Python

# The Complete Guide to Django REST Framework and Vue JS

## Introduction and Setup(pdf)

These slides are part of the downloadable resources of The Complete Guide to Django REST Framework and Vue JS Udemy course. Let’s keep in touch: ►YouTube Channel: <https://www.youtube.com/channel/UCxPWtz5N--X3IyYJ13Zr99A?sub_confirmation=1>

►Twitter: <https://www.twitter.com/pymike00>

►GitHub: <https://github.com/pymike00>

Chi tiết tham khảo file pdf phần 1

1. Install package in vs code

Shortcut of VS code:

CTRL SHIFT F: find

CTRL SHIFT E: file explorer

Install:

* Python
* Vetur
* Visual studio intellicode – Preview

File/ preferences/ Settings => gõ ruler => edit settings json

Source: <https://github.com/pymike00/The-Complete-Guide-To-DRF-and-VueJS>

Khái niệm về API đọc sau ở chap2

## Build Your First REST API with Django

ls: hiển thị danh sách file trong folder hiện tại

cài venv vào folder cha:

python -m venv venv

- Kích hoạt môi trường ảo trong win:

F:\Tai-lieu\Source-udemy\django\vue\_api\The-Complete-Guide-To-DRF-and-VueJS-master\02-WEB-APIs\venv\Scripts\activate.bat

python –version hay gõ python để check

pip list

pip install requests => cài để test

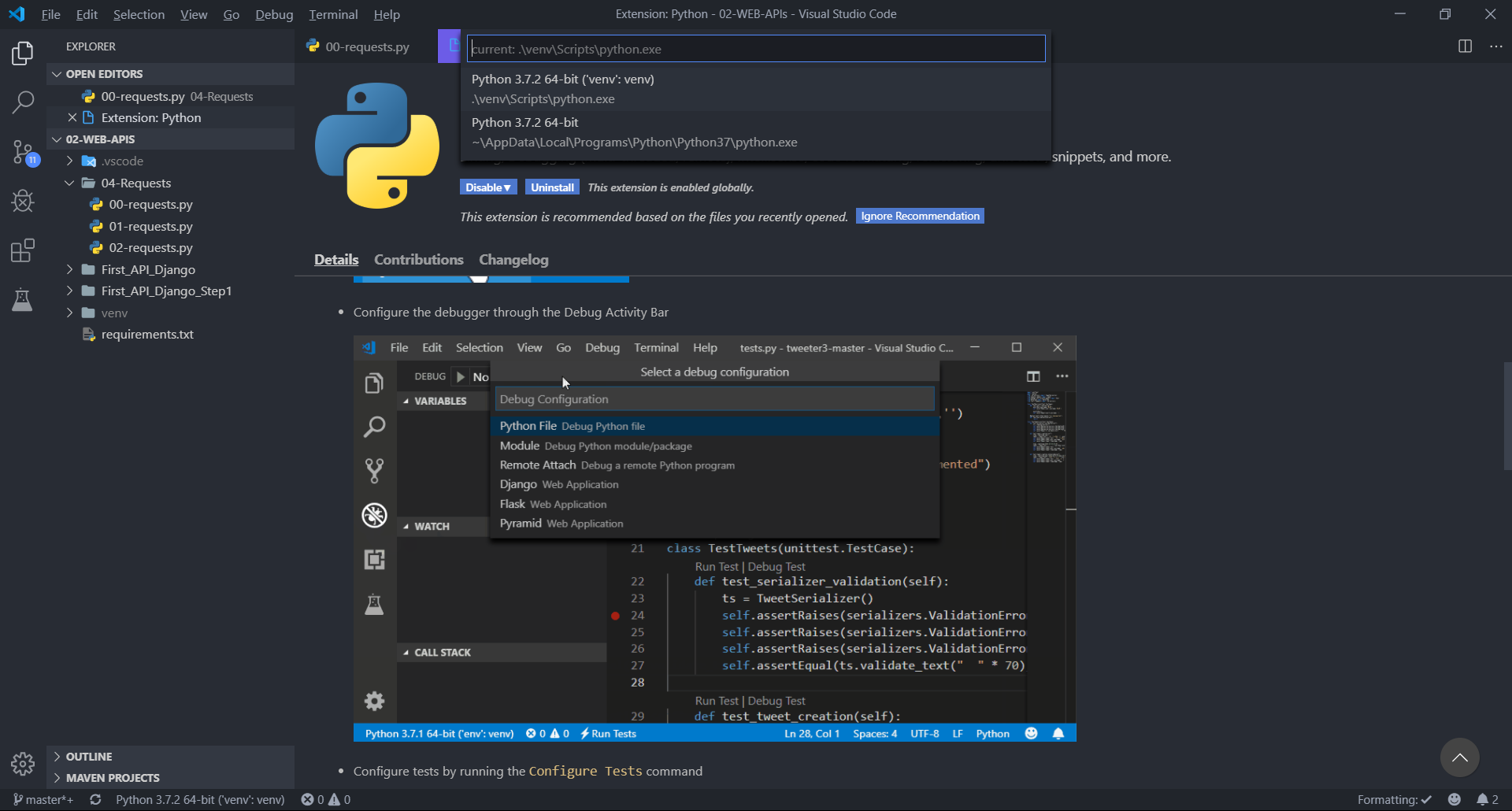
- Kích hoạt môi trường ảo trong mac:

Source venv/bin/activate

4. The Requests Module

Source : 00-requests.py

Đối với VS code nếu có thông báo cài pylink thì nhắp chọn Install



* Có thể ấn chuột phải vào file và chọn Run with python file

import requests

def main():

    response = requests.get("http://www.google.com")

*# response = requests.get("http://www.google.com/random-address/")*

    print("Status Code: ", response.status\_code)

*# print("Headers: ", response.headers)*

*# print("'Content-Type': ", response.headers['Content-Type'])*

    print("Content: ", response.text)

if \_\_name\_\_ == "\_\_main\_\_":

    main()

Lấy json theo tham số:

import requests

def main():

*# response = requests.get("https://api.exchangeratesapi.io/latest?base=USD&symbols=GBP")*

    payload = {"base": "USD", "symbols": "SEK"}

    response = requests.get("https://api.exchangeratesapi.io/latest",

                            params=payload)

    if response.status\_code != 200:

        print("Status Code: ", response.status\_code)

        raise Exception("There was an error!")

    data = response.json()

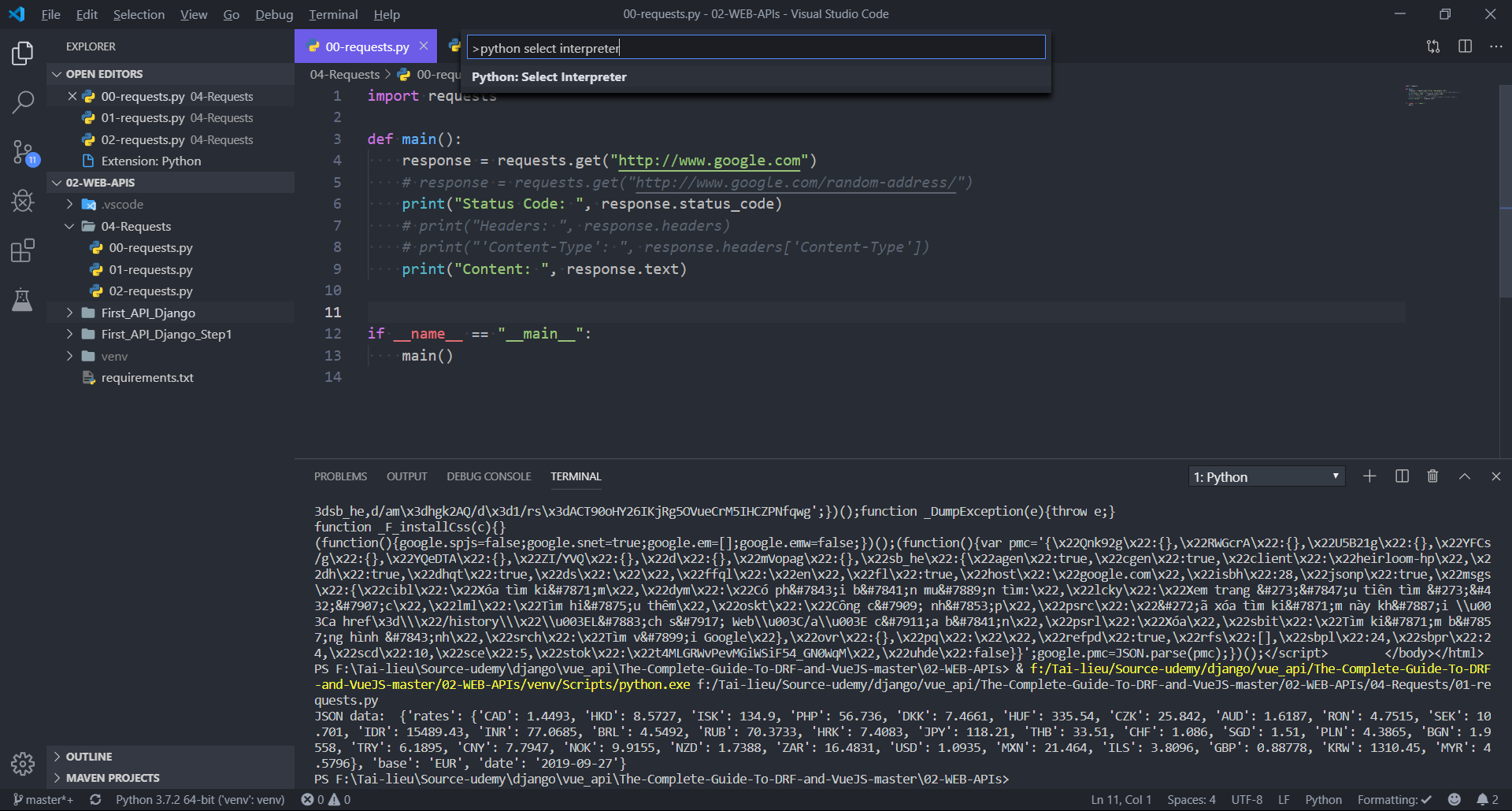
    print("JSON data: ", data)

if \_\_name\_\_ == "\_\_main\_\_":

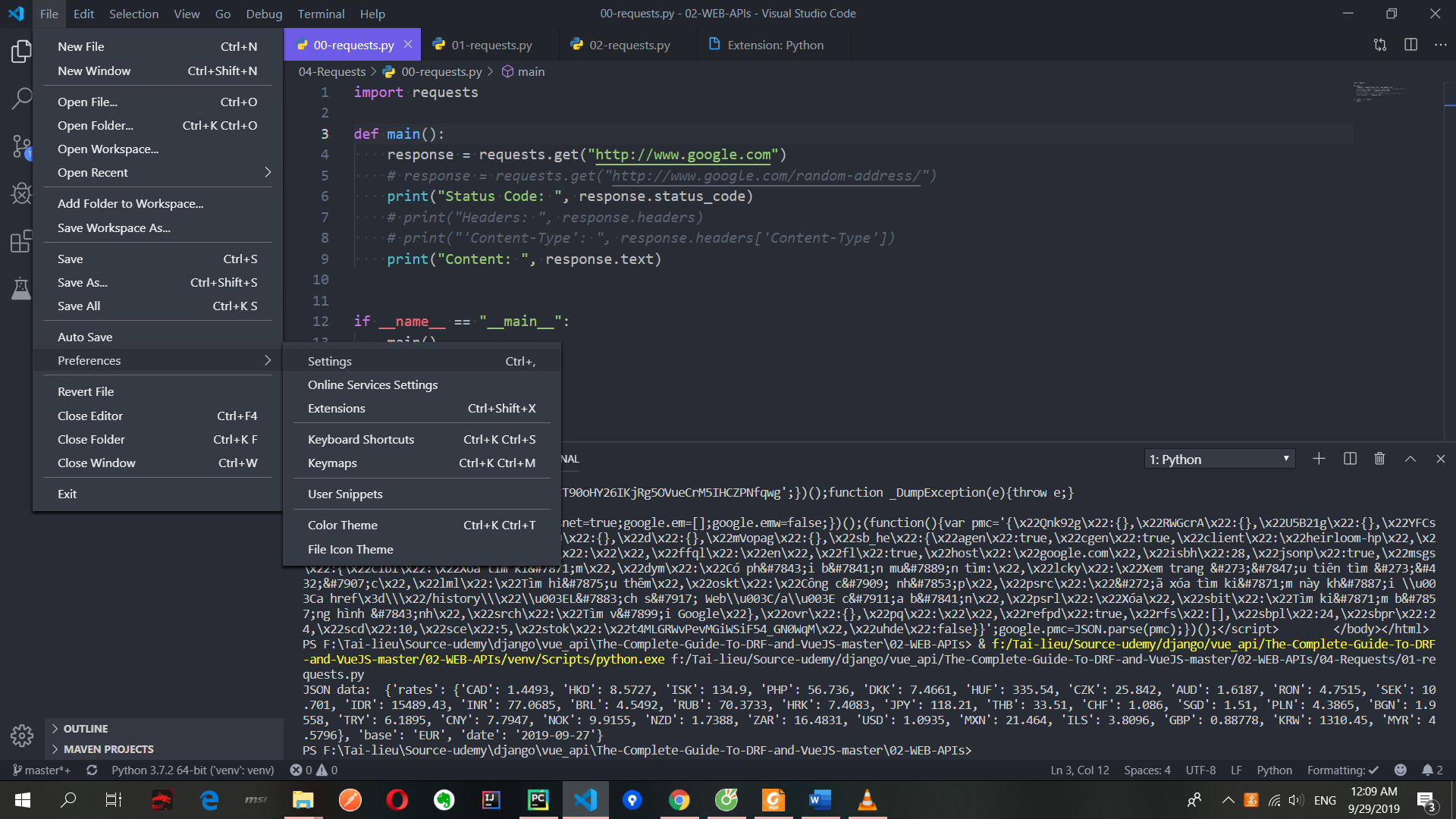
    main()

5. Your First Django API - Part One

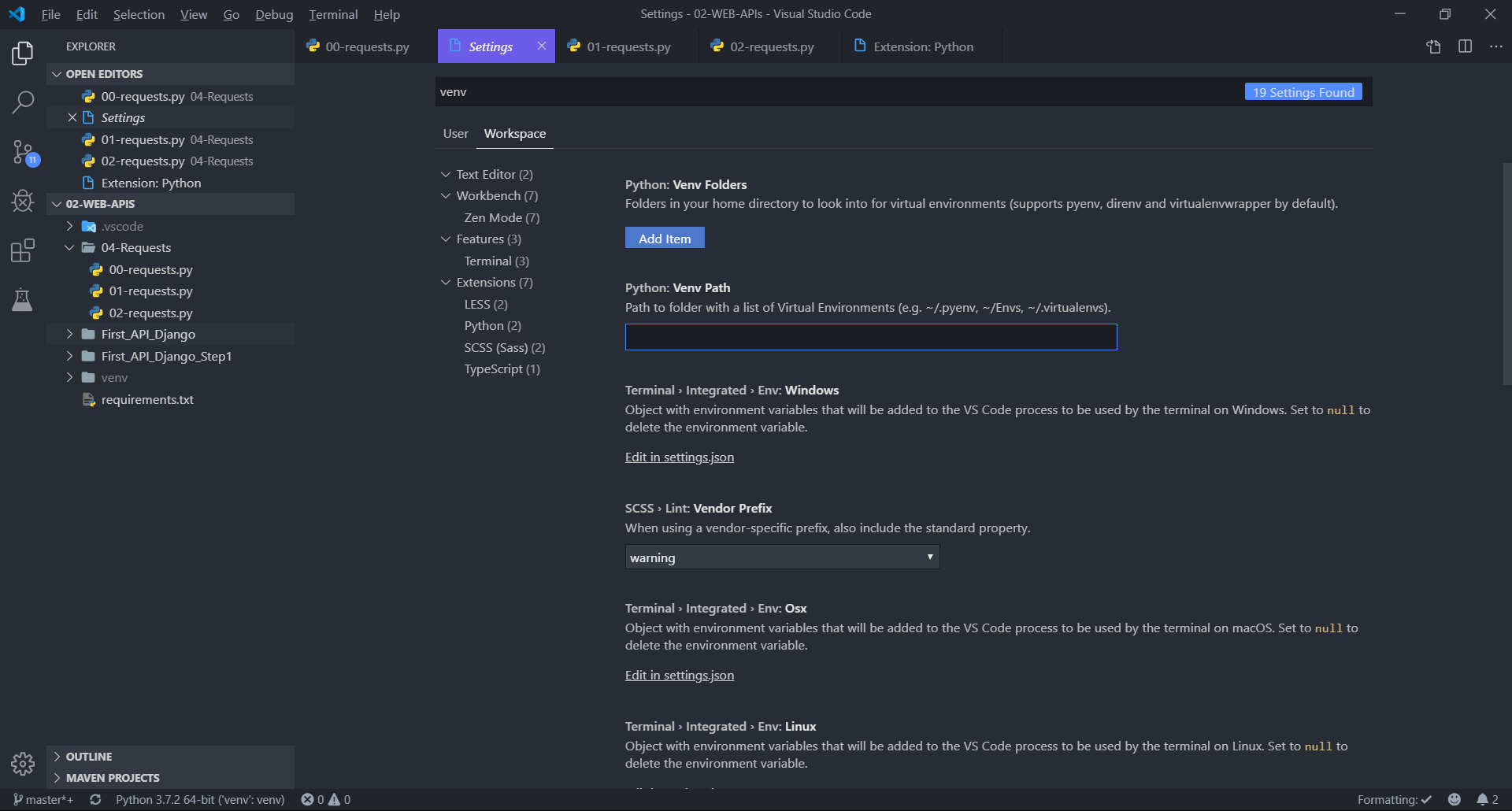
Để con fig cho VS code: Ấn **CTRL SHIFT P**



Sau đó chọn link đến venv local



Vào tab Workspace, gõ venv(nếu có lỗi mới làm bước này)



Trước hết phải active venv

pip install django

pip install pillow

pip freeze > requirements.txt

cd First-App

django-admin startproject onlinestore

cd onlinestore

clear

ls

python manage.py migrate # tạo file db.sqllite3

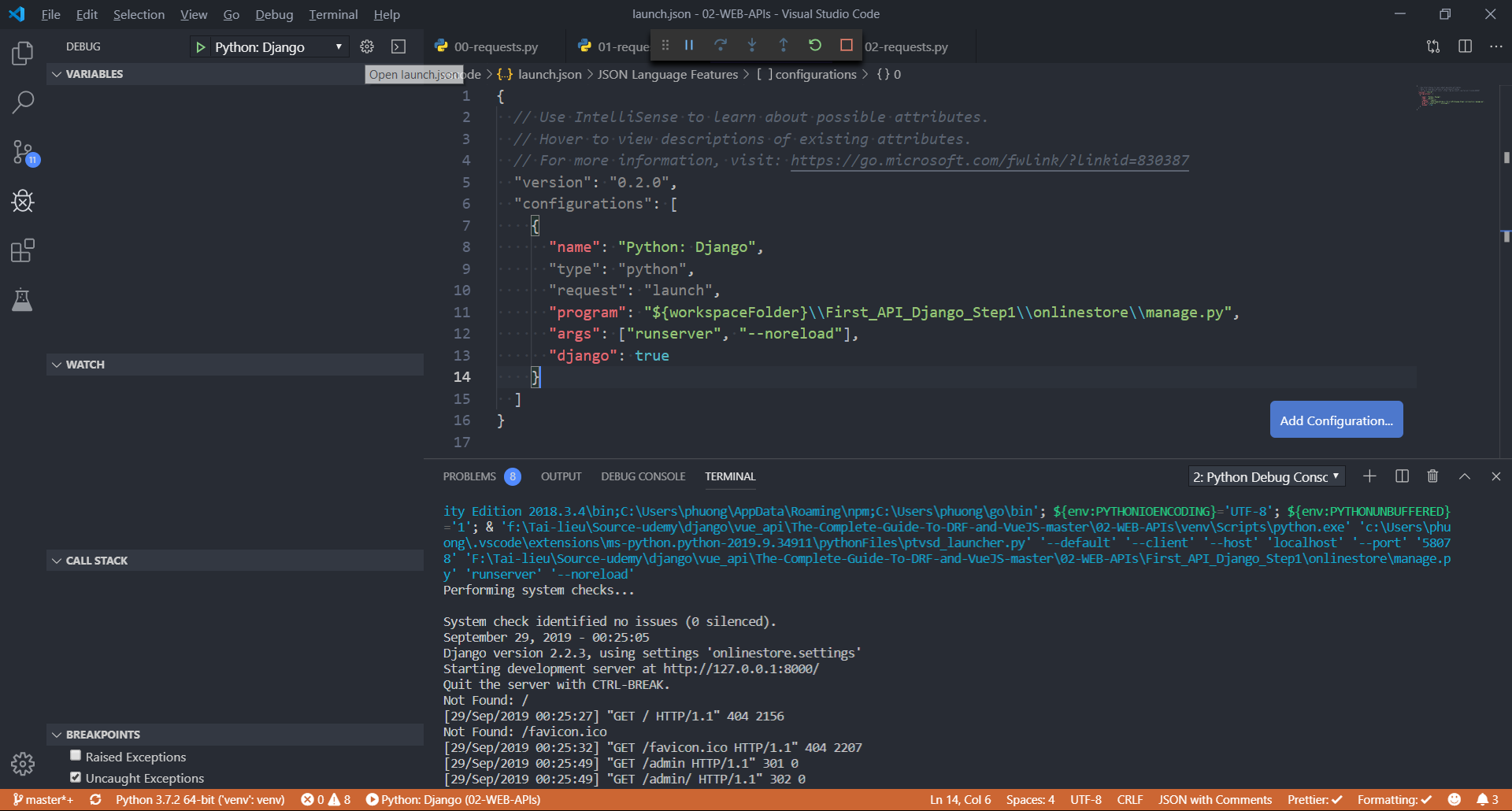
python manage.py createsuperuser

Nhập: admin

ấn enter 2 lần r chọn y (pass phải nhập - admin)

runserver

Để run trên vs code ta thực hiện như sau : Vào Tab debug/ setting/ chọn Python django/ set lại path đến file manage.py trong file launch.json



cd onlinestrore

python manage.py startapp products

Sau đó vào file settings.py thêm

*# Application definition*

INSTALLED\_APPS = [

    'django.contrib.admin',

    'django.contrib.auth',

    'django.contrib.contenttypes',

    'django.contrib.sessions',

    'django.contrib.messages',

    'django.contrib.staticfiles',

    'products' // add new app

]

Vào \First\_API\_Django\onlinestore\products\models.py tạo Model

from django.db import models

class Manufacturer(models.Model):

    name = models.CharField(max\_length=120)

    location = models.CharField(max\_length=120)

    active = models.BooleanField(default=True)

    def \_\_str\_\_(self):

        return self.name

class Product(models.Model):

    manufacturer = models.ForeignKey(Manufacturer,

                                     on\_delete=models.CASCADE,

                                     related\_name="products")

    name = models.CharField(max\_length=120)

    description = models.TextField(blank=True, null=True)

    photo = models.ImageField(blank=True, null=True)

    price = models.FloatField()

    shipping\_cost = models.FloatField()

    quantity = models.PositiveSmallIntegerField()

    def \_\_str\_\_(self):

        return self.name

Explain:

<https://medium.com/@doanhtu/h%C6%B0%E1%BB%9Bng-d%E1%BA%ABn-h%E1%BB%8Dc-django-ph%E1%BA%A7n-ii-models-api-v%C3%A0-admin-7dff23bf4306>

Mặc định thì Django sẽ sử dụng SQLite

* **ENGINE** — Bạn có các lựa chọn phổ biến sau: *‘django.db.backends.sqlite3’, ‘django.db.backends.postgresql’,’django.db.backends.mysql’, hoặc ‘django.db.backends.oracle’*. Còn nếu sử dụng Database khác thì hãy tham khảo ở link [này](https://docs.djangoproject.com/en/1.9/ref/databases/?source=post_page---------------------------#third-party-notes).( <https://docs.djangoproject.com/en/1.9/ref/databases/?source=post_page---------------------------#third-party-notes>)
* **NAME** — Tên của Database bạn đã tạo. Nếu dùng SQLite, thì đó sẽ là đường dẫn tuyệt đối (absolute path) đến file. Giá trị mặc định *os.path.join(BASE\_DIR,’db.sqlite3'),* sẽ lưu file ngay tại thư mục chứa project hiện tại.

Lưu ý: Nếu đang làm một site tiếng Việt, nhớ tạo một Database hỗ trợ UTF-8 đầy đủ.

**INSTALLED\_APPS** sẽ chứa tên tất cả các app của Django đang và sẽ dùng. Các app này có thể sử dụng trong nhiều project khác nhau, và bạn có thể tự làm các package của mình và chia sẻ cho người khác sử dụng. Ví dụ thì có thể vào tham khảo ở [**Awsome Django**](https://github.com/rosarior/awesome-django?source=post_page---------------------------).

Các app mặc định của Django sẽ được đặt đầu tiên:

* [django.contrib.admin](https://docs.djangoproject.com/en/2.0/ref/contrib/admin/?source=post_page---------------------------#module-django.contrib.admin): Chính là trang Admin.
* [django.contrib.auth](https://docs.djangoproject.com/en/2.0/topics/auth/?source=post_page---------------------------#module-django.contrib.auth): Dùng để Authentication
* [django.contrib.contenttypes](https://docs.djangoproject.com/en/2.0/ref/contrib/contenttypes/?source=post_page---------------------------#module-django.contrib.contenttypes): Làm việc với các model
* [django.contrib.sessions](https://docs.djangoproject.com/en/2.0/topics/http/sessions/?source=post_page---------------------------#module-django.contrib.sessions): Làm việc với *session*
* [django.contrib.messages](https://docs.djangoproject.com/en/2.0/ref/contrib/messages/?source=post_page---------------------------#module-django.contrib.messages): Dùng cho việc *thông/cảnh báo*
* [django.contrib.staticfiles](https://docs.djangoproject.com/en/2.0/ref/contrib/staticfiles/?source=post_page---------------------------#module-django.contrib.staticfiles): Dùng để quản lý *static file*

**$** python manage.py migrate

**migrate** sẽ tìm trong **INSTALLED\_APPS** xem có thằng nào dùng database không, để tạo các bảng tương ứng. Quá tiện, phải không nào. Và nó cũng tự động tạo các **migrations**tương ứng.

**Tạo các Model**

**Models** — hay nói cách khác là mô hình database của bạn. Hãy xem tạo chúng như thế nào nào.

***Triết lý đằng sau Model***

Model trong Django được thiết kế dựa trên DRY (Don’t repeat yourself). Nghĩa là, chỉ có một, và một file duy nhất trong mỗi app, được dùng để định nghĩa Model của app đó. Ở đây, mỗi model được thừa hưởng các tính chất của class **django.db.models.Model**. Mỗi model đều chứa một số biến, và nó đại diện cho các field trong database *(Hiểu một cách đơn giản thì nó chính là các cột trong các bảng cơ sở dữ liệu)*

Một vài Field sẽ có các argument bắt buộc. Ví dụ như CharField, thì bạn sẽ bắt buộc phải có *max\_length*. Cái này sẽ được giải thích sau. Và Field cũng có thể có các argument optional nữa, ví dụ như *default=0* trong phần votes.

Cuối cùng, để thể hiện sự tương quan giữa các model, chúng ta dùng **ForeignKey**. Nó sẽ cho Django biết rằng, mỗi Choice đều phải kết nối đến một Question nhất định. Django hỗ trợ tất cả các mối tương quan phổ biến nhất trong cơ sở dữ liệu, như: *many-to-one, many-to-many, và one-to-one.*

To include the app in our project, we need to add a reference to its configuration class in the [**INSTALLED\_APPS**](https://docs.djangoproject.com/en/2.0/ref/settings/?source=post_page---------------------------#std:setting-INSTALLED_APPS) setting. The **PollsConfig** class is in the **polls/apps.py** file, so its dotted path is **'polls.apps.PollsConfig'**. Edit the **mysite/settings.py** file and add that dotted path to the [**INSTALLED\_APPS**](https://docs.djangoproject.com/en/2.0/ref/settings/?source=post_page---------------------------#std:setting-INSTALLED_APPS) setting. It’ll look like this:

**$** python manage.py makemigrations polls

Kết quả:

Migrations for 'polls':  
 polls/migrations/0001\_initial.py:  
 - Create model Choice  
 - Create model Question  
 - Add field question to choice

By running **makemigrations**, you’re telling Django that you’ve made some changes to your models (in this case, you’ve made new ones) and that you’d like the changes to be stored as a migration.

Migrations are how Django stores changes to your models (and thus your database schema) — they’re just files on disk. You can read the migration for your new model if you like; it’s the file **polls/migrations/0001\_initial.py**. Don’t worry, you’re not expected to read them every time Django makes one, but they’re designed to be human-editable in case you want to manually tweak how Django changes things.

Và lệnh để chạy các migration đó và thay đổi database một cách tương ứng là migrate. Bạn có thể chạy **python manage.py** check để xem có lỗi gì không trước đã.

Nếu không có lỗi gì, vầy thì triển thôi.

**$** python manage.py migrate  
Operations to perform:  
Apply all migrations: admin, auth, contenttypes, polls, sessions  
Running migrations:  
Rendering model states... DONE  
Applying polls.0001\_initial... OK

Các bước sẽ như sau:

* Tạo thay đổi trong file models.py
* Chạy python manage.py makemigrations để tạo file migration tương ứng.
* Chạy python manage.py migrate để tạo các thay đổi tương ứng trong database.

Tham khảo: <https://docs.djangoproject.com/en/2.0/intro/tutorial02/?source=post_page---------------------------#activating-models>

**Generate migration**

python manage.py makemigrations

python manage.py migrate

# run

Python manage.py runserver

Sau đó vào file products/views.py tiếp tục chỉnh sửa

from .models import Product, Manufacturer

*# step 1*

from django.views.generic.detail import DetailView

from django.views.generic.list import ListView

class ProductDetailView(DetailView):

    model = Product

    template\_name = "products/product\_detail.html"

class ProductListView(ListView):

    model = Product

    template\_name = "products/product\_list.html"

Tạo folder templates/ tạo 2 file html

Product\_detail.html

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <meta http-equiv="X-UA-Compatible" content="ie=edge">

    <title>{{ object.name }}</title>

</head>

<body>

    <p>Product: {{ object.name }}</p>

    <p>Manufacturer: {{ object.manufacturer }}</p>

    <p>Quantity: {{ object.quantity }}</p>

    <p>Description: {{ object.description }}</p>

    <p>Price: {{ object.price }}</p>

    <p>Shipping Cost: {{ object.shipping\_cost }} Dollars</p>

    <img src="{{ object.photo.url }}" alt="product photo">

</body>

</html>

Product\_list.html

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <meta http-equiv="X-UA-Compatible" content="ie=edge">

    <title>Product List</title>

</head>

<body>

    {% for object in object\_list %}

        <p>Product: {{ object.name }}</p>

        <p>Manufacturer: {{ object.manufacturer }}</p>

        <p>Price: {{ object.price }}</p>

    {% endfor %}

</body>

</html>

Gõ html:5 để gợi ý nhanh

Chỉnh lại products/urls.py

from .views import ProductDetailView, ProductListView

from django.urls import path

urlpatterns = [

    path("products/", ProductListView.as\_view(), name="product-list"),

    path("products/<int:pk>/", ProductDetailView.as\_view(), name="product-detail"),

]

Xong chỉnh lại urls ở folder cha:

from django.conf.urls.static import static

from django.conf import settings

from django.contrib import admin

from django.urls import include, path

urlpatterns = [

    path('admin/', admin.site.urls),

    path("api/", include("products.urls"))

]

if settings.DEBUG:

    urlpatterns += static(settings.MEDIA\_URL, document\_root=settings.MEDIA\_ROOT)

Sau đó vào file settings.py thêm ở dưới cùng:

MEDIA\_ROOT = "uploads"

MEDIA\_URL = "/media/"

Tôi đang cố gắng tải lên hình ảnh qua quản trị viên Django và sau đó xem hình ảnh đó trong một trang trên giao diện người dùng hoặc chỉ qua một URL.

Vào file products/admin.py thêm đăng kí:

from django.contrib import admin

from .models import Product, Manufacturer

admin.site.register(Product)

admin.site.register(Manufacturer)

* Allow create instance in admin panel

Sau đó ta vào <http://127.0.0.1:8000/admin/> để thêm data(name = pass = admin)

6. Your First Django API - Part Two

Ở part 2 tiến hành chỉnh sửa view để trả về json khi gọi api:

from django.http import JsonResponse

from .models import Product, Manufacturer

def product\_list(request):

    products = Product.objects.all() *# [:30] vd muốn lấy 30 pt đầu*

    data = {"products": list(products.values())} *# products.values("pk", "name"): lấy những tt này thôi*

    response = JsonResponse(data)

    return response

def product\_detail(request, pk):

    try:

        product = Product.objects.get(pk=pk)

        data = {"product": {

                    "name": product.name,

                    "manufacturer": product.manufacturer.name,

                    "description": product.description,

                    "photo": product.photo.url,

                    "price": product.price,

                    "shipping\_cost": product.shipping\_cost,

                    "quantity": product.quantity,

                }}

        response = JsonResponse(data)

    except Product.DoesNotExist:

        response = JsonResponse({

            "error": {

                "code": 404,

                "message": "product not found!"

            }},

            status=404)

    return response

Sau đó chỉnh sửa lại urls.py

urlpatterns = [

    path("products/", product\_list, name="product-list"),

    path("products/<int:pk>/", product\_detail, name="product-detail"),

    path("manufacturers/", manufacturer\_list, name="manufacturer-list"),

    path("manufacturers/<int:pk>/", manufacturer\_detail, name="manufacturer-detail")

]

Assignment

def manufacturer\_list(request):

    manufacturers = Manufacturer.objects.filter(active=True) # lấy những obj có active là True

    data = {"manufacturers": list(manufacturers.values())}

    response = JsonResponse(data)

    return response

def manufacturer\_detail(request, pk):

    try:

        manufacturer = Manufacturer.objects.get(pk=pk)

        manufacturer\_products = manufacturer.products.all()

        data = {"manufacturer": {

                    "name": manufacturer.name,

                    "location": manufacturer.location,

                    "active": manufacturer.active,

                    "products": list(manufacturer\_products.values())

                }}

        response = JsonResponse(data)

    except Manufacturer.DoesNotExist:

        response = JsonResponse({

            "error": {

                "code": 404,

                "message": "manufacturer not found!"

            }},

            status=404)

    return response

Vào trang json formatter copy kết quả rồi paste vào cho dễ nhìn

## Django REST Framework - Level One

What is Django REST Framework?

Django REST Framework is a powerful and flexible toolkit for

building Web APIs with Python and Django

2. Introduction to DRF and NewsAPI Project Setup

We can install it via pip and include it in any existing Django Project just by

adding “rest\_framework” to the **INSTALLED\_APPS** list.

Cài đặt venv:

pip install django

pip install djangorestframework

pip freeze > requirements.txt

django-admin startproject newsapi

python manage.py startapp news

Sau đó vào file settings.py thêm

INSTALLED\_APPS = [

    'django.contrib.admin',

    'django.contrib.auth',

    'django.contrib.contenttypes',

    'django.contrib.sessions',

    'django.contrib.messages',

    'django.contrib.staticfiles',

    'rest\_framework',

    'news'

]

Sau đó tạo ở news/model class Article

from django.db import models

class Article(models.Model):

    author = models.CharField(max\_length=120)

    title = models.CharField(max\_length=120)

    description = models.CharField(max\_length=200)

    body = models.TextField()

    location = models.CharField(max\_length=120)

    publication\_date = models.DateField()

    active = models.BooleanField(default=True)

    created\_at = models.DateTimeField(auto\_now\_add=True)

    updated\_at = models.DateTimeField(auto\_now=True)

    def \_\_str\_\_(self):

        return f"{ self.author } { self.title }"

python manage.py makemigrations

python manage.py migrate

python manage.py createsuperuser

Vào file news/admin đăng ký instance

from django.contrib import admin

from news.models import Article

admin.site.register(Article)

# run

python manage.py runserver --noreload

Vào admin thêm object Article

3. What are the Serializers

<https://www.django-rest-framework.org/api-guide/fields/>

**Serializers** allow complex data such as querysets and model instances to be  
converted to native Python data types that can then be easily rendered into  
useful formats like JSON: this process is known as **Serialization** of Data.  
Serializers are a very important component of DRF, that we can easily use by  
employing the **Serializer** and **ModelSerializer** classes.  
Serializers also provide **deserialization**, allowing parsed data to be converted  
back into complex types, after first validating the incoming data

Tạo folder api trong folder new sau đó tạo class serialiers.py

from datetime import datetime

from django.utils.timesince import timesince

from rest\_framework import serializers

from news.models import Article

class ArticleSerializer(serializers.Serializer):

    id = serializers.IntegerField(read\_only=True)

    author = serializers.CharField()

    title = serializers.CharField()

    description = serializers.CharField()

    body = serializers.CharField()

    location = serializers.CharField()

    publication\_date = serializers.DateField()

    active = serializers.BooleanField()

    created\_at = serializers.DateTimeField(read\_only=True)

    updated\_at = serializers.DateTimeField(read\_only=True)

def create(self, validated\_data):

        print(validated\_data)

        return Article.objects.create(\*\*validated\_data)

    def update(self, instance, validated\_data):

        instance.author = validated\_data.get('author', instance.author)

        instance.title = validated\_data.get('title', instance.title)

        instance.description = validated\_data.get('description',

                                                  instance.description)

        instance.body = validated\_data.get('body', instance.body)

        instance.location = validated\_data.get('location', instance.location)

        instance.publication\_date = validated\_data.get('publication\_data',

                                                       instance.publication\_date)

        instance.active = validated\_data.get('active', instance.active)

        instance.save()

        return instance

Khi def 1 hàm có thể đặt pass ở thân hàm tạm thời cho không báo lỗi

def create(self, validated\_data):

pass

Sau đó gõ:

python manage.py shell

from news.models import Article

from news.api.serializers import ArticleSerializer

article\_instance = Article.objects.first()

article\_instance

serializer = ArticleSerializer(article\_instance)

serializer

serializer.data # trả về json

from rest\_framework.renderers import JSONRenderer

json = JSONRenderer().render(serializer.data) # chuyển True => true

import io

from rest\_framework.parsers import JSONParser

stream = io.BytesIO(json)

data = JSONParser().parse(stream) # => chuyển thành True

serializer = ArticleSerializer(data=data)

serializer.is\_valid() # True

serializer.validated\_data

serializer.save() # save new instance

exit()

Kết quả:

>>> from news.models import Article

>>> from news.api.serializers import ArticleSerializer

>>> article\_instance = Article.objects.first()

>>> article\_instance

<Article: phuong title> => obj được tạo ban đầu ở admin page

>>> serializer = ArticleSerializer(article\_instance)

>>> serializer => python native data type

ArticleSerializer(<Article: phuong title>):

id = IntegerField(read\_only=True)

author = CharField()

title = CharField()

description = CharField()

body = CharField()

location = CharField()

publication\_date = DateField()

active = BooleanField()

created\_at = DateTimeField(read\_only=True)

updated\_at = DateTimeField(read\_only=True)

>>> serializer.data

{'id': 1, 'author': 'phuong', 'title': 'title', 'description': 'no des', 'body': 'body phuong', 'location': 'vietnam', 'publication\_dat

e': '2019-07-28', 'active': True, 'created\_at': '2019-07-28T04:11:08.730900Z', 'updated\_at': '2019-07-28T04:11:08.730900Z'}

>>> from rest\_framework.renderers import JSONRenderer

>>> json = JSONRenderer().render(serializer.data)

>>> json

b'{"id":1,"author":"phuong","title":"title","description":"no des","body":"body phuong","location":"vietnam","publication\_date":"2019-0

7-28","active":true,"created\_at":"2019-07-28T04:11:08.730900Z","updated\_at":"2019-07-28T04:11:08.730900Z"}'

>>> import io

>>> from rest\_framework.parsers import JSONParser

>>> stream = io.BytesIO(json)

>>> data = JSONParser().parse(stream)

>>> data

{'id': 1, 'author': 'phuong', 'title': 'title', 'description': 'no des', 'body': 'body phuong', 'location': 'vietnam', 'publication\_dat

e': '2019-07-28', 'active': True, 'created\_at': '2019-07-28T04:11:08.730900Z', 'updated\_at': '2019-07-28T04:11:08.730900Z'}

>>> serializer = ArticleSerializer(data=data)

>>> serializer.is\_valid()

True

>>> serializer.validated\_data

OrderedDict([('author', 'phuong'), ('title', 'title'), ('description', 'no des'), ('body', 'body phuong'), ('location', 'vietnam'), ('p

ublication\_date', datetime.date(2019, 7, 28)), ('active', True)])

>>> serializer.save()

{'author': 'phuong', 'title': 'title', 'description': 'no des', 'body': 'body phuong', 'location': 'vietnam', 'publication\_date': datet

ime.date(2019, 7, 28), 'active': True}

<Article: phuong title>

>>> Article.objects.all()

<QuerySet [<Article: phuong title>, <Article: long title>, <Article: phuong title>]>

>>> exit()

4. The @api\_view Decorator - Part One

Now that we have learned how to use Serializers to serialize and deserialize  
data, it’s finally time to write our first **API View** to use them with!  
Django REST Framework provides two *wrappers* we can use to write API Views:  
● The **@api\_view** decorator, for working with Function Based API Views  
● The **APIView** class, for working with Class Based API Views

Tạo file **views**, **urls** trong folder **api**/

from rest\_framework import status

from rest\_framework.decorators import api\_view

from rest\_framework.response import Response

from rest\_framework.generics import get\_object\_or\_404

*# step 2 start*

@api\_view(["GET", "POST"])

def article\_list\_create\_api\_view(request):

    if request.method == "GET":

        articles = Article.objects.filter(active=True)

        serializer = ArticleSerializer(articles, many=True)

        return Response(serializer.data)

    elif request.method == "POST":

        serializer = ArticleSerializer(data=request.data)

        if serializer.is\_valid():

            serializer.save()

            return Response(serializer.data, status=status.HTTP\_201\_CREATED)

        return Response(serializer.errors, status=status.HTTP\_400\_BAD\_REQUEST)

@api\_view(["GET", "PUT", "DELETE"])

def article\_detail\_api\_view(request, pk):

    try:

        article = Article.objects.get(pk=pk)

    except Article.DoesNotExist:

        return Response({"error": {

                            "code": 404,

                            "message": "Article not found!"

                        }}, status=status.HTTP\_404\_NOT\_FOUND)

    if request.method == "GET":

        serializer = ArticleSerializer(article)

        return Response(serializer.data)

    elif request.method == "PUT":

        serializer = ArticleSerializer(article, data=request.data)

        if serializer.is\_valid():

            serializer.save()

            return Response(serializer.data)

        return Response(serializer.errors, status=status.HTTP\_400\_BAD\_REQUEST)

    elif request.method == "DELETE":

        article.delete()

        return Response(status=status.HTTP\_204\_NO\_CONTENT)

5. The @api\_view Decorator - Part Two

File urls.py trong folder api

from django.urls import path

from news.api.views import (article\_detail\_api\_view

                            ,article\_list\_create\_api\_view)

urlpatterns = [

    path("articles/", article\_list\_create\_api\_view, name="article-list"),

    path("articles/<int:pk>/", article\_detail\_api\_view, name="article-detail")

]

Sau đó vào urls trong newapi/ thêm đoạn:

from django.contrib import admin

from django.urls import path, include

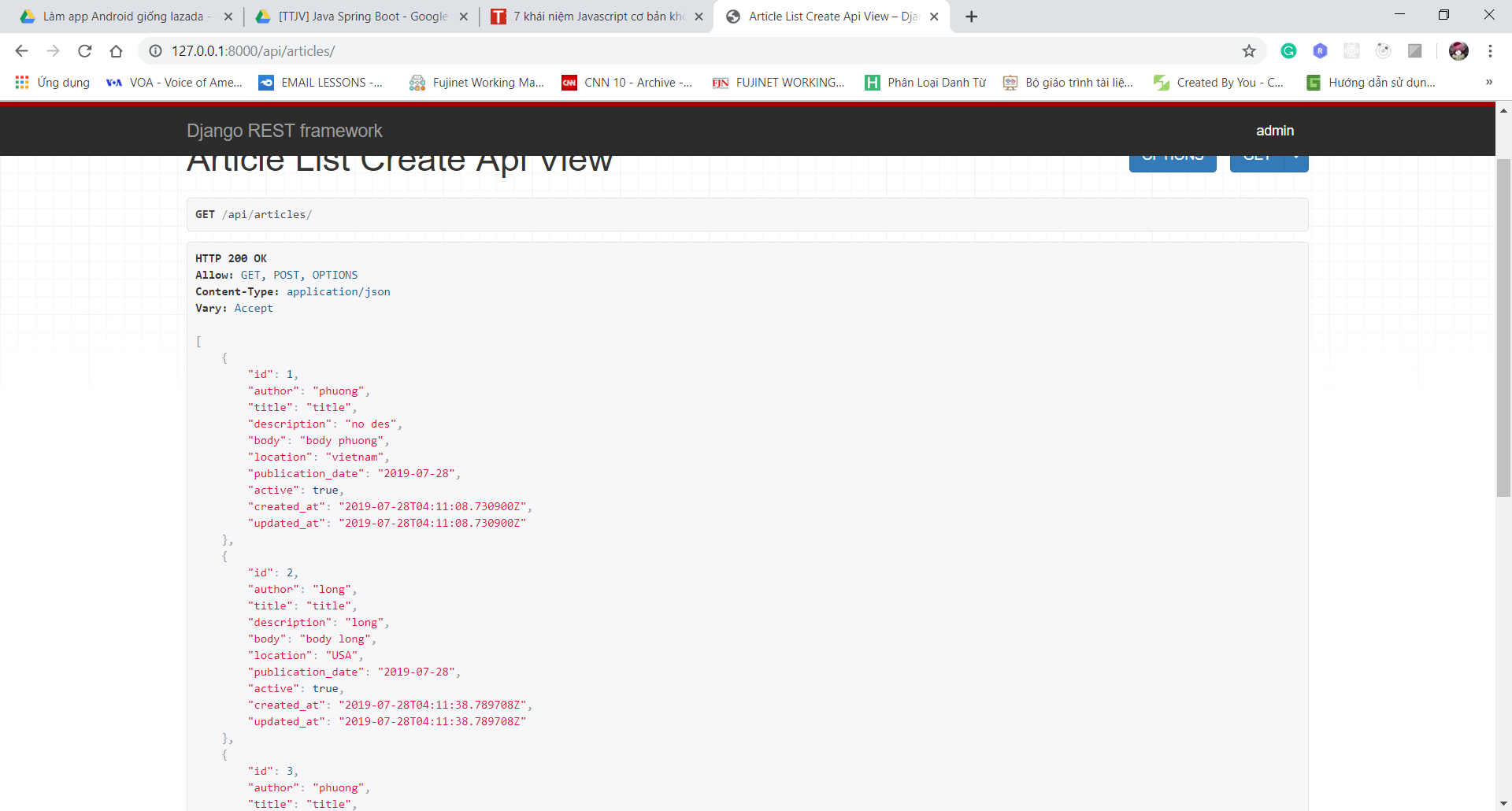
urlpatterns = [

    path('admin/', admin.site.urls),

    path("api/", include("news.api.urls"))

]

Kết quả:



6. The APIView Class

In this lesson we are going to start understanding how Class Based Views are  
used in the context of REST API development, by using DRF’s **APIView** class.  
Even though at a first glance the code we will write might look similar to the  
code we have written in the previous lecture, it is important to note that the  
**APIView** class is the class upon which all the Generic CBVs that well introduce  
later on are based, therefore it’s important to make some examples of its use

Thêm vào file **views.py** ở trong api folder:

from rest\_framework.views import APIView

from rest\_framework.response import Response

from rest\_framework.generics import get\_object\_or\_404

from news.models import Article

from news.api.serializers import ArticleSerializer

class ArticleListCreateAPIView(APIView):

    def get(self, request):

        articles = Article.objects.filter(active=True)

        serializer = ArticleSerializer(articles, many=True)

        return Response(serializer.data)

    def post(self, request):

        serializer = ArticleSerializer(data=request.data)

        if serializer.is\_valid():

            serializer.save()

            return Response(serializer.data, status=status.HTTP\_201\_CREATED)

        return Response(serializer.errors, status=status.HTTP\_400\_BAD\_REQUEST)

class ArticleDetailAPIView(APIView):

    def get\_object(self, pk):

        article = get\_object\_or\_404(Article, pk=pk)

        return article

    def get(self, request, pk):

        article = self.get\_object(pk)

        serializer = ArticleSerializer(article)

        return Response(serializer.data)

    def put(self, request, pk):

        article = self.get\_object(pk)

        serializer = ArticleSerializer(article, data=request.data)

        if serializer.is\_valid():

            serializer.save()

            return Response(serializer.data)

        return Response(serializer.errors, status=status.HTTP\_400\_BAD\_REQUEST)

    def delete(self, request, pk):

        article = self.get\_object(pk)

        article.delete()

        return Response(status=status.HTTP\_204\_NO\_CONTENT)

File urls.py

urlpatterns = [

    path("articles/",

         ArticleListCreateAPIView.as\_view(),

         name="article-list"),

    path("articles/<int:pk>/",

         ArticleDetailAPIView.as\_view(),

         name="article-detail"),]

7. Serializers Validation

So far we have seen how important it is to call the *is\_valid()* method during the data  
deserialization step before accessing the *validated\_data* dict or saving a new instance.  
If any errors occur, the **.error** property will contain a dictionary with the corresponding  
error messages:  
serializer = ContactSerializer(data={'body': 'some text', 'email': 'name#gmail.com'})  
serializer.is\_valid()  
# False  
serializer.errors  
# {'email': [u'Enter a valid e-mail address.']}

Thêm vào file serializers.py 2 hàm validate

def validate(self, data):

        """ check that description and title are different

        https://www.django-rest-framework.org/api-guide/serializers/#object-level-validation

        """

        if data["title"] == data["description"]:

            raise serializers.ValidationError("Title and Description must be different from one another!")

        return data

    def validate\_title(self, value):

        """ check that title is at least 60 chars long

        https://www.django-rest-framework.org/api-guide/serializers/#field-level-validation

        """

        if len(value) < 60:

            raise serializers.ValidationError("The title has to be at least 60 chars long!")

        return value

Khi validate value đặt tên validate\_ + tên field cần validate

8. The ModelSerializer Class

<https://www.django-rest-framework.org/api-guide/serializers/#modelserializer>

Update lại file serialize.py

class ArticleSerializer(serializers.ModelSerializer):

    time\_since\_publication = serializers.SerializerMethodField()

*# author = JournalistSerializer(read\_only=True) # step 2*

*# author = serializers.StringRelatedField() # step 1*

    class Meta:

        model = Article

        exclude = ("id",)

*# fields = "\_\_all\_\_" # we want all the fields of our model*

*# fields = ("title", "description", "body") # we want to choose a couple of fields!*

# sẽ trả về trong json

    def get\_time\_since\_publication(self, object):

        publication\_date = object.publication\_date

        now = datetime.now()

        time\_delta = timesince(publication\_date, now)

        return time\_delta

    def validate(self, data):

        """ check that description and title are different

        https://www.django-rest-framework.org/api-guide/serializers/#object-level-validation

        """

        if data["title"] == data["description"]:

            raise serializers.ValidationError("Title and Description must be different from one another!")

        return data

    def validate\_title(self, value):

        """ check that title is at least 30 chars long

        https://www.django-rest-framework.org/api-guide/serializers/#field-level-validation

        """

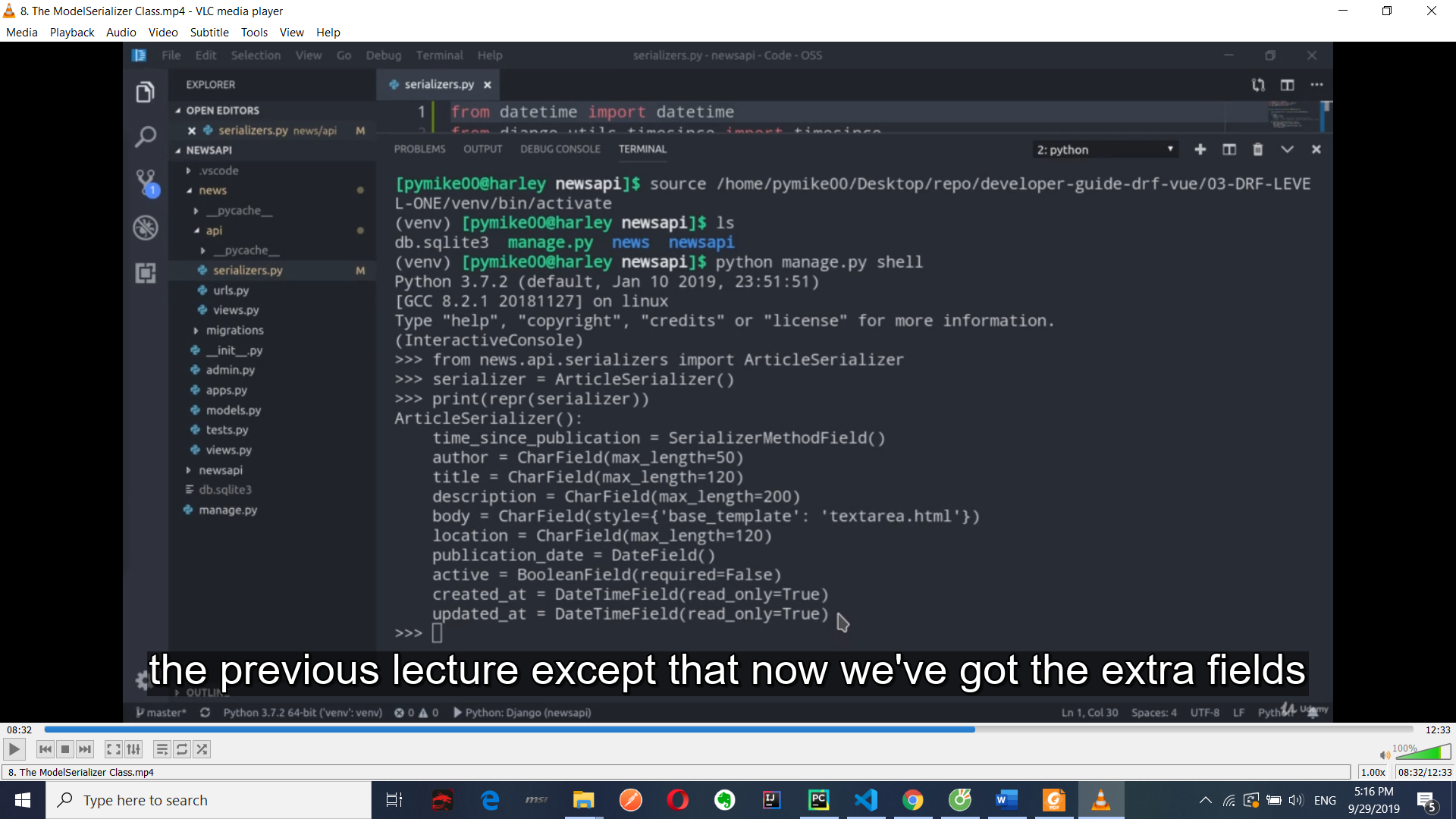
        if len(value) < 30:

            raise serializers.ValidationError("The title has to be at least 30 chars long!")

        return value

Trong class Meta có 3 cách nói vs django sẽ serialize những tt nào mà k cần viết thêm 2 hàm:

* \_\_all\_\_ là sẽ serialize tất cả các fields
* Cách 2 là chọn 1 số fields
* Exclude là ngoại trừ field này
* Bằng cách này chỉ cần khai báo class Meta sẽ thay thế tất cả đoạn code cũ



9. How to handle Nested Relationships

In this lesson we are going to learn how to define **Nested Relationships**in our Serializer Classes!  
As an example we are going to extend our News API project by creating a  
new Journalist model, that we are going to bind to our Article model with a  
ForeignKey Field, instead of the CharField that we are currently using

Tạo đối tượng Journalist

class Journalist(models.Model):

    first\_name = models.CharField(max\_length=60)

    last\_name = models.CharField(max\_length=60)

    biography = models.TextField(blank=True)

    def \_\_str\_\_(self):

        return f"{ self.first\_name } { self.last\_name }"

class Article(models.Model):

    author = models.ForeignKey(Journalist,

                               on\_delete=models.CASCADE,

                               related\_name="articles")

python manage.py makemigrations

rm db.sqlite3

python manage.py migrate

python manage.py createsuperuser

Vào file admin create an instance

from django.contrib import admin

from news.models import Article, Journalist

admin.site.register(Article)

admin.site.register(Journalist)

runserver thử

sau đó vào tranh admin thêm 2 Article, 1 Journalist

Thêm vào file serializer.py

# author = JournalistSerializer(read\_only=True) => 2 hiện JSON của obj tham chiếu

# author = serializers.StringRelatedField() => 1 hiện tên Journalist ra thông qua string method

class JournalistSerializer(serializers.ModelSerializer):

*# articles = ArticleSerializer(many=True, read\_only=True) # step 3*

    class Meta:

        model = Journalist

        fields = "\_\_all\_\_"

Thêm để post tạo article không lỗi(step 3):

# articles = ArticleSerializer(many=True, read\_only=True)

Sau đó vào file api/views.py thêm:

class JournalistListCreateAPIView(APIView):

    def get(self, request):

        journalists = Journalist.objects.all()

        serializer = JournalistSerializer(journalists,

                                          many=True,

                                          context={'request': request})

        return Response(serializer.data)

    def post(self, request):

        serializer = JournalistSerializer(data=request.data)

        if serializer.is\_valid():

            serializer.save()

            return Response(serializer.data, status=status.HTTP\_201\_CREATED)

        return Response(serializer.errors, status=status.HTTP\_400\_BAD\_REQUEST)

sau đó chỉnh lại urls được cập nhật mới

path("journalists/",

         JournalistListCreateAPIView.as\_view(),

         name="journalist-list")

Vào file seralizers.py thêm để khi get Journalist sẽ có link detail của articles nhắp vào để xem, lúc trước là sẽ hiện hết thông tin:

class JournalistSerializer(serializers.ModelSerializer):

*# step 4*

    articles = serializers.HyperlinkedRelatedField(many=True,

                                                   read\_only=True,

                                                   view\_name="article-detail") *# get name of end point in file urls.py*

Thêm context tại file views:

class JournalistListCreateAPIView(APIView):

    def get(self, request):

        journalists = Journalist.objects.all()

        serializer = JournalistSerializer(journalists,

                                          many=True,

                                          context={'request': request})

10. DRF Level One - Competency Assessment - JobBoardAPI Project Introduction

DRF end

Cài đặt venv:

pip install django

pip install djangorestframework

pip freeze > requirements.txt

django-admin startproject job\_board

cd job\_board

python manage.py startapp jobs

pwd

Khai báo settings rồi tạo model, register => makemigrations, create super user

## Django REST Framework - Level Two

1. DRF Level Two - Section Objectives

* Learn to use the GenericAPIView and Mixins classes
* Learn to use and customize Generic Class Based Views
* Group the results provided by your APIs with a pagination system
* Secure your Web APIs with a Permissions System

One of the key benefits of using the Generic Views in Django as in Django

REST Framework, is that these offer us a large amount of ready-to-use code

that is useful in most of the most common development scenarios.

CRUD operations in a model-backed API for example will be implemented in

the same way in most cases: reinventing the wheel every time would be a

waste of time!

DRF has a class called **GenericAPIView** that extends the **APIView** class already seen in previous lessons, adding to this some very useful methods and

attributes.

The GenericAPIView class is often used with **Mixins**, classes that provide  
further functionalities to our views increasing their capabilities.  
It is important to note that as we will see, the Mixin Classes provide **action  
methods** such as .list() or .create() rather than defining the handler methods,  
such as .get() or .post() directly, as we did using the APIView class.

2. EbooksAPI - Project Setup

Cài đặt venv nhớ active trước:

Pythom -m venv venv

pip install django

pip install djangorestframework

pip freeze > requirements.txt

django-admin startproject apiebook

cd job\_board

python manage.py startapp ebooks

pwd

Sau đó vào file settings.py thêm

INSTALLED\_APPS = [

    'django.contrib.admin',

    'django.contrib.auth',

    'django.contrib.contenttypes',

    'django.contrib.sessions',

    'django.contrib.messages',

    'django.contrib.staticfiles',

    'rest\_framework',

    'ebooks'

]

Sau đó tạo ở news/model class Ebook rồi register

from django.db import models

from django.core.validators import MinValueValidator,

MaxValueValidator

class Ebook(models.Model):

    title = models.CharField(max\_length=140)

    author = models.CharField(max\_length=60)

    description = models.TextField()

    publication\_date = models.DateField()

    def \_\_str\_\_(self):

        return self.title

class Review(models.Model):

    created\_at = models.DateTimeField(auto\_now\_add=True)

    updated\_at = models.DateTimeField(auto\_now=True)

    review\_author = models.CharField(max\_length=60, blank=True, null=True)

    review = models.TextField(blank=True, null=True)

    rating = models.PositiveIntegerField(validators=[MinValueValidator(1),

                                                     MaxValueValidator(5)])

    ebook = models.ForeignKey(Ebook,

                              on\_delete=models.CASCADE,

                              related\_name="reviews")

    def \_\_str\_\_(self):

        return str(self.rating)

python manage.py makemigrations

python manage.py migrate

python manage.py createsuperuser

Vào file news/admin đăng ký instance

Tạo file ebooksapi\ebooks\api\serializers.py và thêm nội dung

from rest\_framework import serializers

from ebooks.models import Ebook, Review

class ReviewSerializer(serializers.ModelSerializer):

*# review\_author = serializers.StringRelatedField(read\_only=True)*

    class Meta:

        model = Review

*# exclude = ("ebook",)*

        fields = "\_\_all\_\_"

class EbookSerializer(serializers.ModelSerializer):

    reviews = ReviewSerializer(many=True, read\_only=True)

    class Meta:

        model = Ebook

        fields = "\_\_all\_\_"

3. The GenericAPIView Class and Mixins

Vào file views.py thêm:

from rest\_framework import generics

from rest\_framework import mixins

class EbookListCreateAPIView(mixins.ListModelMixin,

                             mixins.CreateModelMixin,

                             generics.GenericAPIView):

    queryset = Ebook.objects.all()

    serializer\_class = EbookSerializer

    def get(self, request, \*args, \*\*kwargs):

        return self.list(request, \*args, \*\*kwargs)

    def post(self, request, \*args, \*\*kwargs):

        return self.create(request, \*args, \*\*kwargs)

Rồi sửa lại urls trong folder api

urlpatterns = [

    path("ebooks/",

         EbookListCreateAPIView.as\_view(),

         name="ebook-list"),

]

Trong urls.py

from django.contrib import admin

from django.urls import include, path

urlpatterns = [

    path('admin/', admin.site.urls),

    path("api/", include("ebooks.api.urls"))

]

Sau đó gọi api để post get data test

4. Concrete View Classes

Each one of the **concrete view classes** extends the **GenericAPIView** class and those **Mixins** that offer the functionalities that the class is meant to provide.  
**RetrieveUpdateAPIView** for example will extend the **GenericAPIView** class plus both **RetrieveModelMixin** and **UpdateModelMixin**.  
Because of their abstraction level they are surely the fastest to write and the  
easiest to read… but at the cost of being also *the most magical!*It’s important to keep in mind what we have seen so far in order to use them  
comfortably, knowing exactly if, when and how to customize them.

Sửa **view** rồi cập nhật lại urls

# tương đương với phần 3

# có thể vào chi tiết để xem sự tương đồng của ListCreatAPIView

class EbookListCreateAPIView(generics.ListCreateAPIView):

    queryset = Ebook.objects.all().order\_by("id")

    serializer\_class = EbookSerializer

class EbookDetailAPIView(generics.RetrieveUpdateDestroyAPIView):

    queryset = Ebook.objects.all()

    serializer\_class = EbookSerializer

class ReviewCreateAPIView(generics.CreateAPIView):

    queryset = Review.objects.all()

# xem cuốn sách nào dk review

def perform\_create(self, serializer):

        ebook\_pk = self.kwargs.get("ebook\_pk")

        ebook = get\_object\_or\_404(Ebook, pk=ebook\_pk)

        serializer.save(ebook=ebook,)

class ReviewDetailAPIView(generics.RetrieveUpdateDestroyAPIView):

    queryset = Review.objects.all()

File urls.py

urlpatterns = [

    path("ebooks/",

         EbookListCreateAPIView.as\_view(),

         name="ebook-list"),

    path("ebooks/<int:pk>/",

         EbookDetailAPIView.as\_view(),

         name="ebook-detail"),

    path("ebooks/<int:ebook\_pk>/review/",

         ReviewCreateAPIView.as\_view(),

         name="ebook-review"),

    path("reviews/<int:pk>/",

         ReviewDetailAPIView.as\_view(),

         name="review-detail")

]

Sau đó vào serializers cập nhật

class Meta:

        model = Review

        exclude = ("ebook",)

*# fields = "\_\_all\_\_"*

Khi gọi vào api list ebooks thì mỗi cuốn sách sẽ có review: ebooks/1/review => Post 1 review => reviews/1 để update

1. The Permissions System - Part One

In the context of our Ebooks API project, we will see how to:

* Grant access to our API only to authenticated users
* Grant write permissions only to authorized users

<https://www.django-rest-framework.org/api-guide/permissions/#setting-the-permission-policy>

Vào file settings.py thêm rồi run:

[Setting the permission policy](https://www.django-rest-framework.org/api-guide/permissions/#setting-the-permission-policy)

The default permission policy may be set globally, using the **DEFAULT\_PERMISSION\_CLASSES** setting. For example.

REST\_FRAMEWORK = {

'DEFAULT\_PERMISSION\_CLASSES': [

'rest\_framework.permissions.IsAuthenticated',

]

}

Khi chạy lại server api sẽ báo lỗi 403 forbidden authen, nếu muốn xem thì sửa thành IsAuthenticatedOrReadOnly.

rest\_framework.permissions.IsAuthenticatedOrReadOnly

File views.py thêm vào các class

    permission\_classes = [permission. IsAuthenticatedOrReadOnly]

Sửa lại urls trong ebookapi sau khi tạo 1 user mới ở page admin

from django.contrib import admin

from django.urls import include, path

urlpatterns = [

    path('admin/', admin.site.urls),

    path('api-auth/', include("rest\_framework.urls")),

    path("api/", include("ebooks.api.urls"))

]

Tạo file **permissions** trong folder api

from rest\_framework import permissions

class IsAdminUserOrReadOnly(permissions.IsAdminUser):

    def has\_permission(self, request, view):

        is\_admin = super().has\_permission(request, view)

        return request.method in permissions.SAFE\_METHODS or is\_admin

Thêm import class trên vào views và sử dụng

    permission\_classes = [IsAdminUserOrReadOnly]

6. The Permissions System - Part Two

Continuing the discussion we have started in the previous lesson, we are now

going to see how to secure our review instances so that they can be updated

or deleted only by the same users who have created them.

In order to do so we will first need to modify our Review model, binding it to

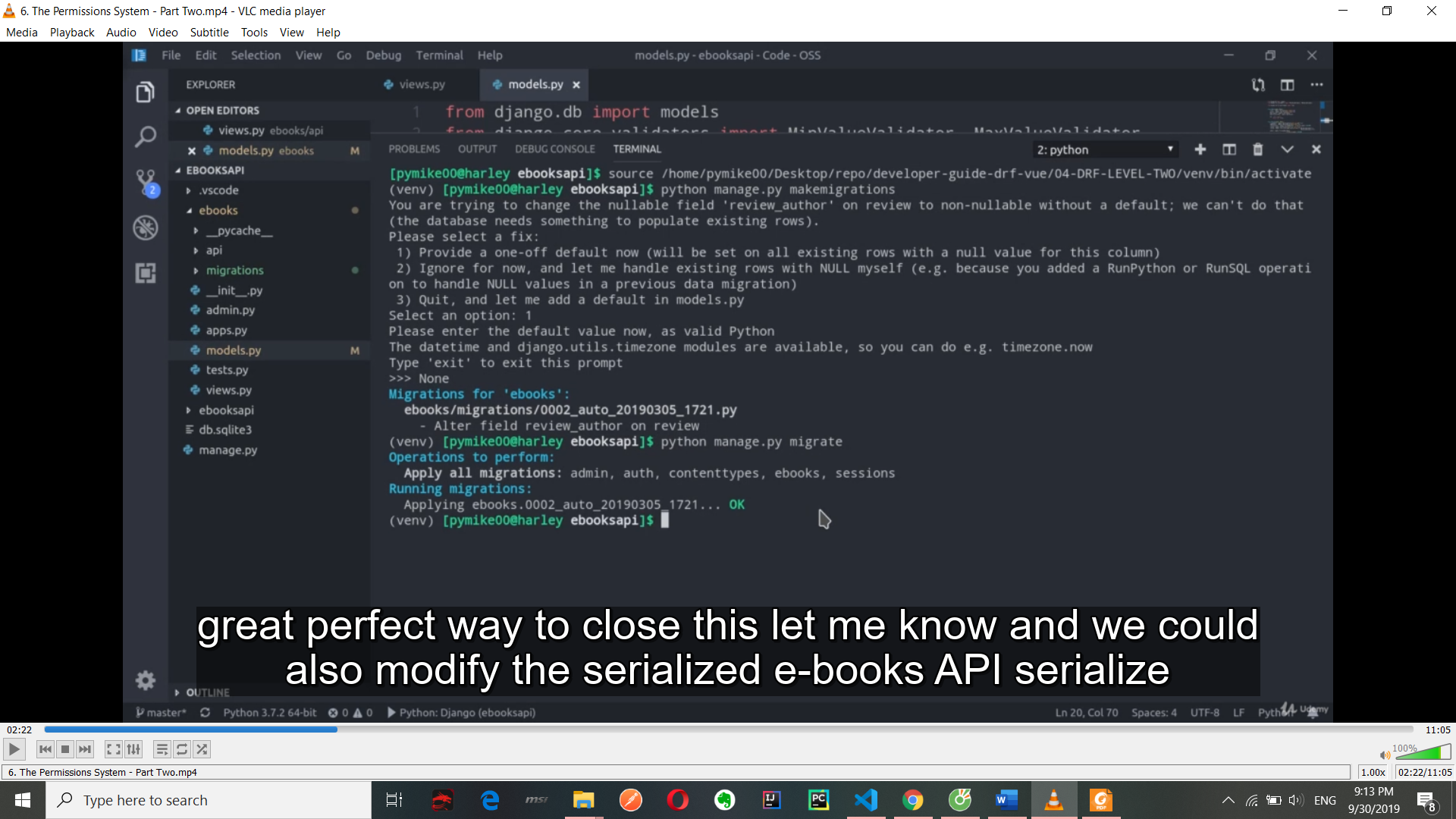
Django’s User model using a ForeignKey field

Part 2 chỉnh sửa lại model, trước hết vào page admin xóa hết review

from django.contrib.auth.models import User

Sửa:

review\_author = models.ForeignKey(User, on\_delete=models.CASCADE)



Sau đó vào file serializers.py thêm:

    review\_author = serializers.StringRelatedField(read\_only=True)

Vào file views:

class ReviewCreateAPIView(generics.CreateAPIView):

    queryset = Review.objects.all()

    serializer\_class = ReviewSerializer

    permission\_classes = [permissions.IsAuthenticatedOrReadOnly] // add new

    def perform\_create(self, serializer):

        ebook\_pk = self.kwargs.get("ebook\_pk")

        ebook = get\_object\_or\_404(Ebook, pk=ebook\_pk)

        review\_author = self.request.user // add new

// lấy ra review của user

        review\_queryset = Review.objects.filter(ebook=ebook,

                                                review\_author=review\_author)

        if review\_queryset.exists():

            raise ValidationError("You Have Already Reviewed this Ebook!")

        serializer.save(ebook=ebook, review\_author=review\_author)

Chạy ebooks/1/review để thêm review nhưng có thể thêm nhiều review => bad vì chỉ có 1 review cho 1 cuốn sách

Xong thêm lại hàm trong file permission

class IsReviewAuthorOrReadOnly(permissions.BasePermission):

    def has\_object\_permission(self, request, view, obj):

        if request.method in permissions.SAFE\_METHODS:

            return True

        return obj.review\_author == request.user

Sau đó vào file view thêm:

class ReviewDetailAPIView(generics.RetrieveUpdateDestroyAPIView):

    queryset = Review.objects.all()

    serializer\_class = ReviewSerializer

    permission\_classes = [IsReviewAuthorOrReadOnly] // add

Vào reviews/1 với user admin mới có thể put

7. Pagination in Django REST Framework

[Setting the pagination style](https://www.django-rest-framework.org/api-guide/pagination/#setting-the-pagination-style)

The pagination style may be set globally, using the DEFAULT\_PAGINATION\_CLASS and PAGE\_SIZE setting keys. For example, to use the built-in limit/offset pagination, you would do something like this:

File settings.py

# REST\_FRAMEWORK = {

# 'DEFAULT\_PAGINATION\_CLASS': 'rest\_framework.pagination.PageNumberPagination',

# 'PAGE\_SIZE': 3

# }

Cách 2: tạo file pagnation

from rest\_framework.pagination import PageNumberPagination

class SmallSetPagination(PageNumberPagination):

    page\_size = 3

rồi vào view cập nhật(có thêm order by -d là từ sau ra đầu)

    permission\_classes = [IsReviewAuthorOrReadOnly]

8. DRF Level Two - Competency Assessment - QuotesAPI Project Introduction

Bài tập: tạo quotesapi, app quotes

Tạo model, serializers, urls

## 5. Django REST Framework - Level Three

* Knowledge of the main Authentication Methods in DRF
* Set Up a Registration and Authentication System via REST
* Familiarity with Django-REST-Auth
* Knowledge of Viewset and Router Classes
* Filtering via Django REST Framework
* Being able to write automated tests with Django and DRF

Bonus: how to extend Django’s User Model with a custom Profile Model

2. UserProfilesAPI - Project Setup - Part One

We will see how to extend Django’s basic User model with a second model

that we’ ll create, allowing us to store further information about our users such as a user’s biography, city and avatar.

Each user will then be able to write status messages that will be bind to his

profile, similarly to what happens in most Social Networks

This will also give us the chance to introduce, to those who do not already know them, Django’s **Signals**!  
Signals allow certain *senders* to notify a set of *receivers* that some action has taken place elsewhere in the framework.  
In our UserProfile API project we are going to use signals to automatically create and bind a Profile’s Instance to a User Object as soon as a new one is created.

Install thêm pillow

Pythom -m venv venv

pip install django

pip install djangorestframework

pip freeze > requirements.txt

django-admin startproject profileapi

cd job\_board

python manage.py startapp profiles

Vào file settings thêm app:

'rest\_framework',

'profiles'

-------------------------

MEDIA\_URL = "/media/"

MEDIA\_ROOT = "uploads"

File urls.py

if settings.DEBUG:

    urlpatterns += static(settings.MEDIA\_URL, document\_root=settings.MEDIA\_ROOT)

Tạo class profile, ProfileStatus trong models

from django.db import models

from django.contrib.auth.models import User

# có thể vào User xem có field gì

class Profile(models.Model):

    user = models.OneToOneField(User, on\_delete=models.CASCADE)

    bio = models.CharField(max\_length=240, blank=True)

    city = models.CharField(max\_length=30, blank=True)

    avatar = models.ImageField(null=True, blank=True)

    def \_\_str\_\_(self):

        return self.user.username

class ProfileStatus(models.Model):

    user\_profile = models.ForeignKey(Profile, on\_delete=models.CASCADE)

    status\_content = models.CharField(max\_length=240)

    created\_at = models.DateTimeField(auto\_now\_add=True)

    updated\_at = models.DateTimeField(auto\_now=True)

    class Meta:

        verbose\_name\_plural = "statuses"

    def \_\_str\_\_(self):

        return str(self.user\_profile)

Sau đó vào admin.py đăng kí => migrate

3. UserProfilesAPI - Project Setup - Part Two

Ở phần tiếp ta tạo file signal.py để tạo instance

from django.contrib.auth.models import User

from django.db.models.signals import post\_save

from django.dispatch import receiver

from profiles.models import Profile

@receiver(post\_save, sender=User)

def create\_profile(sender, instance, created, \*\*kwargs):

*#     print("Created: ", created)*

    if created:

        Profile.objects.create(user=instance)

File \_\_init\_\_.py

default\_app\_config = "profiles.apps.ProfilesConfig"

apps.py

from django.apps import AppConfig

class ProfilesConfig(AppConfig):

    name = 'profiles'

# add new

    def ready(self):

        import profiles.signals

Sửa serializers

from rest\_framework import serializers

from profiles.models import Profile, ProfileStatus

class ProfileSerializer(serializers.ModelSerializer):

    user = serializers.StringRelatedField(read\_only=True)

    avatar = serializers.ImageField(read\_only=True)

    class Meta:

        model = Profile

        fields = "\_\_all\_\_"

class ProfileAvatarSerializer(serializers.ModelSerializer):

    class Meta:

        model = Profile

        fields = ("avatar",)

class ProfileStatusSerializer(serializers.ModelSerializer):

    user\_profile = serializers.StringRelatedField(read\_only=True)

    class Meta:

        model = ProfileStatus

        fields = "\_\_all\_\_"

4. Authentication in DRF

Django REST Framework provides us **different authentication  
systems** *out of the box*. In this lesson we are going to talk about the  
most important ones, evaluating their pros and cons, their most  
appropriate use cases and implementation details.  
We are also going to talk about a new authentication standard called  
**JWT**, that can be easily implemented in Django REST Framework with  
the help of some **third party packages**

It’s the **most primitive** and the **least secure** authentication system provided by Django  
REST Framework. The request/response cycle looks like this:  
● The client makes a HTTP request to the server  
● The server responds with a HTTP 401 Unauthorized response containing the WWW-Authenticate header, explaining how to authenticate (WWW-Authenticate: Basic)  
● The client sends its auth credentials in base 64 with the Authorization header.  
Authentication credentials are here unencrypted.  
● The server evaluates the access credentials and responds with the 200 or 403  
status code, therefore authorizing or denying the client’s request.

As DRF’s documentation suggests:  
*If you use* ***BasicAuthentication*** *in production you must ensure that your API is only available over https. You should also ensure that your API clients will always re-request the username and password at login, and will never store those details to persistent storage.*

Token Authentication

This is the ideal system for authenticating smartphone and desktop clients.  
The request/response flow looks like this:  
● The client sends it’s authentication credentials once  
● The server checks the credentials, and if they are valid it creates an *exclusive signed  
token* made of a string of characters that then sends back to the client as response  
● The client sends its token within the *Authorization Header* of every following request  
● The server checks the received token and if valid, allows the request to proceed.

Often times in such cases the authentication token gets saved either  
in a cookie or in the browser’s localStorage...  
...but it is very important to underline that saving the authentication  
token in **localStorage** is very **dangerous**, as it makes it vulnerable to  
XSS attacks!  
Using a httpOnly cookie on the other hand is much safer because this  
way the token can’t be accessed via JavaScript... but because of that  
you are now losing the flexibility that you would get by using  
localStorage instead!

For circumstances like ours, Django REST Framework’s official  
documentation suggests to use **Session Authentication**

DRF - Session Authentication

This authentication scheme uses Django's default session backend for  
authentication.  
Session authentication is the safest and most appropriate way for  
authenticating AJAX clients that are running in the same session context as  
your website, and uses a combination of **Sessions** and **Cookies**

The request/response cycle looks like this:  
● Users send their authentication credentials, typically using a Login Form  
● The server checks the data and if correct, it creates a corresponding *Session  
Object* that will be saved in the database, sending back to the client a *Session ID*● The Session ID gets saved in a Cookie in the browser and will be part of every future request to the server, that will check it every time  
● When the client logs out, the Session ID is destroyed by both the client and the server, and a new one will be created at the next login.

If successfully authenticated using Session Authentication, Django will  
provide us the corresponding User Object, accessible via  
**request.user**.  
For non-authenticated requests, an *AnonymousUser* instance will be  
provided instead

**Important:** once authenticated via session auth, the framework will  
require a valid CSRF token to be sent for any *unsafe* HTTP method  
request such as **PUT, PATCH, POST, DELETE.**The CSRF token is an important Cross-Site Request Forgery  
vulnerability protection, and we will see how to properly include it in  
our requests in the Final Project

5. Django REST Auth - Part One

<https://www.django-rest-framework.org/api-guide/authentication/#setting-the-authentication-scheme>

Chỉnh sửa file settings

INSTALLED\_APPS = [

    'django.contrib.admin',

    'django.contrib.auth',

    'django.contrib.contenttypes',

    'django.contrib.sessions',

    'django.contrib.messages',

    'django.contrib.staticfiles',

    'django.contrib.sites',

    'rest\_framework',

    'rest\_framework.authtoken',

}

REST\_FRAMEWORK = {

    'DEFAULT\_AUTHENTICATION\_CLASSES': (

        'rest\_framework.authentication.SessionAuthentication',

        'rest\_framework.authentication.TokenAuthentication',

    )

}

Thêm trong Installed\_app

Python manage.py migrate

Run lại server để vào page admin xem có token chưa

Pip install django-rest-auth

Pip install requests

Cd ..

Pip freeze > requirement.txt

Khai báo thêm rest-auth ở INSTALLED\_APPS

Urls.py

urlpatterns = [

    path('admin/', admin.site.urls),

    path("api/", include("profiles.api.urls")),

    path("api-auth/", include("rest\_framework.urls")), // add

    path("api/rest-auth/", include("rest\_auth.urls")), // add

    path("api/rest-auth/registration/", include("rest\_auth.registration.urls"))

]

Tạo folder clients và chỉnh sửa lại file view => urls

class ProfileViewSet(generics.ListAPIView):

    queryset = Profile.objects.all()

    serializer\_class = ProfileSerializer

    permission\_classes = [IsAuthenticated]

6. Django REST Auth - Part Two

Chạy: pip install django-allauth (Nhớ khai báo setting)

    'django.contrib.sites',

    'rest\_framework',

    'rest\_framework.authtoken',

    'allauth',

    'allauth.account',

    'allauth.socialaccount',

    'rest\_auth',

    'rest\_auth.registration',

...

SITE\_ID = 1

ACCOUNT\_EMAIL\_VERIFICATION = "none"

ACCOUNT\_EMAIL\_REQUIRED = (True)

Chạy:

Cd profilesapi

python manage.py migrate

File urls.py

    path("api/rest-

auth/registration/", include("rest\_auth.registration.urls"))

File test 02

7. Viewsets and Routers

**ViewSet** classes allow us to combine the logic for a set of related views in a

single class: a **ViewSet** could for example allow us to get a list of elements

from a queryset, but also allow us to get the details of a single instance of the

same model.

**ViewSet** work at the highest abstraction level compared to all the API views

that we have learned to use so far.

**ViewSets** are in fact another kind of Class Based View, that does not provide

any method handlers such as .get() or .post(), and instead provides action

methods such as .list() and .create(), like some other classes we already know!

We will understand how and why by analyzing their source code.

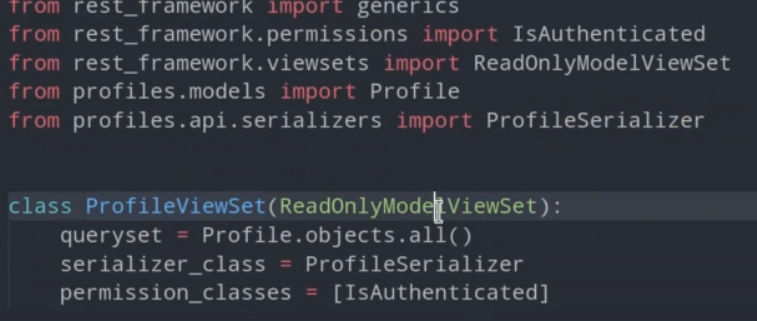
**ViewSets** are typically used in combination with the **Router** class, allowing us

to automatically get a url path configuration that is appropriate to the

different kind of actions that the ViewSet provides, following convention

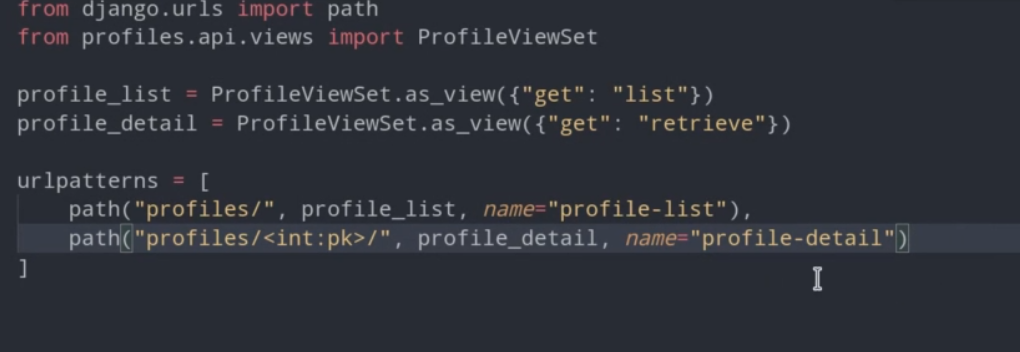
standards

File views(Xem lại)



Với cách này chỉ với 1 class => 2 end point có thể vào class trên để xem rõ có class gì

File urls



Update urls

router = DefaultRouter()

router.register(r"profiles", ProfileViewSet)

router.register(r"status", ProfileStatusViewSet, basename="status")

urlpatterns = [

    path("", include(router.urls)),

    path("avatar/", AvatarUpdateView.as\_view(), name="avatar-update")

]

Sửa lại file py và permission

class ProfileViewSet(mixins.UpdateModelMixin,

                     mixins.ListModelMixin,

                     mixins.RetrieveModelMixin,

                     viewsets.GenericViewSet):

    queryset = Profile.objects.all()

    serializer\_class = ProfileSerializer

    permission\_classes = [IsAuthenticated]

Xem lại…

8. The Filtering System

View

def get\_queryset(self):

        queryset = ProfileStatus.objects.all()

        username = self.request.query\_params.get("username", None)

        if username is not None:

            queryset = queryset.filter(user\_profile\_\_user\_\_username=username)

        return queryset

run api/status/?username=admin

Vào page admin sửa lại city

class ProfileViewSet(mixins.UpdateModelMixin,

                     mixins.ListModelMixin,

                     mixins.RetrieveModelMixin,

                     viewsets.GenericViewSet):

    queryset = Profile.objects.all()

    serializer\_class = ProfileSerializer

    permission\_classes = [IsAuthenticated, IsOwnProfileOrReadOnly]

    filter\_backends = [SearchFilter]

    search\_fields = ["city"] // add

9. Automated Testing of a REST API - Part One

File test.py

import json

from django.contrib.auth.models import User

from django.urls import reverse

from rest\_framework import status

from rest\_framework.authtoken.models import Token

from rest\_framework.test import APITestCase

from profiles.api.serializers import ProfileSerializer, ProfileStatusSerializer

from profiles.models import Profile, ProfileStatus

class RegistrationTestCase(APITestCase):

    def test\_registration(self):

        data = {"username": "testcase", "email": "test@localhost.app",

                "password1": "some\_strong\_psw", "password2": "some\_strong\_psw"}

        response = self.client.post("/api/rest-auth/registration/", data)

        self.assertEqual(response.status\_code, status.HTTP\_201\_CREATED)

class ProfileViewSetTestCase(APITestCase):

    list\_url = reverse("profile-list")

    def setUp(self):

        self.user = User.objects.create\_user(username="davinci",

                                             password="some-very-strong-psw")

        self.token = Token.objects.create(user=self.user)

        self.api\_authentication()

    def api\_authentication(self):

        self.client.credentials(HTTP\_AUTHORIZATION="Token " + self.token.key)

    def test\_profile\_list\_authenticated(self):

        response = self.client.get(self.list\_url)

        self.assertEqual(response.status\_code, status.HTTP\_200\_OK)

    def test\_profile\_list\_un\_authenticated(self):

        self.client.force\_authenticate(user=None)

        response = self.client.get(self.list\_url)

        self.assertEqual(response.status\_code, status.HTTP\_403\_FORBIDDEN)

    def test\_profile\_detail\_retrieve(self):

        response = self.client.get(reverse("profile-detail", kwargs={"pk": 1}))

        self.assertEqual(response.status\_code, status.HTTP\_200\_OK)

        self.assertEqual(response.data["user"], "davinci")

    def test\_profile\_update\_by\_owner(self):

        response = self.client.put(reverse("profile-detail", kwargs={"pk": 1}),

                                   {"city": "Anchiano", "bio": "Renaissance Genius"})

        self.assertEqual(response.status\_code, status.HTTP\_200\_OK)

        self.assertEqual(json.loads(response.content),

                         {"id": 1, "user": "davinci", "bio": "Renaissance Genius",

                          "city": "Anchiano", "avatar": None})

    def test\_profile\_update\_by\_random\_user(self):

        random\_user = User.objects.create\_user(username="random",

                                               password="psw123123123")

        self.client.force\_authenticate(user=random\_user)

        response = self.client.put(reverse("profile-detail", kwargs={"pk": 1}),

                                   {"bio": "hacked!!!"})

        self.assertEqual(response.status\_code, status.HTTP\_403\_FORBIDDEN)

## 6. Introduction to Vue JS

**Vue JS** is an open source JavaScript framework for building user

interfaces, reactive components and Single Page Applications (SPA).

It is called a progressive framework because of its specific design, that

makes it easy to create both simple and complicated projects

3. Your First Vue Instance

Vào chrome cài extension **vue.js dev tool**

Có Thể vào console gõ app.message để in ra message trong el hay update value

<h2>{{ message.split('').reverse().join('') }}</h2>

*<!-- <img v-bind:src="imgSrc">*

*<a v-bind:href="link">Official Website</a> -->*

        <img :src="imgSrc">

        <a :href="link">Official Website</a>

Vue JS - Events and Methods  
v-on

var app = new Vue({

    el: '#app',

    data: {

        lesson: "Events and Methods",

        counter: 0

    },

    methods: {

        incrementCounter() {

            this.counter += 1;

            console.log(this.counter);

            if (this.counter === 10) {

                alert("Counter is at 10!");

            }

        },

        overTheBox() {

            console.log("Over The Green Box!");

        }

    }

})

File html: có thể sử dụng @ thay cho v-on

<button v-on:click="incrementCounter">Increment ++</button>

        <br><br>

        <div

            class="box"

            @mouseover="overTheBox">

        </div>

5. Conditional Rendering

v-if, v-else, v-else-if, v-show

 <div id="app">

        <h1>Product: {{ product }}</h1>

        <h2 v-if="quantity >= 20">Available</h2>

        <h2 v-else-if="quantity > 0 && quantity < 20">Limited Availability</h2>

        <h2 v-else>Sold Out</h2>

        <br>

        <h3 v-show="sale">On Sale!!!</h3>

    </div>

6. Class and Style Binding

<div id="app">

        <div class="row justify-content-center mt-3">

            <div v-bind:class="{ circle: flag, square: !flag }"

                 v-bind:style="styleObject"></div>

        </div>

        <div class="row justify-content-center mt-3">

            <button class="btn btn-success" @click="changeShape">Change Shape</button>

        </div>

    </div>

Main.js

var app = new Vue({

    el: "#app",

    data: {

        flag: true,

        styleObject: {

            backgroundColor: 'green',

            border: '5px solid orange'

        }

    },

    methods: {

        changeShape() {

            this.flag = !this.flag;

        }

    }

})

7. List Rendering with v-for

<div class="card"

             v-for="user in users"

             :key="user.id">

            <div class="card-body">

                <p><strong>Name:</strong> {{ user.name }}</p>

             <p><strong>Profession:</strong> {{ user.profession }}</p>

            </div>

        </div>

8. Computed Properties

Không cần dấu () như method

9. Forms and User Input

v-model

            <input type="text" v-model="color">

<form @submit.prevent="onSubmit">

                    <div class="form-group">

                        <label for="commentText">Publish a comment!</label>

                        <textarea class="form-control"

                                  v-model="comment"

                                  id="commentText"

                                  cols="80"

                                  rows="5">

                        </textarea>

                    </div>

                    <button class="btn btn-sm btn-primary"

                            type="submit"

                            >Publish

                    </button>

                </form>

10. Components and Props

Vue.component("comment", {

    props: {

        comment: {

            type: Object,

            required: true

        }

    },

    template: `

        <div>

            <div class="card-body">

                <p>{{ comment.username }}</p>

                <p>{{ comment.content }}</p>

            </div>

            <hr>

        </div>

    `

})

Html

<div id="app">

        <comment

            v-for="(comment, index) in comments"

            :comment="comment"

            :key="index"

        ></comment>

    </div>

11. How To Use $emit

Vue.component("comment-list", {

    props: {

        comments: {

            type: Array,

            required: true

        }

    },

    data: function() { // dùng vs mỗi component có thể tái SD

        return {

            new\_comment: null,

            comment\_author: null,

            error: null

        }

    },

    methods: {

        submitComment() {

            if (this.new\_comment && this.comment\_author) {

                this.$emit('submit-comment', { username: this.comment\_author,

                    content: this.new\_comment });

                this.new\_comment = null;

                this.comment\_author = null;

                if (this.error) {

                    this.error = null;

                }

            } else {

                this.error = "please fill out both fields!"

            }

        }

    },

    template: `

        <div class="mt-2">

            <div class="container">

                <single-comment

                    v-for="(comment, index) in comments"

                    :comment="comment"

                    :key="index"

                ></single-comment>

                <hr>

                <h3>{{ error }}</h3>

                <form @submit.prevent="submitComment" class="mb-3">

                    <div class="form-group">

                        <label for="commentAuthor">Your Username</label>

                        <input class="form-control"

                            id="commentAuthor"

                            type="text"

                            v-model="comment\_author">

                    </div>

                    <div class="form-group">

                        <label for="commentText">Add a comment</label>

                        <textarea class="form-control"

                                id="commentText"

                                rows="3"

                                cols="40"

                                v-model="new\_comment">

                        </textarea>

                    </div>

                    <button class="btn btn-sm btn-primary"

                            type="submit"

                            >Publish

                    </button>

                </form>

            </div>

        </div>

    `

})