## EXERCISES QUE TRABAJAREMOS EN EL CUE

- **EXERCISE 1: DJANGO.CONTRIB.ADMIN**
- **EXERCISE 2: DJANGO.CONTRIB.AUTH**
- **EXERCISE 3: DJANGO.CONTRIB.SESSION**
- **EXERCISE 4: DJANGO.CONTRIB.CONTENTTYPES**

## EXERCISE 1: DJANGO.CONTRIB.ADMIN

El administrador de Django es una interfaz de usuario generada automáticamente para los modelos de Django. La interfaz de administración se puede personalizar en gran medida, a continuación visualizamos cómo implementar algunas de las partes de la personalización de una aplicación.

Por ejemplo, en nuestro proyecto de `django-orm-postgresql` tenemos en la aplicación de relaciones un modelo definido entre Automotriz y modelos de carros. Podemos personalizarlo en el administrador de Django de la siguiente manera:

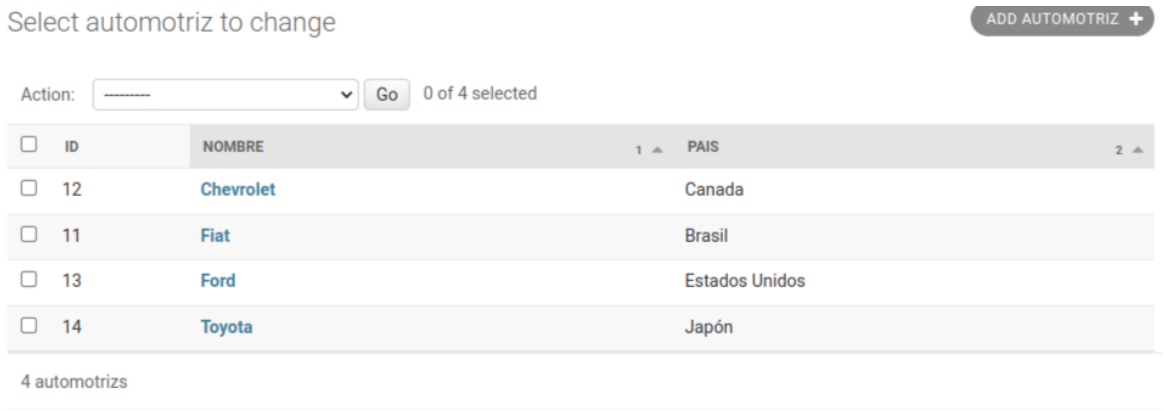
**RELACIONES/ADMIN.PY**

```
from django.contrib import admin

from .models import Automotriz, ModeloCarro

class AutomotrizAdmin(admin.ModelAdmin):
    fields = ['nombre', 'pais']
    list_display = ('id','nombre', 'pais')
    list_display_links = ['nombre']
    ordering = ('nombre','pais')


class ModeloCarroAdmin(admin.ModelAdmin):
    list_display = ('id','nombre', 'automotriz', 'f_fabricacion',
'costo_fabricacion', 'precio_venta', 'costo')
    ordering = ('automotriz',)
    list_display_links = ['nombre','automotriz']
    list_filter = ('nombre', 'automotriz')
    list_per_page = 4

    def costo(self, obj):
        return "Costo Alto" if obj.precio_venta >= 5000 else "Costo
Bajo"

admin.site.register(Automotriz, AutomotrizAdmin)
admin.site.register(ModeloCarro, ModeloCarroAdmin)
```

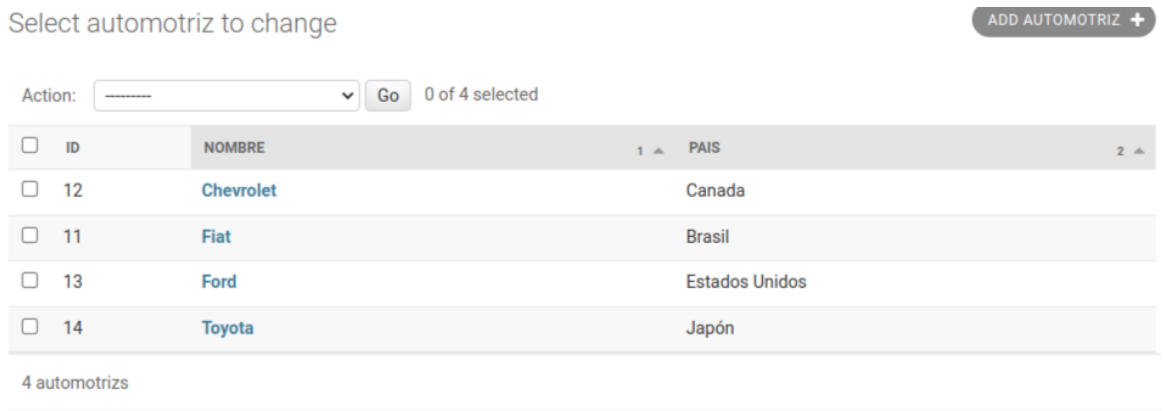
# REVISANDO LAS APLICACIONES PREINSTALADAS

Al ingresar al sitio administrativo de Django tenemos:

En Automotriz: http://localhost:8000/admin/relaciones/automotriz/




Para los modelos de carro en: http://localhost:8000/admin/relaciones/modelocarro/




Una de las partes más poderosas de Django es la interfaz de administración automática. Lee los metadatos de sus modelos para proporcionar una interfaz rápida y centrada en el modelo, donde los usuarios de confianza pueden administrar el contenido de su sitio. El uso recomendado por el administrador se limita a la herramienta de gestión interna de una organización; no está diseñado para construir todo el frontend.

El administrador tiene muchas opciones para la personalización, pero se debe ser cuidadoso al intentar usar estas opciones exclusivamente. Si necesita proporcionar una interfaz más centrada en el proceso, que abstraiga los detalles de implementación de las tablas y los campos de la base de datos, entonces probablemente diseñe las propias vistas. Puedes leer parte de la documentación aquí.

## EXERCISE 2: DJANGO.CONTRIB.AUTH

El sistema de autenticación de Django muestra la configuración predeterminada del framework. Esta configuración permite satisfacer las necesidades de los proyectos más comunes, maneja un conjunto de tareas, y tiene una implementación detallada de permisos y contraseñas. Es decir, brinda a los proyectos las necesidades de autenticación predeterminadas, permitiendo una amplia extensión y personalización de la autenticación.

Vamos a crear un registro de usuarios en el sistema. Creamos una nueva aplicación llamada usuario en la terminal de nuestro proyecto:

```
python manage.py startapp usuario
```

Agregamos la aplicación al proyecto:

**CONFIG/SETTINGS.PY**

```
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    # App Local
    'blogsite.apps.BlogsiteConfig',
    'relaciones.apps.RelacionesConfig',
    'migraciones.apps.MigracionesConfig',
    'crudapp.apps.CrudappConfig',
    'usuario.apps.UsuarioConfig',

]
```

Agregamos la URLs al proyecto de la aplicación:

**CONFIG/URLS.PY**

```
urlpatterns = [
    path('admin/', admin.site.urls),
    path('post/', include('blogsite.urls')),
    path('crudapp/', include('crudapp.urls')),
    path('usuario/', include('usuario.urls')),
]
```

Configuramos la URLs de la aplicación:

**USUARIO/URLS.PY**

```
from django.urls import path

from .views import registro_view

urlpatterns = [
    path('registro/',  registro_view, name='registro'),
]
```

Creamos el archivo de formulario, haciendo uso de la clase `Forms` y de la clase `meta User`:

**USUARIO/FORMS.PY**

```
from django import forms
from django.contrib.auth.forms import UserCreationForm
from django.contrib.auth.models import User

class RegistroUsuarioForm(UserCreationForm):
    email = forms.EmailField(required=True)

    class Meta:
        model = User
        fields = ("username", "email", "password1", "password2")

    def save(self, commit=True):
        user = super(RegistroUsuarioForm, self).save(commit=False)
        user.email = self.cleaned_data['email']
        if commit:
            user.save()
        return user
```

Procedemos a crear la vista:

**USUARIO/VIEWS.PY**

```
from django.shortcuts import render
from django.http import HttpResponseRedirect
from django.contrib import messages
from django.contrib.auth import login, authenticate, logout

from .forms import RegistroUsuarioForm

# Create your views here.

def registro_view(request):
```

```
    if request.method == "POST":
        form = RegistroUsuarioForm(request.POST)
        if form.is_valid():
            print("entro")
            user = form.save()
            login(request, user)
            messages.success(request, "Registrado Satisfactoriamente.")
            return HttpResponseRedirect ('/crudapp/mostrar/')
        messages.error(request, "Registro invalido. Algunos datos son
incorrectos.")
    form = RegistroUsuarioForm()
    return render (request= request, template_name="registro.html",
context={"register_form":form})
```

Creamos el template básico:

**USUARIO/TEMPLATE/REGISTRO.HTML**

```
<div class="container py-5">
    <h1>Registro</h1>
    <form method="POST">
      {% csrf_token %} {{ register_form }}
      <button class="btn btn-primary" type="submit">Registro</button>
    </form>
    <p class="text-center">
      Si tienes actualemnte una cuenta, inicia su sesión
      <a href="login">login</a>
    </p>
  </div>
```

Verificamos en: http://localhost:8000/usuario/registro/

## EXERCISE 3: DJANGO.CONTRIB.SESSION

Toda la comunicación entre los navegadores web y los servidores se realiza a través de HTTP, que no tiene estado. El hecho de que el protocolo no tenga estado significa que los mensajes entre el cliente y el servidor son completamente independientes entre sí; no existe una noción de "secuencia" o comportamiento basado en mensajes anteriores. Como resultado, si desea tener un sitio que realice un seguimiento de las relaciones en curso con un cliente, debe implementarse.

Las sesiones son el mecanismo utilizado por Django (y la mayor parte de Internet) para realizar un seguimiento del "estado" entre el sitio y un navegador en particular. Éstas permiten almacenar datos arbitrarios por navegador, y tenerlos disponibles para el sitio cada vez que se conecta el navegador.

Django usa una cookie que contiene una identificación de sesión especial para identificar cada navegador y su sesión asociada con el sitio. Los datos reales de la sesión se almacenan en la base de datos del sitio de forma predeterminada (esto es más seguro que almacenar los datos en una cookie, donde son más vulnerables a los usuarios malintencionados).

Por ejemplo: vamos a actualizar nuestra aplicación con la finalidad de mostrar cuántas veces ha visitado la página el usuario de los registros de los empleados en http://localhost:8000/crudapp/mostrar/

Procedemos a actualizar la vista:

**CRUDAPP/VIEWS.PY**

```python
def mostrar_emp(request):
    empleados = Empleado.objects.all()
    num_visits = request.session.get('num_visits', 0)
    request.session['num_visits'] = num_visits + 1
    context = {
        'empleados':empleados,
        'num_visits': num_visits,
    }

    return render(request,'mostrar.html', context)
```

Seguidamente, actualizamos la plantilla agregando al final la siguiente linea:

```
<p>Usted ha visitado esta página {{ num_visits }} veces.</p>
```

**CRUDAPP/TEMPLATES/MOSTRAR.HTML**

```
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-
scale=1.0">
    <title>Buscar Empleados</title>
</head>
<style>
    table {
        border: 8px outset;
        border-radius: 10px;
        border-spacing: 10px;
        padding: 20px;
        margin-left: auto;
        margin-right: auto;
    }

    th,
    td {
        padding: 5px;
        border: 1px solid;
    }
</style>

<body>
    <h2 style="text-align:center"> Información de Empleados </h2>
    <table align="center" style="margin: 0px auto;">
        <thead>
            <tr>
                <th>Id de Empleado</th>
                <th>Nombre</th>
                <th>Correo</th>
                <th>Designación</th>
                <th>Edit</th>
                <th>Delete</th>
            </tr>
        </thead>
        <tbody>
            {% for empleado in empleados %}
            <tr>
                <td>{{empleado.emp_id}}</td>
                <td>{{empleado.emp_nombre}}</td>
                <td>{{empleado.emp_correo}}</td>
                <td>{{empleado.emp_designacion}}</td>
                <td>
                    <a
href="/crudapp/editar/{{empleado.pk}}">Actualizar</a>
```

```
                </td>
                <td>
                    <a
href="/crudapp/eliminar/{{empleado.pk}}">Eliminar</a>
                </td>
            </tr>
            {% endfor %}
        </tbody>
    </table>

    <br><br>
    <div class="alert alert-danger" role="alert">
        ¿Quieres ingresar más empleadas?
    </div>

    <p>
        <a href="{% url 'insertar-emp' %}"><-- Ir a la pagina de
Inicio</a>
    </p>
    <p>Usted ha visitado esta página {{ num_visits }} veces.</p>
</body>

</html>
```

Verificamos en el sitio web: http://localhost:8000/crudapp/mostrar/

## Información de Empleados

| Id de Empleado | Nombre | Correo | Designación | Edit | Delete |
|---|---|---|---|---|---|
| 452 | Jose Camargo | jose@test.com | Desarrollador Web | Actualizar | Eliminar |
| 123 | Juan Perez | juan@test.com | Project Manager | Actualizar | Eliminar |

¿Quieres ingresar más empleadas?

<-- Ir a la pagina de Inicio

Usted ha visitado esta página 42 veces.

## EXERCISE 4: DJANGO.CONTRIB.CONTENTTYPES

`ContentTypes` es un modelo especial de Django que registra cada uno de los modelos que existen dentro de nuestra aplicación, tanto los que nosotros creamos, como los que vienen instalados por defecto.

Este sirve para relacionar modelos con otros modelos, como si fuera una foreign key (llave foránea), pero con la ventaja de que el tipo de modelo con el cual lo relacionemos puede ser diferente para cada entrada de la tabla.

También permite crear un modelo que haga referenciar a cualquiera de nuestros modelos creados. Cada objeto del modelo `ContentType` tendrá una propiedad llamada `app_label`, y otra model, las cuales son el nombre de la aplicación y el nombre del modelo, respectivamente.

Por ejemplo:

```
$ python manage.py shell
```

Importamos la librería.

```
>>> from django.contrib.contenttypes.models import ContentType
```

Importamos el modelo de empleado que está en la aplicación crudapp.

```
>>> from crudapp.models import Empleado
```

Verificamos:

```
>>> ContentType.objects.get_for_model(Empleado)
```

Salida:

```
<ContentType: crudapp | empleado>
```

Cada entrada de una tabla tiene un identificador único, la `id`.

```
>>> ContentType.objects.get(app_label='crudapp', model="empleado").id
```

Salida:

```
13
```

Existen más modelos registrados en la app de `ContentType`.

```
>>> ContentType.objects.get(id=1)
<ContentType: admin | log entry>
>>> ContentType.objects.get(id=2)
<ContentType: auth | permission>
>>> ContentType.objects.get(id=3)
<ContentType: auth | group>
```