update\_task(task\_id, updates)  
  
def search\_tasks(query: str, board\_id: Optional[int] = None) -> List[dict]:  
 """Tìm kiếm tasks theo từ khóa"""  
 all\_tasks = list(tasks\_db.values())  
   
 if board\_id:  
 all\_tasks = [task for task in all\_tasks if task["board\_id"] == board\_id]  
   
 # Tìm trong title và description  
 search\_query = query.lower()  
 matching\_tasks = []  
   
 for task in all\_tasks:  
 if (search\_query in task["title"].lower() or   
 (task["description"] and search\_query in task["description"].lower())):  
 matching\_tasks.append(task)  
   
 return matching\_tasks  
  
# Khởi tạo dữ liệu mẫu khi import module  
init\_sample\_data()

**Bước 4: Tạo API Routers**

**7.1 User Management Routes**

**File: app/routers/users.py**

from fastapi import APIRouter, HTTPException, status

from typing import List

from app.schemas.user import UserCreate, UserResponse, UserUpdate, PasswordChange

from app.models.mock\_data import (

    get\_user\_by\_username, get\_user\_by\_id,

    create\_user, update\_user, users\_db

)

router = APIRouter(prefix="/users", tags=["users"])

@router.post("/", response\_model=UserResponse, status\_code=status.HTTP\_201\_CREATED)

def create\_new\_user(user\_data: UserCreate):

    """Tạo user mới"""

    # Kiểm tra username đã tồn tại chưa

    if get\_user\_by\_username(user\_data.username):

        raise HTTPException(

            status\_code=status.HTTP\_409\_CONFLICT,

            detail="Username đã tồn tại"

        )

    # Tạo user mới (chưa hash password, sẽ làm ở buổi sau)

    user\_dict = user\_data.dict()

    # Tạm thời lưu password thẳng (không an toàn, sẽ sửa ở buổi JWT)

    user = create\_user(user\_dict)

    return UserResponse(\*\*user)

@router.get("/", response\_model=List[UserResponse])

def get\_users():

    """Lấy danh sách tất cả users"""

    all\_users = list(users\_db.values())

    return [UserResponse(\*\*user) for user in all\_users]

@router.get("/{user\_id}", response\_model=UserResponse)

def get\_user(user\_id: int):

    """Lấy thông tin user theo ID"""

    user = get\_user\_by\_id(user\_id)

    if not user:

        raise HTTPException(

            status\_code=status.HTTP\_404\_NOT\_FOUND,

            detail="User không tồn tại"

        )

    return UserResponse(\*\*user)

@router.put("/{user\_id}", response\_model=UserResponse)

def update\_user\_info(user\_id: int, user\_update: UserUpdate):

    """Cập nhật thông tin user"""

    user = get\_user\_by\_id(user\_id)

    if not user:

        raise HTTPException(

            status\_code=status.HTTP\_404\_NOT\_FOUND,

            detail="User không tồn tại"

        )

    update\_data = user\_update.dict(exclude\_unset=True)

    if update\_data:

        updated\_user = update\_user(user\_id, update\_data)

        return UserResponse(\*\*updated\_user)

    return UserResponse(\*\*user)

@router.patch("/{user\_id}/password")

def change\_user\_password(user\_id: int, password\_change: PasswordChange):

    """Đổi mật khẩu user"""

    user = get\_user\_by\_id(user\_id)

    if not user:

        raise HTTPException(

            status\_code=status.HTTP\_404\_NOT\_FOUND,

            detail="User không tồn tại"

        )

    # Kiểm tra mật khẩu hiện tại (tạm thời so sánh trực tiếp)

    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 mật khẩu mới

    update\_user(user\_id, {"password": password\_change.new\_password})

    return {"message": "Đổi mật khẩu thành công"}

@router.get("/me/{user\_id}", response\_model=UserResponse)

def get\_current\_user\_info(user\_id: int):

    """Lấy thông tin user hiện tại (giả lập /users/me)"""

    user = get\_user\_by\_id(user\_id)

    if not user:

        raise HTTPException(

            status\_code=status.HTTP\_404\_NOT\_FOUND,

            detail="User không tồn tại"

        )

    return UserResponse(\*\*user)

**7.2 Board Management Routes**

**File: app/routers/boards.py**

from fastapi import APIRouter, HTTPException, status, Query  
from typing import List, Optional  
from app.schemas.board import BoardCreate, BoardResponse, BoardUpdate, BoardWithTasks  
from app.schemas.task import TaskResponse  
from app.models.mock\_data import (  
 get\_all\_boards, get\_board\_by\_id, create\_board,   
 update\_board, delete\_board, get\_tasks\_by\_board  
)  
  
router = APIRouter(prefix="/boards", tags=["boards"])  
  
@router.get("/", response\_model=List[BoardResponse])  
def get\_boards(  
 page: int = Query(1, ge=1, description="Trang hiện tại"),  
 limit: int = Query(10, ge=1, le=100, description="Số lượng boards mỗi trang")  
):  
 """Lấy danh sách boards với pagination"""  
 boards = get\_all\_boards()  
   
 # Pagination đơn giản  
 start = (page - 1) \* limit  
 end = start + limit  
 paginated\_boards = boards[start:end]  
   
 # Thêm tasks\_count cho mỗi board  
 response\_boards = []  
 for board in paginated\_boards:  
 board\_tasks = get\_tasks\_by\_board(board["id"])  
 board\_response = BoardResponse(\*\*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\_new\_board(  
 board\_data: BoardCreate,  
 owner\_id: int = Query(..., description="ID của user tạo board")  
):  
 """Tạo board mới"""  
 # Tạm thời dùng query parameter để biết ai tạo board  
 # Sau này sẽ lấy từ JWT token  
 board\_dict = board\_data.dict()  
 board\_dict["owner\_id"] = owner\_id  
   
 board = create\_board(board\_dict)  
 board\_response = BoardResponse(\*\*board)  
 board\_response.tasks\_count = 0  
 return board\_response  
  
@router.get("/{board\_id}", response\_model=BoardWithTasks)  
def get\_board\_detail(board\_id: int):  
 """Lấy chi tiết board kèm tasks"""  
 board = get\_board\_by\_id(board\_id)  
 if not board:  
 raise HTTPException(  
 status\_code=status.HTTP\_404\_NOT\_FOUND,  
 detail="Board không tồn tại"  
 )  
   
 # Lấy tất cả tasks trong board  
 tasks = get\_tasks\_by\_board(board\_id)  
 task\_responses = [TaskResponse(\*\*task) for task in tasks]  
   
 board\_response = BoardWithTasks(\*\*board)  
 board\_response.tasks = task\_responses  
 return board\_response  
  
@router.put("/{board\_id}", response\_model=BoardResponse)  
def update\_board\_info(board\_id: int, board\_update: BoardUpdate):  
 """Cập nhật thông tin board"""  
 board = get\_board\_by\_id(board\_id)  
 if not board:  
 raise HTTPException(  
 status\_code=status.HTTP\_404\_NOT\_FOUND,  
 detail="Board không tồn tại"  
 )  
   
 update\_data = board\_update.dict(exclude\_unset=True)  
 if update\_data:  
 updated\_board = update\_board(board\_id, update\_data)  
 board\_response = BoardResponse(\*\*updated\_board)  
 tasks = get\_tasks\_by\_board(board\_id)  
 board\_response.tasks\_count = len(tasks)  
 return board\_response  
   
 board\_response = BoardResponse(\*\*board)  
 tasks = get\_tasks\_by\_board(board\_id)  
 board\_response.tasks\_count = len(tasks)  
 return board\_response  
  
@router.delete("/{board\_id}")  
def delete\_board\_and\_tasks(board\_id: int):  
 """Xóa board và tất cả tasks trong board"""  
 board = get\_board\_by\_id(board\_id)  
 if not board:  
 raise HTTPException(  
 status\_code=status.HTTP\_404\_NOT\_FOUND,  
 detail="Board không tồn tại"  
 )  
   
 # Đếm số tasks sẽ bị xóa  
 tasks = get\_tasks\_by\_board(board\_id)  
 deleted\_tasks\_count = len(tasks)  
   
 # Xóa board (sẽ tự động xóa tasks trong mock\_data.py)  
 if delete\_board(board\_id):  
 return {  
 "message": "Xóa board thành công",  
 "deleted\_tasks\_count": deleted\_tasks\_count  
 }  
   
 raise HTTPException(  
 status\_code=status.HTTP\_500\_INTERNAL\_SERVER\_ERROR,  
 detail="Không thể xóa board"  
 )  
  
@router.get("/{board\_id}/stats")  
def get\_board\_statistics(board\_id: int):  
 """Lấy thống kê tasks trong board"""  
 board = get\_board\_by\_id(board\_id)  
 if not board:  
 raise HTTPException(  
 status\_code=status.HTTP\_404\_NOT\_FOUND,  
 detail="Board không tồn tại"  
 )  
   
 all\_tasks = get\_tasks\_by\_board(board\_id)  
   
 # Thống kê theo status  
 todo\_count = len([t for t in all\_tasks if t["status"] == "todo"])  
 in\_progress\_count = len([t for t in all\_tasks if t["status"] == "in\_progress"])  
 done\_count = len([t for t in all\_tasks if t["status"] == "done"])  
   
 # Thống kê theo priority  
 high\_priority\_count = len([t for t in all\_tasks if t["priority"] == "high"])  
 medium\_priority\_count = len([t for t in all\_tasks if t["priority"] == "medium"])  
 low\_priority\_count = len([t for t in all\_tasks if t["priority"] == "low"])  
   
 return {  
 "board\_id": board\_id,  
 "board\_name": board["name"],  
 "total\_tasks": len(all\_tasks),  
 "status\_stats": {  
 "todo": todo\_count,  
 "in\_progress": in\_progress\_count,  
 "done": done\_count  
 },  
 "priority\_stats": {  
 "high": high\_priority\_count,  
 "medium": medium\_priority\_count,  
 "low": low\_priority\_count  
 }  
 }

**7.3 Task Management Routes**

**File: app/routers/tasks.py**

from fastapi import APIRouter, HTTPException, status, Query  
from typing import List, Optional  
from app.schemas.task import TaskCreate, TaskResponse, TaskUpdate, TaskMove, TaskAssign  
from app.models.mock\_data import (  
 get\_tasks\_by\_board, get\_task\_by\_id, create\_task,   
 update\_task, delete\_task, move\_task, search\_tasks,  
 get\_board\_by\_id  
)  
  
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="Lọc theo status"),  
 priority: Optional[str] = Query(None, description="Lọc theo priority"),  
 assigned\_to: Optional[int] = Query(None, description="Lọc theo user được gán")  
):  
 """Lấy danh sách tasks theo board với các filter options"""  
 # Kiểm tra board có tồn tại không  
 board = get\_board\_by\_id(board\_id)  
 if not board:  
 raise HTTPException(  
 status\_code=status.HTTP\_404\_NOT\_FOUND,  
 detail="Board không tồn tại"  
 )  
   
 tasks = get\_tasks\_by\_board(board\_id, status, priority)  
   
 # Filter theo assigned\_to nếu có  
 if assigned\_to is not None:  
 tasks = [task for task in tasks if task["assigned\_to"] == assigned\_to]  
   
 return [TaskResponse(\*\*task) for task in tasks]  
  
@router.post("/", response\_model=TaskResponse, status\_code=status.HTTP\_201\_CREATED)  
def create\_new\_task(task\_data: TaskCreate):  
 """Tạo task mới"""  
 # Kiểm tra board có tồn tại không  
 board = get\_board\_by\_id(task\_data.board\_id)  
 if not board:  
 raise HTTPException(  
 status\_code=status.HTTP\_400\_BAD\_REQUEST,  
 detail="Board không tồn tại"  
 )  
   
 task\_dict = task\_data.dict()  
 task = create\_task(task\_dict)  
 return TaskResponse(\*\*task)  
  
@router.get("/{task\_id}", response\_model=TaskResponse)  
def get\_task\_detail(task\_id: int):  
 """Lấy chi tiết task"""  
 task = get\_task\_by\_id(task\_id)  
 if not task:  
 raise HTTPException(  
 status\_code=status.HTTP\_404\_NOT\_FOUND,  
 detail="Task không tồn tại"  
 )  
 return TaskResponse(\*\*task)  
  
@router.put("/{task\_id}", response\_model=TaskResponse)  
def update\_task\_info(task\_id: int, task\_update: TaskUpdate):  
 """Cập nhật thông tin task"""  
 task = get\_task\_by\_id(task\_id)  
 if not task:  
 raise HTTPException(  
 status\_code=status.HTTP\_404\_NOT\_FOUND,  
 detail="Task không tồn tại"  
 )  
   
 update\_data = task\_update.dict(exclude\_unset=True)  
 if update\_data:  
 updated\_task = update\_task(task\_id, update\_data)  
 return TaskResponse(\*\*updated\_task)  
   
 return TaskResponse(\*\*task)  
  
@router.patch("/{task\_id}/move", response\_model=TaskResponse)  
def move\_task\_status(task\_id: int, task\_move: TaskMove):  
 """Di chuyển task sang status/position mới"""  
 task = get\_task\_by\_id(task\_id)  
 if not task:  
 raise HTTPException(  
 status\_code=status.HTTP\_404\_NOT\_FOUND,  
 detail="Task không tồn tại"  
 )  
   
 moved\_task = move\_task(task\_id, task\_move.status.value, task\_move.position)  
 if not moved\_task:  
 raise HTTPException(  
 status\_code=status.HTTP\_500\_INTERNAL\_SERVER\_ERROR,  
 detail="Không thể di chuyển task"  
 )  
   
 return TaskResponse(\*\*moved\_task)  
  
@router.patch("/{task\_id}/assign", response\_model=TaskResponse)  
def assign\_task\_to\_user(task\_id: int, task\_assign: TaskAssign):  
 """Gán task cho user"""  
 task = get\_task\_by\_id(task\_id)  
 if not task:  
 raise HTTPException(  
 status\_code=status.HTTP\_404\_NOT\_FOUND,  
 detail="Task không tồn tại"  
 )  
   
 # Nếu có assigned\_to, kiểm tra user có tồn tại không (optional)  
 # if task\_assign.assigned\_to:  
 # user = get\_user\_by\_id(task\_assign.assigned\_to)  
 # if not user:  
 # raise HTTPException(400, "User không tồn tại")  
   
 updated\_task = update\_task(task\_id, {"assigned\_to": task\_assign.assigned\_to})  
 return TaskResponse(\*\*updated\_task)  
  
@router.delete("/{task\_id}")  
def delete\_task\_by\_id(task\_id: int):  
 """Xóa task"""  
 task = get\_task\_by\_id(task\_id)  
 if not task:  
 raise HTTPException(  
 status\_code=status.HTTP\_404\_NOT\_FOUND,  
 detail="Task không tồn tại"  
 )  
   
 if delete\_task(task\_id):  
 return {  
 "message": "Xóa task thành công",  
 "deleted\_task\_id": task\_id  
 }  
   
 raise HTTPException(  
 status\_code=status.HTTP\_500\_INTERNAL\_SERVER\_ERROR,  
 detail="Không thể xóa task"  
 )  
  
@router.get("/search/", response\_model=List[TaskResponse])  
def search\_tasks\_by\_keyword(  
 q: str = Query(..., min\_length=1, description="Từ khóa tìm kiếm"),  
 board\_id: Optional[int] = Query(None, description="Tìm kiếm trong board cụ thể")  
):  
 """Tìm kiếm tasks theo từ khóa trong title hoặc description"""  
 if board\_id:  
 # Kiểm tra board có tồn tại không  
 board = get\_board\_by\_id(board\_id)  
 if not board:  
 raise HTTPException(  
 status\_code=status.HTTP\_404\_NOT\_FOUND,  
 detail="Board không tồn tại"  
 )  
   
 matching\_tasks = search\_tasks(q, board\_id)  
 return [TaskResponse(\*\*task) for task in matching\_tasks]

**8. Bước 5: Main Application**

**File: app/main.py**

from fastapi import FastAPI

from fastapi.middleware.cors import CORSMiddleware

from app.routers import users, boards, tasks

# Tạo FastAPI app instance

app = FastAPI(

title="Kanban TODO API",

version="1.0.0",

description="API để quản lý công việc theo mô hình Kanban board",

docs\_url="/docs",

redoc\_url="/redoc"

)

# Thêm CORS middleware để frontend có thể gọi API

app.add\_middleware(

CORSMiddleware,

allow\_origins=["\*"], # Trong production nên chỉ định cụ thể domain

allow\_credentials=True,

allow\_methods=["\*"],

allow\_headers=["\*"]

)

# Include các routers

app.include\_router(users.router)

app.include\_router(boards.router)

app.include\_router(tasks.router)

# Root endpoint

@app.get("/")

def read\_root():

"""Welcome endpoint"""

return {

"message": "Chào mừng đến với Kanban TODO API!",

"version": "1.0.0",

"docs\_url": "/docs",

"redoc\_url": "/redoc",

"endpoints": {

"users": "/users",

"boards": "/boards",

"tasks": "/tasks"

}

}

# Health check endpoint

@app.get("/health")

def health\_check():

"""Kiểm tra API có hoạt động không"""

return {

"status": "healthy",

"service": "Kanban TODO API",

"version": "1.0.0"

}

if \_\_name\_\_ == "\_\_main\_\_":

import uvicorn

# The FastAPI application object is defined in this module (main.py),

# so point uvicorn to "main:app" when running from the package root.

uvicorn.run("main:app", host="0.0.0.0", port=8000, reload=True)

**9. Chạy và Test API**

**9.1 Khởi động server**

# Từ thư mục gốc project  
uvicorn app.main:app --reload  
  
# Hoặc   
python -m uvicorn app.main:app --reload

Nếu thành công, sẽ thấy:

INFO: Uvicorn running on http://127.0.0.1:8000 (Press CTRL+C to quit)  
INFO: Started reloader process  
INFO: Started server process

**9.2 Truy cập Swagger UI**

Mở browser và vào: [**http://127.0.0.1:8000/docs**](http://127.0.0.1:8000/docs)

**Test Users API:**

POST /users/ - Tạo user mới

GET /users/ - Xem danh sách users

GET /users/{user\_id} - Xem chi tiết user

PUT /users/{user\_id} - Cập nhật user

**Test Boards API:**

POST /boards/?owner\_id=2 - Tạo board mới

GET /boards/ - Xem danh sách boards

GET /boards/{board\_id} - Xem chi tiết board + tasks

PUT /boards/{board\_id} - Cập nhật board

**Test Tasks API:**

POST /tasks/ - Tạo task mới

GET /tasks/?board\_id=1 - Xem tasks của board

PATCH /tasks/{task\_id}/move - Di chuyển task

PATCH /tasks/{task\_id}/assign - Gán task cho user

## Git workflow và teamwork

**Bước 1: Git commit 🡪 Local repository**

git add .

git status

git commit -m "feature: Initialize FastAPI TODO API with CRUD operations"

**Bước 2: Connect to GitHub**

1. Tạo repository trên GitHub.com (tên: todo-project)
2. Connect local với remote:

git remote add origin https://github.com/YOUR\_USERNAME/todo-api-session4.git

git branch -M main

git push -u origin main

**Bước 3: Practice branching**

# Tạo feature branch

git checkout -b feature/add-categories

# Thêm field category vào Task model (in main.py)

# ... make some changes ...

# Commit changes

git add .

git commit -m "feat: Add categories support to tasks"

# Merge back to main

git checkout main

git merge feature/add-categories

# Push updated main

git push origin main

# Clean up branch

git branch -d feature/add-categories

# Tips và Lỗi thường gặp

## Lỗi môi trường và cài đặt

# Lỗi: ModuleNotFoundError: No module named 'fastapi'

# Giải pháp: Kiểm tra virtual environment

which python # Phải trỏ đến venv

pip install fastapi uvicorn

# Lỗi: uvicorn command not found

# Giải pháp: Cài đặt lại hoặc dùng python -m

python -m uvicorn app.main:app --reload

## Lỗi code thường gặp

# Lỗi: Quên async

@app.get("/")

def root(): # ❌ Nên dùng async def

return {"message": "hello"}

@app.get("/")

async def root(): # ✅ Đúng

return {"message": "hello"}

# Lỗi: Sai indent

@app.get("/")

async def root():

return {"message": "hello"} # ❌ Sai indent

# Lỗi: Quên return

@app.get("/")

async def root():

{"message": "hello"} # ❌ Thiếu return

## Lỗi server và network

# Lỗi: Port 8000 already in use

# Giải pháp: Dùng port khác

uvicorn app.main:app --reload --port 8001

# Lỗi: Cannot connect to server

# Kiểm tra: Server có đang chạy không?

# Kiểm tra: URL đúng chưa (127.0.0.1:8000)

## Debug tips

# Thêm logging để debug

import logging

logging.basicConfig(level=logging.INFO)

logger = logging.getLogger(\_\_name\_\_)

@app.post("/tasks")

async def create\_task(task: TaskCreate):

logger.info(f"Creating task: {task.title}")

# ... rest of code

# Bài tập thực hành

## Bài tập cơ bản

* Test tất cả APIs: Sử dụng Swagger UI để test từng endpoint
* Tạo dữ liệu mẫu: Tạo 3 users, 5 boards, 10 tasks qua API
* Test error cases:
  + Tạo user với username trùng
  + Tạo task với board\_id không tồn tại
  + Cập nhật user với ID không tồn tại

## Bài tập nâng cao

1. Thêm validation:

* Email phải có format đúng
* Board name không được trùng với boards của cùng 1 user
* Task title không được chứa ký tự đặc biệt

Mở rộng chức năng:

* Thêm field created\_by cho tasks
* Thêm endpoint để copy task sang board khác
* Thêm endpoint lấy tasks theo date range

Cải thiện mock data:

* Thêm function để export/import data từ JSON file
* Thêm seed data với nhiều user và board hơn