

EXERCISES QUE TRABAJAREMOS EN EL CUE

- EXERCISE 1: RENDERIZAR FORMULARIOS HTML (GET Y POST).
- EXERCISE 2: USO DE FORMULARIOS EN DJANGO.
- EXERCISE 3: RENDERIZAR FORMULARIOS DE DJANGO MANUALMENTE.
- EXERCISE 4: WIDGETS PERSONALIZADOS DE CAMPO DE FORMULARIO DE DJANGO.
- EXERCISE 5: EL MODELFORM DE DJANGO.

EXERCISE 1: RENDERIZAR FORMULARIOS HTML (GET Y POST).

El objetivo del presente ejercicio es plantear una guía paso a paso para entender como renderizar los formularios en Django, su uso, personalización y el concepto básico del ModelForm.

Requisitos previos:

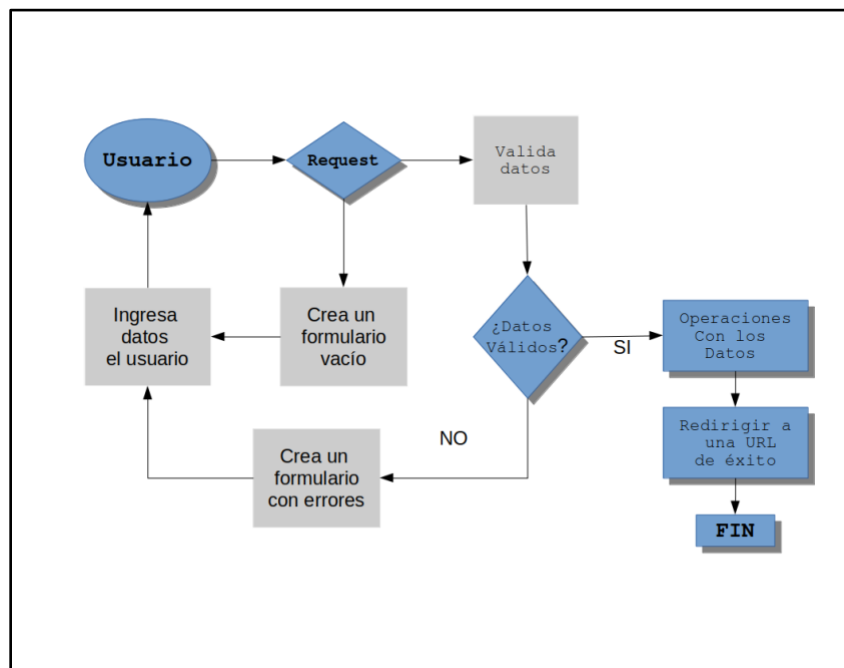
- Tener conocimiento de una terminal o línea de comando e instalación de paquetes de software en el sistema operativo, tanto en Windows 10 como en Linux (en este caso, Linux Ubuntu).
- En caso de Windows, se debe tener instalado el entorno virtual explicado en el Exercises del CUE03 y la configuración del proyecto en el CUE04.
- Tener previamente instalado la versión de Python 3 y el entorno virtual con (virtualenvwrapper).
- Hacer uso de la herramienta Visual Studio Code.

RENDERIZAR FORMULARIOS HTML (GET Y POST)

Al crear una clase de formulario, la parte más importante del mismo es definir los campos del formulario. Cada campo tiene una lógica de validación personalizada, junto con algunos otros conectores. En esta parte de la práctica veremos cómo utilizamos los campos del formulario, así como las técnicas y características relacionadas con Django Forms. Los formularios se usan básicamente para tomar información del usuario de alguna manera, y usarla para operaciones lógicas en las bases de datos. Por ejemplo: registro de datos del usuario como el nombre, correo electrónico, contraseña, entre otros.

Django mapea los campos definidos en los formularios de Django en campos de entrada HTML. Maneja tres partes distintas del trabajo relacionado con los formularios:

- Preparar y reestructurar datos para que estén listos para renderizar.
- Crear formularios HTML para los datos.
- Recibir y procesar formularios enviados y datos del cliente.



Los formularios en HTML son una colección de elementos dentro de `<form>...</form>`, que permite a un visitante hacer cosas como ingresar texto, seleccionar opciones, manipular objetos o controles, entre otros, y luego enviar esa información de vuelta al servidor. Básicamente, es una colección de datos para procesarlos para cualquier propósito, incluido guardarlos en la base de datos o recuperar datos de la base de datos. Django admite todos los tipos de formularios HTML, y la representación de datos de ellos en una vista para su procesamiento mediante varias operaciones lógicas.

Para comenzar con los formularios, se debe estar familiarizado con las solicitudes GET y POST en los formularios.

- GET: agrupa los datos enviados en una cadena y los usa para componer una URL. La URL contiene la dirección donde se deben enviar los datos, así como las claves y valores de los datos. Por ejemplo: <https://docs.djangoproject.com/search/?q=forms&release=1>.
- POST: cualquier solicitud que pueda usarse para cambiar el estado del sistema. Por ejemplo: una solicitud que realice cambios en la base de datos debe usar POST.

Para esta práctica vamos a considerar el entorno virtual `project_django`, y dentro creamos el proyecto `site_web_django` y una aplicación `boards`. Para abrir el proyecto nos ubicamos en el directorio `site_web_django`, y lo hacemos con VSC:

```
1 $ cd site_web_django
2 $ code .
```

Activamos el proyecto con `workon`:

```
1 $ workon projects_django
```

Ejecutamos la aplicación:

```
1 $ python manage.py runserver
```

Observamos en la terminal que se está ejecutando el servidor en `http://127.0.0.1:8000/`.

```
1 Watching for file changes with StatReloader
2 Performing system checks...
3
4 System check identified no issues (0 silenced).
5 July 10, 2022 - 20:35:21
6 Django version 4.0.5, using settings 'site_web_django.settings'
7 Starting development server at http://127.0.0.1:8000/
8 Quit the server with CONTROL-C.
```

Observamos que tenemos un contenido estático que muestra lo siguiente:



Para efectos prácticos, procedemos a crear un formulario HTML simple para mostrar cómo puede ingresar los datos de un usuario y usarlos en su vista. Creamos el archivo `datosform.html` dentro de `boards/templates`.

BOARDS/TAMPLATES/DATOSFORM.HTML

```
1 <form action = "" method = "get">
2   <label for="tu_nombre">Tu nombre: </label>
3   <input id="tu_nombre" type="text" name="tu_nombre">
4   <input type="submit" value="OK">
5 </form>
```

Seguidamente, procedemos a generar nuestra vista. Debemos modificar `urls.py` para la aplicación boards.

BOARDS/URLS.PY

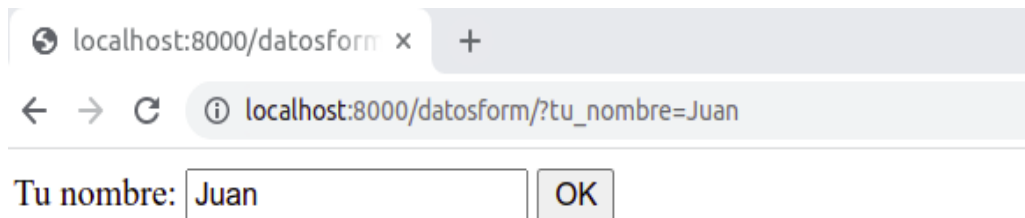
```
1 from django.urls import path
2 from .views import IndexPageView, obtenerFecha, menuView, mostrar,
3   datosform_view
4 urlpatterns = [
5     path('', IndexPageView.as_view(), name='index'),
6     path('fecha/<name>', obtenerFecha, name='index'),
7     path('menu/', menuView, name='menu'),
8     path('mostrar/', mostrar, name='mostrar'),
9     path('datosform/', datosform_view, name='datosform'),
```

Procedemos a crear nuestra vista `datosform_view`, y comenzamos a verificar cómo vamos a obtener los datos. Los datos completos de un formulario HTML en Django se transfieren como un objeto JSON llamado solicitud (request). Primero creamos una vista, y luego probaremos todos los métodos para obtener datos del formulario.

BOARDS/VIEWS.PY

```
1 from tokenize import PseudoExtras
2 from django.views.generic import TemplateView
3 from django.shortcuts import render
4 import datetime
5
6 def datosform_view(request):
7     # la logica de la vista se implementa aqui
8     return render(request, "datosform.html")
```

Ejecutamos el servidor, y verificamos el formulario en: <http://localhost:8000/datosform/>



The screenshot shows a web browser window with a single tab titled 'localhost:8000/datosform x'. The address bar displays the URL 'localhost:8000/datosform/?tu_nombre=Juan'. Below the address bar, the form content is visible, starting with the label 'Tu nombre:' followed by a text input field containing the value 'Juan' and an 'OK' button.

Por defecto, todos los formularios escritos en HTML realizan una solicitud GET en el back-end de una aplicación, la solicitud GET normalmente funciona mediante consultas en la URL. Colocar el nombre en el campo y presionar OK, observamos que se adjunta el nombre a la variable `tu_nombre`, esto es: http://localhost:8000/datosform/?tu_nombre=Juan. Así es como funciona la solicitud GET, independientemente del número de entradas que se agregarían a la URL para enviar los datos al back-end de una aplicación. Para visualizar estos datos, modificamos la vista con la finalidad de obtener el valor de la función de entrada.

BOARDS/VIEWS.PY

```
1 from django.views.generic import TemplateView
2 from django.shortcuts import render
3 import datetime
4
5 def datosform_view(request):
6     # la logica de la vista se implementa aquí
7     print(request.GET)
8     return render(request, "datosform.html")
```

Observamos en la terminal:

```
PROBLEMS 3 OUTPUT DEBUG CONSOLE TERMINAL
Django version 4.0.5, using settings 'site_web_django.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CONTROL-C.
<QueryDict: {'tu_nombre': ['Juan']}>
[01/Aug/2022 00:03:10] "GET /datosform/?tu_nombre=Juan HTTP/1.1" 200 172
```

El método `request.GET` devuelve un diccionario de consulta al que se puede acceder como cualquier otro diccionario de Python, y finalmente usar sus datos para aplicar alguna lógica. De manera similar, si el método de transmisión es POST, puede usar `request.POST` como diccionario de consulta para representar los datos del formulario en vistas.

BOARDS/TEMPLATES/DATOSFORM.HTML

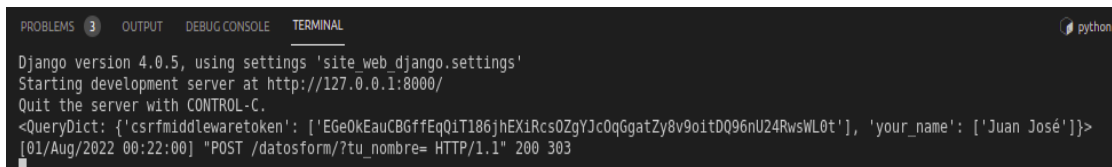
```
1 <form action="" method="POST">
2     {% csrf_token %}
3     <label for="your_name">Your name: </label>
4     <input id="your_name" type="text" name="your_name" />
5     <input type="submit" value="OK" />
6 </form>
```

Tenga en cuenta que cada vez que creamos una solicitud de formulario, Django requiere que agregue `{% csrf_token %}` en el formulario por motivos de seguridad. Modifiquemos la vista con el `request.POST` en el archivo `views.py`.

BOARDS/VIEWS.PY

```
1 def datosform_view(request):
2     print(request.POST)
3     return render(request, "datosform.html")
```

Observamos la salida en la terminal de VSC:



```
PROBLEMS 3 OUTPUT DEBUG CONSOLE TERMINAL python
Django version 4.0.5, using settings 'site.web.django.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CONTROL-C.
<QueryDict: {'csrfmiddlewaretoken': ['EGe0kEauCBGffEqQiT186jhEXiRcs0ZgYJc0qGgatZy8v9oitDQ96nU24RwsWL0t'], 'your_name': ['Juan José']}>
[01/Aug/2022 00:22:00] "POST /datosform/?tu_nombre= HTTP/1.1" 200 303
```

De esta manera, se pueden usar estos datos para consultar la base de datos o para procesarlos usando alguna operación lógica y pasarlos usando el diccionario de contexto a la plantilla.

EXERCISE 2: USO DE FORMULARIOS EN DJANGO.

Crear un formulario en Django es completamente similar a crear un modelo, se necesita especificar qué campos existirían en el formulario y de qué tipo. Por ejemplo: para ingresar un formulario de registro, es posible que necesite Nombre (**CharField**), Número de lista (**IntegerField**), entre otros.

Procedemos a crear un formulario en `boards/forms.py`, con el siguiente código:

BOARDS/FORMS.PY

```
1 # importar la librería estándar de Django Forms
2 from django import forms
3
4 # Creación de un forms
5 class InputForm(forms.Form):
6     nombres = forms.CharField(max_length = 200)
7
8     apellidos = forms.CharField(max_length = 200)
```

```
9
10     prioridad = forms.IntegerField(min_value=1, max_value=3)
11
12     contrasenna = forms.CharField(widget = forms.PasswordInput())
```

Seguidamente, procedemos a renderizar los campos del formulario.

Los campos de formulario de Django tienen varios métodos integrados para facilitar el trabajo del desarrollador, pero a veces es necesario implementar cosas manualmente para personalizar la interfaz de usuario (UI). Un formulario viene con 3 métodos incorporados que se pueden usar para representar los campos de formulario de Django.

- `{{form.as_table}}` los representará como celdas de tabla envueltas en etiquetas `<tr>`.
- `{{form.as_p}}` los renderizará envueltos en etiquetas `<p>`.
- `{{form.as_ul}}` los renderizará envueltos en etiquetas ``.

Procedemos a crear vista el formulario `views.py`.

BOARDS/VIEWS.PY

```
1 from django.views.generic import TemplateView
2 from django.shortcuts import render
3 import datetime
4 from .forms import InputForm
5
6
7 def datosform_view(request):
8     context = {}
9     context['form'] = EntradaForm()
10    return render(request, "datosform.html", context)
```

En la vista se necesita crear una instancia de la clase formulario, ya creada anteriormente en `forms.py`. Procedemos adecuar la plantilla.

BOARDS/TEMPLATES/DATOSFORM.HTML

```
1 <form action="" method="post">
2
3     {% csrf_token %}
4     {{form }}
5
6     <input type="submit" value=Enviar">
7 </form>
```

EXERCISE 3: RENDERIZAR FORMULARIOS DE DJANGO MANUALMENTE.

El formulario funciona correctamente, pero las imágenes son decepcionantes. Podemos representar estos campos manualmente para mejorar algunas cosas visuales. Cada campo está disponible como un atributo del formulario usando `{{form.name_of_field}}`, y en una plantilla de Django, se representará apropiadamente. Modificamos el formulario `datosform.html`.

BOARDS/TEMPLATES/DATOSFORM.HTML

```
1 <html>
2   <head>
3     <link
4       rel="stylesheet"
5       href="https://stackpath.bootstrapcdn.com/bootstrap/4.4.1/css/bootstrap.min
6       .css"/>
7
8     <style>
9       .i-am-centered {
10         margin: auto;
11         max-width: 300px;
12         padding-top: 20%;
13       }
14     </style>
15   </head>
16
17   <body>
18     <div class="i-am-centered">
19       <form method="POST">
20         {% csrf_token %}
21         <div class="form-group">
```

```
22         <label>Nombres </label>
23         {{ form.nombres }}
24     </div>
25     <div class="form-group">
26         <label>Apellidos </label>
27         {{ form.apellidos }}
28     </div>
29     <div class="form-group">
30         <label>Prioridad</label>
31         {{ form.prioridad }}
32     </div>
33     <div class="form-group">
34         <label>Contraseña:</label>
35         {{ form.contrasenna }}
36     </div>
37     <button type="submit" class="btn btn-primary">Enviar</button>
38 </form>
39 </div>
40 </body>
41
42 </html>
```

Estas fueron solo algunas modificaciones básicas usando Bootstrap. Pueden ser personalizadas a un nivel avanzado usando varios trucos y métodos de CSS.

EXERCISE 4: WIDGETS PERSONALIZADOS DE CAMPO DE FORMULARIO DE DJANGO.

Un widget es la representación de Django de un elemento de entrada HTML. El widget maneja la representación del HTML y la extracción de datos de un diccionario **GET/POST**, que corresponde al widget. Cada vez que especifique un campo en un formulario, Django usará un widget predeterminado que sea apropiado para el tipo de datos que se mostrarán. Para encontrar qué widget se usa en qué campo, consulte la documentación sobre las clases de campo incorporadas. Esta publicación trata sobre el uso avanzado de widgets para modificar la estructura del formulario y el tipo de entrada.

Cada campo tiene un widget predefinido. Por ejemplo, **IntegerField** tiene un widget predeterminado de **NumberInput**. Demostremos esto con la ayuda de nuestro proyecto base `site_web_django` y la aplicación `boards`. Actualizamos el formulario:

BOARDS/FORMS.PY

```
1 # importar la librería estándar de Django Forms
2 from django import forms
3
4 class WidgetForm(forms.Form):
5     nombre = forms.CharField()
6     apellido = forms.CharField()
7     prioridad = forms.IntegerField()
8     habilitado = forms.BooleanField()
```

Ahora, para representar este formulario, creamos la vista y la plantilla que se usarán para mostrar el formulario al usuario. En `boards/views.py`, crea una vista.

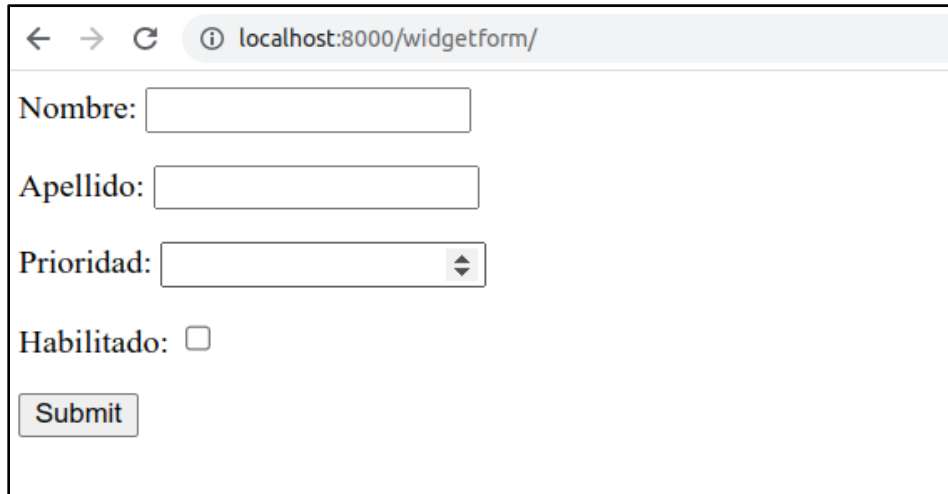
BOARDS/VIEWS.PY

```
1 from django.shortcuts import render
2 from .forms import WidgetForm
3
4 def widget_view(request):
5     context = {}
6     form = WidgetForm(request.POST or None)
7     context['form'] = form
8     return render(request, "widget_home.html", context)
```

BOARDS/TEMPLATES/WIDGET_HOME.HTML

```
1 <form method="POST">
2     {% csrf_token %} {{ form.as_p }}
3     <input type="submit" value="Submit"/>
4 </form>
```

Ejecutamos el servidor y verificamos la URL: <http://localhost:8000/widgetform/>



Como se ve en la captura de pantalla anterior, hay un tipo diferente de campo de entrada para **IntegerField**, **BooleanField**, etc. Se puede modificar esto de las siguientes maneras:

Se puede anular el widget predeterminado de cada campo para varios propósitos. La lista de widgets se puede ver en la [Documentación Django](#). Para anular el widget predeterminado, debemos definir explícitamente el widget que queremos asignar a un campo. Por ejemplo:

BOARDS/FORMS.PY

```
1 # importar la librería estándar de Django Forms
2 from django import forms
3
4 class WidgetForm(forms.Form):
5     nombre = forms.CharField(widget = forms.Textarea)
6
7     apellido = forms.CharField(widget = forms.CheckboxInput)
8
9     prioridad = forms.IntegerField(widget = forms.TextInput)
10
11     habilitado = forms.BooleanField(widget = forms.Textarea)
12
13     nombre = forms.CharField()
```

Por lo tanto, podemos asignar cualquier widget a cualquier campo usando el widget atributo.

Nota: las validaciones impuestas a los campos seguirán siendo las mismas. Por ejemplo, incluso si un IntegerField se hace igual que CharField, solo aceptará entradas de enteros.

Los widgets tienen un gran uso en los campos de formulario, especialmente cuando se usa Seleccionar tipo de widgets, donde uno quiere limitar el tipo y la cantidad de entradas de un usuario. Demostremos esto con la ayuda de modificar **DateField**.

BOARDS/FORMS.PY

```
1 # importar la librería estándar de Django Forms
2 from django import forms
3
4 class WidgetForm(forms.Form):
5     nombre = forms.CharField()
6     apellido = forms.CharField()
7     prioridad = forms.IntegerField()
8     habilitado = forms.BooleanField()
9     date = forms.DateField()
```

Ejecutamos el servidor, y verificamos la URL: <http://localhost:8000/widgetform/>

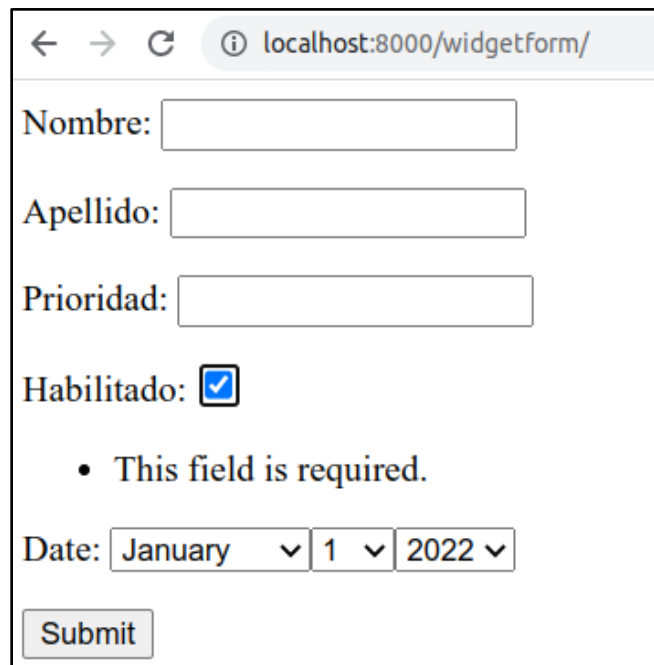


Ahora cambiemos el widget para una entrada mejor y más conveniente del usuario de una fecha. Se agrega el `SelectDateWidget` a `DateField` en `forms.py`.

BOARDS/FORMS.PY

```
1 # importar la librería estándar de Django Forms
2 from django import forms
3
4 class WidgetForm(forms.Form):
5     nombre = forms.CharField()
6     apellido = forms.CharField()
7     prioridad = forms.IntegerField()
8     habilitado = forms.BooleanField()
9     date = forms.DateField(widget = forms.SelectDateWidget)
```

Ejecutamos el servidor, y verificamos la URL: <http://localhost:8000/widgetform/>



← → ↻ ⓘ localhost:8000/widgetform/

Nombre:

Apellido:

Prioridad:

Habilitado: ☒

- This field is required.

Date:

EXERCISE 5: EL MODELFORM DE DJANGO.

Django `ModelForm` es una clase que se utiliza para convertir directamente un modelo en un formulario Django. Si está creando una aplicación basada en una base de datos, es probable que tenga formularios que se correspondan estrechamente con los modelos de Django. Procedemos a crear el modelo.

BOARDS/MODEL.PY

```
1 from django.db import models
2
3 # Declaramos el nuevo modelo con el nombre de BoardsModel"
4 class BoardsModel(models.Model):
5     # Campos del modelo
6     titulo = models.CharField(max_length = 200)
7     descripcion = models.TextField()
8     modificado = models.DateTimeField(auto_now_add = True)
9
10 def __str__(self):
11     return self.titulo
```

Antes de crear un modelo, registremos nuestra aplicación en el proyecto principal. En el archivo `site_web_django/settings.py`, agregue la aplicación boards en la lista `INSTALLED_APPS`. Si hacemos migraciones antes de este paso, se nos mostrará un mensaje indicando que no hay cambios realizados.

Ahora, ejecute los siguientes comandos para crear el modelo:

```
1 python manage.py makemigrations
2 python manage.py migrate
```

Para crear un formulario directamente para este modelo, acceda a `boards/forms.py` e introduzca el siguiente código.

BOARDS/FORMS.PY

```
1 # importar la librería estándar de Django Forms
2 from django import forms
3
4 # import BoardsModel from models.py
5 from .models import BoardsModel
6
7 # Crear el BoardsForm
8 class BoardsForm(forms.ModelForm):
9     # specify the name of model to use
10     class Meta:
11         model = BoardsModel
12         fields = "__all__"
```

- **fields**: se recomienda que se establezcan explícitamente todos los campos que deben editarse en el formulario utilizando el atributo de campos. Si no lo hace, puede generar fácilmente problemas de seguridad cuando un formulario permite inesperadamente que un usuario establezca ciertos campos, especialmente cuando se agregan nuevos campos a un modelo. Dependiendo de cómo se represente el formulario, es posible que el problema ni siquiera sea visible en la página web.
- Establezca el atributo de campos en el valor especial `'__all__'`, para indicar que se deben usar todos los campos del modelo.
- **exclude**: establezca el atributo de exclusión de la metaclass interna de ModelForm en una lista de campos que se excluirán del formulario. Por ejemplo

```
1 class EjemploForm(ModelForm):
2     class Meta:
3         model = Persona
4         exclude = ['direccion']
```

Finalmente, para completar nuestra estructura MVT, se crea una vista que represente el formulario y lo guarde directamente en la base de datos.

BOARDS/VIEWS.PY

```
1 from django.http import HttpResponseRedirect
```

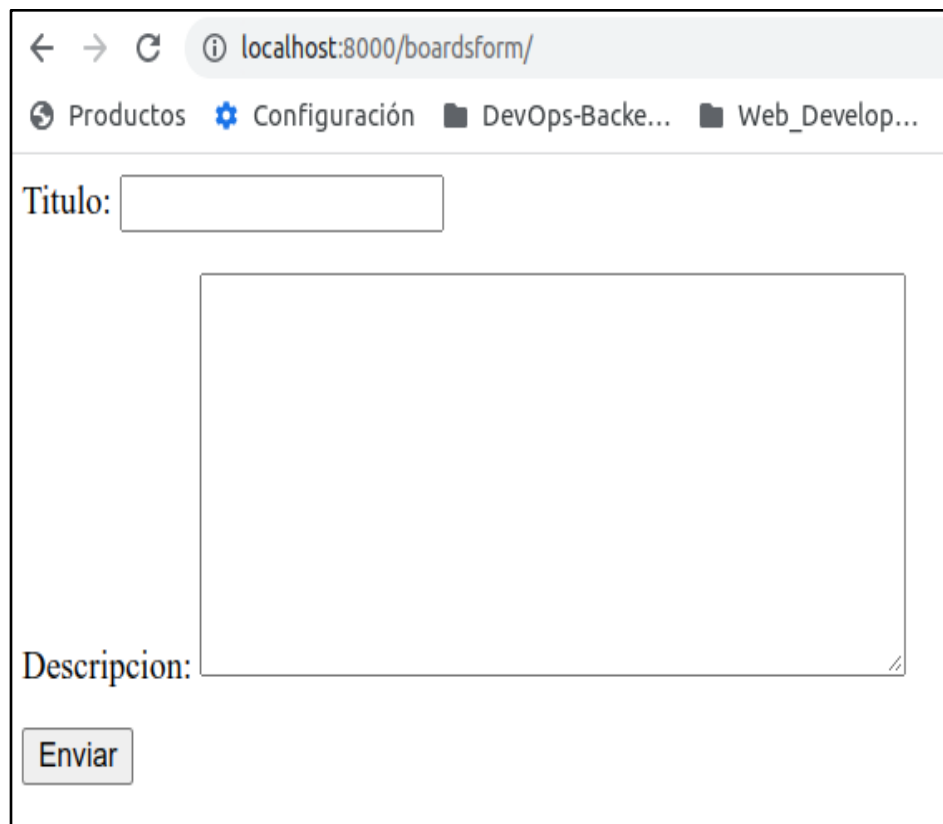


```
2 from django.views.generic import TemplateView
3 from django.shortcuts import render
4 import datetime
5 from .forms import EntradaForm, BoardsForm
6
7 def boardsform_view(request):
8     context = {}
9     # crear el objeto form
10    form = BoardsForm(request.POST or None, request.FILES or None)
11    # verificar si el formulario es valido
12    if form.is_valid():
13        # guardar los datos del formulario al modelo
14        form.save()
15        return HttpResponseRedirect('/')
16    context['form'] = form
17    return render(request, "datosform.html", context)
```

BOARDS/URLS.PY

```
1 from django.urls import path
2
3 from .views import IndexPageView, obtenerFecha, menuView, mostrar,
4 datosform_wiew, widget_view, boardsform_view
5
6 urlpatterns = [
7     path('', IndexPageView.as_view(), name='index'),
8     path('fecha/<name>', obtenerFecha, name='index'),
9     path('menu/', menuView, name='menu'),
10    path('mostrar/', mostrar, name='mostrar'),
11    path('datosform/', datosform_wiew, name='datos_form'),
12    path('widgetform/', widget_view, name='widgetform'),
13    path('boardsform/', boardsform_view, name='boardsform'),
```

<http://localhost:8000/boardsform/>



← → ↻ ⓘ localhost:8000/boardsform/

🔍 Productos ⚙ Configuración 📁 DevOps-Backe... 📁 Web_Develop...

Titulo:

Descripcion: