Python Django Programming

Lập trình Python nâng cao

Pisces Kibo

Advanced Python Programming

Mục lục

Bài 1: Xử lý các file Json, Base64, CSV

1. **Làm việc với Json:**

+ JavaScript Object Notation (JSON) là thư viện chuyển đổi từ các object của Python sang các dạng dữ liệu khác phù hợp → định dạng phổ biến

- Dùng để lưu trữ và thể hiện các dữ liệu có cấu trúc
- Sử dụng để truyền và nhận dữ liệu giữa ứng dụng web và web server
- Thao tác JSON dưới dạng chuỗi hoặc lưu đối tượng JSON vào trong file
- Dùng để response cho API

a) Cú pháp JSON:

```
{
    "Key1": "Value1",
    "Key2": "Value2",
    ...
}
```

→ Giống với kiểu dữ liệu Dictionary trong Python

b) Chuyển đổi từ các Object sang kiểu dữ liệu khác phù hợp:

- JSON dùng để mã hóa (tuần tự hóa) và giải mã dữ liệu
- Nó sẽ chuyển đổi dữ liệu thành một chuỗi byte có thể được lưu trữ và truyền giữa các máy chủ và các ứng dụng web

- Cú pháp chuyển đổi qua lại:

```
import json

dict = {
    "Key1": "Value1",
    "Key2": "Value2",
    ...
}

# Chuyển đổi mã hóa json từ dictionary sang string JSON (ghi file để gửi đi)
newJSON = json.dumps(dict)

# Giải mã lại json từ string sang dictionary (đọc file)
newDict = json.loads(newJSON)
```

- Cú pháp tạo và xử lý file JSON:

```
import json

dict = {
    "Key1": "Value1",
    "Key2": "Value2",
    ...
}
```

```
# Tạo và ghi thành file JSON → mã hóa chuỗi ghi trong tệp
with open("path/file.json", mode = "w") as file:
   json.dump(sinh_vien, file)

# Load lại file JSON và chuyển về Dictionary
file = open("path/file.json ")
content = json.load(file)
```

- Từ khóa quan trọng:
  - dumps(object, indent, sort_keys):
    - indent dùng để chỉ định kích thước thụt dòng (number)
    - sort_keys sắp xếp từ điển theo từ khóa nếu sort_keys là True (mặc định là False)
  - loads() → trả về một đối tượng dictionary chứa dữ liệu được chứa trong JSON string (thao tác giải mã)

- Bảng chuyển đổi qua lại giữa JSON và Python:

JSON	Python
object	dict
'array |list`	
string	str
number (int)	int
number (real)	float
true	True
false	False
null	None

- Lưu ý với JSON Dict:
  - Nếu không tồn tại key:
    - JSON luôn trả về Null
    - Kiểu Dict:
      - Sẽ báo lỗi khi gọi dict_name[key]
      - Nếu gọi dict.get(key) sẽ trả về None

  - So sánh Dict và JSON:
    - Dictionary trong Python được hiểu như một container chứa các dữ liệu và thao tác với các dữ liệu
    - JSON về bản chất dùng để mã hóa và giải mã dữ liệu, tức là nó sẽ chuyển đổi dữ liệu thành một chuỗi byte có thể lưu trữ và truyền qua mạng, dùng để response cho API

2. Làm việc với CSV:

- CSV (Comma Separated Values) là định dạng tệp đơn giản được sử dụng để lưu trữ dữ liệu dạng bảng văn bản (bảng tính, CSDL)
- Mỗi dòng tệp là một bản ghi và các giá trị được phân tách bằng dấu phẩy (các cột cách nhau bằng dấu phẩy và không để khoảng trắng giữa các cột)

- Tệp file.csv:

```
Col1.Col2,Col3,Col4,...
No1,No2,No3,No4,..
...
```

- Cú pháp câu lệnh:

```python
# Xuất thư viện file --> dạng chuẩn
import csv

# ĐỌC FILE
# Mở file.csv --> List
with open("path/fileName.csv", mode = "r") as file:
    content = csv.reader(file)
    for line in content:
        print(line)

# Mở file.csv --> Dict (dòng đầu tiên mặc định là các key)
with open("Session08/fileName.csv", mode = "r") as file:
    content = csv.DictReader(file)
    for line in content:
        print(line)


# GHI FILE dạng writerow
with open("path/fileName.csv", mode = "a") as file:
    writer = csv.writer(file,delimiter=',',quotechar='"',quoting=csv.QUOTE_MINIMAL)
    writer.writerow(['Col1', 'Col2', 'Col3', ...])

# GHI FILE dạng writerows
with open("path/fileName.csv", mode = "w") as file:
    writer = csv.writer(file,delimiter=",",quotechar='"',quoting=csv.QUOTE_MINIMAL)

    # Cho phép ghi cả một list[list] đồng thời vào
    writer.writerows(data)          # data = [list1, list2, ...]
```

3. Làm việc với Base64:

- Là phương thức convert dạng mã hóa 2 chiều từ binary (ký tự mã ASCII) sang string để có thể gửi đi được trong network một cách dễ dàng

- Cú pháp:
  - Xuất thư viện → import base64

- Mã hóa nội dung bằng encode → encoded = base64.b64encode(b'string')
- Giải mã nội dung bên trong bằng decode → decoded = base64.b64decode(encoded)

Bài 2: Gửi Email tự động với thư viện requests

1. **Thư viện requests:**
   - Thư viện requests sử dụng để gửi yêu cầu HTTP qua các dịch vụ web API
   - Cài đặt → ***pip install requests***
   - Sử dụng module → ***import requests***
   - Kiểm tra trạng thái của link → statusCode = ***response.status_code***
   - Chuyển đổi giao diện web về json
     → ***linkWeb = 'https://link.com/posts/pathID&sort=True/comments/'***
     (thêm /posts vào cuối link để xem được file json của web)
     pathID là phần tử trong dict và comments là lấy tất cả chú thích code

   - Các phương thức HTTP:
     - GET → lấy dữ liệu từ một nguồn xác định
       - Cú pháp:
         - ***requests.get(url, params = {key: value}, args)***
           - url → đường link dạng chuỗi nháy đơn
           - params → chuỗi truy vấn, có thể thêm trực tiếp vào url
           - args → có hoặc thêm các tham số

       - Cách các viết theo parameter:
         - Cách 1: (viết theo parameter lồng)
           - url = 'https://link.com/comments?key1=value1&key2=value2'
           - x = requests.get(url)

         - Cách 2: (viết theo từng parameter)
           - url = 'https://link.com/comments'
           - params = {key1: value1, key2: value2}
           - x = requests.get(url, params)

       - Ví dụ:

```
import requests

x = requests.get('https://linkweb.com')
print(x.status_code)

# Chuyển về json thông tin về đường link đó
If x.status_code == 200:
    x.json()
else:
    print(f"Error code: {x.status_code}")
```

```
>> import requests

>> link = 'https://api.github.com/events'

>> response = requests.get(link)

>> response.status_code

200

>> response.json()[0]['actor']

{'avatar_url': 'https://avatars.githubusercontent.com/u/45685134?',
 'display_login': 'trinhdinhnam',
 'gravatar_id': '',
 'id': 45685134,
 'login': 'trinhdinhnam',
 'url': 'https://api.github.com/users/trinhdinhnam'}
```

- POST → thêm mới và gửi yêu cầu đến máy chủ web (insert)
  - Thêm dữ liệu mới chưa tồn tại vào
  - Cú pháp:
    - *requests.post(url, data={key: value}, json={key: value}, args)*
  - Ví dụ:

```
import requests

url = 'https://www.w3schools.com/python/demopage.php'
myobj = {'somekey': 'somevalue'}

x = requests.post(url, json = myobj)

print(x.text)

# https://www.w3schools.com/python/ref_requests_post.asp
```

- PUT → sửa và gửi yêu cầu đến webserver
  - Sửa một đối tượng đã tồn tại
  - Cú pháp:

```
import requests

json = {
    "key1": "update_value1",
    "key2": "update_value2",
    ...
```

```
}
response = requests.put(url, json = json)
print(response.json())
```

- Ví dụ:

```
import requests

json = {
  "my_key": "Hello, World!"
}

response = requests.put("https://httpbin.org/put", json=json)

print(response.json()["json"])
```

- DELETE → gửi yêu cầu đến webserver
  - Cú pháp:
    - ***requests.delete(url, args)***
  - Ví dụ:

```
import requests

x = requests.delete('https://w3schools.com/python/demopage.php')

print(x.text)

# https://www.w3schools.com/python/ref_requests_delete.asp
```

- Các trạng thái HTTP Response status code:
  - 200: OK
  - 400: Bad request
  - 401: Unauthorized
  - 403: Forbidden
  - 404: Not found
  - 409: Conflict
  - 500: Internal server error

2. Sending EMAIL bằng giao thức SMTP:

- SMTP (Simple Mail Transfer Protocol) là giao thức ở lớp ứng dụng (trên TCP) được sử dụng để giao tiếp với mail servers từ các dịch vụ bên ngoài)
- SMTP là giao thức vận chuyển (không thể nhận được email từ nó, chỉ có thể gửi mail thông qua SMTP)
- Tạo kết nối smtp không an toàn → kết nối an toàn được bảo vệ thông qua SSL/TLS thường gọi là smtps (tương tự giao thức an toàn trên internet là https)

- Các loại cách thức gửi mail:

- L1: Tạo kết nối smtp bằng thư viện smtplib (port mặc định là 25 nhưng port gửi email là 587)

```
import smtplib

smtp_server = "smtp.example.com"
smtp_port = 587
smtp_connection =
smtplib.SMTP(host=smtp_server,
port=smtp_port)
```

```
import smtplib

try:
    server = smtplib.SMTP('smtp.gmail.com', 587)
    server.ehlo()
except:
    print ('Something went wrong...')
```

- L2: Tạo kết nối smtp an toàn bằng starttls()

```
import smtplib

try:
    server = smtplib.SMTP('smtp.gmail.com', 587)
    server.ehlo()
    server.starttls()
    # ...send emails
except:
    print 'Something went wrong...'
```

- L3: Tạo kết nối smtp an toàn ngay từ đầu:

```
import smtplib

try:
    server_ssl = smtplib.SMTP_SSL('smtp.gmail.com', 465)
    server_ssl.ehlo()   # optional
    # ...send emails
except:
    print ('Something went wrong...')
```

3. Các thành phần của Email Python:

- From: you@gmail.com
- To: me1@gmail.com, me2@gmail.com
- Subject: Tiêu đề đề chính của email
- Body (Message) → nội dung chính của email

- Viết lệnh cú pháp:

sent_from = 'you@gmail.com' to = ['me1@gmail.com', 'me2@gmail.com'] subject = 'Tiêu đề của Email' body = 'Nội dung bên trong email'	email_text = """\ From: %s To: %s Subject: %s %s """ % (sent_from, ", ".join(to), subject, body)

  - Khởi tạo kết nối → server = smtplib.SMTP('smtp.gmail.com', 587)
  - Nâng cấp kết nối an toàn → server.starttls()
  - Login → server.login(smtp_login_email, smtp_login_password)
  - Send email → server.sendmail(email_me, email_list, msg.as_string())
  - Ngắt kết nối → server.quit()

- Chuyển thành các trường trong python:
  - Định nghĩa message:

```
msg = MIMEMultipart()
message = "Content"
msg.attach(message)
msg['Subject'] = subject
msg['From'] = email_me
msg['To'] = email_to
# msg['CC'] = cc    # String: email1, email2, …
```

- Đính kèm file:

```
with open(attach_pdf, "rb") as file:
    attach = MIMEApplication(fiile.read(),_subtype="pdf")
    attach.add_header('Content-Disposition','attachment', filename=str(attach_pdf))
    msg.attach(attach)
```

4. Tạo hàm gửi email:

- Xuất thư viện:
  - import smtplib
  - from email.mime.text import MIMEText

- Lấy mật khẩu App Passwork từ Email (https://support.google.com/mail/answer/185833?hl=en)
  - B1: Vào tài khoản Gmail
  - B2: Chọn **Bảo mật** và Xác minh 2 bước
  - B3: Ở cuối trang chọn Mật khẩu ứng dụng (**App passwords**)
  - B4: Tạo mật khẩu ứng dụng

- Hàm gửi email:

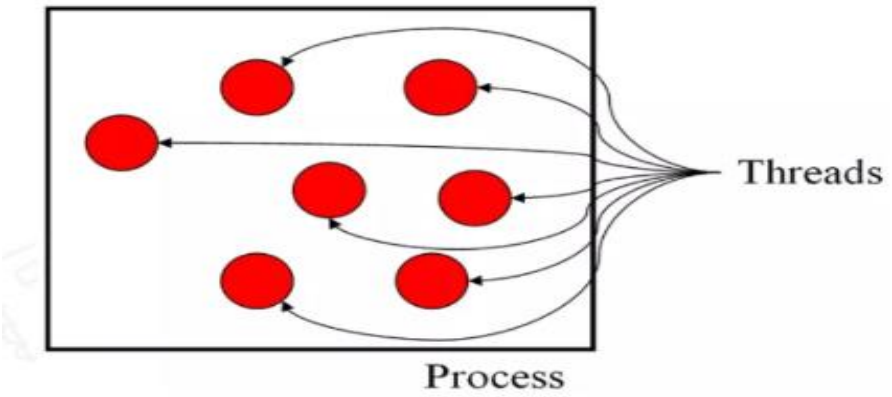
```
def sending_email():
    # Nội dung email
    subject = "Tiêu đề email"
    email_from = "kibo0603@gmail.com"
    email_to = "taquangtung2003@gmail.com"

    # Gửi email mà tất cả mọi người biết những ai
    emai_cc = ["me1@gmail.com"]
    # Gửi email mà chỉ có mình bạn mới biết
    email_bcc = ["me2@gmail.com"]

    body = """
        This is the email's message Body
    """

    message = MIMEText(body)
    message["Subject"] = subject
    message["From"] = email_from
    message["To"] = email_to
    message["CC"] = emai_cc
    message["BCC"] = email_bcc

    # Khởi tạo kết nối
```

```
    server = smtplib.SMTP("smtp.gmail.com", 587)
    server.starttls()       # Bật vùng an toàn

    # Login tài khoản
    smtp_email_login = "kibo0603@gmail.com"
    smtp_password_login = "lwuqjkkziuutymfj"   # Mật khẩu ẩn App Password
    server.login(smtp_email_login, smtp_password_login)

    # Gửi email
    server.sendmail(email_from, [email_to], message.as_string())

    # Thoát exit
    server.quit()
    print("Sent successful")

# Chạy chương trình
if __name__ == "__main__":
    sending_email()
```

- Phân biệt CC và BCC: (luôn là dạng list)
    - To → người nhận mặc định (dạng list)
    - CC → người nhận thứ n có thể nhìn thấy tất cả người nhận khác (không bí mật)
    - BCC → người nhận chỉ nhìn thấy mỗi email mình (bảo mật cao)

Bài 3: Xử lý đa luồng và tiến trình Threading

1. **Process và Thread:**
   + Tiến trình Process là quá trình hoạt động của một ứng dụng, chứa đựng thông tin tài nguyên, trạng thái thực hiện của chương trình
   + Luồng Thread là đơn vị cơ bản trong CPU:
      - Một luồng sẽ chia sẻ với các luồng khác trong cùng Process về thông tin data, các dữ liệu của mình
      - Việc tạo ra Thread giúp cho các chương trình có thể chạy được nhiều công việc cùng lúc
      - Một thread có thể làm bất cứ nhiệm vụ gì của một process có thể làm
      - Thread nằm trong một process dùng một không gian bộ nhớ giống nhau (Process thì không) → dễ dàng hơn, tài nguyên dùng ít hơn



   → Cho các hàm function1, function2, ...
      - Process sẽ chạy lần lượt tuần tự hết từng hàm một (mất nhiều thời gian hơn)
      - Thread chạy đồng thời song song các hàm riêng biệt với nhau (tối thiểu hóa thời gian hơn) → chạy song song nhiều luồng cùng lúc

2. **Xử lý đa luồng MultiThreading:**
   - Chương trình đa luồng chứa hai hoặc nhiều phần mà có thể chạy đồng thời
   - Mỗi phần có thể xử lý tác vụ khác nhau tại cùng một thời điểm
   - Module thread và threading để bắt đầu một thread mới cũng như tác vụ khác khi lập trình đa luồng đều có vòng đời chung là bắt đầu chạy và kết thúc
   - Một Thread bị ngắt (interrupt) hoặc tạm dừng (sleep) trong khi các Thread khác đang chạy
   - Cấu trúc:
      - Xuất thư viện thời gian → import time
      - Xuất thư viện đa luồng → import threading
      - Khởi tạo các hàm function1, function2, ...
      - Gọi đối tượng Thread:

```
t1 = time.time()

# multithread
thread1 = threading.Thread(target=function1, args=(val1,val2,))
thread2 = threading.Thread(target=function2, args=(val1,val2,))
...
```

```
# start – chạy đa luồng
thread1.start()
thread2.start()

# join --> kết thúc chương trình đa luồng
thread1.join()
thread2.join()

t2 = time.time()

# Khoảng chênh lệch thời gian
print(f"Time spend: {t2 - t1}")
```

3. **<u>Đa luồng trong kế thừa hướng đối tượng:</u>**
   - Mô phỏng thời gian chạy hoạt động song song của các đối tượng:
     - Xuất thư viện: threading và time
     - Tạo đối tượng lớp class:

```
import threading
import time

class ObjectDoiTuong(threading.Thread):
    def __init__(self, arg1, arg2,...):
        super().__init__()
        self.arg1 = arg1
        self.arg2 = arg2
        self.arg3 = arg3

    # Phương thức mặc định trong Thread
    def run(self):
        # Các hành động của Thread chạy trong run() mặc định
        # Chạy các function phía ngoiaf vào trong run()

    def function1(val1,...):
    def function2(val1,...):
    ...


if __name__ == "__main__":
    # Khởi tạo đối tượng
    Obj1 = ObjectDoiTuong(arg1, arg2, ...)
    Obj2 = ObjectDoiTuong(arg1, arg2, ...)
    ...

    # start
    Obj1.start()
    Obj2.start()
    ...
```

```
# join
Obj1.join()
Obj2.join()
```

- Ta có thể chọn cặp phương thức để tạo điểm bắt đầu và kết thúc cho Thread:
  - Cặp 1:
    - Obj.start() → bắt đầu đa luồng
    - Obj.join() → kết thúc đa luồng
      => Tính khoảng thời gian chạy đa luồng bằng time()
  - Cặp 2: (cho phép dừng trong khoảng thời gian nhất định)
    - Obj.daemon() = True → Cho phép dừng thời gian cố định đa luồng
    - Obj.start() → bắt đầu đa luồng
    - time.sleep() → thời gian được yêu cầu giới hạn
      => Dừng thread khi gặp vòng lặp vô hạn mà không ảnh hưởng tới dữ liệu

4. <u>Các phương thức đa luồng Module threading:</u>

- run() → entry point bắt buộc của Thread (chạy trong đối tượng)
- start() → bắt đầu một thread đa luồng
- join([time]) → kết thúc thời gian đa luồng
- isAlive() → kiểm tra thread có đang thực thi không
- getName() → tên của một thread
- setName() → thiết lập tên cho một thread

- threading.activeCount() → trả về số đối tượng thread mà active
- threading.currentThread() → trả về số đối tượng thread trong Thread control của Caller
- threading.enumerate() → trả về danh sách tất cả đối tượng thread mà đang active

5. <u>Kỹ thuật Lock():</u>

- Các threads chia sẻ chung tài nguyên của tiến trình nên có những thời điểm nhiều luồng sẽ thay đổi dữ liệu chung → cần cơ chế tại một thời điểm chỉ có duy nhất một luồng được phép truy cập và dữ liệu chung
- Nếu các luồng khác muốn truy cập vào đoạn dữ liệu này thì cần phải đợi cho thread trước đó hoàn thành công việc của mình → đồng bộ hóa các Thread dễ dàng bằng Locking()
  - Một lock mới được tạo → phương thức Lock()
  - Ép các Thread chạy đồng bộ → phương thức acquire(blocking)
    - Tham số blocking (điều khiển xem Thread cần đợi đạt được lock không)
    - Nếu block = 0 → không thu được lock
    - Nếu block = 1 → thu được lock (cần đợi cho đến khi lock giải phóng)
- Quy luật trên Lock():
  - Nếu state = unlocked → gọi acquire() sẽ chuyển sang locked
  - Nếu state = locked → gọi acquire() sẽ đợi (block) chi đến khi tiến trình khác gọi release()
  - Nếu trạng thái là unlocked, gọi method release() sẽ phát ra RuntimeError exception
  - Nếu trạng thái là locked, gọi method release() sẽ chuyển trạng thái của Lock sang unlocked

Bài 4: Hệ thống phần mềm UNIT TEST

1. **Khái niệm:**
   - Unit là các thành phần độc lập với nhau: (kiểm thử phần mềm)
     - Function → Class → Module → Package (Python)

   - Unit-test như black-box-testing với tập dữ liệu đầu vào khi chạy qua một unit và tập dữ liệu đầu ra phải khớp với mọi tập dữ liệu đã được tính toán trước → đảm bảo tính đúng đắn

   - Yêu cầu của unit-test:
     - Ngắn gọn, dễ hiểu, dễ đọc → mô tả từng nhóm dữ liệu input/output
     - Mỗi unit-test cần phát triển riêng biệt
     - Đặt tên theo quy chuẩn từng nhóm unit-test
     - Dễ dàng refactor code → phát hiện lỗi tiềm ẩn
     - Các function trong unit-test luôn trả về True/False

2. **Các loại UNIT TEST:**

   a) Các function trả về True/False:

Method	Mô tả TRUE
assertEqual(value1, value2)	value1 == value2
assertTrue(value)	value == True
assertFalse(value)	value == False
assertNotEqual(a, b)	a != b
assertIs(a, b)	a is b
assertIsNone(x)	x is None
assertIsNotNone(x)	x is not None
assertIn(a, b)	a in b
assertNotIn(a, b)	a not in b
assertIsInstance(a, b)	isinstance(a, b)
assertAlmostEqual(a, b)	round(a-b, 7) == 0
assertNotAlmostEqual(a, b)	round(a-b, 7) != 0
assertGreater(a, b)	a > b
assertGreaterEqual(a, b)	a >= b
assertLess(a, b)	a < b
assertLessEqual(a, b)	a <= b
assertRegex(s, r)	r.search(s)
assertNotRegex(s, r)	not r.search(s)

   b) Các function so sánh các kiểu dữ liệu khác nhau:

Method	So sánh
asserMultiLineEqual(value1, value2)	strings
assertSequenceEqual(value1, value2)	sequences
asserListEqual(value1, value2)	lists
assertTuple(a, b)	tuples
assertSetEqual(a, b)	set/frozenset
assertDictEqual(a, b)	dicts

3. **<u>Cấu trúc câu lệnh UNIT-TEST:</u>**
   - Xuất thư viện → import unittest
   - Gọi hàm xử lý chính ở bên ngoài → function()
   - Tạo lớp đối tượng class kế thừa ***<u>unittest.TestCase</u>***:
     - Trong lớp đối tượng tạo các hàm test kiểm thử từng trường hợp cho hàm
   - Chạy chương trình đối tượng ra màn hình → ***unittest.main()***

   - Cấu trúc:

```python
import unittest

# Hàm xử lý bên ngoài class
def nameFunction(val1, val2, ...):
    # Khối lệnh thực hiện trong hàm


class NameObject(unittest.TestCase):
    def test_function(self):
        val1, val2, ... = ...
        output = nameFunction(val1, val2, ...)
        output_expected = "Đầu ra kỳ vọng"

        # Xử dụng các function assert của UNIT-TEST
        self.assertEqual(output, output_expected)
        ...

if __name__ == "__main__":
    unittest.main()
```

Bài 5: Tổng quan về Django Python

1. **Django trong thực tế:**
   - Hoàn thiện:
     - Django phát triển thep tư tưởng "Batteries include" (tích hợp toàn bộ, gọi ra mà dùng)
     - Cung cấp mọi thứ cho developer không cần phải nghĩ phải dùng cái ngoài
     - Dev chỉ cần tập trung vào sản phẩm, tất cả đều hoạt động liền mạch với nhau
   - Đa năng:
     - Django có thể được dùng để xây dựng hầu hết các loại website, từ hệ thống quản lý nội dung, cho đến các trang mạng xã hội hay web tin tức
     - Có thể làm việc với framework client-side, và chuyển nội dung hầu hết các loại format (HTML, RESS, JSON, XML, ...)
   - Bảo mật:
     - Django giúp các developer tránh các lỗi bảo mật thông thường để bảo vệ website
     - Cung cấp bảo mật quản lý tên tài khoản và mật khẩu, tránh các lỗi cơ bản như để để thông tin session lên cookie, mã hóa mật khẩu thay vì lưu thẳng
   - Dễ Scale:
     - Sử dụng kiến trúc shared-nothing dựa vào component (mỗi phần của kiến trúc sẽ độc lập với nhau, và có thể thay thế, sửa đổi nếu cần)
     - Có sự chia tách rõ ràng giữa các phần > scale cho việc gia tăng traffic bằng cách thêm phần cứng ở mỗi cấp độ: catching, servers, database servers, hoặc application servers
   - Dễ maintain:
     - Code Django được viết theo nguyên tắc thiết kế và pattern có thể khuyến khích ý tưởng bảo trì và tái sử dụng code
     - Don't Repeat Yourself làm cho không có sự lặp lại không cần thiết, giảm một lượng code
   - Tính linh động:
     - Được viết bằng Python, nó có thể chạy đa nền tảng > không ràng buộc một platform server cụ thể
     - Django được hỗ trợ tốt ở nhiều nhà cung cấp hosting, họ sẽ cung cấp hạ tầng và tài liệu cụ thể cho hosting web Django

2. **Các bước Set up và tạo project:**
   - B1: Tạo folder chính để chứa project python
   - B2: SET UP trên terminal → pip install django
   - B3: Tạo Project → django-admin startproject <Tên Project> .
     (chú ý: <Tên Project trùng với tên Folder đã tạo)
   - B4: Run Project → python manage.py runserver <Tên cổng nếu đặt>
     (tên cổng mặc định là 8000)

3. **Cấu trúc Project:**

- File manage.py:
  - Giúp tương tác Project qua các command (tạo tài khoản admin, database, chạy server ảo, ...)
  - Xuất thư viện → ***import os, sys***

- Folder PythonWeb gồm 4 file:
  - __init__.py: file cơ bản trong Python dùng để biến folder chứa nó thành package, giúp ta có thể import
  - setting.py: đây là file cấu hình project (cấu hình database, đặt múi giờ, cài thêm thư viện, ...) → đưa các WEB APP vào trong này
    - Xuất thư viện → ***from pathlib import Path***
  - urls.py: đây là file giúp ta tạo các đường dẫn urls của trang web để liện kết các webpage lại với nhau
    - Xuất thư viện:
      - ***from django.contrib import admin***
      - ***from django.urls import path, include***
  - wsgi.py: đây là file giúp ta deploy project lên server

4. **Tạo WEB APP:**
   - B1: Cấu trúc → ***python manage.py startapp <Tên APP>***
   - B2: Khai báo Project về App mới (một trang web mới trong Project)
     - File settings.py trong Project chính → khai báo tên app mới vào INSTALLED_APPS
   - B3: Cập nhật settings:
     - Câu lệnh → ***python manage.py migrate***
     - Nếu chạy migrate project lần đầu thì Project Django sẽ tạo một số bảng cho chức năng user, admin cho database hiện tại
     - Bản chất Django hỗ trợ cho chúng ta hệ thống user, admin để thuận tiện cho việc phát triển trang web nhanh hơn

5. **Request, Session, GET/POST method WEB APP:**
   - Thêm WEB APP mới tạo vào file settings.py của Project PythonWeb ở INSTALLED_APP

```
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
```

```
    'django.contrib.messages',
    'django.contrib.staticfiles',
    '<newWebApp>',
]
```

- Tạo định tuyến đường dẫn ở file urls.py của Project PythonWeb: (3 cách)
  - Path mặc định → ***path('admin/', admin.site.urls),***
  - Path tổng → ***path('', include('newWebApp.urls')),***
  - Page tùy chọn → ***path('home1/', include('newWebApp.urls')),***

  → Xuất thư viện:
  - ***from django.urls import path***
  - ***from . import views***

- Tạo views.py định tuyến cho <newWebApp>:
  - Xuất thư viện → ***from django.shortcuts import render, HttpResponse***
  - Mở file urls.py của <newWebApp> để tạo các đường định tuyến:
    - Mặc định → ***path('', views.nameFunctionView),***
    - Tùy chọn → ***path('about/', views.about),***
  - Mở file views.py của <newWebApp>:
    - Tạo các hàm nameFunctionView(request) như đã khai báo ở trên
    - Trả về kết quả → return HttpResponse("ValueData")
      - ValueData có thể là text hoặc HTML, CSS
    - Trả về file HTML → return render(request, "folder/file.html")

```
from django.contrib import admin
from django.urls import path, include

urlpatterns = [
    path('admin/', admin.site.urls),
    path('home', include('home.urls')),
]
```

```
from django.urls import path
from . import views

urlpatterns = [
    path('', views.index)
]
```

```
from django.shortcuts import render
from django.http import HttpResponse
# Create your views here.
def index(request):
    response = HttpResponse()
    response.writelines('<h1>Xin chào</h1>')
    response.write('Đây là app home')
    return response
```

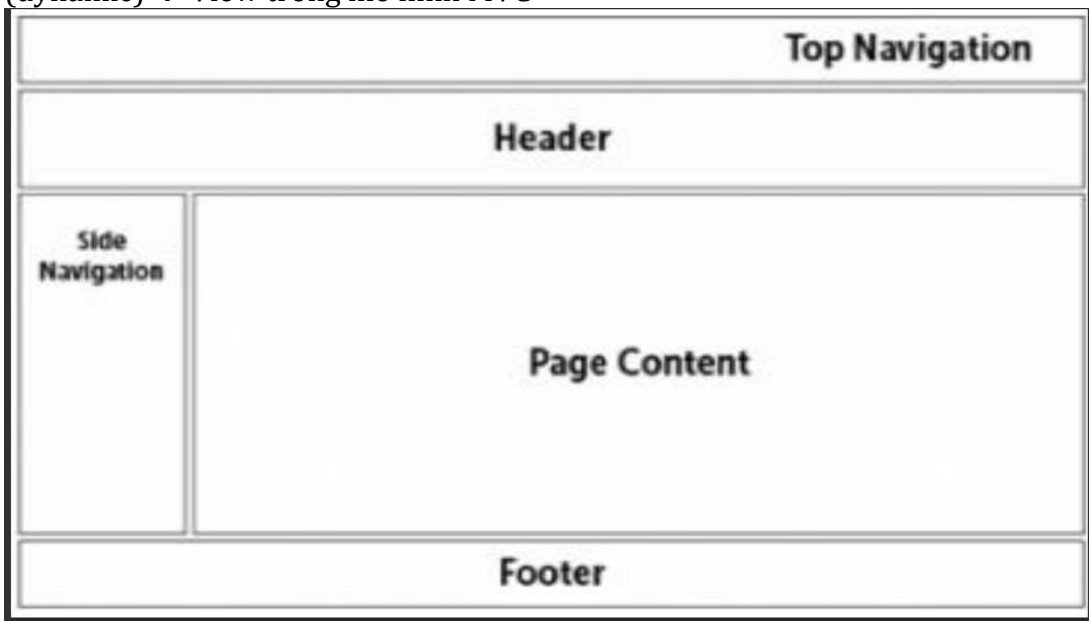
urls.py (Python Web) → urls.py (newWebApp) → views.py (newWebApp)

6. <u>Tổng hợp lại bước tạo webapp:</u>

- B1: run lệnh tạo app
- B2: Khai báo app trong setting của project
- B3: Tạo file urls.py của project
- B4: Khai báo url trong urls.py của project
- B5: Tạo view
- B6: Khai báo view theo url trong urls.py của app

Bài 6: Template của Django

1. **Template với HTML:**
   - Template là bản mẫu, là công cụ tạo ra các giao diện web (DTL và Jinja2)
   - Trong một bản mẫu sẽ có những phần cố định (static) và những phần được bổ sung sau (dynamic) → View trong mô hình MVC



   → Phần tĩnh (không đổi với mọi trang): Top Navigation, Header, Side Navigation, Footer
   → Phần động (thay đổi với mỗi trang): Page Content
   => Kế thừa Template với Base Django

2. **Các bước liên kết Django với HTML:**
   - B1: Tạo folder 'templates' ở trong webApp folder
   - B2: Trong folder templates đó tạo tiếp folder trùng tên webApp folder
   - B3: Tạo file HTML và CSS trong templates/webAppFolder
   - B4: Mở views.py của webApp sửa hàm với đầu ra thành
     → ***return render(request, "folder/file.html")***

3. **Cú pháp của Template Jinja2:**
   - Variables (biến):
     - Sử dụng để chèn dữ liệu động trong template
     - Cú pháp: {{nameVariable}}
     - Biến làm việc theo kiểu key:value → giống format code trong Python
     - Ví dụ:

       Họ của tôi là {ho} và tên của tôi là {ten}
       → ho:Ta, ten:Tung
       => KQ: Họ của tôi là Ta và tên của tôi là Tung

   - Tags (thẻ):
     - Thể hiện xử lý logic trong quá trình kết xuất từ template ra trang kết quả HTML
     - Có thể sử dụng thẻ để thực hiện lệnh if, for để lấy dữ liệu từ CSDL

- Thẻ được bao bằng cặp dấu ***{% Khối lệnh %}***
- Cú pháp: ***{% block content %} Dữ liệu động {% endblock content %}***
- Ví dụ:

```
<title>{% block title %}{% endblock %}</title>
```

- Filter (bộ lọc)
- Comments (chú thích)

4. **Phần động và phần tĩnh của trang web:**
   - B1: Tạo file base.html là HTML cha để cho các con khác kế thừa
   - B2: Trong file base.html, chú ý:
     - Dữ liệu tĩnh nằm ngoài {% block content %} {% endblock content %}
     - Dữ liệu động nằm trong {% block content %} {% endblock content %}
   - B3: Đối với các file.html khác sẽ kế thừa dữ liệu tĩnh từ base.html
     - Cú pháp:

```
{% extends "folder/base.html" %}
<!-- Nội dung thay đổi -->
{% block content %}
        <p> Change Content </p>
{% endblock content %}
```

5. **Đường dẫn tuyệt đối và tương đối:**
   - Dùng trong thẻ <tag href=”đường dẫn”>
   - Đường dẫn tuyệt đối: <tag href=”đường dẫn tuyệt đối/file”>
   - Đường dẫn tương đối:
     - B1: Mở urls.py của webApp đặt app_name = “nameWebApp”
     - B2: Trong urlpatterns thêm name = “namePath”)
     - B3: Mở base.html và thay đổi href thành:
       - <tag href="{% url 'nameWebApp:namePath' %}"> Content </tag>

     - Ví dụ minh họa:

```
# Đặt tên biến cho đường dẫn (biến đại diện cho đường dẫn tương ứng)
app_name = "home"

urlpatterns = [
    path('', views.home, name="home"),
    path('about/', views.about, name="about"),
    path('blog/detail/', views.detail),
]
<a class="nav-link" href="{% url 'home:about' %}"> Giới thiệu </a>
```

6. **Định tuyến với ID:**
   - B1: Vào urls.py của webApp và thêm định tuyến trong path “/<int:id>/”

```
from django.urls import path
from . import views

app_name = "news"
urlpatterns = [
```

```
    path('detail/<int:id>/', views.detail, name = "news_detail"),
]
```

- B2: Sửa thẻ <tag href="{% url 'nameWebApp:namePath' id %}"> Content </tag>

```
<a href="{% url 'news:news_detail' id %}"> PYTHON DJANGO FRAMEWORK </a>
```

- B3: Vào file view.py của webApp thêm tham số id và hàm

```
def detail(request, id):
    return render(request, "news/detail.html")
```

7. Truyền CSDL từ View lên Template:

- Truyền dữ liệu từ Dictionary đơn:
  - B1: Trong hàm của views.py, tạo dict context = {'key1': 'value1', 'key2': 'value2',...} và return thêm cả tham số context

```
# views.py
def detail(request, id):
    context = {
        'title': f'tiêu đề {id}',
        'summary': f'tóm tắt {id}',
        'content': f'nội dung chi tiết {id}',
    }
    return render(request, "news/detail.html", context)
```

  - B2: Thay các biến 'key1', 'key2',... trong thẻ tag là: {{key1}}, {{key2}}, ...

```
<h3> {{title}} </h3>
<p> {{summary}} </p>
<p> {{content}} </p>
```

- Danh sách dữ liệu Dictionary bội:
  - B1: Trong hàm của views.py, tạo mảng list gồm các dictionary1, dictionary2, ...
  - B2: Tạo dict context = {'key': 'valueArray'} và return thêm cả tham số context

```
# views.py
def list(request, id):
    # Mảng gồm nhiều Dictionary
    news = [
        {
            "id": "1",
            "title": "tiêu đề 1",
            "summary": "tóm tắt 1",
        },
        {
            "id": "2",
            "title": "tiêu đề 2",
            "summary": "tóm tắt 2",
        },
        {
            "id": "3",
            "title": "tiêu đề 3",
```

```
            "summary": "tóm tắt 3",
        },
    ]
    context = {
        "news": news,
    }
    return render(request, "news/list.html", context)
```

- B3: Sử dụng vòng lặp for để duyệt từng phần tử và xuất ra dữ liệu tương ứng
  - Cú pháp: {% for item in arrayList %} {% endfor %}
  - Truy xuất dữ liệu → item.key
  - VD:

```
{% for item in news %}
<a href="{% url 'news:news_detail' item.id %}"> {{item.title}} </a>
<p> {{item.summary}} </p>
<hr>
{% endfor %}
```

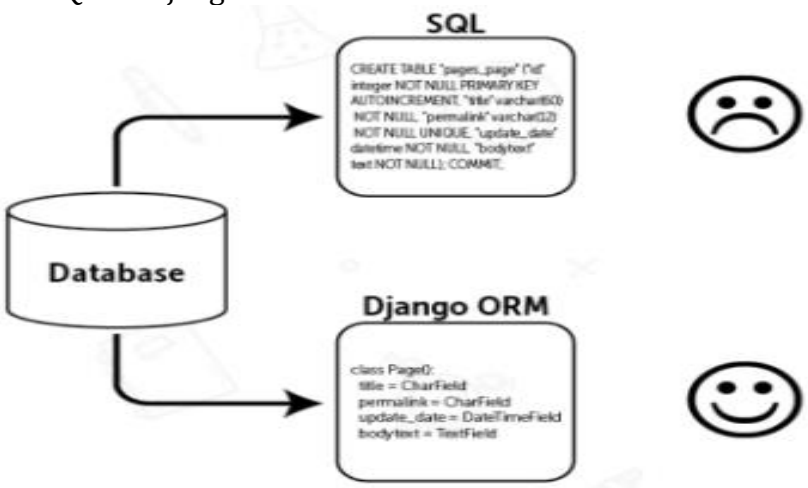
8. Kiểm tra định tuyến ACTIVE với trang:

- Click vào đâu sẽ active đến trang đó
- Cấu trúc → request.resolver_match.url_name
- Ví dụ:

```
<li class="nav-item">
   <a class="nav-link {% if request.resolver_match.url_name == 'news_list' %}
active {% endif %}" href="{% url 'product:news_list' %}"> Sản phẩm </a>
</li>

<li class="nav-item">
   <a class="nav-link {% if request.resolver_match.url_name == 'product_list'
%} active {% endif %}" href="{% url 'product:product_list' %}"> Product </a>
</li>
```

Bài 7: Cơ sở dữ liệu ORM của Django

1. **Object-Relational Mapping (ORM):**
   - ORM là kỹ thuật cho phép truy vấn và quản lý CSDL từ database bằng cách sử dụng mô hình hướng đối tượng
   - Những thư viện ORM là những thư viện đã hoàn chỉnh được và nó đã gói nhóm code cần để quản lý data
   - Không cần sử dụng CSDL mà tương tác trực tiếp qua hướng đối tượng
   - So sánh SQL và Django ORM:



2. **Kết nối Database với MySQL:**
   - B1: Mở CSDL và tạo database mới
   - B2: Mở file setting.py của Project để liên kết tới database (DATABASES)
     - Cách 1: Định nghĩa trực tiếp với mysql
       - Cài đặt mysqlclient → ***pip install mysqlclient***
       - Chỉnh lại DATABASES trong file setting.py của Project:

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.mysql',
        'NAME': 'mysite',
        'USER': 'root',
        'PASSWORD': 'root', //nếu có
        'HOST': 'localhost',
        'PORT': 3306,
    }
}
```

       - Chạy câu lệnh python manage.py migrate trên terminal để xác nhận đã liên kết

     - Cách 2: Định nghĩa qua file config với mysql (recommend)
       - Chỉnh lại DATABASES trong file setting.py của Project:

```
DATABASES = {
    "default": {
```

```
        "ENGINE": "django.db.backends.mysql",
        "OPTIONS": {
            "read_default_file": "./db.cnf",
        },
    }
}
```

- Tạo file mới ngoài cùng → ***db.cnf***

```
[client]
database = <nameDatabase>
user = root
default-character-set = utf8
```

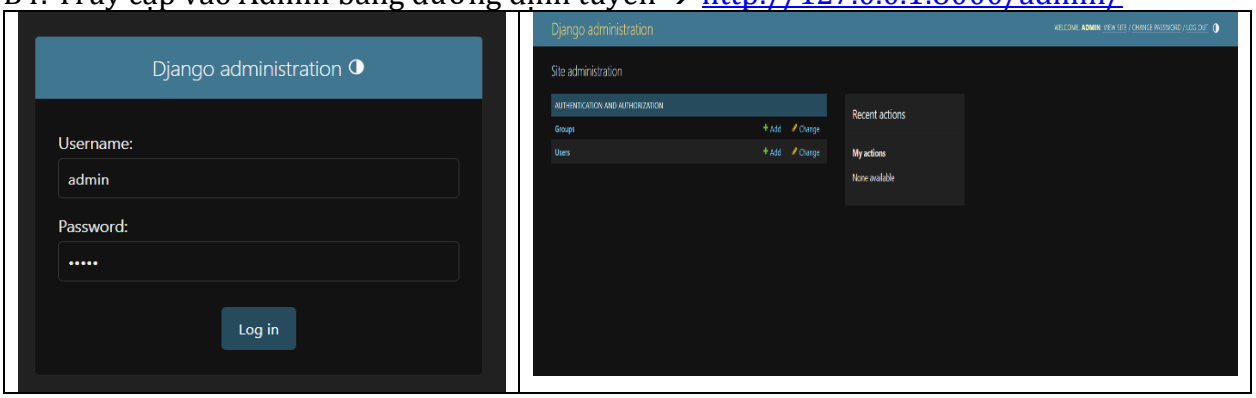
- Chạy câu lệnh ***python manage.py migrate*** trên terminal để xác nhận đã liên kết

- Cách 3: Kết nối từ Dbeaver
  - Tạo kết nối tới SQLite
  - Chọn file db.sqlite3
  - Chạy câu lệnh python manage.py migrate trên terminal để xác nhận đã liên kết

- B3: Tạo Django Admin
  - Tạo superuser để login vào trang admin → ***python manage.py createsuperuser***
  - Nhập các thông tin bảo mật:

```
PS D:\Python Django\Session16\MyShop> python manage.py createsuperuser
Username (leave blank to use 'admin'): admin
Email address: kibo0603@gmail.com
Password:
Password (again):
The password is too similar to the username.
This password is too short. It must contain at least 8 characters.
This password is too common.
Bypass password validation and create user anyway? [y/N]: y
Superuser created successfully.
```

- B4: Truy cập vào Admin bằng đường định tuyến → http://127.0.0.1:8000/admin/



- B5: Cấp quyền truy cập cho các user khác bằng cách add User
  - Superuser status → toàn quyền sử dụng

- Staff status → quyền xem
- Active → không có quyền truy cập

3. Django Models:

- Định nghĩa về cách lưu trữ thông tin
- Một model gồm các trường và các hành động về dữ liệu sẽ lưu trữ
- Model chính là cách gọi khác của Class trong Django
- Mỗi model đại diện cho một bảng CSDL
- Tính chất Model trong Django:
  - Mỗi model phải được kế thừa từ lớp django.db.models.Model
  - Mỗi thuộc tính của model đại diện cho một trường trong CSDL
  - Django sẽ tự động tạo các hàm thao tác với CSDL

- Các bước tạo Model:
  - Vào file models.py trong webApp để tạo đối tượng Object
  - Cú pháp:

```python
# webApp/models.py
from django.db import models

# Create your models here.
class NameClass(models.Model):
    ThuocTinh1 = models.CharField(max_length=<maxValue>)
    ThuocTinh2 = models.CharField(max_length=500, null=True)
    ThuocTinh3 = models.IntegerField(default=0)
    description = models.TextField()        # Không giới hạn độ dài
    ThuocTinhBoolean = models.BooleanField  # Đúng/Sai
    ...

    # Thuộc tính thời gian
    created_at = models.DateTimeField(auto_now_add=True)
    updated_at = models.DateTimeField(auto_now=True)
    ...

    # Thuộc tính về ảnh
    image = models.ImageField(null=True, upload_to="images/")

    # Thuộc tính về file
    file = models.FileField(null=True, upload_to="files/")

    # Thuộc tính URL khác như video, ảnh, ...
    url = models.URLField(null=True)

    # Phương thức toString
    def __str__(self):
        return self.ThuocTinhHienThi
```

- Thực hiện câu lệnh ***python manage.py makemigrations*** trên terminal để tạo file migration thực hiện các câu lệnh SQL (nếu không ghi tên webApp thì sẽ quét toàn bộ app hiện tại)

- Chạy câu lệnh ***python manage.py migrate*** trên terminal để xác nhận sự thay đổi trên databases

4. **Khai báo Model vào trong Admin để quản trị:**
   - B1: Vào file admin.py của webApp
   - B2: Thêm thư viện → ***from .models import NameClass***
   - B3: Đưa mô hình vào admin → ***admin.site.register(NameClass)***
     VD:

```python
# product/admin.py
from django.contrib import admin
from .models import ProductModel

# Register your models here.
admin.site.register(ProductModel)
```

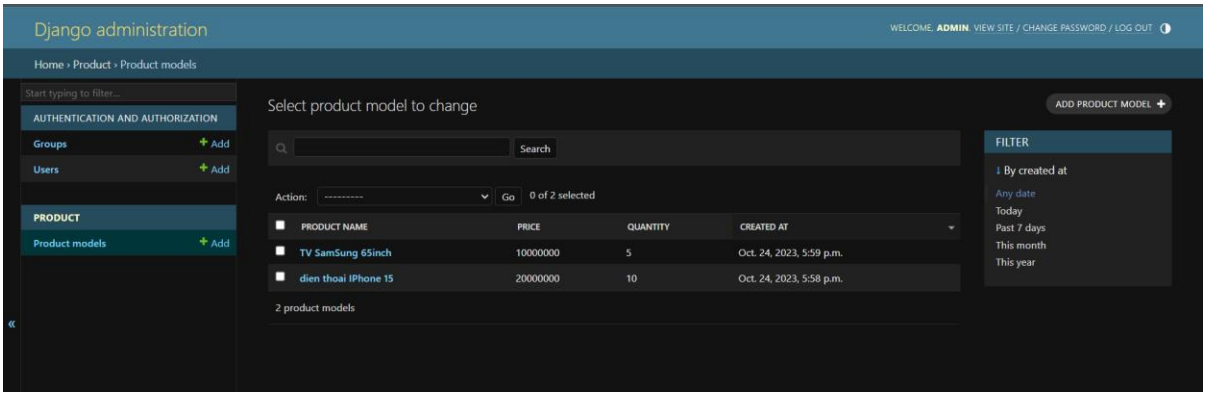
5. **Cập nhật các trường thông tin trên Admin Panel: (Custom)**
   - B1: Vào file admin.py của webApp để chỉnh sửa
   - B2: Khởi tạo class đối tượng mới để hiển thị trên giao diện cho Admin

```python
from django.contrib import admin
from .models import <NameClass>

class NameClassAdmin(admin.ModelAdmin):
    # Các thuộc tính cố định (hiển thị trên trang admin)
    list_display = ["thuocTinh1", "thuocTinh2", "thuocTinh3", ...]

    # Thuộc tính tạo bộ lọc
    List_filter = ["thuocTinh1", "thuocTinh2", "thuocTinh3", ...]

    # Thuộc tính tìm kiếm
    search_fields = ["thuocTinhSearch1", “thuocTinhSearch2”, ...]
```

   - B3: Register your models here

```python
admin.site.register(<NameClass>, NameClassAdmin)
```

Ví dụ mô tả:



6. **Danh mục quản lý Category:**
   - B1: Tạo webApp mới có tên là category → python manage.py startapp category
   - B2: Định nghĩa CategoryModel (giống mục 3) cho file models.py cho category class (khởi tạo đối tượng các thuộc tính)
   - B3: Khai báo category vào trong settings.py của Project
   - B4: Thực hiện câu lệnh ***python manage.py makemigrations*** và ***python manage.py migrate*** trên terminal để đồng bộ lên CSDL
   - B5: Khai báo Model category trong Admin (giống mục 4) để hiển thị giao diện trên Admin Panel

7. **Khóa chính và khóa ngoại trong Django:**
   - B1: Khai báo khóa ngoại Foreign Key giữa webApp với category trong ***webApp/models.py***
     - Xuất thư viện:

```
from django.db import models
from category.models import CategoryModel
```

     - Chỉnh sửa class models.py của webApp:

```
class NameClassAdmin(admin.ModelAdmin):
    # Khởi tạo khóa ngoại
    category = models.ForeignKey(CategoryModel, on_delete=models.PROTECT, null=True)

    # Các thuộc tính cố định (hiển thị trên trang admin)
    ... (giống mục 5)
```

     - Chú ý:
       - PROTECT --> xóa hết con rồi mới xóa cha (nên chọn)
       - CASCADE --> xóa cả con và cha
   - B2: Thực hiện câu lệnh ***python manage.py makemigrations*** và ***python manage.py migrate*** trên terminal để đồng bộ lên CSDL
   - B3: Kiểm tra sự thay đổi trên Dbeaver và Admin Panel

8. **Tài liệu tham khảo:**
   https://docs.djangoproject.com/en/4.2/ref/databases/

Bài 8: Truy vấn CSDL Django

1. **Django Query:**
   - Get All: (list)
     - Lấy tất cả dữ liệu từ CSDL → model.objects.all()
       - Xuất thư viện trong webApp/views.py → from .models import NameClassModel
       - Xử lý trong hàm gọi:

```
def name_function(request):
    database_list = NameClassModel.objects.all()       # Lấy hết dữ liệu
    context = {
        "database_list": database_list
    }
    return render(request, 'webApp/fileList.html', context)
```

   - Get by ID: (detail)
     - Lấy dữ liệu theo ID trong webApp/views.py → model.objects.get(id=id)

```
def function_detail(request, id):
    databaseID = NameClassModel.objects.get(id=id)
    context = {
        "productKey": databaseID
    }

    return render(request, 'webApp/fileDetail.html', context)
```

     - Truy vấn trong file HTML → productKey.<thuộc tính>

2. **QuerySet:**
   - Là thể hiện một tập các object trong database
   - QuerySet tương đương với một câu lệnh SELECT
   - filter là một mệnh đề giới hạn như là WHERE hoặc LIMIT
   - Mỗi model có ít nhất một Manager, mặc định là objects (thao tác trong view.py)
     - all_post = PostModel.objects.all() → Lấy hết data
     - QuerySet – GET:
       - data = Member.objects.all().values() → Trả về mỗi đối tượng kiểu dictionary
       - data = Member.objects.values_list('firstname',) → chỉ trả về các trường dữ liệu chỉ định 'firstname'
       - data = Member.objects.get(pk=id) → lấy một object có pk = id
     - QuertSet – Filter:
       - AND:
         - Member.objects.filter(<điều kiện lọc 1>, <điều kiện lọc 2>, ...).values() → trả về các bản ghi thỏa mãn điều kiện → AND là dấu phẩy
       - OR:
         - Member.objects.filter(key1='value1').values() | Member.objects.filter(key2='value2').values()
         - from django.db.models import Q
           Member.objects.filter(Q(key1='value1') | Q(key2='value2')).values()

- Limit → trả về số lượng đối tượng đã định giới hạn

```
data = Member.objects.all().values()[:limit]
```

- Ví dụ một số QuerySet khác:
  - filter() trả về các bản ghi thỏa mãn điều kiện:

```
Member.objects.filter(firstname__startswith='L').values()
Member.objects.filter(post_date__year=2023).values()

endswith, contains, in, range,
gt, gte, lt, lte, exact,
year, month, day, date, hour, minute, second, ...
```

  - exclude() loại các bản ghi thỏa mãn điều kiện:

```
Post.objects.filter(
        title__startswith='What'
).exclude(
        post_date__gte=datetime.date.today()
).filter(
        post_date__gte=datetime(2005, 1, 30)
)
```

  - order_by() sắp xếp kết quả:

```
Member.objects.all().order_by('firstname').values()
→ SELECT * FROM members ORDER BY firstname;

Member.objects.all().order_by('-firstname').value >> DESC
→ sắp xếp giảm dần
```

3. **Minh họa ví dụ:**
   - Tìm kiếm dữ liệu bằng POST trong webApp/view.py:

```python
# Lấy dữ liệu từ CSDL
from .models import ProductModel

def product_list(request):
    product_list = []

    # Sự kiện hiển thị
    if request.method == "GET":
        product_list = ProductModel.objects.all()   # Lấy hết dữ liệu của product

    # Sự kiện tìm kiếm
    if request.method == "POST":
        keyword = request.POST.get("keyword")
        product_list = ProductModel.objects.filter(product_name__contains=keyword)


    context = {
        "product_list": product_list
    }
```

```
    return render(request, 'product/product_list.html', context)
```

- Tìm kiếm dữ liệu bằng GET trong webApp/view.py: (khuyến khích)

```
# file.html
<form action="" method="GET">
  {% csrf_token %}
</form>
```

```
# webApp/view.py
def product_list(request):
    product_list = []
    keyword = request.GET.get("keyword")

    # # Sự kiện hiển thị
    # if request.method == "GET":
    #     product_list = ProductModel.objects.all().order_by("-
created_at")       # Lấy hết dữ liệu của product

    # Sự kiện tìm kiếm
    if keyword:
        product_list = ProductModel.objects.filter(
            Q(product_name__contains=keyword)
            | Q(summary__contains=keyword)
            | Q(description__contains=keyword))
    else:
        product_list = ProductModel.objects.all().order_by("-created_at")

    context = {
        "product_list": product_list,
        "keyword": keyword if keyword else "",
    }

    return render(request, 'product/product_list.html', context)
```

Bài 9: Phân cấp cấu hình thuộc tính Media

1. **Phân cấp cấu hình Image trong CSDL:**
   - B1: Tạo folder "media" chứa các folder con "images" và "files"

   - B2: Cấu hình media cho Project bằng cách vào settings.py của Project (mục STATIC FILES):
     - Xuất thư viện → import os
     - STATIC_URL = 'static/' → mặc định
     - MEDIA_URL = 'media/' → đường dẫn tương đối đến thư mục media
     - MEDIA_ROOT = os.path.join(BASE_DIR, 'media/') → đường dẫn vật lý tuyệt đối

   - B3: Chỉnh sửa urls.py của Project:
     - Thêm thư viện cấu hình:
       - from django.conf.urls.static import static
       - from django.conf import settings
     - Nối thêm đường dẫn:
       - urlpatterns += static(settings.MEDIA_URL, document_root = settings.MEDIA_ROOT)

   - B4: Thêm thuộc tính vào trong webApp/models.py:
     - image = models.ImageField(null=True, upload_to="***images/***")

   - B5: Thực hiện câu lệnh ***python manage.py makemigrations*** và ***python manage.py migrate*** trên terminal để đồng bộ lên CSDL

   - B6: Hiển thị đường dẫn ảnh ra Template:
     - Mở file HTML của webApp cần hiển thị
     - Chỉnh sửa đường dẫn src của thẻ <img>:

```
{% if item.image %}
<img src="{{item.image.url}}" class="card-img-top" alt="https://placehold.co/200x150">
{% endif %}
```

2. **Phân cấp cấu hình File trong CSDL:**
   - B1: Giống các bước làm Image (b1-b2-b3)
   - B2: Thêm thuộc tính vào trong webApp/models.py:
     - file = models.FileField(null=True, upload_to="files/")
   - B3: Thực hiện câu lệnh ***python manage.py makemigrations*** và ***python manage.py migrate*** trên terminal để đồng bộ lên CSDL
   - B4: Hiển thị đường dẫn ảnh ra Template:
     - Mở file HTML của webApp cần hiển thị
     - Cách 1: Không đặt tên:
       - Chỉnh sửa đường dẫn src của thẻ <a>:

```
{% if item.file %}
<a href="{{item.file.url}}" target="_blank"> {{item.file}} </a>
{% endif %}
```

- Cách 2: Có đặt tên:
  - Có thể đặt tên file bằng cách tạo property trong webApp/models.py:

```
# Đặt tên cho file
@property       # Đây là thuộc tính
def get_file_name(self):
    return os.path.basename(self.file.url) if self.file else ""
```

  - Khai báo trong template HTML bằng cách: (có thể dùng hoặc không)

```
{% if item.file %}
<a href="{{item.file.url}}" target="_blank"> {{item.get_file_name}} </a>
{% endif %}
```

3. Phân cấp cấu hình video Youtube trong CSDL:

- B1: Giống các bước làm Image (b1-b2-b3)

- B2: Thêm thuộc tính vào trong webApp/models.py:
  - url = models.URLField(null=True)

- B3: Thực hiện câu lệnh ***python manage.py makemigrations*** và ***python manage.py migrate*** trên terminal để đồng bộ lên CSDL

- B4: Hiển thị đường dẫn ảnh ra Template:
  - Mở file HTML của webApp cần hiển thị
  - Chỉnh sửa đường dẫn src:

```
{% if item.url %}
<div class="embed-responsive embed-responsive-16by9">
<iframe class="embed-responsive-item" src="{{item.url}}" allowfullscreen></iframe>
</div>
{% endif %}
```

- B5: Mở link youtube → share → Nhúng → lấy src trong embed

Bài 10: Django Form

1. **Form thông tin:**
   - Là một phần của Django framework cho phép xây dựng các form trong ứng dụng web Django dễ dàng nhanh chóng
   - Form cho phép nhập dữ liệu vào ứng dụng web và gửi đến máy chủ để xử lý
   - Cung cấp nhiều hỗ trợ để xây dựng các form (validation, widget, formset, form class)

2. **Cách tạo Form cho webApp:**
   - B1: Tạo forms.py trong webApp
   - B2: Xuất thư viện → from django import forms
   - B3: Định nghĩa class form trong webApp/forms.py:

```
from django import forms

class NameClassForm(forms.Form):
    # Tạo Form nhập
    name_form1 = forms.CharField(label="TitleName1", max_length=100, required=True)
    name_form2 = forms.CharField(label="TitleName2", max_length=500)
    name_form3 = forms.CharField(label="TitleName3", widget=forms.Textarea)        # Ô nhập lớn
    name_form3 = forms.IntegerField(label="TitleName4")
```

   - B4: Tạo thêm chuyển trang khác trong webApp/urls.py
   - B5: Khai báo hàm sử dụng vào webApp/views.py:

```
def form_add(request):
    # Thêm mới hoàn toàn
    form = NameClassForm()

    # Lấy dữ liệu từ người dùng
    if request.method == "POST":
        form = NameClassForm(request.POST)
        if form.is_valid():
            form.luuDoiTuong()
            return redirect("webApp:fileNameHTML")
    context = {
        "form": form
    }
    return render(request, 'webApp/add.html', context)
```

   - B6: Thực hiện thiết kế form từ HTML add.html:
     - Cấu trúc:

```
<!-- Đưa form vào đây -->
<form action="" method="POST">
    <!-- Chống giả danh Django -->
    {% csrf_token %}

    {{form.as_p}}
    <button type="submit" class="btn btn-primary p-2"> Thêm mới </button>
```

```
</form>
```

- Render tự động qua các thẻ tag:
  - form.as_p: thẻ <p>
  - form.as_div: thẻ
  - form.as_table: thẻ <tr>
  - form.as_ul: thẻ

```
<form action="" method="post">
    {% csrf_token %}
        <table>
            <tr>
                <th>
                    <label for="{{ form.username.id_for_label }}">User:</label
                </th>
                <td>
                    {{ form.username }}
                <td>
            </tr>
            <tr>
                <th>
                    <label for="{{ form.password.id_for_label }}">Password:</l
                </th>
                <td>
                    {{ form.password }}
                </td>
            </tr>
            <tr>
                <th>
                    <label for="{{ form.email.id_for_label }}">Email:</label>
                </th>
                <td>
                    {{ form.email }}
                </td>
            </tr>
            <tr>
                <td></td>
                <td>
                    <input type="submit" value="Submit">
                </td>
            </tr>
    </table>
</form>
```

- B7: Lưu đối tượng từ form vào models
  - B1: Xử dụng folder webApp/forms.py
  - B2: Xuất thư viện:
    - from django import forms
    - from .models import NameClassModel
  - B3: Tạo phương thức lưu đối tượng trong class

```
def luuDoiTuong(self):
    model = ProductModel(
       name_model1 = self.cleaned_data["name_form1"],
       name_model2 = self.cleaned_data["name_form2"],
       name_model3 = self.cleaned_data["name_form3"],
       ...
    )
model.save()
```

3. Sửa form thông tin:

- B1: Tạo thêm chuyển trang khác trong webApp/urls.py:

```
path('edit/<int:id>/', views.form_edit, name="name_edit"),
```

- B2: Thực hiện thiết kế form từ HTML edit.html:
  - Cấu trúc:

```
<!-- Đưa form vào đây -->
<form action="" method="POST", enctype="multipart/form-data">
```

```
    <!-- Chống giả danh Django -->
    {{% csrf_token %}}

    {{form.as_p}}
    <button type="submit" class="btn btn-primary p-2"> Sửa </button>
</form>
```

- B3: Định nghĩa class sửa trong webApp/forms.py:

```
# Sửa dữ liệu
class NameClassModelForm(forms.ModelForm):
    class Meta():
        model = ProductModel
        # Các trường thông tin sẽ hiện lên web
        fields = ["name1", "name2", "name3", ...]
```

- B3: Khai báo hàm sử dụng vào webApp/views.py:

```
def form_edit(request, id):
    # Lấy model từ database
    model = NameClassModel.objects.get(id=id)
    form = NameClassModelForm(instance=model)

    # Xử lý sự kiện
    if request.method == "POST":
        form = NameClassModelForm(request.POST, request.FILES, instance=model)
        if form.is_valid():
            form.luuDoiTuong()

    context = {
        "form": form
    }
    return render(request, 'webApp/edit.html', context)
```

4. Xóa Form thông tin:

- B1: Tạo thêm chuyển trang khác trong webApp/urls.py:

```
path('delete/<int:id>/', views.product_delete, name="product_delete"),
```

- B2: Thực hiện thiết kế form từ HTML edit.html:
  - Cấu trúc:

```
<!-- Đưa form vào đây -->
<form action="" method="POST" enctype="multipart/form-data">
    <!-- Chống giả danh Django -->
    {% csrf_token %}

    {{form.as_p}}
    <button type="submit" class="btn btn-primary p-2"> Xóa </button>
</form>
{% endblock content %}
```

- B3: Khai báo hàm sử dụng vào webApp/views.py:

```
def form_delete(request, id):
    # Lấy model từ database
    model = ProductModel.objects.get(id=id)
    model.delete()
    return redirect("webApp:fileNameHTML")
```

5. Bộ soạn thảo Ckeditor cho Django Form Admin:

- Link tham khảo: https://pypi.org/project/django-ckeditor/
- B1: Tải thư ckeditor → pip install django-ckeditor
- B2: Tạo file requirements.txt để liệt kê tất cả các thư viện mình từng cài

```
# pip list
django==4.2.6
django-ckeditor==6.7.0
mysqlclient==2.1.1
Pillow==10.0.1
djangorestframework==3.14.0

# pip install -r requirements.txt
```

- B3: Sử dụng câu lệnh trên terminal để hiển thị các phiên bản của Django → pip list
- B4: Tải các thư viện thích hợp bằng câu lệnh → pip install -r rquirements.txt
- B5: Vào settings.py của Project trong INSTALL_APPS để thêm 'ckeditor'
- B6: Trong settings.py của Project thêm → CKEDITOR_UPLOAD_PATH = "uploads/"
- B7: Tạo folder uploads bên trong thư mục media
- B8: Mở webApp/models.py:
  - Khai báo thư viện → from ckeditor.fields import RichTextField
  - Trong classModel → description = RichTextField(null=True)
- B9: Định nghĩa lại trường thông tin trong webApp/admin.py:
  - Khai báo thư viện:
    → from ckeditor.widgets import CKEditorWidget # Không upload
    → from django import forms
    → from ckeditor_uploader.widgets import CKEditorUploadingWidget # Có upload
  - Tạo class cho ckeditor:

```
class ProductAdminForm(forms.ModelForm):
    description = forms.CharField(widget=CKEditorUploadingWidget())
    class Meta:
        model = ProductModel
        fields = '__all__'
```

  - Thêm trường thông tin trong class ProductAdmin:

```
# Ckeditor
form = ProductAdminForm
```

- B10: python manage.py makemigrations
- B11: python manage.py migrate

- B12: Tạo định tuyến urls.py cho ckeditor trong project
  → path('ckeditor/', include('ckeditor_uploader.url'))

6. **<u>Bộ soạn thảo Ckeditor cho Django Form Web</u>**:
   - B1: Mở webApp/urls.py:
     - Xuất thư viện → from ckeditor_uploader.widgets import CKEditorUploadingWidget
     - Sửa dữ liệu trong ProductModelForm:

```python
# Sửa dữ liệu
class ProductModelForm(forms.ModelForm):
    description = forms.CharField(widget=CKEditorUploadingWidget())
    content = forms.CharField(widget=CKEditorUploadingWidget())
    class Meta():
        model = ProductModel
        fields = '__all__'
```

   - B2: Vào settings.py của Project để thêm 'ckeditor' vào INSTALL_APPS
   - B3: Vào urls.py của Project để thêm → path('ckeditor/', include('ckeditor_uploader.url'))
   - B4: Vào template chỉnh sửa file HTML:

```html
<form action="" method="POST" enctype="multipart/form-data">
    <!-- Chống giả danh Django -->
    {% csrf_token %}
    {{form.media}}
    {{form.as_p}}
    <button type="submit" class="btn btn-primary p-2"> Thêm mới </button>
</form>
```

Bài 11: Django Generic View

1. **Generic:**
   - Generic View được phát triển để giải quyết vấn đề thu gom những cái chung của những kiến thức và mô hình ở view (controller) để phát triển và abstract chúng sao cho ta có thể viết xử lý data trong views

2. **Phân loại List View - Detail View:**
   - B1: Tạo định tuyến trong Project/urls.py
   - B2: Truy cập vào webApp/views.py:
     - Xuất thư viện → from django.views.generic import ListView, DetailView
     - Tạo hai đối tượng: (nameclass = app_name)

```python
from django.views.generic import ListView, DetailView

class NameClassListView(ListView):
    queryset = NameClassModel.objects.all()
    template_name = 'folder/list.html'    # Nội dung đến đường dẫn

    # Lấy từ trong dict
    context_object_name = "nameclass_list"

class NameClassDetailView(DetailView):
    model = NameClassModel
    template_name = 'folder/detail.html'    # Nội dung đến đường dẫn

    # Lấy từ trong dict
    context_object_name = "nameclass"
```

   - B3: Chỉnh sửa định tuyến webApp/urls.py:

```python
from django.urls import path, include
from . import views

app_name = "nameclass"
urlpatterns = [
    path('', views.NameClassListView.as_view(), name="nameclass_list"),
    path('<int:pk>', views.NameClassDetailView.as_view(), name = "nameclass_detail"),
]
```

3. **Ví dụ so sánh mô hình View:**

```python
# webApp/urls.py
from django.urls import path, include
from . import views

app_name = "category"
urlpatterns = [
    # Cách 1: thông thường
```

```
    path('', views.category_list, name="category_list"),
    path('<int:id>', views.category_detail, name = "category_detail"),

    # Cách 2: Generic
    path('view/', views.CategoryListView.as_view(), name="category_list_view"),
    path('view/<int:pk>', views.CategoryDetailView.as_view(), name =
"category_detail_view"),
]
```

```
# webApp/views.py
from django.shortcuts import render
from .models import CategoryModel

# Create your views here.

# Cách 1: Xây dựng View Model cơ bản
def category_list(request):
    category_list = CategoryModel.objects.all()
    context = {
        "category_list": category_list
    }
    return render(request, 'category/list.html', context)

def category_detail(request, id):
    category = CategoryModel.objects.get(id=id)
    context = {
        "category": category
    }
    return render(request, 'category/detail.html', context)


# Cách 2: Xây dựng View Model bằng Generic
from django.views.generic import ListView, DetailView

class CategoryListView(ListView):
    queryset = CategoryModel.objects.all()
    template_name = 'category/list.html'    # Nội dung đến đường dẫn

    # Lấy từ trong dict
    context_object_name = "category_list"

class CategoryDetailView(DetailView):
    model = CategoryModel
    template_name = 'category/detail.html'    # Nội dung đến đường dẫn

    # Lấy từ trong dict
    context_object_name = "category"
```

Bài 12: Phân trang và điều hướng

1. **Tổng quan:**
   - Không phải tất cả các website đều thể hiện đầy đủ thông tin trên một page đơn lẻ
   - Cần sử dụng nhiều Page đích để cải thiện điều hướng và trải nghiệm người dùng
   - Các tham số:
     - limit_number
     - page_number

2. **Sử dụng Paginator:**
   - B1: Vào settings.py của Project thêm DEFAULT_SORT và DEFAULT_LIMIT ở cuối
   - B2: Vào webApp/views.py để import → from django.core.paginator import Paginator
   - B3: Trong hàm xử lý views.py có sử dụng tới limit bằng cách khai báo:
     - limit = settings.DEFAULT_LIMIT
     - page = request.GET.get("page", 1)

```
# Các biến để phân trang
limit = request.GET.get("limit")              # Giới hạn
if limit and limit.isnumeric():
    limit = int(limit)
else:
    limit = settings.DEFAULT_LIMIT

page = request.GET.get("page", 1)             # trang bắt đầu
```

   - B3: Thực hiện paginator ngay bên trong hàm của views.py:
     - Cú pháp → product_list_paging = Paginator(product_list, limit)
     - Truy cập → product_list_paging = product_list_paging.get_page(page)
     - Dictionary tham chiếu:

```
context = {
   "product_list": product_list_paging,
   "keyword": keyword if keyword else "",
}
```

3. **Sử dụng thanh điều hướng Paginator:**
   - B1: Sử dụng Bootstrap để lấy mẫu thanh điều hướng

```
<nav>
  <ul class="pagination">
    <li class="page-item {% if not product_list.has_previous %}disabled{% endif %}">
      {% if product_list.has_previous %}
      <a class="page-link" href="?page={%if product_list.has_previous %}
{{product_list.previous_page_number}} {% endif %}{% for k,v in request.GET.items %} {% if k != 'page'
%}&{{k}}={{v}}{% endif %} {% endfor %}">Previous</a>
      {% endif %}
    </li>
```

```
    <li class="page-item active" aria-current="page">
      <a class="page-link" href="#">{{product_list.number}}/{{product_list.paginator.num_pages}}</a>
    </li>

    <li class="page-item {% if not product_list.has_next %}disabled{% endif %}">
      <a class="page-link" href="?page={%if product_list.has_next %}
{{product_list.next_page_number}} {% endif %}{% for k,v in request.GET.items %} {% if k != 'page'
%}&{{k}}={{v}}{% endif %} {% endfor %}">Next</a>
    </li>
  </ul>
</nav>
```

- B2: Mở webApp/views.py và thêm thư viện
  → from django.core.paginator import Paginator, PageNotAnInteger, EmptyPage

- B3: Chỉnh sửa cài đặt try-catch cho Paginator:

```
product_list_paging = Paginator(product_list, limit)
try:
    product_list_paging = product_list_paging.get_page(page)
except PageNotAnInteger:
    # Trả về trang đầu tiên
    product_list_paging = product_list_paging.get_page(1)
except EmptyPage:
    # Trả về trang cuối cùng
    product_list_paging = product_list_paging.get_page(product_list_paging.num_pages)
```

Bài 13: Static trong Django

1. **Tổng quan:**
   - Sử dụng folder cá nhân static để lưu trữ Bootstrap
   - Tránh rủi ro khi lỗi link hoặc địa chỉ global/local

2. **Static trong Python Project:**
   - B1: Tạo folder static để chứa hai thư mục bootstrap CSS và JS
   - B2: Vào settings.py của project tìm đến chỗ STATIC_URL:

```
import os

STATIC_URL = 'static/'
STATICFILES_DIRS = [
    os.path.join(BASE_DIR, 'static/'),
]
```

   - B3: Mở base.html để sửa đổi:

```
{% load static %}
<link href="{% static 'css/bootstrap.min.css' %}" rel="stylesheet" type="text/css"/>
<script src=="{% static 'js/bootstrap.bundle.min.js' %}"></script>
```

Bài 14: Phân quyền User và Permission

1. **Phân quyền trên Model (Model Permissions):**
   - add → user có quyền này có thể tạo ra 1 đối tượng của model
   - delete → user có quyền này có thể xóa 1 đối tượng của model
   - change → user có quyền này có thể thay đổi thông tin của 1 đối tượng của model
   - views → user có quyền này có thể xem thông tin chi tiết của 1 đối tượng của model
     => Model là tương tác với databases

2. **Quy tắc tên Permission:**
   - Tên của permission đặt theo quy tắc cụ thể → ***<app>.<action>_<modelname>***
     - <app> là tên của app. Ví dụ, model User được import từ app 'auth' (django.contrib.auth)
     - <action> là một trong các thao tác ở trên (add, delete, change, view)
     - <modelname> là tên của model, viết thường

   - Ví dụ: quyền thay đổi 1 user được đặt như sau → ***auth.change_user***

3. **Check Permission – has_perm():**
   - Kiểm tra quyền của user:

```
>>> from django.contrib.auth.models import User
>>> u = User.objects.create_user(username='haki')
>>> u.has_perm('auth.change_user')
False
```

   - Luôn trả về True đối với superuser bất kể permission có tồn tại không:

```
>>> from django.contrib.auth.models import User
>>> superuser = User.objects.create_superuser(
...     username='superhaki',
...     email='me@hakibenita.com',
...     password='secret',
)
>>> superuser.has_perm('does.not.exist')
True
```

4. **Giám sát quyền:**
   - Model trong Django không thực sự giám sát quyền
   - Nới duy nhất giám sát quyền là Django Admin
   - Trong các ứng dụng Django, user thường được lấy từ request nên các quyền được giám sát ở lớp view

- Ví dụ: để ngăn user không có quyền view trên model 'user ' truy cập vào view hiển thị thông tin user

```
from django.core.exceptions import PermissionDenied

def users_list_view(request):
    if not request.user.has_perm('auth.view_user'):
        raise PermissionDenied()
```

- Lưu ý:
  - Nếu user thực hiện yêu cầu đăng nhập và được xác nhận thì request.user sẽ lưu thông tin một đối tượng của User
  - Nếu user không đăng nhập thì request.user sẽ là một đối tượng của AnonymousUser → Đây là một đối tượng đặc biệt được Django sử dụng để chỉ người dùng chưa được xác thực
  - Nếu người dùng thực hiện request không có quyền view_user thì cần đưa ra:
    - exception PermissionDenied
    - Phản hồi với trạng thái 403 được trả về cho client
  - Để kiểm tra các quyền trong view, Django cung cấp một shortcut decorator là ***permisson_required***

```
from django.contrib.auth.decorators import permission_required

@permission_required('auth.view_user')
def users_list_view(request):
    pass
```

- Đối với Template:
  - Có thể truy cập các quyền của người dùng hiện tại thông qua một biến template đặc biệt gọi là '***perms***'
  - Ví dụ: nếu muốn hiển thị nút xóa cho người dùng có quyền xóa:

```
{% if perms.auth.delete_user %}
<button>Delete user!</button>
{% endif %}
```

5. So sánh Django Admin và Model Permissions:

- Django Admin có sự tích hợp rất chặt chẽ với hệ thống xác thực có sẵn, cụ thể là các quyền trên model
- Nếu user không có quyền trên một model thì không thể nhìn thấy nó hoặc truy cập nó trong trang admin
- Nếu user có quyền xem và thay đổi trên một model thì có thể xem và cập nhật các đối tượng của model nhưng không thể thêm các đối tượng mới hoặc xóa các đối tượng hiện có

- Với các quyền thích hợp tại chỗ, user là admin ít có khả năng mắc lỗi và những kẻ xâm nhập sẽ khó khăn hơn trong việc gây hại

Bài 15: Xử lý LOGIN và LOGOUT

1. **Tổng quan:**
   - Django cho chúng ta thiết kế 1 hệ thống user có sẵn
   - Django hỗ trợ cho chúng ta những chức năng đăng nhập và đăng xuất của hệ thống
   - Cần biết cách config các chức năng đó

2. **Xử lý REGISTER:**
   - B1: Tạo newApp register để đăng ký → python manage.py startapp register
   - B2: Mở settings.py của Project thêm app mới 'register' vào INSTALL_APPS
   - B3: Tạo bộ định tuyến trong register/urls.py

```
from django.urls import path
from . import views

app_name = "register"

urlpatterns = [
    path('', views.register, name="register"),
]
```

   - B4: Thêm urls.py cho Project → path('dang-ky/', include('register.urls'))
   - B5: Viết function cho views.register:

```
from django.shortcuts import render

# Create your views here.
def register(request):
    context = {

    }
    return render(request, "register/register.html", context)
```

   - B6: Tạo template cho register → templates/register/register.html

```
{% extends "home/base.html" %}

<!-- Nội dung thay đổi -->
{% block content %}
<div class="container">
    <h1> Đăng ký thành viên </h1>
</div>

{% endblock content %}
```

   - B7: Tạo forms.py trong register

```
# Khai báo thư viện
from typing import Any
from django import forms
from django.contrib.auth.models import User
```

```
import re

# Tạo class đối tượng
class RegisterForm(forms.Form):
    # Các trường thông tin
    username = forms.CharField(label="Tài khoản", max_length=50, required=True)
    password1 = forms.CharField(label="Mật khẩu", required=True,
widget=forms.PasswordInput)
    password2 = forms.CharField(label="Xác nhận mật khẩu", required=True,
widget=forms.PasswordInput)
    email = forms.EmailField(label="Email")
    last_name = forms.CharField(label="Họ")
    first_name = forms.CharField(label="Tên")

    def clean_password2(self):
        if "password1" in self.cleaned_data:
            password1 = self.cleaned_data["password1"]
            password2 = self.cleaned_data["password2"]
            if password1 and password1 == password2:
                return password2
        raise forms.ValidationError("Mật khẩu không hợp lệ")

    def clean_username(self):
        username = self.cleaned_data["username"]

        # Username hợp lệ bằng biểu thức chính quy --> regex username
        if re.search(r'^[A-Za-z0-9]+(?:[_-][A-Za-z0-9]+)*$', username):
            raise forms.ValidationError("username không hợp lệ")

        # Kiểm tra tài khoản đã tồn tại chưa
        try:
            User.objects.get(username=username)
        except User.DoesNotExist:
            # Hợp lệ đăng ký
            return username
        raise forms.ValidationError("username đã tồn tại trong hệ thống")

    # Lưu giá trị từ form vào Model
    def save(self):
        User.objects.create_user(
            username=self.cleaned_data["username"],
            password=self.cleaned_data["password1"],
            email=self.cleaned_data["email"],
            last_name=self.cleaned_data["last_name"],
            first_name=self.cleaned_data["first_name"],
            # is_staff = False,
        )
```

- B8: Đưa Form lên views trong register

```python
from django.shortcuts import render
from .forms import RegisterForm

# Create your views here.
def register(request):
    form = RegisterForm()

    context = {
        "form": form
    }
    return render(request, "register/register.html", context)
```

- B9: Đưa lên Template HTML:

```
{% extends "home/base.html" %}

<!-- Nội dung thay đổi -->
{% block content %}
<div class="container">
    <h1> Đăng ký thành viên </h1>

    <form action="" method="POST">
        <!-- Chống giả danh Django -->
        {% csrf_token %}
        {{ form.as_p }}

        <button type="submit" class="btn btn-primary p-2"> Xác nhận </button>
    </form>
</div>

{% endblock content %}
```

- B10: Viết sự kiện Submit trong register/views.py:

```python
from django.shortcuts import render, redirect
from .forms import RegisterForm

# Create your views here.
def register(request):
    form = RegisterForm()

    # Sự kiện submit
    if request.method == "POST":
        form = RegisterForm(request.POST)
        if form.is_valid():
            form.save()
            return redirect("home:home")

    context = {
        "form": form
    }
```

```
    return render(request, "register/register.html", context)
```

- B11: Kiểm tra trên Admin Panel

3. Xử lý LOGIN:

- B1: Tạo newApp login2 để đăng nhập → python manage.py startapp login2
- B2: Tạo newApp logout2 để đăng nhập → python manage.py startapp logout2
- B3: Mở settings.py của Project thêm app mới 'login2' và 'logout2' vào INSTALL_APPS
- B4: Tạo urls.py trong Project → path('dang-nhap/', include('login2.urls'))
- B5: Tạo login2/urls.py

```
from django.urls import path
from . import views

app_name = "login2"

urlpatterns = [
    path('', views.login2, name="login2"),
]
```

- B6: Tạo templates/login2/login.html

```
{% extends "home/base.html" %}

<!-- Nội dung thay đổi -->
{% block content %}
<div class="container">
    <h1> Đăng nhập </h1>
</div>

{% endblock content %}
```

- B7: Mở login2/views.py để viết hàm:

```
from django.shortcuts import render

# Create your views here.
def login2(request):
    return render(request, "login2/login.html")
```

- B8: Viết Form cho Login (khuyến khích làm giống register) hoặc viết trần trên HTML

```
{% extends "home/base.html" %}

<!-- Nội dung thay đổi -->
{% block content %}
<div class="container">
    <h1> Đăng nhập </h1>
    <hr/>

    <div class="row justify-content-center">
```

```
        <div class="col-5">
        <form action="" method="POST">
            {% csrf_token %}
            <div class="row mb-3">
              <label for="inputEmail3" class="col-sm-2 col-form-label">Username</label>
              <div class="col-sm-10">
                <input type="text" class="form-control" id="inputEmail3" name="username">
              </div>
            </div>
            <div class="row mb-3">
              <label for="inputPassword3" class="col-sm-2 col-form-label">Password</label>
              <div class="col-sm-10">
                <input type="password" class="form-control" id="inputPassword3" name="password">
              </div>
            </div>

            <button type="submit" class="btn btn-primary">Log in</button>
          </form>
        </div>
    </div>
</div>

{% endblock content %}
```

- B9: Viết sự kiện trong login2/views.py → Sử dụng phương thức Login của Django

```
from django.shortcuts import render, redirect
from django.contrib.auth import authenticate, login
from django.contrib import messages

# Create your views here.
def login2(request):
    # Tạo danh sách error
    errors = []

    if request.method == "POST":
        username = request.POST.get("username")
        password = request.POST.get("password")
        is_auth = authenticate(username=username, password=password)

        if is_auth:
            login(request, is_auth)
            return redirect("home:home")
        else:
            error = "Tài khoản hoặc mật khẩu không đúng"
            errors.append(error)
            messages.warning(request, error)

    context = {
```

```
        "errors": errors
    }

    return render(request, "login2/login.html", context)
```

- B10: Hiển thị lỗi đăng nhập trên HTML

```
{% extends "home/base.html" %}

<!-- Nội dung thay đổi -->
{% block content %}
            {% for error in errors %}
                <p> {{ error }} </p>
            {% endfor %}
{% endblock content %}
```

4. Xử lý LOGOUT:

- B1: Tạo định tuyến urls.py của logout2 trong Project
  → path('dang-xuat/', include('logout2.urls'))

- B2: Tạo định tuyến trong logout2/urls.py:

```
from django.urls import path
from . import views

app_name = "logout2"

urlpatterns = [
    path('', views.logout2, name="logout2"),
]
```

- B3: Viết sự kiện logout2 trong logout2/views.py

```
from django.shortcuts import render, redirect
from django.contrib.auth import logout

# Create your views here.
def logout2(request):
    logout(request)
    return redirect("login2:login2")
```

- B4: Chỉnh sửa lại trang base.html

```
<!-- Hiển thị tài khoản hiện có -->
{% if not request.user.is_authenticated %}
<!-- Đăng ký tài khoản -->
<li class="nav-item">
<a class="nav-link {% if request.resolver_match.url_name == 'register' %}
active {% endif %}" href="{% url 'register:register' %}"> Register </a>
</li>

<!-- Đăng nhập tài khoản -->
```

```
<li class="nav-item">
<a class="nav-link {% if request.resolver_match.url_name == 'login2' %} active
{% endif %}" href="{% url 'login2:login2' %}"> Login </a>
</li>

{% else %}
<p> {{request.user.firstname}} {{request.user.lastname}} </p>

<!-- Đăng xuất tài khoản -->
<li class="nav-item">
<a class="nav-link {% if request.resolver_match.url_name == 'logout2' %} active
{% endif %}" href="{% url 'logout2:logout2' %}"> Logout </a>
</li>

{% endif %}
```

5. **<u>Hiển thị dữ liệu thẻ theo tài khoản:</u>**
   - Trong webApp/views.py:
     - Xuất thư viện
       → ***from django.contrib.auth.decorators import login_required***
       → ***from django.conf import settings***

     - Thêm ***@login_required(login_url='dang-nhap/')*** hoặc ***@login_required(login_url=settings.LOGIN_URL)*** phía trên hàm sử dụng
   - Trong settings.py của Project:
     - Thêm LOGIN_URL = ***"/dang-nhap/"*** cuối cùng

Bài 16: Thiết kế đối tượng Customer, Partner

1. **Tổng quan:**
   - Tạo bảng CSDL chuyên quản lý về thông tin đăng nhập của khách hàng
   - Công cụ: Dbeaver

2. **Cách thức bước làm:**
   - B1: Tạo webApp mới tên customer → python manage.py startapp customer
   - B2: Khai báo webApp customer vào trong settings.py của Project chỗ INSTALLED_APPS
   - B3: Xây dựng models.py cho customer:

```
from django.db import models
from django.contrib.auth.models import User

# Create your models here.
class CustomerModel(models.Model):
    # Mỗi khách hàng chỉ có một user (1 - 1)
    user = models.OneToOneField(User, on_delete=models.PROTECT)

    full_name = models.CharField(max_length=100)
    phone_number = models.CharField(max_length=20)
    address = models.CharField(max_length=100)

    def __str__(self):
        return self.full_name
```

   - B4: Đưa model vào trong admin.py:

```
from django.contrib import admin
from .models import CustomerModel

# Register your models here.
admin.site.register(CustomerModel)
```

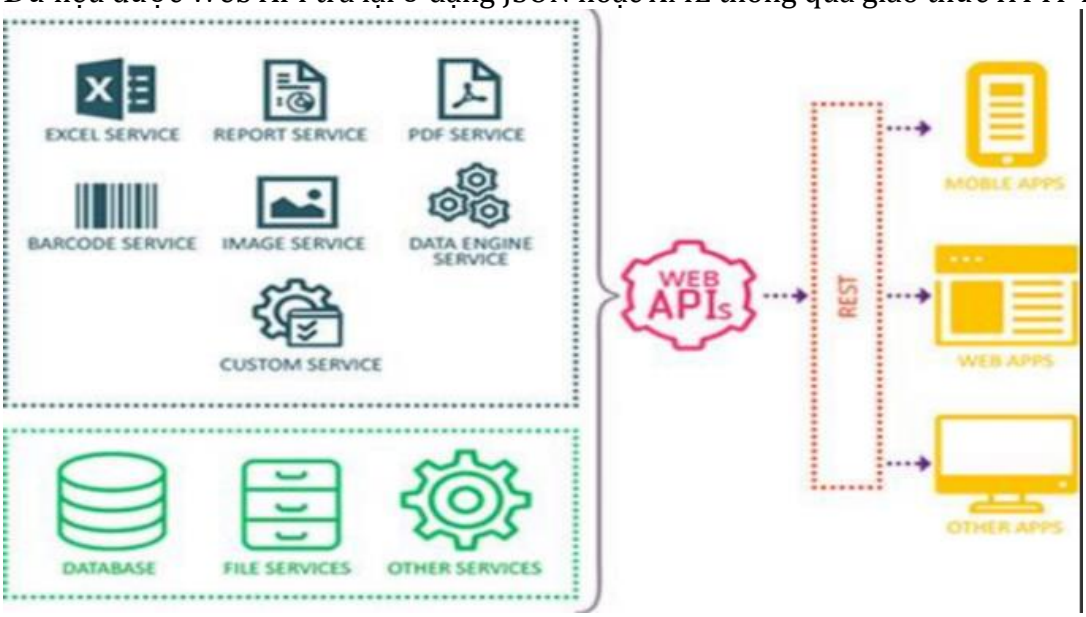
   - B5: Lưu lại thao tác bằng python manage.py makemigrations và python manage.py migrate
   - B6: Chỉnh sửa lại thông tin trong register/forms.py:

```
from typing import Any
from django import forms
from django.contrib.auth.models import User
import re
from customer.models import CustomerModel

class RegisterForm(forms.Form):
    # Các trường thông tin
    username = forms.EmailField(label="Email", required=True)
    password1 = forms.CharField(label="Mật khẩu", required=True,
widget=forms.PasswordInput)
    password2 = forms.CharField(label="Xác nhận mật khẩu", required=True,
widget=forms.PasswordInput)
```

```
    full_name = forms.CharField(label="Họ và Tên")

    phone_number = forms.CharField(label="Số điện thoại")
    address = forms.CharField(label="Địa chỉ")

    def clean_password2(self):
        if "password1" in self.cleaned_data:
            password1 = self.cleaned_data["password1"]
            password2 = self.cleaned_data["password2"]
            if password1 and password1 == password2:
                return password2
        raise forms.ValidationError("Mật khẩu không hợp lệ")

    def clean_username(self):
        username = self.cleaned_data["username"]

        # Kiểm tra tài khoản đã tồn tại chưa
        try:
            User.objects.get(username=username)
        except User.DoesNotExist:
            # Hợp lệ đăng ký
            return username
        raise forms.ValidationError("Email đã tồn tại trong hệ thống")

    # Lưu giá trị từ form vào Model
    def save(self):
        user = User.objects.create_user(
            username=self.cleaned_data["username"],
            password=self.cleaned_data["password1"],
        )

        # Xử lý cho khách hàng
        customer = CustomerModel(
            user = user,
            full_name=self.cleaned_data["full_name"],
            phone_number=self.cleaned_data["phone_number"],
            address=self.cleaned_data["address"]
        )
        customer.save()
```

Bài 17: RESTFUL Service với DRF

1. **API – Application Programming Interface:**
   - Giao diện lập trình ứng dụng là phần mềm trung gian cho phép hai ứng dụng khác nhau có thể giao tiếp với nhau
   - Cung cấp khả năng truy xuất đến một tập các hàm hay dùng
   - Dữ liệu được Web API trả lại ở dạng JSON hoặc XML thông qua giao thức HTTP hoặc HTTPS



2. **RESTful:**
   - Là kiểu thiết kế phần mềm cho các hệ thống web service
   - Sử dụng nguyên tắc của kiến trúc REST (Representational State Transfer) để phát triển các API cho các ứng dụng web
   - RESTful API là tiêu chuẩn trong việc thiết kế các API cho các ứng dụng web để quản lý các resource

3. **REST - REpresentational State Transfer:**
   - Là một dạng chuyển đổi cấu trúc dữ liệu, một kiểu kiến trúc để viết API
   - Sử dụng phương thức HTTP đơn giản để tạo cho giao tiếp giữa các máy
   - Thay vì sử dụng một URL cho việc xử lý một số thông tin người dùng, REST gửi một yêu cầu HTTP như GET, POST, PUT, DELETE đến một URL để xử lý dữ liệu

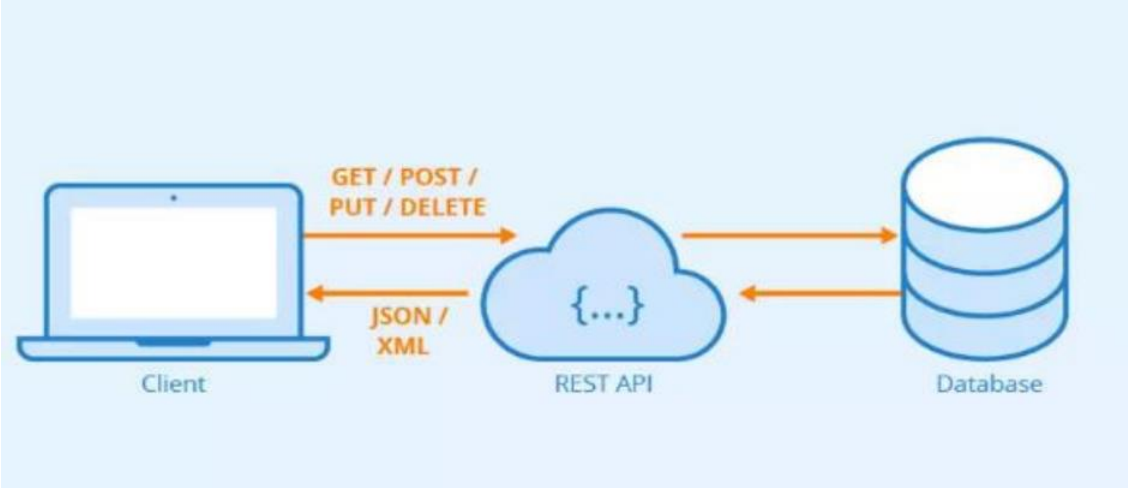
   - Các phương thức hoạt động của REST gọi là CRUD:
     - GET (SELECT) → trả về Resource hoặc một danh sách Resource
     - POST (CREATE) → tạo mới một Resource
     - PUT (UPDATE) → cập nhật thông tin cho Resource
     - DELETE → xóa một Resource

   - 4 nguyên tắc thiết kế:
     - Sử dụng các phương thức HTTP một cách rõ ràng
     - Phi trạng thái → các API độc lập lẫn nhau
     - Hiển thị cấu trúc thư mục như URLs

- Truyền tải JavaScript Object Notation (JSON), XML

4. RESTful API:

- Là tiêu chuẩn dùng trong việc thiết kế các API cho các ứng dụng web để quản lý các resource
- Tuân theo tất cả các quy tắc của kiến trúc REST
- Được coi là một phương pháp triển khai của kiến trúc REST
- Không quy định logic code ứng dụng
- Không giới hạn bởi ngôn ngữ lập trình ứng dụng, bất kì ngôn ngữ hoặc framework nào cũng có thể sử dụng để thiết kế một RESTful API
- Trang web tham khảo:
  - Django API: https://www.django-rest-framework.org/
  - Thiết kế API: https://petstore.swagger.io
  - Giả lập code: https://jsonplaceholder.typicode.com
  - Postman → công cụ kiểm tra đường định tuyến



5. DRF - Django Rest Framework:

- Là framework dành cho Django để xây dựng các API RESTful
- Cung cấp các công cụ để xây dựng các API Web bằng cách sử dụng các phương thức HTTP một cách rõ ràng và đáp ứng các tiêu chuẩn của kiến trúc REST
- Django REST framework viết bằng Python và được phát hành theo giấy phép BSD

6. Thực hành tạo các API với DRF trên Postman (GET, POST, PUT, DELETE):

- B1: Cài thư viện:
  - pip install djangorestframework
- B2: Khai báo 'rest_framework' trong INSTALLED_APPS của settings.py Project
- B3: Tạo webApp mới → python manage.py startapp partner
- B4: Khai báo 'parner' trong settings.py của Project
- B5: Tạo định tuyến urls.py cho 'partner'
  - Định tuyến project:

```
path('doi-tac/', include('partner.urls')),
```

  - Định tuyến partner:

```
from django.urls import path, include
from . import views
```

```
app_name = "partner"
urlpatterns = [
    path('list/', views.partner_list, name="partner_list"),
    path('<int:id>/', views.partner_detail, name="partner_detail"),
]
```

- B6: Xây dựng models.py của 'partner':

```
from django.db import models

# Create your models here.
class PartnerModel(models.Model):
    name = models.CharField(max_length=100)
    description = models.TextField(null=True)
    created_at = models.DateTimeField(auto_now_add=True)
    updated_at = models.DateTimeField(auto_now=True)

    def __str__(self):
        return self.name
```

- B7: Đưa model vào trong admin.py:

```
from django.contrib import admin
from .models import PartnerModel

# Register your models here.
admin.site.register(PartnerModel)
```

- B8: Lưu lại thao tác bằng python manage.py makemigrations và python manage.py migrate

GET → SELECT:

- B9: Viết API trong views.py của 'partner':

```
from django.shortcuts import render
from rest_framework.decorators import api_view
from rest_framework.response import Response
from rest_framework import status

# Chỉ thực thi phương thức với @
@api_view(['GET'])
def partner_list(request):
    return Response(data={}, status=status.HTTP_200_OK)
```

- B10: Tạo serializers.py trong 'partner' tương tự như forms.py cho phép người dùng nhập thông tin

```
from rest_framework import serializers
from .models import PartnerModel

class PartnerSerializers(serializers.ModelSerializer):
    class Meta:
        model = PartnerModel
```

```
        # fields = ["name", "description"]
        fields = "__all__"
```

- B11: Sửa lại views.py của 'partner':

```
from django.shortcuts import render
from rest_framework.decorators import api_view
from rest_framework.response import Response
from rest_framework import status
from .models import PartnerModel
from .serializers import PartnerSerializers

# Chỉ thực thi phương thức với @
@api_view(['GET'])
def partner_list(request):
    # Lấy hết danh sách từ partner databases
    model = PartnerModel.objects.all()

    # Trả về JSON
    serializer = PartnerSerializers(model, many=True)
    return Response(data=serializer.data, status=status.HTTP_200_OK)

@api_view(['GET'])
def partner_detail(request, id):
   # Kiểm tra nếu id nhập sai
    try:
        model = PartnerModel.objects.get(id=id)
    except PartnerModel.DoesNotExist:
        return Response(data={"error": "invalid"}, status=status.HTTP_400_BAD_REQUEST)

    # Trả về JSON
    serializer = PartnerSerializers(model)
    return Response(data=serializer.data)
```

- B12: Sử dụng Postman gọi các phương thức để kiểm tra phương thức GET

POST → THÊM:

- B13: Tạo định tuyến cho POST trong urls.py:

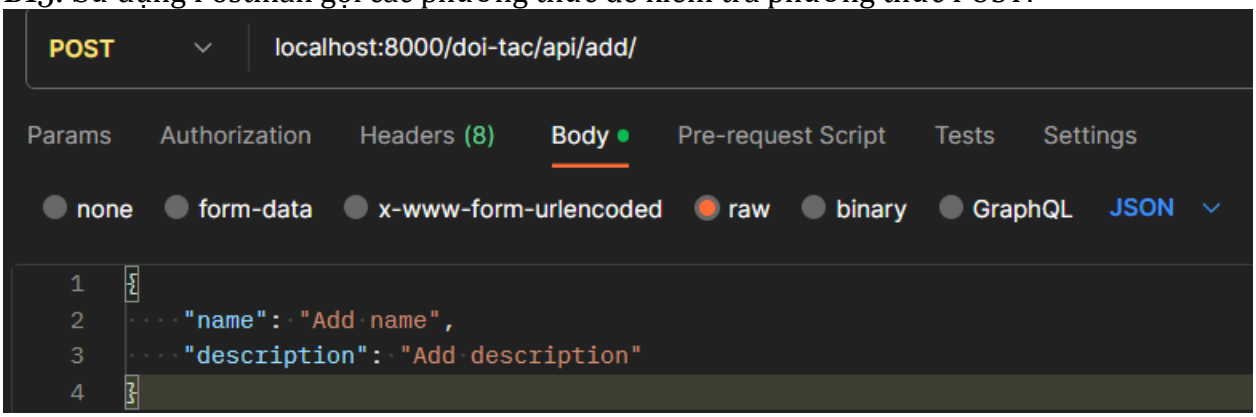
```
path('api/add/', views.partner_add, name="partner_addlist"),
```

- B14: Thêm phương thức trong views.py của 'partner':

```
from django.shortcuts import render
from rest_framework.decorators import api_view
from rest_framework.response import Response
from rest_framework import status
from .models import PartnerModel
from .serializers import PartnerSerializers

# GET
# Chỉ thực thi phương thức với @
```

```python
@api_view(['GET'])
def partner_list(request):
    # Lấy hết danh sách từ partner databases
    model = PartnerModel.objects.all()

    # Trả về JSON
    serializer = PartnerSerializers(model, many=True)
    return Response(data=serializer.data, status=status.HTTP_200_OK)

@api_view(['GET'])
def partner_detail(request, id):
    # Kiểm tra nếu id nhập sai
    try:
        model = PartnerModel.objects.get(id=id)
    except PartnerModel.DoesNotExist:
        return Response(data={"error": "invalid"}, status=status.HTTP_400_BAD_REQUEST)

    # Trả về JSON
    serializer = PartnerSerializers(model)
    return Response(data=serializer.data)

# POST
@api_view(['POST'])
def partner_add(request):
    if request.method == "POST":
        serializer = PartnerSerializers(data=request.data)
        if serializer.is_valid():
            serializer.save()
            return Response()
        else:
            return Response(serializer.errors, status=status.HTTP_400_BAD_REQUEST)
```

- B15: Sử dụng Postman gọi các phương thức để kiểm tra phương thức POST:



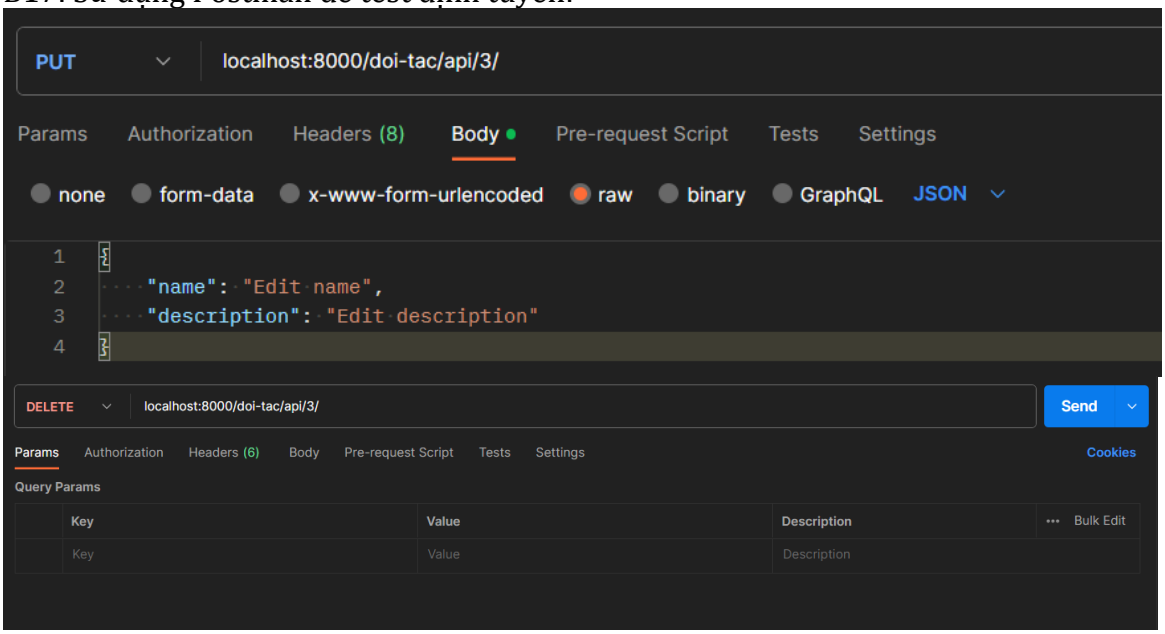
PUT → SỬA và DELETE → XÓA: (đã tối ưu định tuyến)

- B16: Dùng lại định tuyến partner_detail trong views.py:

```python
@api_view(['GET', 'PUT', 'DELETE'])
def partner_detail(request, id):
```

```
    # Kiểm tra nếu id nhập sai
    try:
        model = PartnerModel.objects.get(id=id)
    except PartnerModel.DoesNotExist:
        return Response(data={"error": "invalid"},
status=status.HTTP_400_BAD_REQUEST)

    # Chung cho cả GET và PUSH
    if request.method == "GET":
        # Trả về JSON
        serializer = PartnerSerializers(model)
        return Response(data=serializer.data)

    if request.method == "PUT":
        serializer = PartnerSerializers(model, data=request.data)
        if serializer.is_valid():
            serializer.save()
            return Response()
        else:
            return Response(serializer.errors,
status=status.HTTP_400_BAD_REQUEST)

    if request.method == "DELETE":
        model.delete()
        return Response()
```

- B17: Sử dụng Postman để test định tuyến:



7. <u>Tối ưu phương thức của API:</u>

- Mục tiêu: gộp cả hai GET và POST trong cùng một phương thức với cùng định tuyến (tiết kiệm được urls)
- Bước thực hiện:

- B1: Sửa lại định tuyến urls.py:

```
path('api/', views.partner_list, name="partner_list"),
```

- B2: Gộp hai phương thức GET và POST trong cùng 1 hàm:

```
# Chỉ thực thi phương thức với @
@api_view(['GET', 'POST'])
def partner_list(request):
    if request.method == "GET":
        # Lấy hết danh sách từ partner databases
        model = PartnerModel.objects.all()

        # Trả về JSON
        serializer = PartnerSerializers(model, many=True)
        return Response(data=serializer.data, status=status.HTTP_200_OK)

    if request.method == "POST":
        serializer = PartnerSerializers(data=request.data)
        if serializer.is_valid():
            serializer.save()
            return Response()
        else:
            return Response(serializer.errors, status=status.HTTP_400_BAD_REQUEST)
```

8. Hiển thị dữ liệu bằng API thay vì CSDL:

- B1: Tạo định tuyến mới trong urls.py:

```
path('api/', views.partner_list_api, name="partner_list_api"),
```

- B2: Hiển thị dữ liệu trên templates bằng API trong views.py:

```
import requests
# Tạo trang quản lý partner
def partner_list(request):
    # Lấy csdl từ API (qua thư viện requests)
    url = "http://localhost:8000/doi-tac/api/"
    respone = requests.get(url)
    partners = respone.json()

    context = {
        "partners": partners
    }
    return render(request, "partner/partner.html", context)
```

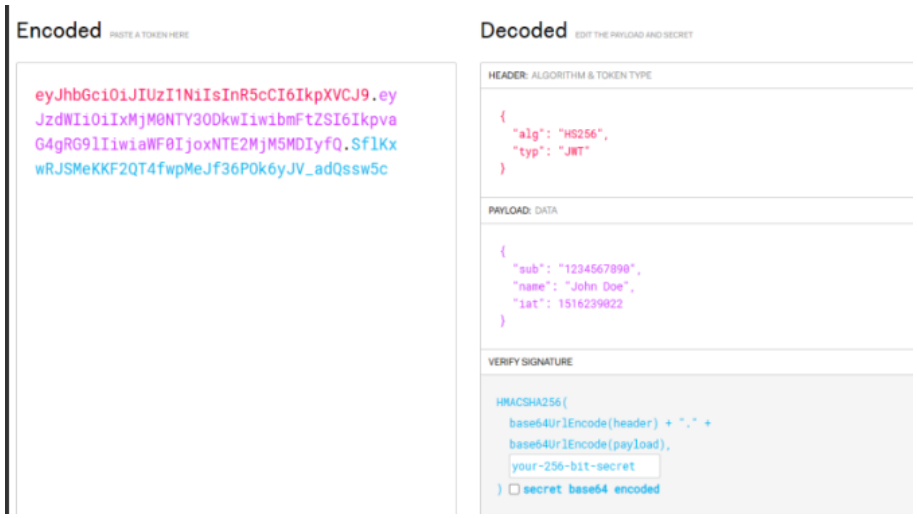
Bài 18: Bảo mật API với JWT

1. **Token:**
   - Đây là một chữ ký điện tử được mã hóa thành các con số và các con số này tạo thành một dãy số
   - Do được tạo dưới dạng OTP (One Time Password) nên loại mã này chỉ được tạo ngẫu nhiên và sử dụng trong một lần giao dịch khác nhau

2. **JWT – JSON Web Tokens:**
   - Là một chuỗi mã hóa được sử dụng như một phương tiện đại diện có khả năng thông qua chuỗi JSON để truyền đạt thông tin giữa server và client
   - Cấu trúc:

     - Màu đỏ → Header (chứa loại mã hóa)
     - Màu tím → Payload (chứa dữ liệu)
     - Màu xanh → Signature (Sử dụng Base64)
   - Trang web tham khảo: https://jwt.io/
   - Tài liệu hướng dẫn: https://django-rest-framework-simplejwt.readthedocs.io/en/latest/

3. **Cấu trúc JWT:**
   - JWT Header:
     - Chứa những thuật toán và kiểu dữ liệu được sử dụng cho mục đích mã hóa chuỗi JWT
     - Hai phần tử chính của Header:
       - Type: loại Token, mặc định là JSON Web Token
       - ALG (Algorithm): là thuật toán được sử dụng để mã hóa JWT
   - JWT Payload:
     - Là nơi sử dụng nội dung của thông tin (claim) mà người sử dụng muốn truyền đi ở bên trong chuỗi
     - Các thông tin này góp phần mô tả thực thể một cách đơn giản và nhanh chóng hoặc cũng có thể là các thông tin bổ sung thêm cho phần Header
   - SWT Signature:
     - Là phần chữ ký bí mật, được tạo ra bởi mã hóa phần Header cùng với phần Payload kèm theo đó là một chuỗi secret (khóa bí mật)
     - Khi kết hợp 3 phần Header, Payload, Signature lại với nhau sẽ thu được chuỗi JWR hoàn chính



4. Cách thức hoạt động của JWT:

- Dùng để xác thực (Authentication) API và ủy quyền (Authorization)
- Khi client đăng nhập thì server sẽ tiến hành xác thực và trả về cho người dùng chuỗi JWT
- Mỗi khi client truy cập vào tài nguyên được bảo vệ, server sẽ xác thực chữ ký JWT
- Sau khi xác minh thành công, token này có thể gán hoặc từ chối quyền truy cập vào tài nguyên
- Mục đích: mã hóa được dữ liệu Token (cơ chế bảo mật JWT)

5. Các bước thực hiện:

- B1: Cài đặt thư viện → pip install djangorestframework-simplejwt
- B2: Viết phiên bản thư viện vào trong file requirements.txt
- B3: Khai báo REST_FRAMEWORK trong settings.py của Project

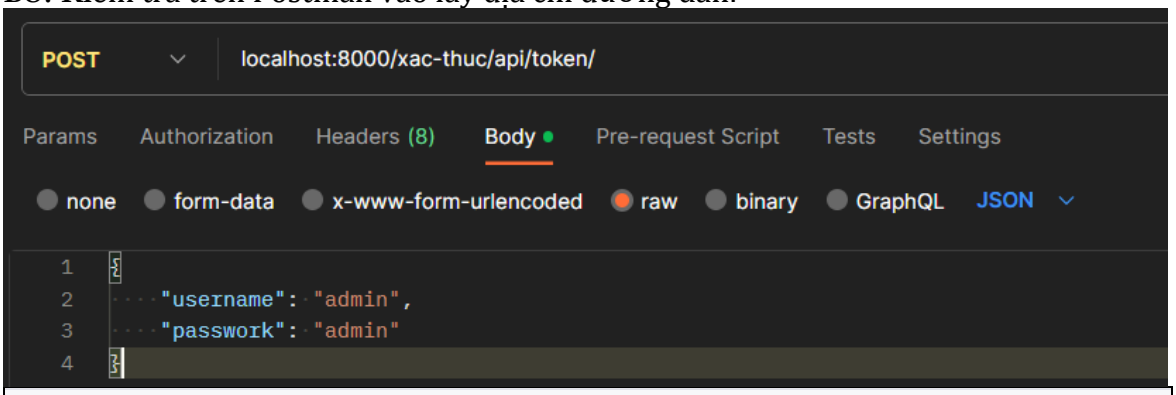
```
REST_FRAMEWORK = {
    'DEFAULT_AUTHENTICATION_CLASSES': (
        'rest_framework_simplejwt.authentication.JWTAuthentication',
    )
}
```

- B4: Tạo webApp mới để xác thực → python manage.py startapp auth2
- B5: Khai báo 'auth2' trong settings của Project với INSTALLED_APPS
- B6: Tạo urls.py cho 'auth2':
    - Trong project → path('xac-thuc/', include('auth2.urls')),
    - Trong webApp:

```
from django.urls import path, include
from . import views

app_name = "auth2"

from rest_framework_simplejwt.views import (
    TokenObtainPairView,
    TokenRefreshView,
)

urlpatterns = [
```

```
    path('api/token/', TokenObtainPairView.as_view(), name='token_obtain_pair'),
    path('api/token/refresh/', TokenRefreshView.as_view(), name='token_refresh'),
]
```

- B7: Chạy chương trình → python manage.py runserver
- B8: Kiểm tra trên Postman vào lấy địa chỉ đường dẫn:

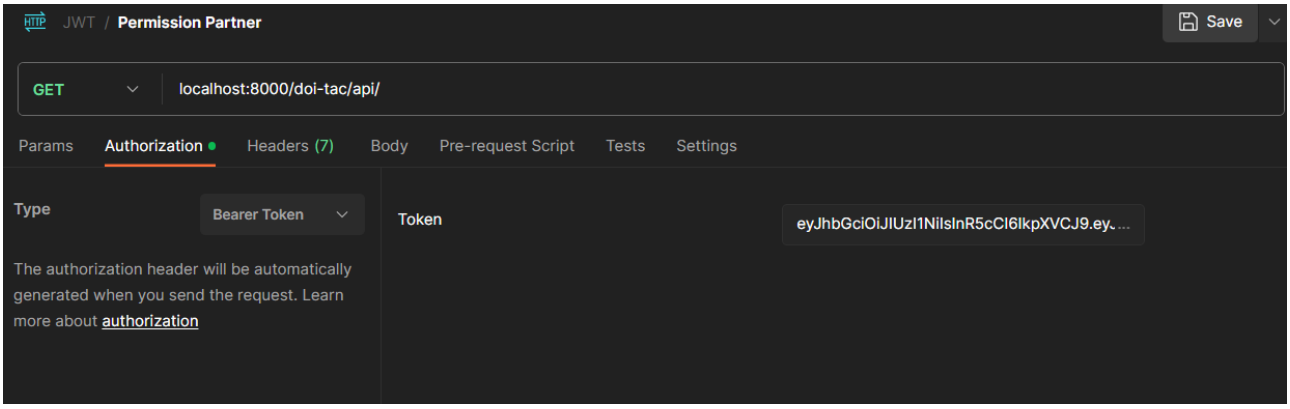



```
HTTP 200 OK
Allow: POST, OPTIONS
Content-Type: application/json
Vary: Accept

{
    "refresh": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ0b2tlbl90eXBlIjoicmVmcmVzaCIsImV4cCI6MTcwMTA2NzY1NywiaWF0IjoxNzAwOTgxMjU3LCJqdGkiOiI0YWUyOWM4MTdjNzk0ODM0YjViZWFkMGNkYjkwZTc0NCIsInVzZXJfaWQiOjF9.uWngNay62ep9GFLKfX8cWJ7onGJSsasfJAl0F5DEH1Y",
    "access": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ0b2tlbl90eXBlIjoiYWNjZXNzIiwiZXhwIjoxNzAwOTgxNTU3LCJpYXQiOjE3MDA5ODEyNTcsImp0aSI6Ijk0YTVjNTc0ODNmNzRkZWJhNzg5OTg5MmJkZDI5MWIwIiwidXNlcl9pZCI6MX0.NMEjroQ1yH0QMZRiDQC90RJcARI7NhQtsqa6GbpF7hU"
}
# Token luôn bị thay đổi sau 5p nên cần phải thực hiện lại
```

- B9: Copy đoạn mã "access" đưa vào https://jwt.io/ ở phần ENCODE
- B10: Vào partner/views.py để chỉnh sửa:

```
from rest_framework.permissions import IsAuthenticated
from rest_framework.decorators import api_view, permission_classes

# Chỉ thực thi phương thức với @
@api_view(['GET', 'POST'])
@permission_classes([IsAuthenticated])    # Cấu hình xác thực thông tin
def partner_list_api(request):
    ...
```

- B11: Test trên Postman → truyền lại accessToken vào Token để lấy được dữ liệu



- B12: Cài đặt thêm cấu hình JWT trong settings.py của Project:

```
from datetime import timedelta
from django.conf import settings

REST_FRAMEWORK = {
    'DEFAULT_AUTHENTICATION_CLASSES': (
        'rest_framework_simplejwt.authentication.JWTAuthentication',
    ),
    'DEFAULT_PERMISSION_CLASSES': (
        'rest_framework.permissions.IsAuthenticated',
    )
}

SIMPLE_JWT = {
    "ACCESS_TOKEN_LIFETIME": timedelta(minutes=60),
    "REFRESH_TOKEN_LIFETIME": timedelta(days=1),
    "ROTATE_REFRESH_TOKENS": False,
    "BLACKLIST_AFTER_ROTATION": False,
    "UPDATE_LAST_LOGIN": False,

    "ALGORITHM": "HS256",
    "SIGNING_KEY": settings.SECRET_KEY,
    "VERIFYING_KEY": "",
    "AUDIENCE": None,
    "ISSUER": None,
    "JSON_ENCODER": None,
    "JWK_URL": None,
    "LEEWAY": 0,

    "AUTH_HEADER_TYPES": ("Bearer",),
    "AUTH_HEADER_NAME": "HTTP_AUTHORIZATION",
    "USER_ID_FIELD": "id",
    "USER_ID_CLAIM": "user_id",
    "USER_AUTHENTICATION_RULE":
"rest_framework_simplejwt.authentication.default_user_authentication_rule",

    "AUTH_TOKEN_CLASSES": ("rest_framework_simplejwt.tokens.AccessToken",),
```

```
    "TOKEN_TYPE_CLAIM": "token_type",
    "TOKEN_USER_CLASS": "rest_framework_simplejwt.models.TokenUser",

    "JTI_CLAIM": "jti",

    "SLIDING_TOKEN_REFRESH_EXP_CLAIM": "refresh_exp",
    "SLIDING_TOKEN_LIFETIME": timedelta(minutes=5),
    "SLIDING_TOKEN_REFRESH_LIFETIME": timedelta(days=1),

    "TOKEN_OBTAIN_SERIALIZER":
"rest_framework_simplejwt.serializers.TokenObtainPairSerializer",
    "TOKEN_REFRESH_SERIALIZER":
"rest_framework_simplejwt.serializers.TokenRefreshSerializer",
    "TOKEN_VERIFY_SERIALIZER":
"rest_framework_simplejwt.serializers.TokenVerifySerializer",
    "TOKEN_BLACKLIST_SERIALIZER":
"rest_framework_simplejwt.serializers.TokenBlacklistSerializer",
    "SLIDING_TOKEN_OBTAIN_SERIALIZER":
"rest_framework_simplejwt.serializers.TokenObtainSlidingSerializer",
    "SLIDING_TOKEN_REFRESH_SERIALIZER":
"rest_framework_simplejwt.serializers.TokenRefreshSlidingSerializer",
}
```

Bài 19: Một số cấu hình và chức năng giỏ hàng

1. **Tạo trang web lỗi 404:**
   - Mục đích: tạo một trang lỗi 404 hiển thị nếu người dùng nhập link url không tồn tại
   - Cách làm:
     - B1: Tạo file 404.html trong home/templates/home/404.html

```
{% extends "home/base.html" %}

<!-- Nội dung thay đổi -->
{% block content %}
<h2> Page Not Found (404) </h2>
{{ message }}
{% endblock content %}
```

     - B2: Tạo urls.py cho page404 trong Project:

```
# Thư viện 404
from django.conf.urls import handler404

urlpatterns = [
    ...
]

handler404 = 'home.views.error'
```

     - B3: Tạo function error cho page404 trong home/views.py:

```
from django.shortcuts import render, HttpResponse

def error(request, exception):
   context = {
        "message": exception
    }
    return render(request, "home/404.html", context)
```

     - B4: Chỉnh sửa lại trong settings.py của Project:

```
DEBUG = False       # Nếu có tạo page 404
ALLOWED_HOSTS = ["*"]
```

2. **Tạo chức năng giỏ hàng:**
   - B1: Tạo giỏ hàng Cart trong home/templates/home/base.html

```
<!-- Hiển thị tài khoản hiện có -->
{% if not request.user.is_authenticated %}
<!-- Đăng ký tài khoản -->
<li class="nav-item">
<a class="nav-link {% if request.resolver_match.url_name == 'register' %}
active {% endif %}" href="{% url 'register:register' %}"> Register </a>
</li>
```

```
<!-- Đăng nhập tài khoản -->
<li class="nav-item">
<a class="nav-link {% if request.resolver_match.url_name == 'login2' %} active
{% endif %}" href="{% url 'login2:login2' %}"> Login </a>
</li>

{% else %}
<p> {{request.user.firstname}} {{request.user.lastname}} </p>

<!-- Đăng xuất tài khoản -->
<li class="nav-item">
<a class="nav-link {% if request.resolver_match.url_name == 'logout2' %} active
{% endif %}" href="{% url 'logout2:logout2' %}"> Logout </a>
</li>

<!-- Tạo giỏ hàng -->
<a class="btn btn-success" href="#"> Cart </a>

{% endif %}
```

- B2: Vào product/product_list thêm chức năng Add to Cart:

```
<div class="card-body">
<h5 class="card-title"> {{item.product_name}} </h5>
<p class="card-text">{{item.summary}}</p>

<!-- Add to Cart -->
<button href="#" class="btn btn-success add-to-cart" data-item-id="{{item.id}}"
data-action="add">Add to Cart</button>

 </div>
```

- B3: Viết sự kiện add-to-cart trong static/js → cart.js:

```
var btnAddToCarts = document.getElementsByClassName("add-to-cart");
for (i=0; i<btnAddToCarts.length; i++) {
    btnAddToCarts[i].addEventListener("click", function() {
        var itemId = this.dataset.itemId
        var action = this.dataset.action
        console.log("itemId", itemId, "action", action)

        addToCart(itemId, action)
    });
}

// Gọi API: https://docs.djangoproject.com/en/4.2/howto/csrf/
function addToCart(itemId, action) {
    var url = "/cart/add_to_cart/"
    fetch(url, {
        method: "POST",
```

```
        headers: {
            "Content-Type": "application/json",
            "X-CSRFToken": csrftoken,
        },
        body: JSON.stringify({
            "itemId": itemId,
            "action": action
        })
    })
    .then((response)=>{
        return response.json()
    })
    .then((data)=>{
        console.log("data", data)
    })
    .catch((error)=>{
        console.log("error", error)
    })
}
```

- B4: Khai báo file cart.js trong base.html:

```
<head>
<script>
    // Bắt buộc
    function getCookie(name) {
    let cookieValue = null;
    if (document.cookie && document.cookie !== '') {
        const cookies = document.cookie.split(';');
        for (let i = 0; i < cookies.length; i++) {
            const cookie = cookies[i].trim();
            // Does this cookie string begin with the name we want?
            if (cookie.substring(0, name.length + 1) === (name + '=')) {
                cookieValue = decodeURIComponent(cookie.substring(name.length +
1));
                break;
            }
        }
    }
    return cookieValue;
}
const csrftoken = getCookie('csrftoken');
</script>
<head

<body>
<script src="{% static 'js/cart.js'%}"></script>
</body>
```

3. Tạo giao diện quản lý giỏ hàng:

- B1: Tạo webApp order và order_detail → khai báo trong settings.py của Project:
  - python manage.py startapp order
  - python manage.py startapp order
- B2: Tạo models.py cho 'order'

```python
from django.db import models
from customer.models import CustomerModel

# Create your models here.
class OrderModel(models.Model):
    customer = models.ForeignKey(CustomerModel, on_delete=models.PROTECT)
    order_number = models.CharField(max_length=50)
    status = models.IntegerField(default=0)
    description = models.CharField(max_length=200, null=True)
    order_at = models.DateTimeField(null=True)
    created_at = models.DateTimeField(auto_now_add=True)
    updated_at = models.DateTimeField(auto_now=True)

    def __str__(self):
        return self.order_number
```

- B3: Khai báo models trong admin.py của 'order':

```python
from django.contrib import admin
from .models import OrderModel

# Register your models here.
admin.site.register(OrderModel)
```

- B4: Tương tự với models.py của 'order_detail':

```python
from django.db import models
from order.models import OrderModel
from product.models import ProductModel

# Create your models here.
class OrderDetailModel(models.Model):
    order = models.ForeignKey(OrderModel, on_delete=models.PROTECT)
    product = models.ForeignKey(ProductModel, on_delete=models.PROTECT)
    price = models.IntegerField()
    quantity = models.IntegerField()
    created_at = models.DateTimeField(auto_now_add=True)
    updated_at = models.DateTimeField(auto_now=True)
```

- B5: python manage.py makemigrations và python manage.py migrate
- B6: Tạo thêm webApp 'cart' và khai báo trong settings.py của Project
  → python manage.py startapp cart
- B7: Tạo urls.py cho 'cart' trong Project và webApp:

Project	

```python
urlpatterns = [
    path('cart/', include('cart.urls')),
]
```

webApp ‘cart’	

```python
from django.urls import path
from . import views

app_name = "cart"

urlpatterns = [
    path('', views.cart, name='cart'),
    path('add_to_cart/', views.add_to_cart, name='add_to_cart'),
]
```

- B8: Thực hiện chức năng add_to_cart(request) trong views/py của ‘cart’:

```python
from django.shortcuts import render
import json
from django.http import JsonResponse

# Create your views here.
def add_to_cart(request):
    data = json.loads(request.body)
    product_id = data["itemId"]
    print(f"data: {data}")

    return JsonResponse({"data": data})
```

- B9: Thực hiện chỉnh sửa customer và order trong function ‘add_to_cart’ của webApp ‘cart’ trên views.py:

```python
from django.shortcuts import render
import json
from django.http import JsonResponse
from customer.models import CustomerModel
from order.models import OrderModel
from order_detail.models import OrderDetailModel
from product.models import ProductModel

# Create your views here.
def add_to_cart(request):
    data = json.loads(request.body)
    product_id = data["itemId"]
    print(f"data: {data}")

    # customer - cần phải đăng nhập hoặc đăng ký
    customer = CustomerModel.objects.get(user=request.user)

    # order - kiểm tra có hóa đơn dạng nháp nào (customer)
    order, created = OrderModel.objects.get_or_create(
        status = 0,
        customer = customer
    )

    # product
```

```
    product = ProductModel.objects.get(id=product_id)

    # order detail - hóa đơn chi tiết
    try:
        order_detail = OrderDetailModel.objects.get(
            order = order,
            product = product
        )
    except OrderDetailModel.DoesNotExist:
        order_detail = OrderDetailModel(
            order = order,
            product = product,
            price = product.price,
            quantity = 0
        )
    order_detail.quantity = order_detail.quantity + 1 if order_detail.quantity
else 1

    # Cập nhật lại giá
    order_detail.price = product.price
    order_detail.save()

    return JsonResponse({"data": data})
```

4. Hiển thị giỏ hàng từ đơn hàng:

- B1: Thêm chức năng hiển thị card danh sách trong cart/views.py:

```
from django.shortcuts import render
import json
from django.http import JsonResponse
from customer.models import CustomerModel
from order.models import OrderModel
from order_detail.models import OrderDetailModel
from product.models import ProductModel

# Hiển thị view của 'cart'
def cart(request):
    # Lấy thông tin
    customer = CustomerModel.objects.get(user=request.user)
    order = OrderModel.objects.filter(
        status = 0,
        customer = customer
    ).first()

    order_detail = OrderDetailModel.objects.filter(order=order)

    context = {
        'order': order,
        'order_detail': order_detail,
    }
```

```
    return render(request, 'cart/cart.html', context)
```

- B2: Tạo templates/cart/cart.html:

```
{% extends "home/base.html" %}

{% block content %}
<div class="container">
    <h1>Cart Page</h1>

    <section class="h-100 h-custom" style="background-color: #eee;">
        <div class="container py-5 h-100">
            <div class="row d-flex justify-content-center align-items-center h-100">
            <div class="col">
                <div class="card">
                <div class="card-body p-4">

                    <div class="row">

                    <div class="col-lg-7">
                        <h5 class="mb-3"><a href="{% url 'product:product_list' %}" class="text-
body"><i
                            class="fas fa-long-arrow-alt-left me-2"></i>Continue
shopping</a></h5>
                        <hr>

                        <div class="d-flex justify-content-between align-items-center mb-4">
                        <div>
                            <p class="mb-1">Shopping cart</p>
                            <p class="mb-0">You have {{order_detail|length}} items in your
cart</p>
                        </div>
                        </div>

                        {% for item in order_detail %}

                        <div class="card mb-3">
                        <div class="card-body">
                            <div class="d-flex justify-content-between">
                            <div class="d-flex flex-row align-items-center">
                                <div>
                                <img
                                    src="https://mdbcdn.b-cdn.net/img/Photos/new-
templates/bootstrap-shopping-carts/img1.webp"
                                    class="img-fluid rounded-3" alt="Shopping item"
style="width: 65px;">
                                </div>
                                <div class="ms-3">
                                <h5>{{item.product.name}}</h5>
                                <p class="small mb-0">{{item.product.description|safe}}</p>
```

```
                            </div>
                        </div>
                        <div class="d-flex flex-row align-items-center">
                            <div style="width: 30px;">
                            <h5 class="fw-normal mb-0">{{item.quantity}}</h5>
                            </div>
                            <div style="width: 120px;">
                            <h5 class="mb-0">{{item.product.price|floatformat:"0g"}}</h5>
                            </div>
                            <a href="#!" style="color: #cecece;"><i class="fas fa-trash-
alt"></i></a>
                        </div>
                        </div>
                    </div>
                    </div>

                    {% endfor %}

                </div>
                <div class="col-lg-5">

                    <div class="card bg-primary text-white rounded-3">
                    <div class="card-body">
                        <div class="d-flex justify-content-between align-items-center mb-4">
                        <h5 class="mb-0">THONG TIN DON HANG</h5>
                        <img src="https://mdbcdn.b-cdn.net/img/Photos/Avatars/avatar-6.webp"
                            class="img-fluid rounded-3" style="width: 45px;" alt="Avatar">
                        </div>

                        <p class="form-label" for="typeName">{{order.customer.last_name}}
{{order.customer.first_name}}</p>
                        <p class="form-label" for="typeName">{{order.customer.address}}</p>
                        <p class="form-label"
for="typeName">{{order.customer.phone_number}}</p>

                        <hr class="my-4">

                        <div class="d-flex justify-content-between">
                        <p class="mb-2">Subtotal</p>
                        <p class="mb-2"></p>
                        </div>

                        <div class="d-flex justify-content-between">
                        <p class="mb-2">Shipping</p>
                        <p class="mb-2">
                        </p>
                        </div>

                        <div class="d-flex justify-content-between mb-4">
```

```
                        <p class="mb-2">Total(Incl. taxes)</p>
                        <p class="mb-2">
                        </p>
                        </div>

                        {% if order.id %}
                        <form class="mt-4" action="{% url 'cart:order' order.id %}"
method="POST">
                            {% csrf_token %}
                            <div class="form-outline form-white mb-4">
                                <input type="text" id="typeName" class="form-control form-
control-lg" size="17"
                                placeholder="Ghi chu don hang" name="description" />
                            </div>
                            <input type="submit" type="button" class="btn btn-info btn-block
btn-lg" value="Xac nhan don hang" />
                        </form>
                        {% endif %}

                    </div>
                    </div>

                </div>

                </div>

            </div>
            </div>
        </div>
        </div>
    </div>
    </section>
</div>
{% endblock content %}
```

- B3: Tạo định tuyến cho chức năng 'Xác nhận mua hàng' trong cart/urls.py:

```
urlpatterns = [
    path('', views.cart, name='cart'),
    path('add_to_cart/', views.add_to_cart, name='add_to_cart'),
    path('order/<int:id>/', views.order, name="order"),
]
```

- B4: Tạo thêm chức năng 'Đặt hàng thành công' trong cart/views.py:

```
import uuid     # Sinh số ngẫu nhiên

# Trang đặt hàng thành công
def order(request, id):
    order = OrderModel.objects.get(id=id, status=0)
    order.status = 1
```

```
    order.description = request.POST.get("description")
    order.order_number = str(uuid.uuid4())      # Mã hóa đơn

    order_detail = OrderDetailModel.objects.filter(order=order)
    # Lưu lại giá
    for item in order_detail:
        item.price = item.product.price
        item.save()
    order.save()

    context = {
        'order': order
    }
    return render(request, 'cart/order.html', context)
```

- B5: Tạo trang 'Đặt hàng thành công' trong templates/cart/order.html:

```
{% extends "home/base.html" %}

<!-- Nội dung thay đổi -->
{% block content %}
<div class="container">
    <h1> Order Success Page </h1>
    <p> Mã đơn hàng: {{order.order_number}}</p>
</div>
{% endblock content %}
```

- B6: Hiện thị tổng số sản phẩm trên 'cart' trong cart/views.py:

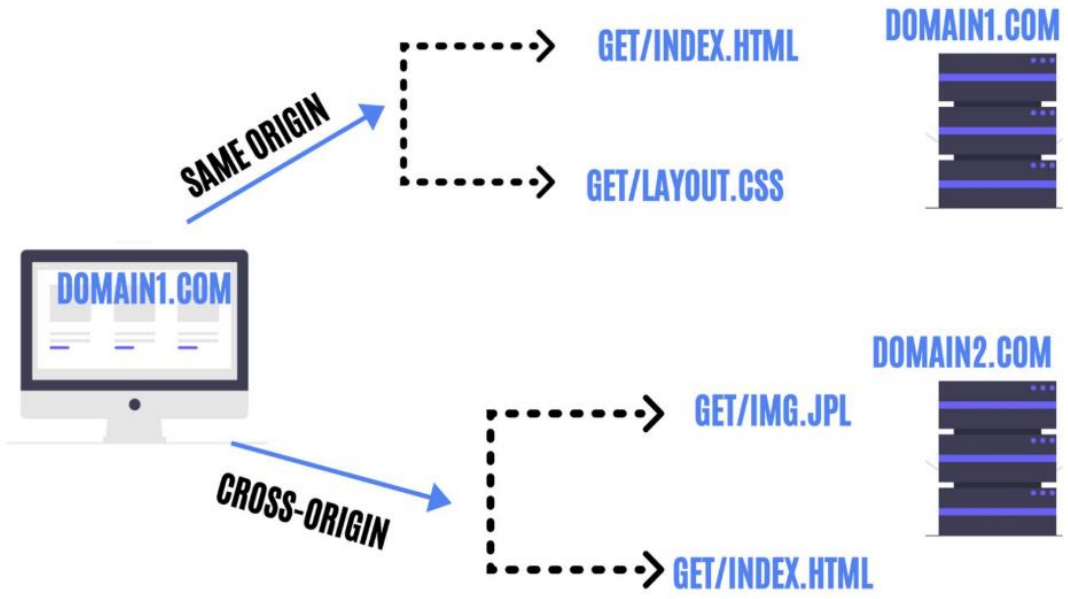
```
from django.db.models import Sum

# Tổng số item trong cart
def get_total_number_item_in_cart(customer):
     order = OrderModel.objects.filter(
        status = 0,
        customer = customer
    ).first()

     total_items =
OrderDetailModel.objects.filter(order=order).aaggregate(Sum('quantity'))
     total_items = list(total_items.values())[0]
     return total_items
```

- B7: Hiển thị tổng sản phẩm trên trang base.html:

```
<a class="btn btn-success" href="{% url 'cart:cart' %}"> Cart
     <span id="cart">{{cart}}</span>
</a>
```

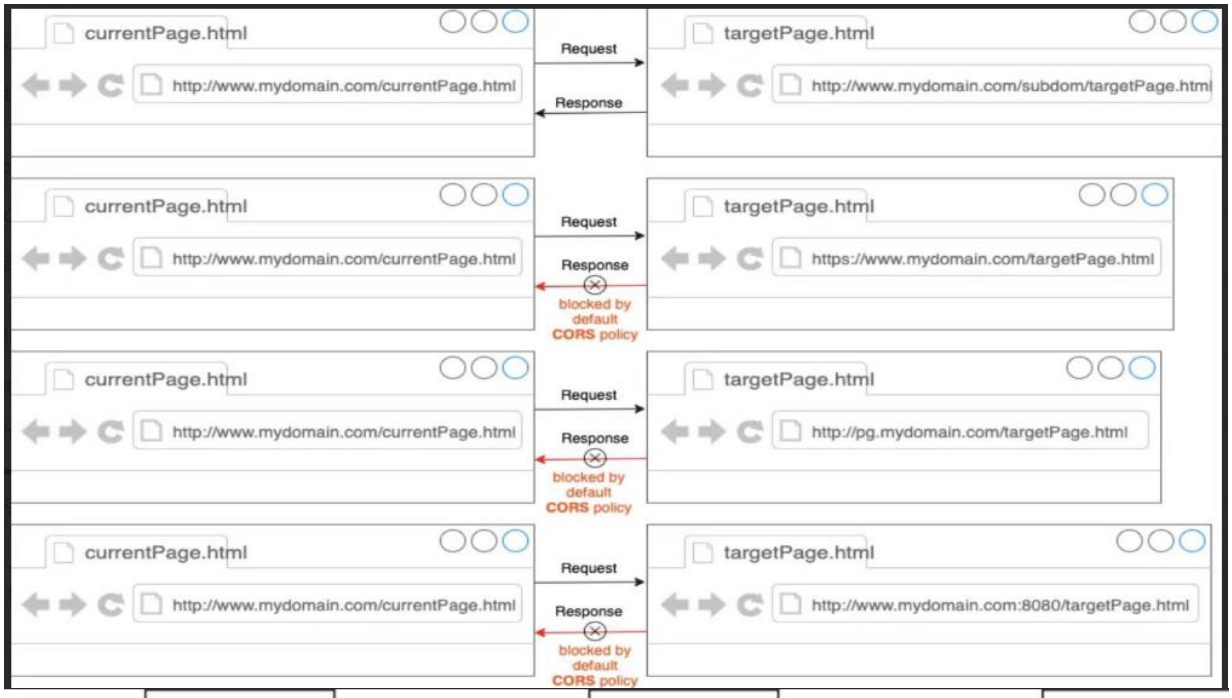
Bài 20: CORS và ghép nối API với Fontend

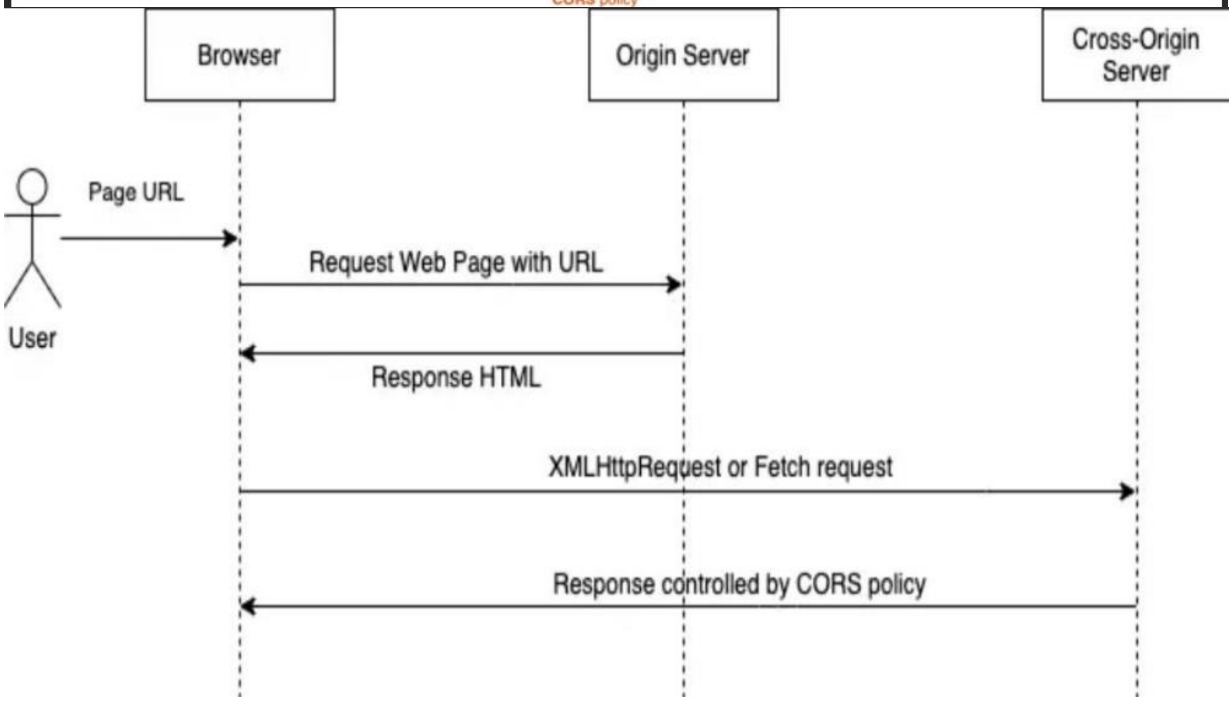
1. **CORS – Cross Origin Resource Sharing:**
   - Là một tiêu chuẩn để truy cập tài nguyên web trên các tên miền khác nhau
   - Cho phép các web scripts tương tác cởi mở hơn với nội dung bên ngoài tên miền gốc, dẫn đến sự tích hợp tốt giữa các dịch vụ web



   - Nguyên lý hoạt động:
     - Khi trình duyệt gửi một request đến một domain khác thì những request này sẽ được gắn thêm header có tên là origin để xác định origin của client
     - Giá trị này được thêm tự động bởi trình duyệt và không ai có thể thay đổi nó được
     - Header này đại diện cho nguồn gốc truy vấn
   - Mục đích:
     - Định tuyến khác miền ban đầu sẽ không cho phép truy cập vào đường link đó
     - Cho phép chạy lên nhiều cổng khác nhau

2. **CORS – Origin:**
   - Origin được cấu tạo dựa trên ba thành phần:
     - Protocol/Scheme: (Http/Https) → giao thức Skin
     - Host: server/domain
     - Port: cổng
   - Cơ chế hoạt động:
     - Server sẽ xem xét xem origin trong request có hợp lệ không, nếu hợp lệ sẽ trả về response kèm với header Access-Controll-Allow-Origin
     - Header này sẽ cho biết client có phải là hợp lệ không rồi mới tiếp tục thực hiện quá trình request
     - Access-Controll-Allow-Origin liệt kê các tên miền được phép thực hiện yêu cầu của CORS
     - Ký tự * cho phép tất cả các domain khác thực hiện yêu cầu





3. **<u>Các bước thực hiện CORS:</u>**
   - Tài liệu tham khảo: https://pypi.org/project/django-cors-headers/
   - Thực trạng tình huống:
     - B1: Tạo một trang cors.html để gọi API:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>
<body>
```

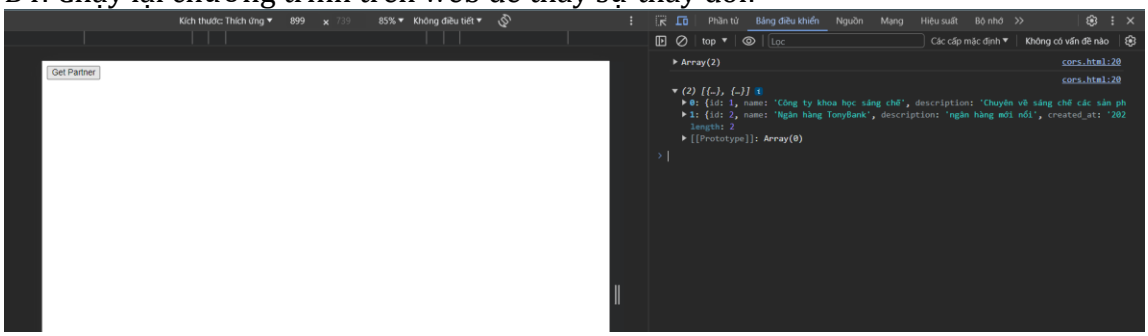
```
    <script
src="https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.min.js"></script>
    <script>
        function getPartners() {
  // token lấy từ encoding từ Postman hoặc http://localhost:8000/xac-thuc/api/token/
            var token =
"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ0b2tlbl90eXBlIjoiYWNjZXNzIiwiZXhwIjoxNzAwOTk
wNDk4LCJpYXQiOjE3MDA5ODY4OTgsImp0aSI6Ijg3NWM2ZmRiZTgyODRmN2ZiYjJiYTQyOTBlZTRhOGQ2Iiwi
dXNlcl9pZCI6MX0.AoIomi9EP2yYJCYWWV9aJmc54WmFbhxJuDwzxBY7JrY";
            $.ajax({
                url: "http://localhost:8000/doi-tac/api/",
                type: "GET",
                headers: {
                    "Authorization": "Bearer " + token
                },
                success: function(data) {
                    console.log(data)
                },
                error: function(error) {
                    console.log(error)
                }
            });
        }
    </script>

    <button onclick="getPartners()"> Get Partner </button>
</body>
</html>
```

- B2: Kiểm tra trên trình duyệt thấy bị lỗi

- Giải pháp cấu hình:
  - B1: Cài đặt thư viện → pip install django-cors-headers
  - B2: Điền phiên bản thư viện vừa cài vào file requirements.txt (kiểm tra pip list)
  - B3: Khai báo 'corsheaders' vào trong settings.py của Project mục INSTALLED_APPS và thêm 'corsheaders.middleware.CorsMiddleware' vào MIDDLEWARE:

```
MIDDLEWARE = [
    'django.middleware.security.SecurityMiddleware',
    'django.contrib.sessions.middleware.SessionMiddleware',
    'corsheaders.middleware.CorsMiddleware',
    'django.middleware.common.CommonMiddleware',
    'django.middleware.csrf.CsrfViewMiddleware',
    'django.contrib.auth.middleware.AuthenticationMiddleware',
    'django.contrib.messages.middleware.MessageMiddleware',
    'django.middleware.clickjacking.XFrameOptionsMiddleware',
]

CORS_ALLOW_ALL_ORIGINS = True
```

- B4: Chạy lại chương trình trên web để thấy sự thay đổi:

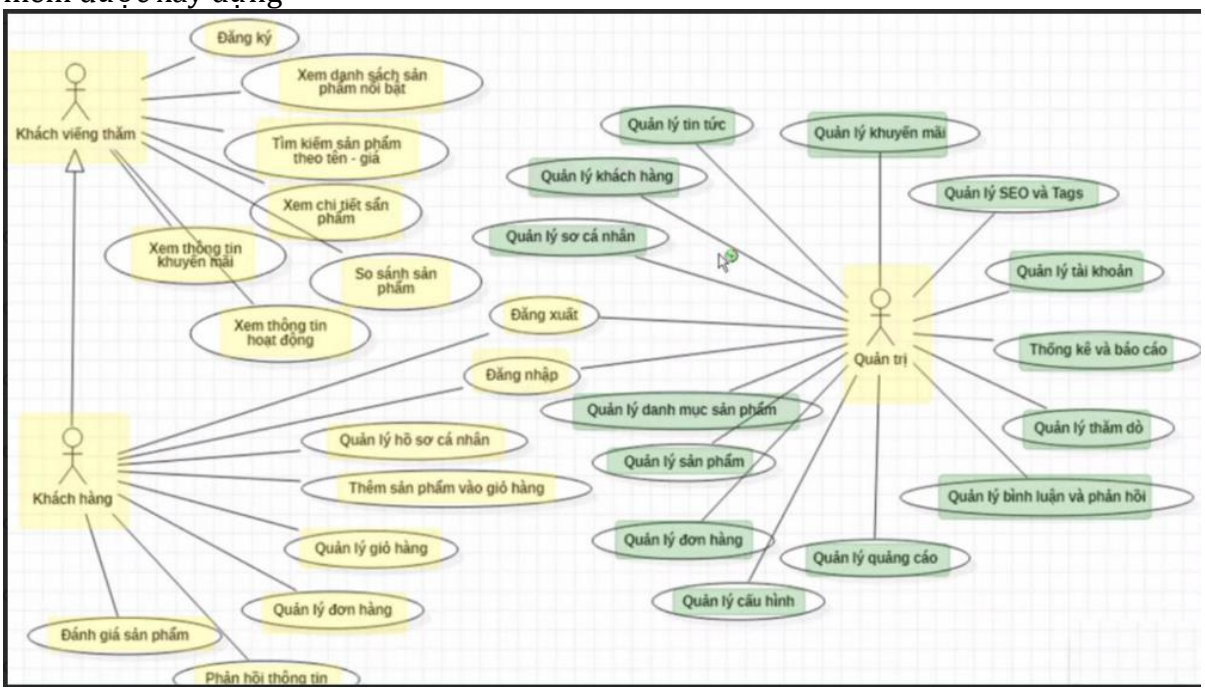


Bài 21: MINI PROJECT SINH VIÊN

1. **Đề bài:**
   - Cơ bản:
     - Sử dụng app sinh_vien với model sinh_vien (ho, ten, ngay_sinh, dia_chi, diem_toan, diem_van, diem_anh)
     - Tạo page hiển thị danh sách, thông tin chi tiết
   - Nâng cao:
     - Tạo page có chức năng nhập, sửa, xóa sinh viên
     - Trên page danh sách thực hiện tìm kiếm, phân trang

2. **Trình tự bước làm:**
   - B1: Mở terminal chạy lệnh → python manage.py startapp sinh_vien
   - B2: Vào settings.py của Project khai báo app "sinh_vien" ở mục INSTALLED_APPS
   - B3: Tạo urls.py của sinh_vien trong Project và trong sinh_vien
   - B4: Tạo view.py cho sinh_vien
   - B5: Tạo templates cho sinh_vien
   - B6: Xây dựng models.py cho sinh_vien
   - B7: Báo cáo model và trong admin.py để quản trị
   - B8: Truy vấn CSDL từ Admin để hiển thị lên templates

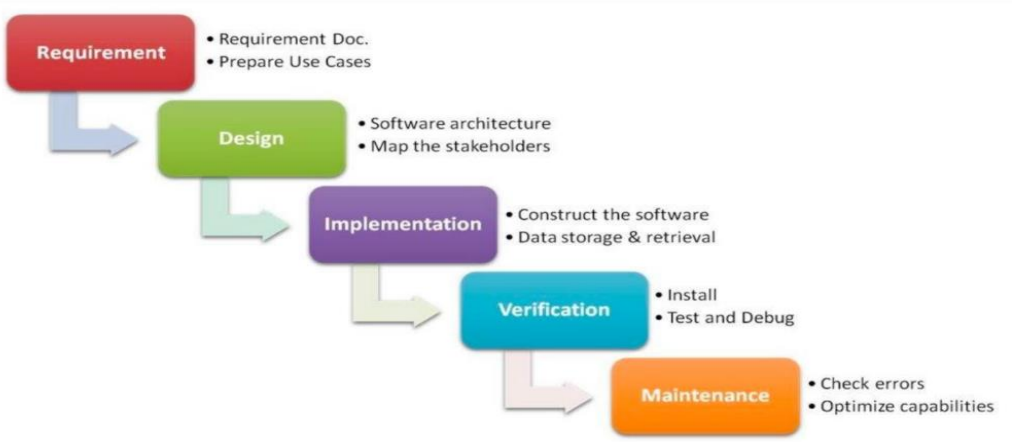
3. **Quy trình phát triển phần mềm:**
   - Software Development Methodology và Software Development Life Cycle là một quy trình có hệ thống để xây dựng phần mềm nhằm đảm bảo chất lượng và tính đúng đắn của phần mềm được xây dựng



   - Bao gồm nhiều giai đoạn khác nhau, tùy thuộc vào mô hình thực hiện:
     - Thu thập thông tin & Phân tích yêu cầu → Tổng quan dự án, đối tượng phục vụ, ước lượng thời gian

- Thiết kế (DB, màn hình, chức năng)
- Coding
- Testing
- Release

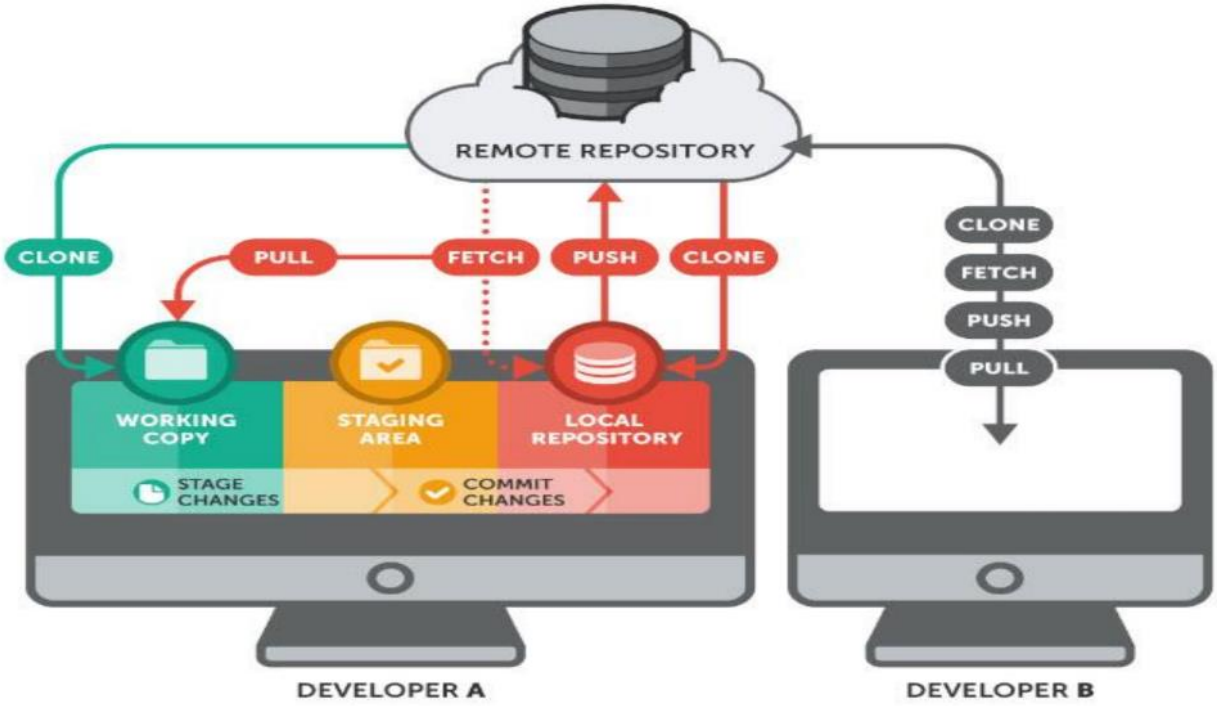
- Các mô hình:
  - Mô hình Thác nước (Waterfall Model):
    - Xác định yêu cầu
    - Thiết kế
    - Xây dựng (hay "triển khai", "mã hóa", "viết mã")
    - Liên kết
    - Kiểm thử và Chỉnh sửa
    - Cài đặt
    - Bảo trì



  - Phát triển phần mềm linh hoạt (Agile software development):

4. Deploy Application WSGI:

- WSGI (Web Server Gateway Interface) là một giao diện giữa máy chủ và ứng dụng web được viết bằng Python
- Cho phép các ứng dụng web Python chạy trên các máy chủ web chuẩn như Apache, Nginx, ...
- WSGI kết nối giữa máy chủ web với ứng dụng web Django → cung cấp hướng dẫn triển khai ứng dụng với WSGI như Gunicorn, uWSGI, Apache2, ...


REMOTE REPOSITORY
CLONE
PULL
FETCH
PUSH
CLONE
WORKING COPY
STAGING AREA
LOCAL REPOSITORY
STAGE CHANGES
COMMIT CHANGES
DEVELOPER A
CLONE
FETCH
PUSH
PULL
DEVELOPER B

Bài 22: DEPLOY PYTHONANYWHERE

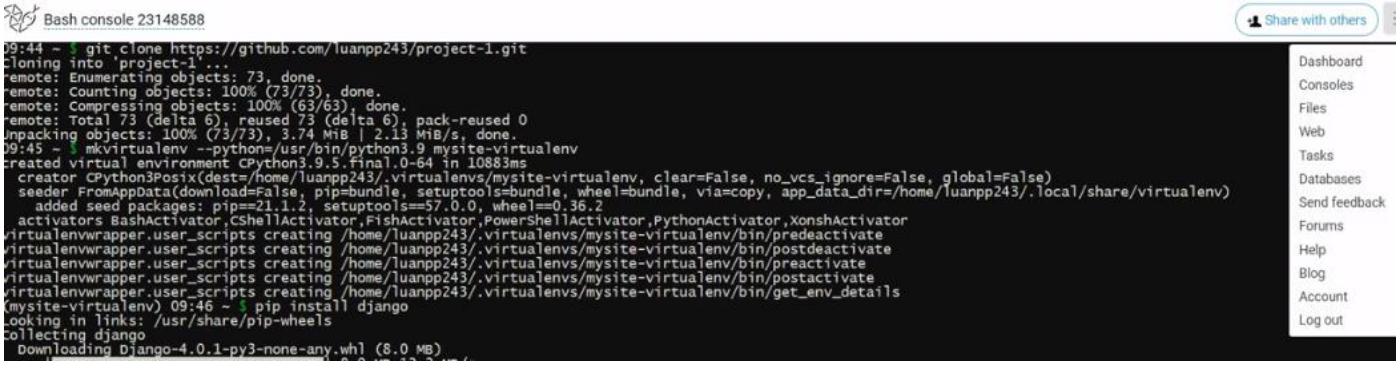
1. **Hướng dẫn cài đặt:**
   - Video: https://www.youtube.com/watch?v=wokG8tG6vhg&list=LL&index=1&t=617s
   - Pythonanywhere: https://help.pythonanywhere.com/pages/DeployExistingDjangoProject/

2. **Cách thức Deploy website:**
   - B1: Đẩy code lên github và copy link github chứa project đó
   - B2: Vào pythonanywhere và nhấp vào Console → Bash:

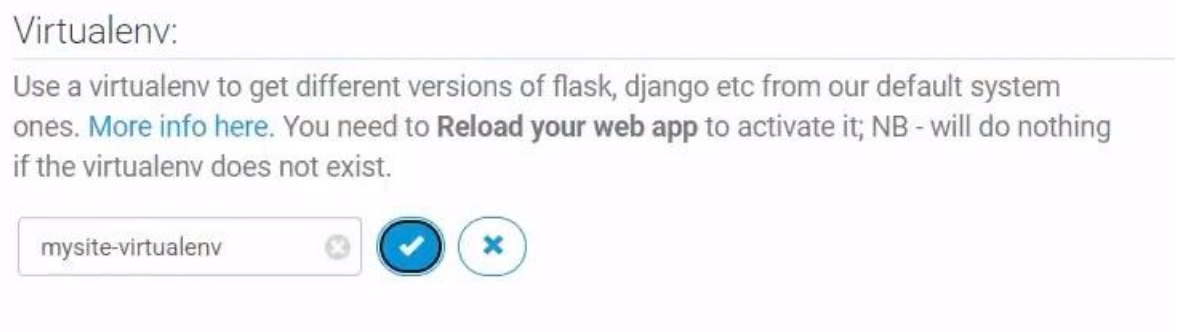
   - B3: Gõ các lệnh sau trên Bash Console:
     - git clone git@github.com:myusername/myproject.git (linkGithub của mình)
     - Tạo môi trường ảo:

```
$ mkvirtualenv --python=/usr/bin/python3.10 mysite-virtualenv
(mysite-virtualenv)$ pip install django
# or, if you have a requirements.txt:
(mysite-virtualenv)$ pip install -r requirements.txt
```

   - B4: Tạo webApp → Web:



     - Add a new web app → Manual Configuration



- B5: Chỉnh sửa file WSGI configuration file:
  - Xóa hết dữ liệu mặc định

```
# ++++++++++ DJANGO ++++++++++
# To use your own Django app use code like this:
import os
import sys

# assuming your Django settings file is at '/home/myusername/mysite/mysite/set
tings.py'
path = '/home/myusername/mysite'
if path not in sys.path:
    sys.path.insert(0, path)

os.environ['DJANGO_SETTINGS_MODULE'] = 'thư mục chứa fileSettings.settings'

## Uncomment the lines below depending on your Django version
###### then, for Django >=1.5:
from django.core.wsgi import get_wsgi_application
application = get_wsgi_application()
###### or, for older Django <=1.4
#import django.core.handlers.wsgi
#application = django.core.handlers.wsgi.WSGIHandler()
```

  - myusername là username của pythonanywhere
  - mysite là nameProject trên github

- B6: Chạy chương trình