



UTN.BA
UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL BUENOS AIRES

**Centro de
e-Learning**

UNIDAD DIDÁCTICA II

APLICACIÓN POO II

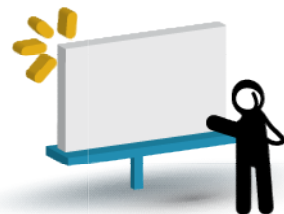
Centro de e-Learning SCEU UTN - BA.

Medrano 951 2do piso (1179) // Tel. +54 11 4867 7589 / Fax +54 11 4032 0148

www.sceu.frba.utn.edu.ar/e-learning

Modulo II – Arquitectura de sitio

Unidad VI – Patrón MVC, primer página, git y desarrollo de frontend I.



Presentación:

Django es una plataforma de código abierto escrita en python que utiliza el patrón de desarrollo MVC (Modelo – Vista – Controlador). Fue desarrollado en origen para gestionar varias páginas orientadas a noticias de la World Company de Lawrence, Kansas, y fue liberada al público bajo una licencia BSD en julio de 2005; el framework fue nombrado en alusión al guitarrista de jazz gitano Django Reinhardt.



Objetivos:

Que los participantes:

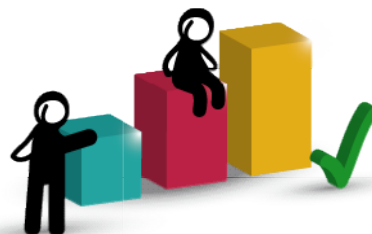
Puedan comenzar a trabajar en el desarrollo de una aplicación web.

Comprendan la forma adecuada de instalar y comenzar a utilizar Django.



Bloques temáticos:

- 1.- Asignación – Patrón MVC
- 2.- Instalación
- 3.- Print – Verificación
4. Primer proyecto
- 5.- Base de datos.
- 6.- Primera página.
- 7.- Git.



Consignas para el aprendizaje colaborativo

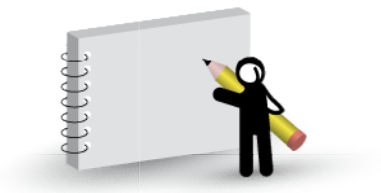
En esta Unidad los participantes se encontrarán con diferentes tipos de actividades que, en el marco de los fundamentos del MEC*, los referenciarán a tres comunidades de aprendizaje, que pondremos en funcionamiento en esta instancia de formación, a los efectos de aprovecharlas pedagógicamente:

- Los foros proactivos asociados a cada una de las unidades.
- La Web 2.0.
- Los contextos de desempeño de los participantes.

Es importante que todos los participantes realicen algunas de las actividades sugeridas y compartan en los foros los resultados obtenidos.

Además, también se propondrán reflexiones, notas especiales y vinculaciones a bibliografía y sitios web.

El carácter constructivista y colaborativo del MEC nos exige que todas las actividades realizadas por los participantes sean compartidas en los foros.



Tomen nota

Las actividades son opcionales y pueden realizarse en forma individual, pero siempre es deseable que se las realice en equipo, con la finalidad de estimular y favorecer el trabajo colaborativo y el aprendizaje entre pares. Tenga en cuenta que, si bien las actividades son opcionales, su realización es de vital importancia para el logro de los objetivos de aprendizaje de esta instancia de formación. Si su tiempo no le permite realizar todas las actividades, por lo menos realice alguna, es fundamental que lo haga. Si cada uno de los participantes realiza alguna, el foro, que es una instancia clave en este tipo de cursos, tendrá una actividad muy enriquecedora.

Asimismo, también tengan en cuenta cuando trabajen en la Web, que en ella hay de todo, cosas excelentes, muy buenas, buenas, regulares, malas y muy malas. Por eso, es necesario aplicar filtros críticos para que las investigaciones y búsquedas se encaminen a la excelencia. Si tienen dudas con alguno de los datos recolectados, no dejen de consultar al profesor-tutor. También aprovechen en el foro proactivo las opiniones de sus compañeros de curso y colegas.



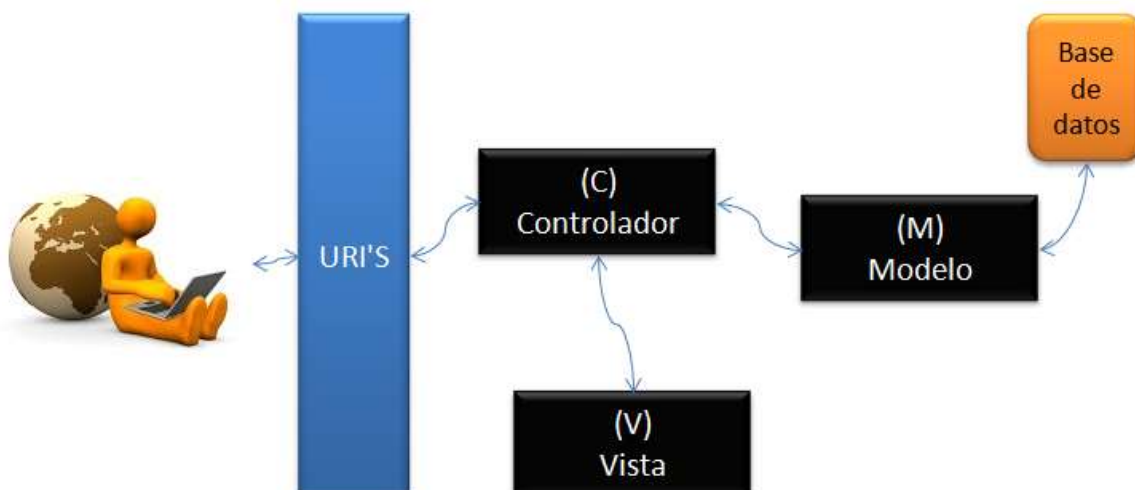
1. Asignación – Patrón MVC

Django utiliza el patrón TMV (Template – Model - View), el cual es análogo al modelo MVC (Modelo - Vista - Controlador) considerando que:

- Template equivale a Vista
- Model equivale a Modelo
- View equivale a Controlador

La clasificación anterior suele confundir un poco al inicio a quien está acostumbrado a utilizar el patrón MVC ya que los controladores secundarios son archivos llamados “views.py” que el que inicia a utilizar la plataforma tendería a pensar que son archivos de la Vista. Salvada esta aclaración comenzaremos a analizar el funcionamiento.

En este patrón el usuario se comunica con el sitio web, en donde toda consulta a través de la URL llega al controlador frontal, el cual tiene como única finalidad administrar el funcionamiento de la aplicación. El controlador le envía al “Modelo” la solicitud, y este en función a la consulta posee la lógica necesaria como para responder y de ser necesario conectarse con la base de datos correspondiente. Una vez que el “Modelo” le retorna la información al controlador, este aún no sabe como presentarle la información al usuario en pantalla, por lo que le envía la información a la vista, la cual le da formato a los datos y se los retorna al controlador, el cual ahora le puede retornar al usuario la página con los datos solicitados.





Nota: A medida que avancen las siguientes unidades, comenzaremos a comprender la forma en la cual Django lleva a la práctica este patrón de desarrollo y a reconocer cada uno de los componentes.

2. Instalación

Django es rápido de instalar y de actualizar si utilizamos la forma correcta y recomendada, de otra forma tendremos que previo a realizar la actualización de una versión desinstalar la versión previa e instalar Django nuevamente. Vamos a ver aquí únicamente la forma correcta de llevar a cabo la instalación.

2.1. Paso 1 – Abrir cmd

Asumiendo que estamos trabajando en Windows, debemos abrir una terminal:

2.2. Paso 2 – Instalar vía pip

Escribimos:

```
pip install Django==2.1.2
```

Esto hace que Django instale en este caso la versión 2.1.2 y de existir otra versión previa la desinstala antes de de la instalación.

```
Microsoft Windows [Versión 10.0.17134.285]
(c) 2018 Microsoft Corporation. Todos los derechos reservados.

C:\Users\juanb>pip install Django==2.1.2
Collecting Django==2.1.2
  Downloading https://files.pythonhosted.org/packages/32/ab/22530cc1b2114e6067eece94a333d6c749fa1c56a009f0721e51c181ea53/Django-2.1.2-py3-none-any.whl (7.3MB)
    100% |#####| 7.3MB 1.5MB/s
Collecting pytz (from Django==2.1.2)
  Downloading https://files.pythonhosted.org/packages/30/4e/27c34b62430286c6d59177a0842ed90dc789ce5d1ed740887653b898779a/pytz-2018.5-py2.py3-none-any.whl (510kB)
    100% |#####| 512kB ...
Installing collected packages: pytz, Django
Successfully installed Django-2.1.2 pytz-2018.5
```

3. Print - Verificación

Para verificar si Django se ha instalado correctamente podemos entrar al cmd y escribir:

```
python
```

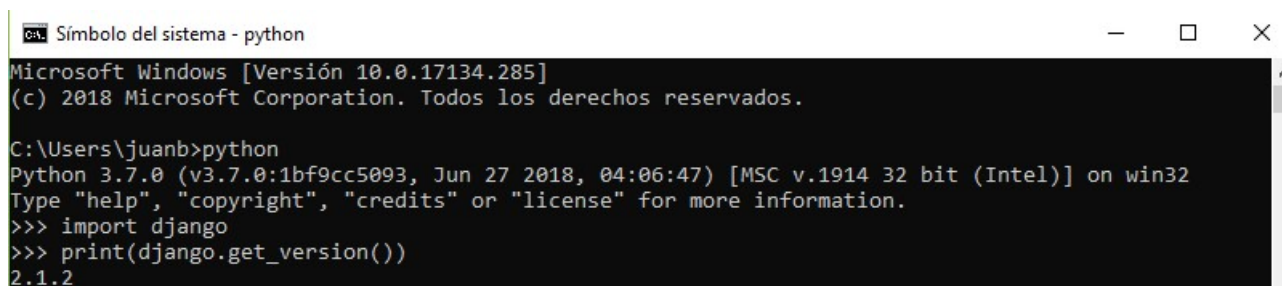
De esta forma en principio pasarán dos cosas, la primera es que se nos dará la versión de python que estamos utilizando actualmente, y en segundo lugar entraremos al editor de python a través del shell, en donde podemos importar el módulo de django como cualquier otro paquete de python

```
import django
```

Para ver si todo funciona correctamente imprimimos la versión actual de python, con la ayuda del método **get_version()**

```
print(django.get_version())
```

Si todo está bien tendría que mostrarnos la versión, en nuestro caso: 2.1.2.



```
Símbolo del sistema - python
Microsoft Windows [Versión 10.0.17134.285]
(c) 2018 Microsoft Corporation. Todos los derechos reservados.

C:\Users\juanb>python
Python 3.7.0 (v3.7.0:1bf9cc5093, Jun 27 2018, 04:06:47) [MSC v.1914 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> import django
>>> print(django.get_version())
2.1.2
```

Comprobar directorios instalados

Dentro del directorio site-packages de nuestra versión de python, se tienen que haber instalado dos directorios, “django” y “Django-2.1.2.dist-info”

```
C:/.../Python37/Lib/site-packages/django
```

```
C:/.../Python37/Lib/site-packages/Django-2.1.2.dist-info
```



< Python37 > Lib > site-packages		
	Nombre	Fecha de
	pycache	29/09/20
	django	06/10/20
	Django-2.1.2.dist-info	06/10/20

Instalación en Linux

Prerrequisitos

```
sudo apt-get install libapache2-mod-wsgi-py3
```

```
sudo apt-get install python3.4-dev
```

```
sudo apt-get install python3-setuptools
```

```
sudo apt-get install python3-pip
```

```
sudo pip3 install Django==1.11.10
```

Django se instala en

```
usr/local/lib/python3.4/dist-packages/django
```

```
usr/local/lib/python3.4/dist-packages/Django-1.11.10.dist-info
```

4. Primer proyecto

Vamos ahora a crear un primer proyecto de una venta online de libros “librosonline”

4.1. Crear proyecto

Paso 1: Entramos al cmd y desde el directorio en el cual vamos a trabajar escribimos:

```
django-admin startproject librosonline
```

Esto crea en el directorio que este parado, el siguiente sistema de directorios:

```
librosonline/  
    librosonline/  
        __init__.py  
        settings.py  
        urls.py  
        wsgi.py  
    manage.py
```

Estos archivos son:

__init__.py: Un archivo vacío que le dice a Python que este directorio debería ser considerado un paquete Python.

manage.py: Una utilidad de línea de comandos que permite interactuar de distintas formas con el proyecto.

settings.py: Archivo de configuración.

urls.py: Las URLs a definir en el sitio.

wsgi.py: Archivo para obtener compatibilidad con servidor apache.



4.2. Ejecutar servidor

Entramos al directorio “librosonline” desde el cmd:

```
cd librosonline
```

y ejecutamos:

```
python manage.py check
```

En el cmd nos sale si tenemos errores en los modelos

Y si ahora escribimos:

```
python manage.py
```

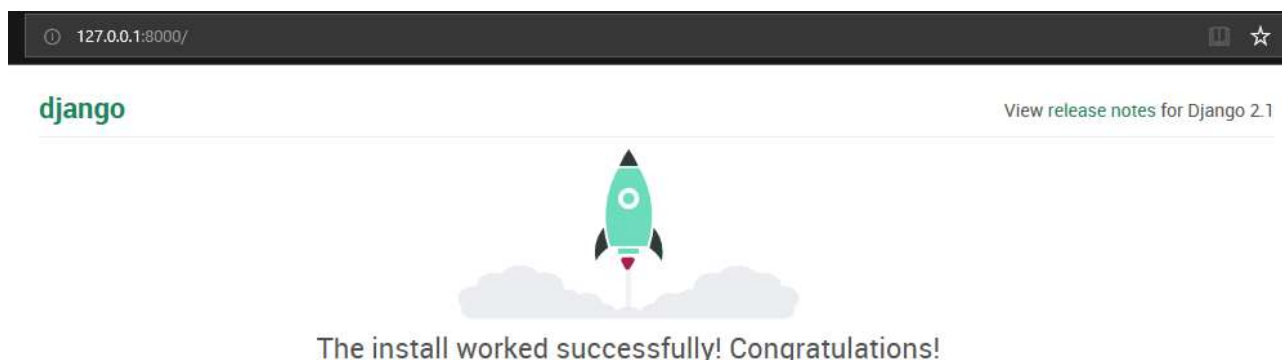
Nos muestra los diferentes tipos de comandos. Por otro lado si queremos lanzar el servidor, escribimos:

```
python manage.py runserver
```

Esto lanza el servidor en: <http://127.0.0.1:8000/>

Con lo cual si escribimos esto en el navegador accedemos a la página de bienvenida.

OJO: Al escribir “python manage.py” estamos asumiendo que o estamos trabajando en Windows o con python 2 en Linux, si quisiéramos trabajar con python 3 en Linux tendríamos que escribir “python3 manage.py”. Esto lo veremos en detalle al pasar el trabajo a producción más adelante. Ante cualquier duda por favor comparte tu duda en el foro.



Nota 1: 127.0.0.1 equivale a localhost, o sea el servidor local y 8000 es el puerto en donde encontramos la página web

Nota 2: Cuando buscamos páginas en internet, estamos asumiendo que el puerto es el 80, pero podríamos utilizar otros puertos fuera del 80 para transmitir información.

Nota 3: El servidor que estamos utilizando es un servidor de desarrollo que trae Django, el cual es útil para armar el sitio, pero tiene la desventaja de que no nos va a permitir tener un gran número de consultas simultaneas, por lo que en producción deberíamos utilizar un servidor como “apache”. Por ahora esto no es necesario, con lo cual seguiremos con el servidor por defecto.



5. Base de datos

Antes de comenzar a analizar la plataforma, debemos decidir con qué tipo de base de datos vamos a trabajar, Django nos permite trabajar con diferentes tipos de base de datos, de hecho por defecto trae un tipo de base de datos para proyectos de baja escala “sqlite3”, sin embargo vamos a trabajar con “MySQL” que en Windows para “python 3” no viene instalada por defecto y que vamos a tener que descargar un paquete de adaptación.

Nota: La decisión de utilizar “python 2” o “python 3” con Django no es una decisión menor, debido a que Python se encuentra en una etapa de transición similar a la que pasó PHP allá por el 2009 en el cambio de plataforma de la versión 4 a la 5, cuando introdujo la programación orientada a objetos. Python ha modificado temas de seguridad, los cuales se han mejorado considerablemente desde la versión 3.4, sin embargo la gran mayoría de plataformas y de paquetes están sufriendo una transición por lo que aún no se encuentran todos disponibles para la versión 3.X, esto es notorio en los paquetes que tienen que ver con aplicaciones móviles y videojuegos que ya tienen un tiempo en el mercado, no así con las nuevas plataformas y paquetes que van surgiendo, los cuales optan por la versión 3.X.

Sqlite3 en Windows

Administrador de base de datos: SQLite Studio

<https://sqlitestudio.pl/index.rvt?act=download>

Configuración de settings.py

settings.py

```
DATABASES = {  
    'default': {  
        'ENGINE': 'django.db.backends.sqlite3',  
        'NAME': os.path.join(BASE_DIR, 'db.sqlite3'),  
    }  
}
```

Ahora por línea de comandos ejecutamos:

python manage.py migrate

Centro de e-Learning SCEU UTN - BA.

Medrano 951 2do piso (1179) // Tel. +54 11 4867 7589 / Fax +54 11 4032 0148

www.sceu.frba.utn.edu.ar/e-learning

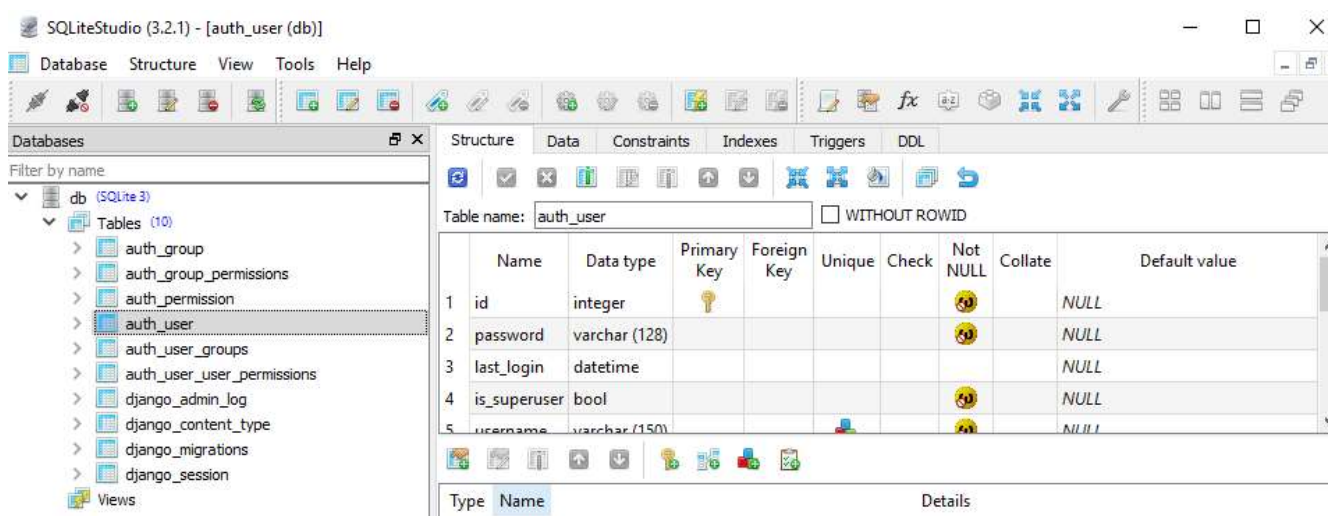


Para que Django migre dentro de la base de datos creada una serie de tablas que vienen por defecto para poder manejar por ejemplo el registro de usuarios. Esto lo hace según lo declarado en settings.py dentro de:

settings.py

```
INSTALLED_APPS = [  
    'django.contrib.admin',  
    'django.contrib.auth',  
    'django.contrib.contenttypes',  
    'django.contrib.sessions',  
    'django.contrib.messages',  
    'django.contrib.staticfiles',  
]
```

En el cmd tendríamos que ver este proceso, y al ir a la base de datos tendríamos que poder ver las tablas creadas:



6. Primer página - Home

Django pone toda la lógica de comunicación con la base de datos y diseño de nuestro sitio, dentro de paquetes llamados “Modelos” por lo que crearemos un primer modelo para representar el home de nuestro sitio, es decir la página de inicio.

6.1. Creamos Modelo

Asumiendo que estamos dentro de nuestro directorio de proyecto desde el cmd (si no es así ingresamos al directorio) escribimos:

django-admin startapp vistaprevia

Esto crea en nuestra aplicación un nuevo directorio (vistaprevia) con el siguiente contenido:

```
librosonline/  
    librosonline/  
        __init__.py  
        settings.py  
        urls.py  
wsgi.py  
vistaprevia/  
    migrations  
    __init__.py  
    admin.py  
    models.py  
    test.py  
    views.py  
    manage.py
```

Directorio “migrations”: directorio que registra los cambios producidos en nuestro modelo a ser migrados a la de datos.

admin.py: Archivo destinado a la lógica de la página del administrador del sitio

models.py: Archivo en el cual se crea la lógica de cada modelo.

test.py: Archivo para realizar test de funcionamiento de las aplicaciones.

Centro de e-Learning SCEU UTN - BA.

Medrano 951 2do piso (1179) // Tel. +54 11 4867 7589 / Fax +54 11 4032 0148
www.sceu.frba.utn.edu.ar/e-learning

views.py: Archivo en el cual se declaran las vistas de nuestro sitio.

6.2. – settings.py

El modelo creado, lo debemos declarar dentro del archivo settings.py para que Django pueda encontrarlo:

settings.py

```
INSTALLED_APPS = [  
    'django.contrib.admin',  
    'django.contrib.auth',  
    'django.contrib.contenttypes',  
    'django.contrib.sessions',  
    'django.contrib.messages',  
    'django.contrib.staticfiles',  
    'vistaprevia',  
]
```

6.3. – views.py

En el archivo “views.py” de cada modelo, vamos a crear funciones que van a tener la ruta a cada una de las vistas de nuestro sitio (páginas web), por defecto este archivo viene con el siguiente código:

```
from django.shortcuts import render
```

Y lo vamos a modificar para iniciar por el siguiente:

vistaprevia/views.py

```
from django.http import HttpResponse  
  
def index(request):  
    return HttpResponse("Hola Mundo!.")
```

De esta forma creamos una respuesta “HTTP RESPONSE”, que va a aparecer en la web al ingresar a la ruta que vamos a crear a continuación



6.4. – urls.py del Modelo

Dentro del directorio “vistaprevia” necesitamos crear un archivo “urls.py” en donde vamos a declarar las urls de cada página creada dentro de nuestro modelo:

```
librosonline/  
    librosonline/  
        __init__.py  
        settings.py  
        urls.py  
        wsgi.py  
vistaprevia/  
    migrations/  
        __init__.py  
        __init__.py  
        admin.py  
        models.py  
        test.py  
        views.py  
        urls.py  
manage.py
```

Para iniciar le agregamos el siguiente código:

vistaprevia/urls.py

```
from django.urls import path  
from vistaprevia import views  
  
urlpatterns = [  
    path("", views.index, name='index'),  
]
```

Notemos que lo que estamos realizando es importando el archivo views.py y dentro de urlpatterns apuntamos a la función “index”, indicando que cuando en la url escribamos:

<http://127.0.0.1:8000/...../>

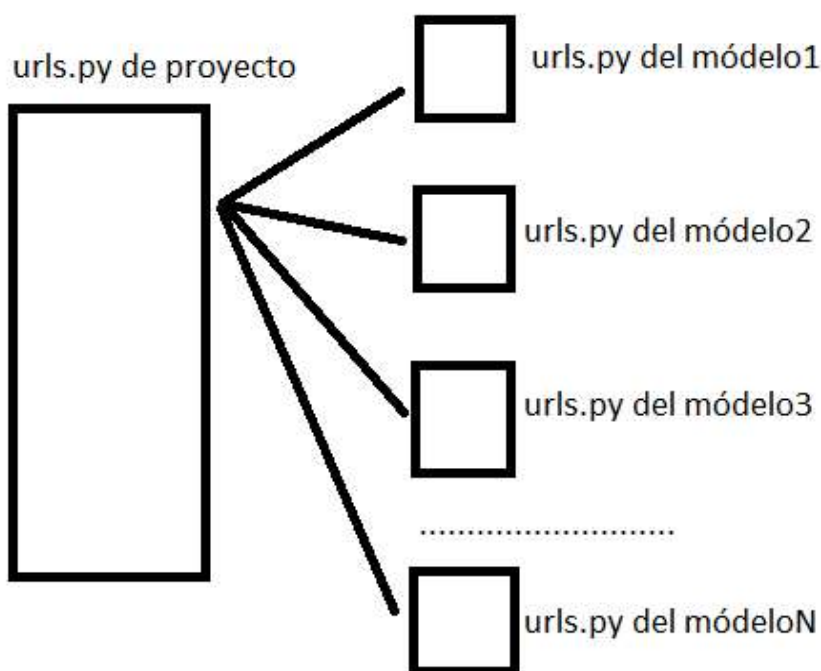


Vamos a ejecutar la función `index` del archivo `views.py` la cual nos va a permitir visualizar en pantalla la página de inicio que aún no hemos creado.

Nota 1: Luego vamos a entrar en detalles en cuanto a la nomenclatura de este archivo que utiliza expresiones regulares “Regex”

Nota 2: El archivo `urls.py` es un archivo local del modelo en el cual está definido, que guarda todas las urls de las páginas que correspondan a este modelo, sin embargo y dado que podemos tener muchos modelos, es necesario tener un `urls.py` general en donde se declare la ruta a cada `urls.py` de cada modelo para tener nuestro sitio bien organizado. Por este motivo en la url anterior he dejado unos puntos suspensivos previo al slash de `index` ya que aquí pondremos un nombre que identifique al paquete, en este caso voy a utilizar “`vistaprevia`”, esto debe de estar declarado en el `urls.py` general.

Una representación gráfica de como Django encuentra cada página se da en la siguiente representación:





6.5. – urls.py del proyecto

El archivo urls.py del proyecto ya se encuentra creado y es el que se encuentra dentro de en nuestro caso “librosonline/librosonline”

```
librosonline/  
    librosonline/  
        __init__.py  
        settings.py  
        urls.py  
        wsgi.py  
    vistaprevia/  
        migrations  
        __init__.py  
        __init__.py  
        admin.py  
        models.py  
        test.py  
        views.py  
        urls.py  
    manage.py
```

Por defecto, el contenido de este archivo es:

```
urls.py  
from django.contrib import admin  
from django.urls import path,  
  
urlpatterns = [  
    path('admin/', admin.site.urls),  
]
```

En donde podemos ver que ya existe una ruta “admin/” declarada que analizaremos más adelante, ahora crearemos la ruta a nuestro urls.py del modelo vistaprevia agregando:

```
urls.py  
from django.contrib import admin  
from django.urls import path, include  
  
urlpatterns = [  
    path('admin/', admin.site.urls),  
    path('vistaprevia/', include('vistaprevia.urls')),  
]
```



```
path('vistaprevia/', include('vistaprevia.urls')),  
path('admin/', admin.site.urls),
```

```
]
```

Cada ruta debe ir declarada dentro de la lista `urlpatterns`, en este caso estamos diciendo que vamos a poner como prefijo “vistaprevia/” cada vez que queramos acceder a una página declarada en el modelo “vistaprevia”, con lo cual la ruta a nuestra página index nos queda:

<http://127.0.0.1:8000/vistaprevia/>

Vista web:



Hola Mundo!.



7. GIT - Básico

Sin que nos hayamos dado cuenta nos ha surgido un problema muy grande que debemos resolver antes de continuar, para lo cual deberíamos realizarnos las siguientes preguntas.

¿Qué paso con el archivo previo a la modificación?

¿Qué pasaría si realizamos modificaciones a un archivo y queremos deshacer los cambios?

La respuesta seguramente sería que iríamos realizando copias de las versiones para ir registrando las modificaciones. Esto nos genera muchos problemas en el mantenimiento del código ya que al finalizar el curso tendríamos los directorios:

librosonline-1,

librosonline-2,

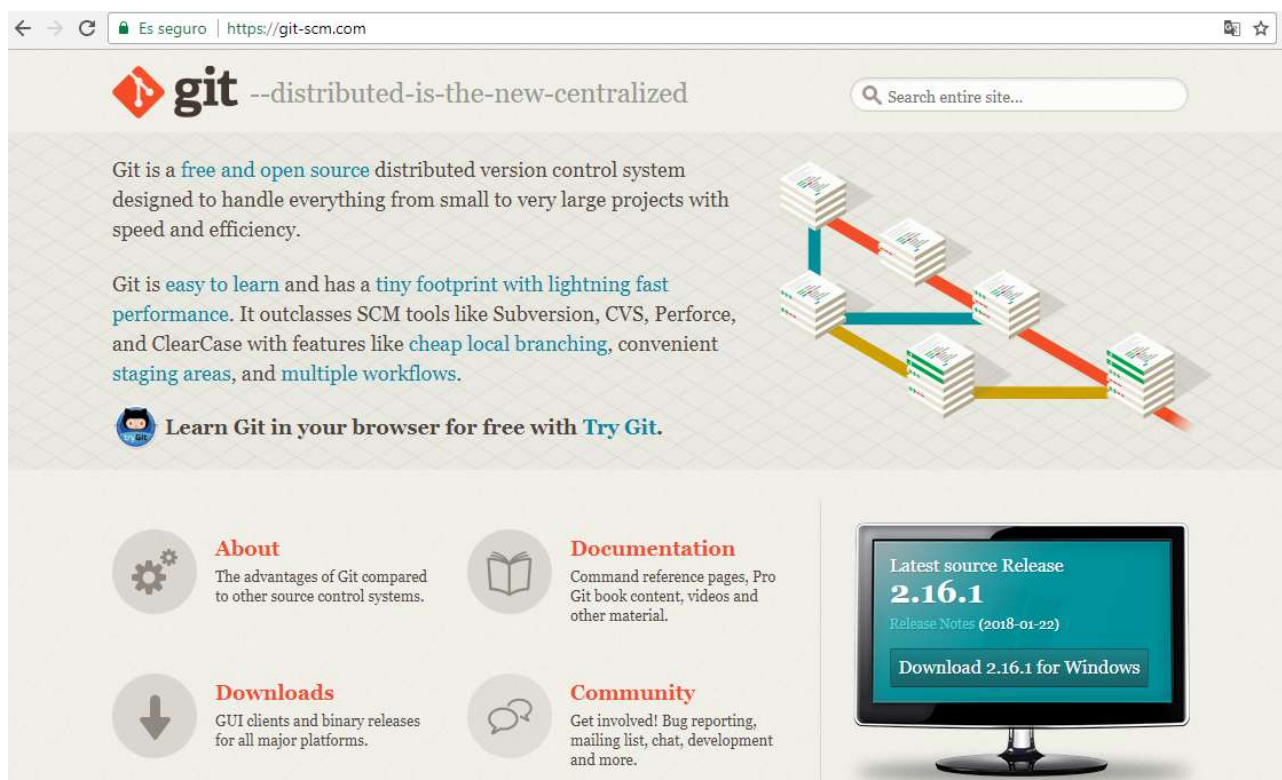
.....

librosonline-N,

etc.

Para evitarnos este problema vamos a trabajar con un controlador de versiones como Git el cual podemos descargar de:

<https://git-scm.com/>



Forma de uso:

El uso de git es relativamente sencillo, como primer paso procedo a instalar git presionando en siguiente hasta finalizar. Una vez que se ha completado la instalación podemos dirigirnos al buscador e iniciar el CMD de git.



PASO 1 - Iniciamos el control con git

Como primer paso nos paramos en el directorio en el cual queremos realizar un control de las versiones, para esto desde el cmd escribo:

```
cd /
```

para ir al directorio raíz, en mi caso el disco C.

Luego busco la ruta al directorio librosonline:

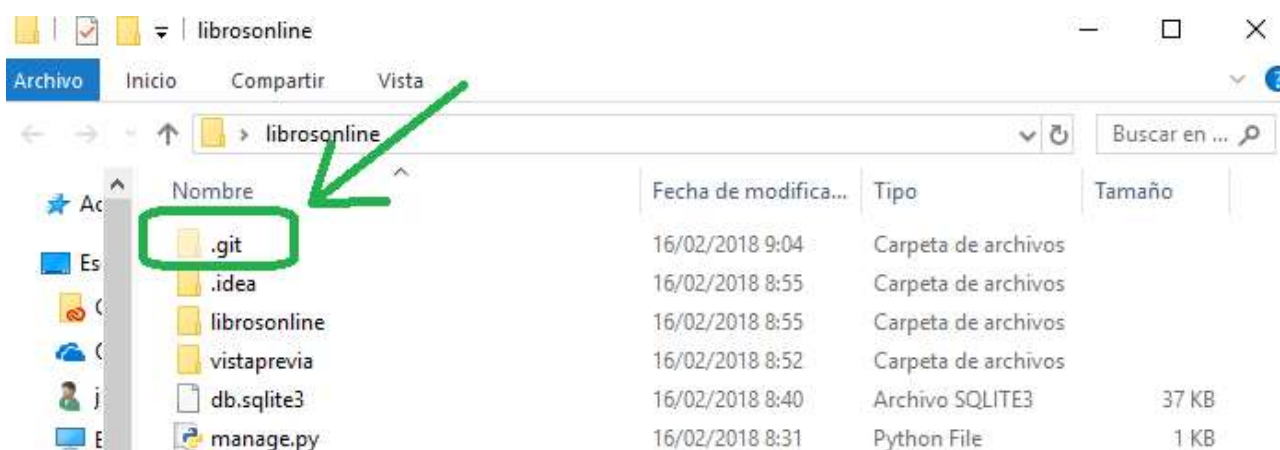
```
cd Users/juan/Desktop/librosonline
```

La ruta anterior depende de donde tenga cada uno la carpeta librosonline.

Lo único que nos resta realizar para que git comience a realizar un seguimiento de las versiones es ejecutar:

```
git init
```

Si ahora observamos nuestro directorio podemos ver como git a creado un directorio ".git" desde el cual realizará el control de las versiones.





A continuación le indicamos a git quienes somos agregando un mail y un nombre con los siguientes códigos:

```
git config --global user.email "tu email"  
git config --global user.name "tu nombre"
```

PASO 2 – Chequeamos el Status

Si ejecutamos ahora:

```
git status
```

Podemos ver cuál es el estado de nuestras modificaciones en donde git nos informa que aún no hemos guardado las modificaciones.

```
Git CMD  
C:\Users\juan>cd desktop  
C:\Users\juan\Desktop>cd librosonline  
C:\Users\juan\Desktop\librosonline>git init  
Initialized empty Git repository in C:/Users/juan/Desktop/librosonline/.git/  
C:\Users\juan\Desktop\librosonline>git status  
On branch master  
  
No commits yet  
  
Untracked files:  
  (use "git add <file>..." to include in what will be committed)  
  
    .idea/  
    db.sqlite3  
    librosonline/  
    manage.py  
    vistaprevia/  
  
nothing added to commit but untracked files present (use "git add" to track)  
C:\Users\juan\Desktop\librosonline>
```

Este comando podemos ejecutarlo cada vez que queramos ver los cambios previos a guardarlos.

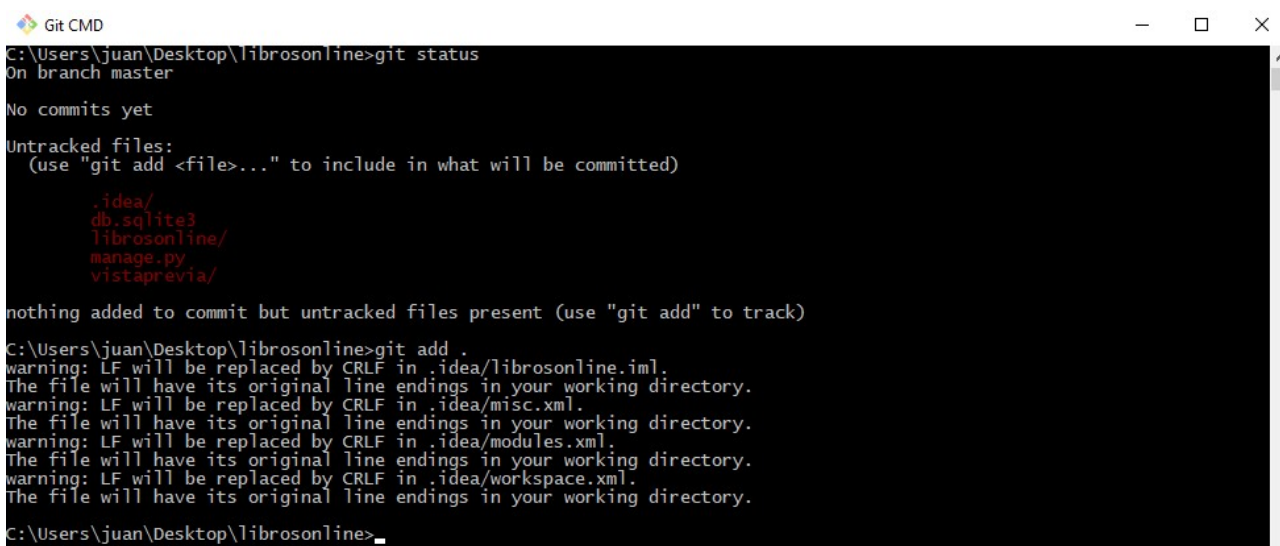


PASO 3 – Indicamos archivos a realizar seguimiento

Para realizar el seguimiento de algún archivo o directorio podemos utilizar el comando “git add” seguido del nombre de archivo o agregando un punto para que haga el seguimiento de todo:

git add .

En el cmd podremos ver un listado de todos los archivos a los cuales les vamos a realizar el seguimiento



```
Git CMD
C:\Users\juan\Desktop\librosonline>git status
On branch master

No commits yet

Untracked files:
  (use "git add <file>..." to include in what will be committed)

        .idea/
        db.sqlite3
        librosonline/
        manage.py
        vistaprevia/

nothing added to commit but untracked files present (use "git add" to track)

C:\Users\juan\Desktop\librosonline>git add .
warning: LF will be replaced by CRLF in .idea/librosonline.iml.
The file will have its original line endings in your working directory.
warning: LF will be replaced by CRLF in .idea/misc.xml.
The file will have its original line endings in your working directory.
warning: LF will be replaced by CRLF in .idea/modules.xml.
The file will have its original line endings in your working directory.
warning: LF will be replaced by CRLF in .idea/workspace.xml.
The file will have its original line endings in your working directory.

C:\Users\juan\Desktop\librosonline>
```



Si ahora ejecutamos un “git status” podemos ver cómo nos aparecen todos los archivos a los cuales se les va a realizar un seguimiento ya listados.

```
Git CMD
warning: LF will be replaced by CRLF in .idea/workspace.xml.
The file will have its original line endings in your working directory.

C:\Users\juan\Desktop\librosonline>git status
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)

    new file:   .idea/librosonline.iml
    new file:   .idea/misc.xml
    new file:   .idea/modules.xml
    new file:   .idea/workspace.xml
    new file:   db.sqlite3
    new file:   librosonline/__init__.py
    new file:   librosonline/__pycache__/__init__.cpython-34.pyc
    new file:   librosonline/__pycache__/settings.cpython-34.pyc
    new file:   librosonline/__pycache__/urls.cpython-34.pyc
    new file:   librosonline/__pycache__/wsgi.cpython-34.pyc
    new file:   librosonline/settings.py
    new file:   librosonline/urls.py
    new file:   librosonline/wsgi.py
    new file:   manage.py
    new file:   vistaprevia/__init__.py
    new file:   vistaprevia/__pycache__/__init__.cpython-34.pyc
    new file:   vistaprevia/__pycache__/admin.cpython-34.pyc
    new file:   vistaprevia/__pycache__/models.cpython-34.pyc
    new file:   vistaprevia/__pycache__/urls.cpython-34.pyc
    new file:   vistaprevia/__pycache__/views.cpython-34.pyc
    new file:   vistaprevia/admin.py
    new file:   vistaprevia/apps.py
    new file:   vistaprevia/migrations/__init__.py
    new file:   vistaprevia/migrations/__pycache__/__init__.cpython-34.pyc
    new file:   vistaprevia/models.py
    new file:   vistaprevia/tests.py
    new file:   vistaprevia/urls.py
    new file:   vistaprevia/views.py

C:\Users\juan\Desktop\librosonline>
```



PASO 4 – Realizamos el seguimiento

Ahora para realizar el seguimiento agregamos el comando (`git commit -m "mensaje"`) indicando entre comillas el mensaje que va a identificar la modificación realizada.

`git commit -m "UNIDAD 1: PUNTO 7 - Inicio de seguimiento"`

Si lo ejecutamos y luego chequeamos el status veremos que nos indica que ya no hay nada que guardar.

```
C:\Users\juan\Desktop\librosonline>git commit -m "Unidad 1: Inicio de seguimiento"
[master (root-commit) 70891b4] Unidad 1: Inicio de seguimiento
28 files changed, 504 insertions(+)
create mode 100644 .idea/librosonline.iml
create mode 100644 .idea/misc.xml
create mode 100644 .idea/modules.xml
create mode 100644 .idea/workspace.xml
create mode 100644 db.sqlite3
create mode 100644 librosonline/__init__.py
create mode 100644 librosonline/__pycache__/__init__.cpython-34.pyc
create mode 100644 librosonline/__pycache__/settings.cpython-34.pyc
create mode 100644 librosonline/__pycache__/urls.cpython-34.pyc
create mode 100644 librosonline/__pycache__/wsgi.cpython-34.pyc
create mode 100644 librosonline/settings.py
create mode 100644 librosonline/urls.py
create mode 100644 librosonline/wsgi.py
create mode 100644 manage.py
create mode 100644 vistaprevia/__init__.py
create mode 100644 vistaprevia/__pycache__/__init__.cpython-34.pyc
create mode 100644 vistaprevia/__pycache__/admin.cpython-34.pyc
create mode 100644 vistaprevia/__pycache__/models.cpython-34.pyc
create mode 100644 vistaprevia/__pycache__/urls.cpython-34.pyc
create mode 100644 vistaprevia/__pycache__/views.cpython-34.pyc
create mode 100644 vistaprevia/admin.py
create mode 100644 vistaprevia/apps.py
create mode 100644 vistaprevia/migrations/__init__.py
create mode 100644 vistaprevia/migrations/__pycache__/__init__.cpython-34.pyc
create mode 100644 vistaprevia/models.py
create mode 100644 vistaprevia/tests.py
create mode 100644 vistaprevia/urls.py
create mode 100644 vistaprevia/views.py
```

PASO 5 – Datos de commit realizado

Finalmente podemos realizar un chequeo de lo realizado mediante el comando:

git log

```
C:\Users\juan\Desktop\librosonline>git log
commit 65bb4f00ba82337a6501e113ec0bf1acff62e770 (HEAD -> master)
Author: juan barreto <juanbarretor@gmail.com>
Date: Fri Feb 16 09:14:44 2018 -0300
```

Unidad 1: PUNTO 7 - Inicio de seguimiento

PASO 6 – Regresar a un punto de la rama principal

Para recuperar los archivos como los teníamos en un punto de commit debemos indicar el hash del commit correspondiente como sigue:

git checkout numero-de-hash

En nuestro caso para recuperar el ejercicio realizado sería:

git checkout 65bb

Notar que no es necesario pasar el número completo, alcanza con pasar los primeros números.



8. Templates.

Para crear vistas en Django utilizamos el lenguaje HTML para realizar la maquetación, css y css3 para agregarle estilos, y javascript para adicionarle interacciones. Django posee una lógica extraordinaria para crear sitios escalables pero que suele resultar un poco compleja de asimilar al inicio. Para agregar vistas html debemos crear un directorio llamada template dentro de cada app y adicionar un directorio extra con el nombre de la app, en los pasos siguientes seguiremos este proceso paso a paso.

8.1. – Templates: Creamos una vista un poco más compleja

Modificamos el contenido de view.py por:

views.py

```
from django.shortcuts import render
from django.http import HttpResponse
from django.template import RequestContext, loader

def index(request):
    contenido = { 'nombre_sitio': 'LibrosOnline' }
    return render(request, 'vistaprevia/index.html', contenido)
```

Aquí hemos creado la función “index” la cual va a contener la ruta a la página “index.html” (la cual aún no hemos creado) y que se encuentra dentro del directorio “vistaprevia” el cual aún tampoco hemos creado. Cada vez que queramos agregar una nueva página a nuestro sitio, que posea la lógica de comunicación con la base de datos que estamos guardando dentro de la “app” vistaprevia, debemos crear una función que registre la ruta a la misma.

Dentro de la función estamos declarando un diccionario “contenido” que en este caso solo tiene almacenado un registro que vamos a utilizar para pasarle a la página de inicio el nombre de nuestro sitio web, esto lo realizamos en la línea:

views.py

```
-----
return render(request, 'vistaprevia/index.html', contenido)
```




En donde estamos indicando mediante render() que contenido pasamos a que página web.

Debemos señalar varios puntos importantes:

- 1.- Las vistas van dentro de directorios templates dentro de cada modelo.
- 2.- En el archivo settings.py necesitamos agregar la siguiente línea para que Django pueda encontrar un template dentro de cualquier directorio “templates” que tengamos en el sitio:

settings.py

```
TEMPLATES = [
    {
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
        'DIRS': [os.path.join(BASE_DIR, 'templates')],
        'APP_DIRS': True,
        'OPTIONS': {
            'context_processors': [
                'django.template.context_processors.debug',
                'django.template.context_processors.request',
                'django.contrib.auth.context_processors.auth',
                'django.contrib.messages.context_processors.messages',
            ],
        },
    ],
]
```

No confundir con versiones anteriores como la 1.8.2 la forma de declarar los templates era la siguiente:

TEMPLATE_DIRS = [os.path.join(BASE_DIR, 'templates')]

- 2.- Dado que Django busca dentro de todos los directorios templates es necesario crear dentro de cada uno un directorio con el nombre de la “app” en este caso “vistaprevia”.
- 3.- Dentro de este último directorio van las vistas, en este caso index.html que es el archivo al cual estamos apuntando desde views.py.



Con lo indicado hasta aquí ahora nuestro sitio se ve así:

```
librosonline/  
librosonline/  
    __init__.py  
    settings.py  
    urls.py  
    wsgi.py  
vistaprevia/  
    migrations  
    __init__.py  
    templates  
        vistaprevia  
            index.html  
admin.py  
models.py  
test.py  
views.py  
urls.py  
manage.py
```

Nota: html es un lenguaje de etiquetas, en donde cada etiqueta representa algo, por ejemplo:

<h2> = subtítulo

<header> = cabecera de contenido

<nav> = caja para el menú de navegación

<footer> = pie de página

Todo documento html tiene una etiqueta raíz <html>, una declaración de tipo de documento <!DOCTYPE html>, una cabecera links <head> y un cuerpo de documento <body>. De esta forma una plantilla básica de html5 se vería como sigue:



index.html

```
<!DOCTYPE html>
<html lang="es">
  <head>
    -----
  </head>
  <body>
    -----
  </body>
</html>
```

Si consideramos el siguiente contenido para nuestro index.html

index.html

```
<!Doctype html>
<html lang="es">
  <head>
    <meta charset="utf-8">
    <title>Patrón mvc</title>
    <meta name="viewport" content="width=device-width, initial-scale=1">
  </head>
  <body>
    <h2>Bienvenido a : {{ nombre_sitio }}.</h2>
  </body>
</html>
```

Notemos que el diccionario lo pasamos de view.py a index.html mediante el uso de **{{ nombreDeClave }}**



Bienvenido a : LibrosOnline.!



Bibliografía utilizada y sugerida

Libros

Arun Ravindram - Django Design Patterns and Best Practices - Packt Publishing - 2015

Sanjeev Jaiswal and Ratan Kumar - Learning Django Web Development - Packt Publishing - 2015

Manual online

<https://www.djangoproject.com/>

<https://git-scm.com/>



Lo que vimos

En esta unidad dimos los primeros pasos en la comprensión de la arquitectura de la plataforma Django.



Lo que viene:

En la siguiente unidad comenzaremos a trabajar sobre el desarrollo de frontend.