Módulos del Sistema

Nota: proyecto propuesto = Sistema de inventario, compras y facturación

**MODULOS DEL SISTEMA**

INV

CMP

FAC

BASE

**APP**

Vamos a desarrollar el contenido cada una de las aplicaciones

BASE: contiene los siguientes archivos:

* Plantilla base
* Clase modelo (contendrá una clase del cual se va a heredar ya que se tiene varios campos repetidos)
  + Class ClaseModelo (models.Model):
    - Estado
    - Fecha\_creada
    - Fecha\_modificada
    - Usuario\_crea
    - Usuario\_modifica

App **inv** contiene los siguientes campos

* Categorías
* Subcategorías
* Marca
* Unidad de medida
* Inventario

App cmp contiene los siguientes campos (falta completar)

App cmp contiene los siguientes campos (falta completar)

**Modelos Catalogo APP INV**

**Subcategoria**

* id
* descripcion
* estado
* fecha\_creada
* fecha\_modificada
* usuario\_crea
* usuario\_modifica

**Categoria**

* id
* descripcion
* estado
* fecha\_creada
* fecha\_modificada
* usuario\_crea
* usuario\_modifica

**Unidad de Medida**

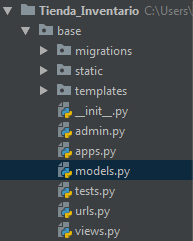
* id
* descripcion
* estado
* fecha\_creada
* fecha\_modificada
* usuario\_crea
* usuario\_modifica

**Marca**

* id
* descripcion
* estado
* fecha\_creada
* fecha\_modificada
* usuario\_crea
* usuario\_modifica

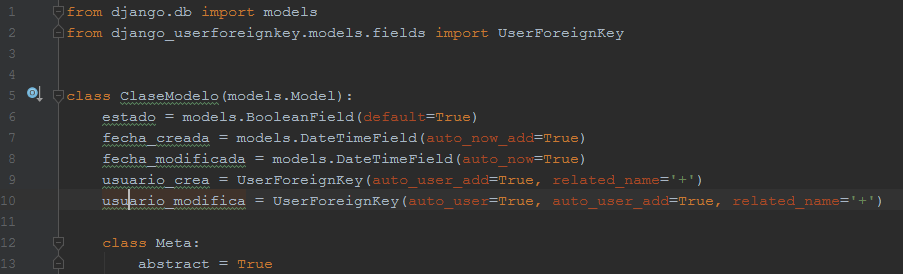
**Producto**

* Id
* codigo
* codigo\_barra
* descripcion
* precio
* existencia
* marca
* unidad\_medida
* subcategoria
* estado
* fecha\_creada
* fecha\_modificada
* usuario\_crea
* usuario\_modifica

vamos a crear el modelo dentro del app “base” en models

**MODELO**

Vamos a crear la clase “ClaseModelo”

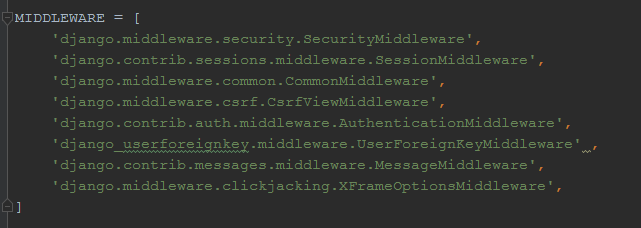


UserForeignKey: Django UserForeignKey es una aplicación simple de Django que le dará un campo de modelo UserForeignKey para modelos Django. Este campo extiende un campo de modelo de ForeignKey normal y tiene la opción de configurar automáticamente el usuario actualmente conectado al insertar y / o actualizar

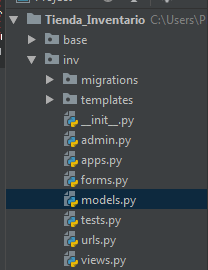
1. Descargue e instale

pip install django-userforeignkey

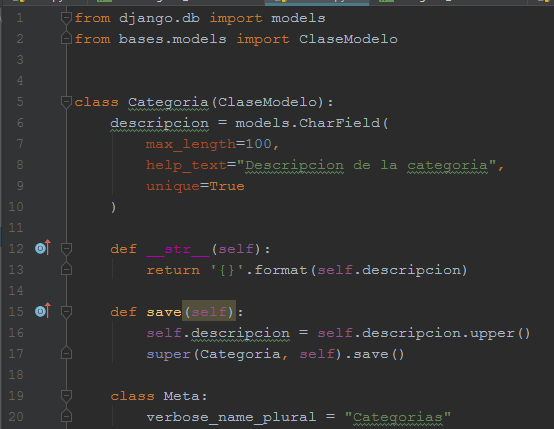
1. Agregue django\_userforeignkey a su configuración INSTALLED\_APPS de esta manera:
2. Agregue django\_userforeignkey.middleware.UserForeignKeyMiddleware a su configuración de MIDDLEWARE de esta manera



Comenzamos a crear la clase CATEGORIA dentro de la app “inv”

>>python manage.py startapp inv

­

 **from Django.db import models** (importamos models, )

**from base.models import ClaseModelo** (importamos la clase “ClaseModelo” del cual vamos a heredar)

descripcion = models.CharField(  
 max\_length=100,  
 help\_text="Descripcion de la categoria",  
 unique=True  
)

describimos los valores que va a tomar “descripcion”,

max\_length: longitud máxima de 100 caracteres

help\_text: texto que se guardará por defecto en la db

unique: único valor dentro de la bd, no se va a repetir.

def \_\_str\_\_(self):  
 return '{}'.format(self.descripcion)

redefine y retorna el valor del campo “descripcion”

def save(self):  
 self.descripcion = self.descripcion.upper()  
 super(Categoria, self).save()

redefinimos el valor de descripción con mayúscula, llamamos a la función save() de Categoria y lo guardamos.

class Meta:  
 verbose\_name\_plural = "Categorias"

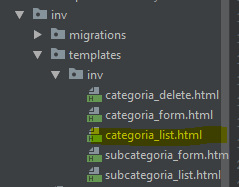
el nombre de la clase “Categoria” en plural lo redefinimos como = Categorias

Dado que es modelo, tenemos que ejecutar nuestro modelo dentro de la bd.

1. Creamos el archivo el cual a afectar a la bd
   * **Python manage.py makemigrations**
2. Ejecutamos el archivo.

**\* Python manage.py migrate inv**

**TEMPLATE**

Creamos el template para nuestra Categoria:

**Nota:** recordar que la carpeta dentro de templates tiene que tener el mismo nombre que la app.

{% extends 'base/base\_blank.html' %}  
  
{% block page\_content %}  
 <div class="card shadow mb-4">  
 <div class="card-header py-3 d-flex flex-row align-items-center justify-content-between">  
 <h6 class="m-0 font-weight-bold text-primary">Listado de Categorías</h6>  
 <div class="dropdown no-arrow">  
 <a class="dropdown-toggle" href="#" role="button" id="dropdownMenuLink" data-toggle="dropdown" aria-haspopup="true" aria-expanded="false">  
 <i class="fas fa-ellipsis-v fa-sm fa-fw text-gray-400"></i>  
 </a>  
 <div class="dropdown-menu dropdown-menu-right shadow animated--fade-in" aria-labelledby="dropdownMenuLink">  
 <div class="dropdown-header">Acciones: </div>  
 <a class="dropdown-item" href="{% url 'inv:categoria\_new' %}"><i class="far fa-file"></i> Nuevo</a>  
 </div>  
 </div>  
 </div>

{% extends 'base/base\_blank.html' %}

Estamos tomando como base el archivo base\_blank.html

{% block page\_content %}

{% endblock page\_content %}

Aquí va todo el contenido que vamos a insertar, solo se modificara el contenigo de este bloque, los otros bloques tendrán el contenido de la plantilla base.

<div class="dropdown-header">Acciones: </div>

- menu que tendrá como única accion a “nuevo”  
 <a class="dropdown-item" href="{% url 'inv:categoria\_new' %}"><i class="far fa-file"></i> Nuevo</a>

- enlace a :categoria\_new, llama a la vista “categoria\_new”

<div class="card-body">   
 {% if **not** objeto %}  
 No hay Categoria (objetos)  
 {% else %}

// verificamos si hay un objeto capturado (usado para editar)  
<table class="table table-striped table-hover">  
 <thead>  
 <th>ID</th>  
 <th>Descripcion</th>  
 <th>Estado</th>  
 <th>Fecha Creada</th>  
 <th>Fecha Modificada</th>  
 <th>Acciones</th>  
 </thead>

//muestra la cabecera de la table, desde id hasta Acciones.

<tbody>  
{% for item **in** objeto %}  
 <tr>  
 <td>{{ item.id }}</td>  
 <td>{{ item.descripcion }}</td>  
 <td>{{ item.estado| yesno:'activo, inactivo' }}</td>  
 <td>{{ item.fecha\_creada|date:"d/m/Y H:i:s" }}</td>  
 <td>{{ item.fecha\_modificada |date:"d/m/Y H:i:s" }}</td>  
 <td>  
 <a href="{% url 'inv:categoria\_edit' item.id %}" class="btn btn-warning btn-circle" role="button"><i class="far fa-edit"></i></a>  
 <a href="{% url 'inv:categoria\_delete' item.id %}" class="btn btn-danger btn-circle" role="button"><i class="far fa-trash-alt"></i></a>  
 </td>  
 </tr>  
{% endfor %}  
</tbody>

<tbody> //cuerpo de la tabla  
{% for item **in** objeto %}

//hacemos una iteración para los campos del objeto (en caso haya uno)

<tr>  
 <td>{{ item.id }}</td>  
 <td>{{ item.descripcion }}</td>  
 <td>{{ item.estado| yesno:'activo, inactivo' }}</td>  
 <td>{{ item.fecha\_creada|date:"d/m/Y H:i:s" }}</td>  
 <td>{{ item.fecha\_modificada |date:"d/m/Y H:i:s" }}</td>  
 <td>

//muestra los campos del objeto  
 <a href="{% url 'inv:categoria\_edit' item.id %}" class="btn btn-warning btn-circle" role="button"><i class="far fa-edit"></i></a>  
 <a href="{% url 'inv:categoria\_delete' item.id %}" class="btn btn-danger btn-circle" role="button"><i class="far fa-trash-alt"></i></a>  
 </td>

// muestra enlaces de “categoria\_edit(editar) y categoria\_delete(eliminar)”  
</tr>

{% block js\_page %}  
<script>  
 // Call the dataTables jQuery plugin  
 $(document).ready(**function**() {  
 $('.table').DataTable( {  
 dom: '<"top"fi><"toolbar">rt<"bottom"lpB><"clear">',  
 "paging": **true**,  
 "ordering": **true**,  
 "info": **true**,  
 "language": {  
 "decimal": ".",  
 "emptyTable": "No hay datos en la tabla",  
 "info": "Mostrando del \_START\_ al \_END\_ de \_TOTAL\_ registros",  
 "infoEmpty": "Mostrando 0 a 0 de 0 registros",  
 "infoFiltered": "(filtrados a partir de \_MAX\_ registros)",  
 "infoPostFix": "",  
 "thousands": ",",  
 "lengthMenu": "Mostrando \_MENU\_ registros",  
 "loadingRecords": "Cargando...",  
 "processing": "Procesando...",  
 "search": "Busqueda:",  
 "zeroRecords": "No se encontraron registros coincidentes",  
 "paginate": {  
 "first": "Primero",  
 "last": "Ultimo",  
 "next": "Siguiente",  
 "previous": "Anterior"  
 },  
 "aria": {  
 "sortAscending": ": activa la ordenación ascendente",  
 "sortDescending": ": activate la ordenanción descendente"  
 }  
 }  
 }  
  
 );  
 });  
</script>  
{% endblock js\_page %}

Un scritpt con el cual activamos el datatable y cambiamos el idioma a español.

**VISTA**

from django.contrib.auth.mixins import LoginRequiredMixin

// permite que el usuario pueda accede a las vistas, si se llama en una clase requiere que primero se haga login de lo contrario no se podrá visualizar.

from django.views import generic

// generic: importamos las vistas propias de Django y nos permite implemetarlo

from .models import Categoria  
// llamamos al modelo Categoria

from .forms import CategoriaForm

from django.urls import reverse\_lazy

// permite regresar a una vista una vez se haya llamado al formulario

from django.contrib.auth.mixins import LoginRequiredMixin  
from django.views import generic  
from .models import Categoria  
from .forms import CategoriaForm  
from django.urls import reverse\_lazy  
# Create your views here.  
  
  
class CategoriaView(LoginRequiredMixin, generic.ListView):  
 model = Categoria  
 template\_name = 'inv/categoria\_list.html'  
 context\_object\_name = 'objeto'  
 login\_url = 'bases:login'

class CategoriaView(LoginRequiredMixin, generic.ListView):

// hereda de LoginRequeridMixin= oblige a loguearte antes de ver la vista

// generic.ListView = ¿?

**URLS**

from django.urls import path  
from .views import CategoriaView, CategoriaNew, CategoriaEdit, CategoriaDelete  
  
urlpatterns =[  
 path('categorias/', CategoriaView.as\_view(), name='categoria\_list'),  
 path('categorias/new/', CategoriaNew.as\_view(), name='categoria\_new'),  
 path('categorias/edit/<int:pk>', CategoriaEdit.as\_view(), name='categoria\_edit'),  
 path('categorias/delete/<int:pk>', CategoriaDelete.as\_view(), name='categoria\_delete'),  
]

from django.urls import path

// importamos path para implementar las urls

from .views import CategoriaView

// importamos CategoriaView para usarlo en la llamada a url

urlpatterns =[  
 path('categorias/', CategoriaView.as\_view(), name='categoria\_list')   
]

// “categorias/”= sera parte o todo de nuestra url

// Categoria.as\_view()= llamamos a la vista y le asignamos la url con el nombre de “categoría\_list”

Incluir en un tag para llamar a la vista “categoría\_list” con la url “inv/categorias”