

Hito 3 - Añadiendo interacción a nuestro sitio web (modelos y formularios)

- Para realizar este desafío debes haber estudiado previamente todo el material disponible en el LMS correspondiente a la unidad.
- Una vez terminado el presente Hito, comprime la carpeta que contiene el desarrollo de los requerimientos solicitados en un archivo .zip y súbelo al sistema LMS.
- Desarrollo desafío: **En parejas.**
Se recomienda utilizar el navegador Google Chrome al acceder a la documentación en inglés, de manera de que el navegador permita traducir automáticamente su contenido.
- Para la realización del desafío necesitarás apoyarte de la documentación de bootstrap e inspirarte en su sitio de ejemplos:
 - <https://getbootstrap.com/docs/5.0/getting-started/introduction/>
 - <https://getbootstrap.com/docs/5.0/examples/>

Además de lo anterior, puedes revisar un “[apunte](#)” con todos los elementos de bootstrap en el siguiente enlace:

- <https://getbootstrap.com/docs/5.0/examples/cheatsheet/>
- Para la realización de este hito necesitarás apoyarte en la documentación de Django y tutoriales complementarios
 - <https://docs.djangoproject.com/es/3.2/intro/tutorial04/>
 - <https://docs.djangoproject.com/es/3.2/topics/forms/#forms-in-django>
 - <https://developer.mozilla.org/es/docs/Learn/Server-side/Django/Forms>
 - https://developer.mozilla.org/es/docs/Learn/Server-side/Django/Admin_site#registrando_los_modelos
 - <https://docs.djangoproject.com/es/3.2/ref/forms/api/#django.forms.Form>
 - <https://docs.djangoproject.com/en/3.2/topics/forms/#reusable-form-templates>
 - <https://docs.djangoproject.com/en/3.2/topics/forms/modelforms/>
 - <https://docs.djangoproject.com/en/3.2/topics/forms/modelforms/#validation-on-a-modelform>
 - <https://docs.djangoproject.com/es/3.2/topics/forms/#building-a-form-in-django>

Habilidades a evaluar

- **Forms en Django:**
 - Utiliza las clases provistas por el framework Django para la integración de un formulario básico.
 - Procesa un formulario Django utilizando plantillas (templates) para dar solución a un requerimiento.
 - Implementa plantillas de formulario reutilizables para dar solución a un requerimiento.
 - Maneja mensajes de errores de formularios en plantillas (templates) para dar solución a un requerimiento.

Descripción

Ya tenemos la estructura básica de nuestro sitio web, es hora de darle contenido y agregar cierta interacción con el usuario. Actualmente tenemos solo header, navbar y footer ¿pero donde quedaron los flanes? Esto será lo primero que resolveremos, para esto, crearemos el modelo **Flan** y lo agregaremos al panel de administración de Django, luego agregaremos algunos flanes a nuestro sitio web para darle más contenido y finalmente crearemos un formulario de contacto que permitirá a los potenciales clientes comunicarse con los administradores de la PYME.

Tal como en el Hito II, agregaremos más elementos de bootstrap a nuestro sitio web, que permitirá darle más personalidad e interacción a los usuarios. Deberás agregar una lista de Flan que se mostrarán en la página principal del sitio web, así como algunos Flan especiales que serán mostrados en la web de bienvenida, esto te permitirá familiarizarte con el panel de administración de Django y sus formularios integrados.

Además de lo anterior, agregaremos un formulario de contacto personalizado, que permitirá crear un registro de los usuarios que deseen contactarse a través de nuestro sitio web, permitiéndonos ver los datos ingresados en éste en el panel de administración de Django.

Requerimientos

1. Crear modelo **Flan**, generar y aplicar sus migraciones y agregar el modelo al panel de administración de Django.

Primero deberemos crear un modelo llamado **Flan** que contenga los siguientes atributos:

- **flan_uuid** del tipo `UUIDField`
- **name** del tipo `CharField` (largo máximo 64 caracteres)
- **description** del tipo `TextField`
- **image_url** del tipo `URLField`
- **slug** del tipo `SlugField`
- **is_private** del tipo `BooleanField`

Esto permitirá crear distintos **Flan** que serán presentados en la web, junto con una imagen (**image_url**), nombre (**name**), descripción (**description**), un identificador único (**flan_uuid**), un campo que permite especificar un nombre corto de url (**slug**) y un campo que permite definir si el flan es privado o no (**is_private**).

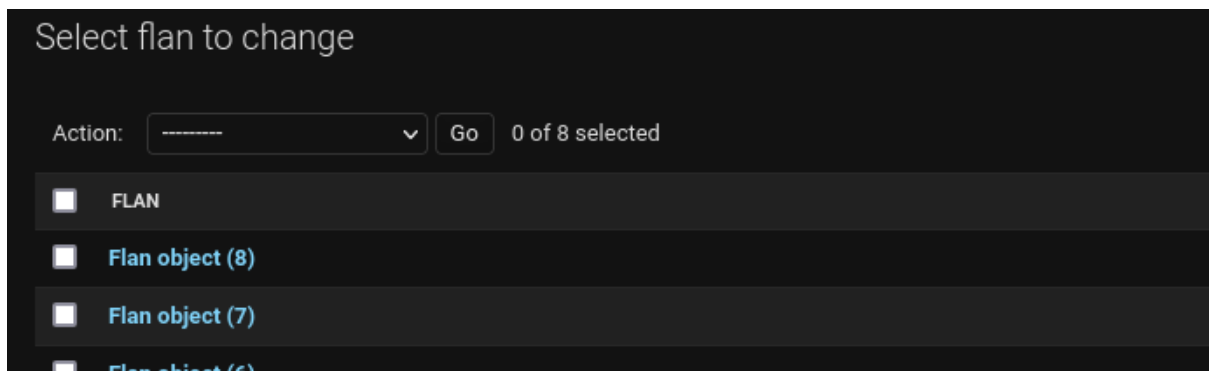
Un campo UUID o [Identificador Único Universal](#) es un estándar de generación de identificadores únicos, los cuales pueden ser asociados como atributos de un modelo en Django a través del tipo **UUIDField** permitiendo el registro de un identificador distinto para cada registro de un modelo creado, en este caso, un identificador distinto para cada **Flan**.

Una vez definido el modelo **Flan** en el archivo **models.py** del app **web**, genera las migraciones con el comando **makemigrations** y aplícalas con el comando **migrate**.

Si todo va bien y las migraciones se aplicaron correctamente, [registra el modelo Flan](#) en el panel de administración de Django agregándolo al archivo **admin.py** del app **web**, luego crea las credenciales de super usuario con el comando **python manage.py createsuperuser** y posteriormente ejecuta el servidor y visita <http://127.0.0.1:8000/admin/> ingresando con las credenciales recientemente creadas.

Finalmente, busca el modelo **Flan** dentro del panel de administración de Django, Agrega al menos 8 nuevos elementos **Flan** con su descripción, imagen, etc. Puedes generar un uuid para cada flan con [esta herramienta](#). Las imágenes que se pueden agregar son cualquier imagen pública, podemos seleccionar con “click derecho” cualquier imagen de flan que nos interese y presionar “Copiar enlace a imagen” o algún texto similar que variará según el navegador que ocupemos. Se recomienda definir tanto ítems que tengan el valor de **is_private** como **True** (casilla marcada en el panel de administración) como **False** (casilla desmarcada en el panel de administración).

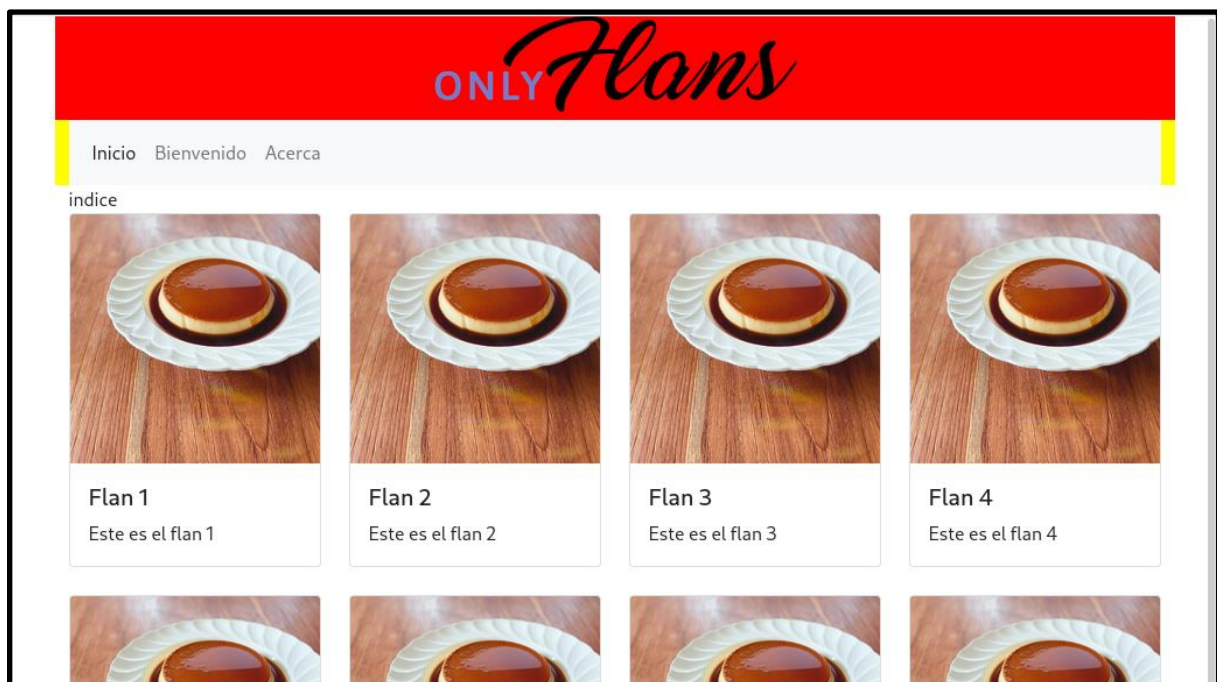
Luego de agregar los 8 flanes se debería tener algo como esto:



Al finalizar, genera un “pantallazo” de la lista de estos 8 elementos dentro del panel de administración Django y guárdalo bajo el formato jpg o png.

(2 Puntos)

2. Es hora de mostrar tus Flanes recién creados al público. Agrégale el resultado de todos los Flanes existentes en tu sitio web como contexto a tu vista de la ruta / (Index o Índice) e “imprime” los resultados a través de la plantilla **index.html** a través del [componente card de bootstrap](#), utilizando [el ciclo for de las plantillas de django](#) para mostrar cada uno de los **Flan** anteriormente creados, hasta lograr algo como esto:



Muestra de productos en página principal

La lista de flanes puede ser obtenida de la siguiente manera:

- `flanes = Flan.objects.all()`
- `flanes_privados = Flan.objects.filter(is_private=True)`

- `flanes_publicos = Flan.objects.filter(is_private=False)`

Una vez hayamos logrado mostrar nuestros Flan recién creados en la página principal, deberemos agregar lo mismo en la página de bienvenida, esta vez diferenciando los contenidos de la siguiente manera:

- En la página principal(Index o Índice) se mostrarán solo los **Flan** cuyo atributo **is_private** es igual a **False**
- En la página de bienvenida (Welcome o Bienvenida), se mostrarán solo los **Flan** cuyo atributo **is_private** es igual a **True**

Finalizado lo anterior, visita las rutas / y **/bienvenido** de tu sitio web, realiza un “pantallazo” de cada una de las 2 rutas y guárdalo en formato jpg o png.

(2 Puntos)

3. Crear modelo **ContactForm**, generar y aplicar sus migraciones y agregar el modelo al panel de administración de Django.

Primero deberemos crear un modelo llamado **ContactForm** que contenga los siguientes atributos:

- **contact_form_uuid** del tipo **UUIDField**, con valor por defecto **uuid.uuid4**, no editable
- **customer_email** del tipo **EmailField**
- **customer_name** del tipo **CharField** (largo máximo 64 caracteres)
- **message** del tipo **TextField**

Esto permitirá crear distintos **ContactForm** que podrán ser creados en la web y presentados a través del panel de administración de Django, contendrán un identificador único (**contact_form_uuid**) que será agregado automáticamente (a través de la invocación a **uuid.uuid4**) cada vez que un formulario de contacto se cree, el email del cliente (**customer_email**), su nombre (**customer_name**) y el texto del mensaje que nos dejará en el formulario (**message**). Una vez creado el modelo, debemos generar y aplicar las migraciones.

Como comentamos anteriormente, **UUID** es un estándar de generación de identificadores únicos, en este caso lo que haremos será definir que su valor por defecto será **uuid.uuid4**, lo que permitirá definir automáticamente un **UUID** en su versión 4(al azar) cada vez que se genere un nuevo registro del modelo **ContactForm**.

```
1 import uuid
2 from django.db import models
3
4 class Flan(models.Model):
5     flan_uuid = models.UUIDField()
6     name = models.CharField(max_length=64)
7     description = models.TextField()
8     image_url = models.ImageField()
9     slug = models.SlugField()
10    is_private = models.BooleanField()
11
12 class ContactForm(models.Model):
13     contact_form_uuid = models.UUIDField(default=uuid.uuid4, editable=False)
14     customer_email = models.EmailField()
15     customer_name = models.CharField(max_length=64)
16     message = models.TextField()
17
```

Tu modelo ContactForm debería quedar de esta manera

TIP: Utilizamos **uuid.uuid4** en vez de **uuid.uuid4()** debido a que de esta manera se llamará a la función **uuid4** cada vez que se genere un nuevo registro. Si definiríamos **uuid4()** como valor por defecto lo que haríamos sería generar 1 solo uuid para todos nuestros registros, lo que nos generaría errores ya sea al crear nuevos registros, generar migraciones, etc.

```
^C(onlyflans) [mcueto@manjaro onlyflans]$ python manage.py makemigrations
Migrations for 'web':
  web/migrations/0002_contactform.py
  - Create model ContactForm
(onlyflans) [mcueto@manjaro onlyflans]$ python manage.py migrate
Operations to perform:
  Apply all migrations: admin, auth, contenttypes, sessions, web
Running migrations:
  Applying web.0002_contactform... OK
(onlyflans) [mcueto@manjaro onlyflans]$
```

Las migraciones fueron creadas y aplicadas con éxito

Una vez aplicadas las migraciones, debemos [registrar el modelo **ContactForm** en el archivo **admin.py**](#) y revisar que aparezca en nuestro [panel de administración](#) de Django. Probablemente notes algunas diferencias en nuestro formulario si intentas agregar un elemento nuevo, por ejemplo la ausencia del campo **contact_form_uuid**, esto es debido a que le definimos un valor por defecto y además especificamos que

no es editable; es por esto que no aparecerá en los formularios del panel de administración de Django.

Crea una vista y una nueva url que permita mostrar el mensaje “Contacto” en la ruta **/contacto**, esta ruta debe tener su propia plantilla html que extienda de la plantilla base. Agrega el enlace hacia esa ruta en el navbar de manera de permitir su fácil acceso.



<http://127.0.0.1:8000/contacto>

Para [construir un formulario](#), debes crear un archivo **forms.py** en el app **web**, importar **django.forms** y crear un formulario llamado **ContactFormForm** que contenga los atributos necesarios para poder recibir los datos necesarios en el modelo **ContactForm**, agregamos el sufijo **Form** al nombre de nuestro formulario para poder diferenciar **ContactForm** (modelo) de **ContactFormForm** (formulario del modelo **ContactForm**), esto puede llegar a ser confuso, por lo que comúnmente se recomienda utilizar un nombre con sufijo distinto de **Form** en el modelo, como por ejemplo **ContactData**, para evitar la repetición de la palabra Form en el formulario del modelo (En ese caso se llamaría **ContactDataForm**), para fines del hito, se mantiene el nombre del modelo como **ContactForm**.

Un formulario contiene atributos, similarmente a como se define un modelo en Django. Podemos definir las etiquetas con las que se mostrarán sus campos, límites, valores iniciales, etc.

```
1 from django import forms
2
3 class ContactFormForm(forms.Form):
4     # contact_form_uuid No necesita ser declarado en nuestro form
5     customer_email = forms.EmailField(label='Correo')
6     customer_name = forms.CharField(max_length=64, label='Nombre')
7     message = forms.CharField(label='Mensaje')
8
```

Archivo web/forms.py

En la vista de contacto, debemos realizar las validaciones necesarias para validar el método de la solicitud(GET o POST), obtener su data, pasarle esa data a nuestro formulario, validarlo y luego redirigir al usuario a otra ruta.

```
31 def contacto(request):
32     if request.method == 'POST':
33         # create a form instance and populate it with data from the request:
34         form = ContactFormForm(request.POST)
35         # check whether it's valid:
36         if form.is_valid():
37             # process the data in form.cleaned_data as required
38             # ...
39             # redirect to a new URL:
40             return HttpResponseRedirect('/')
41
42     # if a GET (or any other method) we'll create a blank form
43     else:
44         form = ContactFormForm()
45
46     return render(request, 'contactus.html', {'form': form})
47
```

Archivo web/views.py

La plantilla usada para el contacto debe contener el `{%csrf_token%}` que sirve como “llave” para identificar que las llamadas desde el formulario provienen de la misma web. Es una característica de seguridad que previene los [ataques CSRF](#).


```
1 {% extends 'base.html' %}
2
3 {% block 'content' %}
4 <div class="container">
5     contacto
6
7     <form action="/contacto" method="post">
8         {% csrf_token %}
9         {{ form }}
10        <input type="submit" value="Submit">
11    </form>
12
13 </div>
14 {% endblock %}
15
```

Archivo web/templates/contactus.html

<http://127.0.0.1:8000/contacto>

Como podemos observar, al utilizar `{{form}}` dentro de una plantilla que recibe un formulario llamado **form** a través de su contexto, Django realiza el trabajo de separar cada atributo definido en nuestro **ContactFormForm**, luego nuestro formulario es validado en nuestra vista **contacto** y si el formulario enviado cumple con los requerimientos el usuario es redirigido a través de **HttpResponseRedirect** a la ruta / (La función **HttpResponseRedirect** debe ser importada en el archivo **webs/views.py**).

Sin agregar procesamiento del formulario todavía, debes crear una vista y una plantilla de éxito, registrando su url pero sin añadirla a tu navbar, de manera de mostrar un mensaje de éxito una vez de que un usuario envíe un formulario de contacto.

Luego en tu formulario **ContactFormForm** debes redirigir al usuario a la nueva url de éxito cuando el formulario sea válido, de manera de al llenar un formulario de contacto exitosamente, el usuario sea redirigido a la nueva vista de éxito y se le muestre un mensaje del estilo “Gracias por contactarte con OnlyFlans, te responderemos en breve”

```
31 def contacto(request):
32     if request.method == 'POST':
33         # create a form instance and populate it with data from the request:
34         form = ContactFormForm(request.POST)
35         # check whether it's valid:
36         if form.is_valid():
37             # process the data in form.cleaned_data as required
38             # ...
39             # redirect to a new URL:
40             return HttpResponseRedirect('/exito')
41
42     # if a GET (or any other method) we'll create a blank form
43     else:
44         form = ContactFormForm()
45
46     return render(request, 'contactus.html', {'form': form})
47
```

Una vez configurado correctamente, deberíamos ser capaces de mostrar algo como lo siguiente tras rellenar y enviar nuestro formulario de contacto:



<http://127.0.0.1:8000/exito>

A este punto ya tenemos un formulario funcional y una url de éxito para cuando los usuarios lo completen correctamente, ahora toca registrar los datos del formulario en nuestro sistema según lo ingresado por los usuarios de nuestro sitio web.

Para esto debemos importar el modelo **ContactForm** en nuestro archivo **web/views.py** (línea 3) y dentro de la validación de nuestro formulario, debemos crear un nuevo registro usando los datos del formulario a través de **form.cleaned_data** (línea 39)

```
1 from django.shortcuts import render
2 from django.http import HttpResponseRedirect, HttpResponseRedirect
3 from .models import Flan, ContactForm
4 from .forms import ContactFormForm
5
```

Debemos importar el modelo ContactForm, archivo web/views.py

```
31 def contacto(request):
32     if request.method == 'POST':
33         # create a form instance and populate it with data from the request:
34         form = ContactFormForm(request.POST)
35         # check whether it's valid:
36         if form.is_valid():
37             # process the data in form.cleaned_data as required
38             # ...
39             contact_form = ContactForm.objects.create(**form.cleaned_data)
40
41             # redirect to a new URL:
42             return HttpResponseRedirect('/exit')
43
```

Debemos crear un nuevo "objeto" de ContactForm, basado en los datos del formulario, archivo web/views.py

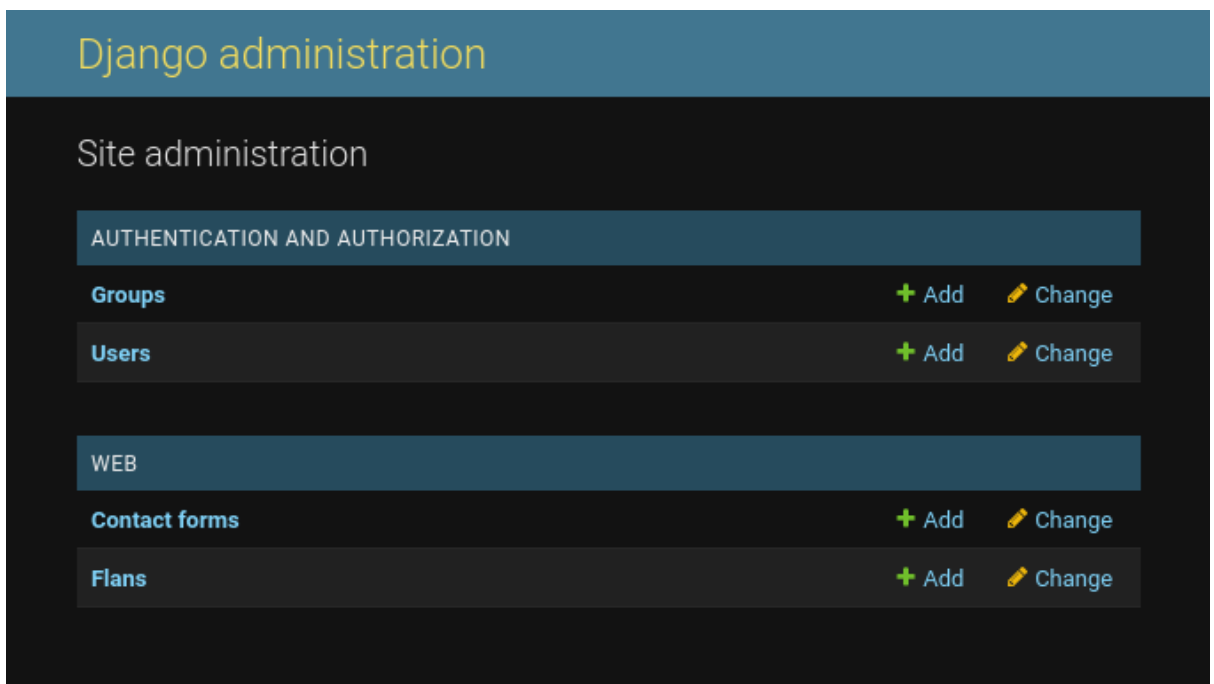
Una vez funcionando lo anterior, procederemos a ejecutar el servidor de django y a navegar hasta el formulario de contacto, llenándolo con datos correctos que permitan la creación exitosa, la cual corroboramos tanto por el mensaje de éxito que se le mostrará al usuario como por los datos registrados que podremos acceder a través del panel de administración de Django.



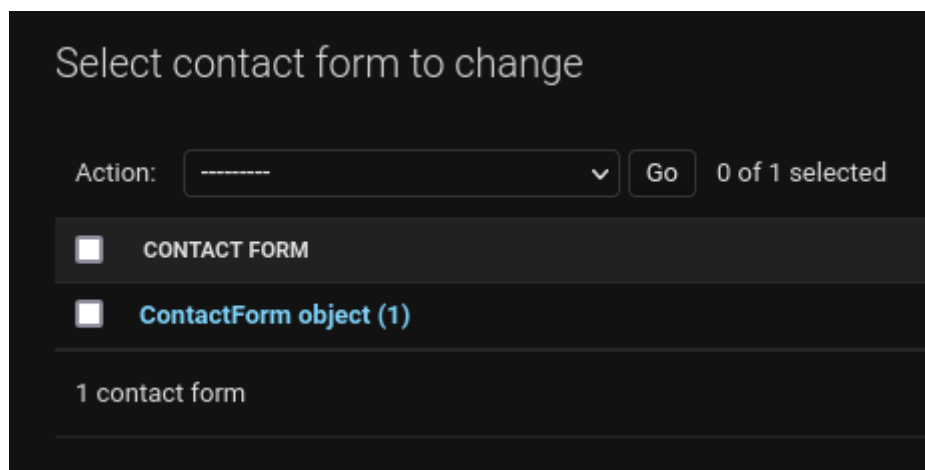
Formulario con datos correctos, <http://127.0.0.1:8000/contacto>



Mensaje de éxito luego de haber llenado correctamente el formulario,
<http://127.0.0.1:8000/exito>



Panel de administración con nuestro modelo ContactForm registrado,
<http://localhost:8000/admin>



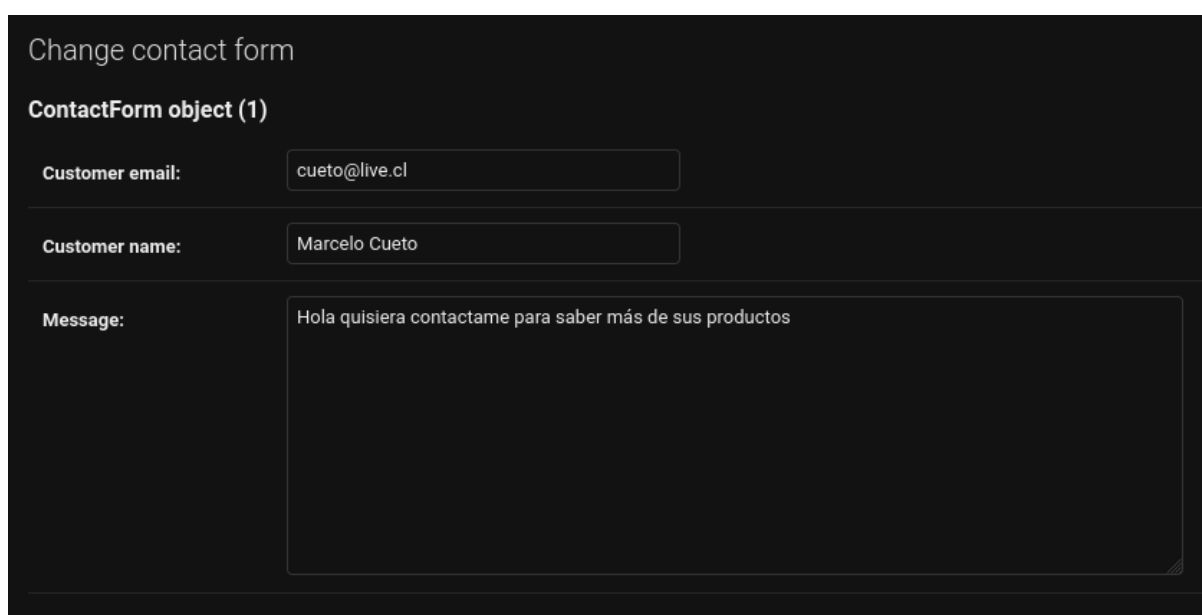
Select contact form to change

Action: 0 of 1 selected

<input type="checkbox"/>	CONTACT FORM
<input type="checkbox"/>	ContactForm object (1)

1 contact form

Lista de ContactForm registrados en el sistema,
<http://127.0.0.1:8000/admin/web/contactform/>



Change contact form

ContactForm object (1)

Customer email:

Customer name:

Message:

Detalle de ContactForm creado exitosamente, con los datos antes ingresados.

A este punto, ya tenemos un formulario 100% funcional, que muestra un mensaje a los usuarios que lo llenan y registra sus datos en nuestra base de datos, los cuales pueden ser accedidos desde nuestro panel de administración de Django. Ahora debes modificar la plantilla de contacto de manera de [mostrar los campos del formulario de forma individual](#), con el fin de lograr una página más estilizada. También es la hora de quitar los colores de fondo de nuestros header, contenido y footer si es que aún los tenemos habilitados.

En vez de ocupar `{{form}}` debemos acceder directamente a los valores de cada campo del formulario, por ejemplo `{{form.customer_name.errors}}` nos mostrará los errores que se hayan presentado en el campo **customer_name** formulario si es que éste se envió y no fué validado correctamente, `{{form.customer_name.label_tag}}` nos mostrará la etiqueta(label) del campo en particular, entre otros.

```
3  {% block 'content' %}
4  <div class="container">
5
6      <div class="col-12 col-md-6 offset-0 offset-md-3 mt-4 mb-4">
7          <h3 class="text-center">
8              Contáctanos
9          </h3>
10
11      <div>
12
13          <form action="/contacto" method="post">
14              {% csrf_token %}
15              <!-- {{ form }} -->
16
17              {{ form.non_field_errors }}
18
19              <div class="mb-3 text-center">
20                  <label for="{{ form.customer_name.id_for_label }}" class="form-label">
21                      {{ form.customer_name.label_tag }}
22                  </label>
23                  <div>
24                      {{ form.customer_name }}
25                  </div>
26                  <div class="text-danger">
27                      {{ form.customer_name.errors }}
28                  </div>
29              </div>
30
```

Ejemplo de valores accedidos directamente en plantilla, para mostrar un formulario,
archivo web/templates/contactus.html

Una vez terminado el punto anterior, al ingresar datos incorrectos, ejemplo el correo sin la extensión del dominio(ejemplo: .cl o .com) nuestro sitio debería verse similar a esto.

ONLY Hans

Inicio Bienvenido Acerca Contacto

Contáctanos

Nombre:

Juan Perez

Correo:

correo@correo

• Enter a valid email address.

Mensaje:

Hola quisiera contactarme

Enviar

Creado por Juan Perez y Juana Pereira para Desafio Latam - 2021

Formulario con errores al ser enviado, se muestran en la misma página,
<http://127.0.0.1:8000/contacto>

Una vez terminado todo lo anterior, modifica el contenido de la plantilla mostrada en la ruta **/acerca** para que contenga un texto sobre qué se trata la web, elimina aquellos elementos sobrantes como la palabra "índice", "bienvenido cliente", "acerca" y "contacto" que aparecen bajo el navbar pero sobre el contenido, y reemplázalo en los casos que sea necesario con un texto u otros componentes de bootstrap, luego genera los siguientes pantallazos y guárdalos en formato jpg o png:

- Página Inicio
- Página Bienvenido
- Página Acerca
- Página Contacto(sin datos)
- Página Contacto(con dato de correo erroneo)
- Página Contacto(con dato de correo no erroneo)
- Página de éxito luego de enviar un contacto
- Detalle del contacto creado en el panel de administración de Django

(4 Puntos)

4. Ya tenemos un sitio web que permite a sus usuarios enviar un formulario de contacto que queda registrado en nuestro panel de administración de Django. Lo que debemos hacer ahora es crear un modelo [basado en ModelForm](#) que permita reemplazar nuestro **ContactFormForm**, lo podemos llamar **ContactFormModelForm**.

Luego reemplazamos los llamados de **ContactFormForm** por **ContactFormModelForm** y probaremos el formulario de contacto para realizar los siguientes “pantallazos” que deberán ser guardados en formato jpg o png:

- Página Contacto(sin datos)
- Página Contacto(con dato de correo erroneo)
- Página Contacto(con dato de correo no erróneo)
- Página de éxito luego de enviar un contacto
- Detalle del contacto creado en el panel de administración de Django

(el formulario debería lucir un poco diferente a como lo teníamos configurado en el punto anterior, no te preocupes por su estética, solo verifica su funcionamiento y genera los “pantallazos”, luego puedes volver a como tenias el formulario de contacto en el punto anterior o resolver los detalles estéticos de la forma que desees)

(2 Puntos)



¡Mucho éxito!