DESARROLLO CON PYTHON

Contenido de formación


Programación


Pyhton


Paradigma POO


Bases de datos SQL


Tkinter


Django


Flask

1. Primeros Instalación de SW
2. Variables y tipos de datos
3. Operadores (aritméticos, asignación)
4. Entrada y salida de datos
5. Estructuras de control
   5.1 Condicionales (operadores lógicos, operadores de comparación)
   5.2 Bucles (estructuras interactivas for, while)
6. Bloque de ejercicios de aplicación de conceptos
7. Funciones (parámetros, return, invocación, lambda)
   7.1 Variables locales y globales, funciones y métodos predefinidos
8. Lista y tuplas (creación, índices, recorrer y mostrar listas, listas multidimensionales)
9. Diccionarios y sets
10. Bloque de ejercicios aplicación de conceptos
11. Módulos y paquetes (creación y funcionalidad)
12. Sistemas de archivos y directorios
13. Manejo de errores (captura de excepciones, errores personalizados)
14. Programación orientada a objetos
15. Bases de datos SQLite
16. Bases de datos MySQL
17. Proyecto con Python
18. Interfaces graficas con Tkinter (aplicación de escritorio Python – Tkinter)
19. Desarrollo Web con Django
20. Interfaces graficas con Flask (aplicación de escritorio Python – Flask)

Formularios, crear vistas y urls

1. En el proceso de intentar definir una función, hay un parámetro llamado solicitud que se utiliza para recibir la información solicitada:
La información solicitada incluye: encabezado / cuerpo
Solicitud de clasificación:

get	Obtenga, desee que el servidor solicite el texto sin formato del recurso. obtener solicitud de? Inicio, clave es igual a valor, separados por &
post	Enviar, el método se utiliza para la transmisión física
head	Similar al método get. Solo no se devolverá el cuerpo de la respuesta, generalmente se usa para confirmar la validez de la URL y el momento de la actualización de recursos
put	Generalmente se usa para cargar archivos
delete	Especificar para eliminar un elemento
options	Método de soporte para consultar el recurso URL especificado
trace	El cliente puede rastrear la ruta de transmisión del mensaje de solicitud de esta manera
connect	Se requiere crear un túnel cuando se comunica con el servidor proxy para realizar la comunicación del protocolo TCP utilizando el protocolo de túnel.

2. En el desarrollo web, la mayoría de los datos se envían al servidor a través del formulario
Realizamos una copia de la vista crear_articulo

```python
def crear_articulo(request, title, content, public):
    articulo = Article(
        title = title, #title(nombre del modelo) = title(nombre de la variable)
        content = content,
        public = public
    )
    #Para guardar este artículo en la BD
    articulo.save()
    return HttpResponse(f"Artículo creado: {articulo.title} - {articulo.content}")
```

y hacemos estos cambios a lo pegado

```python
    return HttpResponse(f"Artículo creado: {articulo.title} - {articulo.content}")

#Copia de crear_articulo y lo llamamos save_articulo
def save_article(request):# No pasamos parametros por url y lo vamos a pasar por formulario
    if request.method == 'POST':
        title = request.POST['title']
        if len(title) < 5:
            return HttpResponse(f"El titulo es mu pequeño")
        content = request.POST['content']
        public = request.POST['public']
        articulo = Article(
            title = title, #title(nombre del modelo) = title(nombre de la variable)
            content = content,
            public = public
        )
        #Para guardar este articulo en la BD
        articulo.save()
        return HttpResponse(f"Artículo creado: <strong> {articulo.title} - {articulo.content} </strong>")
    else:
        return HttpResponse(f"<h2> No se ha podido crear el articulo </h2>")


def create_article(request): #soporte para plantilla para visualizar el formulario
    return render(request, 'create_article.html')
```

Ahora creamos la template para la vista lo llamamos create_articulo.html

views.py | layout.html | create_article.html | articulos.html | urls.py

22-django > AprendiendoDjango > miapp > templates > create_article.html

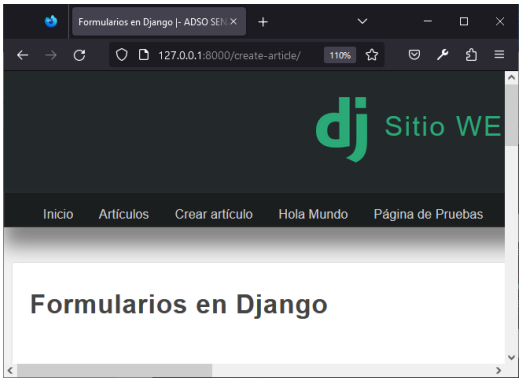
```
1  {% extends 'layout.html' %}
2  {% block title %} Formularios en Django {% endblock %}
3  {% block content %}
4  <h1 class="title">Formularios en Django</h1>
5
6
7  {% endblock  %}
```

Creamos las url

```
40      path('borrar-articulo/<int:id>', views.borrar_articulo, name="borrar"),
41      #ruta para formularios
42      path('save-article/',views.save_article, name="save"),
43      path('create-article/', views.create_article, name= "create"),
44  ]
```

Agregamos la vista a la barra de navegación layout.html

```
25          <nav>
26              <ul>
27                  <li>
28                      <a href="{% url 'inicio' %}">Inicio</a>
29                  </li>
30                  <li>
31                      <a href="{% url 'articulos' %}">Artículos</a>
32                  </li>
33                  <li>
34                      <a href="{% url 'create' %}">Crear artículo</a>
35                  </li>
36                  <li>
37                      <a href="{% url 'hola_mundo' %}">Hola Mundo</a>
38                  </li>
39                  <li>
40                      <a href="{% url 'pagina' %}">Página de Pruebas</a>
41                  </li>
42
43              </ul>
44          </nav>
```



create_article.html

22-django > AprendiendoDjango > miapp > templates > create_article.html

```
1  {% extends 'layout.html' %}
2  {% block title %} Formularios en Django {% endblock %}
3  {% block content %}
4  <h1 class="title">Formularios en Django</h1>
5
6  <form action="/save-article/" method="GET">
7      <!--Campo titulo-->
8      <label for="title">Titulo</label>
9      <input type="text"name="title" />
10     <!--Campo contenido-->
11     <label for="content">Contenido</label>
12     <textarea name="content"> </textarea>
13     <!--Campo publicado-->
14     <label for="public">Publicado</label>
15     <select name="public">
16         <option value="1">Si</option>
17         <option value="0">No</option>
18     </select>
19     <!--Botón-->
20     <input type="submit" value="Guardar" />
21 </form>
22 {% endblock  %}
```

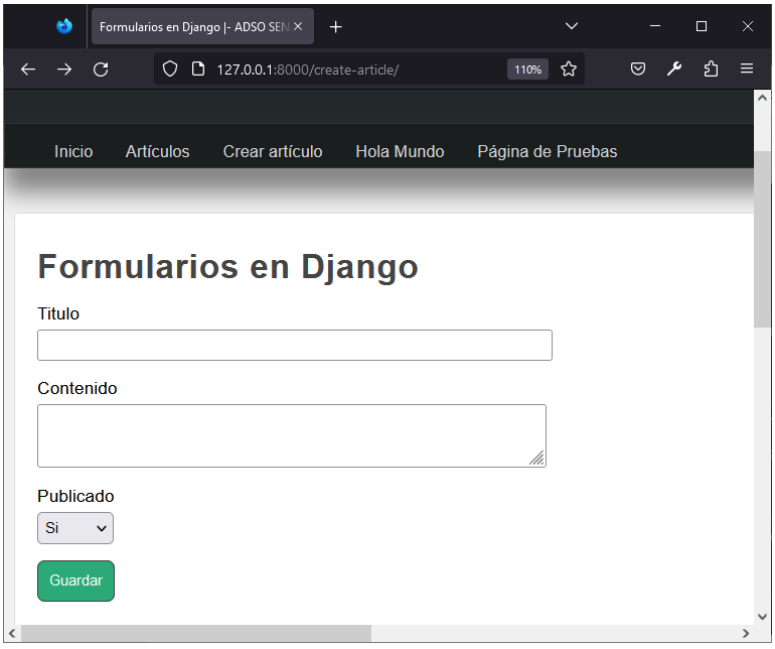
Estilos CSS

En nuestro proyecto tenemos ya un archivo en **static → css → styles.css**

Y vamos a **.box** donde tendremos el contenido del formulario

```css
.box{
    background: white;
    min-height: 930px;
    width: 95%;
    padding: 20px;
    border: 1px solid #ddd;
    border-radius: 2px;
    margin: 0 auto;
}
/*Estilos para formulario*/
.box form{
    width: 40%;/*ancho del formulario*/
}
/*los controles ocupan una linea*/
.box form input,
.box form label{
    display: block;
    padding: 5px;
    padding-left: 0px;
}
/*ancho de los controles*/
.box form input[type="text"],
.box form textarea,
.box form select{
    width: 100%;
    margin-bottom: 10px;
}
```

```css
    margin-bottom: 10px;
}
/*slect*/
.box form select{
    width: 70px;
    padding: 5px;
}
/*Boton*/
.box form input[type="submit"],
.box form input[type="buttom"],
.box form button{
    padding: 10px;
    margin-top: 5px;
    background: #2ba977;
    border: 1px solid #444;
    color: white;
    transition: 300ms all;
    border-radius: 7px;
}
/* Color del boton cuando pase por encima */
.box form input[type="submit"]:hover,
.box form input[type="buttom"]:hover,
.box form button:hover{
    cursor: pointer;
    background: #1f7e58;
}
/*Estilos para title*/
```



Recibir datos del formulario por GET

Vamos a **views.py** y vamos a la función **def save_article(request)** línea 95
Y rellenamos las variables y comprobamos si nos llegan datos por GET

```python

#Copia de crear_articulo y lo llamamos save_articulo
def save_article(request):# No pasamos parametros por url y lo vamos a pasar por formulario
    if request.method == 'GET':
        title = request.GET['title']
        content = request.GET['content']
        public = request.GET['public']
        articulo = Article(
            title = title, #title(nombre del modelo) = title(nombre de la variable)
            content = content,
            public = public
        )
        #Para guardar este articulo en la BD
        articulo.save()
        return HttpResponse(f"Artículo creado: <strong> {articulo.title} - {articulo.content} </strong>")
    else:
        return HttpResponse(f"<h2> No se ha podido crear el articulo </h2>")


def create_article(request): #soporte para plantilla para visualizar el formulario
```

views.py ...\pages | views.py ...\miapp | create_article.html | articulos.html

22-django > AprendiendoDjango > miapp > views.py > articulos

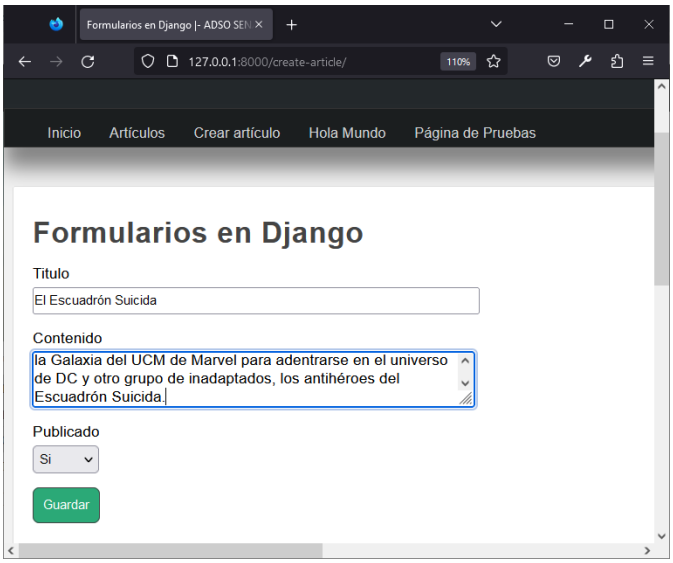
```python
#Mostrar articulos
def articulos(request):#creacion de la vista articulos
    #Creamos una template para listar varos articulos
    #1. hacemos la peticion a la base de datos
    #Creamos la variable articulos y LLamamos al modelo  Article, usamos objects para porder hacer una consulta
    articulos = Article.objects.all().order_by('-id') #En este caso no usamos el metodo get si no el metodo all()
    #para sacar toda la información
    #dentro de articulos tenemos una lista de objetos un array de objetos
    """
    #Consultas con condiciones filter para filtrar por un valor especifico
    articulos = Article.objects.filter(title__contains = "Tiburón")

    #Realizar una consulta que muestre solo los esten publicados y excluya bajo una condición
    articulos = Article.objects.filter(title = "Tiburón",).exclude(public=True)

    #Consulta ejecutando SQL
    articulos = Article.objects.raw("SELECT * FROM miapp_article WHERE title='Tiburón' AND public=1")

    #Consulta utilizando el OR con ORM
    articulos = Article.objects.filter(
        Q(title__contains="John") | Q(title__contains="Señor")
    )
    """
    return render(request, 'articulos.html', { #pasamos el request y la template articulos.html y como tercer parametro
        'articulos' : articulos #pasamos un diccionario con las variables que deseamos mostrar
    })
```

articulos.html

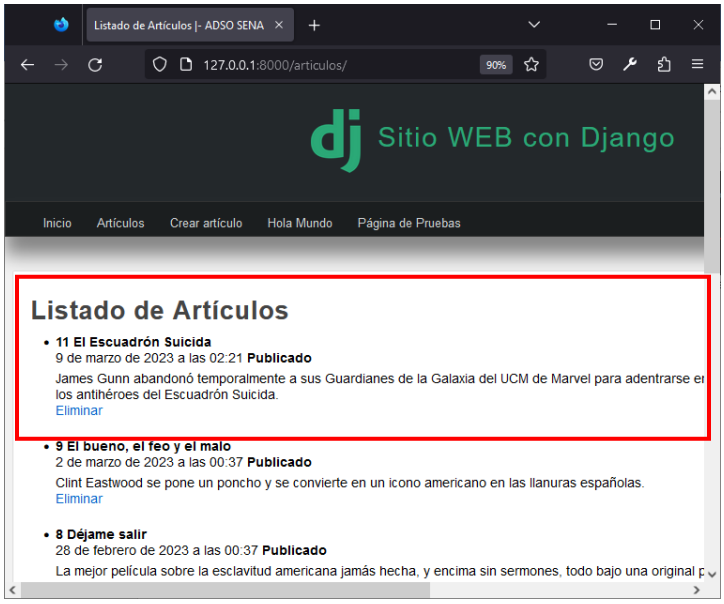
22-django > AprendiendoDjango > miapp > templates > articulos.html

```
{% extends 'layout.html' %}
{% block title %} Listado de Artículos {% endblock %}
{% block content %}
<h1 class="title">Listado de Artículos</h1>
{% if messages %}
    {% for message in messages %}
        <div class="message">
            {{message}}
        </div>
    {% endfor %}
{% endif %}
    <ul>
        {% for articulo in articulos %}
            <li>
                <h4>{{articulo.id}} {{articulo.title}}</h4>
                <span>{{articulo.created_at}}</span>
                {% if articulo.public %}
                    <strong>Publicado</strong>
                    {% else %}
                    <strong>Privado</strong>
                {% endif %}
                <p>
                    {{articulo.content}}
                    <br/>
                    <a href = "{% url 'borrar' id=articulo.id %}">Eliminar</a>
                </p>
                <br/>
            </li>
        {% endfor %}
    </ul>
{% endblock  %}
```

Llenamos el formulariocon crear articulo



Observamos el resultado de lista en la Artículos



Recoger datos del formulario por POST

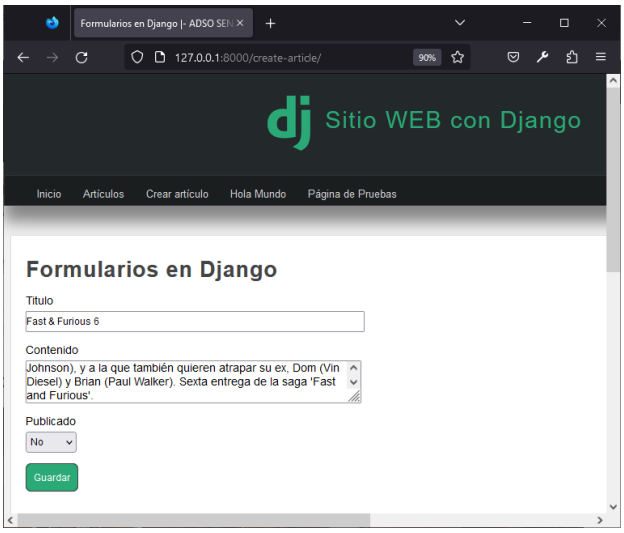
create_article.html

```
views.py | create_article.html | urls.py
22-django > AprendiendoDjango > miapp > templates > create_article.html
1  {% extends 'layout.html' %}
2  {% block title %} Formularios en Django {% endblock %}
3  {% block content %}
4  <h1 class="title">Formularios en Django</h1>
5
6  <form action="{% url 'save' %}" method="POST">
7      {% csrf_token %}
8      <!--Campo titulo-->
9      <label for="title">Titulo</label>
```

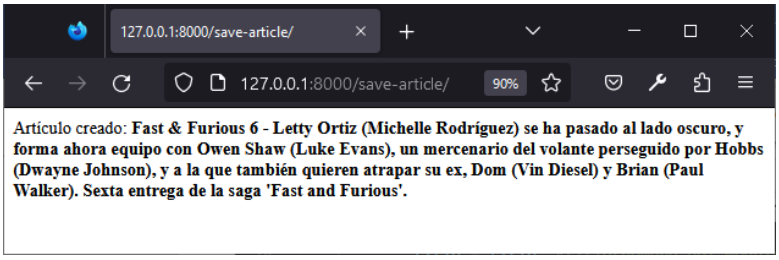
views.py

```
67  #Copia de crear_articulo y lo llamamos save_articulo
68  def save_article(request):# No pasamos parametros por u
69      if request.method == 'POST':
70          title = request.POST['title']
71          content = request.POST['content']
72          public = request.POST['public']
73          articulo = Article(
74              title = title, #title(nombre del modelo) =
75              content = content,
```

Agregamos información al formulario



Aquí se puede evidenciar que no se muestra la información enviada por la url



Formularios basados en clases:

Si se está construyendo una aplicación que gestiona una base de datos, lo más apropiado es usar los modelos ya declarados como formularios, y así evitar estar repitiendo las mismas reglas para procesar los datos.

Por esta razón, Django provee una clase de ayuda que permite crear un formulario a partir de un modelo, esta clase se llama ModelForm, y se emplea así (forms.py):

Creamos a nivel de **miapp** un fichero que se llamara **forms.py.** En este fichero importamos ***from django import forms*** Definimos las clases de la siguiente manera: recomendación que la clase se llame igual o similar a como se llama el modelo ***(class Article(models.Model)) .*** Como concepto es importante conocer los diferentes field que existen para los diferentes controles o propiedades
https://docs.djangoproject.com/en/4.1/ref/forms/fields/

creación del formulario

forms.py | views.py | create_article.html | create_full_article.html

22-django > AprendiendoDjango > miapp > forms.py > FormArticle

```python
from django import forms

class FormArticle(forms.Form): #Hereda de la clase forms

    #creamos las propiedades
    title = forms.CharField(
        label= "Titulo"
    )
    content = forms.CharField(
        label= "Contenido",
        widget=forms.Textarea
    )
```

En **views.py** para nuestro ejemplo vamos a crear un método ***def create_full_article***
Para crear este método debemos importar el formulario que hemos creado en forms.py para poder usar el objeto

forms.py | views.py | urls.py

22-django > AprendiendoDjango > miapp > views.py > ...

```python
from django.shortcuts import render, HttpResponse, redirect
from miapp.models import Article
from django.db.models import Q
from miapp.forms import FormArticle #Importamos el formulario desde forms.py

```

Y mas abajo creamos el método

```python
def create_article(request): #soporte para plantilla para visualizar el
    return render(request, 'create_article.html')

def create_full_article(request):
    #creamos una variable llamada formulario para instanciar el objeto
    formulario = FormArticle()
    #se carga la vista en html
    return render(request, 'create_full_article.html', {
        'form' : formulario
    })

def articulo(request):
```

El siguiente paso será crear la vista html **create_full_article.html** en template y podemos copiar todo el código o la plantilla de **create_article.html** y se sustituye algunas cosas como se muestra a continuación:

forms.py | views.py | create_full_article.html | urls.py

22-django > AprendiendoDjango > miapp > templates > create_full_article.html

```
1  {% extends 'layout.html' %}
2  {% block title %} Formularios en Django {% endblock %}
3  {% block content %}
4  <h1 class="title">Formularios en Django</h1>
5
6  <form action="" method="POST">
7      {% csrf_token %}
8
9      <!--aqui mostramos el Formularios mandando a llamar
10         la variable form que se creo en la vista-->
11     {{ form }}
12
13     <!--Botón-->
14     <input type="submit" value="Guardar" />
15 </form>
16 {% endblock  %}
```

Ahora creamos la ruta

```
43      path('create-article/', views.create_article, name= "create"),
44      path('create-full-article/', views.create_full_article, name="create_full")
45  ]
```

Y comprobamos los cambios en la **ruta create-full-article/**

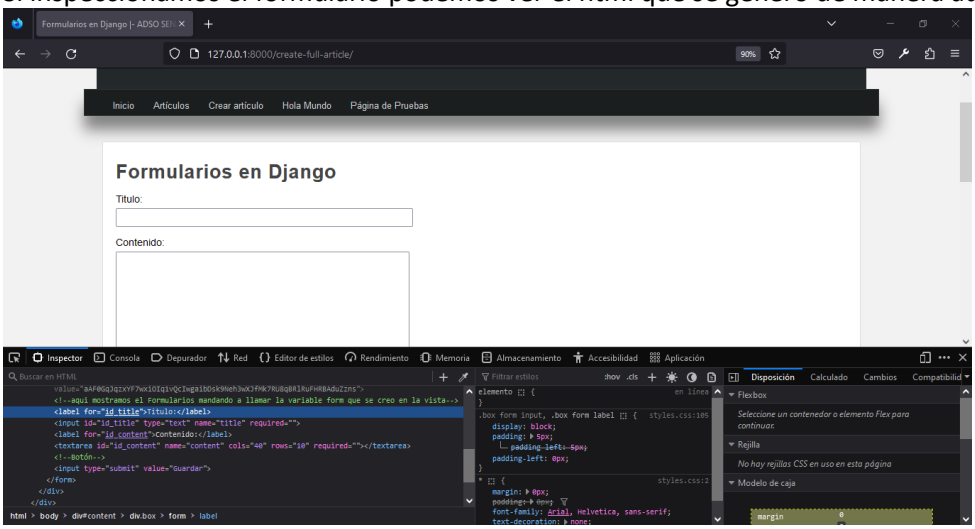
Hay muchos otros tipos de campos de formulario, que reconocerá en gran medida por su similitud con las clases de campo de modelo

equivalentes: *BooleanField*, *CharField*, *ChoiceField*, *TypedChoiceField*, *DateField*, *DateTimeField*, *DecimalField*, *DurationField*, *EmailField*, *FileField*, *FilePathField*, *FloatField*, *ImageField*, *IntegerField*, *GenericIPAddressField*, *MultipleChoiceField*, *TypedMultipleChoiceField*, *NullBooleanField*, *RegexField*, *SlugField*, *TimeField*, *URLField*, *UUIDField*, *ComboField*, *MultiValueField*, *SplitDateTimeField*, *ModelMultipleChoiceField*, *ModelChoiceField*.

Los argumentos que son comunes a la mayoría de los campos se enumeran a continuación (estos tienen valores predeterminados sensibles):

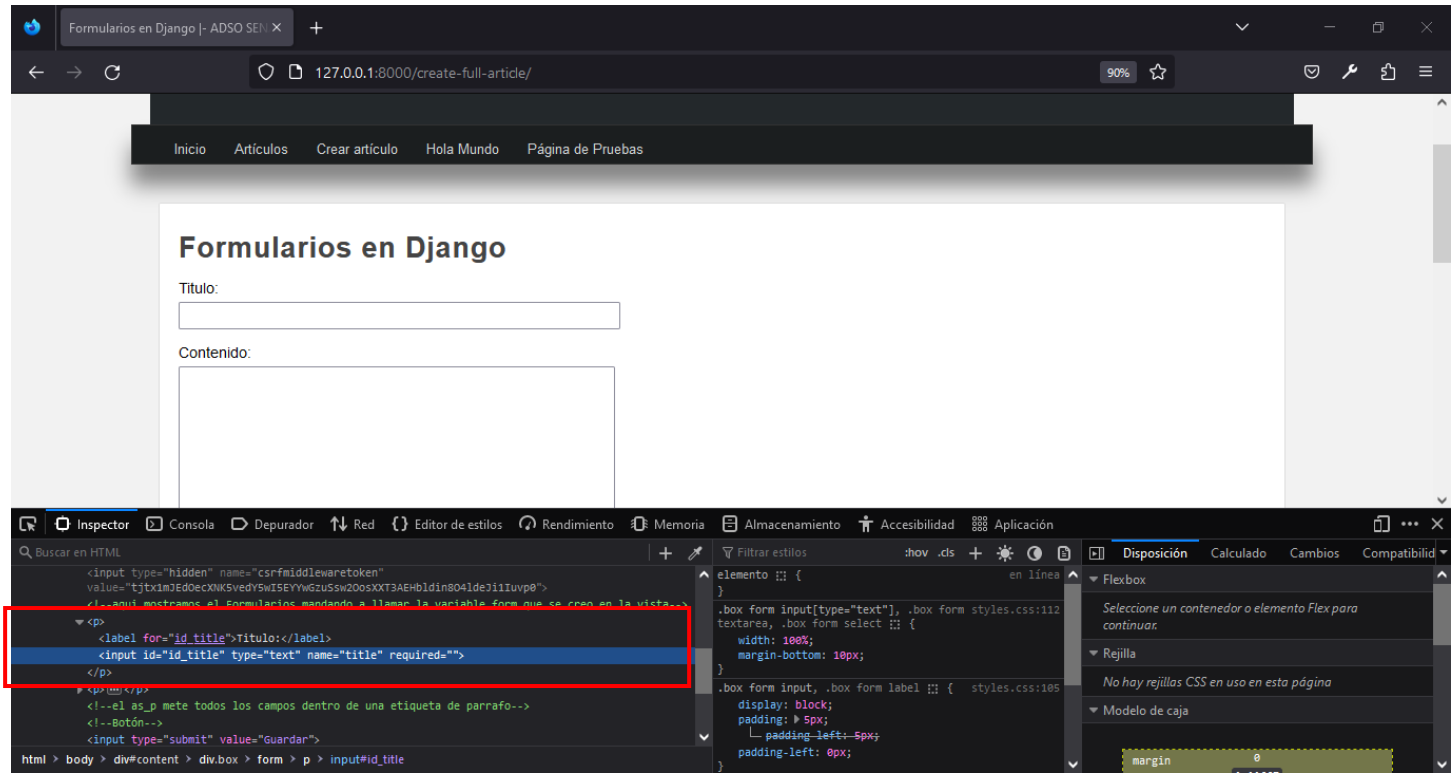
- required: Si es True, el campo no se puede dejar en blanco o dar un valor None. Los Campos son obligatorios por defecto, también puedes establecer required=False para permitir valores en blanco en el formulario.
- label: label es usado cuando renderizamos el campo en HTML. Si label no es especificado entonces Django crearía uno a partir del nombre del campo al poner en mayúscula la primera letra y reemplazar los guiones bajos por espacios (por ejemplo. *Renewal date*).
- label_suffix: Por defecto, se muestran dos puntos después de la etiqueta (ejemplo. Renewal date:). Este argumento le permite especificar como sufijo diferente que contiene otros caracteres.
- initial: El valor inicial para el campo cuando es mostrado en el formulario.
- widget: El widget de visualización para usar.
- help_text (como se ve en el ejemplo anterior): texto adicional que se puede mostrar en formularios para explicar cómo usar el campo.
- error_messages: Una lista de mensajes de error para el campo. Puede reemplazarlos con sus propios mensajes si es necesario.
- validators: Una lista de funciones que se invocarán en el campo cuando se valide.
- localize: Permite la localización de la entrada de datos del formulario (consulte el enlace para obtener más información).
- disabled: El campo se muestra, pero su valor no se puede editar si esto es True. Por defecto es False.

Ahora se puede generar el formulario de diferentes formas, por ejemplo:
Si inspeccionamos el formulario podemos ver el html que se genero de manera automatizada

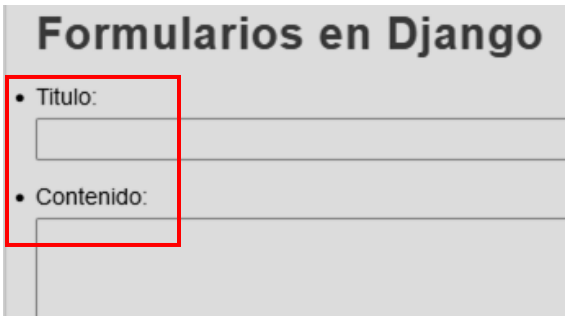


Podemos mostrar el formulario y se quiere que todos los campos estén bajo una etiqueta en concreto lo que se puede hacer es:

```
6   <form action="" method="POST">
7       {% csrf_token %}
8
9       <!--aqui mostramos el Formularios mandando a llamar
10          la variable form que se creo en la vista-->
11      {{ form.as_p }} <!-- el as_p mete todos los campos dentro de una etiqueta de parrafo-->
12
13      <!--Botón-->
```




```
11      {{ form.as_ul }} <!-- el as_p mete todos los campos dentro de una etiqueta de lista-->
```




```
<li>
    ::marker
    <label for="id_content">Contenido:</label>
    <textarea id="id_content" name="content" cols="40" rows="10" required=""></textarea>
</li>
```

Campo tipo select

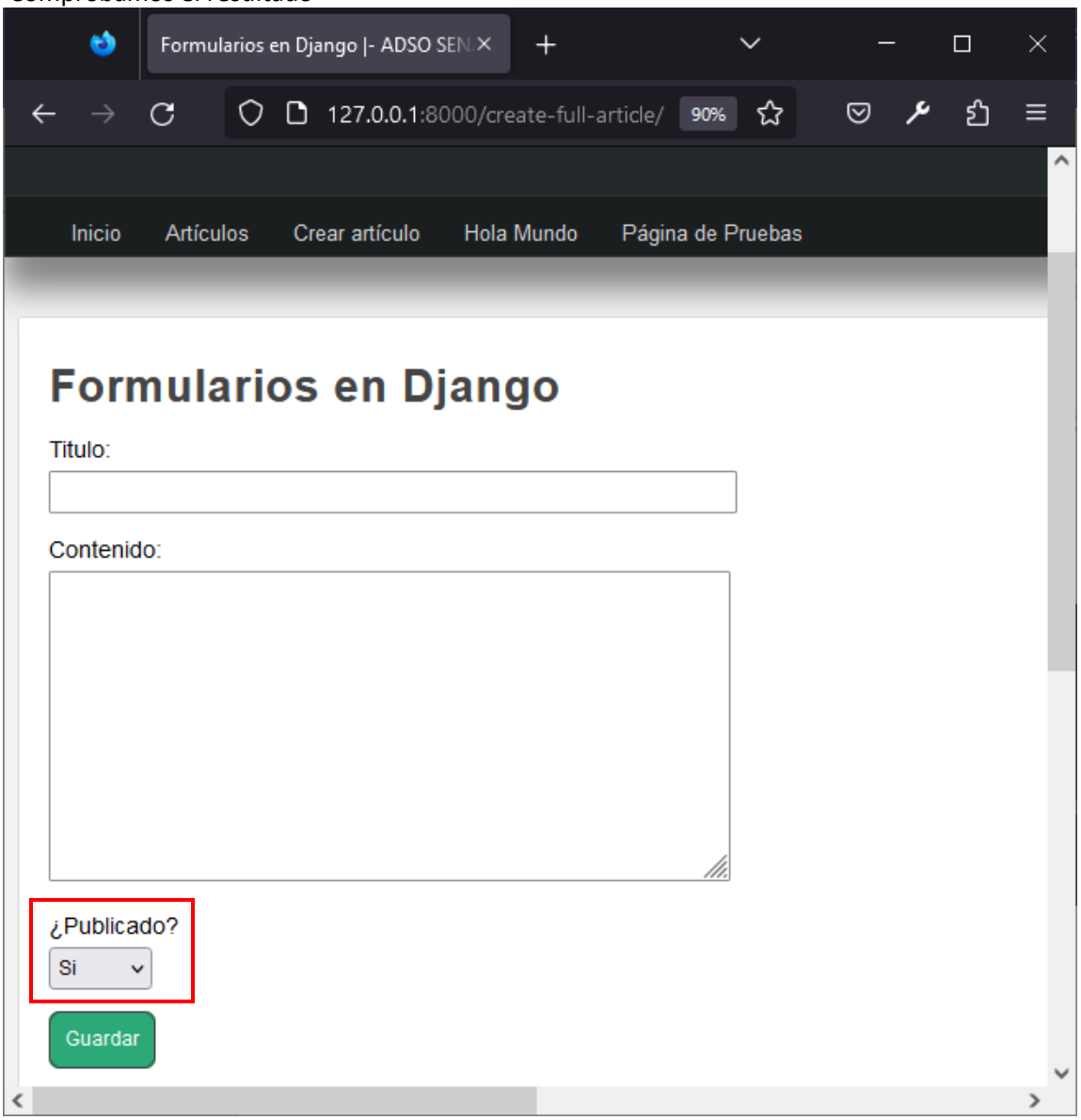
Dejamos el form.as_p

Creamos el public que son las opciones y las integramos a un choices

forms.py

```
11          widget=forms.Textarea
12      )
13      #con choices se pueden pasar una serie de opciones
14      public_options = [
15          (1, 'Si'),
16          (0, 'No')
17      ]
18      public = forms.TypedChoiceField(#permite mostrar un campo select y pasarle las opciones anteriores
19          label = "¿Publicado?",
20          choices = public_options
21         )
```

Comprobamos el resultado



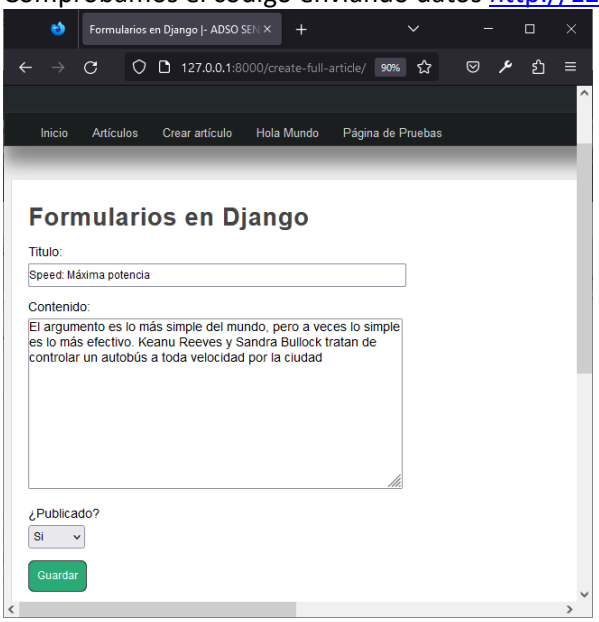
Recibir datos y guardar el formulario(Django Form API)

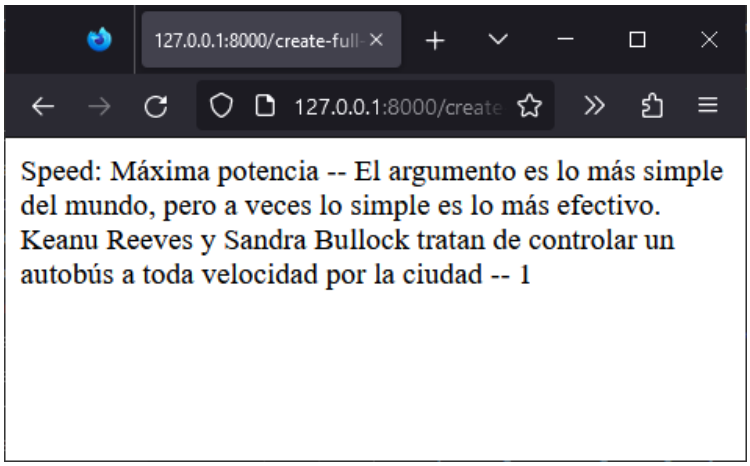
Para lo cual vamos a la vista ***def create_full_article*** y realizamos los siguientes cambios, comprobamos si nos envían datos por el formulario mediante que método (POST ó GET)

views.py

```python
def create_article(request): #soporte para plantilla para visualizar el formulario
    return render(request, 'create_article.html')

def create_full_article(request):
    #Realizamos la comprobacion del metodo (POST-GET)
    if request.method == 'POST':
        #si llega datos por POST se debe:
        formulario = FormArticle(request.POST)
        #aqui podemos validar el formulario con un metodo is_valid
        if formulario.is_valid():
            #generamos una variable para recoger los datos del formulario
            data_form = formulario.cleaned_data #que son los datos limpios que nos llegan
            title = data_form.get('title') #lo puedo hacer asi ó
            content = data_form['content'] # lo puedo hacer asi
            public = data_form['public']
            return HttpResponse(title + ' -- ' + content + ' -- ' + str(public))

    else:
        #si nos llegan datos por POST debemos generar un formulario vacio
        formulario = FormArticle() #creamos una variable llamada formulario para instanciar el objeto
        #se carga la vista en html
        return render(request, 'create_full_article.html', {
            'form' : formulario
        })

def articulo(request):
```

Comprobamos el código enviando datos http://127.0.0.1:8000/create-full-article/ **y aquí como se reciben**

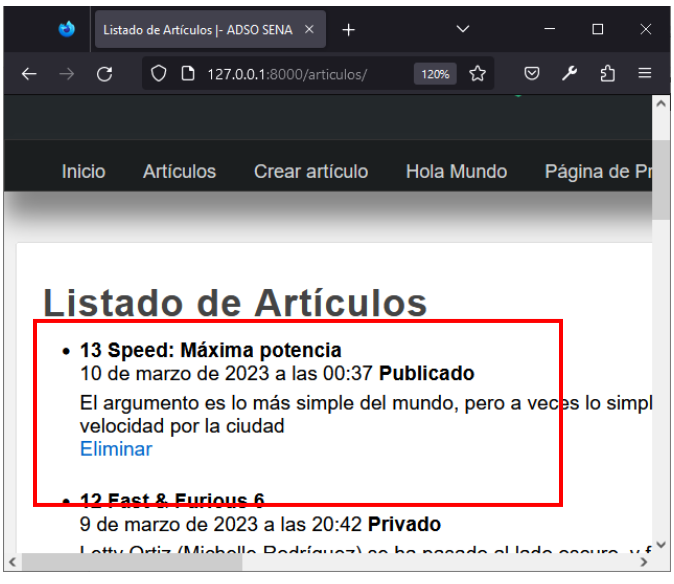


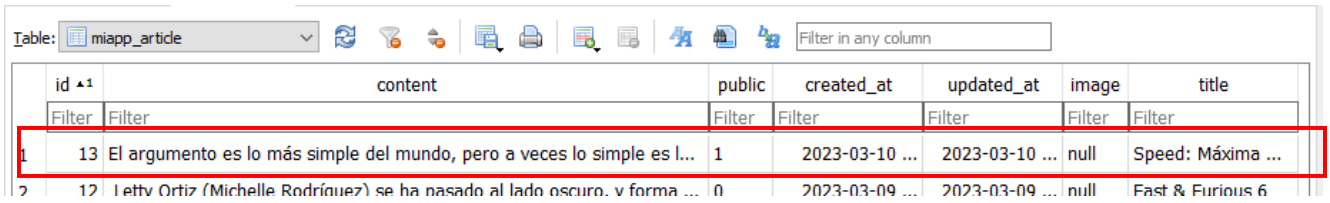


Ahora como se guardar esta información

Se puede utilizar por el momento el modelo de articulo y le paso las variables que se han recogido .

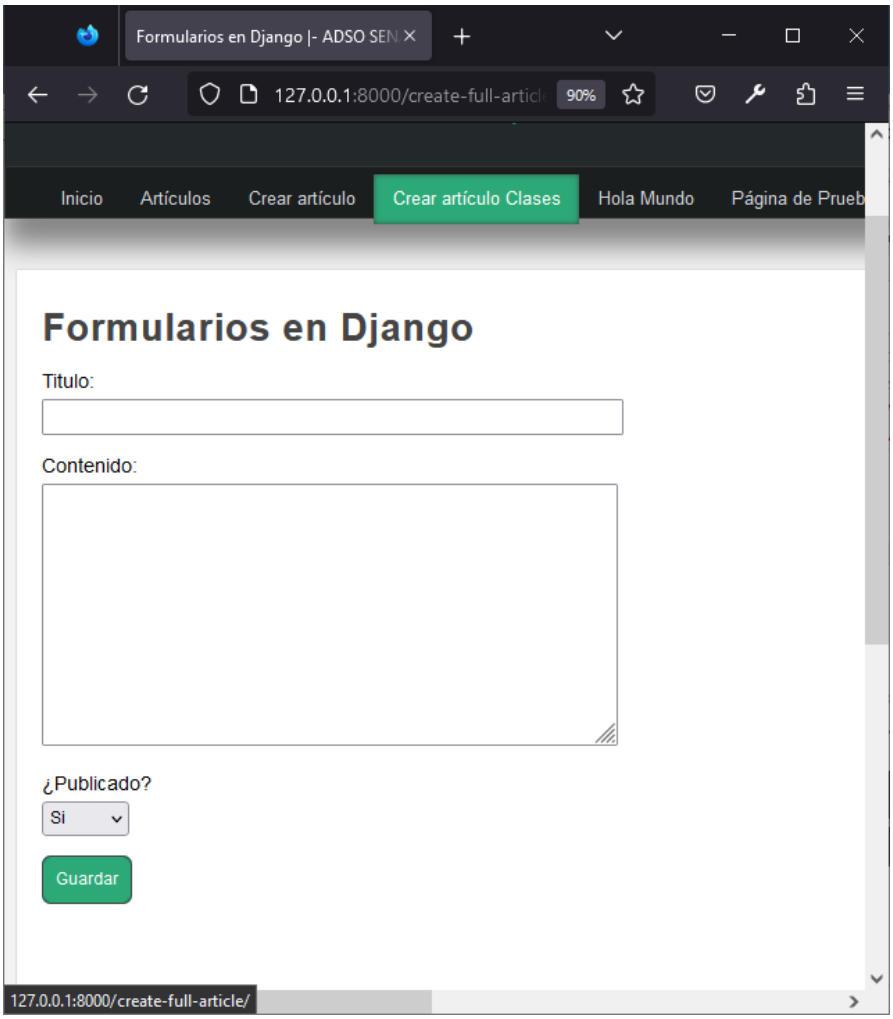
```python
def create_full_article(request):
    #Realizamos la comprobacion del metodo (POST-GET)
    if request.method == 'POST':
        #si llega datos por POST se debe:
        formulario = FormArticle(request.POST)
        #aqui podemos validar el formulario con un metodo is_valid
        if formulario.is_valid():
            #generamos una variable para recoger los datos del formulario
            data_form = formulario.cleaned_data #que son los datos limpios que nos llegan
            title = data_form.get('title') #lo puedo hacer asi ó
            content = data_form['content'] # lo puedo hacer asi
            public = data_form['public']

            articulo = Article(
                title = title, #title(nombre del modelo) = title(nombre de la variable)
                content = content,
                public = public
                )
            #Para guardar este articulo en la BD
            articulo.save()
            #rediccionar la pagina a ariculos despues de guardado
            return redirect('articulos')
            #return HttpResponse(articulo.title + ' -- ' + articulo.content + ' -- ' + str(articulo.public))

    else:
```





Se puede crear en layout otra nueva entrada para que aparezca en la barra de menú el nuevo tipo de formulario con clases.

```
    <li>
        <a href="{% url 'create_full' %}">Crear artículo Clases</a>
    </li>
```



Validación de formularios

Vamos a comprobar si el formulario trae un error → **create_full_article.html**

Importamos validators → **from django.core import validators**
Documentación https://docs.djangoproject.com/en/4.1/ref/validators/
Importar en forms.py la librería de **validators**

forms.py | views.py | layout.html | create_full_article.html | styl

22-django > AprendiendoDjango > miapp > forms.py > FormArticle

```python
from django import forms
from django.core import validators

class FormArticle(forms.Form): #Hereda de la clase forms

```

forms.py | views.py | layout.html | create_full_article.html | styles.css | urls.py

22-django > AprendiendoDjango > miapp > templates > create_full_article.html

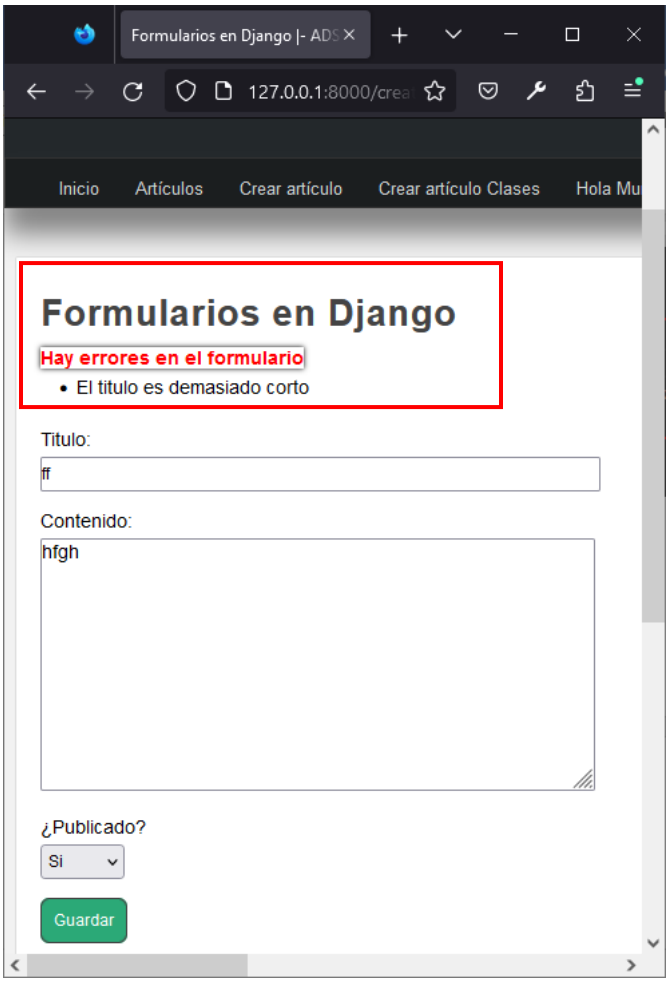
```
{% extends 'layout.html' %}
{% block title %} Formularios en Django {% endblock %}
{% block content %}
<h1 class="title">Formularios en Django</h1>
{% if form.errors %}
    <strong class="rojo">
        Hay errores en el formulario
    </strong>
{% endif %}
<form action="" method="POST">
    {% csrf_token %}

    <!--aqui mostramos el Formularios mandando a llamar
        la variable form que se creo en la vista-->
    {{ form.as_p }} <!-- el as_p mete todos los campos dentro de una etiqueta de parrafo-->

    <!--Botón-->
    <input type="submit" value="Guardar" />
</form>
{% endblock  %}
```

Estilos de clase rojo

```
    background: #1f7e58;
}
/* estilos para clase rojo de validadcion */
.rojo{
    color: red;
    box-shadow: 0px 0px 4px black;
}
/*Estilos para title*/
.title{
```

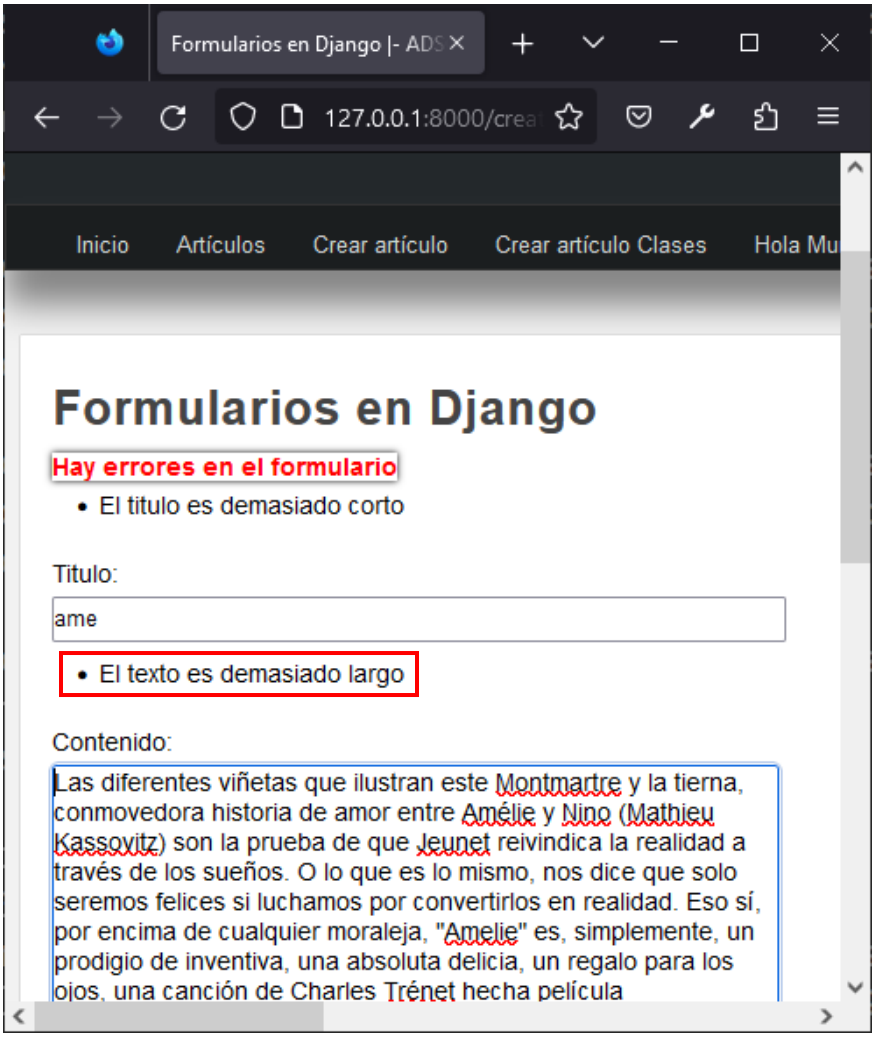
Validaciones en **forms.py** aquí se valida el primer campo del título `'^[A-Za-z-9ñÑáéíóúÁÉÍÓÚ ]*$'`
Todas las validaciones de acuerdo al alfabeto español que no las posee el alfabeto ingles

```python
    #creamos las propiedades
    title = forms.CharField(
        label= "Titulo",
        max_length=20,
        required=True,
        widget=forms.TextInput(
            attrs={
                'placeholder':'Digite el titulo',
                'class': 'titulo_form_article'
            }
        ),
        validators=[
            validators.MinLengthValidator(4, 'El titulo es demasiado corto'),
            validators.RegexValidator('^[A-Za-z-9]*$', 'El titulo esta mal formado','invalid_title')#expresion regular

        ]
    )
    content = forms.CharField(
```



Validación del **content**

```
19              validators.RegexValidator('^[A-Za-z-9 ]*$', 'El titulo esta mal for
20
21          ]
22      )
23      content = forms.CharField(
24          label= "Contenido",
25          widget=forms.Textarea,
26          validators=[
27              validators.MaxLengthValidator(20, 'El texto es demasiado largo')
28
29          ]
30
31      )
32      #con choices se pueden pasar una serie de opciones
```



Mensajes flash / sesiones flash

Vamos **views.py** e importamos la librería ***from django.contrib import messages***

views.py

22-django > AprendiendoDjango > miapp > views.py > ...

```
1   from django.shortcuts import render, HttpResponse, redirect
2   from miapp.models import Article
3   from django.db.models import Q
4   from miapp.forms import FormArticle #Importamos el formulario desde f
5   from django.contrib import messages #libreria mensajes flash
6   layout = """
```

Todo eso se hace en la redirección después de la validación

```
106                 )
107             #Para guardar este articulo en la BD
108             articulo.save()
109             #Crear mensaje flash(sesión que solo se muestra 1 vez)
110             messages.success(request, f'Ha creado correctamente el articulo {articulo.id}')
111
112             #rediccionar la pagina a ariculos despues de guardado
113             return redirect('articulos')
```

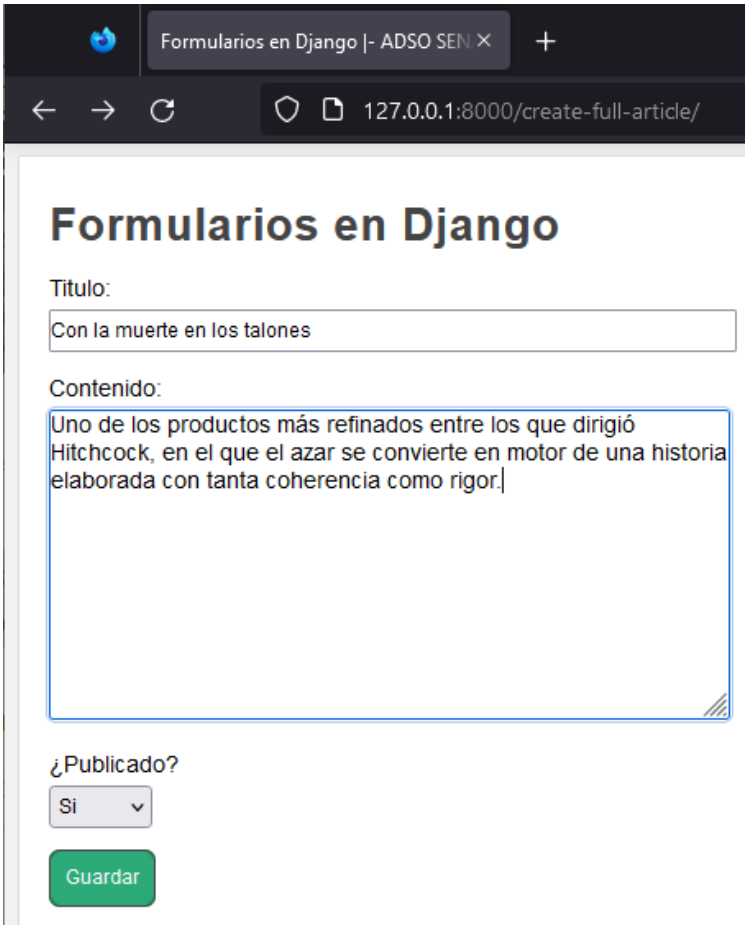
Y para mostrar el mensaje se hace en la vista artículos.html

views.py | articulos.html

22-django > AprendiendoDjango > miapp > templates > articulos.html

```
1   {% extends 'layout.html' %}
2   {% block title %} Listado de Artículos {% endblock %}
3   {% block content %}
4   <h1 class="title">Listado de Artículos</h1>
5   {% if messages %}
6       {% for message in messages %}
7           <div class="messages">
8               {{message}}
9           </div>
10      {% endfor %}
11  {% endif %}
12      <ul>
13          {% for articulo in articulos %}
14              <li>
15                  <h4>{{articulo.id}} {{articulo.title}}</h4>
16                  <span>{{articulo.created_at}}</span>
17                  {% if articulo.public %}
18                      <strong>Publicado</strong>
19                      {% else %}
20                      <strong>Privado</strong>
21                  {% endif %}
22                  <p>
23                      {{articulo.content}}
24                      <br/>
25                      <a href = "{% url 'borrar' id=articulo.id %}">Elimin
```

Probamos el código: recordemos que las sesiones flash solo duran una recarga de pantalla y desaparecen.



Colocar estilos a la clase messages