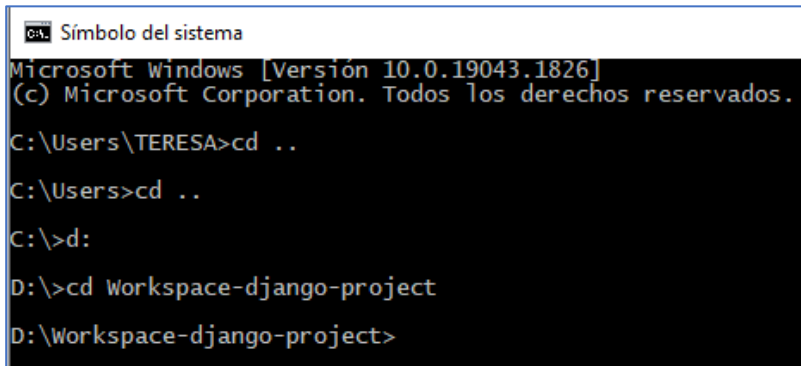


Laboratorio N°12: CREAR PROYECTO CON FORMULARIO Y BD

Crear otro Proyecto **AdmEmpleadosForm** y una aplicación **moduloEmpleadoForm**

Pasos:

1. Abrir CMD
2. Ubicarse dentro del espacio de trabajo **workspace-django-project**



```
Simbolo del sistema
Microsoft Windows [Versión 10.0.19043.1826]
(c) Microsoft Corporation. Todos los derechos reservados.

C:\Users\TERESA>cd ..

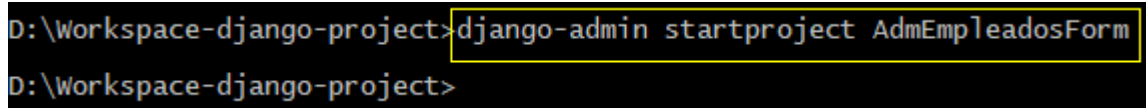
C:\Users>cd ..

C:\>d:

D:\>cd workspace-django-project

D:\workspace-django-project>
```

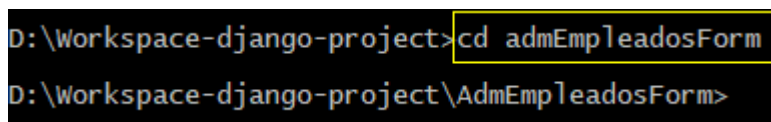
3. Crear el proyecto **AdmEmpleadosForm** ejecutando la siguiente instrucción:
django-admin startproject AdmEmpleadosForm



```
D:\workspace-django-project>django-admin startproject AdmEmpleadosForm

D:\workspace-django-project>
```

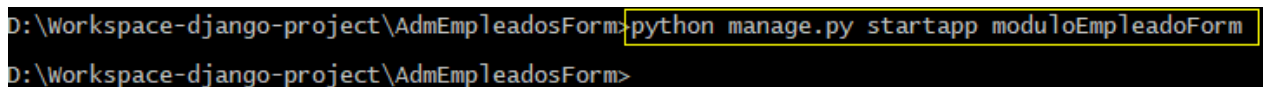
4. Entrar al proyecto **AdmEmpleadosForm**



```
D:\workspace-django-project>cd admEmpleadosForm

D:\workspace-django-project\AdmEmpleadosForm>
```

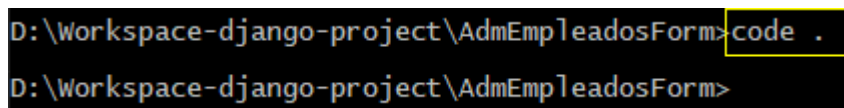
5. Crear la aplicación **moduloEmpleadoForm** ejecutando la siguiente instrucción:
python manage.py startapp moduloEmpleadoForm



```
D:\workspace-django-project\AdmEmpleadosForm>python manage.py startapp moduloEmpleadoForm

D:\workspace-django-project\AdmEmpleadosForm>
```

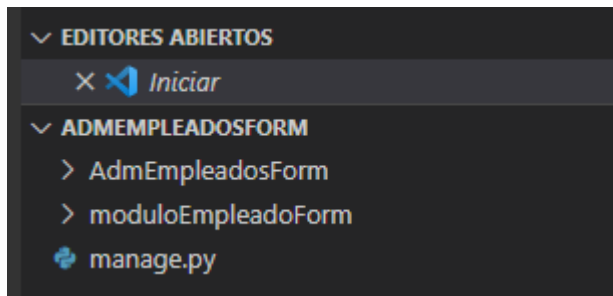
6. Abrir visual code



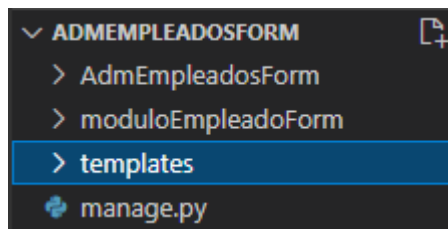
```
D:\workspace-django-project\AdmEmpleadosForm>code .

D:\workspace-django-project\AdmEmpleadosForm>
```

En la siguiente imagen, se puede visualizar el proyecto **AdmEmpleadosForm** y la aplicación **moduloEmpleadoForm**

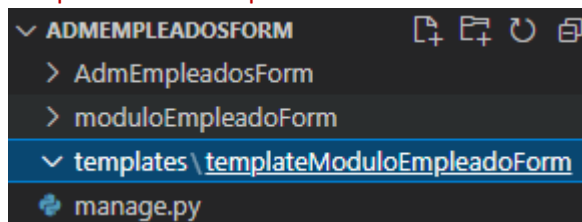


7. En el proyecto `AdmEmpleadosForm`, crear la carpeta `templates` que almacenará los templates de las aplicaciones que posee el proyecto. Revisar la creación en la imagen siguiente.

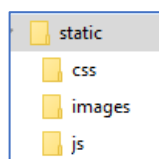


Nota: también se puede crear desde el CMD con la instrucción `mkdir`

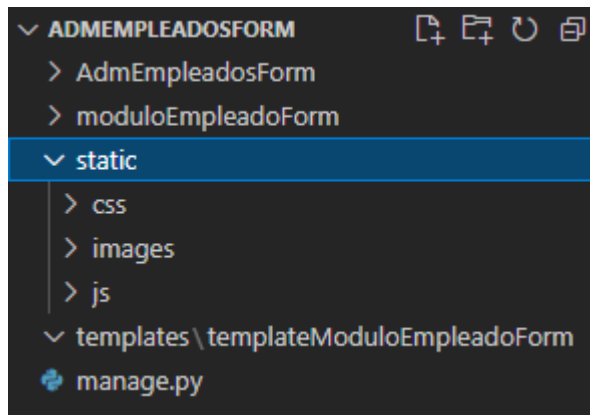
8. Crear la carpeta `templateModuloEmpleadoForm` dentro de la carpeta `templates`. La carpeta `templateModuloEmpleadoForm` almacenará los templates de la aplicación `moduloEmpleadoForm`



9. Para almacenar contenido estático crear la siguiente estructura de carpetas dentro del proyecto `AdmEmpleadosForm`



Nota: la carpeta `static` está a la misma altura de la carpeta `templates`. Se usan los mismos pasos de creación de carpetas del punto anterior.



Nota: para crear las carpetas css, js e images hacer click derecho del mouse sobre la palabra static.

10. Modificar el archivo de configuración `settings.py` del proyecto `AdmEmpleadosForm`. Las modificaciones son:

- a) Incluir la aplicación `moduloEmpleadoForm` al Proyecto `AdmEmpleadosForm`

```
INSTALLED_APPS = [  
    'django.contrib.admin',  
    'django.contrib.auth',  
    'django.contrib.contenttypes',  
    'django.contrib.sessions',  
    'django.contrib.messages',  
    'django.contrib.staticfiles',  
    'moduloEmpleadoForm'  
]
```

- b) Indicar la ruta de los templates del Proyecto `AdmEmpleadosForm`

Las instrucciones son:

- i. Incluir import os

```
12 import os  
13 from pathlib import Path  
14
```

- ii. Crear la variable `TEMPLATE_DIR` con la ruta de la carpeta templates del Proyecto y agregarla

```
15 # Build paths inside the project like this: BASE_DIR / 'subdir'.  
16 BASE_DIR = Path(__file__).resolve().parent.parent  
17 TEMPLATE_DIR=os.path.join(BASE_DIR, 'templates')
```

- iii. Especificar la variable `TEMPLATE_DIR` en el código que se indica en el recuadro amarillo

```

55  TEMPLATES = [
56      {
57          'BACKEND': 'django.template.backends.django.DjangoTemplates',
58          'DIRS': [],
59          'APP_DIRS': True,
60          'OPTIONS': {
61              'context_processors': [
62                  'django.template.context_processors.debug',
63                  'django.template.context_processors.request',
64                  'django.contrib.auth.context_processors.auth',
65                  'django.contrib.messages.context_processors.messages',
66              ],
67          },
68      ],
69  ]

```

El resultado es:

```

55  TEMPLATES = [
56      {
57          'BACKEND': 'django.template.backends.django.DjangoTemplates',
58          'DIRS': [TEMPLATE_DIR],
59          'APP_DIRS': True,
60          'OPTIONS': {
61              'context_processors': [
62                  'django.template.context_processors.debug',
63                  'django.template.context_processors.request',
64                  'django.contrib.auth.context_processors.auth',
65                  'django.contrib.messages.context_processors.messages',
66              ],
67          },
68      ],
69  ]

```

- c) Revisar y configurar contenido estático
- i. Revisar la existencia de la variable **STATIC_URL** con la asignación de la carpeta **static** del Proyecto, sino agregarla

```

116 # Static files (CSS, JavaScript, Images)
117 # https://docs.djangoproject.com/en/4.1/howto/static-files/
118
119 STATIC_URL = 'static/'

```

- ii. Especificar la variable **STATICFILES_DIRS** en el código como se indica en el recuadro amarillo. En otras palabras, agregar la variable **STATICFILES_DIRS**

```

116 # Static files (CSS, JavaScript, Images)
117 # https://docs.djangoproject.com/en/4.1/howto/static-files/
118
119 STATIC_URL = 'static/'
120 STATICFILES_DIRS=[os.path.join(BASE_DIR, 'static')]

```

- d) Configurar que la consola de administración sea en español

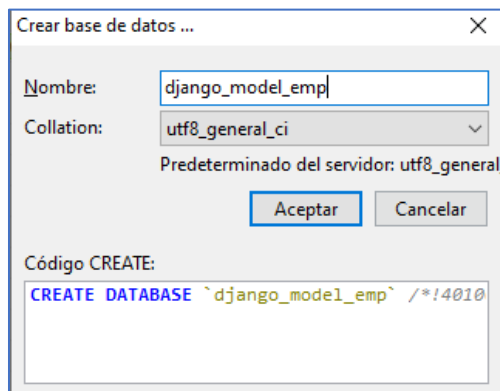
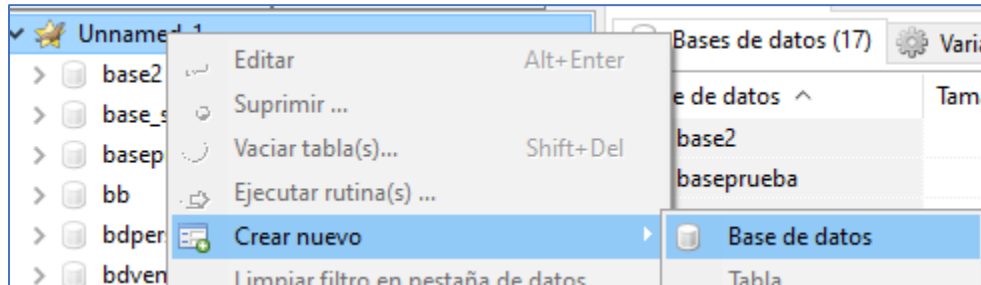
```

LANGUAGE_CODE = 'es-Es' # 'en-us' en ingles

```

e) Crear y configurar la Base de Datos

i. Crear la Base de Datos **django_model_emp** en Maria DB



ii. Instalar PyMySQL si es que no está instalado

pip install PyMySQL

iii. Configurar la base de datos en el archivo **settings.py**. En otras palabras, reemplazar el siguiente código que se ilustra en la siguiente imagen

```
77 DATABASES = {
78     'default': {
79         'ENGINE': 'django.db.backends.sqlite3',
80         'NAME': BASE_DIR / 'db.sqlite3',
81     }
82 }
```

Se reemplaza por:

```
# Database
# https://docs.djangoproject.com/en/4.1/ref/settings/#databases
#pip install PyMySQL
import pymysql
pymysql.install_as_MySQLdb()

DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.mysql',
        'NAME': 'django_model_emp',
        'USER': 'root',
        'PASSWORD': 'root1502'
    }
}
```

iv. Iniciar el servidor

```
PS D:\Workspace-django-project\AdmEmpleadosForm> Python manage.py runserver
Watching for file changes with StatReloader
Performing system checks...

System check identified no issues (0 silenced).
August 15, 2022 - 11:45:42
Django version 4.1, using settings 'AdmEmpleadosForm.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CTRL-BREAK.
```

11. Crear la vista para el template **index.html**. Editar el archivo **views.py** de la aplicación **moduloEmpleadoForm** y agregar la función **index** como se indica en la imagen.

```
moduloEmpleadoForm > views.py > ...
1  from django.shortcuts import render
2
3  # Create your views here.
4  def index(request):
5      return render(request, 'templateModuloEmpleadoForm/index.html')
```

12. Crear el templates del Proyecto, específicamente en la carpeta **templates/templateModuloEmpleadoForm** crear el archivo **index.html**

```
▼ ADMEMPLEADOSFORM
  > AdmEmpleadosForm
  > moduloEmpleadoForm
  > static
  ▼ templates \ templateModuloEmpleadoForm
    <> index.html
    📁 manage.py
```

13. Editar el archivo **index.html** para diseñar la siguiente imagen.

```
templates > templateModuloEmpleadoForm > <> index.html
1  <!DOCTYPE html>
2  {% load static %}
3  <html>
4      <head>
5          <meta charset="utf-8">
6          <meta name="viewport" content="width=device-width, initial-scale=1.0">
7          <meta http-equiv="X-UA-Compatible" content="ie=edge">
8          <meta name="description" content="">
9          <title>DEMO FORMS</title>
10         <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/css/bootstrap.min.css" rel="stylesheet" integrity="sha384-1220737297cd31E645f064636c766666666666666666666666666666666666666" crossorigin="anonymous">
11         <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/js/bootstrap.bundle.min.js" integrity="sha384-1220737297cd31E645f064636c766666666666666666666666666666666666666" crossorigin="anonymous"></script>
12     </head>
13     <body class="container mt-5">
14         <div class="alert alert-info display-1 text-center"> DJANGO FORMS </div>
15     </body>
16 </html>
```

14. Modificar el archivo **urls.py** del proyecto **AdmEmpleadosForm** e incluir el path a la función **index**

```
from django.contrib import admin
from django.urls import path

from moduloEmpleadoForm import views

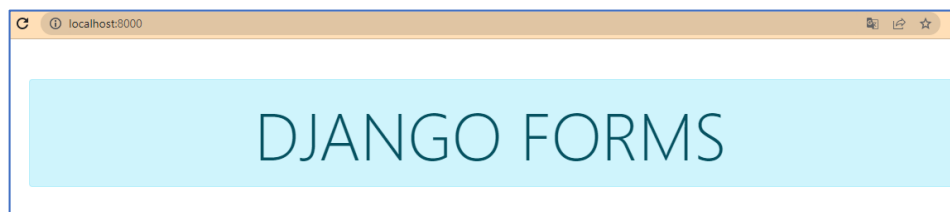
urlpatterns = [
    path('admin/', admin.site.urls),
    path('', views.index),
]
```

15. Iniciar el servidor y revisar. Abrir el terminal del visual code

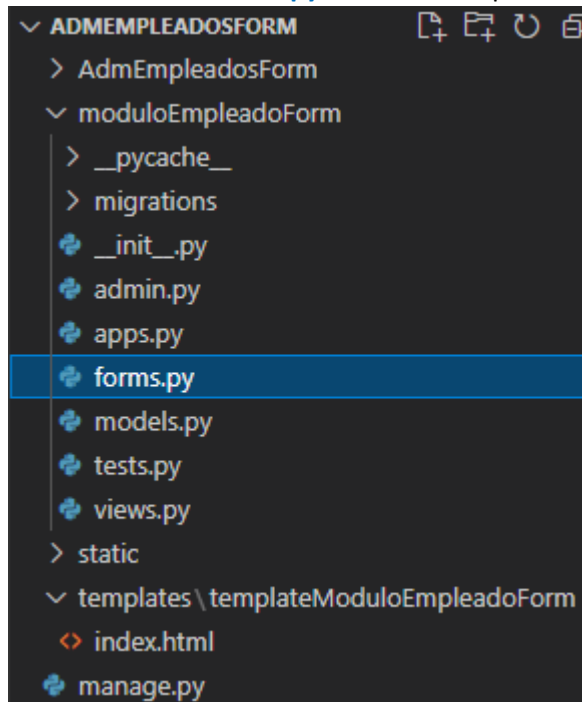
```
PS D:\Workspace-django-project\AdmEmpleadosForm> Python manage.py runserver
Watching for file changes with StatReloader
Performing system checks...

System check identified no issues (0 silenced).
August 15, 2022 - 12:35:24
Django version 4.1, using settings 'AdmEmpleadosForm.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CTRL-BREAK.
```

Localhost:8000



16. Crear un archivo **forms.py** dentro de la aplicación **moduloEmpleadoForm**.



17. Editar el archivo **forms.py** de la aplicación **moduloEmpleadoForm** y crear el formulario. Ingresar el código que se ilustra en la imagen.

```
moduloEmpleadoForm > forms.py > ...
1  from django import forms
2
3  class UserRegistrationForm(forms.Form):
4      nombre =forms.CharField()
5      email =forms.CharField()
6      fono =forms.CharField()
7
```

Nota: es un formulario básico que extiende de forms.Form

18. Modificar la vista para el template **index.html**. Editar el archivo **views.py** de la aplicación **moduloEmpleadoForm** y modificar la función **index** como se indica en la imagen.

```
moduloEmpleadoForm > views.py > ...
1  from django.shortcuts import render
2
3  from . import forms
4
5  # Create your views here.
6  def index(request):
7      form = forms.UserRegistrationForm()
8      data = {'form': form}
9      return render(request, 'templateModuloEmpleadoForm/index.html', data)
10
```


19. Editar el archivo **index.html** para incluir el formulario en el diseño.

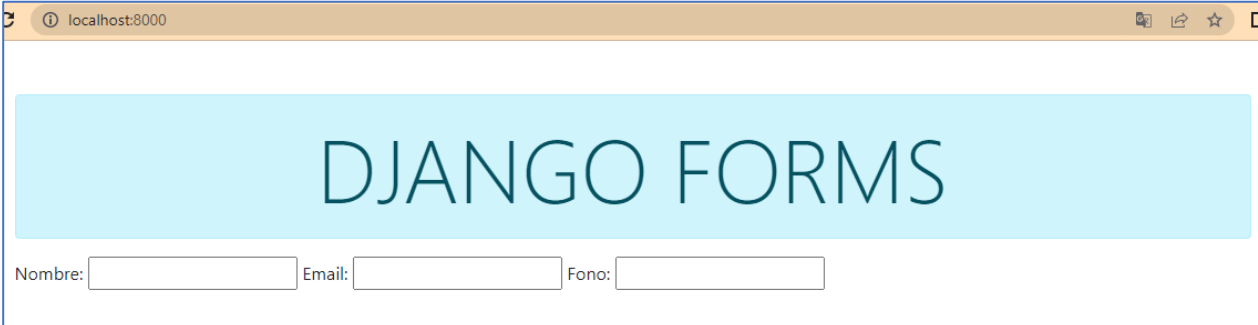
```
<body class="container mt-5">
  <div class="alert alert-info disp
    <form method="POST">
      {{form}}
    </form>
  </div>
</body>
```

20. Iniciar el servidor y revisar. Abrir el terminal del visual code

```
PS D:\Workspace-django-project\AdmEmpleadosForm> Python manage.py runserver
Watching for file changes with StatReloader
Performing system checks...

System check identified no issues (0 silenced).
August 15, 2022 - 14:10:05
Django version 4.1, using settings 'AdmEmpleadosForm.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CTRL-BREAK.
```

Revisar en el navegador



localhost:8000

DJANGO FORMS

Nombre: Email: Fono:

21. Editar el archivo **index.html** para incluir cambios en el diseño.

```
<form method="POST">
  {{form.as_p}}
</form>
```

desplegarlo como párrafo

DJANGO FORMS

Nombre:

Email:

Fono:

► Formas para renderizar el formulario en pantalla

Form rendering options

There are other output options though for the `<label>/<input>` pairs:

- `{{ form.as_div }}` will render them wrapped in `<div>` tags.
- `{{ form.as_table }}` will render them as table cells wrapped in `<tr>` tags.
- `{{ form.as_p }}` will render them wrapped in `<p>` tags.
- `{{ form.as_ul }}` will render them wrapped in `` tags.

Note that you'll have to provide the surrounding `<table>` or `` elements yourself.

```
<form method="POST">
  <table class="table">
    | {{form.as_table }}
  </table>
</form>
```

ordenado en una tabla

localhost:8000

DJANGO FORMS

Nombre:	<input type="text"/>
Email:	<input type="text"/>
Fono:	<input type="text"/>

Para incluir el botón Guardar

```
<form method="POST">
  <table class="table">
    {{form.as_table }}

    <tr>
      <td colspan="2">
        <input type="submit" class="btn btn-success" value="GUARDAR">
      </td>
    </tr>
  </table>
</form>
```

DJANGO FORMS

Nombre:	<input type="text"/>
Email:	<input type="text"/>
Fono:	<input type="text"/>
<input type="submit" value="GUARDAR"/>	

```
<tr>
  <td colspan="2">
    <input type="submit" class="btn btn-success" value="GUARDAR">
  </td>
</tr>
```

DJANGO FORMS

Nombre:	<input type="text"/>
Email:	<input type="text"/>
Fono:	<input type="text"/>
<input type="submit" value="GUARDAR"/>	

22. Editar el archivo **index.html** para incluir un token de seguridad **csrf_token**

```
<form method="POST">
  <table class="table">
    {{form.as_table}}
    {% csrf_token %}
  <tr>
    <td colspan="2">
      <input type="submit" class="btn btn-success" value="GUARDAR">
    </td>
  </tr>
</table>
</form>
```

token que agrega el hash de seguridad

Para ver el texto...inspeccionamos el html. Colocamos input a texto y aparece el hash



```
<input type="hidden" name="csrfmiddlewaretoken" value="kDF5RnoE5Vh8JYAj6ftodnr5zQ1mnAhrcgud00s603nnS7QScmKnMBV6I6DfGtvE">
```

```
<input type="hidden" name="csrfmiddlewaretoken"
value="kDF5RnoE5Vh8JYAj6ftodnr5zQ1mnAhrcgud00s603nnS7QScmKnMBV6I6DfGtvE">
```

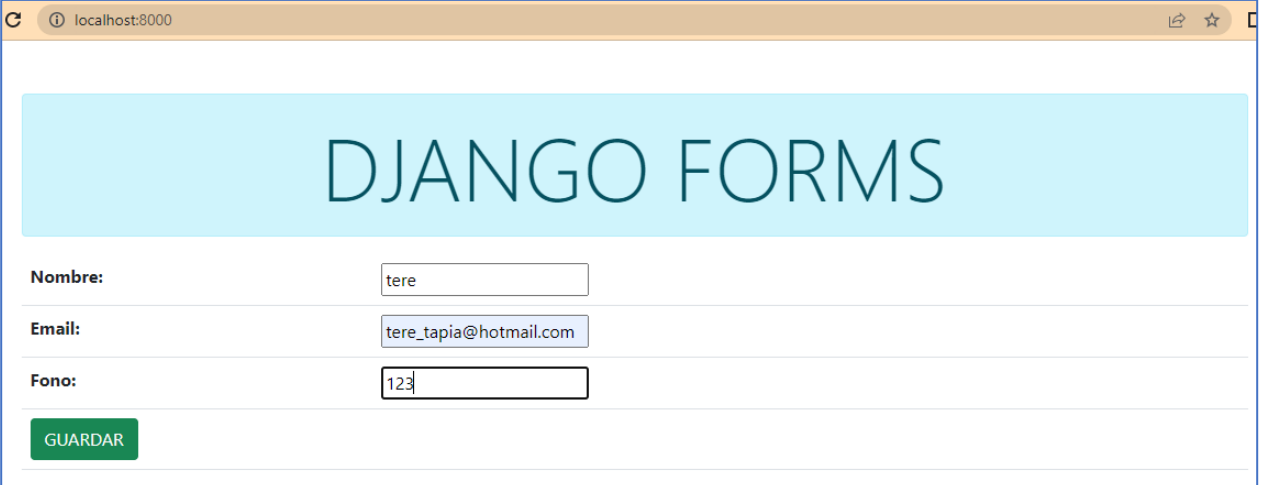
Al refrescar la página se va actualizando el token

```
value="aY0mrSrI2SrEXG86LNl90DBR4dq2Cp9L2BPuAajvaX0xT6PoFRUC8zR5Sdt2VVinY">
```

23. Modificar la vista para el template **index.html**. Procesar la petición request guardando la información del formulario. Editar el archivo **views.py** de la aplicación **moduloEmpleadoForm** como se indica en la imagen.

```
moduloEmpleadoForm > views.py > index
1  from django.shortcuts import render
2
3  from . import forms
4
5  # Create your views here.
6  def index(request):
7      form = forms.UserRegistrationForm()
8
9      if request.method == 'POST':
10         form = forms.UserRegistrationForm(request.POST)
11         if form.is_valid():
12             print("Form es Valido")
13             print("Nombre: ", forms.cleaned_data['nombre'])
14             print("Email: ", forms.cleaned_data['email'])
15             print("Fono: ", forms.cleaned_data['fono'])
16
17         data = {'form': form}
18         return render(request, 'templateModuloEmpleadoForm/index.html', data)
19
20
```

24. Revisar el index.html en el navegador, ingresar datos y presionar el botón Guardar



localhost:8000

DJANGO FORMS

Nombre:

Email:

Fono:

25. Al presionar el botón Guardar del formulario del index.html, revisar la ejecución en la consola del terminal

```
Starting development server at http://127.0.0.1:8000/
Quit the server with CTRL-BREAK.
[15/Aug/2022 15:07:15] "GET / HTTP/1.1" 200 1821
D:\Workspace-django-project\AdmEmpleadosForm\AdmEmpleadosForm\urls.py changed, reloading.
Watching for file changes with StatReloader
Performing system checks...

System check identified no issues (0 silenced).
August 15, 2022 - 15:10:10
Django version 4.1, using settings 'AdmEmpleadosForm.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CTRL-BREAK.
[15/Aug/2022 15:10:19] "GET / HTTP/1.1" 200 1821
[15/Aug/2022 15:18:18] "GET / HTTP/1.1" 200 1820
Form es Valido
Nombre: tere
Email: tere_tapia@hotmail.com
Fono: 123
[15/Aug/2022 15:19:07] "POST / HTTP/1.1" 200 1876
[]
```

26. Editar el archivo **forms.py** y agregar nuevos estilos al form. Agregar al campo password la visual de esconder la palabra a digitar y a los demás asignarle un estilo de clase a los campos input texto.

```
moduloEmpleadoForm > forms.py > ...
1  from django import forms
2
3  class UserRegistrationForm(forms.Form):
4      nombre =forms.CharField()
5      email =forms.CharField()
6      fono =forms.CharField()
7      password=forms.CharField(widget=forms.PasswordInput)
8
9      nombre.widget.attrs['class']='form-control'
10     email.widget.attrs['class']='form-control'
11     fono.widget.attrs['class']='form-control'
12     password.widget.attrs['class']='form-control'
13
```

27. Editar el archivo **forms.py** y agregar especificación del campo email

```
moduloEmpleadoForm > forms.py > ...
1  from django import forms
2
3  class UserRegistrationForm(forms.Form):
4      nombre =forms.CharField()
5      email =forms.CharField(widget=forms.EmailInput)
6      fono =forms.CharField()
7      password=forms.CharField(widget=forms.PasswordInput)
8
9      nombre.widget.attrs['class']='form-control'
10     email.widget.attrs['class']='form-control'
11     fono.widget.attrs['class']='form-control'
12     password.widget.attrs['class']='form-control'
13
14
```

28. Editar el archivo `forms.py` y agregar especificación del campo estado que es un select en el formulario

```
moduloEmpleadoForm > forms.py > ...
1  from django import forms
2
3  class UserRegistrationForm(forms.Form):
4
5      ESTADOS = (('activo', 'ACTIVO'), ('inactivo', 'INACTIVO'))
6
7      nombre = forms.CharField()
8      email = forms.CharField(widget=forms.EmailInput)
9      fono = forms.CharField()
10     password = forms.CharField(widget=forms.PasswordInput)
11     estados = forms.CharField(widget=forms.Select(choices=ESTADOS))
12
13     nombre.widget.attrs['class'] = 'form-control'
14     email.widget.attrs['class'] = 'form-control'
15     fono.widget.attrs['class'] = 'form-control'
16     password.widget.attrs['class'] = 'form-control'
17     estados.widget.attrs['class'] = 'form-control'
18
```


localhost:8000

DJANGO FORMS

Nombre:

Email:

Fono:

Password:

Estados:

- ACTIVO
- ACTIVO
- INACTIVO

GUARDAR

Y se agrega estado bloqueado.

```
moduloEmpleadoForm > forms.py > UserRegistrationForm
1  from django import forms
2
3  class UserRegistrationForm(forms.Form):
4
5      ESTADOS = [('activo', 'ACTIVO'), ('inactivo', 'INACTIVO'), ('bloqueado', 'BLOQUEADO')]
6
```

Estados:

- ACTIVO
- ACTIVO
- INACTIVO
- BLOQUEADO

GUARDAR

29. Implementar validaciones. Por defecto todos los campos son obligatorios.


DJANGO FORMS

Nombre:	<input type="text"/>
Email:	<input type="text"/>
Fono:	<input type="text"/>
Password:	<input type="password"/>
Estados:	<input type="text" value="ACTIVO"/>
<input type="button" value="GUARDAR"/>	

El fono no es obligatorio, en la siguiente imagen las validaciones se saltan el campo fono.

```
moduloEmpleadoForm > forms.py > ...
1  from django import forms
2
3  class UserRegistrationForm(forms.Form):
4
5      ESTADOS = [('activo', 'ACTIVO'), ('inactivo', 'INACTIVO'), ('bloqueado', 'BLOQUEADO')]
6
7      nombre = forms.CharField()
8      email = forms.CharField(widget=forms.EmailInput)
9      fono = forms.CharField(required=False)
10     password = forms.CharField(widget=forms.PasswordInput)
11     estados = forms.CharField(widget=forms.Select(choices=ESTADOS))
12
```

Nombre:	<input type="text" value="tere"/>
Email:	<input type="text" value="ed@d.cl"/>
Fono:	<input type="text"/>
Password:	<input type="password"/>
Estados:	<input type="text" value="ACTIVO"/>
<input type="button" value="GUARDAR"/>	

 Completa este campo

30. Implementar validaciones personalizadas. El campo nombre, tendrá un largo mínimo de 5 y máximo de 10

```

moduloEmpleadoForm > forms.py > UserRegistrationForm
2  from django import forms
3  from django.core import validators
4
5  class UserRegistrationForm(forms.Form):
6
7      ESTADOS=[('activo', 'ACTIVO'),('inactivo','INAC
8
9      nombre =forms.CharField(validators=[
10         validators.MinLengthValidator(5),
11         validators.MaxLengthValidator(10)
12     ])
13
14     email =forms.CharField(widget=forms.EmailInput)
15     fono =forms.CharField(required=False)
16     password=forms.CharField(widget=forms.PasswordIn
17     estados=forms.CharField(widget=forms.Select(choi
18
19     nombre.widget.attrs['class']='form-control'
20     email.widget.attrs['class']='form-control'
21     fono.widget.attrs['class']='form-control'
22     password.widget.attrs['class']='form-control'
23     estados.widget.attrs['class']='form-control'
  
```

Nombre: • Asegúrese de que este valor tenga al menos 5 carácter(es) (tiene4).

Email: a@d.cl

Fono: 123

Password:

Estados: ACTIVO

GUARDAR

Siempre valida la obligatoriedad de los campos y después procede a las validaciones particulares

Nombre: • Asegúrese de que este valor tenga menos de 10 caracteres (tiene 20).

31. Implementar validaciones personalizadas. El campo nombre no puede ser Juan

```

moduloEmpleadoForm > forms.py > UserRegistrationForm
1  from django import forms
2  from django.core import validators
3
4  class UserRegistrationForm(forms.Form):
5
6      ESTADOS = [('activo', 'ACTIVO'), ('inactivo', 'INACTIVO'), ('bloqueado', 'BLOQUEADO')]
7
8      nombre = forms.CharField(validators=[
9          validators.MinLengthValidator(2),
10         validators.MaxLengthValidator(10)
11     ])
12
13     email = forms.CharField(widget=forms.EmailInput)
14     fono = forms.CharField(required=False)
15     password = forms.CharField(widget=forms.PasswordInput)
16     estados = forms.CharField(widget=forms.Select(choices=ESTADOS))
17
18     def clean_nombre(self):
19         inputNombre = self.cleaned_data['nombre']
20         if inputNombre == 'Juan':
21             raise forms.ValidationError("No se aceptan más juanes")
22         return inputNombre
23
24
25     nombre.widget.attrs['class'] = 'form-control'
26     email.widget.attrs['class'] = 'form-control'
27     fono.widget.attrs['class'] = 'form-control'
28     password.widget.attrs['class'] = 'form-control'
29     estados.widget.attrs['class'] = 'form-control'
30

```

Para crear una función de validación por atributo debe ser siempre escrito `clean_nombreatributo`. En `cleaned_data` viene toda la información validada.

Nombre:	<ul style="list-style-type: none">• No se aceptan más juanes
	<input type="text" value="Juan"/>

32. Implementar validaciones personalizadas. Las validaciones se pueden hacer todas juntas si ocupamos la función `clean`.

```
18 def clean_nombre(self):
19     inputNombre=self.cleaned_data['nombre']
20     if inputNombre == 'Juan':
21         raise forms.ValidationError("No se aceptan más juanes")
22     return inputNombre
23
24
25 def clean(self):
26     user_clean_data =super().clean()
27
28     inputNombre =user_clean_data['nombre']
29     if inputNombre == 'Juan':
30         raise forms.ValidationError("No se aceptan más juanes")
31
32
33
```