Contenido

Aplicación Contact

1. Crear aplicación “contact”
2. Crear la estructura de templates y el archivo contact.html
3. Del archivo contac.html estático, copiar el html requerido
4. Crear el archivo forms.py y dentro de el, crear la clase ContactForm

```python
from django import forms

class ContactForm(forms.Form):
    name = forms.CharField( label='Nombre',
                            required=True,
                            widget=forms.TextInput(
                                attrs={
                                    'class'         :'form-control',
                                    'palceholder'   :'Escribe tu nombre'
                                }
                            ),
                            min_length=3,
                            max_length=100
                          )
    email = forms.EmailField(
                            label='Email',
                            required=True,
                            widget=forms.EmailInput(
                                attrs={
                                    'class'         :'form-control',
                                    'palceholder'   :'Escribe tu email'
                                }
                            ),
                            min_length=3,
                            max_length=100

                          )
    content = forms.CharField( label='Contenido',
                            required=True,
                            widget=forms.Textarea(
                                attrs={
                                    'class'         :'form-control',
                                    'rows'          : '4',
                                    'palceholder'   :'Escribe tu nombre'
                                }
                            ),
                            min_length=3,
                            max_length=100
                          )
```

5. En el archivo views.py crear la clase ContactForm

```python
from django.shortcuts import render
from django.http import HttpResponseRedirect
from .forms import ContactForm

def contact(request):
    if request.method == 'POST':
        form = ContactForm(request.POST)
        if form.is_valid():
            print(form.cleaned_data.get('name'))
            print(form.cleaned_data['email'])
            return HttpResponseRedirect('/contact/thanks/')
    else:
        form = ContactForm()
    return render(request, 'contact/contact.html', {'form':form})

def thanks(request):
    return render(request, 'contact/thanks.html')
```

6. Crear el archivo urls.py

```python
from django.urls import path
from contact import views

contact_urlpatterns = ([
    path('', views.contact, name='contact'),
    path('thanks/', views.thanks, name='thanks'),
], 'contact')
```

7. Modificar el archivo contact.html para adecuarlo al formulario

```html
<!-- Formulario de contacto -->
<form action='' method='post'>
  {% csrf_token %}
  {{form}}
  <div class="text-center">
    <input type="submit" class="btn btn-primary btn-block py-2" value="Enviar">
  </div>
</form>
<!-- Fin formulario de contacto -->
```

8. Crear el archivo thanks.html

```
{% extends "core/base.html" %}
{% block content %}
{% load static %}

<section class="page-section about-heading">
    <div class="container">
      <div class="about-heading-content mbtm">
        <div class="row">
          <div class="col-xl-9 col-lg-10 mx-auto">
            <div class="bg-faded rounded p-5 forced">
              <h2 class="section-heading mb-4">
                <span class="section-heading-lower">Gracias!!</span>
              </h2>
              <div class="section-content">
                <p>
                    Nos podremos en contacto contigo lo antes posible.
                </p>
                <p>
                    La Recova.
                </p>
              </div>
            </div>
          </div>
        </div>
      </div>
    </section>

{% endblock %}

```

9. Agregar la aplicación

```
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'django_cleanup',
    'ckeditor',
    'core',
    'blog',
    'pages',
    'services',
    'contact',
]
```

10. Modificar el archivo urls.py del proyecto

```python
from services.urls import services_urlspatterns
from contact.urls import contact_urlpatterns

urlpatterns = [
    path('admin/', admin.site.urls),
    path('', include(core_urlpatterns)),
    path('blog/', include(post_urlpatterns)),
    path('pages/', include(pages_urlpatterns)),
    path('services/', include(services_urlspatterns)),
    path('contact/', include(contact_urlpatterns)),
]
```

11. Modificar el menú de opciones

```
<li class="nav-item px-lg-4 {% if request.path == '/contact/' %} active {% endif %}">
  <a class="nav-link text-uppercase text-expanded"
     href="{% url 'contact:contact' %}">Contacto</a>
</li>
```

12. Verificar su funcionamiento.
13. Cambiar el código duro del url en views.py

```python
from django.urls import reverse_lazy

def contact(request):
    if request.method == 'POST':
        form = ContactForm(request.POST)
        if form.is_valid():
            print(form.cleaned_data.get('name'))
            print(form.cleaned_data['email'])
            # return HttpResponseRedirect('/contact/thanks/')
            return HttpResponseRedirect(reverse_lazy('contact:thanks'))
    else:
        form = ContactForm()
```

14. Validar que el correo sea de Gmail necesariamente.

```python
        if form.is_valid():
            name = form.cleaned_data.get('name')
            email = form.cleaned_data['email']
            content = form.cleaned_data['content']

            pos_arroba = email.find('@')
            domino = email[pos_arroba+1:]

            if domino != "gmail.com":
                form.add_error('email', 'dominio inválido')
                return render(request, 'contact/contact.html', {'form':form})
            else:
                return HttpResponseRedirect(reverse_lazy('contact:thanks'))
    else:
        form = ContactForm()
```

15. Realice el siguiente cambio en el archivo html

```
<!-- Formulario de contacto -->
<form action='' method='post'>
  {% csrf_token %}
  <div class="form-group">
    <label>Nombre *</label>
    <div class="input-group">
      {{form.name}}
    </div>
    {{form.name.errors}}
  </div>
  <div class="form-group">
    <label>Email *</label>
    <div class="input-group">
      {{form.email}}
    </div>
    {{form.email.errors}}
  </div>
  <div class="form-group">
    <label>Mensaje *</label>
    <div class="input-group">
      {{form.content}}
    </div>
    {{form.content.errors}}
  </div>
  <div class="text-center">
    <input type="submit" class="btn btn-primary btn-block py-2" value="Enviar">
  </div>
</form>
<!-- Fin formulario de contacto -->
```

Validación de formularios

Crear el formulario en el template

```
<!-- Formulario de contacto -->
<form action='' method='post'>
    {% csrf_token %}
    <div class="form-group">
        <label>Nombre *</label>
        <div class="input-group">
            <input name='name' type="text" class="form-control">
        </div>
    </div>
    <div class="form-group">
        <label>Email *</label>
        <div class="input-group">
            <input name='email' type="text" class="form-control">
        </div>
        <ul class="errorlist">
        </ul>
    </div>
    <div class="form-group">
        <label>Mensaje *</label>
        <div class="input-group">
            <textarea name='content' class="form-control"></textarea>
        </div>
    </div>
    <div class="text-center">
        <input type="submit" class="btn btn-primary btn-block py-2" value="Enviar">
    </div>
</form>
```

Crear un formulario básico en forms.py

```
from django import forms

You, 10 seconds ago | 1 author (You)
class ContactForm(forms.Form): ...


You, 10 seconds ago | 1 author (You)
class ContactForm2(forms.Form):
    name    = forms.CharField ()
    email   = forms.EmailField()
    content = forms.CharField ()
```

Hacer uso del nuevo formulario en la vista.

Ahora que no existen validaciones del lado del navegador, Django será quien RESPONDA que el formulario no es válido, por lo que debemos de manejar dichos errores.

```python
def contact(request):
    if request.method == 'POST':
        print('post')
        form = ContactForm2(request.POST)
        if form.is_valid():
            name        = form.cleaned_data.get('name')
            email       = form.cleaned_data['email']
            content     = form.cleaned_data['content']
            print(name, email, content)
            dominio     = email[email.find('@')+1:]
            if dominio  != "gmail.com":
                form.add_error('email', 'dominio inválido')
                return render(request, 'contact/contact.html', {'form':form})
            return HttpResponseRedirect (reverse_lazy('contact:thanks'))
        else:
            errors      = form.errors
            return render(request, 'contact/contact.html', {'form':form, 'errors':errors})
    else:
        form = ContactForm2()
    return render(request, 'contact/contact.html', {'form':form})
```

Observa que cuando el formulario es inválido, se regresa la vista del formulario nuevamente, pero con los errores que está “cachando” Django, por lo que es necesario incluirlos en el template.

```
  </form>
  {% if form.errors %}
    <div class="errores">
      <p>Hubo errores en el formulario</p>
      {{form.errors}}
    </div>
  {% endif %}

</div>
```

Verifica alguna validación externa, por ejemplo, si coloca alguna restricción en el formulario de mínimo de caracteres, ésta se validará.

```python
class ContactForm2(forms.Form):
    name    = forms.CharField (min_length=5)
    email   = forms.EmailField()
```

Validación Personalizada

```python
class ContactForm2(forms.Form):
    name = forms.CharField(min_length=5)
    email = forms.EmailField()
    content = forms.CharField()

    # validación exclusiva para el campo email
    def clean_email(self):
        email = self.cleaned_data.get('email')
        dominio = 'google.com'
        if dominio not in email:
            raise forms.ValidationError(f'El correo debe ser {dominio}')
        return email
```

Ya que la validación se está colocando en el formulario, cuando no se cumpla la validación, se regresará un formulario inválido, dado esto, se debe eliminar la validación personalizada en la vista.

Por otro lado, también puede realizar validaciones para incluir a mas de un campo, por ejemplo:

```python
class ContactForm2(forms.Form):
    name = forms.CharField(min_length=5)
    email = forms.EmailField()
    content = forms.CharField()

    # validación exclusiva para el campo email
    def clean_email(self):...

    # validacion general
    def clean(self):
        cleaned_data = super().clean()
        name = cleaned_data.get('name')
        email = cleaned_data.get('email')
        if name and email:
            if email in email:
                raise forms.ValidationError(
                    "El nombre de usuario no puede estar en el correo electrónico.")

        return cleaned_data
```