Desarrollo Web con Django, HTML, CSS y Bootstrap

Oscar Daniel Acosta González
Licenciado en Matemáticas Aplicadas y Computación

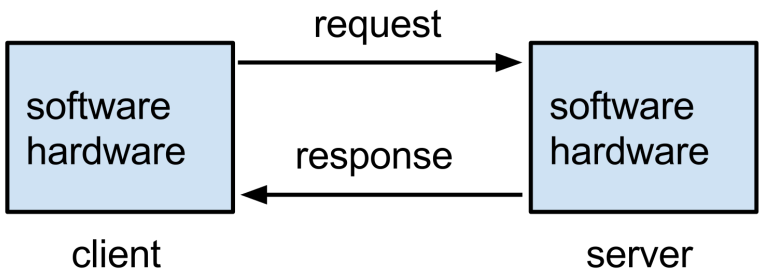
FES Acatlán - UNAM
26 de abril de 2018

Índice

1. Introducción

La llegada de Internet trajo consigo un mundo de posibilidades, ya que las aplicaciones que antes solo existían de forma *stand-alone*, ahora podrían verse en varios dispositivos a la vez, sin necesidad de instalar más programas más que un navegador web.

Para que esto sea posible, las máquinas se conectan mediante la arquitectura *cliente-servidor*, donde el servidor ofrece un servicio (como una página web) y el cliente recibe y se alimenta de éste.



Una página web se compone de dos partes fundamentales: *backend* y *frontend*, en donde la primera reside en el servidor y funciona como elemento lógico, mientras que el segundo opera del lado del cliente y se encarga de la presentación y el diseño. Para nuestro caso, Django funcionará como backend, mientras que el frontend correrá a cargo de HTML, CSS, Bootstrap.

2. HTML

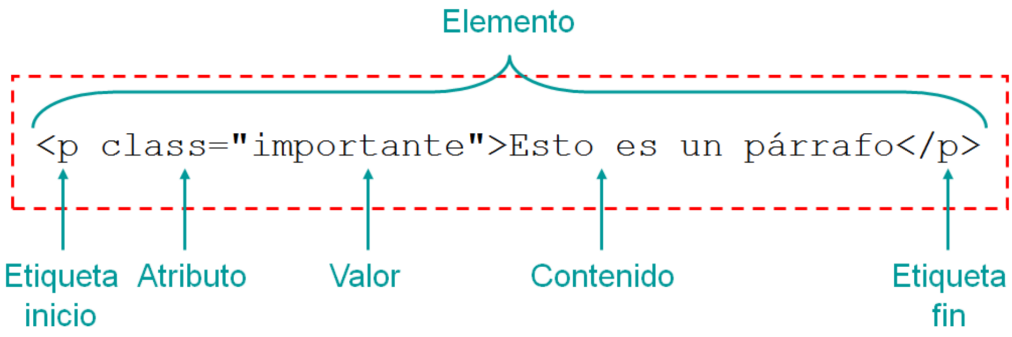
***H**yper**T**ext **M**arkup **L**anguage* (Lenguaje de marcado de hipertexto), es el código que estructura el contenido web, además de darle significado y propósito.

HTML **no es un lenguaje de programación, es un lenguaje de marcado**, ya que es usado para decirle al navegador como mostrar las páginas que se visitan, además de que por sí mismo no presenta una lógica formal de

programación como *Javascript, Python*, etc.

HTML consiste en una serie de elementos que para encerrar o envolver diferentes partes del contenido para hacerlos ver de cierta manera o actuar de cierto modo. Los *tags* o etiquetas sirven para mostrar texto, imágenes, botones, hipervínculos, etc.

La estructura base de un elemento de HTML es la siguiente:



Código ejemplo:

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8" />
        <title> Pagina de prueba </title>
    </head>
    <body>
        <img src="image/mi-foto.png" alt="Mi imagen de prueba">
    </body>
</html>
```

3. CSS

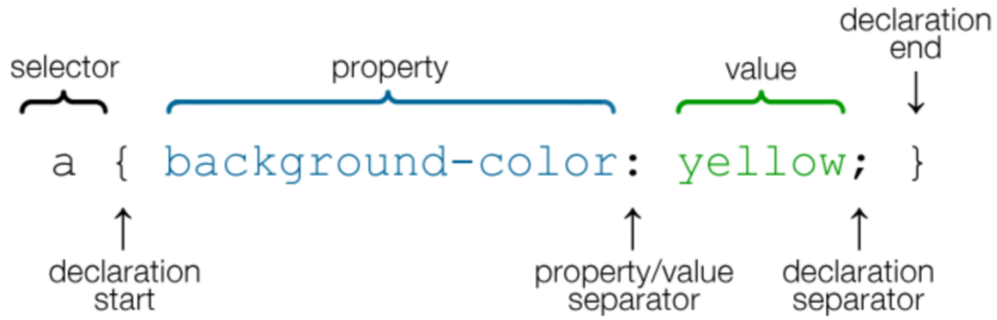
***C**ascading **S**tyle**S**heets* es el código que se utiliza para dar estilo al sitio web. Es usado para dar color, tamaño, etc a los elementos de HTML.

Tampoco es considerado un lenguaje de programación, pero tampoco es un lenguaje de marcado. Mas bien, es un lenguaje de hoja de estilo.

En HTML las hojas de estilo se llaman de la siguiente manera:

```
<link href = "mi_hoja_de_estilo.css" rel = "stylesheet"
    type = "text/css">
```

La estructura base de un elemento de CSS es la siguiente:



Código ejemplo:

```
p {
    color: red;
}

.mi-clase {
    background-color: green;
    font-size: 14px;
    width: 100%;
    height: 300px;
}
```

4. Bootstrap

Es un framework de código abierto para desarrollar HTML, CSS y Javascript de forma rápida. Antes de Bootstrap, se usaban varias librerías para el desarrollo de interfaces de usuario, las cuales llevaban a inconsistencias y a una gran carga de trabajo en su mantenimiento.

Bootstrap nace de la necesidad de unificar ciertos patrones de diseño, además de poner como prioridad el diseño móvil y responsivo, dando como resultado

un framework bastante eficiente y fácil de usar.

Fue creado por Twitter en 2011 y en 2012 se convirtió en el proyecto de desarrollo más popular de GitHub.

Para poder utilizarlo, se le llama como a una hoja de estilo, apuntando a los archivos, o bien a su sitio. Ejemplo:

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8" />
        <!-- Latest compiled and minified CSS -->
        <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com
            /bootstrap/3.3.5/css/bootstrap.min.css">
        <title> Pagina de prueba </title>
    </head>
    <body>
        <!-- Stack the columns on mobile by making one full-width
            and the other half-width -->
        <div class="row">
            <div class="col-xs-12 col-md-8">.col-xs-12 .col-md-8</div
                >
            <div class="col-xs-6 col-md-4">.col-xs-6 .col-md-4</div>
        </div>
    </body>
</html>
```

5. Django

Es un framework web de Python de alto nivel que alienta el desarrollo rápido y limpio. En la documentación de su sitio web podemos encontrar las siguientes características:

- Ridículamente veloz
- Tranquilizadoramente seguro
- Extremadamente escalable

Utiliza la filosofía *Modelo-Vista-Controlador*:

- Modelo: representación (interfaz) de los datos
- Vista: lo que el usuario ve
- Controlador: Controla el flujo de información entre el modelo y la vista

5.1. Apps y URL's

Un proyecto de Django se compone de diversas apps, donde una app es un módulo independiente del sitio web. Por ejemplo, en un sitio web como *Spotify*, una app sería *Mi perfil*, otra *Artistas*, etc. Para que Django pueda acceder a estas apps, es necesario darlas de alta, ya que de otro modo no podrá ser ejecutada por Python. Esta configuración se lleva a cabo en el archivo *settings.py*.

Para poder acceder a las apps instaladas (o dadas de alta), es necesario anexar su URL a la lista de ligas conocidas, solamente así Django sabrá como acceder y hacer uso de dicha aplicación. Para hacer esto, cada aplicación cuenta con su propio archivo de *urls.py* y el proyecto en general también tiene el suyo, a partir del cual es que se vincula todo.

La gran ventaja de este sistema, es que el sitio web completo se vuelve una especie de *sandbox*, donde únicamente es posible acceder al contenido previamente configurado y así se evitan intrusiones no deseadas.

5.2. Instalación

Para este caso se va a asumir que se está trabajando en un ambiente Ubuntu, al ser Python y Django desarrollos de código abierto, es sencillo encontrar sus implementaciones en distintas versiones de Sistemas Operativos basados en Linux. Si actualmente no se cuenta con Python, podemos instalarlo mediante el siguiente comando:

```
$ sudo apt install python2.7 python-pip # Instala Python y el
                                        # instalador de paquetes pip
$ pip install django # Instala Django mediante pip
```

Una vez hecho esto, podemos utilizar todas las ventajas del lenguaje de programación Python y las bondades del framework web Django.

6. Manos a la obra

Una vez que conocemos las herramientas que vamos a utilizar, es momento de poner en práctica los conocimientos. Teniendo instalados Python y Django en el ordenador, lo único que nos resta es iniciar el proyecto, para esto, nos movemos hacia el directorio en el que vamos a trabajar y desde linea de comandos ejecutamos lo siguiente:

```
$ django-admin startproject taller_django
```

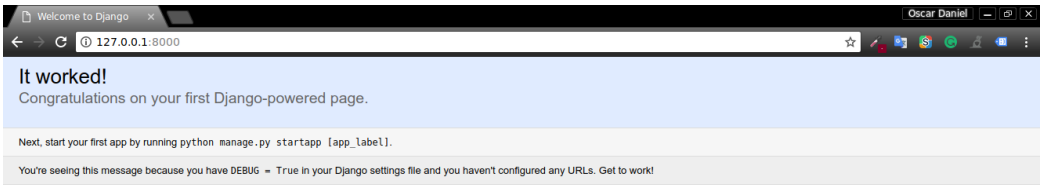
El comando anterior nos creará la siguiente estructura de archivos:

```
taller_django
|--- manage.py
|--- taller_django
    |--- __init__.py
    |--- settings.py
    |--- urls.py
    |--- wsgi.py
```

Para verificar que todo haya salido de forma correcta, nos cambiamos de directorio para estar al nivel del documento *manage.py*. Una vez allí, ejecutamos el siguiente comando:

```
$ python manage.py runserver # Echa a andar nuestro sitio
```

Una vez hecho esto, podemos observar en un navegador el *Hola mundo* de Django, para esto, es necesario abrir la url que django nos indica en la consola, en nuestro caso *http://127.0.0.1:8000/*. Si todo salió de forma correcta, observaremos lo siguiente:



¡Funciona! Sin embargo, este no es para nada un sitio web profesional, ni mucho menos informativo, con esto solamente logramos que Django corra una especia de mini servidor en nuestra máquina, la cual podemos acceder

mediante el navegador. Ahora es momento de crear nuestra primera aplicación. Para eso, en el directorio que estamos, corremos el siguiente comando:

```
$ python manage.py startapp app_ejemplo # Crea una app dentro del
                                        # proyecto
```

Con esto, nuestra estructura de archivos es la siguiente:

```
taller_django
|--- app_ejemplo
| |--- admin.py
| |--- apps.py
| |--- __init__.py
| |--- migrations
| | |--- __init__.py
| |--- models.py
| |--- tests.py
| |--- views.py
|--- db.sqlite3
|--- manage.py
|--- taller_django
  |--- __init__.py
  |--- __init__.pyc
  |--- settings.py
  |--- settings.pyc
  |--- urls.py
  |--- urls.pyc
  |--- wsgi.py
  |--- wsgi.pyc
```

Django creó para nosotros la infraestructura base de una app, así nosotros solo tenemos que preocuparnos por agregar los pequeños detalles, diseño y demás. Para que Django reconozca nuestra aplicación, es necesario dar de alta, para esto, nos dirigimos al archivo *settings.py*, y editamos la sección de apps instaladas de la siguiente manera:

```
INSTALLED_APPS = [
    'app_ejemplo',
    'django.contrib.admin',
    ...
]
```

Una vez dada de alta nuestra app, es necesario crear su *template* que es aquello que el usuario visualiza a través del navegador. Para esto, es necesario crear una carpeta de templates dentro de nuestra nueva app, y dentro de ella un documento HTML cualquiera, para nuestro caso, usaremos el siguiente:

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8" />
        <meta http-equiv="X-UA-Compatible" content="IE=edge">
        <title>Pagina ejemplo</title>
        <meta name="viewport" content="width=device-width, initial-
            scale=1">
        <!-- Latest compiled and minified CSS -->
        <script src = "https://code.jquery.com/jquery-3.3.1.min.js">
             </script>
        <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com
            /bootstrap/3.3.7/css/bootstrap.min.css" integrity="sha384
            -BVYiiSIFeK1dGmJRAkycuHAHRg32OmUcww7on3RYdg4Va+PmSTsz/
            K68vbdEjh4u"
            crossorigin="anonymous">
        <!-- Optional theme -->
        <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com
            /bootstrap/3.3.7/css/bootstrap-theme.min.css" integrity="
            sha384-
            rHyoN1iRsVXV4nD0JutlnGaslCJuC7uwjduW9SVrLvRYooPp2bWYgmgJQIXwl
            /Sp"
            crossorigin="anonymous">
        <!-- Latest compiled and minified JavaScript -->
        <script src="https://maxcdn.bootstrapcdn.com/bootstrap
            /3.3.7/js/bootstrap.min.js" integrity="sha384-
            Tc5IQib027qvyjSMfHjOMaLkfuWVxZxUPnCJA7l2mCWNIpG9mGCD8wGNIcPD7Txa
            "
            crossorigin="anonymous"></script>
    </head>
    <body>
        <h1>Hola mundo</h1>
        <div class = "container-fluid">
            <div class = "row">
                <div class = "col-xs-6 col-sm-4 col-md-3 col-lg-2 col
```

```
                -xl-1 bg-success" style = "height: 150px">
                Bloque 1
            </div>
            <div class = "col-xs-6 col-sm-4 col-md-3 col-lg-2 col
                -xl-1 bg-primary" style = "height: 150px">
                Bloque 2
            </div>
            <div class = "col-xs-6 col-sm-4 col-md-3 col-lg-2 col
                -xl-1 bg-info" style = "height: 150px">
                Bloque 3
            </div>
            <div class = "col-xs-6 col-sm-4 col-md-3 col-lg-2 col
                -xl-1 bg-warning" style = "height: 150px">
                Bloque 4
            </div>
        </div>
    </div>
</body>
</html>
```

Sin embargo, aún no le hemos dicho a Django como acceder y que hacer con esta página. Para ello, es necesario que creemos su *vista*, nos basta con editar el archivo *views.py* de nuestra app de la siguiente manera:

```
# -*- coding: utf-8 -*-
from __future__ import unicode_literals

from django.shortcuts import render


def index(request):
    path = "app_ejemplo/home.html"
    return render(request, path)
```

Ahora solo falta decirle a Django que hacer cuando un usuario entre a la sección de la app de nuestro proyecto, para este cometido es necesario que creemos un archivo de urls dentro de la app, el archivo contendrá lo siguiente:

```
from django.conf.urls import url
from .views import *
```

```
urlpatterns = [
    url(r'^$', index, name='index')
]
```

Finalmente, corresponde informarle al proyecto en general que existen esas urls y que tiene permitido visitarlas, para lograr esto, requerimos editar el archivo *urls.py* del proyecto en general (aquel que vive en la carpeta del mismo nombre del proyecto). El archivo debe verse de la siguiente manera:

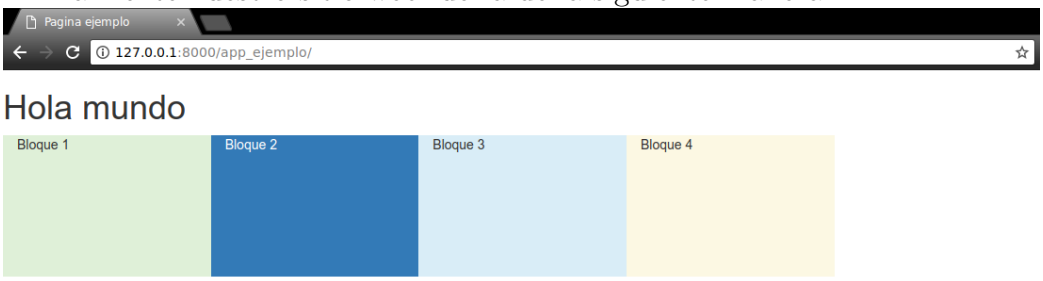
```
from django.conf.urls import url, include
from django.contrib import admin

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

Hasta este punto, nuestro proyecto tiene la siguiente estructura:

```
|--- app_ejemplo
| |--- admin.py
| |--- admin.pyc
| |--- apps.py
| |--- __init__.py
| |--- __init__.pyc
| |--- migrations
| | |--- __init__.py
| | |--- __init__.pyc
| |--- models.py
| |--- models.pyc
| |--- templates
| | |--- app_ejemplo
| | |--- home.html
| |--- tests.py
| |--- urls.py
| |--- urls.pyc
| |--- views.py
| |--- views.pyc
|--- db.sqlite3
|--- manage.py
|--- taller_django
```

```
    |--- __init__.py
    |--- __init__.pyc
    |--- settings.py
    |--- settings.pyc
    |--- urls.py
    |--- urls.pyc
    |--- wsgi.py
    |--- wsgi.pyc
```

Y finalmente nuestro sitio web lucirá de la siguiente manera:



Y en versión móvil: