

INSTITUTO
TECNOLOGICO
SUPERIOR
GUAYAQUIL
NOMBRE DEL
ESTUDIANTE:
CARLA BARRIGA
RUGEL
CURSO: 3G
2021-2022

¿Qué es Django?.....	1
¿Qué es la máquina virtual en Django?.....	4
¿Qué es MVT en Django?.....	4
¿Qué es la Carpeta Templates?.....	5
¿Qué es la Carpeta static?.....	6
Que es un modelo en Django.....	7
Como se llaman a los CSS desde el archivo base HTML.....	8-9
Como consume un archivo hijo HTML al utilizar la herencia del archivo base HTML.....	8-9
Crear un archivo base HTML en la APPS core.....	10
Crear un view que llame al HTML hijo.....	10
Crear la urls que llame al views.....	11
Crear las tablas del sistema de usuarios para utilizar el panel de administración.....	11
Crear un usuario para poder ingresar al Panel de Administración.....	12
Crear un modelo en Django.....	12
Migrar el Modelo a la base del Panel de Administración.....	13
Integrar el Modelo al Panel de Administración.....	13
Ingresar información al modelo por el Panel de Administración	14
Realizar la consulta de todo lo ingresado en el modelo desde el views.....	14
Mostrar los datos guardados en el modelo al HTML hijo.....	15
Crear un proyecto con la máquina virtual.....	15
Descargar los instaladores de Django al proyecto.....	16

¿Qué es Django?

Django es un framework web de alto nivel que permite el desarrollo rápido de sitios web seguros y mantenibles. Desarrollado por programadores experimentados, Django se encarga de gran parte de las complicaciones del desarrollo web, por lo que puedes concentrarte en escribir tu aplicación sin necesidad de reinventar la rueda. Es gratuito y de código abierto, tiene una comunidad próspera y activa, una gran documentación y muchas opciones de soporte gratuito y de pago.

Cuando construyes un sitio web, siempre necesitas un conjunto de componentes similares: una manera de manejar la autenticación de usuarios (registrarse, iniciar sesión, cerrar sesión), un panel de administración para tu sitio web, formularios, una forma de subir archivos, etc.

Por suerte para nosotros, hace tiempo que otros desarrolladores se dieron cuenta de que siempre se enfrentaban a los mismos problemas cuando construían sitios web, y por eso se unieron y crearon frameworks (Django es uno de ellos) con componentes listos para usarse.

Los frameworks sirven para que no tengamos que reinventar la rueda cada vez y que podamos avanzar más rápido al construir un nuevo sitio.

¿Por qué necesitas un framework?

Para entender para qué sirve realmente Django, necesitamos fijarnos en cómo funcionan los servidores. Lo primero es que el servidor necesita enterarse de que tú quieres que te sirva una página web.

Imagina un buzón (puerto) en el que alguien está constantemente mirando si hay cartas entrantes (peticiones). Esto es lo que hace un servidor web. El servidor web lee la carta, y envía una respuesta con la página web. Pero para enviar algo, tenemos que tener algún contenido. Y Django nos ayuda a crear ese contenido.

¿Qué es la máquina virtual en Django?

El entorno de desarrollo es una instalación de Django en tu computadora local que puedes usar para desarrollar y probar apps Django antes de desplegarlas al entorno de producción.

¿Qué es MVT en Django?

Modelo: el modelo actuará como la interfaz de sus datos. Es responsable de mantener los datos. Es la estructura de datos lógica detrás de toda la aplicación y está representada por una base de datos (generalmente bases de datos relacionales como MySQL, Postgres). Para ver más, visite - Django Models

Vista: La vista es la interfaz de usuario, lo que ve en su navegador cuando representa un sitio web. Está representado por archivos HTML / CSS / Javascript y Jinja. Para ver más, visite - Vistas de Django .

Plantilla: una plantilla consta de partes estáticas de la salida HTML deseada, así como una sintaxis especial que describe cómo se insertará el contenido dinámico. Para ver más, visite - Plantillas Django

Estructura del proyecto:

Un proyecto Django cuando se inicializa contiene archivos básicos por defecto como `manage.py`, `view.py`, etc. Una estructura de proyecto simple es suficiente para crear una aplicación de una sola página.

`_init_.py`: es un paquete de Python.

`settings.py`: como su nombre lo indica, contiene todas las configuraciones del sitio web. En este archivo registramos las aplicaciones que creamos, la ubicación de nuestros archivos estáticos, los detalles de configuración de la base de datos, etc.

`urls.py`: en este archivo almacenamos todos los enlaces del proyecto y las funciones a llamar.

`wsgi.py`: este archivo se utiliza para implementar el proyecto en WSGI. Se utiliza para ayudar a su aplicación Django a comunicarse con el servidor web.

¿Qué es la Carpeta Templates?

Plantilla. Una plantilla es un archivo de texto que determina la estructura o diseño de un archivo (como una página HTML), con marcadores usados para representar el contenido real. Django automáticamente buscará plantillas en un directorio llamado 'templates' de su aplicación.

Las siglas en inglés MTV corresponden a Model (Modelo), Template (Plantilla) y View (Vista). En este post me referiré a las plantillas como templates, ya que su uso en el español es bastante común.

En la práctica el patrón MTV es muy similar al MVC a tal punto que se puede decir que Django es un framework MVC. Realmente este no se desvió demasiado del patrón Modelo Vista Controlador, simplemente lo implementa de una manera distinta y para evitar confusiones es llamado MTV.

Modelo Vista Controlador (MVC)

Modelo: Es el que se encarga de manipular la información de la aplicación, la cual usualmente está almacenada en la base de datos.

Vista: Decide qué información mostrar y cómo mostrarla.

Controlador: Es quien responde a las peticiones, decide que vista usar y si es necesaria información accede al modelo.

Model Template Vista (MTV)

En Django, el controlador sigue estando presente, nada más que de una manera intrínseca, ya que todo el framework Django es el controlador.

Modelo: Maneja todo lo relacionado con la información, esto incluye como acceder a esta, la validación, relación entre los datos y su comportamiento.

Vista: Es un enlace entre el modelo y el template. Decide qué información será mostrada y por cual template.

Template: Decide cómo será mostrada la información.

Al momento de hacer clic en un enlace o escribir una dirección a lo primero que se accede es al mapa de URLs (también conocido como URL map o URL conf), en este archivo cada ruta está asociado con una view, si se necesita algún dato se solicitara este a model el cual a su vez generara la consulta a la base de datos, cuando los datos han sido traídos estos son enviados al template que contiene la lógica de presentación para estos. Luego de "pintar" la página esta se envió al navegador que hizo la solicitud.

¿Qué es la Carpeta static?

Muchos de los desarrolladores que trabajan con Django se quedan sorprendidos con el funcionamiento de las vistas y plantillas, pero eso no es todo, también se deben tener en cuenta las otras partes de una aplicación: como las imágenes, las hojas de estilo, Javascript y otros elementos. Estas partes se les conocen en general como el contenido estático.

Cuando se tienen proyectos pequeños, no es mucho el trabajo al respecto, se pueden incluir este tipo de contenido en las plantillas sin ningún problema. Sin embargo cuando el proyecto deja de ser pequeño, y empieza a tener muchas partes, lidiar con este tipo de contenido puede ser un dolor de cabeza.

Para evitar jaquecas innecesarias Django mediante: `django.contrib.staticfiles`, gestiona el contenido estático para las aplicaciones, y los ordena en una sola ubicación fácil de referenciar y de usar.

Cambios en Settings.py

El primer lugar donde inicia el manejo de los archivos estáticos reside en el archivo de configuraciones del proyecto: `settings.py`, en este archivo tenemos líneas exclusivamente dedicadas al manejo del contenido estático.

En este archivo existen 4 elementos: `STATIC_ROOT`, `STATIC_URL`, `STATICFILES_DIRS`, `STATICFILES_FINDERS` cada uno de ellos con un propósito documentado en el mismo archivo `settings.py` a modo de comentario.

Para empezar con nuestro ejemplo del capítulo, debemos prestar atención a `STATICFILES_DIRS`, este elemento permite declarar la ruta, desde la cual se enlazará el contenido estático, lo dejamos de la siguiente manera:

```
STATICFILES_DIRS = (
    # Put strings here, like "/home/html/static" or "C:/www/django/static".
    # Always use forward slashes, even on Windows.
    # Don't forget to use absolute paths, not relative paths.
    os.path.join(RUTA_PROYECTO, 'static'),
)
```

El resto de elementos referentes al contenido estático, no se manipulan. Ya que estamos en la versión de desarrollo y aún falta para la etapa de producción. No olvidar guardar el archivo, para proseguir sin errores.

Que es un modelo en Django

Un modelo en Django es un tipo especial de objeto que se guarda en la base de datos. Una base de datos es una colección de datos. Es un lugar en el cual almacenarás la información sobre usuarios, tus entradas de blog, etc. Utilizaremos una base de datos SQLite para almacenar nuestros datos.

Un Django model generalmente se refiere a una tabla de la base de datos, los atributos de ese modelo se convierten en las columnas de esa tabla esos atributos reciben el nombre de django fields los cuales manejan automáticamente las conversiones de tipos de datos para la base de datos que estemos usando.

Una de las grandes características de Django es su ORM, gracias a el no tenemos que escribir ninguna consulta de base de datos, e incluso se recomienda NO escribir una al usar Django. ORM convierte sus Django models y todas las operaciones que realiza con las consultas de base de datos correspondientes. Esto significa que toda la manipulación se hará ahora con los objetos de Python creados a partir de ese modelo, y todo el material de la base de datos subyacente será cuidado por el ORM de Django.

El ORM de Django es compatible con todas las bases de datos principales como Postgres , MySQL , sqlite3 y otras bases de datos de empresas provistas con los controladores adecuados. Esto también significa que no tenemos que preocuparnos por la base de datos subyacente que estemos utilizando, o incluso si deseamos cambiar de una base de datos a otra, podemos hacerlo sin cambiar una sola línea de la lógica de su aplicación, solo cambiamos la cadena de la base de datos en el archivo de configuración settings.py.

Ejemplo de modelo Django

Un ejemplo simple sería para una aplicación de administración de bibliotecas, tendríamos 2 modelos: student y book

Por lo que en el fichero models.py pondríamos:

```
from django.db import models

class student(models.Model):
    roll_no = models.IntegerField(primary_key=True)
    first_name = models.CharField(max_length=30)
    last_name = models.CharField(max_length=30)
```

Como se llaman a los CSS desde el archivo base HTML.

Originalmente, las páginas HTML sólo incluían información sobre sus contenidos de texto e imágenes. Con el desarrollo del estándar HTML, empezaron a incluir también información sobre el aspecto de sus contenidos: tipos de letra, colores y márgenes.

La posterior aparición de tecnologías como JavaScript, provocaron que las páginas HTML también incluyeran el código de las aplicaciones (scripts) que se utilizan para crear páginas web dinámicas.

Incluir en una misma página HTML los contenidos, el diseño y la programación complica en exceso su mantenimiento. Normalmente, los contenidos y el diseño de la página web son responsabilidad de diferentes personas, por lo que es adecuado separarlos. CSS es el mecanismo que permite separar los contenidos definidos mediante XHTML.

Otra ventaja de la separación de los contenidos y su presentación es que los documentos XHTML creados son más flexibles, ya que se adaptan mejor a las diferentes plataformas: pantallas de ordenador, pantallas de dispositivos móviles, impresoras y dispositivos utilizados por personas discapacitadas.

De esta forma, utilizando exclusivamente XHTML se crean páginas web "feas" pero correctas. Aplicando CSS, se pueden crear páginas "bonitas" a partir de las páginas XHTML correctas.

Como consume un archivo hijo HTML al utilizar la herencia del archivo base HTML.

Una forma clásica de solucionar este problema es usar includes, insertando dentro de las páginas HTML a "incluir" una página dentro de otra. Es más, Django admite esta aproximación, con la etiqueta {% include %} anteriormente descrita. Pero la mejor forma de solucionar este problema con Django es usar una estrategia más elegante llamada herencia de plantillas.

En esencia, la herencia de plantillas te deja construir una plantilla base "esqueleto" que contenga todas las partes comunes de tu sitio y definir "bloques" que los hijos puedan sobrescribir.

Veamos un ejemplo de esto creando una plantilla completa para nuestra vista `current_datetime`, editando el archivo `current_datetime.html`:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN">
<html lang="en">
<head>
  <title>The current time</title>
</head>
<body>
```



```
<h1>My helpful timestamp site</h1>
```

```
<p>It is now {{ current_date }}.</p>
```

```
<hr>
```

```
<p>Thanks for visiting my site.</p>
```

```
</body>
```

```
</html>
```

Esto se ve bien, pero ¿Qué sucede cuando queremos crear una plantilla para otra vista — digamos, ¿La vista `hours_ahead` del Capítulo 3? Si queremos hacer nuevamente una agradable, válida, y completa plantilla HTML, crearíamos algo como:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN">
```

```
<html lang="en">
```

```
<head>
```

```
  <title>Future time</title>
```

```
</head>
```

```
<body>
```

```
  <h1>My helpful timestamp site</h1>
```

```
  <p>In {{ hour_offset }} hour(s), it will be {{ next_time }}.</p>
```

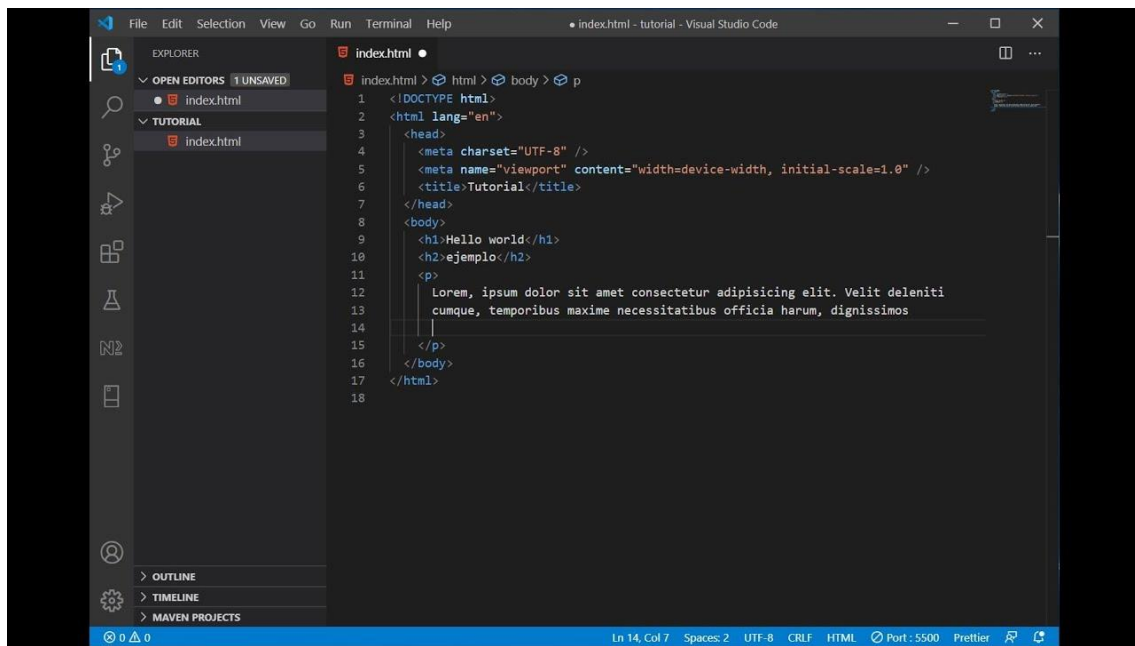
```
  <hr>
```

```
  <p>Thanks for visiting my site.</p>
```

```
</body>
```

```
</html>
```

Crear un archivo base HTML en la APPS core



Crear un view que llame al HTML hijo



Crear la urls que llame al views.

```

base64.cc
31 void base64_encode(const uint8_t * data, size_t len, char * dst)
32 {
33     size_t src_idx = 0;
34     size_t dst_idx = 0;
35     for (; (src_idx + 2) < len; src_idx += 3, dst_idx += 4)
36     {
37         uint8_t s0 = data[src_idx];
38         uint8_t s1 = data[src_idx + 1];
39         uint8_t s2 = data[src_idx + 2];
40
41         dst[dst_idx + 0] = charset[(s0 & 0xfc) >> 2];
42         dst[dst_idx + 1] = charset[((s0 & 0x03) << 4) | ((s1 & 0xf0) >> 4)];
43         dst[dst_idx + 2] = charset[((s1 & 0x0f) << 2) | (s2 & 0xc0) >> 6];
44         dst[dst_idx + 3] = charset[(s2 & 0x3f)];
45     }
46
47     if (src_idx < len)
48     {
49         uint8_t s0 = data[src_idx];
50         uint8_t s1 = (src_idx + 1 < len) ? data[src_idx + 1] : 0;
51
52         dst[dst_idx++] = charset[(s0 & 0xfc) >> 2];
53         dst[dst_idx++] = charset[((s0 & 0x03) << 4) | ((s1 & 0xf0) >> 4)];
54         if (src_idx + 1 < len)
55             dst[dst_idx++] = charset[((s1 & 0x0f) << 2)];
56     }
57 }

```

Line 31, Column 52 Spaces: 4 C++

Crear las tablas del sistema de usuarios para utilizar el panel de administración.

Salman Raiwoof 0 + New Howdy, The Admin

Dashboard Posts Media Pages Comments Appearance Plugins **Users** All Users Add New Your Profile Tools Settings Collapse menu

Users Add New

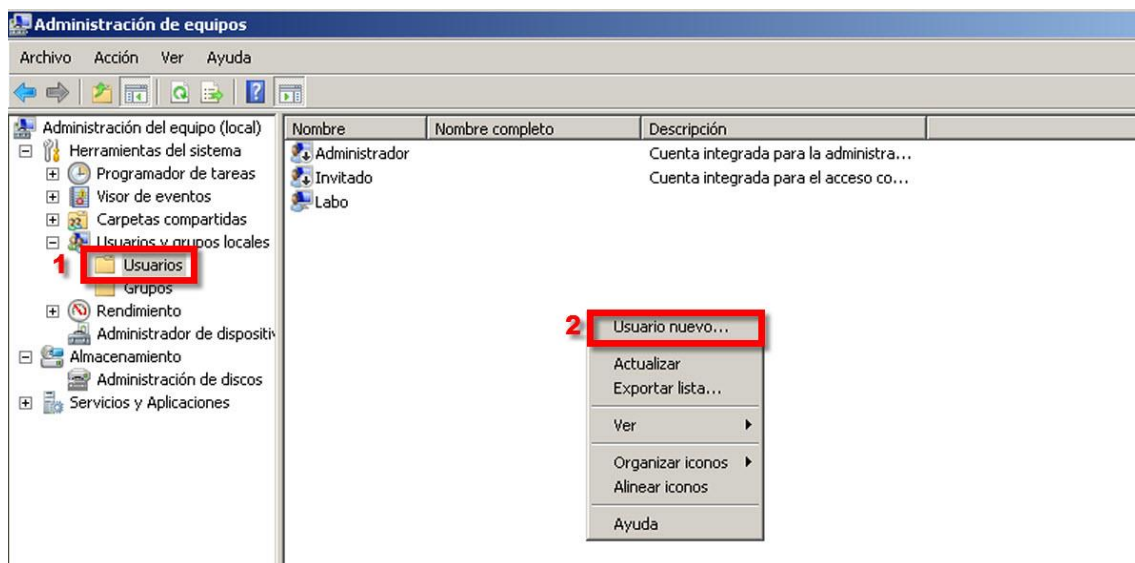
All (1) | Administrator (1)

Bulk Actions Apply Change role to... Change

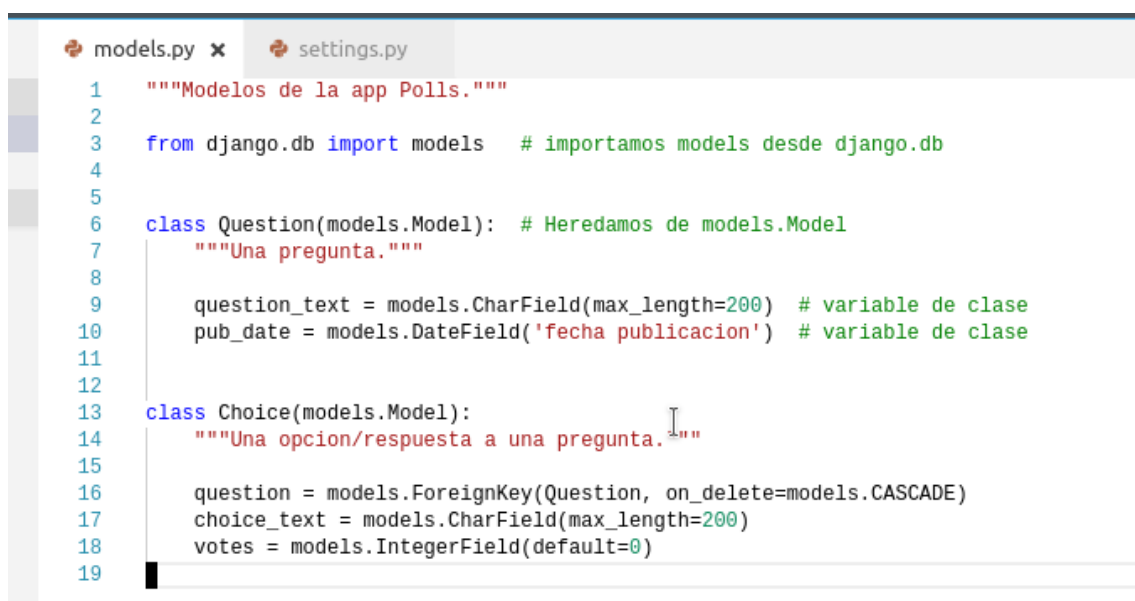
<input type="checkbox"/>	Username	Name	Email	Role	Posts
<input type="checkbox"/>	user_admin	The Admin	admin@domain.com	Administrator	1
<input type="checkbox"/>	user_editor	The Editor	editor@domain.com	Editor	1
<input type="checkbox"/>	user_author	The Author	author@domain.com	Author	1
<input type="checkbox"/>	user_contributor	The Contributor	contributor@domain.com	Contributor	1
<input type="checkbox"/>	user_subscriber	The Subscriber	subscriber@domain.com	Subscriber	1

Bulk Actions Apply Change role to... Change 1 item

Crear un usuario para poder ingresar al Panel de Administración



Crear un modelo en Django.



Migrar el Modelo a la base del Panel de Administración.

The screenshot shows a code editor with a file explorer on the left and a code editor on the right. The file explorer shows a project named 'MY_VIDEO_RENTAL' with a folder 'videos' containing files like 'models.py', 'admin.py', 'apps.py', 'tests.py', 'views.py', 'db.sqlite3', and 'manage.py'. The code editor shows the content of 'models.py' with the following code:

```

1 from django.db import models
2
3 # Create your models here.
4
5 class Movie(models.Model):
6     title=models.CharField(max_length=200)
7     length=models.PositiveIntegerField()
8     release_year=models.PositiveIntegerField()
9
10
11     def __str__(self):
12         return self.title
13
14 class Customer(models.Model):
15     first_name=models.CharField(max_length=20)
16     last_name=models.CharField(max_length=20)
17     phone=models.PositiveIntegerField()
18
19     def __str__(self):
20         return self.first_name+' '+self.last_name
21

```

Integrar el Modelo al Panel de Administración.

The screenshot shows a web browser displaying the Django Admin interface for a 'todo' application. The browser's address bar shows 'http://localhost:8000/admin/'. The page title is 'todo - base.html'. The page content shows a list of todos, with a red arrow pointing to the 'logo.png' file in the file explorer on the left. The file explorer shows a project named 'todowoo' with a folder 'static' containing files like 'logo.png', 'base.html', 'completedodos.html', 'createtodo.html', 'currenttodos.html', 'home.html', 'loginuser.html', 'signupuser.html', 'viewtodo.html', 'init.py', and 'admin.py'. The code editor shows the content of 'base.html' with the following code:

```

1 {% load static %}
2 <!doctype html>
3 <html lang="en">
4 <head>
5     <!-- Required meta tags -->
6     <meta charset="utf-8">
7     <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">
8
9     <!-- Bootstrap CSS -->
10    <link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.4.1/css/bootstrap.min.css">
11
12    <link rel="icon" type="image/png" href="{% static 'todo/logo.png' %}">
13
14    <title>Todo Woo</title>
15 </head>
16 <body>
17

```

The terminal at the bottom shows the following output:

```

[12/Dec/2020 17:45:19] "POST /login/?next=/create/ HTTP/1.1" 302 0
[12/Dec/2020 17:45:19] "GET /current/ HTTP/1.1" 200 3549
[12/Dec/2020 17:45:19] "GET /static/todo/logo.png HTTP/1.1" 404 1665
[12/Dec/2020 17:46:32] "GET /current/ HTTP/1.1" 200 3549
[12/Dec/2020 17:46:32] "GET /static/todo/logo.png HTTP/1.1" 404 1665
[12/Dec/2020 17:46:35] "GET /login/?next=/create/ HTTP/1.1" 200 3978
[12/Dec/2020 17:46:40] "GET /login/?next=/create/ HTTP/1.1" 200 3978
[12/Dec/2020 17:46:40] "GET /static/todo/logo.png HTTP/1.1" 404 1665

```

Ingresar información al modelo por el Panel de Administración.

```

1  from django.db import models
2
3  # Create your models here.
4
5  class Movie(models.Model):
6      title=models.CharField(max_length=200)
7      length=models.PositiveIntegerField()
8      release_year=models.PositiveIntegerField()
9
10
11     def __str__(self):
12         return self.title
13
14     class Customer(models.Model):
15         first_name=models.CharField(max_length=20)
16         last_name=models.CharField(max_length=20)
17         phone=models.PositiveIntegerField()
18
19     def __str__(self):
20         return self.first_name+' '+self.last_name
21

```

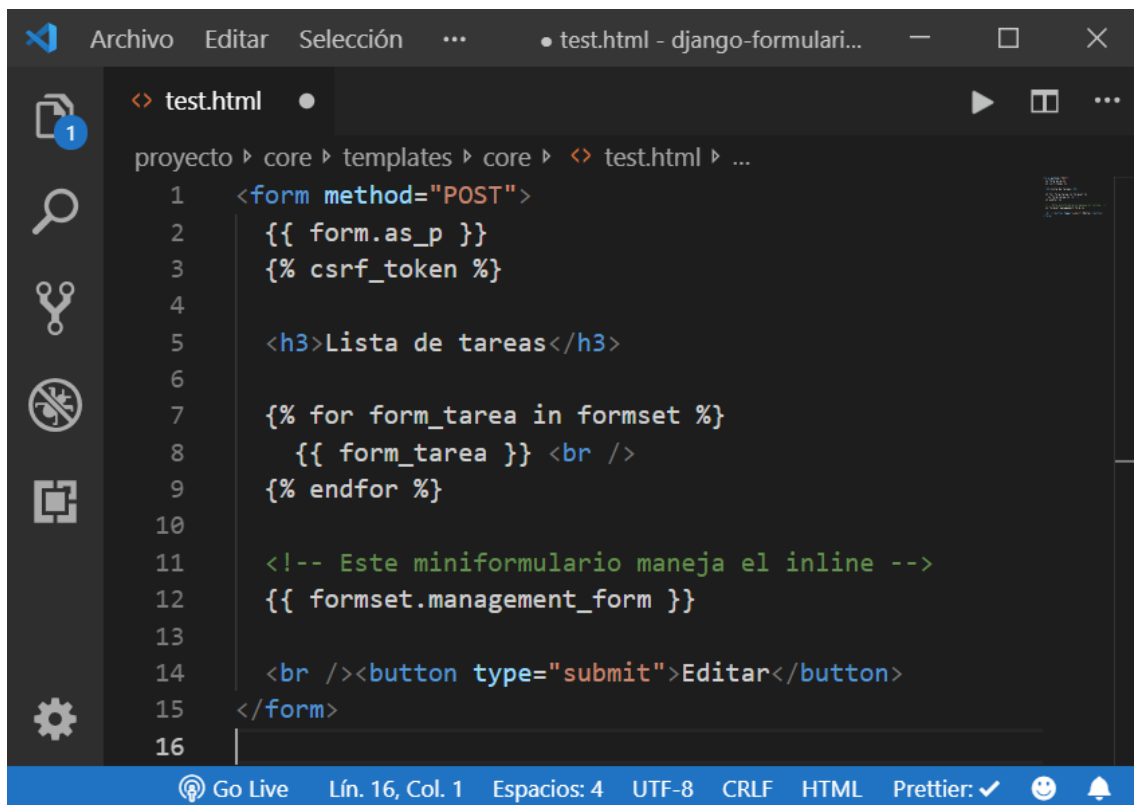
Realizar la consulta de todo lo ingresado en el modelo desde el views.

```

1  # Build paths inside the project like this: os.path.join(BASE_DIR, ...)
2  import os
3
4  BASE_DIR = os.path.dirname(os.path.dirname(os.path.abspath(__file__)))
5
6
7  # Quick-start development settings - unsuitable for production
8  # See https://docs.djangoproject.com/en/1.8/howto/deployment/checklist/
9
10 # SECURITY WARNING: keep the secret key used in production secret!
11 SECRET_KEY = '!68&wz7nu$vw%do7zpi_c1vt=1#qw4y+d*+%4ja2o7ybyr7tk^'
12
13 # Application definition
14
15 INSTALLED_APPS = (
16     'django.contrib.admin',
17     'django.contrib.auth',
18     'django.contrib.contenttypes',
19     'django.contrib.sessions',
20     'django.contrib.messages',
21     'django.contrib.staticfiles',
22     'apps.tarea',
23 )

```

Mostrar los datos guardados en el modelo al HTML hijo.



```

1  <form method="POST">
2      {{ form.as_p }}
3      {% csrf_token %}
4
5      <h3>Lista de tareas</h3>
6
7      {% for form_tarea in formset %}
8          {{ form_tarea }} <br />
9      {% endfor %}
10
11     <!-- Este miniformulario maneja el inline -->
12     {{ formset.management_form }}
13
14     <br /><button type="submit">Editar</button>
15 </form>
16

```

Go Live Lín. 16, Col. 1 Espacios: 4 UTF-8 CRLF HTML Prettier: ✓

Crear un proyecto con la máquina virtual.

```

(proyecto) C:\Users\Andrey\Documents\proyecto>python Scripts\django-admin.py startproject tareas

(proyecto) C:\Users\Andrey\Documents\proyecto>dir
El volumen de la unidad C no tiene etiqueta.
El número de serie del volumen es: 0A32-ABC3

Directorio de C:\Users\Andrey\Documents\proyecto

06/03/2016  04:39 p. m.      <DIR>          .
06/03/2016  04:39 p. m.      <DIR>          ..
29/01/2016  11:39 p. m.      <DIR>          Include
06/03/2016  04:08 p. m.      <DIR>          Lib
06/03/2016  04:26 p. m.              60 pip-selfcheck.json
06/03/2016  04:26 p. m.      <DIR>          Scripts
06/03/2016  04:39 p. m.      <DIR>          tareas
                        1 archivos             60 bytes
                        6 dirs  176.982.204.416 bytes libres

(proyecto) C:\Users\Andrey\Documents\proyecto>_

```

Descargar los instaladores de Django al proyecto

```
C:\Users\sdkca\Desktop>python get-pip.py
Collecting pip
  Downloading pip-8.1.2-py2.py3-none-any.whl (1.2MB)
    100% |#####| 1.2MB 589kB/s
Collecting wheel
  Downloading wheel-0.29.0-py2.py3-none-any.whl (66kB)
    100% |#####| 71kB 1.3MB/s
Installing collected packages: pip, wheel
  Found existing installation: pip 8.1.1
    Uninstalling pip-8.1.1:
      Successfully uninstalled pip-8.1.1
Successfully installed pip-8.1.2 wheel-0.29.0

C:\Users\sdkca\Desktop>pip

Usage:
  pip <command> [options]

Commands:
  install          Install packages.
  download         Download packages.
  uninstall        Uninstall packages.
  freeze           Output installed packages in requirements format.
  list             List installed packages.
  show             Show information about installed packages.
  search           Search PyPI for packages.
  wheel            Build wheels from your requirements.
  hash             Compute hashes of package archives.
  completion       A helper command used for command completion
  help             Show help for commands.
```