# Demo

## Phát biểu bài toán

### Yêu cầu bài toán:

Mỗi đơn vị kinh doanh phần mềm muốn xây dựng một ứng dụng web todo list cho việc theo dõi và giám sát các công việc được giao. Tuy nhiên, cơ sở dữ liệu về công việc này lại đang được quản lí bởi một bên khác nên họ không thể truy cập vào CSDL trên. Vì vậy cả hai bên thống nhất xây dựng một webservice có khả năng nhận các yêu cầu của các web application và gửi lại các kết quả một cách nhanh chóng và chính xác nhất.

### Hướng giải quyết

Đầu tiên, webservice sẽ được kết nối với database và có 5 API chính với các nhiệm vụ khác nhau:

**GET** /todo: lấy toàn bộ thông tin từ database

**POST** /todo: thêm một task vào database

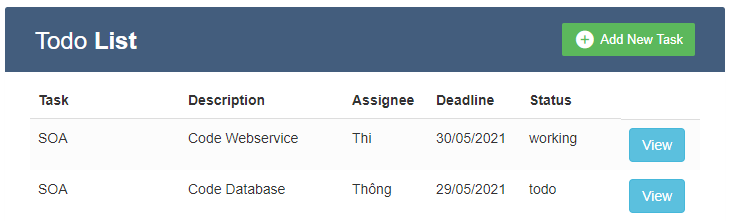
**GET /**todo/<id>: lấy thông tin của task có #ID = <id>

**PUT /**todo/<id>: sửa thông tin của task có #ID = <id>

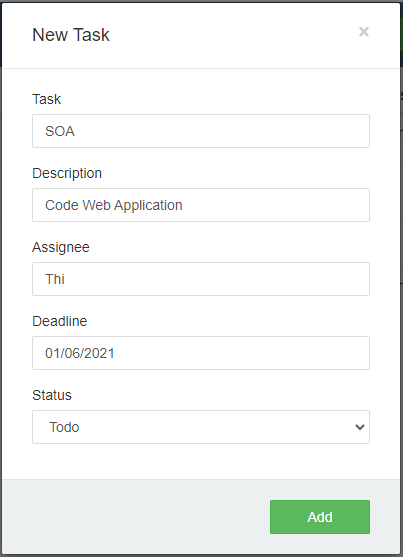
**DELETE /**todo/<id>: xóa task có #ID = <id>

Tiếp theo, ở web application sẽ có những chức năng như sau:

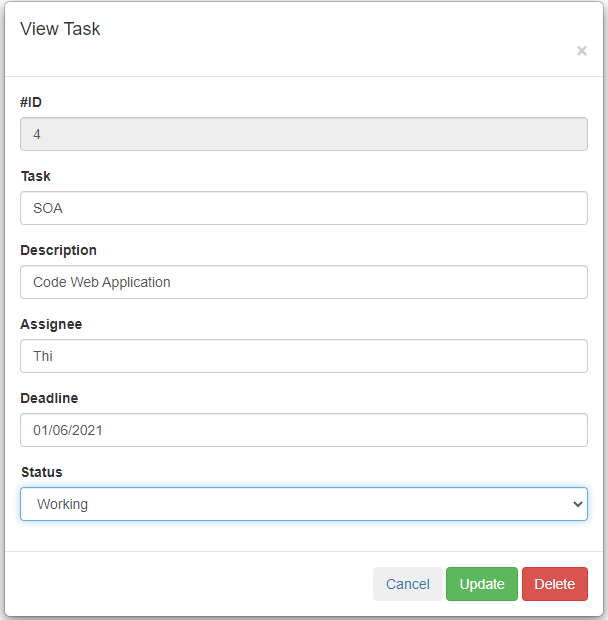
* Khi truy cập vào homepage: Backend sẽ gửi yêu cầu **GET** /todo về webservice bằng thư viện requests, sau khi nhận được dữ liệu sẽ đổ lên frontend.



* Khi thực hiện việc thêm task: User điền các thông tin vào form. Sau khi submit, một yêu cầu **POST** /todo cùng với gói tin có dạng json chứa toàn bộ thông tin trong form sẽ được gửi đến webservice. Sau khi nhận request trên, webservice tiến hành thêm task vào database.



* Khi muốn xem chi tiết một task: Backend sẽ gửi yêu cầu **GET** /todo/<id> và nhận lại dữ liệu tương ứng
* Tương tự trong việc sửa và xóa task, lần lượt các yêu cầu PUT /todo/id hoặc DELETE /todo/id sẽ được gửi đến webservice



## Các bước thực hiện

### Tạo database

### a. Giới thiệu DynamoDB

DynamoDB là môt dịch vụ quản lý cơ sở dữ liệu NoSQL của AWS, có khả năng đáp ứng hiệu suất cao và nhanh kèm theo khả năng mở rộng. Khác với có dạng cơ sở dữ liệu có cấu trúc khác, thành phần chính cần khai báo khi sử dụng DynamoDB chỉ là khóa chính và các đặc điểm của khóa chính, các record sẽ được lưu dưới dạng json với một cấu trúc bất kí, miễn là thỏa mãn các yêu cầu của khóa chính.

Các thư viện để có thể thực hiện các tương tác với dynamoDB trên localhost là localstack và boto3.

**b. Cài đặt các thư viện**

./ boto3: pip install boto3

./ localstack: pip install localstack

**c. Tạo database với python**

import boto3

#Connect with boto3 at localhost:4566

dynamodb = boto3.resource('dynamodb', endpoint\_url = "http://localhost:4566")

#Module create database with 2 parameter: name, key

def create\_table(name, key):

    table = dynamodb.create\_table(

    TableName=name,

    KeySchema=[

        {

            'AttributeName': key,

            'KeyType': 'HASH'

        }

    ],

    AttributeDefinitions=[

             {

            'AttributeName': 'id',

            'AttributeType': 'S'

        }

    ],

    ProvisionedThroughput={

        'ReadCapacityUnits': 5,

        'WriteCapacityUnits': 5

    }

    )

if \_\_name\_\_ == "\_\_main\_\_":

    # Create database todo\_list

    database = "todo\_list"

    key = "id"

    create\_table(database, key)

### Tạo query\_db.py làm module tương tác với database

import boto3

dynamodb = boto3.resource('dynamodb', endpoint\_url = "http://localhost:4566")

database\_name = "todo\_list"

database\_count = "todo\_list\_COUNT"

table\_todo = dynamodb.Table(database\_name)

count\_todo = dynamodb.Table(database\_count)

def get\_all\_task():

    data = []

    scanResponse = table\_todo.scan(TableName=database\_name)

    items = scanResponse['Items']

    for item in items:

        data.append(item)

    return data

def create\_task(new\_task):

    id = str(count\_todo.item\_count + 1)

    count\_item = {"id": id}

    count\_todo.put\_item(Item=count\_item)

    new\_task["id"] = id

    table\_todo.put\_item(Item=new\_task)

    return get\_all\_task()

def get\_task\_by\_id(id):

    res = table\_todo.get\_item(Key = {'id': id}, ConsistentRead=True)["Item"]

    return res

def update\_task(update\_info, id):

    try:

        table\_todo.update\_item(Key = {"id": id},

                    UpdateExpression = "set task=:t",

                    ExpressionAttributeValues = {':t': update\_info["task"]})

        print("Updated task to ", update\_info["task"])

    except:

        print("---Can't update task")

    try:

        table\_todo.update\_item(Key = {"id": id},

                    UpdateExpression = "set description=:d",

                    ExpressionAttributeValues =

{':d': update\_info["description"]})

        print("Updated description to ", update\_info["description"])

    except:

        print("---Can't update description")

    try:

        table\_todo.update\_item(Key = {"id": id},

                    UpdateExpression = "set progress=:p",

                    ExpressionAttributeValues =

{':p': update\_info["progress"]})

        print("Updated progress to ", update\_info["progress"])

    except:

        print("---Can't update progress")

    try:

        table\_todo.update\_item(Key = {"id": id},

                    UpdateExpression = "set deadline=:de",

                    ExpressionAttributeValues =

{':de': update\_info["deadline"]})

        print("Updated deadline to ", update\_info["deadline"])

    except:

        print("---Can't update deadline")

    try:

        table\_todo.update\_item(Key = {"id": id},

                    UpdateExpression = "set assignee=:as",

                    ExpressionAttributeValues =

{':as': update\_info["assignee"]})

        print("Updated assignee to ", update\_info["assignee"])

except:

        print("---Can't update assignee")

    return get\_all\_task()

def delete\_task(id):

    table\_todo.delete\_item(Key = {'id': id})

    return get\_all\_task()

### Tạo webservice.py để tạo một webservice

1. **Khởi tạo webservice.py**

from flask import Flask, request, jsonify

from flask\_cors import CORS

import query\_db as db

app = Flask(\_\_name\_\_)

CORS(app)

cors = CORS(app, resources = {

    r"/\*": {

        "origin": "\*"

    }

})

@app.route("/todo")

def get\_all\_task():

    return jsonify(db.get\_all\_task())

@app.route("/todo", methods = ["POST"])

def create\_task():

    return jsonify(db.create\_task(request.get\_json()))

@app.route("/todo/<string:id>")

def get\_task\_by\_id(id):

    return jsonify(db.get\_task\_by\_id(id))

@app.route("/todo/<string:id>", methods = ["PUT"])

def update\_task(id):

try:

        return jsonify(db.update\_task(request.get\_json(),id))

    except:

        return "Can't put"

@app.route("/todo/<string:id>", methods = ["DELETE"])

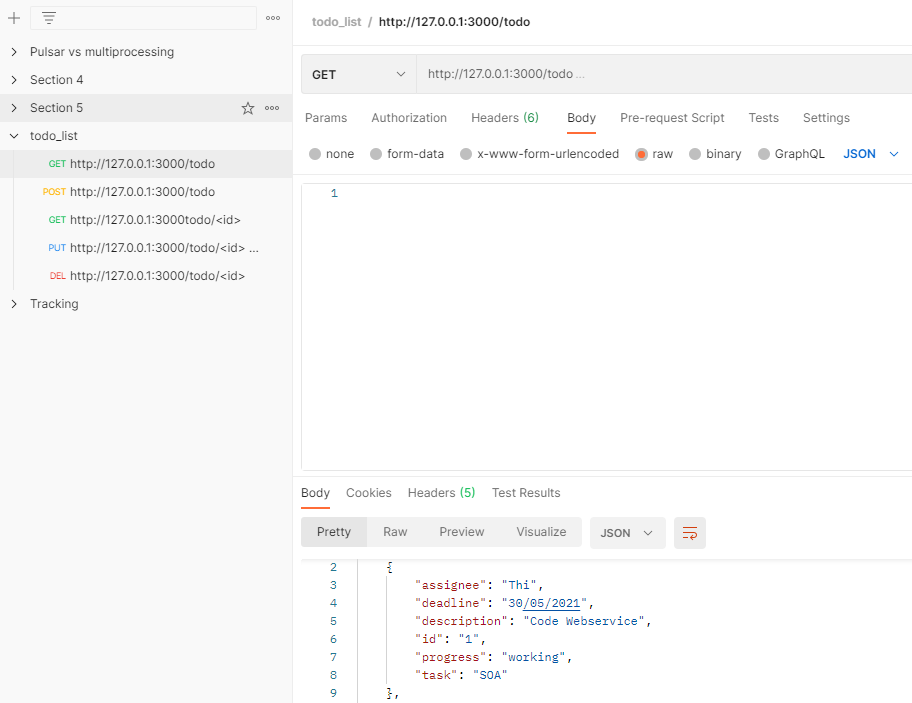
def delete\_task(id):

    return jsonify(db.delete\_task(id))

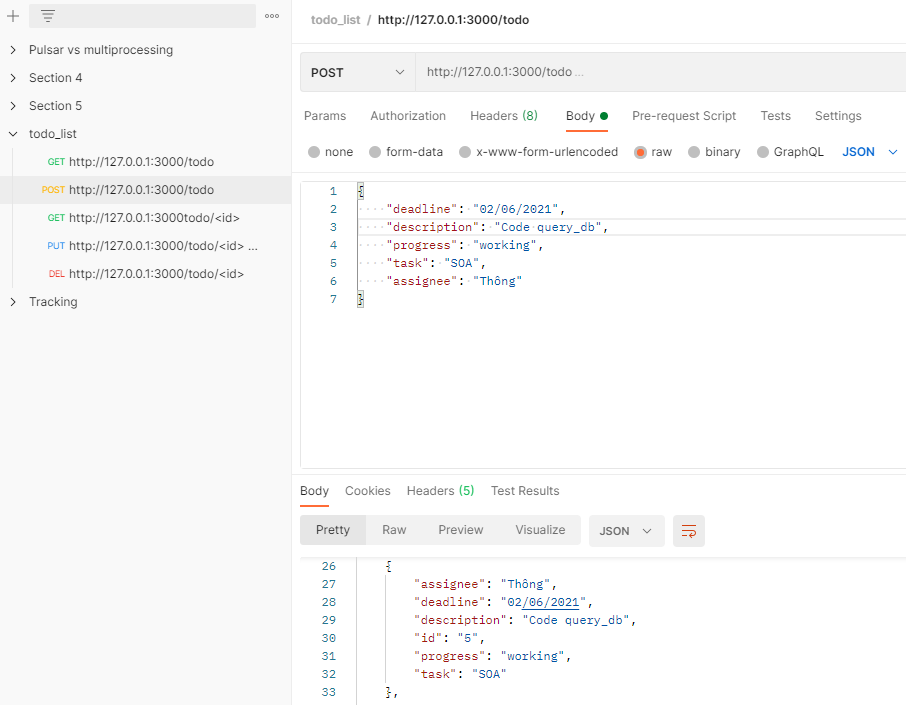
app.run(port=3000)

1. **Kiểm tra API bằng Postman**

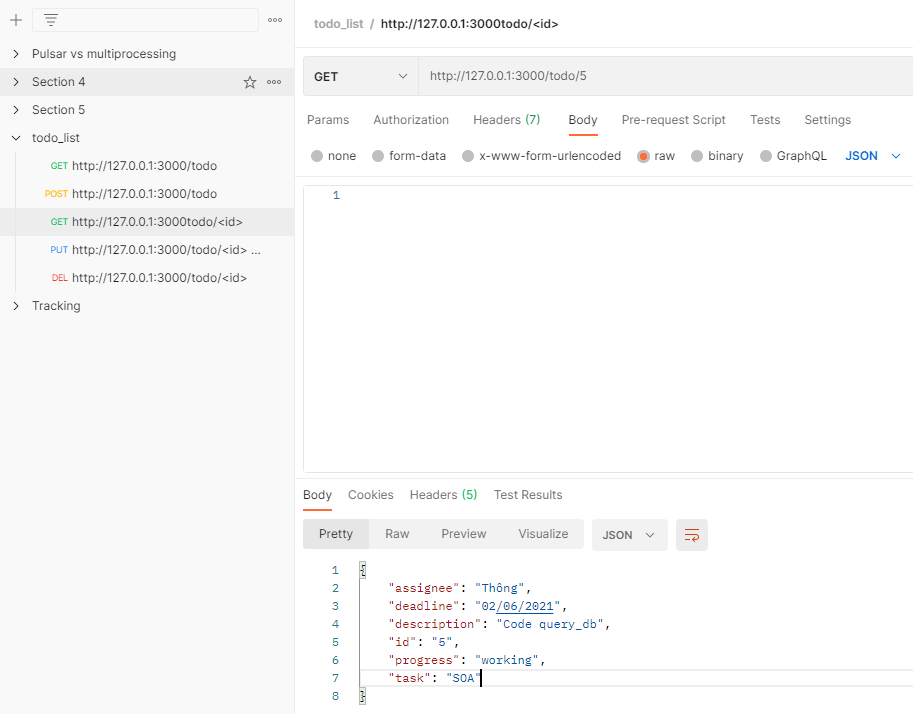
Với **GET** /todo: lấy toàn bộ database



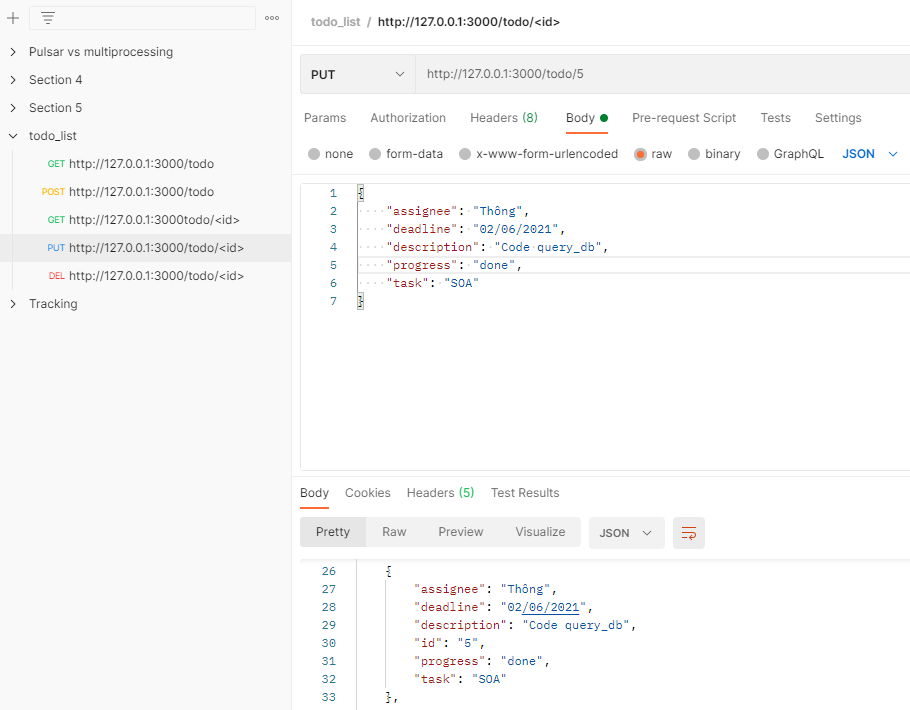
Với **POST** /todo: thêm task vào database



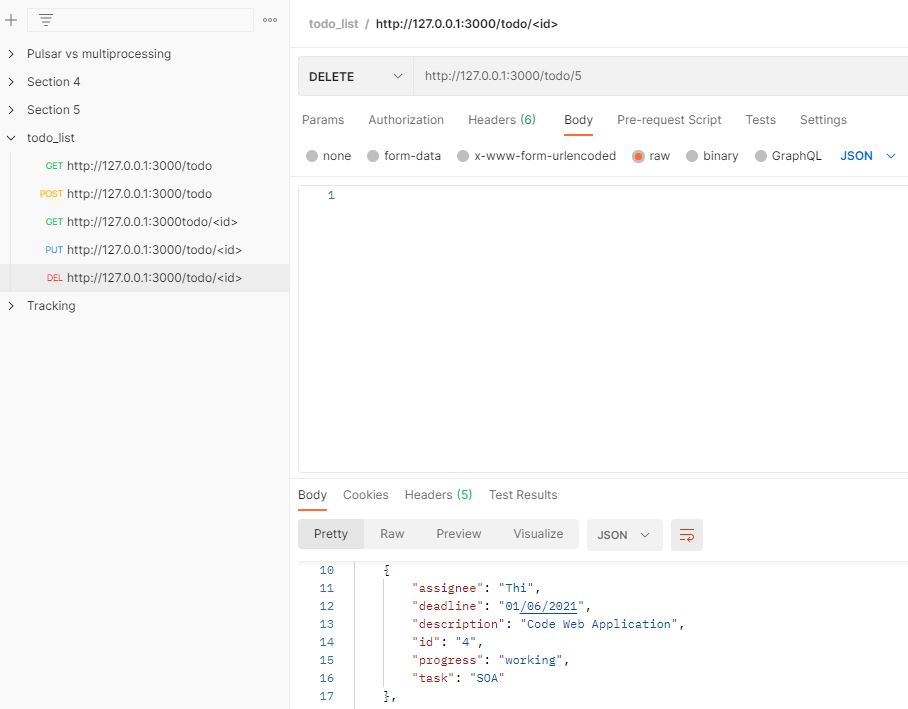
Với **GET** /todo/<id>: Lấy thông tin của task có id = 5



Với **PUT** /todo/<id>: Sửa thông tin của task có id = 5



Với **DELETE** /todo/<id>: Xóa task có id = 5



### Xây dựng web application

1. **Phần Backend**

Công nghệ sử dụng: Flask với RestAPI

1. **Phần Frontend**

Công nghệ sử dụng: HTML, CSS, javascript, ajax