**ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH**

**KHOA CÔNG NGHỆ PHẦN MỀM**

**------------------------**



BÁO CÁO ĐỒ ÁN

**ĐỀ TÀI: TÌM HIỂU VỀ DJANGO & XÂY DỰNG ỨNG DỤNG LIÊN QUAN**

MÔN HỌC: ĐỒ ÁN CHUYÊN NGHÀNH – SE112.J11

GVHD : Hoàng Văn Hà

Sinh viên thực hiện : Trần Thiện Hòa

MSSV : 15520243

# LỜI MỞ ĐẦU

Trong thời gian thực hiện đồ án với sự giúp đỡ tạo điều kiện của trường Đại Học Công Nghệ Thông Tin – Đại Học Quốc Gia Thành Phố Hồ Chí Minh, sự góp ý của các bạn và đặc biệt là sự hướng dẫn trực tiếp, chỉ bảo tận tình của giảng viên hướng dẫn Hoàng Văn Hà em đã hoàn thành đề tài cùng với bản báo cáo đúng thời gian quy định. Với khả năng và thời gian có hạn nên không tránh khỏi những thiếu sót, em rất mong nhận được sự quan tâm, giúp đỡ, tạo điều kiện của thầy cô giáo để em hoàn thiện hơn đề tài nghiên cứu trong thời gian tới.

Một lần nữa em xin chân thành cảm ơn tất cả các thầy, cô giáo trong trường Đại Học Công Nghệ Thông Tin – Đại Học Quốc Gia Thành Phố Hồ Chí Minh cũng như các thầy cô giáo của các trường, các tổ chức liên kết đã dạy dỗ, chỉ bảo chúng em trong suốt thời gian học. Đặc biệt em xin gửi lời cảm ơn sâu sắc tới thầy giáo Hoàng Văn Hà đã hướng dẫn em suốt quá trình làm đồ án.

Em xin chân thành cảm ơn!

Mục lục

[I. Giới thiệu chung. 3](#_Toc373192269)

[1. Python 3](#_Toc373192270)

[2. Django 3](#_Toc373192271)

[3. Cài đặt 3](#_Toc373192272)

[4. Cấu hình Database 5](#_Toc373192273)

[II. Các đặc trưng chính. 5](#_Toc373192274)

[1. Ánh xạ đối tượng - quan hệ (Object-relational mapper) 5](#_Toc373192275)

[a. Projects và .apps 5](#_Toc373192276)

[b. Model 6](#_Toc373192277)

[c. Class 6](#_Toc373192278)

[d. Giao diện lập trình ứng dụng truy cập cơ sở dữ liệu (Database access API) 6](#_Toc373192279)

[2. Giao diện quản trị (Admin Interface) 7](#_Toc373192280)

[a. Sử dụng giao diện quản trị 7](#_Toc373192281)

[b. Tùy biến lớp ModelAdmin 8](#_Toc373192282)

[c. Quản lý quyền (Permission) 8](#_Toc373192283)

[3. URL linh động 9](#_Toc373192284)

[a. Sử dụng URL 9](#_Toc373192285)

[b. Quá trình xử lý yêu cầu 10](#_Toc373192286)

[c. URL động 10](#_Toc373192287)

[4. Hệ thống mẫu (Template system). 11](#_Toc373192288)

[a. Thành phần 11](#_Toc373192289)

[b. Đặc điểm 12](#_Toc373192290)

[5. Hệ thống Cache (Cache system) 15](#_Toc373192291)

[a. Các kiểu cache 15](#_Toc373192292)

[b. Các tham số của CACHE 18](#_Toc373192293)

[c. Các cấp độ cache 18](#_Toc373192294)

[6. Quốc tế hóa (Internationalization). 21](#_Toc373192295)

[III. Tổng kết 22](#_Toc373192296)

[1. Ưu điểm: 22](#_Toc373192297)

[2. Nhược điểm: 22](#_Toc373192298)

[3. Phân công công việc 22](#_Toc373192299)

[IV. Tài liệu tham khảo 22](#_Toc373192300)

**Nhận xét của giảng viên**

……………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………

# Giới thiệu chung.

## Python

Cùng với sự phát triển của nhiều ngôn ngữ lập trình web như PHP, Ruby, Scheme thì Python là một cái tên đáng chú ý. Python là một ngôn ngữ có hình thái rất sáng sủa, cấu trúc rõ ràng, thuận tiện cho người mới bắt đầu học lập trình. Cấu trúc của Python còn cho phép người sử dụng viết mã lệnh với số lần gõ phím tối thiểu nhất. Python là một ngôn ngữ lập trình đơn giản nhưng lại rất hiệu quả. Bên cạnh đó, Python là một ngôn ngữ có tính hướng đối tượng cao.

Ưu điểm:

* Cú pháp đơn giản, dễ đọc.
* Vừa hướng thủ tục (procedural-oriented), vừa hướng đối tượng (object-oriented).
* Hỗ trợ module và hỗ trợ gói (package).
* Xử lý lỗi bằng ngoại lệ (Exception).
* Kiểu dữ liệu động ở mức cao
* Có các bộ thư viện chuẩn và các module ngoài, đáp ứng tất cả các nhu cầu lập trình.
* Có khả năng tương tác với các module khác viết trên C/C++ (Hoặc Java cho Jython, hoặc .Net cho IronPython).
* Có thể nhúng vào ứng dụng như một giao tiếp kịch bản (scripting interface).
* Chạy trên đa nền tảng: MacOSX, Windows, Linux.

## Django

* Django là một web framework khá nổi tiếng được viết hoàn toàn bằng ngôn ngữ Python. Nó không phải là một micro-framework như Flask, mà là một framework với đầy đủ các thư viện, module hỗ trợ các web-developer.
* Django , ban đầu là một framework cho website của một tờ báo ở Kansas và được đưa vào sử dụng rộng rãi từ năm 2005 đến nay.
* Django là một Web framework hướng đối tượng dựa trên nền python, và theo mô hình Model – Controller – View.
* Django cho phép xây dựng những trang web với hiệu suất cao một cách nhanh chóng. Đó là lí do tại sao Django đang dần trở nên phổ biến nhằm triển khai các dự án quy mô lớn nhanh và hiệu quả.

## Cài đặt

* Yêu cầu Python 2.6 trở lên.
* Download Django từ địa chỉ <https://www.djangoproject.com/download/>
* Giải nén tập tin down về, mở cmd tới đường dẫn tập tin đã giải nén và chạy lệnh:

python setup.py install

* Bắt đầu một project bằng lệnh:

python django-admin.py startproject <site name>

Thư mục tạo ra sẽ có cấu trúc:

site name

manage.py

site name

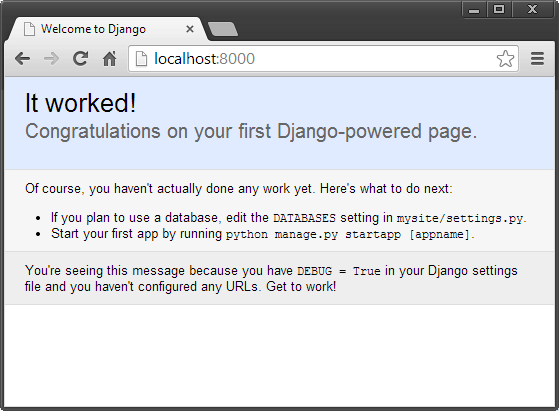
urls.py

settings.py

wsgi.py

\_\_init\_\_.py

* Khởi động server: python manage.py runserver [ip:][port]



Hình 1: Hình ảnh trang chủ khi khởi động server.

## Cấu hình Database

* Django hỗ trợ chính thức các hệ quản trị cơ sở dữ liệu: PostgreSQL, MySQL, Oracle và SQLite.
* Ngoài ra còn có các bản cung cấp của bên thứ ba để dùng với các hệ quản trị cơ sở dữ liệu khác như: Sybase SQL Anywhere, IBM DB2, Microsoft SQL Server 2005, Firebird hay ODBC,…
* Để cài đặt hệ quản trị cơ sở dữ liệu đi kèm, ta có thể chỉnh sửa trong file settings.py tại trường DATABASES.

DATABASES = {

'default': {

'ENGINE': 'django.db.backends. ', *# 'postgresql\_psycopg2', 'mysql',*

*# 'sqlite3' or 'oracle‘ …*

'NAME': '',

'USER': '',

'PASSWORD': '',

'HOST': '',

'PORT': '', }

}

# Các đặc trưng chính.

## Ánh xạ đối tượng - quan hệ (Object-relational mapper)

### Projects và .apps

* Một app là một ứng dụng Web để thực hiện về một việc nào đó nào đó. Ví dụ như ứng dụng lấy ý kiến người dùng trên web hay ứng dụng viết và hiển thị blog, …
* Một project là bao gồm của cả cấu hình và các app cho một trang web cụ thể.
* Một project có thể gồm nhiều app. Tuy nhiên một app cũng có thể thuộc nhiều project. Để sử dụng một app nào đó trong project, ta cần khai báo app đó trong file settings.py tại trường INSTALLED\_APPS.

Ví dụ: INSTALLED\_APPS = ('your\_app',)

* Để tạo một app: python manage.py startapp <app name>

Thư mục tạo ra có cấu trúc như sau:

App name

models.py

views.py

admin.py

\_\_init\_\_.py

tests.py

### Model

* Model là nguồn định nghĩa thông tin từ cơ sở dữ liệu. Nó bao gồm các lớp và các thuộc tính của dữ liệu. Thông thường, mỗi một class model tương ứng với một bảng cơ sở dữ liệu. Mỗi thuộc tính trong class sẽ tương ứng với một trường cơ sở dữ liệu.
* Model có thể được đồng bộ với cơ sở dữ liệu bằng lệnh:

python manage.py syncdb

### Class

* Các class được định nghĩa trong file models.py, kế thừa từ class django.db.models.Model. Mỗi class sẽ tương ứng với một bảng trong cơ sở dữ liệu.

Ví dụ: class Author(models.Model):

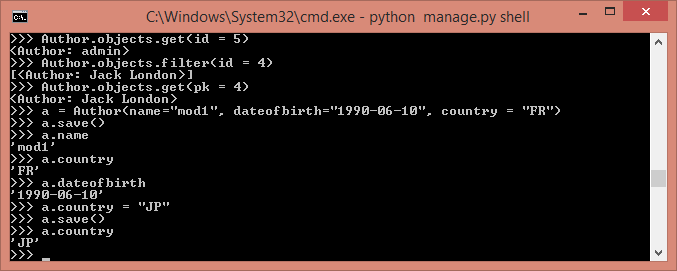
* Mỗi class có các thuộc tính riêng ứng với các trường của bảng đó trong cơ sở dữ liệu. Các class \*Field định nghĩa kiểu của thuộc tính đó.

Ví dụ: name = models.CharField(max\_length=200)

* Django thao tác với các bản ghi của các bảng thông qua các đối tượng (object) thuộc class tương ứng.

### Giao diện lập trình ứng dụng truy cập cơ sở dữ liệu (Database access API)

* Khởi động API bằng lệnh: python manage.py shell



Hình 2: Hình ảnh của giao diện truy cập cơ sở dữ liệu.

* API cho phép ta truy vấn cũng như chèn, sửa, xóa cơ sở dữ liệu một cách nhanh chóng qua các câu lệnh python. Kèm theo đó, python là ngôn ngữ thông dịch nên ta nhận được kết quả ngay mà không cần qua bước biên dịch.
* Các thao tác với bản ghi đều thông qua các đối tượng, điều này thể hiện tính hướng đối tượng cao của Python.
* Tuy nhiên, Django vẫn cho phép ta sử dụng các câu lệnh truy vấn SQL (Raw Query) nếu ta muốn.

Ví dụ:

for a in Author.objects.raw('SELECT id, name FROM blogs\_author'):

... print(a)

...

## Giao diện quản trị (Admin Interface)

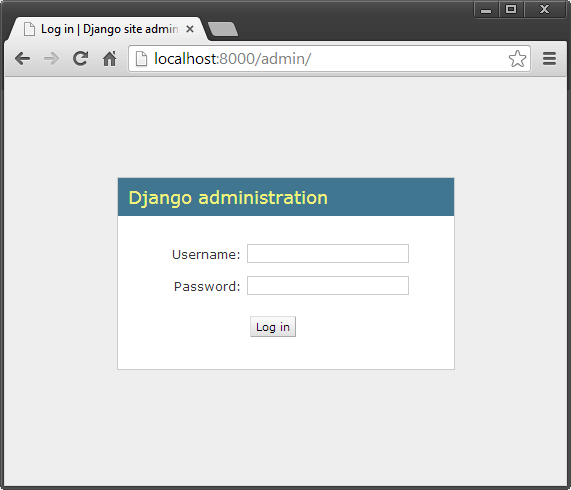
Django cung cấp sẵn một giao diện quản trị cho phép người dùng quản lý các đối tượng cũng như các class của trang web. Việc này rất có lợi:

* Khi người dùng không có nhiều kiến thức về kỹ thuật, họ có thể dùng sẵn trang quản trị này để quản lý thông tin của mình.
* Kiểm tra mô hình dữ liệu (data model) một cách trực quan.
* Cho phép quản lý dữ liệu của trang web.
* Trang giao diện với các chức năng đơn giản, dễ dùng và nhanh chóng.

### Sử dụng giao diện quản trị

* Để sử dụng được giao diện quản trị, ta cần khai báo app 'django.contrib.admin' ở file settings.py tại trường INSTALLED\_APPS:
* Để truy cập vào trang quản trị, ta cần khai báo đường dẫn (url) đến trang trong file urls.py:

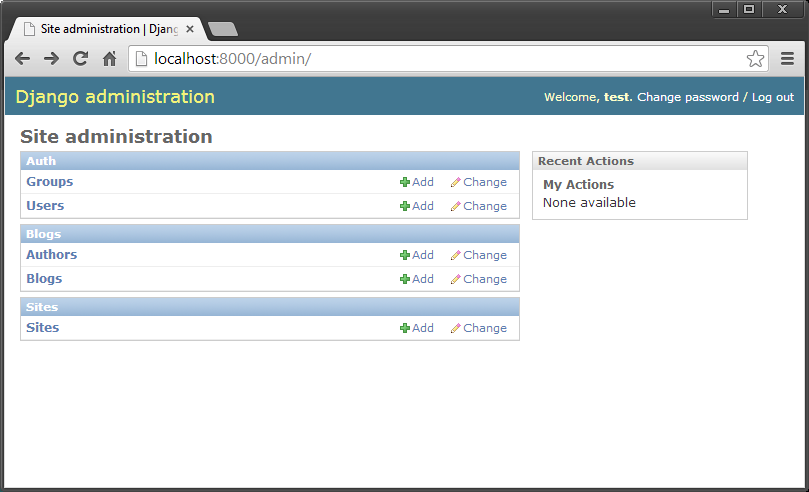
url(r'^admin/', include(admin.site.urls)),



Hình 3: Trang đăng nhập vào trang quản trị

* Để giao diện quản trị quản lý các class, ta cần phải khai báo các class đó trong file admin.py của app đó:

Ví dụ: admin.site.register(your\_class)



Hình 4: Giao diện quản trị

### Tùy biến lớp ModelAdmin

* Django cho phép ta tùy biến các class được quản lý bằng cách khai báo một class kế thừa lớp ModelAdmin. Trong class này ta có thể đặt lại các giá trị của các tùy chọn đã có của lớp cha ModelAdmin hay định nghĩa một hàm chức năng mới.

Ví dụ:

class AuthorAdmin(admin.ModelAdmin):

*# tùy chọn*

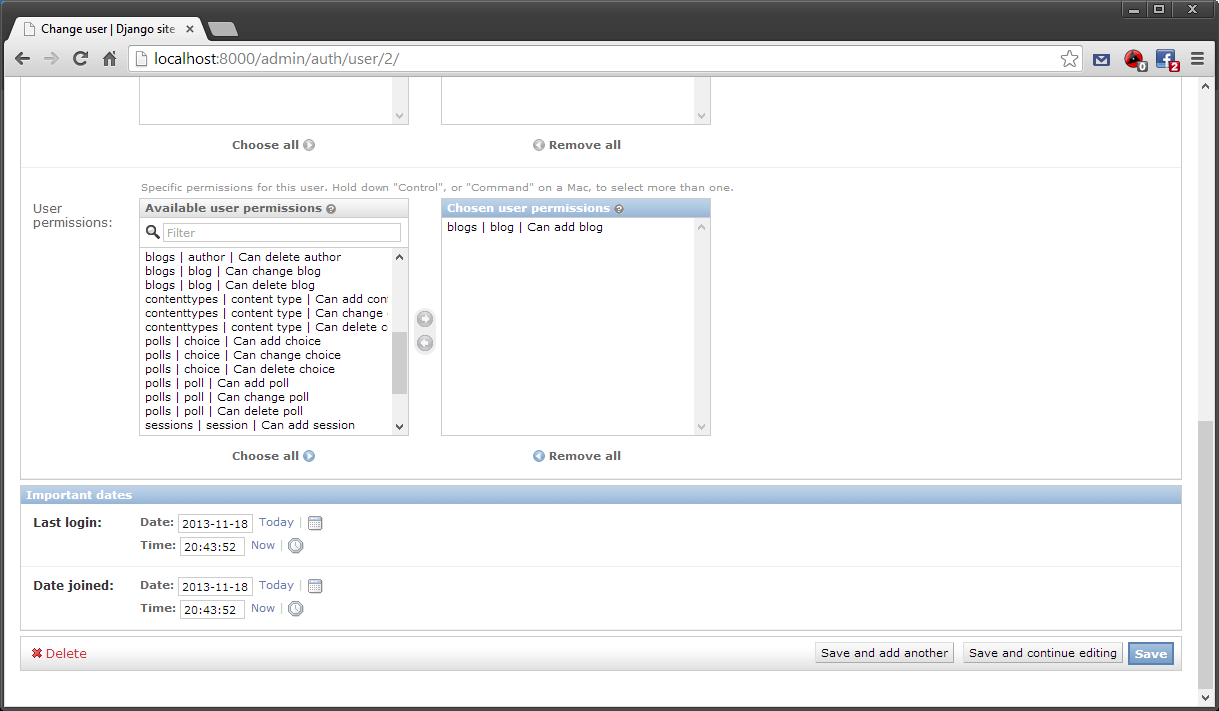
*# hàm*

admin.site.register(Author,AuthorAdmin)

Ta có thể hiểu việc tùy biến trên rằng class Author được khai báo với các tùy chọn của class AuthorAdmin. Nếu các tùy chọn không được đặt thì Django sẽ sử dụng các giá trị mặc định cho các tùy chọn đó.

### Quản lý quyền (Permission)

Trang quản trị cũng cung cấp cho nhà quản trị việc quản lý quyền của các người dùng. Ví dụ như người dùng này có quyền thêm, sửa hay xóa đối với class nào của app nào. Điều này rất hữu ích trong việc hạn chế, cũng như điều chỉnh việc can thiệp của các người dùng đến thông tin của trang web.



Hình 5: Trang quản lý quyền người dùng.

## URL linh động

Để thiết kế các URL, ta cần tạo một module gọi là URLconf. Đây là mã python (đặt trong file urls.py) quy định việc ánh xạ một url này ứng với hàm view nào. Nó có thể dài hay ngắn tùy thuộc vào người lập trình và nó cũng có thể tham chiếu đến những URLconf khác.

### Sử dụng URL

* Các đường dẫn (url) được khai báo trong file urls.py, người dùng có thể thêm, xóa hoặc tùy chỉnh các url theo ý muốn của mình tại đây. URL được định nghĩa bằng đường dẫn của nó và hàm view mà nó ánh xạ đến.

Ví dụ:

from django.conf.urls import patterns, include, url

urlpatterns = patterns('',

url(r'^your\_view/', views.your\_view),

)

* Ta cũng có thể tạo URLconf cho từng app. Django cho phép tham chiếu từ một URLconf này đến URLconf của một app khác bằng lệnh include.

Ví dụ:

urlpatterns = patterns('',

url(r'^your\_app/', include('your\_app.urls')),

)

### Quá trình xử lý yêu cầu

Khi người dùng gửi một URLrequest đến, các bước để trả về trang người dùng cần sẽ là:

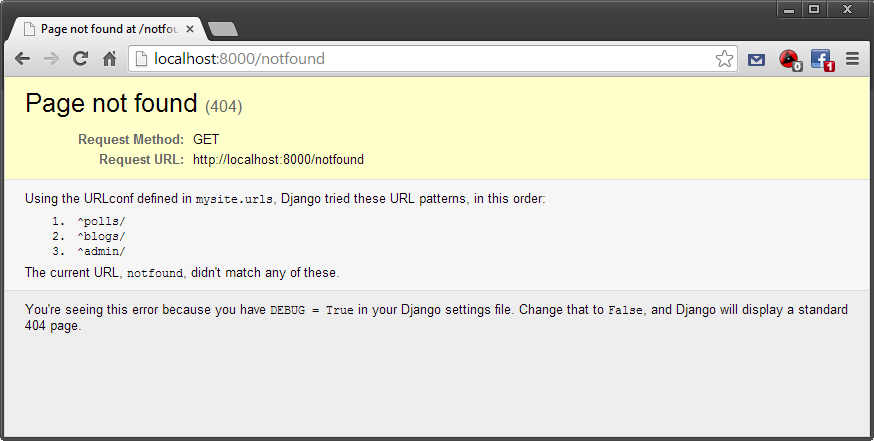
**B1**: Xác định URLconf module.

**B2**: Load các module và tìm kiếm các biến urlpatterns.

**B3**: Chạy lần lượt qua mỗi URL parttern, và dừng lại ở giá trị đầu tiên trùng với URL đã yêu cầu.

**B4**: Nếu tồn tại URL đáp ứng yêu cầu, Django sẽ gọi hàm view tương ứng.

**B5**: Nếu không có URL nào được định nghĩa như yêu cầu, trang xử lý lỗi được trả về.



Hình 6: Trang lỗi được trả về trong trường hợp không tìm thấy URL yêu cầu.

### URL động

* Trong trường hợp, ta muốn với mỗi url cùng tham số truyền sau sẽ trả về cho ta một kết quả khác mà không muốn viết lại nhiều lần định nghĩa url cho từng tham số, có một cách hữu hiệu giải quyết việc này đó là sử dụng URL động.

Ví dụ thay vì ta định nghĩa riêng các url ứng với từng hàm báo giờ:

urlpatterns = patterns('',

url(r'^time/plus/1/$', one\_hour\_ahead),

url(r'^time/plus/2/$', two\_hours\_ahead),

url(r'^time/plus/3/$', three\_hours\_ahead),

url(r'^time/plus/4/$', four\_hours\_ahead),

)

Ta có thể rút gọn lại bằng một định nghĩa url duy nhất:

urlpatterns = patterns('',

url(r'^time/plus/(\d+)/$', hours\_ahead),

)

Hàm hours\_ahead(request, offset)

Ở đây, (\d+) đóng vai trò hậu tố là tham số truyền vào, hàm hours\_ahead được định nghĩa với đầu vào offset là 1 tham số nhận được từ url, do đó nó có thể bắt giá trị từ url và trả về kết quả tương ứng.

## Hệ thống mẫu (Template system).

Cung cấp các templates dựa trên các cú pháp thân thiện của template language, có thể mở rộng để tách riêng rẽ thiết kế, nội dung và mã code python. Nó là Django’s template layer sử dụng template language giúp cân bằng giữa việc dễ sử dụng và công năng hiệu quả của nó. Nó được thiết kế sử dụng tiện lợi như làm việc với HTML và dễ làm quen kể cả với những người vốn dùng các template language khác như Smarty hay Cheetah Template. Về tổng quan, template language cung cấp các thẻ tag kiểm soát tính logic của mẫu qua việc dử dụng các filter được cung cấp sẵn. Một template là một file text có thể được tạo ra dựa trên nhiều định dạng cơ bản như HTML, XML, CSV…

### Thành phần

#### Các thẻ tag

Có dạng {%tag%}, một số bắt buộc có thẻ tag kết thúc {%tag%} ... tag contents ... {% endtag %}, một số có thể tạo đoạn text cho đầu ra, một số điều khiển việc thực hiện các phép logic hay vòng lặp, một số lại load thông tin bên ngoài vào để các biến sử dụng. Django cung cấp khoảng 20 loại thẻ tag khác nhau. Dưới đây là một số thẻ tag phổ biến hay được dùng:

Thẻ tag for: {% for..in…%}...{% endfor %}vòng lặp duyệt từng phần tử trong mảng

Vd: để hiển thị danh sách các vận động viên điền kinh ta sử dụng thẻ tag *for*:

<ul>

{% for athlete in athlete\_list %}

<li>{{ athlete.name }}</li>

{% endfor %}

</ul>

Thẻ tag *if and else* : {% if %}…{% else %} … {% endif %}.

Ngoài ra thông thường ta hay sử dụng các thẻ tag *if* và  *for* kết hợp với nhau. Các thẻ tag *block* and *extends:* thường được hay sử dụng trong việc thừa kế (đề cập trong mục thừa kế) của các templates.

#### Các biến (variables)

Có dạng như {{variable}} tên biến có thể được đặt bất kỳ kết hợp giữa các ký tự chữ cái và dấu gạch dưới “\_” ngoại trừ dấu cách và dấu chấm. Biến có chứa các thuộc tính được truy cập qua dấu chấm “.”. Ví dụ: {{section.title}} biến là một đối tượng có tên là *section* và có thuộc tính *title*. Thuộc tính này được truy cập qua việc sử dụng dấu “.”. Nếu trong trường hợp ta sử dụng một biến không tồn tại thì template system sẽ chèn giá trị mặc định của biến TEMPLATE\_STRING\_IF\_INVALID mặc định là xâu rỗng ‘’. Biến thường được sử dụng kết hợp với các filters để có thể điều chỉnh sự biểu hiện của các biến.

#### Các bộ lọc (filters)

Sử dụng kết hợp với các biến có dạng {{ variable|filter }}. Sử dụng dấu số đứng “|” để gắn filter vào biến.

Ví dụ: {{name|lower}} biến name được lọc qua filter “lower” và chỉ là những tên được viết bằng chữ thường.

Nhiều filter có thể được sử dụng cùng một lúc bằng việc ghép nối với nhau

Ví dụ: {{text|escape|linebreaks}}

Một vài filter có chứa các đối số. Ví dụ: {{ bio|truncatewords:30 }} lấy 30 từ đầu tiên của biến *bio.*

Django cung cấp sẵn khoảng 30 filter khác nhau, đồng thời cho phép ta có thể tự tạo ra các filters riêng theo người dùng. Một số filter hay được sử dụng:

Filter *default* : nếu một biến là false hay rỗng sẽ được gán giá trị ngầm định được gán sẵn.

Ví dụ: {{ value|default:"nothing" }}

Nếu biến value không được cung cấp từ trước hay rỗng thì biến sẽ được hiện ra dưới chữ “nothing*”*.

Filter *length*: trả về độ dài của biến, áp dụng cho cả xâu và danh sách. Giả sử biến value là một danh sách gồm 4 phần tử [1, 2, 3, 4] thì kết quả trả về sẽ là 4.

Ví dụ: {{ value|length }}

Filter *striptags*: gộp tất cả các thẻ [X]HTML trong một biến thành một dải. Giả sử biến *value* là <b>I</b> <button>is</button> a <span>student</span> thì kết quả khi sử dụng với bộ lọc s*triptags* là “I am a student”.

#### Các comment trong file

Có dạng như {#comment#}

Ví dụ: {# greeting #}hello

Một comment có thể bao gồm bất kỳ mã code nào dù chúng hợp lệ hay không.

Ví dụ: {# {% if true %}bar{% else %} #}

### Đặc điểm

#### Thừa kế trong Django’s template

Cho phép ta xây dựng một template cha-cơ bản bao gồm tất cả các thành phần phổ biến trong website và chỉ rõ những *blocks* được khai báo bằng thẻ tag{%block%} mà các template con khác khi kế thừa có thể ghi đè lên trên các *blocks* trên. Thừa kế được khai báo và sử dụng với từ thẻ tag {%*extends…%}* để báo với template engine rằng template này thừa kế (mở rộng) từ một template khác. Khi một template kế thừa từ một template khác nó sẽ có thể ghi đè (override) lên các block của template cha hoặc có thể sử dụng lại các nội dung đã được định nghĩa ở template cha. Các nội dung thuộc các blocks của template cha luôn đươc sử dụng như là dự phòng khi các blocks tương ứng của template con không được override. Ví dụ: template cha.

<!DOCTYPE html>

<html lang="en">

<head>

<link rel="stylesheet" href="style.css" />

<title>{% block title %}My amazing site{% endblock %}</title>

</head>

<body>

<div id="sidebar">

{% block sidebar %}

<ul>

<li><a href="/">Home</a></li>

<li><a href="/blog/">Blog</a></li>

</ul>

{% endblock %}

</div>

<div id="content">

{% block content %}{% endblock %}

</div>

</body>

</html>

Template con:

{% extends "base.html" %}

{% block title %}My amazing blog{% endblock %}

{% block content %}

{% for entry in blog\_entries %}

<h2>{{ entry.title }}</h2>

<p>{{ entry.body }}</p>

{% endfor %}

{% endblock %}

Template con đã thừa kế template cha là *base.html* bằng việc sử dụng tag *extends*. Cụ thể, base.html có 3 blocks là {% block title %}; {% block sidebar %} và {% block content %} có thể được override tại template con. Trong ví dụ này thì template con đã override lại hai blocks của base.html là {% block title %}; {% block content %} và sử dụng nguyên block {% block sidebar %} của base.html. Thừa kế có thể được sử dụng với nhiều cấp độ (levels) nếu cần. Thông thường thừa kế được sử dụng với 3 cấp. Một số chú ý khi sử dụng thừa kế trong hệ thống mẫu:

* Thẻ tag [{% extends %}](https://docs.djangoproject.com/en/1.6/ref/templates/builtins/#std:templatetag-extends) phải là thẻ tag đầu tiên trong template nếu template đó muốn thừa kế từ một template khác, nếu không sẽ không có hiệu lực.
* Nên thiết kế nhiều  [{% block %}](https://docs.djangoproject.com/en/1.6/ref/templates/builtins/#std:templatetag-block) và nên định trước số blocks trong template để khi thừa kế các template con chỉ cần định nghĩa lại những nội dung của một số ít block chứ không phải định nghĩa lại tất cả nội dung nếu sử dụng ít block.
* Nếu cần lấy nội dung trong 1 block của template cha có thể sử dụng {{ block.super }}
* Có thể tùy chọn đặt tên cho các block nhưng không thể đặt tên các block giống nhau trong cùng một template.

#### Truy nhập các phương thức (Accessing method call).

Các template có thể truy nhập nhiều không chỉ là các thuộc tính và các biến truyền từ các view. Ví dụ, Django cung cấp cú pháp entry\_set để tìm kiếm một tập các đối tượng có liên hệ qua một khóa ngoại hay QuerySets cung cấp hàm count() để đếm số các object mà template chứa. Ta có thể truy cập các phương thức đã được xác định rõ ràng trong các models. Ví dụ:

# In model

class Task(models.Model):

def foo(self):

return "bar"

# In template

{{ task.foo }}

Tuy nhiên do Django giới hạn việc xử lý logic trong template nên ta không thể truyền tham số vào các lời gọi hàm từ template mà thay vào đó việc tính toán nên thực hiện ở các view rồi truyền qua template để hiển thị.

#### Tự động thoát khỏi HTML (Automatic HTML escaping).

Trong một số trường hợp người dùng vô ý hay cố ý gửi dữ liệu không hợp lệ có thể gây ra những kết quả không mong đợi ví dụ như thay vì gửi dữ liệu là tên của mình thì họ có thể gửi kèm các thứ khác như *<b>username* làm cho cả phần còn lại của trang web bị in đậm gây sai lệch nội dung. Do đó Django cung cấp 2 tùy chọn để phòng tránh:

* Cho các biến không đáng tin cậy qua filter *escape* để có thể vô hiệu hóa các ký tự HTML có hại cho trang web. Tuy nhên tùy chọn này có thể dẫn đến việc bỏ sót dữ liệu của người dùng.
* Sử dụng automatic HTML escaping của Django. Tùy chọn này được mặc định nếu sử dụng template của hệ thống. Năm ký tự thường mang nhưng nguy cơ có hại được chuyển sang những ký tự vô hại khác:

< được chuyển thành &lt;

> được chuyển thành &gt;

' được chuyển thành &#39;

" được chuyển thành &quot;

& được chuyển thành &amp;

Django cũng cung cấp tùy chọn tắt auto-escape trong một số trường hợp người dùng muốn đưa ra các nội dung chứa các mã thô HTML như việc hiển thị một số mã HTML trong một bài hướng dẫn nào đó. Đối với các biến cá nhân, riêng lẻ thì có thể sử dụng filter *safe* để vô hiệu hóa auto-escape. Vd: {{ data|safe }}thì khi biến data có chứa <b> thì cũng sẽ ko bị chuyển thành *&lt;b&gt;* do đó vẫn giữ được nội dung mà người dùng muốn đưa vào mà không gây hại gì đến trang web. Đối với các block của các template có thể sử dụng thẻ tag *autoescape .*Ví dụ:

{% autoescape off %}

Hello {{ name }}

{% endautoescape %}

Auto-escape cũng có tác dụng với thừa kế đối với các template con.

## Hệ thống Cache (Cache system)

Thông thường đối với một trang web động, với mỗi một yêu cầu request từ client thì bên phía Web server sẽ tiến hành các tính toán – từ thời gian truy vấn cơ sở dữ liệu đến thời gian generate temple, tạo trang giao diện để hiển thị tới cho client, những tính toán này đòi hỏi mất một lượng thời gian nhiều. Đối với những trang web vừa và nhỏ với cơ sở dữ liệu cũng như cấu trúc đơn giản thì lượng thời gian xử lý đó là hoàn toàn có thể chấp nhận được, tuy nhiên đối với những trang web từ trên trung bình tới lớn thì với việc xử lý lượng truy cập khổng lồ cũng như cơ sở dữ liệu lớn, phức tạp thì lượng thời gian để xử lý những công việc trên là lớn, kéo theo hiệu suất của cả trang web xuống. Để giải quyết vấn đề này cần phải cắt giảm thời gian dành cho xử lý và kỹ thuật *cache* được áp dụng vào trong Django Framework. Hiểu đơn giản ta sử dụng một bộ nhớ đệm cache để lưu những kết quả có thời gian xử lý yêu cầu lâu và cho phép sử dụng lại một cách nhanh chóng mà không cần tính toán lại vào những lần tiếp theo. Nghĩa là khi có một yêu cầu được gửi đến thì hệ thống sẽ tìm xem có trang tương ứng trong cache hay không nếu có thì trả về trang đó từ cache còn lại thì sẽ tiến hành xử lý bình thường tạo trang, lưu vào cache để cho những lần kế tiếp đồng thời trang được trả về cho client. Django cung cấp một hệ thống cache (cache system) phong phú được hỗ trợ tốt với các cấp độ: có thể dùng cache để lưu một vài trang giao diện hay chỉ những phần trang web phức tạp tạo ra khó khăn tốn thời gian xử lý hay có thể là toàn bộ trang web. Bạn không trực tiếp điều khiển nhưng có thể đưa ra các gợi ý thông qua HTTP headers về những phần của website cần được lưu trong cache và lưu như thế nào. Trong Django để sử dụng kỹ thuật cache ta cần một vài thiết lập nhỏ đó là việc chỉ ra sử dụng kiểu cache nào. Điều này rất quan trọng và nó ảnh hưởng đến hiệu suất của cache, vì vậy nên cân nhắc để có thể có được hiệu suất phù hợp nhất.

### Các kiểu cache

#### Cache bộ nhớ (Memcached)

Là kiểu cache hiệu quả, nhanh nhất, được đặt và được phân bổ theo một số quy định của bộ nhớ RAM. Nó cung cấp một giao diện nhanh chóng để thêm, lấy và xóa dữ liệu trong cache. Tất cả dữ liệu đều được lưu trực tiếp trên bộ nhớ do đó cơ sở dữ liệu cà filesystem là không cần thiết. Tuy nhiên chi phí cho memcached là cao so với các loại khác, dễ bị mất nếu server xảy ra sự cố. Để sử dụng memcached ta cần cài đặt memcached từ <http://memcached.org/> và tiến hành một số cài đặt liên kết với python-memcached hoặc pylibmc. Sau đó Set BACKEND thành ‘django.core.cache.backends.memcached.MemcachedCache’ hoặc ‘django.core.cache.backends.memcached.PyLibMCCache’ và set location dưới dạng ip:port.

Ví dụ: Memcached chạy trên localhost có ip là 127.0.0.1 với cổng 11211 sử dụng python-memcahed.

CACHES = {

'default': {

'BACKEND': 'django.core.cache.backends.memcached.MemcachedCache',

'LOCATION': '127.0.0.1:11211',

}

}

Trên linux có sẵn thông qua một file socket /tmp/memcahed.sock do đó ví dụ trên sẽ thành

CACHES = {

'default': {

'BACKEND': 'django.core.cache.backends.memcached.MemcachedCache',

'LOCATION': 'unix:/tmp/memcached.sock',

}

}

Memcached có khả năng chia sẻ cache trên nhiều server nghĩa là có thể chạy memcahed trên nhiều máy và chương trình sẽ coi tập các máy đó chỉ là một cache mà không cần phải nhân bản cache trên mỗi máy. Ví dụ ta chia sẻ cache trên các server có địa chỉ IP làn lượt là 10.10.10.1 và 10.10.10.2 trên cổng 11211.

CACHES = {

'default': {

'BACKEND': 'django.core.cache.backends.memcached.MemcachedCache',

'LOCATION': [

'172.19.26.240:11211',

'172.19.26.242:11211',

]

}

}

#### Cache cơ sở dữ liệu (Databse caching)

Sử dụng một bảng được gọi là cache table trong cơ sở dữ liệu làm cache backend. Để tạo một cache table ta sử dụng lệnh command sau:

$ python manage.py createcachetable [cache\_table\_name]

Tham số cache\_table\_name là tên của cache table cần tạo, cần hợp lệ và không trùng với tên các bảng khác đã tạo trong cơ sở dữ liệu. Cache table được tạo sẽ có định dạng phù hợp với database-cach system quy định. Để sử dụng cache table đã tạo ở trên làm backend cần phải thiết lập các BACKEND và LOCATION. Ví dụ với LOCATION là tên của cache table:

CACHES = {

'default': {

'BACKEND': 'django.core.cache.backends.db.DatabaseCache',

'LOCATION': 'my\_cache\_table',

}

}

Cần sử dụng database caching backend giống với database backend được chỉ ra trong file settings. Nếu sử dụng nhiều cơ sở dữ liệu thì cần phải thiết lập hướng dẫn định tuyến cho cache table.

#### Cache file hệ thống (Filesystem caching)

Là kiểu cache được lưu trữ trên filesystem. Mỗi giá trị cache sẽ được lưu dưới dạng 1 file riêng biệt và nội dung của chúng được lưu theo một trình tự sử dụng module *pickle* của Python. Tên của mỗi file là một *cache key* nhằm đảm bảo sử dụng filesystem một cách an toàn không trùng lặp. Để sử dụng ta cũng tiến hành thiết lập cho BACKEND và LOCATION. Ví dụ:

CACHES = {

'default': {

'BACKEND': 'django.core.cache.backends.filebased.FileBasedCache',

'LOCATION': '/var/tmp/django\_cache',

}

}

Lưu ý với LOCATION thì nên để đường dẫn tuyệt đối. Thư mục được chỉ đến phải tồn tại và có thể thực hiện các thao tác đọc/ghi bởi Webserver.

#### Cache bộ nhớ địa phương (Local-memory caching)

Là một lựa chọn nếu không đủ khả năng dùng *memcache* nhưng vẫn muốn có một cache system tốc độ cao. Local-memory cache gồm nhiều process và thread. Tuy nhiên kiểu cache này không hiệu quả đối với môi trường sản xuất (production environment) vì do mỗi process trong đó đều đã có một cache riêng. Ví dụ:

CACHES = {

'default': {

'BACKEND': 'django.core.cache.backends.locmem.LocMemCache',

'LOCATION': 'unique-snowflake'

}

}

LOCATION dùng để xác định memory lưu trữ cache. Nếu chỉ có một locmem cache thì ta có thể bỏ qua LOCATION nhưng nếu có nhiều hơn ta nên đặt tên cho các locmem cache để phân biệt chúng.

#### Dummy caching

Kiểu cache này không hẳn thực sự là cache nó chỉ cài đặt cache interface mà không làm thêm gì nữa. Nó sẽ hữu dụng khi ta có một website sản phầm sử dụng cache lớn đặt ở nhiều nơi khác nhau trong khi với mối trường phát triển hiện thời ta không muốn thay đổi mã nguồn đối với các cache ở những lần sau này. Ví dụ:

CACHES = {

'default': {

'BACKEND': 'django.core.cache.backends.dummy.DummyCache',

}

}

#### Custom cache

Django hỗ trợ một số cache backend ngoài những loại cache trên. Để sử dụng một cache backend khác dùng Python import đường dẫn đối với BACKEND như sau: *'BACKEND': 'path.to.backend'.*

### Các tham số của CACHE

* TIMEOUT: là thời gian sử dụng cache giá trị mặc định là 300s
* OPTION: cung cấp các tùy chọn phụ trợ.
* MAX\_ENTRIES: số lượng tối đa các mục trong cache trước khi các giá trị cũ bị xóa, giá trị mặc định là 300
* CULL\_FREQUENCE: chỉ ra (1/CULL\_FREQUENCE) các mục được lựa chọn để xóa khi đạt tới MAX\_ENTRIES, giá trị mặc định là 3. Nếu truyền vào giá trị 0 sẽ được hiểu là xóa toàn bộ cache
* KEY\_PREFIX: tự động thêm các tiền tố cho tất các cache keys để có Django server có thể sử dụng
* VERSION: phiên bản mặc định cho cache keys được phát bởi Django server.
* KEY\_FUNCTION: xâu chứa đường dẫn tới hàm định nghĩa cách thức tạo ra prefix, version, key info vào cache key.

Ví dụ:

CACHES = {

'default': {

'BACKEND': 'django.core.cache.backends.filebased.FileBasedCache',

'LOCATION': '/var/tmp/django\_cache',

'TIMEOUT': 60,

'OPTIONS': {

'MAX\_ENTRIES': 1000

}

}

}

### Các cấp độ cache

#### The per-site cache

Là lưu đệm toàn bộ trang web. Để làm điều này ta diều chỉnh file settings bằng cách thêm 2 dòng sau vào MIDDLEWARE\_CLASSESS:

'django.middleware.cache.UpdateCacheMiddleware'

'django.middleware.cache.FetchFromCacheMiddleware'

Sau đó ta thêm 3 dòng sau vào file settings

CACHE\_MIDDLEWARE\_ALLIAS: bí danh của cache

CACHE\_MIDDLEWARE\_SECONDS: thời gian (seconds) trang được lưu trong cache.

CACHE\_MIDDLEWARE\_PREFIX: sử dụng khi cache được chia sẻ trên nhiều trang web sử dụng cài đặt Django giống nhau, thiết lập là tên trang web hoặc một chuỗi là duy nhất trong Django nhằm tránh xung đột khóa. Sử dụng xâu rỗng nếu không quan tâm đến vấn đề này.

Hoạt động: Cache middleware lưu GET và HEAD response với status là 200. Các response cho các request của cùng một URL nhưng khác nhau về các tham số sẽ được coi là những page và những cache riêng biệt. Cache middleware sẽ chờ đến khi HEAD request được phản hồi bằng một header response mà được lưu ở trong cache tương ứng với GET request trước đó. Khi đó, per-site cache sẽ trả về một GET response ứng với HEAD request. Cache middleware sẽ tự động thêm vào một vài header trong mỗi HttpResponse là *Last-Moddified* (tiêu đề ngày/giờ hiện tại khi một phiên bản mới của trang được yêu cầu); *Expires* (tiêu đề ngày/giờ hiện tại cộng với CACHE\_MIDDLEWARE\_SECONDS); *Cache-Control* (tiêu đề đưa ra độ tuổi lớn nhất của trang từ CACHE\_MIDDLEWARE\_SECONDS).

#### The per-view cache

Sử dụng bộ nhớ đệm để lưu kết quả của các view riêng biệt. d*jango.views.decorators.cache* sẽ định nghĩa một *cache\_page* tự động lưu kết quả của view. Ví dụ:

from django.views.decorators.cache import cache\_page

# cache\_page có một tham số là cache timeout thời gian lưu kết quả của view my\_view trong cache

@cache\_page(60 \* 15)

def my\_view(request):

Với nhiều URLs cùng trỏ đến cùng một view thì với mỗi URL sẽ được lưu trữ riêng rẽ. Và khi có một URL cụ thể giống với một URL đã được yêu cầu trước đó thì sẽ sử dụng cache tương ứng. *Cache\_page* có một vài tham số tùy chọn như *cache* (hướng đến sử dụng một cache cụ thể); *key\_prefix* (tương tự như CACHE\_MIDDLEWARE\_KEY\_PREFIX). Ta có thể chỉ định trực tiếp cache view trong URLconf. Ví dụ trên ta sẽ điều chỉnh một chút file urls.py như sau:

from django.views.decorators.cache import cache\_page

urlpatterns = ('',

(r'^foo/(\d{1,2})/$', cache\_page(60 \* 15)(my\_view)),

)

#### Template fragment caching

Lưu trữ cache các đoạn (phần) của template bằng cách sử dụng thẻ tag *cache* template. Để sử dụng thẻ tag này trong template ta cần đặt thẻ tag {% load cache %} vào gần đầu template. Thẻ tag {% cache %} sẽ lưu trữ block trong một khoảng thời gian tính bằng giây. Thẻ này có hai tham số là cache timeout và tham số tên của cache. Ví dụ:

{% load cache %}

{% cache 500 sidebar %}

.. sidebar ..

{% endcache %}

#### The low-level cache API

Ngoài các phương pháp cache trên ta có thể sử dụng low-level cache API. Có thể sử dụng để lưu các đối tượng vào cache như bạn muốn như xâu, danh sách các model objects… Django cung cấp module cache, *django.core.cache* có một đối tượng cache được tạo tự động mặc định *from django.core.cache import cache*

* Lấy một đối tượng cache nếu có nhiều cache

>>> from django.core.cache import get\_cache

>>> cache **=** get\_cache('alternate')

* Thiết lập và get cache: set(key, value, timeout) và get(key). Ví dụ:

>>> cache**.**set('my\_key', 'hello, world!', 30)

>>> cache**.**get('my\_key')

* Phương thức add() để thêm vào một key duy nhất nếu nó chưa tồn tại, có các tham số giống với set() nhưng nó sẽ không cố gắng thay đổi một key đã tồn tại như set()
* Phương thức set\_many(), get\_many() truy cập nhiều cache một lúc:

>>> cache**.**set\_many({'a': 1, 'b': 2, 'c': 3})

>>> cache**.**get\_many(['a', 'b', 'c'])

* Phương thức xóa delete(): >>> cache**.**delete('a')
* Phương thức xóa nhiều key một lúc delete\_many()

>>> cache**.**delete\_many(['a', 'b', 'c'])

* Xóa toàn bộ key clear(): >>> cache**.**clear()
* Tăng, giảm key: incr() và decr()

>>> cache**.**set('num', 1)

>>> cache**.**incr('num') 2

>>> cache**.**incr('num', 10) 12

>>> cache**.**decr('num') 11

>>> cache**.**decr('num', 5) 6

* Tiền tố cache (Cache key prefix): dùng cho trường hợp chia sẻ cache giữa các server và có sự khác nhau về định dạng lưu trữ giữa các server. Django cung cấp tiền tố cho tất cả các cache key được sử dụng bởi 1 server. Khi các cache key được lưu hay lấy ra thì Django sẽ tự động thêm tiền tố vào cache key với giá trị tiền tố được quy định bỏi KEY\_PREFIX trong phần cài đặt cache. Bằng cách này sẽ đảm bào được mỗi trường hợp sẽ có key\_prefix khác nhau và không xảy ra xung đột các giá trị cache.
* Phiên bản Cache (Cache version): thiết lập và lấy các cache theo version, mặc định giá trị version là 1.

# Đặt phiên bản cache cho một cache key là 2

>>> cache.set('my\_key', 'hello world!', version=2)

# Lấy phiên bản cache mặc định của cache key

>>> cache.get('my\_key')

# Lấy phiên bản 2 của cache key

>>> cache.get('my\_key', version=2)

#Tăng, giảm version của 1 key cụ thể

>> cache.incr\_version('my\_key')

* Cache key warning: django.core.cache.backends.base.CacheKeyWarning đưa ra cảnh báo nếu một cache key được sử dụng có thể gây ra lỗi như vượt quá 250 ký tự, có dấu cách hay có các ký tự điều khiển. Để tắt các cảnh báo ta có thể dùng

warnings**.**simplefilter("ignore", CacheKeyWarning)

#### Kiểm soát Cache (Controlling cache).

Dữ liệu được gồm hai loại là dữ liệu công khai và dữ liệu riêng tư, bí mật và tương ứng là hai kiểu cache: private cache và public cache. Điều này đặt ra vẫn đề với những dữ liệu riêng tư không được để trong public cache, việc này đòi hỏi ứng dụng web cần nói cho cache biết được đâu là dữ liệu riêng tư, bí mật đâu là dữ liệu công khai. Giải pháp mà Django cung cấp là chỉ ra những page’s cache cần được để chế độ private qua việc sử dụng *cache\_control*. Với *cache\_control* các HTTP header sẽ được xử lý một cách thích hợp khi được gửi đi. Ví dụ:

from django.views.decorators.cache import cache\_control

@cache\_control(private=True)

def my\_view(request):

Một vài tham số của *cache\_control*:

* public=True
* private=True
* no\_cache=True
* no\_transform=True
* must\_revalidate=True: Chỉ định một cache luôn được kiểm tra các phiên bản, chỉ cung cấp các nội dung của cache khi không có thay đổi.
* proxy\_revalidate=True
* max\_age=num\_seconds: Đặt ra maximum time cho một page lưu trong cache.
* s\_maxage=num\_seconds

Django cung cấp header để vô hiệu hóa caching bằng

django.views.decorators.cache.never\_cache*.*

Ví dụ:

from django.views.decorators.cache import never\_cache

@never\_cache

def myview(request):

# ...

## Quốc tế hóa (Internationalization).

Django hỗ trợ đầy đủ các ứng dụng đa ngôn ngữ với translation of text, formatting of date, times and numbers và time zones. Cơ bản Django thực hiện hai việc: một là cho phép các nhà phát triển, tác giả các template xác định những phần càn dịch hoặc cần định dạng sang ngôn ngữ và văn hóa địa phương, hai là địa phương hóa trang web qua những hook (móc nối các phần) để phù hợp với sở thích, thói quen của người bản địa.

* Translate of text: Django cung cấp các tiện ích để trích xuất các chuỗi dịch (translation strings) thành một message file. File này là một cách thuận tiện đối với người dịch tương đương với việc chuyển các chuỗi dịch thành ngôn ngữ đích. Một khi người dịch đã điền vào message file thì nó phải được biên dịch. Qua trình này dựa vào công cụ GNU gettext.
* Formatting of date: định dạng ngày cho phù hợp với các vùng miền khác nhau.
* Times and numbers: chuyển định dạng thời gian và số phù hợp với vùng khác nhau.
* Time zones: định dạng với đúng múi giờ với các vùng khác nhau.

Django tuân thủ ba bước để có thể quốc tế hóa trang web:

* Nhúng các chuỗi dịch (translate string) vào mã Python và mẫu (template)
* Lấy bản dịch cho các chuỗi với ngôn ngữ muốn hỗ trợ
* Kích hoạt các locate middleware trong mục setting của Django.

# Tổng kết

## Ưu điểm:

* Viết trên ngôn ngữ Python là một ngôn ngữ hướng đối tượng, có cú pháp đơn giản, dễ hiểu với mọi người được nhiều lập trình viên sử dụng. Python cũng là một ngôn ngữ mạnh được hỗ trợ bởi nhiều package và thư viện.
* Phát triển theo hướng tổ chức và cấu trúc: các bảng trong cơ sở dữ liệu được trừu tượng hóa thành các model với các thuộc tính và phương thức, các chức năng được coi là các views, hiển thị và thiết kế giao diện qua các mẫu (templates) và chúng được ghép nối qua các URL. Điều này giúp cho nhà phát triển có thể tổ chức, phát triển các phần khác nhau một thuận tiện dễ dàng.
* Dễ dàng chuyển dịch cú pháp: Django cung cấp caodebase phụ trợ giúp nhà lập trình không cần phải hiểu biết sâu về HTML, các loại ngôn ngữ SQL, thậm chí là cả Javascript hay AJAX cũng có thể tích hợp dễ dàng chúng vào trong trang web.
* Django tuân thủ chặt chẽ nguyên tắc DRY (don’t repeat yourself) được biết đến là chuẩn cho việc phát triển theo hướng đối tượng mà ở đây dựa trên Python ngôn ngữ hướng đối tượng.

## Nhược điểm:

* Nhà phá triển chậm cập nhật và nâng cấp ví dụ như hiện tại đã có Python phiên bản từ 3.0 trở lên nhưng Django vẫn mới chỉ hỗ trợ Python 2.6 hoặc mới chỉ có hỗ trợ một phần cho Python 3.0 trở lên.

## Phân công công việc

|  |  |
| --- | --- |
| **Thành viên** | **Tìm hiểu và trình bày** |
| Đỗ Thành Trung | Ánh xạ đồi tượng – quan hệ (Object-relational mapper) |
| Giao diện quản trị (Admin Interface) |
| URL linh động |
| Nguyễn Tiến Trung | Hệ thống mẫu (Template system) |
| Hệ thống cache (Cache system) |
| Tính quốc tế (Internationalization) |

# Tài liệu tham khảo

1. **Books**

*Pro Django - Alchin, Marty (16 December 2008).*

*The Definitive Guide to Django: Web Development Done Right - Holovaty, Adrian; Kaplan-Moss, Jacob (8 July 2009).*

1. **Online Documents**

*https://docs.djangoproject.com/en/1.5/ - Django Official Documentation.*

*http://www.djangobook.com/en/2.0/index.html - The Django Book.*

*http://mherman.org/blog/2012/12/30/django-basics/ - Django Basic.*