Nói một cách ngắn gọn, routing (định tuyến) trong web application là quá trình tuyển chọn thành phần nào sẽ xử lý yêu cầu của người dùng, thông thường dựa vào URL. Vì URL căn bản là chuỗi (string) nên kỹ thuật routing chủ yếu là so sánh chuỗi, phân tích chuỗi rồi đưa ra quyết định với if-else hoặc switch-case.

Routing chủ yếu dựa vào HTTP method và ‘path’ của url để quyết định function nào hoặc method của class nào sẽ được gọi để xử lí request được đưa ra

URL là viết tắt của Uniform Resource Locator, dịch sang tiếng Việt là định vị tài nguyên thống nhất. Cụ thể, URL là địa chỉ của một tài nguyên duy nhất trên Web. Mỗi URL hợp lệ sẽ trỏ đến một tài nguyên duy nhất, tài nguyên đó có thể là trang HTML, tài liệu CSS, hình ảnh, video, file PDF... Trong một số trường hợp ngoại lệ, URL có thể trỏ đến những tài nguyên không còn tồn tại hoặc đã bị di chuyển sang địa chỉ khác (moved).

Route tức là tuyến đường để dẫn web tới một domain nhất định (phần đuôi của web) và sử dụng phương thức http get/post…

Mỗi route có thể có một hay nhiều ‘handler function’(fuction sử dụng nó??) và fuction trỏ đến route này được thực hiện khi mà người nhập nhập đúng route

Graphical user interface

Description automatically generated with medium confidence

Path là phần đuôi, hay route mà người nhập cần match để handler hay function mà các bạn callback được thực thi

Trong code thì ở views chúng ta có 1 def ví dụ Text

Description automatically generated

Thì ở trong url.py

Graphical user interface, text

Description automatically generated with medium confidence

Theo thứ tự là ‘ ‘= path cần match, views.taskList là fuction đc call từ views và respond cho request

https://quantrimang.com/cong-nghe/url-la-gi-158090

Module thực ra là các mục code để hoàn thiện sản phẩm ví dụ

<https://numo.vn/kien-thuc-web/module-website.html>

## HTTP Request methods[[sửa](https://vi.wikipedia.org/w/index.php?title=Hypertext_Transfer_Protocol&veaction=edit&section=4) | [sửa mã nguồn](https://vi.wikipedia.org/w/index.php?title=Hypertext_Transfer_Protocol&action=edit&section=4)]

HTTP Request Method chỉ phương thức để được thực hiện trên nguồn được nhận diện bởi Request-URI đã cung cấp

**GET**

GET được sử dụng để lấy lại thông tin từ Server đã cung cấp bởi sử dụng một URI đã cung cấp. Các yêu cầu sử dụng GET chỉ nhận dữ liệu và không có ảnh hưởng gì tới dữ liệu.

**HEAD**

Tương tự như GET, nhưng nó truyền tải dòng trạng thái và khu vực Header.

**POST**

Một yêu cầu POST được sử dụng để gửi dữ liệu tới Server, ví dụ, thông tin khách hàng, file tải lên, …, bởi sử dụng các mẫu HTML.

**PUT**

Thay đổi tất cả các đại diện hiện tại của nguồn mục tiêu với nội dung được tải lên.

**DELETE**

Gỡ bỏ tất cả các đại diện hiện tại của nguồn mục tiêu bởi URI.

**CONNECT**

Thiết lập một tunnel tới Server được xác định bởi URI đã cung cấp.

**OPTIONS**

Miêu tả các chức năng giao tiếp cho nguồn mục tiêu.

**TRACE**

Trình bày một vòng lặp kiểm tra thông báo song song với path tới nguồn mục tiêu.

## HTTP Response[[sửa](https://vi.wikipedia.org/w/index.php?title=Hypertext_Transfer_Protocol&veaction=edit&section=5) | [sửa mã nguồn](https://vi.wikipedia.org/w/index.php?title=Hypertext_Transfer_Protocol&action=edit&section=5)]

Khi nhận và phiên dịch một HTTP Request, Server sẽ gửi tín hiệu phản hồi là một HTTP Response bao gồm các thành phần sau:

* Một dòng trạng thái (Status-Line)
* Không hoặc nhiều hơn các trường Header (General|Response|Entity) được theo sau CRLF
* Một dòng trống chỉ dòng kết thúc của các trường Header
* Một phần thân thông báo tùy ý

django-crispy-forms provides you with a |crispy filter and {% crispy %} tag that will let you control the rendering behavior of your [Django](https://djangoproject.com/) forms in a very elegant and DRY way. Have full control without writing custom form templates. All this without breaking the standard way of doing things in [Django](https://djangoproject.com/), so it plays nice with any other form application.

<https://dennisivy.com/django-class-based-views>

understand function based view before class method

App là các component của 1 website và Django giúp chúng ta chia nhỏ các phần để dễ làm việc

Hơn nữa đây là các cấu trúc cơ bản của một website

Truowcs hết phải connect views và url files

Vaf để làm vậy thì phải connect setting.py với file ‘base’hay base.app.baseconfig để trỏ vào function đó

Tạo thêm urls.py trong base để tạo ‘path’ fuction và tác dụng thì như trên

\*\*Django đã cung cấp đủ những công cụ cần thiết

Từ views trong base import http để xử lí web, httpresponse là 1 hàm trong Django, sau đó add view vào url trong base để trỏ vào response

Tiếp đó chugns ta cần để cho project (pj) nhận diện được base url (hay url của app) thì cần function ‘include’

Nếu các bạn đã thành công thì tức là chugns ta đã xong bước 1

Bước 2:

Vào models.py trong application để tạo database

Need classes

Need to migrate database

Data Migration (Chuyển đổi dữ liệu) là quá trình di chuyển dữ liệu giữa các hệ thống lưu trữ dữ liệu, các định dạng dữ liệu hay giữa các hệ thống máy tính.



An example of database table

from django.contrib.auth.models import User

cho luôn user info, username email, password

1 user có thể có nhiều attributes nên nó là 1 to many relationship

Graphical user interface, text, application, email

Description automatically generated

MVC model <https://www.youtube.com/watch?v=N8GhaR7K3tI&ab_channel=F8Official>

<https://docs.djangoproject.com/en/4.1/ref/models/fields/#django.db.models.ForeignKey>

Graphical user interface, text, application, email

Description automatically generated

On\_delete tức là nếu delete user thì sao đó

(Users, on\_delete=models.CASCADE, null=True, blank = True)

 If a field has **blank=True**, form validation will allow entry of an empty value

If **True**, Django will store empty values as **NULL** in the database.

Neeus user bị xóa thì giá trị của nó có thể bằng null

nếu blank = true thì giá trị nhập vào có thể là trống

thường để blank khi mà thử hoặc bắt đầu applications hay ‘base’ để dễ hơn trong việc thêm hay xóa items

***class*CharField(*max\_length=None*, *\*\*options*)**[**¶**](https://docs.djangoproject.com/en/4.1/ref/models/fields/#django.db.models.CharField)

là một string field, để giới hạn độ dài của string đó

***class*TextField(*\*\*options*)**[**¶**](https://docs.djangoproject.com/en/4.1/ref/models/fields/#django.db.models.TextField)

dùng text field để có nhiều lựa chọn hơn cho độ dài

***class*BooleanField(*\*\*options*)**[**¶**](https://docs.djangoproject.com/en/4.1/ref/models/fields/#django.db.models.BooleanField)

a true/false field

QUERY PARAMETERS

Ví dụ local xxyyzz ? q = xxuuzzs

Sau ?q= hay ?’key’=’value’

chính là query para, truy vấn cái đ gì đấy, nó có tác dụng làm hoa tiêu trỏ tới một object với key là ‘q’, value là ‘xxuuzzs’

python manage.py migrate

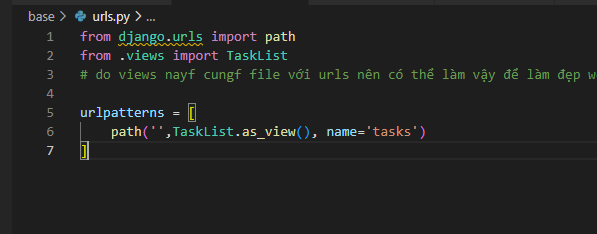
để chuyển toàn bộ migrations vào database

và từ giờ nếu muốn thay đổi hay thêm models thì phải chạy lại để nó update vào db

vaof trang admin của nó để điều chỉnh database dễ hơn mà k cần function

Bước 3: bắt đầu làm view

Sửa url trong base



from django.shortcuts import render

from .models import Product

def productsList(request):

products = Product.objects.all()

if request.method == 'POST':

Product.object.create()

context = {'products':products}

return render(request, 'base/product\_list.html', context)

code truowcs khi sử dụng as\_view() method

blah blah khos vcl

b4: tạo template

tạo một cái task-list qua template html

Context\_object\_name : override the default queryset name of “object\_list”, giúp user dùng những cái tên dễ sử dụng hơn beside just “object list”

Sau khi import các views

from django.views.generic.list import ListView

from django.views.generic.detail import DetailView

thì phải thêm path vào url

Graphical user interface, text, application, email

Description automatically generated

Path phair cos tên task<int:pk> … thì mới tìm được

Tức ;à task/1(2,3,4,)… task/ interger

Để có một cái view chi tiết cho từng item

Sau đó code trên task\_detail.html để thể hiện, có thể ddooir 1,2,3,4 thành tên item hay query gì đó sau

<td><a href="% url 'task' task.id %">View</a></td>

Tạo một cái link

Text

Description automatically generated

Graphical user interface, application

Description automatically generated with medium confidence

Text

Description automatically generated

* Use the <img> element (inside <a>) to use an image as a link

<https://www.w3schools.com/tags/att_a_href.asp>

ví dụ

<a href="{% url 'task-create' %}">Add Task</a>

path('task-create/',TaskCreate.as\_view(), name='task-create'),

class TaskCreate(CreateView):

Là tạo một cái link có tên rút gọn là ‘add task’, có tác dụng trỏ tới ‘task-create’ trong url, url path trỏ tới class ‘TaskCreate’ trong view

<form method="POST" action="" >

    <input type="submit" value ="submit">

</form>

Khi mà xử lí 1 cái form trong template thì sẽ được

Graphical user interface, application

Description automatically generated

Với dạng của nó là POST, gửi giá trị tới db

<form method="POST" action="" >

    {{form.as\_p}}

    <input type="submit" value ="submit">

</form>

Còn form.as\_p chỉ có tác dụng làm đẹp

<https://www.geeksforgeeks.org/form-as_p-render-django-forms-as-paragraph/>

The double curly brackets are not HTML but scripting code.

The term inside, interest, is a placeholder, sort of like the name

and address in a form letter. The string {{interest}}

will be replaced when the HTML template is converted into straight HTML

that is sent over the network to the user.

example:

<p> The interest earned is: {{interest}} </p>

sau khi nhập gì đó thì nó bị lỗi

Graphical user interface

Description automatically generated with low confidence

<https://topdev.vn/blog/csrf-la-gi/>

và fix bằng

{% csrf\_token %}

Sau khi hoàn thiện nó sẽ trở về task\_list đúng như trên

Bước 5: update item, delete item

Delete View

|  |
| --- |
| """ |
|  | View for deleting an object retrieved with self.get\_object(), with a |
|  | response rendered by a template. |
|  | """ |
|  |  |
|  | template\_name\_suffix = "\_confirm\_delete" |

Template suffx laf phần đuôi của template để ứng với method

Sau 5 bước thì chúng ta đã hoàn thiện đủ nhập, sửa, xóa các thứ và tiếp tục tiến đến đki/đnhap

Bước 6:

Format của 1 web thì nên có 1 dòng go back trên cùng

<a href="{% url 'tasks' %}">Go Back</a>

Trong Django thì built-in nhiều nên cần code specific hơn

A screenshot of a computer

Description automatically generated with medium confidence

<HR> là dòng gạch ngang

Shape

Description automatically generated with medium confidence

Để chia cách blah blah

Tiếp theo, để giống như thường htif chúng ta cần đăng nhập rồi mới có thể truy cập thông tin

Nên

Thêm mix-in

Để khi ấn vào web, nó tự động dẫn ta tới login

Qua trình bao gồm add mix-in vào tất cả những chỗ cần quyền thì mới truy cập được

from django.contrib.auth.mixins import LoginRequiredMixin

class TaskList(LoginRequiredMixin, ListView):

LOGIN\_URL = 'login'

ở views và setting

và để ngăn cách các user khỏi việc trùng db thì

CONTEXT OBJECT là những OBJECT mà nó được cố định, không thay đổi giá trị dù truyền tới bất cứ đâu trong hàm con của nó.

<https://www.youtube.com/watch?v=TENin-HxvRg&ab_channel=F8Official>

listView thực ra là một dictionary mà display toàn bộ list và các chi tiết của object mà chúng ta điền vào. Và giờ chúng ta phải xử lí data đó

trong ListView ban đầu

Graphical user interface, text

Description automatically generated

Chúng ta cho chạy toàn bộ các task trong db không quan tâm người dùng là ai

Và những context data đó, những dữ liệu theo ngữ cảnh, hay còn gọi là dẽ liệu cung cấp bối cảnh cho 1 user, cần được modify(biến đổi, xử lí)

\*\*KWARGS

Graphical user interface, application, Teams

Description automatically generated

Làm chi tiết hơn về user, lọc cho chỉ có user1 thì mới dùng đc data1, user2 dùng data2

Bước 7: register

Tạo 1 template, tạo 1 hàm trong views, link 1 path từ views sang url, done

Bước 8 : search

Trước hết là phải có 1 form để search, chính là cái ô để search

Vào task\_lish teamplate

Làm ngay trên bảng item

<form method = "GET">

    <input type = 'text' name = 'search-area' value = "{{ search\_input }}">

    <input type = "submit" value = "Search">

</form>

Mình phải nhập vào giá trị nên submit, giá trị đc tìm là text, value là cái ô ghi ‘search’ còn submit là cái ô trống

Còn cái dòng search-area để phục vụ logic tìm kiếm

A picture containing application

Description automatically generated

Đã có form rồi thì phải add logic

Vì nó ở cùng item các thứ nên làm luôn trong Task\_list

search\_input = self.request.GET.get('search-area')or ''

search input vào phải nhận được từ phương thức “GET” từ user, với giá trị lấy trong “SEARCH-AREA”

và nó có thể là empty string nên là có ‘’

if search\_input:

            context['tasks'] = context['tasks'].filter(title\_\_icontains=search\_input)

nếu chỉ có như này thì sau khi search, từ đc tìm nó sẽ biến mất

Graphical user interface, application

Description automatically generated

Nên chúng ta cần thêm vào search input

ở trong views

context['search\_input'] = search\_input

và ở trong template

<input type = 'text' name = 'search-area' value = "{{ search\_input }}">

Vì mỗi lần search nó sẽ refresh page nên làm vậy cho nó đỡ mất từ khi tìm

Và cái ‘value’ ở đây giúp cho input k bị mất đi

Graphical user interface, text, application

Description automatically generated

Còn bạn muốn tìm hiểu thêm về cái ‘name’ thì

<https://www.w3schools.com/tags/att_name.asp>

<https://www.youtube.com/watch?v=xiQXJtOzGZI&ab_channel=LetsCreateWithHTMLandCSS>

Bước 9: hoàn thiện, tạo 1 main để code, dùng

{% extends 'base/main.html' %}

{% block content %}

{% endblock content %}

Và nhét hết template vào cái block này để code trên main có thể thay đổi đc các màu mè trên trang còn lại

Font chữ thì lên

<https://fonts.google.com/>

rồi copy zô rồi chạy

You have <i>{{count}}</i> incomplete task{{ count|pluralize:"s" }}</h3>

Một dòng để đếm xem còn bao nhiêu task chưa làm

Count/plural … để nó xem là số nhiều thì thêm s, ko thì để là task

CSS quyeets định kiểu dáng của website mà chính là thứ chúng ta đang làm

Nói chung là hphg bảo thầy bảo là k cần quan tâm template lắm nên tự túc nhé hihi 😊