



 netmind

WeKnowIT



Python

© 2017, Netmind SL

WEB DEVELOPMENT

1. Flask
2. Django

1

Web frameworks

Flask

- [Flask](http://flask.pocoo.org/) que es un *microframework* desarrollado por Armin Ronacher que te permite crear aplicaciones web en un abrir y cerrar de ojos, todo con una cantidad absurdamente pequeña de líneas de código.
 - <http://flask.pocoo.org/>
 - <http://flask.pocoo.org/docs/0.12/api/>
- Flask, a diferencia de Django y Pyramid, no trae cientos de módulos para abordar las tareas más comunes en el desarrollo web, más bien se enfoca en proporcionar lo mínimo necesario para que puedas poner a funcionar una aplicación básica en cuestión de minutos. Es perfecto, por ejemplo, para el prototipado rápido de proyectos.



Mi primera web Flask

Instala Flask



➤ pip install flask

Crea una estructura de proyecto web

➤ Crea una estructura basada en el esqueleto

```
▼ skeleton_web
  ► bin
  ► docs
  ▼ NAME
    __init__.py
  ► templates
  ▼ tests
    __init__.py
    NAME_test.py
    setup.py
```

Crea un script de servidor



```
from flask import Flask  
app = Flask(__name__)
```

```
@app.route("/")  
def hello():  
    return "Hello World!"
```

Lanza el servidor

- › SET FLASK_APP=app.py
- › flask run
- › O simplemente: py app.py
- › Accede en el navegador a la dirección: <http://localhost:5000/>
- › CTRL+C para finalizar el servidor

Flask

- Hemos importado la clase Flask
- Seguidamente hemos creado una instancia *app* con el argumento `__main__` necesario para que Flask busque *files*, *plantillas* y *otros archivos*.
- *A continuación* hemos indicado que la ruta `/` (mediante la anotación `@app.route`) va a contener una función llamada *hello* que devolverá el mensaje que hemos escrito.
- Por último vamos a ejecutar la aplicación en el *host* que hemos asignado, creando un servidor web al instante.

Flask

- `@app.route` acepta dos parámetros: ruta y métodos

```
app.route('/hello', methods=["POST"])
```

- Asimismo excepta path params:

```
@app.route('/<username>')
```

```
def show_user(username):
```

```
    pass
```

```
@app.route('/post/<int:post_id>')
```

```
def show_post(post_id):
```

```
    pass
```

Flask - templates

➤ Crearemos las plantillas en el directorio templates

➤ *templates/index.html:*

```
<html>
  <head>
    <title>Gothons Of Planet Percal #25</title>
  </head>
  <body>

    {% if greeting %}
      I just wanted to say
      <em style="color: green; font-size: 2em;">{{ greeting }}</em>.
    {% else %}
      <em>Hello</em>, world!
    {% endif %}

  </body>
</html>
```

➤ Se pueden insertar código python usando la sintaxis `{%<python>%}` e insertar valores usando código mustache `{{<valor>}}`

Flask - templates

- Relacionaremos la template desde el código con `render_template`
- Para ello tendremos que importarlo

```
from flask import Flask  
from flask import render_template
```

```
app = Flask(__name__)
```

```
@app.route('/')  
def index():  
    greeting = 'Hello World'  
    return render_template("index.html", greeting=greeting)
```

```
if __name__ == "__main__":  
    app.run()
```

Flask - Inputs

- Podemos recoger los query params cuando tenemos una petición GET con:
 - `request.args.get(<param_name>', '<default>')`
- Asimismo podemos usar formularios para enviar peticiones GET o POST y recoger los parámetros con:
 - `request.form['<field_form>']`

Flask - Inputs

```
from flask import Flask
from flask import render_template
from flask import request

def create_client(app):
    @app.route('/hello')
    def hello_get():
        name = request.args.get('name', 'Nobody')

        if name:
            greeting = "Hello, %s" % name
        else:
            greeting = "Hello world"
        return render_template("index.html", greeting=greeting)

@app.route('/hello', methods=["POST"])
def hello():
    name = request.form['name']
    greet = request.form['greet']

    if greet:
```

```
        greeting = "%s, " % greet
    else:
        greeting = "Hello, "

    if name:
        greeting = "%s %s" % (greeting, name)
    else:
        greeting = "%s %s" % (greeting, "World")

    return render_template("index.html", greeting=greeting)

@app.route('/')
def index():
    return render_template('hello_form.html')

if __name__ == '__main__':
    app = Flask(__name__)
    create_client(app)
    app.run()
```

Flask templates

➤ Templates/hello_form.html:

```
<h1>Fill Out This Form</h1>

<form method="POST" action="/hello">
  <div class="pure-form">
    <fieldset>
      <legend>A Greeting App</legend>
      <input type="text" placeholder="greeting word" name="greet" />
      <input type="text" placeholder="your name" name="name" />
      <input class="pure-button pure-button-primary" type="submit"/>
    </fieldset>
  </div>
</form>
```

Flask – Layouts

- Se pueden crear esqueletos de plantillas para los elementos comunes y reutilizarlas en las otras plantillas
 - templates/layout.html:

```
<html>
<head>
  <title>Gothons From Planet Percal #25</title>
</head>
<body>

  {% block content %}

  {% endblock %}

</body>
</html>
```

Flask – Layouts

- En la plantilla secundaria se incorpora el layout usando `{% extends "<plantilla base>" %}`

- *templates/index_ly.html*:

```
{% extends "layout.html" %}
```

```
{% block content %}
```

```
{% if greeting %}
```

```
    I just wanted to say
```

```
    <em style="color: green; font-size: 2em;">{{ greeting }}</em>.
```

```
{% else %}
```

```
    <em>Hello</em>, world!
```

```
{% endif %}
```

```
{% endblock %}
```


Flask – Tests automáticos

- Se puede usar nose para implementar tests automáticos que simulen peticiones y el envío de formularios (y mucho más)
- Para ello se debe importar la aplicación y poner la propiedad testing de la aplicación a True

➤ tests/app_test.py:

```
from nose.tools import *  
from app_forms_ly import app
```

```
app.testing = True  
web = app.test_client()
```

```
def test_index():  
    result = web.get('/', follow_redirects=True)  
    assert_equal(result.status_code, 200)  
    assert_in(b"Fill Out This Form", result.data)
```

```
def test_hello():  
    result = web.get('/hello', follow_redirects=True)  
    assert_equal(result.status_code, 200)  
    assert_in(b"Hello, Nobody", result.data)
```

```
data = {'name': 'Ricardo', 'greet': 'Hey!'}  
result = web.post('/hello', follow_redirects=True,  
data=data)  
assert_in(b"Ricardo", result.data)  
assert_in(b"Hey!", result.data)
```



Implementando una API REST

- Crea una api REST que devuelve los datos de un usuario en función del path_param uid para la ruta:
 - /users/<uid>
- Necesitarás usar el jsonify de flask
 - `from flask import Flask, jsonify`
 - `return jsonify({'data':data})`

2

Django

Django

- Django es un framework web de Python. Es de código abierto y gratuito y fue lanzado en 2005.
- Es un framework muy maduro y tiene una excelente documentación y características impresionantes incluidas de forma predeterminada.
- Algunas excelentes herramientas que proporciona son:
 - Excelente servidor ligero para desarrollo y pruebas.
 - Buen lenguaje de plantillas.
 - Las características de seguridad como CSRF incluido desde la caja.
- Django implementa el paradigma Model, View, Template



Instalando y configurando Django

Instalando un entorno virtual



- Para evitar contaminar nuestro contexto global con paquetes innecesarios, vamos a utilizar un entorno virtual para almacenar nuestros paquetes.
- Usaremos virtualenv
 - pip install virtualenv
- Crearemos un directorio de proyecto y accederemos a él
 - mkdir dprojects
 - cd dprojects
- Añadiremos un directorio de aplicación
 - mkdir hello
- Añadiremos el entorno virtual en dicho directorio
 - cd hello
 - virtualenv env
- Activamos el entorno
 - .\env\Scripts\actívale → aparecerá el prompt (*env*)

Instalando Django



- Usamos PIP
 - `pip install django`

Creando una aplicación

- `django-admin startproject helloapp`
- La configuración de la aplicación está en `settings.py`
- `manage.py` nos ayudará a crear nuestras propias apps

```
▼ hello
  ► env
  ▼ helloapp
    ▼ helloapp
      __init__.py
      settings.py
      urls.py
      wsgi.py
      manage.py
```

Creando una aplicación propi



- Crea la aplicación y añadela al settings
 - `cd helloapp`
 - `python manage.py startapp howdy`
 - Añade la aplicación creada al settings

```
INSTALLED_APPS = [  
    ...  
    'django.contrib.staticfiles',  
    'howdy'  
]
```
- Inicializa el servidor
 - `python manage.py runserver`
- Accede en el navegador a: <http://localhost:8000/>
- Para el servidor con CTRL+C

Urls & Templates



- Para configurar las rutas necesitamos modificar el archivo `urls.py`
 - Incluye "include" en `url.py` y añade la ruta del archivo de configuración de rutas de `howdy`

```
from django.conf.urls import url, include
from django.contrib import admin
```

```
urlpatterns = [
    url(r'^admin/', admin.site.urls),
    url(r'', include('howdy.urls')),
]
```

- Crea el archivo `howdy/urls.py`
 - # `howdy/urls.py`
 - from `django.conf.urls` import `url`
 - from `howdy` import `views`

```
urlpatterns = [
    url(r'^$', views.HomePageView.as_view()), # define una clase HomePageView
]
```

La clase de vista



- Este código importa las vistas de nuestra aplicación y espera que se defina la vista denominada `HomePageView`.
- Añadimos la clase en el archivo `views.py`:

```
# howdy/views.py
from django.shortcuts import render
from django.views.generic import TemplateView

# Create your views here.
class HomePageView(TemplateView):
    def get(self, request, **kwargs):
        return render(request, 'index.html', context=None)
```

La plantilla de vista

- Crea la carpeta templates en el directorio howdy

- mkdir templates

- Crea la plantilla index.html

```
<!-- howdy/templates/index.html -->
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Howdy!</title>
  </head>
  <body>
    <h1>Howdy! I am Learning Django!</h1>
  </body>
</html>
```

- Reinicia el servidor

- python manage.py runserver





Añadiendo más páginas

- Modifica la web para añadir una página de "acerca de nosotros"
- Asimismo añade un formulario que pida el nombre del usuario y lo envíe por post
 - Se debe mostrar el mensaje en la página de bienvenida
 - <https://docs.djangoproject.com/en/1.11/topics/forms/>



[...] netmind

WeKnowIT

Barcelona

C. Almogàvers, 123
08018 Barcelona
Tel. 93 304.17.20
Fax. 93 304.17.22

Madrid

Plaza Carlos Trias Bertrán, 7
28020 Madrid
Tel. 91 442.77.03
Fax. 91 442.77.07

www.netmind.es