



# Desarrollo Web Python con Django

**Desarrollador de aplicaciones  
Full Stack**

**Python Trainee**



## 5.3.- Contenido 3: Implementar formularios en un aplicativo web utilizando Django para el despliegue y procesamiento de la información para dar solución a un requerimiento

---

### Objetivo de la jornada

---

1. Construye un proyecto utilizando Django para la integración de un formulario básico
2. Procesa un formulario Django utilizando templates para dar solución a un requerimiento
3. Codifica plantillas de formulario reutilizables para dar solución a un requerimiento
4. Maneja mensaje de errores de formularios en el template para dar solución a un requerimiento

### 5.3.1.- Forms en Django

#### 5.3.1.1. Formularios en Django

Los formularios son una parte omnipresente de la web moderna, pero a veces son complicados de implementar correctamente. Cada vez que acepta la entrada del usuario, hay problemas de seguridad (ataques XSS - Cross-site scripting), se requiere un manejo adecuado de errores y hay consideraciones de la interfaz de usuario sobre cómo alertar al usuario sobre problemas con el formulario.

Para ver más de XSS [https://es.wikipedia.org/wiki/Cross-site\\_scripting](https://es.wikipedia.org/wiki/Cross-site_scripting)

Afortunadamente para nosotros, los formularios integrados de Django eliminan gran parte de la dificultad y brindan un amplio conjunto de herramientas para manejar casos de uso comunes al trabajar con formularios.

#### 5.3.1.2 Forms de Django v/s formularios HTML

Cómo hemos visto anteriormente, un formulario es una colección de elementos dentro de `<form> ... </form>` que permiten a un visitante realizar acciones como ingresar texto, seleccionar opciones, manipular objetos o controles, y así sucesivamente, y luego enviar esa información al servidor.



Algunos de estos elementos de formulario (entrada de texto o casillas de verificación) están integrados en HTML. Otros son mucho más complejos; una interfaz que muestra un selector de fecha o le permite mover un control deslizante o manipular controles normalmente utilizará JavaScript y CSS, así como elementos de formato HTML **<input>** para lograr estos efectos.

Los métodos **GET** y **POST** del protocolo **HTTP** son utilizados para manejar las acciones que ocurren entre el navegador web y el servidor al utilizar formularios. La página que contiene el formulario es cargada inicialmente a través de un requerimiento **GET** desde el navegador. Posteriormente, los datos ingresados por el usuario son enviados a través de un requerimiento **POST** gatillado por el botón de envío del formulario.

El manejo de formularios es un asunto complejo. Considere un sitio web típico, donde existen numerosos elementos de datos de diversos tipos para ser mostrados en un formulario, representarlos como HTML, editarlos usando una interfaz conveniente, devolverlos al servidor, validarlos y limpiarlos, y luego guardarlos o pasarlos para su posterior procesamiento.

La funcionalidad de formulario de Django puede simplificar y automatizar grandes porciones de este trabajo, y también puede hacerlo de forma más segura que la forma en que lo haríamos manualmente como desarrolladores intentando desarrollar los detalles que involucran.

Django maneja tres partes distintas en el ámbito de formularios:

- Preparar y reestructurar los datos para que estén listos para la renderización
- Crear formularios HTML para los datos
- Recibir y procesar el formulario y los datos enviados por el cliente

Es posible escribir código que haga todo esto manualmente, pero Django puede encargarse de todo por nosotros.

### 5.3.1.3. La Clase Form de Django

En el contexto de una aplicación web, la palabra **form** puede referirse a la etiqueta **<form>** HTML, o a la clase Form de Django que lo produce, o a los datos estructurados devueltos cuando se envían.

En el corazón de este sistema de componentes se encuentra la clase **Form** de Django, que describe un formulario y determina cómo funciona y se muestra en nuestra aplicación.



De manera similar a como se definen los campos de una tabla en una base de datos, los campos de una clase Form se asignan a los elementos **<input>** de un formulario HTML.

Los campos de un formulario son en sí mismos clases, que administran los datos del formulario y realizan la validación cuando se envía un formulario. Por ejemplo, un **DateField** o un **FileField** manejan tipos de datos muy diferentes y tienen que hacer cosas diferentes con ellos. Todo esto lo facilita el uso de la clase **Form de Django** y ahorra una cantidad de trabajo y tiempo de desarrollo en comparación a su desarrollo manual caso a caso.

Un campo de un formulario se representa ante un usuario en el navegador como un "widget" HTML, que es un componente de la maquinaria que genera la interfaz de usuario. Cada tipo de campo tiene una clase Widget predeterminada, y pueden ser sobrescritos según sea necesario.

#### 5.3.1.4. Construyendo un formulario en Django

Vamos a retomar nuestros ejemplos utilizando la plantilla que hemos visto en estas últimas clases, recordemos que creamos la aplicación **app** y dejamos nuestra página en esta ubicación <http://127.0.0.1:8000/app/>

The screenshot shows a web browser window with the address bar displaying `127.0.0.1:8000/app/`. The page content includes a heading "Nuestro Producto Estrella", a promotional message "Regístrate dentro de los 100 primeros y recibirás un descuento", and a logo for "Gaines, Faehner & Gordon, Inc." which is an orange triangle with three black dots inside. Below the logo is a copyright notice: "Image copyrights to https://www.cfg.com/". To the right of the logo is a registration form with fields for "Nombre" and "Email", and a "Registrate" button. Below the form is a section titled "¡Nuestros Beneficios!" with a bulleted list: "Rapidez", "Seguridad", "Gran Precio", and "Flexibilidad". At the bottom of the page, there are four links: "Like Red Social 1", "Like Red Social 2", "Like Red Social 3", and "Like Red Social 4".

Nuestro Producto Estrella

Regístrate dentro de los 100 primeros y recibirás un descuento



Image copyrights to <https://www.cfg.com/>

Nombre

Email

¡Nuestros Beneficios!

- Rapidez
- Seguridad
- Gran Precio
- Flexibilidad

[Like Red Social 1](#) | [Like Red Social 2](#) | [Like Red Social 3](#) | [Like Red Social 4](#)



como vimos anteriormente, usamos la herencia de templates para organizar el código en esta template:

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

{% block title %}Landing Page{% endblock %}

{% block header %}    <h1>Nuestro Producto Estrella</h1>
<h2>Regístrate dentro de los      100 primeros y recibirás un
descuento</h2>
{% endblock %}

{% block imagen %}        <p>Image copyrights to
https://www.cfg.com/</p>{% endblock %}{% block formulario %}
<form>                <p>Nombre</p>                <input type="text">
<p>Email</p>                <input type="email"/>                <br>                <br>
<button type="submit">Regístrate</button>                </form>                <br>
<h3>¡Nuestros Beneficios!</h3>                <ul>                <li>Rapidez</li>
<li>Seguridad</li>                <li>Gran Precio</li>
<li>Flexibilidad</li>                </ul>{% endblock %}{% block footer %}
<br>                <p>                <a href="">Like Red Social 1</a> |                <a
href="">Like Red Social 2</a> |                <a href="">Like Red Social
3</a> |                <a href="">Like Red Social 4</a>                </p>                <br>
<p>Copyright &copy; 2019-2020 -                Desarrollos de Landing Pages
Inc.</p>{% endblock %}
```

Lo que nos interesa es el formulario que se encuentra del **{% block formulario %}**

```
...
<form>
    <p>Nombre</p>
    <input type="text">
    <p>Email</p>
    <input type="email"/>
    <br>
    <br>
    <button type="submit">Regístrate</button>
</form>
...
```

### 5.3.1.5. Formularios basados en forms.py

No basaremos en **{% block formulario %}**, del cual ya conocemos su aspecto al implementarlo en HTML puro, para revisar cómo sería esta implementación utilizando las facilidades que Django entrega para la generación de formularios de manera más fácil.

Las etiquetas HTML **<form>** **</form>** definen el elemento **form**, y cada uno de los campos del formulario está contenido dentro del elemento **form**. En este formulario,



hemos definido un campo de texto, un campo de email y un botón de registro. En HTML5, hay muchos otros tipos de elementos, incluidos campos de fecha y hora, casillas de verificación, botones de opción y más. En este ejemplo, renderizaremos los elementos del formulario como elementos de párrafo `<p>`, pero también es común representar los formularios como una lista ordenada `<ol>` o desordenada `<ul>`, o como una tabla con los campos que llenan las filas de la tabla.

Si bien crear un formulario básico es simple, las cosas se complican una vez que necesita usar el formulario en una situación de la vida real. En un sitio web publicado, deben validarse los datos enviados con el formulario. Si el campo es obligatorio, debe comprobar que el campo no esté en blanco. Si el campo no está en blanco, debe verificar si los datos enviados son del tipo de datos válido. Por ejemplo, si está solicitando una dirección de correo electrónico, debe verificar que se ingrese una dirección de correo electrónico válida.

También debe asegurarse de que el formulario maneja los datos ingresados de manera segura. Una forma común en que los sitios con atacados es enviar código malicioso a través de formularios para intentar violar la seguridad del sitio.

Para complicar aún más las cosas, los usuarios del sitio web esperan comentarios cuando no han llenado un formulario correctamente. Por lo tanto, también debe tener alguna forma de mostrar los errores en el formulario para que el usuario los corrija antes de permitirle enviarlo.

Crear formularios, validar datos y proporcionar retroalimentación es un proceso tedioso si lo codifica todo a mano. Django es flexible en su enfoque para la creación y manejo de formularios.

No es recomendable realizar todo el manejo de formularios de forma manual. A menos que tengamos una aplicación muy especializada en mente, Django tiene muchas herramientas y bibliotecas que facilitan la creación de formularios. En particular, la clase **Form** de Django ofrece un conjunto muy conveniente de métodos para encargarse de la mayor parte del procesamiento y la validación de formularios.

Con la clase **Form**, creamos una clase especial que se parece mucho a un modelo de Django (Que abordaremos más adelante). Los campos de clase **Form** tienen una validación incorporada, según el tipo de campo, y un widget HTML asociado.

Exploremos más la clase **Form** con el shell interactivo de Django. Desde dentro de nuestro entorno virtual, ejecutamos el comando:

```
(venv)$ python manage.py shell
```

Con esto, podemos experimentar con la clase **Form** y observar sus facilidades:



```
>>> from django import forms>>> class
FormularioEspecial(forms.Form):...     nombre =
forms.CharField(max_length=50)...     email =
forms.EmailField(max_length=50)...
```

Como se ve arriba, para utilizar la clase **Form** de Django debemos:

1. Importar el módulo **forms** de Django.
2. Crear una clase, que en este caso hemos llamado **FormularioEspecial**, que hereda de la clase **form.Form** de Django.
3. Definimos los campos que nos interesa que nuestro formulario contenga. En este caso, **nombre** y **email**, replicando lo que anteriormente hemos hecho utilizando HTML puro.

Con lo anterior, si ahora creamos una instancia de la clase **FormularioEspecial**, que llamamos **nuestro\_formulario** podemos explorar algunos atributos y métodos:

```
>>> nuestro_formulario = FormularioEspecial()>>>
print(nuestro_formulario.as_p())
<p><label for="id_nombre">Nombre:</label> <input type="text"
name="nombre" maxlength="50" id="id_nombre"></p><p><label
for="id_email">Email:</label> <input type="email" name="email"
required id="id_email"></p>
```

Podemos ver el código HTML que automáticamente Django construye para implementar el formulario en nuestro template. **.as\_p()** es un método de la clase **Form** que construye el formulario en una estructura de párrafos HTML **<p>**. Existen otros métodos, tales como **.as\_ul()** y **.as\_table()**, que generan el formulario con etiquetas de listas desordenadas o de tablas HTML, respectivamente.

Existe el método de validación de formulario, **.is\_valid()**, que permite verificar que las condiciones que se han definido para cada campo en la clase **FormularioEspecial** se cumplen al momento de ingresar datos al formulario. Además, el atributo **.errors** nos entrega los textos descriptivos del tipo de error para cada campo, en caso de corresponder. Podemos experimentar un poco sobre estos aspectos, utilizando el **shell** de Django, simulando el envío de datos vacíos **{}** como argumento en la creación del objeto **nuestro\_formulario**, de la siguiente forma:

```
>>> nuestro_formulario = FormularioEspecial({})>>>
nuestro_formulario.is_valid() False>>>
nuestro_formulario.errors{'email': ['This field is required.']}
```

Como hemos definido el campo **email** como mandatorio, el error enviado es **This field is required**. En caso del campo **nombre** no se indica ningún error pues ha sido definido como opcional.

En nuestro caso, para implementar los códigos que hemos probado utilizando la consola de Django, tendremos que crear un archivo **forms.py** dentro del directorio **app**, con el siguiente contenido, que ya hemos ejecutado más arriba:





### forms.py

```
from django import formsclass FormularioEspecial(forms.Form):
    nombre = forms.CharField(max_length=30, required=True,
        _messages={'required': 'Debe ingresar este dato',
            'max_length': 'El Nombre es muy largo'})
    email = forms.EmailField(required=True,
        error_messages={'required': 'No has ingresado tu email',
            'invalid': 'Éste no parece ser un email'})
```

Como se muestra en este ejemplo, podemos definir mensajes personalizados para los errores de validación de los cambios de nuestro formulario. En caso de el campo **nombre** hemos definido un máximo de 30 caracteres y que sea un campo requerido. Además, para el campo **email** hemos definido que es requerido. Como se muestra en **error\_messages**, es posible definir mensajes diferenciados para distintos errores de validación.

Por otro lado, debemos registrar nuestra clase **FormularioEspecial** en nuestro archivo de vistas **views.py**, agregando el correspondiente **import** que se muestra a continuación junto con la clase **IndexView**, pero modificada para manejar las acciones del formulario.

### views.py

```
...
from .forms import FormularioEspecial
class IndexView(TemplateView):
    template_name = "app/index.html"
    def get_context_data(self, **kwargs):
        context = super(IndexView,
self).get_context_data(**kwargs)
        context['form']=FormularioEspecial()
        return context
    def post(self, request, *args, **kwargs):
        form = FormularioEspecial(request.POST)
        print(form.is_valid())
        if form.is_valid():
            print('valido')
            return HttpResponseRedirect(
                "El formulario ha sido validado con éxito!!!")
        else:
            return render(request, self.template_name, {'form':form})
...
```

El formulario dentro del bloque **{% block formulario %}** que vimos más arriba, queda reducido en este caso a la simple forma:

```
<form action="" method="post" novalidate>
    {% csrf_token %}
    {{form}}    <button type="submit">Regístrate
</button></form>
```

Los elementos de ingreso de datos del formulario son incluidos dinámicamente por Django al momento de renderizar la página para enviarla al cliente.

Por motivos de orden visual es posible tratar individualmente cada uno de los campos de la siguiente forma, con lo que tenemos un poco más de control sobre los saltos de línea y la disposición visual de los elementos:





```
<form action="" method="post" novalidate>
  {% csrf_token %}
  {{form.nombre.label|linebreaks }}
  {{form.nombre|linebreaks }}
  {{form.nombre.errors|linebreaks }}
  {{form.email.label|linebreaks }}
  {{form.email|linebreaks }}
  {{form.email.errors|linebreaks }}
  <button type="submit">Regístrate</button></form>
```

El atributo **novalidate** en el elemento **<form>** HTML se incluye para permitir a Django manejar las validaciones de datos en lugar de los mecanismos que posee el navegador web del usuario. Con esto los mensajes pueden ser más personalizados.

De esta forma, si intentamos hacer envío del formulario sin datos, la página nos entregará los siguientes mensajes de error bajo las casillas de **input** del formulario:

Por otro lado, los mensajes de error diferenciados podemos probarlos ingresando, por ejemplo, en el campo email un texto sin el formato correcto. De esta forma se nos entrega:



## NUESTRO PRODUCTO ESTARÁ

dentro de los 100 primeros y recibirás un descuento



its to <https://www.cfr.com/>

Nombre

Email

- Éste no parece ser un email

Nuestros Beneficios

### 5.3.1.6. Formularios usando modelos en Django

En este caso veremos que Django nos permite confeccionar formularios con una gran facilidad cuando nuestro formulario está relacionado con un modelo de datos. Este tópico es parte de un apartado posterior. Sin embargo, lo esbozaremos de manera simple para conocer sus características.

En este caso no necesitamos incluir un archivo **forms.py**, pues a través de las clases correctas Django puede realizar todos los manejos necesarios para conectar directamente nuestras necesidades de formularios con los modelos creados para nuestra aplicación.

Para poder construir nuestro formulario primero necesitamos un Modelo, de forma que crearemos uno básico en **sitio/app/models.py** el archivo existente

```
from django.db import models
class App(models.Model):
    nombre = models.CharField(max_length=64, blank=True, null=True)
    email = models.EmailField()
```

Necesitamos activar el modelo, para lo cual hacemos **python manage.py makemigrations app**

```
$ python manage.py makemigrations app
Migrations for 'app':
  app/migrations/0001_initial.py
    - Create model App
```

Enlace con la vista



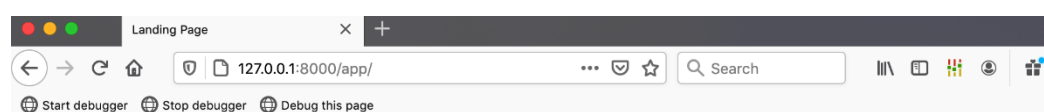
Nuestro modelo está listo, ahora necesitamos engancharlo con la vista, para tales propósitos realizamos los siguientes cambios en nuestro archivo **sitio/app/views.py**, recordemos que estamos trabajando solo con **IndexView**

```
...
from .models import App from django.views.generic.edit import
CreateView
class IndexView(CreateView):
    model = App
    template_name = "app/index.html"
    fields = ['nombre', 'email']
...
```

La variable **fields** indica los campos que queremos desplegar en nuestro formulario. Ahora vamos a intervenir la plantilla **index.html**

```
...
{% block formulario %}
    <form action="" method="post">{% csrf_token %}
        {{ form.as_p }}
        <input type="submit" value="Regístrate">
    </form>
...
{% endblock %}
...
```

Si nos dirigimos a <http://127.0.0.1:8000/app/> veremos



## Nuestro Producto Estrella

**Regístrate dentro de los 100 primeros y recibirás un descuento**



Image copyrights to <https://www.cfg.com/>

Nombre:

Email:

### ¡Nuestros Beneficios!

- Rapidez
- Seguridad
- Gran Precio
- Flexibilidad

[Like Red Social 1](#) | [Like Red Social 2](#) | [Like Red Social 3](#) | [Like Red Social 4](#)

Copyright © 2019-2020 - Desarrollos de Landing Pages Inc.

Eliminamos todo el código HTML y lo reemplazamos por la alternativa que nos da Django, y aunque a primera vista luzca igual a nuestra versión HTML es radicalmente



distinta en su mecánica. Para mostrarlo inspeccionemos un poco el código HTML que se genera:

The screenshot shows a web browser window with the URL `127.0.0.1:8000/app/`. The page displays a registration form titled "Regístrate dentro de los 100 primeros y recibirás un descuento". The form includes a logo for "Faeber & Gordon, Inc." and two input fields: "Nombre:" and "Email:". Below the form, there is a section titled "¡Nuestros Beneficios!". The browser's developer tools are open, showing the HTML source code. The code is a form with a hidden CSRF token, a text input for "Nombre", and an email input for "Email". The form is enclosed in a `<form>` tag with `method="post"`. The CSRF token is a long alphanumeric string.

Este es el fragmento relevante para nuestro formulario

```
<form action="" method="post"><input type="hidden"
  name="csrfmiddlewaretoken"
  value="QmJjvd0rlVzUscngLF4MRniivUmya9DKt3BSVlHL5x6Mp0cVuCDkU2LQjGci4TkQ">
  <p>
    <label for="id_nombre">Nombre:</label> <input type="text"
      name="nombre" maxlength="64" id="id_nombre">
  </p>
  <p>
    <label for="id_email">Email:</label> <input type="email"
      name="email" maxlength="254" required id="id_email">
  </p>
</form>
```

Si vemos con detención nos damos cuenta de la inclusión de un token CSRF, lo cual es fundamental para la seguridad de nuestro sitio. Un token CSRF es un valor único, secreto e impredecible que genera la aplicación del lado del servidor y se transmite al cliente de tal manera que se incluye en una solicitud HTTP posterior realizada por el cliente.

Nuestros campos están dentro de tags de párrafo `<p></p>`, ya que así lo decidimos cuando escribimos esta línea `{{ form.as_p }}`



## Renderizar los campos de forma manual

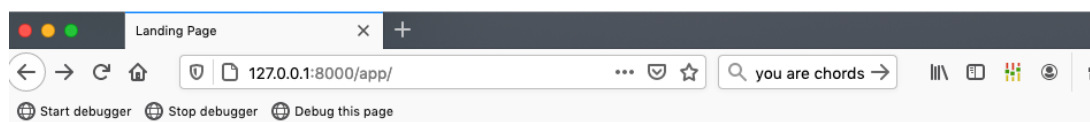
Dentro de nuestro primer ejemplo de creación de un formulario utilizando la clase **Form** en Django, adelantamos un poco el concepto que revisamos en esta sección. Pueden presentarse muchas situaciones en las necesidades que tengamos para mostrar un formulario no se satisfagan con la simple acción de incluir `{{ form }}` en el template respectivo. Por ejemplo, por requerimiento del proyecto, podríamos requerir que el campo email se ubique en la parte superior y el campo nombre en la parte inferior.

Para esta situación podemos usar una forma manual de integrar los campos en la plantilla, en `sitio/app/templates/app/index.html`. Esto lo hacemos llamando a los atributos individuales del objeto **form** que pasamos desde Python al template donde estamos incorporando nuestro formulario.

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

...{% block formulario %}
    <form action="" method="post">{% csrf_token %}
        <label for="email">Email</label>
        {{ form.email }}
        <br/>
        <label for="nombre">Nombre</label>
        {{ form.nombre }}
        <br/>
        <input type="submit" value="Regístrate">
    </form>
    ...
{% endblock %}
```

Podemos ver que esa pequeña modificación obtenemos:



## Nuestro Producto Estrella

Regístrate dentro de los 100 primeros y recibirás un descuento



Image copyrights to <https://www.cfg.com/>

Email   
Nombre

### ¡Nuestros Beneficios!

- Rapidez
- Seguridad
- Gran Precio
- Flexibilidad

[Like Red Social 1](#) | [Like Red Social 2](#) | [Like Red Social 3](#) | [Like Red Social 4](#)

Copyright © 2019-2020 - Desarrollos de Landing Pages Inc.

### 5.3.1.7. Procesamiento de formularios

Los formularios de Django son potentes, flexibles, extensibles y robustos. La combinación de formularios, modelos y vistas nos permite hacer mucho trabajo con poco esfuerzo. La curva de aprendizaje vale la pena: una vez que aprenda a trabajar con fluidez con estos componentes, encontrará que Django ofrece la capacidad de crear una cantidad asombrosa de funcionalidad útil y estable a un ritmo asombroso. El formulario de cambio de datos más simple que podemos hacer es un **ModelForm** que utiliza varios validadores predeterminados tal cual, sin modificaciones. Cuando tenemos esto en nuestro archivo **sitio/app/views.py**

```
...
from django.views.generic.edit import CreateView
class IndexView(CreateView):
    model = App
    template_name = "app/index.html"
    fields = ['nombre', 'email']
...
```

- A **IndexView** le asignamos el modelo **App** como propio.
- La vista genera un **ModelForm** basado en el modelo **App**.
- **ModelForm** se basan en las reglas de validación de campo predeterminadas del modelo **App**.



Django nos ofrece muchos criterios predeterminados excelentes para la validación de datos, tales como los que hemos visto más arriba, tales como validación de formato de email, largo de textos, etc. Sin embargo, a veces los criterios predeterminados no son suficientes y nos referimos aquí a una mayor personalización las validaciones de formularios de Django.

Como primer paso, el siguiente patrón demostrará cómo crear un validador de campo personalizado.

¿Qué pasaría si quisiéramos estar seguros de que cada uso del campo nombre en la aplicación **app** de nuestro proyecto comenzara con la palabra "Estimad@"?

Este es un problema de validación de cadenas que se puede resolver con un validador de campo personalizado (custom field validator) simple.

Creamos **sitio/app/validators.py**

```
from django.core.exceptions import ValidationError
def validate_estimada(valor):
    """Generar un ValidationError si el valor no comienza con la
    palabra 'Estimad@'."""
    if not valor.startswith('Estimad@'):
        mensaje = 'El campo nombre debe comenzar con Estimad@'
        raise ValidationError(mensaje)
```

En Django, un validador de campo personalizado es simplemente un invocable (generalmente una función) que genera un error si el argumento enviado no pasa su prueba.

Si bien nuestra función de validación **validate\_estimada()** solo hace una simple verificación de cadenas para ilustrar como ejemplo, es bueno tener en cuenta que los validadores de campos de formulario pueden volverse bastante complejos en la práctica.

Para hacer funcionar nuestra validación, debemos dárselo a conocer a nuestro modelo, en **sitio/app/models.py** lo incorporamos al campo **nombre**

```
from django.db import models
from .validators import validate_estimada
class App(models.Model):
    nombre = models.CharField(max_length=64, blank=True,
    null=True, validators=[validate_estimada])
    email = models.EmailField()
```

Llenado el formulario y tratando de enviarlo vemos que nuestro mensaje que indica un problema de validación aparece (flecha roja)





Landing Page

127.0.0.1:8000/app/

## Nuestro Producto Estrella

Regístrate dentro de los 100 primeros y recibirás un descuento




Image copyrights to <https://www.cfg.com/>

- El campo nombre debe comenzar con Estimad@

Nombre:

Email:

### ¡Nuestros Beneficios!

- Rapidez
- Seguridad
- Gran Precio
- Flexibilidad

[Like Red Social 1](#) | [Like Red Social 2](#) | [Like Red Social 3](#) | [Like Red Social 4](#)

Copyright © 2019-2020 - Desarrollos de Landing Pages Inc.

Naturalmente si incluimos en nombre **Estimad@** nos encontraremos con un error **OperationalError** relacionado con el hecho de que aún no hemos trabajado con la base de datos.

### 5.3.1.8. Procesamiento de datos desde formularios

Una vez que tenemos el formulario integrado a nuestro template a través de alguno de los métodos revisados hasta este momento. La etapa siguiente es hacer algo útil para la aplicación utilizando los datos recibidos desde el formulario luego de la acción del usuario.

En cierto sentido, ya hemos procesado los datos del formulario de una u otra forma al haberlos validado, pero esto corresponde a un manejo que viene ya integrado como parte del framework.

Tomaremos uno de los primeros ejemplos de formularios Django que revisamos en este documento, el que no utiliza modelos para su funcionamiento.

Si recapitulamos, el archivo de vistas tuvo las siguientes últimas actualizaciones:

`views.py`

```
...
from .forms import FormularioEspecialclass IndexView(TemplateView):
    template_name = "app/index.html"
    def get_context_data(self, **kwargs):
        context = super(IndexView, self).get_context_data(**kwargs)
```



```
context['form']=FormularioEspecial()
return context
def post(self, request, *args, **kwargs):
    form = FormularioEspecial(request.POST)
    print(form.is_valid())
    if form.is_valid():
        print('valido')
        return HttpResponse(
            "El formulario ha sido validado con éxito!!!")
    else:
        return render(request, self.template_name, {'form':form})...
```

En este caso, las únicas acciones ejecutadas al momento de recibir los datos desde el formulario son validarlos y proceder según corresponda, a mostrar mensajes de error, o indicar que el formulario ha sido validado con éxito.

Supongamos algo que es de sentido común para una landing page como la de este prototipo. Es deseable que los datos ingresados por los usuarios que se registran no sólo sean validados sino también almacenados en algún medio que permita ser consultado posteriormente por el administrador del sitio.

Hasta el momento no hemos abordado las facilidades de interacción con bases de datos que provee Django, por lo que nos limitaremos a conocimiento que hemos adquirido en apartados anteriores. Desde el punto de vista de conocimientos de Python, estamos capacitados para realizar manejo de archivos y almacenamiento de datos en éstos.

Crearemos una lógica que, a modo ilustrativo, permita almacenar todos los usuarios que ingresen sus datos para registrarse en el sitio. Asignaremos un **id** único a cada uno de los registros que almacenará nuestro, que será de texto plano y almacenará los datos **id**, **nombre** y **email** de cada usuario registrado.

Para lo anterior deberemos alterar el archivo **views.py** de nuestra aplicación **app** para que incluya la capacidad de generar el código único de identificación de registro, y maneje el almacenamiento de los datos en un archivo que llamaremos **usuarios\_registrados.txt**.

Crearemos las siguientes dos funciones para manejo de datos en archivos dentro del archivo **views.py**, una para guardar datos y otra para leer datos desde el archivo **usuarios\_registrados.txt**. Como ya sabemos podemos hacer lo indicado con el siguiente código:

```
def leer_desde_archivo(nombre_archivo):
    if os.path.exists(nombre_archivo):
        archivo =
        open(nombre_archivo, 'r')
        texto_archivo = archivo.read()
        archivo.close()
    else:
        texto_archivo = 'Archivo sin datos'
    return texto_archivo
def guardar_a_archivo(nombre, email, nombre_archivo):
```



```
if os.path.exists(nombre_archivo):
    modo_archivo = 'a' # append si archivo existe
else:
    modo_archivo = 'w' # crea archivo si no existe
archivo = open(nombre_archivo, modo_archivo)
archivo.write('id: ' + str(uuid.uuid4()) +
             ' Nombre: ' + nombre + ' Email: ' +
             email + '\n')
archivo.close()
```

Con esto, podemos integrar el uso de estas funciones al procesamiento de los datos recibidos con el método POST cuando el formulario es enviado desde el navegador web.

Modificamos el código de la vista correspondiente, dentro de **views.py**, de la siguiente forma:

#### **views.py**

```
...
import uuid
...class IndexView(TemplateView):
    template_name = "app/index.html"

    def get_context_data(self, **kwargs):
        context = super(IndexView, self).get_context_data(**kwargs)
        context['form']=FormularioEspecial()
        return context
def post(self, request, *args, **kwargs):
    form_data = request.POST
    form = FormularioEspecial(form_data)
    print(form.is_valid())
    if form.is_valid():
        print('valido', form_data)
        guardar_a_archivo(form_data['nombre'],
                           form_data['email'],
                           'usuarios_registrados.txt')
        texto_archivo = leer_desde_archivo('usuarios_registrados.txt')
        return HttpResponse(
            "El formulario ha sido validado con éxito!!! \n\n"
            + "LOS USUARIOS REGISTRADOS HASTA AHORA SON:\n\n"
            + texto_archivo,
            content_type="text/plain;charset=utf-8")
    else:
        return render(request, self.template_name, {'form':form})
...
```

Podemos ver que en caso de haberse validado exitosamente los datos del formulario, ahora procedemos a invocar la función **guardar\_a\_archivo()** pasando como argumentos los campos **nombre** y **email** del objeto **form\_data**, que corresponde a la estructura de datos recibida con el método POST.

Si ejecutamos el código de nuestro proyecto e ingresamos datos de muchos usuarios que supuestamente se han deseado registrar en nuestra landing page, obtendremos un archivo **usuarios\_registrados.txt** como el siguiente:

#### **usuarios\_registrados.txt**

```
id: b3bdaad8-4e99-46e2-8449-78b0f5a77524 Nombre: Jericho Jorquera Email: jericho@hotmail.com
```



```
id: 6b11d4fc-36f4-46dc-bb75-028d4ffa77fb Nombre: Carolina Gonzalez Email: carogonza@gmail.com
id: 10584e92-76ad-4ee1-8940-b783fe0cee7a Nombre: Claudia Beda Email: claudiabejar@gmail.com
id: 77654e88-5d0b-492e-b2d4-23103e0c3ba2 Nombre: Samu Jara Email: samux@gmail.com
id: bf8661dd-e807-4738-bf12-147ee39d0c7e Nombre: Luis Jaramillo Email: ljaraque@hotmail.com
id: dd9e7e17-ea62-4f4f-b645-fbeb0aaa81cc Nombre: Anahis Leida Email: Anahis@gmail.com
id: 15dcf684-3abc-4221-9817-bc7da25526ea Nombre: Esteban Dido Email: estadido@gmail.com
id: f4af9c4e-11b7-4c7e-bd18-fd1bead1a318 Nombre: Esteban DiGiorgio Email: estedigorgio@gmail.com
id: 50d77c51-88a8-4538-a131-dcffd4f162b7 Nombre: Emilda Urrutiorrieta Email: Emilda@yandex.com
id: 4a219db9-0ace-4e61-bd4a-fb57d00f6ef6 Nombre: Rocio Jurado Email: rociodelmar@gmail.com
id: 78b699e8-7123-488b-9a30-74fd7cfa7d7 Nombre: jericho Email: a@b.cd
id: 94574c5a-906a-4d87-9328-0b4f2874c41f Nombre: Caupolicán Catrileo Email: caupolican@aol.com
id: 76415b7e-34e0-4638-9f11-e7223ca045cc Nombre: Eusebio Lillo Email: eusebio@uchile.cl
```

Los **id** de usuario han sido generados en la función **guardar\_a\_archivo()** a través del uso de **'id: ' + str(uuid.uuid4())**.

Adicionalmente a guardar los datos en el archivo, implementamos una respuesta simple del servidor para mostrar a través del navegador el estado actualizado de la lista de usuarios registrados cada vez que se ingresa uno nuevo.

De esta forma, luego de un envío de datos de registro a través del botón **Regístrate**, como el que se muestra a continuación:

Landing Page

0.0.0.0:8000/app/

## Nuestro Producto Estrella

**Regístrate dentro de los 100 primeros y recibirás un descuento**



Image copyrights to <https://www.cfg.com/>

Nombre

Joe Tempest

Email

ioe@europe.eu

Regístrate

**¡Nuestros Beneficios!**

- Rapidez
- Seguridad
- Gran Precio
- Flexibilidad

[Like Red Social 1](#) | [Like Red Social 2](#) | [Like Red Social 3](#) | [Like Red Social 4](#)

Copyright © 2019-2020 - Desarrollos de Landing Pages Inc.

se presentará la lista de usuarios de la siguiente forma:



```
Mozilla Firefox
0.0.0.0:8000/app/
El formulario ha sido validado con éxito!!!
LOS USUARIOS REGISTRADOS HASTA AHORA SON:
id: b3bdaad8-4e99-46e2-8449-78b0f5a77524 Nombre: Jericho Jorquera Email: jericho@hotmail.com
id: 6b11d4fc-36f4-46dc-bb75-028d4ffa77fb Nombre: Carolina Gonzalez Email: carogonza@gmail.com
id: 10584e92-76ad-4ee1-8940-b783fe0cee7a Nombre: Claudia Beda Email: claudiabejar@gmail.com
id: 77654e88-5d0b-492e-b2d4-23103e0c3ba2 Nombre: Samu Jara Email: samux@gmail.com
id: bf8661dd-e807-4738-bf12-147ee39d0c7e Nombre: Luis Jaramillo Email: ljaraque@hotmail.com
id: dd9e7e17-ea62-4f4f-b645-fbeb0aaa81cc Nombre: Anahís Leida Email: Anahis@gmail.com
id: 15dcf684-3abc-4221-9817-bc7da25526ea Nombre: Esteban Dido Email: estadido@gmail.com
id: f4af9c4e-11b7-4c7e-bd18-fd1bead1a318 Nombre: Esteban DiGiorgio Email: estedigiorgio@gmail.com
id: 50d77c51-88a8-4538-a131-dcfff4f162b7 Nombre: Emilda Urrutiorrieta Email: Emilda@yandex.com
id: 4a219db9-0ace-4e61-bd4a-fb57d00f6ef6 Nombre: Rocío Jurado Email: rociodelmar@gmail.com
id: 78b699e8-7123-488b-9a30-74fd7cfda7d7 Nombre: jericho Email: a@b.cd
id: 94574c5a-906a-4d87-9328-0b4f2874c41f Nombre: Caupolicán Catrileo Email: caupolican@aol.com
id: 76415b7e-34e0-4638-9f11-e7223ca045cc Nombre: Eusebio Lillo Email: eusebio@uchile.cl
id: 58727386-84e8-49f6-8204-ccf9169e6ad4 Nombre: Joe Tempest Email: joe@europe.eu
```

Hemos utilizado **HttpResponse** con texto plano y definido la codificación utf-8 con el fin de desplegar de manera correcta los caracteres especiales como los que poseen tilde.

### 5.3.1.9. Personalización de estilos en formularios Django

Si bien la renderización de formularios con el uso de la clase **Form** de Django facilita enormemente el manejo de éstos, debemos considerar que, a parte del formato por defecto proporcionado por Django, queremos personalizar los estilos asociados. Existen varios escenarios, de los cuales destacamos los siguientes:

1. Generar estilos para asignarlos a los atributos **class** de nuestros elementos HTML que contienen a **{{ form }}** en las plantillas intervenidas para uso en Django.
2. Definir estilos o asignar clases a los elementos HTML generados por la instanciación de las clases de tipo **Form** que creamos en nuestros proyectos. Para esto podemos incorporar el argumento **widget** dentro de la definición de campos en nuestra definición de clases de formularios.

El escenario 1 corresponde a métodos que hemos experimentado ampliamente a lo largo de los distintos apartados de estos contenidos. Por lo tanto, dejamos al lector la experimentación de esta vía para la personalización de estilos de formularios en Django.

El escenario 2 se implementa en el archivo **forms.py** y en la clase correspondiente, incorporando **widget** a la definición de cada campo, con lo que estaremos definiendo el valor de atributos HTML de los elementos que genere la clase **Form** al crear una



instancia según nuestra aplicación. Es aconsejable definir atributos **class** de HTML que determinen estilos que estén definidos en hojas de estilo de nuestro proyecto. En este caso, solo a modo ilustrativo, para no complejizar más nuestro ejemplo teniendo que agregar archivos CSS, incorporaremos directamente estilos con el atributo **style**. La sintaxis para esto, en nuestro archivo **forms.py** de la aplicación **app** es:

**forms.py**

```
from django import forms

class FormularioEspecial(forms.Form):

    nombre = forms.CharField(max_length=30, required=True,
                             error_messages={
                                 'required': 'Debe ingresar este dato',
                                 'max_length': 'El Nombre es muy largo'},
                             widget=forms.TextInput(
                                 attrs={'style' : 'color:blue; font-weight:bold;'}))

    email = forms.EmailField(
        required=True,
        error_messages={
            'required': 'No has ingresado tu email',
            'invalid': 'Éste no parece ser un email'},
        widget=forms.TextInput(
            attrs={
                'style' : 'color:red; font-weight:bold;'}))
```

Con lo cual hemos instruido a Django que la creación del formulario lo realice con texto en negrita para ambos campos, **nombre** y **email**, y con color de texto azul y rojo, respectivamente.

De esta manera al ingresar al formulario, veremos lo siguiente:

Nombre

**Alumno Programador**

Email

**alumno.p@miemail.cl**

Regístrate



Se sugiere al lector explorar y experimentar con personalización de estilos para formularios de distintos niveles de complejidad. En este caso, la intención principal ha sido ilustrar los conceptos básicos para manejo de formularios y su personalización visual.

### 5.3.2.- Referencias

[1] Django Documentation

<https://docs.djangoproject.com/en/3.1/>

[2] D. Feldroy & A. Feldroy , Two Scoops of Django 3.x, Best Practices for the Django Web Frame-work, 2020.

[3] William S. Vincent, Django for Beginners Build websites with Python & Django, 2020.

[4] Nigel George, Build a Website with Django 3, 2019.

[5] Django Tutorial

<https://www.geeksforgeeks.org/django-tutorial/>