Ayuda para proyectos en Django.

Basados en los videos del proyecto de Algorisoft:

https://algorisoft.com/courses/41aa1066-20d7-42b6-8533-5470caf76b4b/

Contenido

Crear Proyecto

Crear la carpeta del Proyecto, crear dentro, las carpetas “app” que va a contener todo lo referido a la aplicación y la carpeta “venv” que va a contener el entorno virtual del proyecto
Para crear el entorno virtual por terminal:

```
python3 –m venv venv
```

Instalamos django en el entorno virtual con:

```
pip install django
```

Luego creamos el proyecto desde el raíz del mismo, en el mismo nivel que la carpeta del entorno virtual con el comando:

```
django-admin startproject app
```

Para chequear el proyecto, entramos a app y ejecutamos el comando:

```
python3 manage.py runserver
```

Renombrar la carpeta del proyecto “app” por config desde PYCHARM, para agrupar los archivos de configuración, haciendo un refactor por el renombramiento de la carpeta.
Luego procedemos a crear una carpeta llamada “apps” dentro de “app” y dentro de ella ejecutamos el comando:

```
python3 ../manage.py startapp nombreAplicacion
```

Configurar BD del proyecto

Dentro del setting la variable DATABASES contiene la configuración del motor de base de datos, por defecto es SQLITE.
Para SQLITE y POSTGRESQL: crear un archivo con lo siguiente (reemplazar por lo correspondiente USER y PASS)

Crear un archivo dentro de config que se llame db.py

```
import os
BASE_DIR = os.path.dirname(os.path.dirname(os.path.abspath(__file__)))

SQLITE = {
   'default': {
      'ENGINE': 'django.db.backends.sqlite3',
      'NAME': os.path.join(BASE_DIR, 'db.sqlite3'),
   }
}

POSTGRESQL = {
   'default': {
      'ENGINE': 'django.db.backends.postgresql_psycopg2',
      'NAME': 'db',
      'USER': 'postgres',
      'PASSWORD': 'postgres',
      'HOST': 'localhost',
      'PORT': '5432',
      'ATOMIC_REQUESTS': True
   }
}
```

Para poder ejecutar POSTGRESQL en el entorno virtual hay que instalar “psycopg2” mediante el comando:

```
pip install psycopg2-binary
```

Luego en setting importar:

```
import config.db as db y en databases seleccionar: DATABASES = db.SQLITE o DATABASES=db.POSTGRESQL
```

Modelos

Para crear un modelo se debe crear una clase heredando de la clase models.Model

Class Empleado(models.Model):
Class Tipo(models.Model):
Class Category(models.Model):

Para relacionar modelos (FK) 1 a 1 o 1 a Muchos

Definir mediante models.ForeignKey(tabla que referencia, on_delete=models.CASCADE, models.PROTECT, SET_NULL, null=True,)

Para crear relaciones Muchos a Muchos

categ = models.ManyToManyField(Category)

Esto crea una tabla intermedia (en este caso llamada empleado_categ con los id de categoría y de empleado)
Probablemente convenga crear la tabla intermedia de manera manual, para poder agregar campos adicionales.

Al definir campos se pueden utilizar atributos que tienen validaciones ya hechas. Como ser:

- Permitir ingresar valores nulos con null=True, blank=True
- CharField lleva max_lenght=100, TextField no requiere.
- Verbose_name='' (mapea que en el formulario el label tome ese valor indicado)
- Unique=True para que el campo sea único y no repetido
- Default= permite ingresar el valor por defecto, en date puede ser datetime.now
- En datetime se puede utilizar:
  - auto_now=True para que la primera vez se agregue el valor automatico.
  - auto_now_add=True para una actualización de registro.
- IntegerField, se puede utilizar PositiveIntegerField para campos como edad, que solo permita ingresar +
- DecimalField, se puede agregar la cantidad de dígitos con max_digits= y decimal_places=2 despues de la ","
- BooleanField, default=True/False
- ImageField y FileField, se puede especificar la carpeta donde se guarda con upload_to='XXXX' (esto hay que configurar en el SETTINGS en la parte de MEDIA). Se puede agregar formatos a la carpeta como 'XXXX/%YYYY/%mm%dd' para generar carpetas por año mes dia.
  El campo de imágenes necesita de una librería llamada Pillow, hay que instalarla en el entorno virtual.

Luego se define métodos.

__str__(self): (representación toString del objeto. Se definen los atributos que retorna la representación del obj.)
return self.names (retornar los nombres, por ej)
Solo puede devolver valores STRING, valores entero u otros arroja error.

class meta:
verbose_name='' (cuando se registre la aplicación en el módulo de administración de django)
verbose_name_plural='' (cuando se registre la aplicación en el módulo de administración de django)
db_table='empleado' (se asigna el nombre que va a tener la tabla (MODELO) en la BD)
Si no se coloca esto, toma por defecto el nombre de la app y de la clase del modelo. Por ej (app_empleado)
ordering = ['id'] ordering = ['-id'] descendente

Luego de definir los modelos, en el archivo SETTINGS en INSTALLED_APPS agregar la aplicación que contiene los modelos

'app.empleado' (por ejemplo)
Luego ejecutar el comando
python3 manage.py makemigrations
crea una migración para poder crear/actualizar/borrar las tablas/campos en la BD
python3 manage.py migrate

En DJANGO por defecto el framework crea el campo id como clave primaria.

Panel de administración DJANGO

En la app, en el archivo admin.py se registran los modelos y se coloca en la siguiente línea:
admin.site.register(category)

Para crear un super usuario se ejecuta el comando:

```
python3 manage.py createsuperuser
```

ORM Django

(Mapeo de Objeto Relacional) Podemos interactuar con la BD a manera de objetos para realizar insert, update, delete, consultas. Etc.

Para estos ejemplos se utiliza la tabla Type

#List (select * from type)
Type.objects.all()
Type.objects.filter(name_contains ='algo') la "i" pasa todo a mayúscula, compara y devuelve todos los registros
Type.objects.filter(name_startswith ='algo')
Type.objects.filter(name_in =['algo','algoMas']) devuelve todo lo que contenga dentro del vector
Type.objects.filter(name_in =['algo','algoMas']).count()
Type.objects.filter(name_in =['algo','algoMas']).query devuelve como esta armada la consulta SQL
Type.objects.filter(name_endswith ='o').exclude(id=1) excluye algún campo que pasemos como parametro
For i in Type.objects.filter(name_endswith ='o')[:3]: trae los últimos tres registros

```
print (i.name)
```

obj = Employee.objects.filter(type_id=1) traigo todos los empleados que están en una categoría determinada.

En filtros con fechas **date_algo__gt** es mayor o igual **date_algo__gte** es menor o igual

#Insert (insert into type values …..)
T = Type()
T.name = 'algo'
T.save()

También se puede usar:

T = Type(name='algo')
T.save()
T = Type(name='algo').save()

#Update

T = Type.objects.get(id=algo)
T.name = 'algoModificado'
T.save()

#Delete

T = Type.objects.get(id=algo)
T.delete()

Vistas y URL's

Django utiliza un Modelo Vista Template que es similar al Modelo Vista Controlador
Modelo (Base de datos) Vista (Funciones) Template (Pantallas)
Cuando Django solicita una página solicita un HttpRequest como primer argumento de la vista y retorna un objeto HttpResponse, JsonResponse, render
Por cada URL que se crea se tiene que asociar a una vista.

Proceder a crear primero la VISTA y luego la URL

A fines organizativos, por cada aplicación es recomendable agrupar las vistas y las urls de cada aplicación, por ello conviene crear una carpeta "views" y "templates" en la raíz de la aplicación y dentro de cada una de ellas una carpeta con el nombre del modelo que se va a crear la vista.
En el archivo views.py de la aplicacion
def myfirstview(request):

```
return HttpResponse('Esta es la respuesta')
```

En el archivo urls.py de la aplicacion

Incorporar una línea app_name = ‘nombreAplicacion’ para poder utilizar desde el template las vistas de cada APP.
En urlpatterns incorporar la vista

```
path(‘prueba/’, myfirstview, name=’vista1’),
```

En cada aplicación conviene crear el archivo urls.py para agrupar las urls relacionadas a cada aplicación.
Entonces en el urls.py principal:

```
path(‘prueba/’, include(‘core.erp.urls’))
```

Vistas basadas en funciones

Un ejemplo de un listado

```
def category_list(request):
    data = {
        'title': 'Listado de Categorías',
        'categories': Category.objects.all()
    }
    return render(request, 'category/list.html', data)
```

Forms

Para poder ocupar las vistas de CreateView, UpdateView y DeleteView debemos crear un archivo llamado forms.py en la raíz de la aplicación y va a contener los formularios.

Para poder personalizar los componentes del formulario se debe instalar la librería django_widget_tweaks
Esto mediante: pip install django_widget_tweaks
Y para activar se debe ir al archivo settings.py en INSTALLED_APPS agregar: ‘widget_tweaks’
Luego desde el archivo html se debe cargar la librería. {% load widget_tweaks %}
En cada field del formulario, dentro del FOR se lo utiliza mediante:
{{ field|add_class:'form-control'|attr:'autocomplete:off' }} #por ej

Crear formulario:

```
class CategoryForm(ModelForm):
    def __init__(self, *args, **kwargs):
        super().__init__(*args, **kwargs)
        # for form in self.visible_fields():
        #     form.field.widget.attrs['class'] = 'form-control'
        #     form.field.widget.attrs['autocomplete'] = 'off'
        self.fields['name'].widget.attrs['autofocus'] = True

    class Meta:
        model = Category
        fields = '__all__' ## para pasar un orden especifico hay que declarar cada campo [‘’,’’,’’,’’,’’]
        # exclude = [‘’,’’,’’,’’,’’] para poder quitar campos de la visualizacion
        widgets = {             ##Para poder personalizar cuestiones con cada componente.
            'name': TextInput(
                attrs={
                    'placeholder': 'Ingrese un nombre',
                    'autocomplete': 'off',          #para que no autocomplete con lo tipeado anteriormente
                }
            ),
            'desc': Textarea(
                attrs={
                    'placeholder': 'Ingrese un nombre',
                    'rows': 3,
                    'cols': 3
                }
            ),
        }
```

```
def save(self, commit=True):
    data = {}
    form = super()
    try:
        if form.is_valid():
            form.save()
        else:
            data['error'] = form.errors
    except Exception as e:
        data['error'] = str(e)
    return data
```

Vistas basadas en clases

Permiten un mejor mantenimiento del proyecto a medida que crece, que las vistas basadas en funciones. Al agregar un campo, modificar algo, etc.
Son vistas genéricas incorporadas en django que se pueden heredar y extender según lo que necesitemos. Por ej: ListView, CreateView, UpdateView, DeleteView.

ListView

```
class CategoryListView(ListView):
    model = Category
    template_name = 'category/list.html'
```

En el url_patterns se debe llamar a la vista Category.ListView.as_view() incorporando el “as_view()”

También hay que sobre escribir métodos:

```
    @method_decorator(csrf_exempt)
    def dispatch(self, request, *args, **kwargs):
        return super().dispatch(request, *args, **kwargs)

    def post(self, request, *args, **kwargs):
        data = {}
        try:
            action = request.POST['action']
            if action == 'searchdata':
                data = []
                for i in Category.objects.all():
                    data.append(i.toJSON())
            else:
                data['error'] = 'Ha ocurrido un error'
        except Exception as e:
            data['error'] = str(e)
        return JsonResponse(data, safe=False)

    def get_context_data(self, **kwargs):
        context = super().get_context_data(**kwargs)
        context['title'] = 'Listado de Categorías'
        context['create_url'] = reverse_lazy('erp:category_create')
        context['list_url'] = reverse_lazy('erp:category_list')
        context['entity'] = 'Categorias'
        return context

    def get_queryset(self):
        return Category.objetcs.filter()
```

Permite poder obtener un listado con filtro entre otras cosas.

Los decoradores son funciones que añaden funcionalidades a otra función. Permiten alterar de manera dinámica la funcionalidad. Por ej aplicar una validación. El ej más simple es aplicar el decorador de si un usuario esta logueado. Esto se aplica antes del método dispatch

@method_decorator(login_required)

@method_decorator(csrf_exempt) este método es necesario para sobreescribir los métodos POST. Este método permite proteger las vistas en el método POST.

El método dispatch se ejecuta al comienzo de la vista, se encarga de redireccionar para el tipo de método (POST, GET) dependiendo de la petición que se haga. Permite también interactuar con los decoradores. Y cuando se realicen sobre escritura de los métodos GET y POST.

```
def dispatch(self, request, *args, **kwargs):
    return super().dispatch(request, *args, **kwargs)
```

Datatables

Copiar dentro de los archivos estáticos en: static/lib/

Links de datatable

https://datatables.net/download/
https://datatables.net/examples/styling/bootstrap4
https://datatables.net/extensions/fixedheader/examples/styling/bootstrap4.html
https://datatables.net/examples/basic_init/language.html
http://cdn.datatables.net/plug-ins/1.10.20/i18n/Spanish.json
https://datatables.net/examples/basic_init/zero_configuration.html

Código para poder ejecutar:

Se agrega al template list.html.
En el home.html agregar el bloque head y en el list.html
Agregar lo siguiente dentro del bloque head. Reemplazar por la versión que se tenga.

```
<link rel="stylesheet" href="{% static 'lib/datatables-1.10.20/css/dataTables.bootstrap4.min.css' %}"/>
  <link rel="stylesheet" href="{% static 'lib/datatables-1.10.20/plugins/responsive-
2.2.3/css/responsive.bootstrap4.min.css' %}"/>
  <script src="{% static 'lib/datatables-1.10.20/js/jquery.dataTables.js' %}"></script>
  <script src="{% static 'lib/datatables-1.10.20/js/dataTables.bootstrap4.min.js' %}"></script>
  <script src="{% static 'lib/datatables-1.10.20/plugins/responsive-2.2.3/js/dataTables.responsive.min.js' %}"></script>
  <script src="{% static 'lib/datatables-1.10.20/plugins/responsive-2.2.3/js/responsive.bootstrap4.min.js' %}"></script>
```

En el body.html se carga un bloque de javascript casi al final

```
<!-- Block javascript -->
  {% block javascript %}

  {% endblock %}
```

En el list.html

```
{% block javascript %}
  <script type="application/javascript">
    $(function () {
      $('#data').DataTable({
        responsive: true,
        autoWidth: false,
        ##VER Cambiar el idioma editando el archivo.
        "language": {
          url : '{% static 'lib/datatables-1.10.20/spanish.txt' %}'
        }
      });
    });
  </script>
{% endblock %}
```

Cambiar el idioma

Crear un archivo spanish.txt dentro de la carpeta static/lib/datatables/ con lo siguiente:

```
{
          "sProcessing":    "Procesando...",
           "sLengthMenu":    "Mostrar _MENU_ registros",
           "sZeroRecords":   "No se encontraron resultados",
           "sEmptyTable":    "Ningún dato disponible en esta tabla =(",
           "sInfo":          "Mostrando registros del _START_ al _END_ de un total de _TOTAL_ registros",
           "sInfoEmpty":     "Mostrando registros del 0 al 0 de un total de 0 registros",
           "sInfoFiltered":  "(filtrado de un total de _MAX_ registros)",
           "sInfoPostFix":   "",
           "sSearch":        "Buscar:",
           "sUrl":           "",
           "sInfoThousands":  ",",
           "sLoadingRecords": "Cargando...",
           "oPaginate": {
              "sFirst":    "Primero",
              "sLast":     "Último",
              "sNext":     "Siguiente",
              "sPrevious": "Anterior"
           },
           "oAria": {
              "sSortAscending":  ": Activar para ordenar la columna de manera ascendente",
              "sSortDescending": ": Activar para ordenar la columna de manera descendente"
           },
           "buttons": {
              "copy": "Copiar",
              "colvis": "Visibilidad"
           }
}
```

Cambiar el idioma editando el archivo

La otra alternativa es ir a la librería misma (static/lib/datatables/js/jquery.datatables.js) y cambiar los campos citados arriba manualmente.

Datatables con Ajax

Esto se utiliza para optimizar el listado, que no consulte a la BD. Video 33
Dentro de la carpeta de la aplicación crear una carpeta que se llame “static”, dentro de ella crear una carpeta con el nombre del modelo y dentro de ella crear un archivo que se llame “list.js”

Dentro de ese archivo va el código siguiente, modificar según los datos del modelo. Recordar que va un campo duplicado al final por el apartado de las opciones “Botones” (para editar, borrar, etc).

```
$(function () {
   $('#data').DataTable({
      responsive: true,
      autoWidth: false,
      destroy: true,
      deferRender: true,
      ajax: {
         url: window.location.pathname,
         type: 'POST',
         data: {
            'action': 'searchdata'
         },
         dataSrc: ""
      },
```

```
        columns: [
            {"data": "position"},
            {"data": "name"},
            {"data": "desc"},
            {"data": "desc"},
        ],
        columnDefs: [
            {
                targets: [-1],
                class: 'text-center',
                orderable: false,
                render: function (data, type, row) {
                        var buttons = '<a href="/erp/category/update/' + row.id + '/" class="btn btn-warning btn-xs btn-flat"><i
                        class="fas fa-edit"></i></a> ';
                        buttons += '<a href="/erp/category/delete/' + row.id + '/" type="button" class="btn btn-danger btn-xs
                        btn-flat"><i class="fas fa-trash-alt"></i></a>';
                    return buttons;
                }
            },
        ],
        initComplete: function (settings, json) {
        }
    });
});
```

Para poder utilizar esto en el list.html se agrega el “list.js” creado para poder tomar de ahí los datos.

CreateView

```
class CategoryCreateView(CreateView):
    model = Category
    form_class = CategoryForm #Se importa el formulario que toma como base
    template_name = 'category/create.html' #Se importa el template que va a utilizar.
    success_url = reverse_lazy('erp:category_list')          #redirecciona cuando se realiza la operación exitosamente.

def post(self, request, *args, **kwargs):
        data = {}
        try:
            action = request.POST['action']
            if action == 'add':
                form = self.get_form()
                data = form.save()
            else:
                data['error'] = 'No ha ingresado ninguna opción'
        except Exception as e:
            data['error'] = str(e)
        return JsonResponse(data)

def get_context_data(self, **kwargs):
        context = super().get_context_data(**kwargs)
        context['title'] = 'Crear una Categoria'
        context['entity'] = 'Categorias'
        context['list_url'] = reverse_lazy('erp:category_list')
        context['action'] = 'add'
        return context
```

Se debe crear el template de cada funcionalidad dentro de la carpeta template de la aplicación. Create, Update y Delete referencian a la plantilla principal form.html

UpdateView

Es muy similar al CreateView. En el urls, se le agrega en el path /<int:pk>/

En el list.html se agrega el parámetro c.id al a href con la dirección de la url

```
class CategoryUpdateView(UpdateView):
    model = Category
    form_class = CategoryForm
    template_name = 'category/create.html'
    success_url = reverse_lazy('erp:category_list')
    permission_required = 'update_category'
    url_redirect = success_url

    def dispatch(self, request, *args, **kwargs):
        self.object = self.get_object()
        return super().dispatch(request, *args, **kwargs)

    def post(self, request, *args, **kwargs):
        data = {}
        try:
            action = request.POST['action']
            if action == 'edit':
                form = self.get_form()
                data = form.save()
            else:
                data['error'] = 'No ha ingresado ninguna opción'
        except Exception as e:
            data['error'] = str(e)
        return JsonResponse(data)

    def get_context_data(self, **kwargs):
        context = super().get_context_data(**kwargs)
        context['title'] = 'Editar una Categoria'
        context['entity'] = 'Categorias'
        context['list_url'] = self.success_url
        context['action'] = 'edit'
        return context
```

DeleteView

Es muy similar al CreateView. En el urls, se le agrega en el path /<int:pk>/

En el list.html se agrega el parámetro c.id al a href con la dirección de la url

Se crea un template base delete.html que se basa en el form.html, sin los campos. Y se hereda en la aplicación el template.

```
class CategoryDeleteView(DeleteView):
    model = Category
    template_name = 'category/delete.html'
    success_url = reverse_lazy('erp:category_list')
    permission_required = 'delete_category'
    url_redirect = success_url

    def dispatch(self, request, *args, **kwargs):
        self.object = self.get_object()
        return super().dispatch(request, *args, **kwargs)

    def post(self, request, *args, **kwargs):
        data = {}
```

```
        try:
            self.object.delete()
        except Exception as e:
            data['error'] = str(e)
        return JsonResponse(data)

    def get_context_data(self, **kwargs):
        context = super().get_context_data(**kwargs)
        context['title'] = 'Eliminar Categoria'
        context['entity'] = 'Categorias'
        context['list_url'] = self.success_url
        return context
```

LoginView

Es una vista genérica que trae Django que permite autentificar el Login. Se debe crear una aplicación login y se la guarda dentro de la carpeta apps. Se utiliza el archivo views.py por defecto. Se crea la carpeta templates y dentro de ella creamos el archivo login.html

```
class LoginFormView(LoginView):
    template_name = 'login.html'

    def dispatch(self, request, *args, **kwargs):
        if request.user.is_authenticated:                ##se utiliza si el usuario esta logueado redirecciona.
            return redirect(setting.LOGIN_REDIRECT_URL)
        return super().dispatch(request, *args, **kwargs)

    def get_context_data(self, **kwargs):
        context = super().get_context_data(**kwargs)
        context['title'] = 'Iniciar sesión'
        return context
```

LogoutView

Se realiza el logout mediante la clase genérica RedirectView, se llama al método logout y se pasa el redirect_url hacia login

```
class LogoutRedirectView(RedirectView):
    pattern_name = 'login'

def dispatch(self, request, *args, **kwargs):
        logout(request)
        return super().dispatch(request, *args, **kwargs)
```

Se debe agregar al archivo urls.py general el path hacia /login/ y el include del urls.py de la aplicación login

En el archivo settings.py se debe agregar el apartado de LOGIN_REDIRECT_URL

```
LOGIN_REDIRECT_URL = '/dashboard/'
LOGOUT_REDIRECT_URL = '/login/'
```

Validación de formularios

Video 28

Se sobreescribe el método POST de la vista para poder realizar validaciones. También hay que agregar al template la parte de captura de errores después del

```
        {% csrf_token %}
        <input type="hidden" name="action" value="{{ action }}">
        ## esto permite pasar la acción que se está realizando en la manera de un botón oculto
        {% if form.errors %}
```

```
            <div class="alert alert-danger alert-dismissible">
              <button type="button" class="close" data-dismiss="alert" aria-hidden="true">×</button>
              <h5><i class="icon fas fa-ban"></i> Ha ocurrido un error al querer guardar el registro</h5>
              <ul>
                {% for field in form %}
                  {% for error in field.errors %}
                    <li>{{ error }}</li>
                  {% endfor %}
                {% endfor %}
              </ul>
            </div>
          {% endif %}
```

En el template en la parte de script se agrega:

```
<script>
    {% if form.errors %}
      var errors = '';
      {% for field in form %}
        {% for error in field.errors %}
          errors += '{{ error }}\n';
        {% endfor %}
      {% endfor %}
      Swal.fire({
        title: 'Error!',
        text: errors,
        icon: 'error'
      });
    {% endif %}
  </script>
```

Descargar la librería sweetalert2, agregar a la carpeta /static/lib/ y agregar al home.html.

Podemos sobreescribir el método save() de cada vista para hacer los controles de errores mediante lo siguiente:

```
  def save(self, commit=True):
    data = {}
    form = super()
    try:
      if form.is_valid():
        form.save()
      else:
        data['error'] = form.errors
    except Exception as e:
      data['error'] = str(e)
    return data
```

<u>Validacion de sesión iniciada:</u>

Se puede validar con el @method_decorator(login_required) pero conviene utilizar el mixins porque utilizaremos para varias validaciones. Por ej: LoginRequiredMixin, ValidatePermissionRequiredMixin
Se agrega el LOGIN_URL al settings.py= LOGIN_URL = '/login/'
Tambien se debe agregar la línea al template de login antes del {% csrf_token %} para usar el NEXT en el redirect:
<input type="hidden" name="next" value="{{ next }}">

Templates

En la raíz del proyecto crear una carpeta “templates”. Luego modificar el archivo settings, en el apartado TEMPLATES la línea ‘DIRS’ colocar:
[os.path.join(BASE_DIR,’templates’)],
def myfirstview(request):
    return render(request, ‘index.html’, data)
En el html se puede pasar las variables para ser visualizadas. Por ej: {{ name }}
Ademas podemos enviar objetos al template, mediante la declaración en la vista de, por ej:

‘categorias’: Category.objects.all()

Para iterar el objeto en el template:

```
<ul>
{% for category in categories %}
    <li>
        {{ category.name }}
    </li>
{% end for %}
</u>
```

Django permite la herencia de templates mediante la creación de bloques dentro del template, por ej con:

{% extends ‘index.html’ %}
Dentro de la plantilla padre debe existir bloques, que se van a cambiar con la herencia. El mismo bloque debe colocarse en el template que se crea a partir de la herencia.
{% block content %}
{% endblock%}
Además, se puede direccionar a distintos templates invocando al nombre del template. Por Ej:

Para incorporar archivos estáticos, en el template debe estar la línea correspondiente.
{% load static %}

Archivos Estáticos

Crear una carpeta en la raíz del proyecto llamada static
En el SETTINGS hay que incorporar unas líneas
STATIC_URL = ‘/static/‘
STATICFILES_DIRS = [
    [os.path.join(BASE_DIR,’static’)],
]
Dentro de la carpeta static van a ir todas las librerías, css, js, imágenes, etc que se utilizaran en el proyecto.
En cada template tiene que estar:
{% load static %}
Dentro de static crear carpetas lib, js, css, img
En lib se copia las carpetas de bootstrap, adminlte3
De adminlte3 se copia el contenido de la carpeta “dist” y de la carpeta “plugins”
Luego seleccionar las páginas que se van a utilizar, dentro de la carpeta pages. sidebar, top-nav, etc
Copiar el contenido de la página en el template “index.html”

EN EL VIDEO 18 esta como reemplazar las direcciones hacia los archivos estáticos

Conviene separar los templates, para que sea más fácil el mantenimiento por cada template base.
Para separar los templates dentro del template que se requiere en la posición establecida colocar:
{% include header %}
header.html contiene NAVBAR

Ajax

Permite hacer una petición al servidor para actualizar una pequeña parte de la página. Y no toda la página.

Se agrega la funcionalidad a un evento dentro de la página, un click en un botón por ej. Y se agrega dentro del bloque de javascript el bloque Ajax. Explicacion en Video 25 y Video 29

```
$('form').on('submit', function (e) {
        e.preventDefault();
        var parameters = $(this).serializeArray();
        $.ajax({
           url: window.location.pathname,
           type: 'POST',
           data: parameters,
           dataType: 'json'
        }).done(function (data) {
           console.log(data);
           if (!data.hasOwnProperty('error')) {
              location.href = '{{ list_url }}';
              return false;
           }
           message_error(data.error);
        }).fail(function (jqXHR, textStatus, errorThrown) {
           alert(textStatus + ': ' + errorThrown);
        }).always(function (data) {

        });
    });
```

TemplateView

Se crea una nueva aplicación que se llama homepage, para cargar el template por defecto al ingresar a la página.

Se crea la carpeta templates dentro de la aplicación, se crea un archivo index.html

Se utiliza el archivo views.py para esta aplicación porque va a utilizar una sola vista.

```
class IndexView(TemplateView):
   template_name = 'index.html'
```

En el archivo general de urls.py agregar:

```
path('', IndexView.as_view(), name='index'),
```

Para buscar templates se puede utilizar la página: bootstrapmade.com

Jquery-Confirm

Video 41

Esta librería nos sirve para generar un mensaje emergente que podemos utilizar para las confirmaciones o preguntas.
Descargar los archivos necesarios y pegarlos en la carpeta “static/lib/jquery-confirm”.
Los archivos son: jquery-confirm.min.css y jquery-confirm.min.js
En el template home.html se incorpora:

```
   <!-- Jquery Confirm -->
   <link rel="stylesheet" href="{% static 'lib/jquery-confirm-v3.3.4/jquery-confirm.min.css' %}">
   <script src="{% static 'lib/jquery-confirm-v3.3.4/jquery-confirm.min.js' %}"></script>
```

Dentro de /static/js/functions.js agregar la siguiente función:

```
function alert_action(title, content, callback, cancel) {
   $.confirm({
      theme: 'material',
      title: title,
      icon: 'fa fa-info',
```

```
        content: content,   # por ej: ¿Estás seguro de realizar la siguiente acción?
        columnClass: 'small',
        typeAnimated: true,
        cancelButtonClass: 'btn-primary',
        draggable: true,
        dragWindowBorder: false,
        buttons: {
            info: {
                text: "Si",
                btnClass: 'btn-primary',
                action: function () {
                    callback();    # probar con callback(data); # location.href = '{{ list_url }}';
                }
            },
            danger: {
                text: "No",
                btnClass: 'btn-red',
                action: function () {
                    cancel();
                }
            },
        }
    })
}

function submit_with_ajax(url, title, content, parameters, callback) {
    $.confirm({
        theme: 'material',
        title: title,
        icon: 'fa fa-info',
        content: content,
        columnClass: 'small',
        typeAnimated: true,
        cancelButtonClass: 'btn-primary',
        draggable: true,
        dragWindowBorder: false,
        buttons: {
            info: {
                text: "Si",
                btnClass: 'btn-primary',
                action: function () {
                    $.ajax({
                        url: url, //window.location.pathname
                        type: 'POST',
                        data: parameters,
                        dataType: 'json',
                        processData: false,
                        contentType: false,
                    }).done(function (data) {
                        console.log(data);
                        if (!data.hasOwnProperty('error')) {
                            callback(data);
                            return false;
                        }
                        message_error(data.error);
                    }).fail(function (jqXHR, textStatus, errorThrown) {
                        alert(textStatus + ': ' + errorThrown);
```

```
            }).always(function (data) {

            });
          }
        },
        danger: {
          text: "No",
          btnClass: 'btn-red',
          action: function () {

          }
        },
      }
    })
}
```

Para poder utilizar Ajax en el envío de FILES debemos modificar la función con lo siguiente:
En el archivo /static/js/function.js en la función submit_with_ajax agregar:

Al final de $.ajax({ ……

processData: false,
contentType: false,

En el form.html general modificar la función por:

```
$('form').on('submit', function (e) {
        e.preventDefault();
        var parameters = new FormData(this);
        submit_with_ajax(window.location.pathname, 'Notificación', '¿Estás seguro de realizar la siguiente acción?',
parameters, function () {
          location.href = '{{ list_url }}';
        });
      });
```

Media Files

Para poder utilizar, se debe incorporar dos líneas al settings.py
MEDIA_ROOT = os.path.join(BASE_DIR, 'media/')
MEDIA_URL = '/media/'

En modo desarrollo hay que agregar dos líneas al archivo urls.py principal

from django.conf import settings
from django.conf.urls.static import static
Luego agregar debajo de las urls:
urlpatterns += static(settings.MEDIA_URL, document_root=settings.MEDIA_ROOT)

En el form.html se debe modificar la línea:

```
{% block content %}
  <form method="post" action="." enctype="multipart/form-data"> # por el tipo de encriptación multipart para que se
pueda leer el tipo de datos FILES.
```

Para imágenes el campo debería ser de la siguiente manera y se puede colocar para que vaya organizando los archivos de la sgte manera: 'product/%Y/%m/%d' por ej.
image = models.ImageField(upload_to='product/%Y/%m/%d', null=True, blank=True, verbose_name='Imagen')

En el list.js correspondiente al modelo, hay que configurar de la siguiente manera:

```
columnDefs: [
        {
          targets: [-4],    #donde esta es la posición empezando desde la ultima
          class: 'text-center',
          orderable: false,
          render: function (data, type, row) {
```

```
            return '<img src="'+data+'" class="img-fluid d-block mx-auto" style="width: 20px; height: 20px;">';
          }
        },
```

En el models.py:

```
  def toJSON(self):
    item = model_to_dict(self)
    item['full_name'] = '{} / {}'.format(self.name, self.cat.name)
    item['cat'] = self.cat.toJSON()
    item['image'] = self.get_image()
    item['pvp'] = format(self.pvp, '.2f')
    return item

  def get_image(self):   # si tiene imagen devuelve la imagen, sino se le asigna una por defecto
    if self.image:
      return '{}{}'.format(MEDIA_URL, self.image)
    return '{}{}'.format(STATIC_URL, 'img/empty.png')  #buscar la imagen y guardarla en la carpeta /static/img/
```

Select2

Tenemos que instalar este plugin desde select2.org y copiarlo a la carpeta /static/lib/ e importarlo al html donde vamos a utilizar el select2.

```
{% block head %}
  {% block head_form %}
    <link href="{% static 'lib/select2-4.0.13/css/select2.min.css' %}" rel="stylesheet"/>
    <link href="{% static 'lib/select2-4.0.13/css/select2-bootstrap4.min.css' %}" rel="stylesheet"/>
    <script src="{% static 'lib/select2-4.0.13/js/select2.min.js' %}"></script>
    <script src="{% static 'lib/select2-4.0.13/js/i18n/es.js' %}"></script>
  {% endblock %}
{% endblock %}
```

Select anidados

Dentro de la app que necesitemos hacer un select anidado, utilizando el plugin **select2** (por ej localidades, para seleccionar un país y que se carguen las provincias que pertenecen a dicho pais) debemos crear en la carpeta “static/js/” de la aplicación un form.js con lo siguiente:

```
$(function () {
  $('.select2').select2({            #Se inicializa como va a funcionar el select2
    theme: "bootstrap4",
    language: 'es'
  })
  var select_provincias = $('select[name="provincia"]');          #creo la variable y asigno al componente del formulario.
  $('select[name="pais"]').on('change', function () {
    var id = $(this).val();                    #recupero el ID del país seleccionado
    var options = '<option value="">---------</option>';          #se crea la primera opción que es la vacia.
    if (id === '') {
      select_provincias.html(options);
      return false;
    }
    $.ajax({
      url: window.location.pathname,
      type: 'POST',
      data: {
        'action': 'search_provincia_id',
        'id': id
```

```
        },
        dataType: 'json',

    }).done(function (data) {
        if (!data.hasOwnProperty('error')) {
            select_provincias.html('').select2({       #se limpia y se inicializa para que al cambiar de país no quede provincia
                theme: "bootstrap4",
                language: 'es',
                data: data                #aca se van a cargar los datos del select anidado
            });
            return false;
        }
        message_error(data.error);
    }).fail(function (jqXHR, textStatus, errorThrown) {
    }).always(function (data) {
         #esto debe ir vacio
    });
  });
});
```

Dentro del archivo views.py de la aplicación, en el post, se edita para que quede de la siguiente manera:

```
def post(self, request, *args, **kwargs):
    data = {}
    try:
        action = request.POST['action']
        if action == 'search_provincia_id':
            data = [{'id': '', 'text': '---------'}]             #inicializamos la data con el primer campo vacio
            for i in Provincias.objects.filter(pais_id=request.POST['id']):
                data.append({'id': i.id, 'text': i.nombre})        #se utiliza text al pasar como array y no el nombre del campo.
        elif action == 'add', 'edit', etc…
```

En el html se agrega la importación del form.js en el block head_form debajo de la librería select2

```
{% block head %}
  {% block head_form %}
    <link href="{% static 'lib/select2-4.0.13/css/select2.min.css' %}" rel="stylesheet"/>
    <link href="{% static 'lib/select2-4.0.13/css/select2-bootstrap4.min.css' %}" rel="stylesheet"/>
    <script src="{% static 'lib/select2-4.0.13/js/select2.min.js' %}"></script>
    <script src="{% static 'lib/select2-4.0.13/js/i18n/es.js' %}"></script>
    <script src="{% static 'localidades/js/form.js' %}"></script>
  {% endblock %}
{% endblock %}
```

En el forms.py se aumenta el widgets del componente en la class Meta. Para poder utilizar el Select2

```
class Meta:
    model = Localidades
    fields = '__all__'
    widgets = {
        'pais': Select(attrs={
            'class': 'form-control select2',
            'style': 'width: 100%'
        }),
        'provincia': Select(attrs={
            'class': 'form-control select2',
            'style': 'width: 100%'
        }),
    }
```

Modal

Videos 48, 49, 50

Son ventanas emergentes de Bootstrap que permiteb poder acceder a un contenido desde una ventana emergente, sin tener que ir a otra pagina. Al ser de este tipo se tiene que utilizar Ajax para poder hacer el envio de los datos, porque no es una vista.

Para utilizar esto tenemos que pegar el contenido del modal en el bloque javascript de la página. También se tiene que pasar el formulario en el get_context_data de la vista como:

```
get_context_data(self,**kwargs):
        context[‘form’] = ClientForm()
```

En la vista el método post también hay que modificar para poder pasar los parámetros necesarios:

```
def post(self, request, *args, **kwargs):
    data = {}
    try:
        action = request.POST['action']
        if action == 'searchdata':
            data = []
            for i in Client.objects.all():
                data.append(i.toJSON())
        elif action == 'add':
            cli = Client()
            cli.names = request.POST['names']
            cli.surnames = request.POST['surnames']
            cli.dni = request.POST['dni']
            cli.date_birthday = request.POST['date_birthday']
            cli.address = request.POST['address']
            cli.gender = request.POST['gender']
            cli.save()
        elif action == 'edit':
            cli = Client.objects.get(pk=request.POST['id'])
            cli.names = request.POST['names']
            cli.surnames = request.POST['surnames']
            cli.dni = request.POST['dni']
            cli.date_birthday = request.POST['date_birthday']
            cli.address = request.POST['address']
            cli.gender = request.POST['gender']
            cli.save()
        elif action == 'delete':
            cli = Client.objects.get(pk=request.POST['id'])
            cli.delete()
        else:
            data['error'] = 'Ha ocurrido un error'
    except Exception as e:
        data['error'] = str(e)
    return JsonResponse(data, safe=False)
```

Se crea el componente <form> dentro del código del Modal. También debe importarse al template la librería {% load widget_tweaks %}

```
{% block javascript %}
  <!-- Modal -->
  <div class="modal fade" id="myModalClient" tabindex="-1" role="dialog" aria-labelledby="exampleModalLabel"
    aria-hidden="true">
    <form method="post" action="." enctype="multipart/form-data">
      <div class="modal-dialog" role="document">
        <div class="modal-content">
          <div class="modal-header">
```

```
            <h5 class="modal-title">
              <b><i class="fas fa-search"></i> <span></span></b>
            </h5>
            <button type="button" class="close" data-dismiss="modal" aria-label="Close">
              <span aria-hidden="true">&times;</span>
            </button>
          </div>
          <div class="modal-body">                    #Se colocan los componentes del FORMULARIO
            <div class="container-fluid">
              <input name="action" type="hidden" value="add">    #Se incorpora este componente oculto para la accion
              <input name="id" id="id" type="hidden" value="0">   #Se incorpora este componente oculto para el ID
              {% for field in form.visible_fields %}
                <div class="form-group">
                  <label for="email">{{ field.label }}:</label>
                  {{ field|add_class:'form-control'|attr:'autocomplete:off' }}
                </div>
              {% endfor %}
            </div>
          </div>
          <div class="modal-footer">
            <button type="submit" class="btn btn-primary btn-flat btn-block"><i class="fas fa-save"></i> Guardar</button>
          </div>
        </div>
      </div>
    </form>
  </div>
{% endblock %}
```

En el archivo js de la aplicación /static/js/list.js se deben agregar las funciones de modal y del submit para poder capturar los componentes y guardar, actualizar, eliminar, etc.

```
var tblClient;
var modal_title;

$(function () {
  modal_title = $('.modal-title');
  getData();
  $('.btnAdd').on('click', function () {
    $('input[name="action"]').val('add');
    modal_title.find('span').html('Crear un cliente');
    modal_title.find('i').removeClass().addClass('fas fa-plus');
    $('form')[0].reset();              #Esto permite limpiar el modal al iniciar
    $('#myModalClient').modal('show');
  });

  $('#data tbody')
    .on('click', 'a[rel="edit"]', function () {
      modal_title.find('span').html(Editar un cliente');
      modal_title.find('i').removeClass().addClass('fas fa-edit');
        #ESTO ES NECESARIO POR EL RESPONSIVE, solo con el Var data no funciona si se reacomoda la tabla.
      var tr = tblClient.cell($(this).closest('td, li')).index();     #con esta línea se obtiene el index de la fila del datatable
      var data = tblClient.row(tr.row).data();                        #con esta línea se obtiene el registro en la fila del datatable
      $('input[name="action"]').val('edit');                          #cambiamos la acción a edit para poder actualizar
      $('input[name="id"]').val(data.id);                             #Cargamos el valor del ID del registro
      $('input[name="names"]').val(data.names);                       #Cargamos los valores para los campos del modal
      $('input[name="surnames"]').val(data.surnames);
      $('input[name="dni"]').val(data.dni);
      $('input[name="date_birthday"]').val(data.date_birthday);
      $('input[name="address"]').val(data.address);
      $('select[name="gender"]').val(data.gender.id);
      $('#myModalClient').modal('show');                              #Hacemos visible el modal
    })
    .on('click', 'a[rel="delete"]', function () {
```

```
        var tr = tblClient.cell($(this).closest('td, li')).index();
        var data = tblClient.row(tr.row).data();
        var parameters = new FormData();
        parameters.append('action', 'delete');
        parameters.append('id', data.id);
        submit_with_ajax(window.location.pathname, 'Notificación', '¿Estás seguro de eliminar el siguiente registro?', parameters,
function () {
            tblClient.ajax.reload();
        });
    });
```

##Esto es otra manera de implementar que se limpie el modal al inicio

```
  $('#myModalClient').on('shown.bs.modal', function () {
    $('form')[0].reset();
  });

$('form').on('submit', function (e) {
    e.preventDefault();
    var parameters = new FormData(this);
    submit_with_ajax(window.location.pathname, 'Notificación', '¿Estás seguro de realizar la siguiente acción?', parameters,
function () {
        $('#myModalClient').modal('hide');   #oculta el modal una vez realizada la operacion
        tblClient.ajax.reload();             #esto permite recargar por Ajax el Datatable.
    });
  });
```

Para separar de otros templates de listados, se puede poner dentro de un bloque los botones inferiores, de manera de que si una aplicación se implementa con modal, puede sobreescribir la acción de los botones en esa determinada lista.

Esto sería un bloque editado para que funcione con MODAL:

```
{% block buttons_list %}
  <a class="btn btn-primary btn-flat btnAdd" style="color: white;">    #btnAdd se utiliza para utilizar desde la list.js para invocar
    <i class="fas fa-plus"></i> Nuevo registro
  </a>
  <a href="{{ list_url }}" class="btn btn-success btn-flat">
    <i class="fas fa-sync"></i> Actualizar
  </a>
{% endblock %}
```

En el list.js del datatables, se tiene que agregar una Referencia Relativa, o agregar una clase falsa a los botones para poder identificarlos y realizar acciones distintas donde rel=”” es la referencia y btnEdit es la clase falsa, esto se referencia desde el js.

```
        var buttons = '<a href="#" rel="edit" class="btn btn-warning btn-xs btn-flat btnEdit"><i class="fas fa-edit"></i></a> ';
          buttons += '<a href="#" rel="delete" class="btn btn-danger btn-xs btn-flat btnDelete"><i class="fas fa-trash-
alt"></i></a>';
```

en el js seria:

```
            .on('click', 'a[rel="edit"]', function () {                    #para la referencia relativa
            .on('click', '.btnEdit, function () {                          #para la clase falsa
```

Autocomplete

Autocomplete con Ajax

VER SI CON EL EVENTO .on(‘change keyup’) se puede usar que ya autocomplete todo y no haya que seleccionar!

Para esto se utiliza la librería jQuery-ui. Hay que descargar y copiar en /static/lib/
Se incorpora al template donde se va a utilizar el autocomplete en el block head, reemplazar por la versión que se tenga.

```
{% block head %}
  <link href="{% static 'lib/jquery-ui-1.12.1/jquery-ui.min.css' %}" rel="stylesheet"/>
  <script src="{% static 'lib/jquery-ui-1.12.1/jquery-ui.min.js' %}"></script>
```

En el template se agrega el componente que vamos a utilizar para hacer el autocomplete.

```
$('input[name="search"]').autocomplete({            #Elemento del formulario sobre el cual se va a realizar el autocomplete
```

```
        source: function (request, response) {        #mandamos una función para utilizar Ajax contra el modelo de la BD
            $.ajax({
                url: window.location.pathname,        #Esta url es la URL absoluta
                type: 'POST',
                data: {
                    'action': 'autocomplete',         #La acción para detectar en el POST de la vista
                    'term': request.term              #para obtener lo que se va escribiendo
                },
                dataType: 'json',
            }).done(function (data) {
                response(data);                       #retorna la respuesta con la información por el JSON
            }).fail(function (jqXHR, textStatus, errorThrown) {
                //alert(textStatus + ': ' + errorThrown);
            }).always(function (data) {

            });
        },
        delay: 500,         #Tiempo de demora para que tarde en aparecer las opciones
        minLength: 1,       #Caracteres a partir del cual empieza a realizar el autocomplete
        select: function (event, ui) {
                //console.log(ui.item);               #este evento se utiliza para recuperar el objeto cuando hacemos la selección
        }
    });
```

En la vista en el método POST tenemos que agregar lo siguiente para poder capturar lo que viene por el evento:

```
elif action == 'autocomplete':
        data = []
        #sobre el modelo que se utilice, utilizando icontains para que tome mayúscula y minúscula
        for i in Category.objects.filter(name__icontains=request.POST['term'])[0:10]:   #Se agrega el 0:10 para obtener 10 registros
            item = i.toJSON()
        #se aumenta el diccionario agregando el ítem value para que tome lo del autocomplete, por el valor que recupera
            item['value'] = i.name
            data.append(item)
```

Autocomplete con Select2

Se utiliza con un componente de tipo SELECT y se realiza el autocomplete con Ajax.
Se inicializa el componente en el template del tipo SELECT

```
$('select[name="search"]').select2({
        theme: "bootstrap4",
        language: 'es',
        allowClear: true,         #permite borrar la selección anterior, aparece el icono X a la derecha
        ajax: {
            delay: 250,
            type: 'POST',         #se especifica el tipo POST porque por defecto es el GET
            url: window.location.pathname,
            data: function (params) {
                var queryParameters = {
                    term: params.term,              #para enviar a la vista
                    action: 'autocomplete'          #para enviar a la vista
                }
                return queryParameters;
            },
            processResults: function (data) {     #aca llega el array con la busqueda
                return {
                    results: data
                };
            },
        },
        placeholder: 'Ingrese una descripción',
        minimumInputLength: 1,        #minimo de caracteres a ingresar para buscar
```

```
        });
```

En la vista se debe generar en el método POST la acción de autocomplete,

```
elif action == 'autocomplete':
            data = []
            for i in Category.objects.filter(name__icontains=request.POST['term'])[0:10]:
                item = i.toJSON()
            #a diferencia del autocomplete con Ajax, no se pasa “name” sino “text”, porque con name no funciona.
                item['text'] = i.name
                data.append(item)
```

Usuarios en Django

Se reemplaza la clase usuario por defecto que trae Django. Para ello creamos una aplicación “usuarios” que es la que se va a encargar de reemplazar la clase de usuario actual, heredando de la clase AbstractUser que contiene todos los campos que tiene la tabla auth_user en la BD.
Creamos una clase y aumentamos con los campos que necesitamos.

```
#   MODELO

class Usuarios(AbstractUser):
    legajo = models.CharField(max_length=10, null=True, blank=True, verbose_name='Legajo')
    dni = models.PositiveIntegerField(verbose_name='DNI', null=True, blank=True)
    telefono = models.TextField(null=True, blank=True)
    image = models.ImageField(upload_to='users/%Y/%m/%d', null=True, blank=True)

    def get_image(self):                        ///para devolver correctamente la imagen
        if self.image:
            return '{}{}'.format(MEDIA_URL, self.image)
        return '{}{}'.format(STATIC_URL, 'img/empty.png')

    def toJSON(self):                   //devuelve un diccionario con los campos que contiene el MODELO
        item = model_to_dict(self, exclude=['password','groups', 'user_permissions', 'last_login'])
//el model_to_dict permite obtener un diccionario con los campos del modelo, pero con limitantes en cuanto a fechas
//relaciones entre tablas, imagenes y demás. Por ello se aplica el EXCLUDE
        item['full_name'] = self.get_full_name()      ///para obtener el nombre y apellido
//para obtener los grupos que pertenece se itera todos los grupos del usuario y se guarda en un diccionario id y nombre
        item['groups'] = [{'id': g.id, 'name': g.name} for g in self.groups.all()]
        if self.last_login:
            item['last_login'] = self.last_login.strftime('%d-%m-%Y')  //si se logueo alguna vez con tipo de fecha seteada
        item['date_joined'] = self.date_joined.strftime('%d-%m-%Y')          //fecha de creación con tipo de fecha seteada
        item['image'] = self.get_image()                                     //devolver correctamente la imagen
        return item
        return ítem
```

Luego hay que sustituir en el Settings.py por la nueva clase que administra los usuarios:

AUTH_USER_MODEL = 'usuarios.Usuarios'   donde usuarios es la clase Usuarios es el modelo

También, esta aplicación debe cargarse en el settings en la parte de INSTALLED_APPS.

Hacer migrate y makemigrations

Se puede al tener la imagen de usuario agregar para que se visualice en el template con el método definido en la clase usuario get_image. Reemplazar en el HTML {{ request.user.get_image }}

Listar Usuarios

Video 77

```
##VIEWS.PY
class UserListView(LoginRequiredMixin, ValidatePermissionRequiredMixin, ListView):
    model = User
    template_name = 'user/list.html'
    permission_required = 'user.view_user'
```

```
///VER EN LA BD en la tabla auth_permission cual es el permiso necesario combinación TABLA Y PERMISO
//Desactivamos el mecanismo de defensa cuando se SOBREESCRIBE EL METODO POST con el decorador
  @method_decorator(csrf_exempt)
  def dispatch(self, request, *args, **kwargs):  //sirve para direccionar el tipo de petición POST y GET
    return super().dispatch(request, *args, **kwargs)

  def post(self, request, *args, **kwargs):
    data = {}
    try:
      action = request.POST['action']
      if action == 'searchdata':
        data = []                    //enviamos un array de diccionarios con lo solicitado dentro del for
        for i in User.objects.all():
          data.append(i.toJSON())
      else:
        data['error'] = 'Ha ocurrido un error'
    except Exception as e:
      data['error'] = str(e)
    return JsonResponse(data, safe=False)

  def get_context_data(self, **kwargs):
    context = super().get_context_data(**kwargs)
    context['title'] = 'Listado de Usuarios'         //Titulo del datatable
    context['create_url'] = reverse_lazy('user:user_create')      //link del create
    context['list_url'] = reverse_lazy('user:user_list')          //link del listado
    context['entity'] = 'Usuarios'                                //entidad para visualizar en el template
    return context
```

Para poder listar los usuarios debemos crear un archivo list.html que va a heredar del list.html BASE y lo extendemos con lo siguiente:

```
{% extends 'list.html' %}                    //Extiende del LIST.HTML BASE
{% load static %}
{% block head_list %}
  <script src="{% static 'user/js/list.js' %}"></script>      ///DEBEMOS DIRECCIONAR AL JS CON LA FUNCIONALIDAD
{% endblock %}

{% block columns %}                          ///ESPECIFICAMOS LAS COLUMNAS CON SU RESPECTIVO ANCHO
  <tr>
    <th scope="col" style="width: 5%;">ID</th>
    <th scope="col" style="width: 15%;">Nombres</th>
    <th scope="col" style="width: 15%;">Apellidos</th>
    <th scope="col" style="width: 15%;">Username</th>
    <th scope="col" style="width: 20%;">Fecha de registro</th>
    <th scope="col" style="width: 10%;">Imagen</th>
    <th scope="col" style="width: 20%;">Opciones</th>
  </tr>
{% endblock %}

{% block rows %}

{% endblock %}

{% block javascript %}

{% endblock %}
```

Luego creamos el archivo list.js que va a contener la funcionalidad del listado a través de AJAX:

```
$(function () {
```

```
  $('#data').DataTable({
    responsive: true,
    autoWidth: false,
    destroy: true,
    deferRender: true,
    ajax: {
      url: window.location.pathname,
      type: 'POST',
      data: {
        'action': 'searchdata'
      },
      dataSrc: ""
    },
    columns: [                    ///LAS COLUMNAS DEBEN COINCIDIR CON LO DEL HTML
      {"data": "id"},
      {"data": "full_name"},      ///Nombre completo
      {"data": "username"},
      {"data": "date_joined"},    ///fecha de registro
      {"data": "image"},
      {"data": "id"},             ///ESTE CAMPO VA DUPLICADO PARA QUE SE PUEDA MOSTRAR BOTONES de OPCIONES
    ],
    columnDefs: [
      {
        targets: [-2],            ///PERSONALIZAMOS EL ESPACIO PARA LA IMAGEN DE USUARIO
        class: 'text-center',
        orderable: false,
        render: function (data, type, row) {      //se crea el componente con la ruta de la imagen “row.image”
          return '<img src="'+row.image+'" class="img-fluid mx-auto d-block" style="width: 20px; height: 20px;">';
        }
      },
      {
        targets: [-1],
        class: 'text-center',
        orderable: false,
        render: function (data, type, row) {
          var buttons = '<a href="/user/update/' + row.id + '/" class="btn btn-warning btn-xs btn-flat"><i class="fas fa-
edit"></i></a> ';
          buttons += '<a href="/user/delete/' + row.id + '/" type="button" class="btn btn-danger btn-xs btn-flat"><i
class="fas fa-trash-alt"></i></a>';
          return buttons;
        }
      },
    ],
    initComplete: function (settings, json) {
    }
  });
});
```

Luego agregar las url en el archivo urls de la aplicación y chequear que este también en el archivo de URLS principal

```
app_name = 'user'
urlpatterns = [
  # user
  path('list/', UserListView.as_view(), name='user_list'),
  path('add/', UserCreateView.as_view(), name='user_create'),
  path('update/<int:pk>/', UserUpdateView.as_view(), name='user_update'),
  path('delete/<int:pk>/', UserDeleteView.as_view(), name='user_delete'),
]
```

Create Usuarios

Video 78

##VIEWS.PY

```
class UserCreateView(LoginRequiredMixin, ValidatePermissionRequiredMixin, CreateView):
    model = User
    form_class = UserForm
    template_name = 'user/create.html'                  //Hereda del Template BASE Form.html
    success_url = reverse_lazy('user:user_list')        //url de éxito al finalizar la creacion
    permission_required = 'user.add_user'               //permiso necesario para estafuncionalidad
    url_redirect = success_url                          //url de éxito al finalizar la creacion
//en el dispatch no utilizamos el decorador para desactivar el mecanismo de seguridad porque ya se encuentra
//desactivado en el template.

    def dispatch(self, request, *args, **kwargs):
        return super().dispatch(request, *args, **kwargs)

    def post(self, request, *args, **kwargs):
        data = {}
        try:
            action = request.POST['action']
            if action == 'add':
                form = self.get_form()
                data = form.save()
            else:
                data['error'] = 'No ha ingresado a ninguna opción'
        except Exception as e:
            data['error'] = str(e)
        return JsonResponse(data)

    def get_context_data(self, **kwargs):
        context = super().get_context_data(**kwargs)
        context['title'] = 'Crear un Usuario'           //Titulo
        context['entity'] = 'Usuarios'                  //Entidad para visualizar en el template
        context['list_url'] = self.success_url          //se pasa el list.url de la aplicacion
        context['action'] = 'add'                       //el action para la VIEW
        return context
```

Para las vistas genéricas CREATE y UPDATE necesitamos crear un formulario, con lo siguiente:

```
class UserForm(ModelForm):
    def __init__(self, *args, **kwargs):
        super().__init__(*args, **kwargs)
        self.fields['first_name'].widget.attrs['autofocus'] = True          //seteamos el foco al campo nombres

    class Meta:    //En la clase META personalizamos los campos y seteamos los campos que vamos a traer, no el '__all__'
        model = User
        fields = 'first_name', 'last_name', 'email', 'username', 'password', 'image',
        widgets = {
            'first_name': TextInput(
                attrs={
                    'placeholder': 'Ingrese sus nombres',
                }
            ),
            'last_name': TextInput(
                attrs={
                    'placeholder': 'Ingrese sus apellidos',
                }
            ),
```

```
            'email': TextInput(
                attrs={
                    'placeholder': 'Ingrese su email',
                }
            ),
            'username': TextInput(
                attrs={
                    'placeholder': 'Ingrese su username',
                }
            ),
//NECESITAMOS EL RENDER VALUE PARA PODER VISUALIZAR CORRECTAMENTE la contraseña oculta, porque por
//defecto Django implementa de tipo TextInput el campo
            'password': PasswordInput(render_value=True,
                attrs={
                    'placeholder': 'Ingrese su password',
                }
            ),
        }
        'groups': SelectMultiple(attrs={
            'class': 'form-control select2',
            'style': 'width: 100%',
            'multiple': 'multiple'
        })
    }
    exclude = ['user_permissions', 'last_login', 'date_joined', 'is_superuser', 'is_active', 'is_staff']//excluimos los campos

//En el formulario sobreescribimos el método SAVE para poder encriptar la contraseña al momento de guardarla
    def save(self, commit=True):
        data = {}
        form = super()
        try:
            if form.is_valid():
//el método cleaned_data nos permite obtener el valor de un campo, debe ser llamado después del form.is_valid
                pwd = self.cleaned_data['password']              //pasamos  el password en una variable
                u = form.save(commit=False)             //pausamos el commit para que no guarde automaticamente
                if u.pk is None:                    //Controlamos si es un usuario nuevo
                    u.set_password(pwd)         //Si es nuevo encriptamos la contraseña
                else:
                    user = User.objects.get(pk=u.pk)      //creamos una variable para traer al usuario que se esta editando
                    if user.password != pwd:      //Comparamos si el usuario SETEO la contraseña al ser distinto el valor
                        u.set_password(pwd)       //actualizamos la contraseña al valor nuevo y la encriptamos
                u.save()                            //ahora si realizamos el guardado del formulario
            else:
                data['error'] = form.errors
        except Exception as e:
            data['error'] = str(e)
        return data
```

Update Usuarios

Video 79

```
##VIEWS.PY
class UserUpdateView(LoginRequiredMixin, ValidatePermissionRequiredMixin, UpdateView):
    model = User
    form_class = UserForm
    template_name = 'user/create.html'              //utilizamos el mismo template del create
    success_url = reverse_lazy('user:user_list')
    permission_required = 'user.change_user'      //asignamos el permiso necesario para usar la aplicacion
```

```
  url_redirect = success_url

  def dispatch(self, request, *args, **kwargs):
    self.object = self.get_object()                    //Al hacer un UPDATE tenemos que pasar el valor del objeto
    return super().dispatch(request, *args, **kwargs)

  def post(self, request, *args, **kwargs):
    data = {}
    try:
      action = request.POST['action']
      if action == 'edit':                             //la acción se modifica de ADD a EDIT
        form = self.get_form()
        data = form.save()
      else:
        data['error'] = 'No ha ingresado a ninguna opción'
    except Exception as e:
      data['error'] = str(e)
    return JsonResponse(data)

  def get_context_data(self, **kwargs):
    context = super().get_context_data(**kwargs)
    context['title'] = 'Actualizar un Usuario'
    context['entity'] = 'Usuarios'
    context['list_url'] = self.success_url
    context['action'] = 'edit'                         //la acción se modifica de ADD a EDIT
    return context
```

Delete Usuarios

Video 80

```
##VIEWS.PY
class UserDeleteView(LoginRequiredMixin, ValidatePermissionRequiredMixin, DeleteView):
  model = User
  template_name = 'user/delete.html'                   //Hereda del Template BASE delete.html
  success_url = reverse_lazy('user:user_list')
  permission_required = 'user.delete_user'             //permiso correspondiente
  url_redirect = success_url

  def dispatch(self, request, *args, **kwargs):
    self.object = self.get_object()                    //volvemos a asignar la instancia del objeto
    return super().dispatch(request, *args, **kwargs)

  def post(self, request, *args, **kwargs):
    data = {}
    try:
      self.object.delete()
    except Exception as e:
      data['error'] = str(e)
    return JsonResponse(data)

  def get_context_data(self, **kwargs):
    context = super().get_context_data(**kwargs)
    context['title'] = 'Eliminar Usuario'              //titulo
    context['entity'] = 'Usuarios'                     //entidad para visualizar en el template
    context['list_url'] = self.success_url             //url de redirección al list en caso exitoso
    return context
```

Grupos en Django

Los grupos en principio se crean desde el panel de administración de Django, luego, personalizamos el componente del campo grupos del usuario según SELECT2. En el método META del formulario:

```
'groups': SelectMultiple(attrs={            ////Permite la selección de varios grupos en el listado.
            'class': 'form-control select2',
            'style': 'width: 100%',
            'multiple': 'multiple'
        })
    }
```

Luego debemos agregar las librerías necesarias para utilizar el Select2 al template create.html:

```
{% extends 'form.html' %}
{% load static %}
{% block head_form %}
  <link href="{% static 'lib/select2-4.0.13/css/select2.min.css' %}" rel="stylesheet"/>
  <link href="{% static 'lib/select2-4.0.13/css/select2-bootstrap4.min.css' %}" rel="stylesheet"/>
  <script src="{% static 'lib/select2-4.0.13/js/select2.min.js' %}"></script>
  <script src="{% static 'lib/select2-4.0.13/js/i18n/es.js' %}"></script>
  <script src="{% static 'user/js/form.js' %}"></script>
{% endblock %}
```

Tambien se requiere un archivo form.js dentro de la carpeta de la aplicación para poder inicializar el componente de tipo SELECT2.

```
$(function () {
  $('.select2').select2({
    theme: "bootstrap4",
    language: 'es',
    placeholder: 'Buscar..'
  });
});
```

Por ultimo, debemos modificar el método save() del usuario, porque al haberlo sobreescrito para la funcionalidad de contraseña, se pierde la funcionalidad por defecto, por lo que tenemos que agregar para que recorra los grupos seleccionados y los vaya agregando.

```
def save(self, commit=True):
    data = {}
    form = super()
    try:
      if form.is_valid():
        pwd = self.cleaned_data['password']
        u = form.save(commit=False)
        if u.pk is None:
          u.set_password(pwd)
        else:
          user = User.objects.get(pk=u.pk)
          if user.password != pwd:
            u.set_password(pwd)
        u.save()
                ///Limpiamos los grupos del usuario, para que al actualizar tome los cambios
        u.groups.clear()
        for g in self.cleaned_data['groups']:     ///Recorremos los grupos
          u.groups.add(g)                         ///Agregamos los grupos seleccionados al usuario que se edita
      else:
        data['error'] = form.errors
    except Exception as e:
      data['error'] = str(e)
    return data
```

Luego agregamos el campo de Grupos al list.html:

```
<th scope="col" style="width: 15%;">Grupos</th>
```

Y Tambien al list.js:
{"data": "groups"},
Y en la parte de columnDefs:

```
{
        targets: [-2],              //posición empezando desde la ultima columna
        class: 'text-center',
        orderable: false,
        render: function (data, type, row) {
          var html = '';            //se crea la variable html
          $.each(row.groups, function (key, value) {    //se itera y se agrega a la variable cada grupo al que pertenece
            html += '<span class="badge badge-success">' + value.name + '</span> ';
          });                       //badge es una clase de text que devuelve en fondo verde con letras blancas
          return html; //retornamos los grupos
        }
      }
```

Para que esto se liste de la manera correcta también debemos tener en el modelo, en el método toJSON el diccionario correspondiente a los grupos del usuario. Agregamos una línea como la siguiente:

```
//para obtener los grupos que pertenece se itera todos los grupos del usuario y se guarda en un diccionario id y nombre
    item['groups'] = [{'id': g.id, 'name': g.name} for g in self.groups.all()]
```

Mixins

Video 53, 54 ,55

Son clases que permiten validar las vistas, esta validación se pasa como argumento en cada vista. Para ello se debe crear un archivo dentro de la carpeta PRINCIPAL de aplicaciones, llamado mixins.py. Con esto se logra hacer las validaciones de manera general y no en cada método dispatch de cada vista. Se pasa como argumento dentro de la declaración de la clase y tiene un orden de prioridad, es decir se debe pasar primero que el ListView, por ej.
class PaisesListView(LoginRequiredMixin, ValidatePermissionRequiredMixin, ListView):
Cabe mencionar que para que aplique estos permisos, el usuario no debe ser **SuperUsuario.**

La clase LoginRequiredMixin es una clase base que incorpora Django que la llamamos importándola, requiere más prioridad que la clase ValidatePermissionRequiredMixin, por lo que se declara antes. Con este mixins nos evitamos implementar el DECORATOR antes de cada método dispatch.

La clase ValidatePermissionRequiredMixin nos permite saber si el usuario tiene permisos necesarios para acceder a la vista. Estos permisos se generan en base al usuario, en la tabla user_permissions. También en los grupos, en donde podemos definir los grupos (perfiles) de usuarios que van a tener permisos sobre cada una de las entidades del sistema.

```
class ValidatePermissionRequiredMixin(object):
  permission_required = ''
  url_redirect = None

  def get_perms(self):  #Consulta si es un solo valor de permisos, caso contrario manda toda la tupla de permisos
    perms = []              ///SI LA VISTA SOLICITADA REQUIERE VARIOS PERMISOS, AL SER UN ARRAY VA A PERMITIR
    if isinstance(self.permission_required, str):
      perms.append(self.permission_required)          #convierte a una tupla de un solo valor y agrega al ARRAY
    else:
      perms = list(self.permission_required)    #envia todo el listado de permisos
    return perms

  def get_url_redirect(self):
    if self.url_redirect is None:
      return reverse_lazy('dashboard:home')           #se asigna la URL de retorno si no tiene permisos
    return self.url_redirect

  def dispatch(self, request, *args, **kwargs):
    if request.user.has_perms(self.get_perms()):        #si el usuario posee permisos accede a la vista
```

```
            return super().dispatch(request, *args, **kwargs)
        messages.error(request, 'No posee permisos suficientes')   #si el usuario no posee los permisos, devuelve el msj.
        return HttpResponseRedirect(self.get_url_redirect())
```

En cada vista que creemos en nuestro proyecto, se tiene que pasar los parámetros necesarios para que estas validaciones funciones y el atributo permission_required correspondiente.

```
class PaisesListView(LoginRequiredMixin, ValidatePermissionRequiredMixin, ListView):
    model = Paises
    template_name = 'paises/list.html'
    permission_required = 'geografico.view_paises'

class PaisesCreateView(LoginRequiredMixin, ValidatePermissionRequiredMixin, CreateView):
    model = Paises
    form_class = PaisesForm
    template_name = 'paises/create.html'
    success_url = reverse_lazy('geografico:paises_list')
    permission_required = 'geografico.add_paises'
    url_redirect = success_url
```

Para obtener una retroalimentación, se puede enviar los mensajes de falta de permisos suficientes, se puede utilizar la librería de Django **"messages"** esta se incorpora al método dispatch

```
            #messages.error(request, 'No posee permisos suficientes')
```

Esto tenemos que agregar al final del body.html para que pueda salir el mensaje mediante la librería **sweetalert**

```
<script>
        {% if messages %}
            var html = '<p>';
            {% for message in messages %}
                html += '{{ message }}<br>';
            {% endfor %}
            html += '</p>';
            Swal.fire({
                title: 'Error!',
                html: html,
                icon: 'error'
            });
        {% endif %}
    </script>
</body>
```

Roles y Permisos en el Dashboard

Para poder trabajar con Roles y Permisos podemos hacerlo a través de los grupos de usuarios, para ello debemos modificar el mixin ValidatePermissionRequiredMixin en el método dispatch.

Si el usuario va a tener diversos perfiles de grupos, tenemos que tener instalada la librería CRUM para usar el get_current_request y obtener la sesión actual del usuario.

Tambien, si se va

```
    def dispatch(self, request, *args, **kwargs):
        request = get_current_request()                 //con esto obtenemos la sesión actual del usuario
        if request.user.is_superuser:                   //preguntamos si el usuario es SUPERUSUARIO
            return super().dispatch(request, *args, **kwargs)           //SUPERUSUARIO NO REQUIERE PERMISOS, SALTEA
        if 'group' in request.session:                  //Si tiene grupos, sino no podrá tener acceso a nada mas que el home
            group = request.session['group']            //creamos la variable para el grupo de la sesión actual
            perms = self.get_perms()
//PARA PODER VALIDAR DOS O MAS PERMISOS, NECESITO HACER UN FOR Y UN IF CONTRARIO DENTRO, DE MODO QUE
//SI EL USUARIO NO TIENE ALGUNO DE LOS PERMISOS REQUERIDOS PARA ACCEDER A LA FUNCION, ENVIE UN MENSAJE
//DE ERROR Y CORTE LA EJECUCION DEL METODO.
```

```
            for p in perms:
                if not group.permissions.filter(codename=p).exists():
                    messages.error(request, 'No tiene permiso para ingresar a este módulo')
                    return HttpResponseRedirect(self.get_url_redirect())
            return super().dispatch(request, *args, **kwargs)
//EL RETURN DEBAJO ESTA FUERA DEL IF DE GRUPOS, SI NO TIENE ASIGNADO EL USUARIO ALGUN GRUPO NO VA A
//PERMITIR TAMPOCO ACCEDER A LA VISTA SOLICITADA
        messages.error(request, 'No tiene permiso para ingresar a este módulo')
        return HttpResponseRedirect(self.get_url_redirect())
```

Editar perfil del usuario actual

Para poder hacer esto hay que cambiar algunas cuestiones con respecto al template, porque únicamente se va a poder realizar cambios en el perfil de un usuario final del sistema. Por ejemplo, que no se visualice en la url el ID del usuario. Para ello primero vamos al formulario dentro de la aplicación de usuarios y creamos la clase UserProfileForm con lo siguiente:

```
class UserProfileForm(ModelForm):
    def __init__(self, *args, **kwargs):
        super().__init__(*args, **kwargs)
        self.fields['first_name'].widget.attrs['autofocus'] = True

    class Meta:
        model = User
        fields = 'first_name', 'last_name', 'email', 'username', 'password', 'image'
        widgets = {
            'first_name': forms.TextInput(
                attrs={
                    'placeholder': 'Ingrese sus nombres',
                }
            ),
            'last_name': forms.TextInput(
                attrs={
                    'placeholder': 'Ingrese sus apellidos',
                }
            ),
            'email': forms.TextInput(
                attrs={
                    'placeholder': 'Ingrese su correo electrónico',
                }
            ),
            'username': forms.TextInput(
                attrs={
                    'placeholder': 'Ingrese su nombre de usuario',
                }
            ),
            'password': forms.PasswordInput(render_value=True,
                attrs={
                    'placeholder': 'Ingrese su password',
                }
            ),
        }
        exclude = ['user_permissions', 'last_login', 'date_joined', 'is_superuser', 'is_active', 'is_staff', 'groups']
###Se excluyen los campos de grupos y demás para que quede únicamente lo básico que un usuario podría modificar

    def save(self, commit=True):
        data = {}
        form = super()
        try:
```

```
            if form.is_valid():
                pwd = self.cleaned_data['password']
                u = form.save(commit=False)
                if u.pk is None:
                    u.set_password(pwd)
                else:
                    user = User.objects.get(pk=u.pk)
                    if user.password != pwd:
                        u.set_password(pwd)
                u.save()
            else:
                data['error'] = form.errors
        except Exception as e:
            data['error'] = str(e)
        return data
```

En el método save se quita lo que tiene que ver con grupos porque no corresponde al usuario final hacer esos cambios.

Luego debemos crear una vista con lo siguiente:

En esta vista se quita el mixin de ValidatePermissionRequiredMixin porque únicamente va a trabajar con los datos propios y no de los otros usuarios.

```
class UserProfileView(LoginRequiredMixin, UpdateView):
    model = User
    form_class = UserProfileForm                    ###Se direcciona al formulario que creamos anteriormente
    template_name = 'user/profile.html'             ###Se direcciona a la URL que se crea para el caso
    success_url = reverse_lazy('erp:dashboard')     ###Direccionamos al home y no al user_list

    @method_decorator(csrf_exempt)
    def dispatch(self, request, *args, **kwargs):
        self.object = self.get_object()
        return super().dispatch(request, *args, **kwargs)

    def get_object(self, queryset=None):            ##DEBEMOS SOBREESCRIBIR EL METODO PARA PODER ENVIAR EL ID DEL
        return self.request.user                    ##USUARIO ACTUAL LOGUEADO, PARA QUE NO SE VISUALICE EN URL.

    def post(self, request, *args, **kwargs):
        data = {}
        try:
            action = request.POST['action']
            if action == 'edit':
                form = self.get_form()
                data = form.save()
            else:
                data['error'] = 'No ha ingresado a ninguna opción'
        except Exception as e:
            data['error'] = str(e)
        return JsonResponse(data)

    def get_context_data(self, **kwargs):
        context = super().get_context_data(**kwargs)
        context['title'] = 'Editar Perfil'
        context['entity'] = 'Perfil'
        context['list_url'] = self.success_url
        context['action'] = 'edit'
        return context
```

Creamos el template para la edición del usuario: profile.html en la carpeta de la aplicación que extienda del template base form.html: {% extends 'form.html' %}

Creamos el path en el archivo urls de la aplicación: path('profile/', UserProfileView.as_view(), name='user_profile'),

Una vez creado todo hay que llamar desde el template base del header, según corresponda:

```
<li class="nav-item dropdown">
        <a class="nav-link" data-toggle="dropdown" href="#">
          <i class="fas fa-users-cog"></i>
        </a>
        <div class="dropdown-menu dropdown-menu-lg dropdown-menu-right">
            <span class="dropdown-header" style="font-size: 12px;">
              Sú último acceso fue {{ request.user.last_login }}              //datos de ultima vez conectado
            </span>
          <div class="dropdown-divider"></div>
          <a href="{% url 'user:user_profile' %}" class="dropdown-item">       //ACA AGREGAMOS la url para editar perfil
            <i class="fas fa-edit mr-2"></i> Editar perfil
          </a>
          <div class="dropdown-divider"></div>            //Luego usaremos para SETEAR la contraseña del usuario
          <a href="{% url 'user:user_change_password' %}" class="dropdown-item">
            <i class="fas fa-lock mr-2"></i> Editar password
          </a>
        </div>
      </li>
```

Cambiar contraseña del Usuario Actual

Para poder cambiar la contraseña de un usuario, debemos crear una vista con lo siguiente:

```
###Para esta vista vamos a utilizar el FormView y no el UpdateView
class UserChangePasswordView(LoginRequiredMixin, FormView):
  model = User
  form_class = PasswordChangeForm              ###Utilizamos este FORM BASE de Django para editar la contraseña
  template_name = 'user/change_password.html'       ##creamos el template para poder utilizarlo
  success_url = reverse_lazy('login')
##Una vez cambiada la contraseña se cierra la sesión para que se loguee de nuevo con la nueva contraseña
##dentro del dispatch no se utiliza el get_object porque no se utiliza el UpdateView y no se requiere.
  @method_decorator(csrf_exempt)
  def dispatch(self, request, *args, **kwargs):
    return super().dispatch(request, *args, **kwargs)

##sobreescribimos el método get_form
  def get_form(self, form_class=None):
    form = PasswordChangeForm(user=self.request.user)       #la variable form se va a inicializar con el usuario actual
##Se agregan los widgets para mantener la línea con respecto a las otras vistas.
    form.fields['old_password'].widget.attrs['placeholder'] = 'Ingrese su contraseña actual'
    form.fields['new_password1'].widget.attrs['placeholder'] = 'Ingrese su nueva contraseña'
    form.fields['new_password2'].widget.attrs['placeholder'] = 'Repita su nueva contraseña'
    return form

  def post(self, request, *args, **kwargs):
    data = {}
    try:
      action = request.POST['action']
      if action == 'edit':
        form = PasswordChangeForm(user=request.user, data=request.POST)
##como no creamos el formulario, la validación del mismo se debe realizar en el método Post de la vista
        if form.is_valid():
          form.save()
##Esta propiedad permite que se actualice la contraseña y que no cierre la sesión en curso
          update_session_auth_hash(request, form.user)
        else:
          data['error'] = form.errors    ##retornamos los errores a través de AJAX
```

```
            else:
                data['error'] = 'No ha ingresado a ninguna opción'
        except Exception as e:
            data['error'] = str(e)
        return JsonResponse(data)

    def get_context_data(self, **kwargs):
        context = super().get_context_data(**kwargs)
        context['title'] = 'Cambiar Password'
        context['entity'] = 'Password'
        context['list_url'] = self.success_url
        context['action'] = 'edit'
        return context
```

Creamos el template dentro de la aplicación llamado change_password.html que hereda del formulario base:

{% extends 'form.html' %}

Creamos la URL correspondiente para poder acceder en el URLS de la aplicación:

path('change/password/', UserChangePasswordView.as_view(), name='user_change_password'),

Luego seteamos en el template la url para poder acceder como ya lo hicimos en la edición del perfil.

```
<div class="dropdown-divider"></div>
            <a href="{% url 'user:user_change_password' %}" class="dropdown-item">
                <i class="fas fa-lock mr-2"></i> Editar password
            </a>
```

Resetear contraseña de un Usuario mediante correo electronico

Para ello tenemos que tener configurado el correo electrónico como en el siguiente apartado.
En principio para poder hacer esto tenemos que modificar el modelo de usuarios, incorporando un campo llamado TOKEN, el cual va a permitir encriptar la información para poder proteger el envio de datos y no hacerlo sobre el ID directamente. Agregamos el campo al modelo como lo siguiente:
token = models.UUIDField(primary_key=False, editable=False, null=True, blank=True)
Este token se va a generar cuando realicemos un cambio de contraseña, no necesariamente al crear un usuario.
Tambien tenemos que crear una variable en el settings.py para referenciar al dominio del sistema:
DOMAIN = ''

Tenemos que crear un formulario que va a contener lo necesario para el reseteo de la contraseña:

```
class ResetPasswordForm(forms.Form):            ##Utilizamos el form base porque no vamos a utilizar un Modelo
    username = forms.CharField(widget=forms.TextInput(attrs={
        'placeholder': 'Ingrese su nombre de usuario,
        'class': 'form-control',
        'autocomplete': 'off'
    }))

    def clean(self):                    ###Sobreescribimos el método para saber si el usuario ingresado existe
        cleaned = super().clean()
        if not User.objects.filter(username=cleaned['username']).exists():  ##Si no existe
            self._errors['error'] = self._errors.get('error', self.error_class())
            self._errors['error'].append('El usuario no existe')                    ##Retorna un error
        return cleaned

    def get_user(self):                 ###Creamos el método este para obtener el usuario a partir del USERNAME ingresado
        username = self.cleaned_data.get('username')
        return User.objects.get(username=username)
```

Tambien creamos el formulario para poder reestablecer la contraseña luego del envio del correo electrónico:

```
class ChangePasswordForm(forms.Form):
    password = forms.CharField(widget=forms.PasswordInput(attrs={
        'placeholder': 'Ingrese su nueva contraseña',
```

```
        'class': 'form-control',
        'autocomplete': 'off'
    }))

    confirmPassword = forms.CharField(widget=forms.PasswordInput(attrs={
        'placeholder': 'Repita su nueva contraseña',
        'class': 'form-control',
        'autocomplete': 'off'
    }))

    def clean(self):
        cleaned = super().clean()
        password = cleaned['password']
        confirmPassword = cleaned['confirmPassword']
        if password != confirmPassword:
            self._errors['error'] = self._errors.get('error', self.error_class())
            self._errors['error'].append('Las contraseñas deben ser iguales')
        return cleaned
```

Luego creamos una vista para poder presentar la misma:

```
class ResetPasswordView(FormView):   #Se valida que el usuario exista para poder enviar un email
    form_class = ResetPasswordForm      #apuntamos al formulario creado
    template_name = 'login/resetpwd.html'        #debemos crear un template que va a heredar del base.html
    success_url = reverse_lazy(setting.LOGIN_REDIRECT_URL)

    @method_decorator(csrf_exempt)
    def dispatch(self, request, *args, **kwargs):
        return super().dispatch(request, *args, **kwargs)

##Metodo definido para el envio de email para reseteo de contraseña con el USER como parámetro para obtener datos
def send_email_reset_pwd(self, user):
        data = {}
        try:
#Para obtener la URL preguntamos si DEBUG es True es modo DESARROLLO False PRODUCCION. Entonces por el else
# en modo desarrollo que obtenga la URL para poder generarla con el UUID4 para resetear la contraseña.
            URL = settings.DOMAIN if not settings.DEBUG else self.request.META['HTTP_HOST']
            user.token = uuid.uuid4()           #GENERAMOS EL TOKEN PARA VALIDAR EL CAMBIO DE CONTRASEÑA
            user.save()                         #GUARDAMOS EL USUARIO CON EL TOKEN GENERADO
                                                ##Despues modificar con el almacenamiento en BD
            mailServer = smtplib.SMTP(settings.EMAIL_HOST, settings.EMAIL_PORT)
            mailServer.starttls()               ##Despues modificar con el almacenamiento en BD
            mailServer.login(settings.EMAIL_HOST_USER, settings.EMAIL_HOST_PASSWORD)

            email_to = user.email      ##OBTENEMOS EL EMAIL DEL USER
            mensaje = MIMEMultipart()
            mensaje['From'] = settings.EMAIL_HOST_USER
            mensaje['To'] = email_to
            mensaje['Subject'] = 'Reseteo de contraseña'       ##MENSAJE DE ASUNTO
#Se crea un template para el envio del correo electrónico para resetear la contraseña y se pasan los valores siguientes
            content = render_to_string('login/send_email.html', {
                'user': user,      ##pasamos el usuario nomas
##ENVIAMOS LA URL + EL TOKEN DEL USUARIO PARA QUE PUEDA RESETEAR SU CONTRASEÑA
                'link_resetpwd': 'http://{}/login/change/password/{}/'.format(URL, str(user.token)),
                'link_home': 'http://{}'.format(URL)
            })
            mensaje.attach(MIMEText(content, 'html'))
```

```
            mailServer.sendmail(settings.EMAIL_HOST_USER,
                                email_to,
                                mensaje.as_string())
        except Exception as e:
            data['error'] = str(e)
        return data

    def post(self, request, *args, **kwargs):
        data = {}
        try:
            form = ResetPasswordForm(request.POST)  # self.get_form()
                        ###a la variable form se asigna lo que llega del POST
            if form.is_valid():                     #Con el método clean del formulario se chequea que el usuario exista
                user = form.get_user()              #Obtenemos el objeto usuario a partir del método get_user del form.py
#Con el método definido mas arriba enviamos el correo, y si devuelve un error lo capturamos en DATA
                data = self.send_email_reset_pwd(user)
            else:
                data['error'] = form.errors
        except Exception as e:
            data['error'] = str(e)
        return JsonResponse(data, safe=False)

    def get_context_data(self, **kwargs):
        context = super().get_context_data(**kwargs)
        context['title'] = 'Reseteo de Contraseña'
        return context
```

Creamos la vista para cambiar la contraseña después del envio de correo electrónico:

```
class ChangePasswordView(FormView):
    form_class = ChangePasswordForm
    template_name = 'login/changepwd.html'
    success_url = reverse_lazy(setting.LOGIN_REDIRECT_URL)

    @method_decorator(csrf_exempt)
    def dispatch(self, request, *args, **kwargs):
        return super().dispatch(request, *args, **kwargs)

    def get(self, request, *args, **kwargs):
        token = self.kwargs['token']                    ##OBTENEMOS EL TOKEN del usuario
#Si un usuario tiene el token generado, se le da paso para que siga al reseteo de la contraseña
        if User.objects.filter(token=token).exists():
            return super().get(request, *args, **kwargs)
#si no tiene token reenvia al login.
        return HttpResponseRedirect('/login/')

    def post(self, request, *args, **kwargs):
        data = {}
        try:
            form = ChangePasswordForm(request.POST)
            if form.is_valid():
##Pasamos como parámetro el TOKEN que viene de la URL en vez del ID
                user = User.objects.get(token=self.kwargs['token'])
                user.set_password(request.POST['password']) #Seteamos la contraseña
#Generamos un nuevo token para que no pueda generar la contraseña con esa URL
                user.token = uuid.uuid4()
                user.save()                         #guardamos el usuario
            else:
```

```
            data['error'] = form.errors
        except Exception as e:
            data['error'] = str(e)
        return JsonResponse(data, safe=False)

    def get_context_data(self, **kwargs):
        context = super().get_context_data(**kwargs)
        context['title'] = 'Reseteo de Contraseña'
        context['login_url'] = settings.LOGIN_URL
        return context
```

En el archivo urls.py de la aplicación creamos las url:

```
path('reset/password/', ResetPasswordView.as_view(), name='reset_password'),
path('change/password/<str:token>/', ChangePasswordView.as_view(), name='change_password'),
```

El template resetpwd.html se crea en la carpeta de la aplicación y contiene lo siguiente:

```
{% extends 'login/base.html' %}

{% block content %}
    <div class="login-box">
        <div class="login-logo">
            <a href="{% url 'index' %}"><b>Ha</b>des</a>
        </div>
        <div class="card">
            <div class="card-body login-card-body">
                <p class="login-box-msg">Reseteo de contraseña</p> ///AGREGAMOS EL NOMBRE
                <form action=".." method="post">
                    <input type="hidden" name="next" value="{{ next }}">
                    {% csrf_token %}
                    <div class="input-group mb-3">
                        {{ form.username }}                    //Llamamos al campo necesario y quitamos el password
                        <div class="input-group-append">
                            <div class="input-group-text">
                                <span class="fas fa-envelope"></span>
                            </div>
                        </div>
                    </div>
                    <hr>
                    <div class="row">
                        <div class="col-lg-12">
                            <button type="submit" class="btn btn-primary btn-block">
                                <i class="fas fa-envelope"></i> Enviar          //Boton para enviar email para resetear password
                            </button>
                        </div>
                    </div>
                    <p class="text-center mt-2" style="font-size: 13px;">
                        Si deseas volver al login da un click <a
                            href="{% url 'login' %}">aqui </a>
                    </p>
                </form>
            </div>
        </div>
        <script type="application/javascript">
            $(function () {
                $('form').on('submit', function (e) {
                    e.preventDefault();
                    var parameters = new FormData(this);
```

```
            submit_with_ajax(window.location.pathname, 'Notificación', '¿Estas seguro de resetear tu contraseña?',
parameters, function () {
                Swal.fire({
                  title: 'Notificación',
                  text: 'Se ha enviado un correo electrónico con los pasos a seguir para que pueda resetear su contraseña',
                  icon: 'success',
                  timer: 5000,
                  onClose: () => {
                    location.href = '/login';          //QUE RETORNE A LA PAGINA PRINCIPAL DE LOGIN
                  }
                }).then((result) => {

                });
              });
            });
        });
    </script>
  </div>
{% endblock %}
```

Tambien hay que crear un template para el cambio de la contraseña llamado changepwd.html con lo siguiente:

```
{% extends 'login/base.html' %}

{% block content %}
  <div class="login-box">
    <div class="login-logo">
      <a href="{% url 'index' %}"><b>Ha</b>des</a>
    </div>
    <div class="card">
      <div class="card-body login-card-body">
        <p class="login-box-msg">Cambio de contraseña</p>
        <form action=".." method="post">
          <input type="hidden" name="next" value="{{ next }}">
          {% csrf_token %}
          <div class="input-group mb-3">
            {{ form.password }}
            <div class="input-group-append">
              <div class="input-group-text">
                <span class="fas fa-lock"></span>
              </div>
            </div>
          </div>
          <div class="input-group mb-3">
            {{ form.confirmPassword }}
            <div class="input-group-append">
              <div class="input-group-text">
                <span class="fas fa-lock"></span>
              </div>
            </div>
          </div>
          <hr>
          <div class="row">
            <div class="col-lg-12">
              <button type="submit" class="btn btn-primary btn-block">
                <i class="fas fa-lock"></i> Cambiar password
              </button>
            </div>
```

```
            </div>
            <p class="text-center mt-2" style="font-size: 13px;">
               Si deseas volver al login da un click <a
                  href="{% url 'login' %}">aqui </a>
            </p>
         </form>
      </div>
   </div>
   <script type="application/javascript">
      $(function () {
         $('form').on('submit', function (e) {
            e.preventDefault();
            var parameters = new FormData(this);
            submit_with_ajax(window.location.pathname, 'Notificación', '¿Estas seguro de cambiar tu contraseña?',
parameters, function () {
               Swal.fire({
                  title: 'Notificación',
                  text: 'Tu contraseña ha sido cambiada correctamente',
                  icon: 'success',
                  timer: 5000,
                  onClose: () => {
                     location.href = '{{ login_url }}';
                  }
               }).then((result) => {

               });
            });
         });
      });
   </script>
</div>
{% endblock %}
```

En el login.html hay que agregar la etiqueta necesaria para redireccionar al reseteo:

```
<p class="text-center mt-2" style="font-size: 13px;">
    Resetear contraseña click <a href="{% url 'reset_password' %}">aqui </a>
</p>
```

Envío de Correos en Django mediante GMAIL

Para poder realizar el envio de correos electrónicos mediante Django debemos configurar una serie de campos en el archivo settings.py:

```
# Email
EMAIL_HOST = 'smtp.gmail.com'
EMAIL_PORT = 587
EMAIL_HOST_USER = 'djangologin99@gmail.com'        ###Cargamos el correo electrónico a utilizar
EMAIL_HOST_PASSWORD = 'dj@ngo1994.'                ###Cargamos la contraseña del correo electrónico
```

VER DE TENER LOS PARAMETROS DE CORREO Y PASSWORD EN UNA TABLA EN LA PARTE DE CONFIGURACION PARA PODER SACAR DEL ARCHIVO SETTINGS.PY

En este tutorial: http://chuwiki.chuidiang.org/index.php?title=Enviar_y_leer_email_con_python_y_gmail esta la parte de configuración que podríamos utilizar:

```
# Establecemos conexion con el servidor smtp de Gmail

mailServer = smtplib.SMTP('smtp.gmail.com',587)
mailServer.ehlo()
mailServer.starttls()
mailServer.ehlo()
mailServer.login("usuario@gmail.com","password")
```

Aca se podría pasar como lo siguiente por ej:

mailServer = smtplib.SMTP(configuración.email_direccion,configuración.email_puerto)
mailServer.login(configuración.email_correo, configuración.email_password)

En GMAIL hay que permitir el acceso de aplicaciones menos seguras desde la administración de la cuenta.

Creamos un archivo send_email.py dentro de la aplicación que contenga lo siguiente:

```
def send_email(form):
  try:
                                    #Ver de reemplazar por algo almacenado en la BD
    mailServer = smtplib.SMTP(settings.EMAIL_HOST, settings.EMAIL_PORT)
    #print(mailServer.ehlo())   #Para probar la conexión con el servidor
    mailServer.starttls()
    #print(mailServer.ehlo())   #Para probar la conexión segura con el servidor
                                    #Ver de reemplazar por algo almacenado en la BD
    mailServer.login(settings.EMAIL_HOST_USER, settings.EMAIL_HOST_PASSWORD)

    #print('Conectado al servidor de correo')

    # informacion solicitud
    email_to = solicitante_email(form)

    #print('Enviando Correo..')

    # Construimos el mensaje
    #Mensaje plano USAR ESTE
    mensaje = MIMEText("""Mensaje del EMAIL""")
    #Mensaje de tipo COMPUESTO, para usar imágenes, código HTML, archivos adjuntos, USAR ESTE
    mensaje = MIMEMultipart()
        #DESPUES
                            #Ver de reemplazar por algo almacenado en la BD
    mensaje['From'] = settings.EMAIL_HOST_USER      ###email remitente
    mensaje['To'] = email_to    #DESTINATARIO DEL CORREO, Se toma del FORMULARIO
    mensaje['Subject'] = "Sistema de Reservas Académicas - FCEQyN - UNaM"  #Asunto
        #El contenido pasamos
        #Para mensaje plano
        #Se envía diretamente sin el content ni el mensaje.attach
        #Para mensaje compuesto con un template
    content = render_to_string('solicitudes/send_email_solicitud.html', {'email': email_to})
    mensaje.attach(MIMEText(content, 'html'))
    mailServer.sendmail(settings.EMAIL_HOST_USER, email_to, mensaje.as_string())
    #print('Correo enviado correctamente')
  except Exception as e:
    print(e)
```

Para poder enviar mensajes compuestos, debemos descargar una plantilla para envio de correos electrónicos desde:

https://beefree.io/ y debemos guardar en la carpeta base de templates y colocarle un nombre, por ej: send_email.html

Este template se redirecciona en el mensaje, en **content = render_to_string('…/send_email.html', {'email': email_to})**
Es importante que se cambie los atributos de imágenes de la plantilla desde la página web, haciendo click derecho sobre la misma y copiando la dirección de la imagen. Esto debe reemplazarse en el template descargado para poder redireccionar al sitio web donde tiene la imagen. Para probar esto, seria recomendable ir probando la plantilla descargada desde el navegador hasta que quede OK. Esto se debe a que generalmente pasaremos parámetros adicionales en el correo como por ej Nombre de Usuario. Para poder enviar el usuario hay que importar:
config.wsgi import *

Desarrollo de un CRUD Compuesto (SALE-DETSALE)

Video 57 hasta Video 63

Se debe crear una carpeta de vistas para la venta y procedemos a copiar la estructura de la vista Create de cualquiera de las otras vistas realizadas. Luego procedemos a crear el Form porque la vista genérica CreateView referencia a un formulario. Copiamos de otro y reemplazamos con los componentes del modelo.
Para este tipo de vistas, como el proceso es mas complejo, que lleva un detalle, no se utiliza el método save del formulario, porque se realiza de otra manera. Esto también implica incorporar al método init cuestiones de atributos de los componentes, porque no se utiliza la plantilla por defecto del form.html, sino del list.html.
Luego, creamos dentro de la carpeta templates la carpeta correspondiente y creamos el create.html que herede del list.html base. Recordar de que el list.html tenga el bloque de contenido, porque el mismo será editado para que funcione con el modulo de venta. Luego agregamos en el urls.py de la aplicación los datos en el PATH.
Toda la lógica de JS que se involucre en la venta, va a ir en un JS de la aplicación correspondiente. Donde se alojaran las funciones necesarias para que funcione correctamente.

Editar plantilla de template

Para poder dividir el formulario debemos guiarnos de la plantilla de adminlte3. Como la base es el list.html. Los componentes que necesitemos agregar deben ir dentro de:

Para poder encontrar un componente que querramos aplicar a nuestra pagina debemos:

- Inspeccionar elemento (en la pagina)
- Click derecho – Edit has html
- Copiar el contenido y pegar en el template en donde lo necesitemos.

Para poder agrupar los componentes debemos crearlos dentro de un form-group.

Por defecto la plantilla ocupa todo el ancho, para poder agrupar en varias partes al mismo nivel se debe dividir en fragmentos. La grilla total ocupa 12 columnas. Un Diseño podría ser como el siguiente:

```
{% block content %}
  <form method="post">
    <div class="card card-primary">
      <div class="card-header">
        <h3 class="card-title">
          {% if action == 'add' %}
            <i class="fas fa-plus"></i>
          {% else %}
            <i class="fas fa-edit"></i>
          {% endif %}
          {{ title }}
        </h3>
      </div>
       <div class="card-body">
       <div class="row">
        <div class="col-lg-8">              ##Esto se agrupa para ocupar 8 columnas a la izquierda
          <div class="card card-secondary">
            <div class="card-header">
              <h3 class="card-title"><i class="fas fa-boxes"></i> Detalle de productos</h3>
            </div>
            <div class="card-body">
                #### ESTO SERIA UN FORM-GROUP
              <div class="form-group">
                <label>Buscador de productos:</label>
                <div class="input-group">
                  ##se agrega el nombre para referenciar en la función del form.js de buscar productos
                  <input type="text" class="form-control" name="search"
                      placeholder="Ingrese una descripción de producto" autocomplete="off">
                  <span class="input-group-append">
```

```
                ####Agregamos el identificador btnClearSearch para darle la funcionalidad al boton
                <button type="button" class="btn btn-danger btn-flat btnClearSearch"><i
                    class="fas fa-times"></i></button>
              </span>
            </div>
          </div>
          #### ACA TERMINA EL FORM GROUP
          <hr>     ##SALE UNA RAYA PARA ESTABLECER LA SEPARACION
          ###Se agrega un botón para borrar todo y se asocia con la etiqueta btnRemoveAll para hacer la funcion
          <button type="button" class="btn btn-danger btn-xs btn-flat btnRemoveAll">
              <i class="fas fa-trash"></i> Eliminar todos los productos
            </button>
          <hr>     ##SALE UNA RAYA PARA ESTABLECER LA SEPARACION
          <table class="table table-bordered" id="tablaProducts">
          ##Diseñamos la tabla del det_sale utilizando el mismo diseño de base de todos los DATATABLES
            <thead>
            <tr>
              <th>Eliminar</th>
              <th>Producto</th>
              <th>Categoría</th>
              <th>PVP</th>
              <th>Cantidad</th>
              <th>Subtotal</th>
            </tr>
            </thead>
            <tbody>
            </tbody>
          </table>
        </div>
      </div>
    </div>
    <div class="col-lg-4">              ##Esto se agrupa para ocupar 4 columnas a la derecha
      <div class="card card-secondary">
        <div class="card-header">
          <h3 class="card-title"><i class="fas fa-shopping-cart"></i> Datos de la factura</h3>
        </div>
        <div class="card-body">
          <input type="hidden" name="action" value="{{ action }}">        ####Componente OCULTO para enviar el
                                                                          ####action a la VISTA
          <div class="form-group">
            <label>Fecha de venta:</label>
            {{ form.date_joined }}          ###Componente del form, de la clase Sale
          </div>
          <div class="form-group">
            <label>Cliente:</label>
            {{ form.cli }}                  ###Componente del form, de la clase Sale
          </div>
          <div class="form-group">
            <label>Subtotal:</label>
            {{ form.subtotal }}             ###Componente del form, de la clase Sale, se calcula del det_sale
          </div>
          <div class="form-group">
            <label>IVA:</label>
            {{ form.iva }}                  ###Componente del form, de la clase Sale, se calcula del det_sale
          </div>
          <div class="form-group">
            <label>IVA Calculado:</label>###Componente del form, para saber importe del IVA según el %
```

```
                    <input type="text" class="form-control" readonly name="ivacalc" value="0.00">
                  </div>
                  <div class="form-group">
                    <label>Total a pagar:</label>
                    {{ form.total }}                    ###Componente del form, de la clase Sale, se calcula del det_sale
                  </div>
                </div>
              </div>
            </div>
          </div>
        </div>
```

toJSON para moneda con dos decimales

En el mismo modelo se trabaja en el método toJSON para el formato de $ con dos posiciones decimales

```
class Product(models.Model):
    name = models.CharField(max_length=150, verbose_name='Nombre', unique=True)
    cat = models.ForeignKey(Category, on_delete=models.CASCADE, verbose_name='Categoría')
    image = models.ImageField(upload_to='product/%Y/%m/%d', null=True, blank=True, verbose_name='Imagen')
    pvp = models.DecimalField(default=0.00, max_digits=9, decimal_places=2, verbose_name='Precio de venta')

    def __str__(self):
        return self.name

    def toJSON(self):
        item = model_to_dict(self)
        item['cat'] = self.cat.toJSON()         #se invoca al toJSON de categoría para que traiga el objeto completo
        item['image'] = self.get_image()
        item['pvp'] = format(self.pvp, '.2f')           #SE SETEA PARA QUE EL DICCIONARIO DEVUELVA DECIMAL CON 2 POS.
        return item

    def get_image(self):
        if self.image:
            return '{}{}'.format(MEDIA_URL, self.image)
        return '{}{}'.format(STATIC_URL, 'img/empty.png')
```

Busqueda de productos para agregar al detalle

Para esto, se arma una estructura en javascript, dentro de una variable, que va a contener lo que es la cabecera (SALE) y el detalle de los productos (DETSALE) y se envía por AJAX para que reciba la vista, realice las iteraciones e inserte los valores en las tablas correspondientes.
En el form.js se agrega la estructura que se compone de dos variables una cabecera y dentro un array con el detalle.

```
var tblProducts;
var vents = {
    //Variable de la cabecera
    items: {
        cli: '',
        date_joined: '',
        subtotal: 0.00,
        iva: 0.00,
        total: 0.00,
        products: []            //Variable de tipo array que va a contener los productos del detalle
    },
    calculate_invoice: function () {        //función para cargar calcular subtotal, iva y total
        var subtotal = 0.00;                    //inicializar la variable
        var iva = $('input[name="iva"]').val();         //porcentaje que se aplica del IVA
        $.each(this.items.products, function (pos, dict) {
            dict.subtotal = dict.cant * parseFloat(dict.pvp);    //realizamos la operación por cada ítem en el detalle
```

```
            subtotal+=dict.subtotal;
        });
        this.items.subtotal = subtotal;
        this.items.iva = this.items.subtotal * iva;                 //Calculo del subtotal por el % para saber el importe de IVA
        this.items.total = this.items.subtotal + this.items.iva;          //Se asigna en el campo TOTAL
//cargamos subtotal de la venta, la sumatoria de subtotales del detalle y al ser float se puede redondear asi como sigue
        $('input[name="subtotal"]').val(this.items.subtotal.toFixed(2));
        $('input[name="ivacalc"]').val(this.items.iva.toFixed(2));
        $('input[name="total"]').val(this.items.total.toFixed(2));
    },
    add: function(item){
        this.items.products.push(item);
        this.list();
    },
    list: function () {         //Funcion para ir agregando los productos al DATATABLES
    this.calculate_invoice();         //se llama a la función para calcular subtotales, iva y total
    tblProducts = $('#tablaProducts').DataTable({//Nombre según lo definido en el create.html
            responsive: true,
            autoWidth: false,
            destroy: true,
            data: this.items.products,          //en vez de enviar por AJAX tomamos lo que viene de la variable items.
                                                //para enviar una colección de diccionarios.
            columns: [
                {"data": "id"},
                {"data": "name"},
                {"data": "cat.name"},   //al ser FK se pone el nombre del campo de la FK.nombre_campo de la Tabla Foranea
                {"data": "pvp"},
                {"data": "cant"},          //se setea la cantidad antes de enviar en la función de seleccionar el producto.
                {"data": "subtotal"},
            ],
            columnDefs: [
                {
                    targets: [0],              //en el campo del ID se coloca el botón para eliminar el producto.
                    class: 'text-center',
                    orderable: false,
                    render: function (data, type, row) {
                                                //Se agrega la etiqueta relativa “remove” para poder capturarla luego
                        return '<a rel="remove" class="btn btn-danger btn-xs btn-flat" style="color: white;"><i class="fas fa-trash-
alt"></i></a>';                 }
                },
                {
                    targets: [-3],
                    class: 'text-center',
                    orderable: false,
                    render: function (data, type, row) {
                        return '$' + parseFloat(data).toFixed(2);               //se agrega el símbolo y se castea para 2 decimales
                    }
                },
                {
                    targets: [-2],
                    class: 'text-center',
                    orderable: false,
                    render: function (data, type, row) {
                        return '<input type="text" name="cant" class="form-control form-control-sm input-sm"
autocomplete="off" value="'+row.cant+'">';                  //se setea para que tome el valor que viene en cant
                    }
                },
```

```
            {
                targets: [-1],
                class: 'text-center',
                orderable: false,
                render: function (data, type, row) {
                    return '$' + parseFloat(data).toFixed(2);                //se agrega el símbolo y se castea para 2 decimales
                }
            },
        ],
         //esta propiedad permite modificar ciertos valores de la tabla e incorporar el touchspin en cada iteración y en el
         //lugar que corresponde
        rowCallback(row, data, displayNum, displayIndex, dataIndex) {
            $(row).find('input[name="cant"]').TouchSpin({
                min: 1, //valor minimo
                max: 1000000000,    //valor maximo
                step: 1          //se va incrementando de uno en uno
            });

        },
        initComplete: function (settings, json) {
        }
    });
  },
};
```

Dentro del componente tenemos que buscar los productos, por ello se agrega la función autocomplete que ya hemos utilizado anteriormente con jqueryui.

Incorporamos al template:

```
{% block head_list %}
  <link href="{% static 'lib/jquery-ui-1.12.1/jquery-ui.min.css' %}" rel="stylesheet"/>
  <script src="{% static 'lib/jquery-ui-1.12.1/jquery-ui.min.js' %}"></script>
```

Agregamos el bloque de la función al archivo form.js

```
// search products
  $('input[name="search"]').autocomplete({     //CHEQUEAR en el template que el campo tenga el NAME!
    source: function (request, response) {
        $.ajax({
            url: window.location.pathname,
            type: 'POST',
            data: {
                'action': 'search_products',              //AGREGAR este action en la vista
                'term': request.term
            },
            dataType: 'json',
        }).done(function (data) {
            response(data);
        }).fail(function (jqXHR, textStatus, errorThrown) {
            //alert(textStatus + ': ' + errorThrown);
        }).always(function (data) {
        });
    },
    delay: 500,
    minLength: 1,
    select: function (event, ui) {
        event.preventDefault();             //Sirve para detener el evento y poder realizar lo sgte debajo.
        //console.clear();
        ui.item.cant = 1;                   //Seteamos esta cantidad por defecto, luego podremos cambiar con el plugin.
```

```
        ui.item.subtotal = 0.00;            //Seteamos por defecto, luego podremos calcular por la cantidad.
        //console.log(vents.items);         //PARA ver si va llenándose el array de productos en el detalle.
        vents.add(ui.item);                 //se llama a la función add para que agregue y desde ahí llame a list
        $(this).val('');                    //Una vez que selecciono un producto que limpie el buscador
    }
  });
```

//evento borrar todos los productos

```
  $('.btnRemoveAll').on('click', function () {
    if (vents.items.products.length === 0) return false;       //agregamos la pregunta para que si no esta vacio se ejecute
        ##ESTA FUNCION AGREGAR AL functions.js general de static/
    alert_action('Notificación', '¿Estas seguro de eliminar todos los productos del detalle?', function () {
      vents.items.products = [];            //REINICIALIZO el array a CERO
      vents.list();         //Vuelvo a listar de nuevo
    }, function () {
    } );
  });
```

Busqueda de productos a través de Select2

Video 67

Para esto realizaremos con el autocomplete, a través de select2. Debemos en principio quitar el componente del input y el botón integrado, y cambiar por un tipo select como lo siguiente:

```
<div class="form-group">
   <label>Buscador de productos:</label>
   <div class="input-group">
        <select class="form-control select2" style="width: 100%;" name="search"       //agregar name para referenciarlo
        </select>
   </div>
</div>
```

Luego hay que agregar la funcionalidad en el form.js referenciando al componente “search”

```
$('select[name="search"]').select2({
    theme: "bootstrap4",
    language: 'es',
    allowClear: true,
    ajax: {
      delay: 250,
      type: 'POST',
      url: window.location.pathname,
      data: function (params) {
        var queryParameters = {
          term: params.term,
          action: 'search_products'     //el action se reemplaza con el de la vista, en este caso search_products
        }
        return queryParameters;
      },
      processResults: function (data) {
        return {
          results: data
        };
      },
    },
    placeholder: 'Ingrese una descripción',
    minimumInputLength: 1,
    templateResult: formatRepo,                 //para la funcionalidad extra de visualizar mas datos del producto
  }).on('select2:select', function (e) {
```

```
//este evento se realiza para pasar del select2 al datatable de detalle producto
    var data = e.params.data;   //se crea la variable para capturar el data del select2
    data.cant = 1;              //cargamos la información necesaria de la estructura de javascript creada
    data.subtotal = 0.00;
    vents.add(data);            //se llama a la variable vents para cargar el ítem.
    $(this).val('').trigger('change.select2');      //realizamos esto para limpiar el select2
  });
```

Como el select2 no funciona con el ítem['value'] sino con 'text' debemos cambiar ese valor en la vista en el método POST en el action correspondiente de la siguiente manera:

```
if action == 'search_products':
        data = []
        prods = Product.objects.filter(name__icontains=request.POST['term'])[0:10]
        for i in prods:
            item = i.toJSON()
            #item['value'] = i.name
            item['text'] = i.name           //se reemplaza el valor!
            data.append(item)
```

Funcionalidad extra al buscar producto con SELECT2

Se puede extender la funcionalidad del select2 para que al ir filtrando productos traiga los elementos complementarios al producto, por ejemplo la IMAGEN del mismo a través de un método formatRepo(repo). Para ello debemos crear la función en el form.js de la siguiente manera:

```
function formatRepo(repo) {
  if (repo.loading) {
    return repo.text; //sin esto da error, se utiliza por si no encuentra nada, que salga por el texto en el autocomplete
  }
  var option = $(
    '<div class="wrapper container">'+
    '<div class="row">' +
    '<div class="col-lg-1">' +
        //SE PASA el repo.image donde image es el campo del producto que contiene la imagen.
    '<img src="' + repo.image + '" class="img-fluid img-thumbnail d-block mx-auto rounded">' +
    '</div>' +
    '<div class="col-lg-11 text-left shadow-sm">' +
    '<p style="margin-bottom: 0;">' +
    '<b>Nombre:</b> ' + repo.name + '<br>' +
    '<b>Categoría:</b> ' + repo.cat.name + '<br>' +
    '<b>PVP:</b> <span class="badge badge-warning">$'+repo.pvp+'</span>'+
    '</p>' +
    '</div>' +
    '</div>' +
    '</div>');
  return option;
}
```

function alert_action

```
function alert_action(title, content, callback, cancel) {          //VA EN EL static/js/functions.js
  $.confirm({
    theme: 'material',
    title: title,
    icon: 'fa fa-info',
    content: content,
    columnClass: 'small',
    typeAnimated: true,
    cancelButtonClass: 'btn-primary',
```

```
        draggable: true,
        dragWindowBorder: false,
        buttons: {
            info: {
                text: "Si",
                btnClass: 'btn-primary',
                action: function () {
                    callback(data);
                }
            },
            danger: {
                text: "No",
                btnClass: 'btn-red',
                action: function () {
                    cancel();
                }
            },
        }
    })
}

// event cant
    $('#tableProducts tbody')
        .on('click', 'a[rel="remove"]', function () {    //evento ELIMINAR PRODUCTO
            var tr = tblProducts.cell($(this).closest('td, li')).index();    //obtenemos la posición de fila y asigno a la variable TR
            alert_action('Notificación', '¿Estás seguro de eliminar el producto de tu detalle?', function () {
                vents.items.products.splice(tr.row, 1);            //se utiliza el método SPLICE para poder eliminar el producto
                vents.list();                //una vez eliminado realizo refresco el listado
            }, function () {
            });
        })
        .on('change', 'input[name="cant"]', function () {                //EVENTO MODIFICAR CANT. PROD.
            console.clear();
            var cant = parseInt($(this).val());            //se tiene que convertir a INTEGER por el tipo de INPUT
            var tr = tblProducts.cell($(this).closest('td, li')).index();
            vents.items.products[tr.row].cant = cant;            //con esto asigno la cantidad
            vents.calculate_invoice();                        //vuelvo a invocar para calcular
             //$('td:eq(5)', tblProducts.row(tr.row).node()).html con esto se la fila y la columna que hay que actualizar el
             //subtotal en el datatable
             // html('$' + vents.items.products[tr.row].subtotal.toFixed(2)); asignamos a ese lugar la variable SUBTOTAL
            $('td:eq(5)', tblProducts.row(tr.row).node()).html('$' + vents.items.products[tr.row].subtotal.toFixed(2));
        });

// event borrar búsqueda de productos
    $('.btnClearSearch').on('click', function () {
        $('input[name="search"]').val('').focus();    //limpia el botón y adquiere el foco
    });

// event submit                                ####Para guardar
    $('form').on('submit', function (e) {
        e.preventDefault();            // Detenemos el envio para hacerlo mediante AJAX
       //Agregamos la validación para que no permita guardar ventas sin productos
        if(vents.items.products.length === 0){
            message_error('Debe al menos tener un producto');
            return false;       //si no tiene productos se termina el proceso
        }
        vents.items.date_joined = $('input[name="date_joined"]').val();    //Enviamos la fecha de la factura
```

```
    vents.items.cli = $('select[name="cli"]').val();                    //Enviamos los datos del cliente
    var parameters = new FormData();
    parameters.append('action', $('input[name="action"]').val());       //Toma los datos del componente ACTION
//Se agrega como VARIABLE un diccionario convertido a STRING para que pueda ir a la vista y pueda ser leído e iterado
    parameters.append('vents', JSON.stringify(vents.items));
    submit_with_ajax(window.location.pathname, 'Notificación', '¿Estas seguro de realizar la siguiente acción?',
parameters, function () {
        location.href = '/erp/sale/list/';              //Una vez realizado el listado de ventas se coloca esta URL
    });
  });

vents.list();
```

Se agrega esto fuera del proceso para que al inicializarse el formulario se visualize el DATATABLES. Esto se puso aqui para que funcione bien el editar y calcule bien los valores del iva, sino tomaría el valor del iva de la base debe tomar el que pusimos al inicializarlo.

En el VIEWS.py, en el método POST agregar:

```
if action == 'search_products':
        data = []
        prods = Product.objects.filter(name__icontains=request.POST['term']) [0:10]   //agregar cantidad de registros
        for i in prods:
          item = i.toJSON()
          item['value'] = i.name
          data.append(item)
elif action == 'add':
        with transaction.atomic():      //NECESARIO para que si hay un problema realice un ROLLBACK
          //como lo que llega del form es un string con esto que sigue convertimos a JSON
          vents = json.loads(request.POST['vents'])
          sale = Sale()  //creamos una variable de tipo venta y asignamos los valores de la CABECERA
          sale.date_joined = vents['date_joined']
          sale.cli_id = vents['cli']
          sale.subtotal = float(vents['subtotal'])
          sale.iva = float(vents['iva'])
          sale.total = float(vents['total'])        //se puede calcular directamente entre sale.iva * sale.total
          sale.save()   //Guardamos la venta para proceder a hacer el DETALLE
          for i in vents['products']:     //se itera los productos dentro del DETALLE
            det = DetSale()               //Creamos la variable de tipo DETALLE
            det.sale_id = sale.id         //Realizamos la relación del detalle con la venta
            det.prod_id = i['id']
            det.cant = int(i['cant'])
            det.price = float(i['pvp'])
            det.subtotal = float(i['subtotal'])   //Se puede calcular directamente entre det.cant * det.price
            det.save()
```

Se debe especificar en el return por los datos serializados: return JsonResponse(data, safe=False) y se tiene que desactivar el mecanismo de defensa antes del método dispatch. @method_decorator(csrf_exempt)

Buscar y Crear un cliente dentro de la factura

Primero debemos agregar una línea en el formulario para poder inicializar el componente Select2 donde van a estar cargados los clientes:

```
class SaleForm(ModelForm):
  def __init__(self, *args, **kwargs):
    super().__init__(*args, **kwargs)
    self.fields['cli'].queryset = Client.objects.none()
```

Para que el cliente se visualice el nombre completo en el Select2 debemos agregar una función en el modelo de clientes para retornar el nombre completo:

```
  def __str__(self):                    ////El método devuelve el nombre completo y el DNI para poder buscar por esos
    return self.get_full_name()         ////parámetros
```

```
    def get_full_name(self):
        return '{} {} / {}'.format(self.names, self.surnames, self.dni)
```

Luego, se debe crear la función para buscar los clientes en la factura para que funcione por Select2, como el de productos. Para ello debemos incorporar la función al form.js como la siguiente:

```
$('select[name="cli"]').select2({
    theme: "bootstrap4",
    language: 'es',
    allowClear: true,
    ajax: {
        delay: 250,
        type: 'POST',
        url: window.location.pathname,
        data: function (params) {
            var queryParameters = {
                term: params.term,
                action: 'search_clients'
            }
            return queryParameters;
        },
        processResults: function (data) {
            return {
                results: data
            };
        },
    },
    placeholder: 'Ingrese una descripción',
    minimumInputLength: 1,
});

$('.btnAddClient').on('click', function () {                    //Funcion para mostrar el modal de clientes al hacer click
    $('#myModalClient').modal('show');
});

$('#myModalClient').on('hidden.bs.modal', function (e) {
    $('#frmClient').trigger('reset');                           //Funcion para ocultar el modal y resetear el formulario
})

$('#frmClient').on('submit', function (e) {
    e.preventDefault();
    var parameters = new FormData(this);
    parameters.append('action', 'create_client');
    submit_with_ajax(window.location.pathname, 'Notificación',
        '¿Estas seguro de crear al siguiente cliente?', parameters, function (response) {
            //console.log(response);
            var newOption = new Option(response.full_name, response.id, false, true);
            $('select[name="cli"]').append(newOption).trigger('change');
            $('#myModalClient').modal('hide');
        });
});
```

Despues debemos agregar la funcionalidad al evento POST en la creación y en la edición, desde la vista de la aplicación:

```
        elif action == 'search_clients':
            data = []
            term = request.POST['term']                 ///Seteamos en una variable lo que llega del autocompletado
            clients = Client.objects.filter(
                Q(names__icontains=term) | Q(surnames__icontains=term) | Q(dni__icontains=term))[0:10]
```

La “Q” representa la llamada al OR, la “,” representa un AND por eso se utiliza la “Q”

```
            for i in clients:
                item = i.toJSON()
                item['text'] = i.get_full_name()        ///Traemos el método completo para visualizarlo en el Select2
                data.append(item)
        elif action == 'create_client':
            with transaction.atomic():                  //aplicamos por si pasa algo durante la creación del cliente
                frmClient = ClientForm(request.POST)
                data = frmClient.save()
```

Para incorporar el botón para agregar un cliente en caso de que no se lo encuentre, debemos modificar el template create.html a fin de agregarlo en el Select2. Lo modificamos para que quede como lo que sigue:

```
<div class="form-group">
  <label>Cliente:</label>
  <div class="input-group">
    {{ form.cli }}
    <div class="input-group-append">
      <button class="btn btn-success btn-flat btnAddClient" type="button">
        <i class="fas fa-user-plus"></i>
      </button>
    </div>
  </div>
</div>
```

Para que funcione este tipo de botón debemos cambiar también en el form.py por lo siguiente:

```
class Meta:
    model = Sale
    fields = '__all__'
    widgets = {
        'cli': forms.Select(attrs={
            'class': 'custom-select select2',           //Este tipo nos permite incorporar el botón para hacer la creacion
            # 'style': 'width: 100%'            ///Quitamos el style para que el botón aparezca al final del select2 al costado.
        }),
```

Al final del bloque de contenido del template create.html insertamos el código para el modal del cliente con lo siguiente:

```
<!-- Modal -->
<div class="modal fade" id="myModalClient" tabindex="-1" role="dialog" aria-labelledby="exampleModalLabel"
    aria-hidden="true">
    <form id="frmClient" enctype="multipart/form-data" method="post">
//Le damos un nombre y le pasamos el tipo de encriptación para la imagen y el método que se va a utilizar
        <div class="modal-dialog" role="document">
            <div class="modal-content">
                <div class="modal-header">
                    <h5 class="modal-title" id="exampleModalLabel">
                        <b><i class="fas fa-user-plus"></i> Nuevo cliente</b>
                    </h5>
                    <button type="button" class="close" data-dismiss="modal" aria-label="Close">
                        <span aria-hidden="true">&times;</span>
                    </button>
                </div>
                <div class="modal-body">
                    {% for field in frmClient.visible_fields %}
//Iteramos los componentes del cliente, con el frmClient porque lo pasamos en el context_data
                        <div class="form-group">
                            <label for="email">{{ field.label }}:</label>
                            {{ field|add_class:'form-control'|attr:'autocomplete:off' }}
                        </div>
                    {% endfor %}
```

```
            </div>
            <div class="modal-footer">
              <button type="submit" class="btn btn-primary btn-block btn-flat">
                <i class="fas fa-save"></i> Guarda r
              </button>
            </div>
          </div>
        </div>
      </form>
    </div>
```

Para poder visualizar el modal con el formulario del cliente debemos agregar en la vista de crear, en el método context_data lo siguiente:

```
  def get_context_data(self, **kwargs):
    context['frmClient'] = ClientForm()
```

Como se trabaja con dos fomularios, debemos modificar el evento submit, primero debemos agregarle un nombre al formulario principal desde el create.html:

```
{% block content %}
  <form id="frmSale" method="post">
```

Y luego el evento submit del form.js:

```
  $('#frmClient').on('submit', function (e) {
    e.preventDefault();
    var parameters = new FormData(this);
    parameters.append('action', 'create_client');                    //opción que se carga al POST de la vista
    submit_with_ajax(window.location.pathname, 'Notificación',
      '¿Estas seguro de crear el cliente?', parameters, function (response) {
        var newOption = new Option(response.full_name, response.id, false, true);
//Modificamos el ultimo parámetro por TRUE para que aparezca seleccionado en el Select2 al hacer la creacion
        $('select[name="cli"]').append(newOption).trigger('change');
        $('#myModalClient').modal('hide');
      });
  });
```

Para que quede el cliente seleccionado que acabamos de crear en el Select2 debemos cambiar el método save del cliente en el forms.py por lo siguiente:

```
def save(self, commit=True):
    data = {}
    form = super()
    try:
      if form.is_valid():
        instance = form.save()          ///Se crea una variable de la instancia del cliente creado
        data = instance.toJSON()
      else:
        data['error'] = form.errors
    except Exception as e:
      data['error'] = str(e)
    return data
```

Como el Select2 devuelve varios campos, debemos modificar el método toJSON en el modelo de cliente para que devuelva el método que creamos de full_name (nombre completo), para ello:

```
    item['full_name'] = self.get_full_name()
```

Por ultimo debemos sobreescribir el método get_form del UpdateView, para que al editar una venta nos aparezca el cliente, porque al inicializar el formulario lo habíamos inicializado vacio. Para ello:

```
  def get_form(self, form_class=None):
    instance = self.get_object() //variable que obtiene el objeto actual
    form = SaleForm(instance=instance)          //le pasamos la variable creada a la variable de instancia q tiene el form
```

```
form.fields['cli'].queryset = Client.objects.filter(id=instance.cli.id) //seteamos el campo con el objeto actual
return form
```

Busqueda de productos con Modal

Esto nos va a servir para buscar productos cuando se nos complique buscarlos desde el Select2. Para ello debemos modificar el tipo de componente del template create.html por lo siguiente:

```
<div class="form-group">
    <label>Buscador de productos:</label>
    <div class="input-group">
      <select class="form-control select2" name="search"></select>
      <div class="input-group-append">
        <button class="btn btn-primary dropdown-toggle" type="button"
            data-toggle="dropdown" aria-haspopup="true" aria-expanded="false">Opciones
        </button>
        <div class="dropdown-menu">                    //LAS OPCIONES DENTRO DEL DESPLEGABLE
          <a class="dropdown-item btnSearchProducts"> //Nombre para poder llamar desde el js
            <i class="fas fa-search"></i> Buscar productos      //botón para abrir el modal
          </a>
          <a class="dropdown-item btnClearSearch">
            <i class="fas fa-times"></i> Limpiar búsqueda      //botón para limpiar la busqueda
          </a>
        </div>
      </div>
    </div>
  </div>
```

Luego agregamos el modal para la búsqueda de productos al final del create.html:

```
<div class="modal fade" id="myModalSearchProducts" tabindex="-1" role="dialog" aria-hidden="true">
    <div class="modal-dialog modal-lg" role="document">
      <div class="modal-content">
        <div class="modal-header">
          <h5 class="modal-title" id="exampleModalLabel">
            <b><i class="fas fa-search"></i> Búsqueda de Productos</b>
          </h5>
          <button type="button" class="close" data-dismiss="modal" aria-label="Close">
            <span aria-hidden="true">&times;</span>
          </button>
        </div>
        <div class="modal-body">
          <table class="table table-bordered table-hover" id="tblSearchProducts">
            <thead>
            <tr>
              <th>Producto</th>
              <th>Categoría</th>
              <th>Imagen</th>
              <th>Precio Unitario</th>
              <th>Opciones</th>
            </tr>
            </thead>
            <tbody>
            </tbody>
          </table>
        </div>
      </div>
    </div>
  </div>
```

Despues debemos crear la función en el form.js:

```
$('.btnSearchProducts').on('click', function () {
    tblSearchProducts = $('#tblSearchProducts').DataTable({
        responsive: true,
        autoWidth: false,
        destroy: true,
        deferRender: true,
        ajax: {
            url: window.location.pathname,
            type: 'POST',
            data: {
                'action': 'search_products',
                'term': $('select[name="search"]').val()
            },
            dataSrc: ""
        },
        columns: [
            {"data": "name"},
            {"data": "cat.name"},
            {"data": "image"},
            {"data": "pvp"},
            {"data": "id"},
        ],
        columnDefs: [
            {
                targets: [-3],
                class: 'text-center',
                orderable: false,
                render: function (data, type, row) {
                    return '<img src="' + data + '" class="img-fluid d-block mx-auto" style="width: 20px; height: 20px;">';
                }
            },
            {
                targets: [-2],
                class: 'text-center',
                orderable: false,
                render: function (data, type, row) {
                    return '$' + parseFloat(data).toFixed(2);
                }
            },
            {
                targets: [-1],
                class: 'text-center',
                orderable: false,
                render: function (data, type, row) {
                    var buttons = '<a rel="add" class="btn btn-success btn-xs btn-flat"><i class="fas fa-plus"></i></a> ';
                    return buttons;
                }
            },
        ],
        initComplete: function (settings, json) {
        }
    });
    $('#myModalSearchProducts').modal('show');
});
```

Despues tenemos que modificar en la vista del create y del update, el método post, para el action “search_products”:

```
        if action == 'search_autocomplete ':
            data = []
```

```
            term = request.POST['term'].strip()      //asignamos a term lo que viene por el POST de la busqueda
            data.append({'id': term, 'text':term})   //para que no borre lo que ingresamos en la busqueda
            products = Product.objects.filter()      //Cargamos todos los productos
            if len(term):                //si lo que viene del post tiene algo me busca según el filtro, sino muestra todos
               products = products.filter(name__icontains=term)
            for i in products[0:10]:
               item = i.toJSON()
               item['text'] = i.name
               data.append(item)
```

Tenemos que crear una función en el form.js para la tabla dentro del modal para asignarle las acciones:

```
   $('#tblSearchProducts tbody')
      .on('click', 'a[rel="add"]', function () {
         var tr = tblSearchProducts.cell($(this).closest('td, li')).index();      //asignamos a TR lo q se selecciona en tabla
         var product = tblSearchProducts.row(tr.row).data();                      //asignamos a producto todo el objeto
         product.cant = 1;                                                        //agregamos cantidad por defecto
         product.subtotal = 0.00;                                                 //subtotal inicial y se calcula
         vents.add(product);                                                      //invocamos a la función para agregar le prod.
      });
```

tblSearchProducts se debe crear como variable al comienzo del form.js

Tambien debemos agregar un IF a la función formatRepo del form.js después del (repo.loading)

```
   if (!Number.isInteger(repo.id)) {        //si el repo.id no devuelve un valor entero entonces retorna el texto
      return repo.text;
   }
```

Esto mismo tenemos que agregar a la función del Select dentro del select2 y debería quedar como lo que sigue:

```
}).on('select2:select', function (e) {
      var data = e.params.data;
      if(!Number.isInteger(data.id)){       //si el data.id no tiene ID entonces no agregar al datatable
         return false;
      }
      data.cant = 1;
      data.subtotal = 0.00;
      vents.add(data);
      $(this).val('').trigger('change.select2');
   });
```

Validaciones de stock en las ventas

En los modelos hay que agregar el atributo de stock en la clase producto. Tiene que ser de tipo ENTERO.

```
stock = models.IntegerField(default=0, verbose_name='Stock')
```

En el forms.py se puede personalizar el campo si se requiere.

Tambien debemos modificar el list.html y el list.js para poder visualizar el stock en el Datatables. Para ello agregamos la columna en ambos:

```
HTML:       <th scope="col" style="width: 10%;">Stock</th>
JS:         {"data": "stock"},
```

Ademas en el list.js debemos personalizar la columna, por ej con lo que sigue:

```
{
            targets: [-3],
            class: 'text-center',
            orderable: false,
            render: function (data, type, row) {
               if(row.stock > 0){
                  return '<span class="badge badge-success">'+data+'</span>'
               }
               return '<span class="badge badge-danger">'+data+'</span>'
            }
         },
```

Si se requiere que el sistema solo venda los productos que hay en STOCK debemos modificar en las vistas CREATE y UPDATE el método POST con lo siguiente:

```
def post(self, request, *args, **kwargs):
    data = {}
    try:
        action = request.POST['action']
        if action == 'search_products':     //también replicar en el search_autocomplete
            data = []
            ids_exclude = json.loads(request.POST['ids'])
            term = request.POST['term'].strip()
            products = Product.objects.filter(stock__gt=0) //Con esto traemos los productos con stock mayor a CERO
            if len(term):
                products = products.filter(name__icontains=term)
            for i in products.exclude(id__in=ids_exclude)[0:10]:
                item = i.toJSON()
                item['value'] = i.name
                # item['text'] = i.name
                data.append(item)
```

Para presentar el stock en el detalle del datatables y del modal debemos hacer lo siguiente:

En el template create.html modificamos la parte del modal agregando la columna stock en donde iba categoría:

```
<th>Stock</th>
```

También esto tenemos que realizar en el form.js en la función del modal ('.btnSearchProducts') donde además agregamos la personalización a la definición de la columna:

Columnas:

```
        columns: [
            {"data": "full_name"},          //reemplazamos el name por el full_name que definimos en el toJSON del model
            {"data": "image"},
            {"data": "stock"},
            {"data": "pvp"},
            {"data": "id"},
        ],
columnDefs: [
            {
                targets: [-3],
                class: 'text-center',
                render: function (data, type, row) {
                    return '<span class="badge badge-secondary">' + data + '</span>';          //Presentamos con color el stock
                }
            },
```

Para retornar el nombre y la categoría en un solo campo debemos modificar en la tabla models.py el método toJSON y agregamos el campo "full_name" para que nos devuelva concatenado el nombre del producto y la categoría:

```
    item['full_name'] = '{} / {}'.format(self.name, self.cat.name)
```

Para visualizar en la búsqueda por select2 debemos modificar la función:

```
function formatRepo(repo) {
    '<b>Nombre:</b> ' + repo.full_name + '<br>' +       //reemplazamos name por full_name
    '<b>Stock:</b> ' + repo.stock + '<br>' +            //reemplazamos categoría por la de stock
```

Para visualizar en el detalle del datatables debemos modificar en la funcion:

```
  list: function () {
        columns: [
            {"data": "id"},
            {"data": "full_name"},          //reemplazamos name por full_name
            {"data": "stock"},              //reemplazamos categoría por la de stock
        columnDefs: [                       //agregamos la personalización del detalle de la columna de stock
            {
```

```
            targets: [-4],
            class: 'text-center',
            render: function (data, type, row) {
                return '<span class="badge badge-secondary">' + data + '</span>';
            }
        },
```

Si se permite únicamente ventas de productos con stock se debe controlar que no se venda mas que lo que hay en stock, para ello dentro del form.js debemos cambiar la función rowCallback modificando el campo MAX por el de stock del producto:

```
    rowCallback(row, data, displayNum, displayIndex, dataIndex) {

        $(row).find('input[name="cant"]').TouchSpin({
            min: 1,
            max: data.stock,       //ACA SE MODIFICA EL MAXIMO POR EL STOCK DEL PRODUCTO
            step: 1
        });
    },
```

Para descontar el stock una vez realizada la venta debemos modificar el método POST el action ADD y UPDATE por lo siguiente:

```
elif action == 'add':
        with transaction.atomic():
            vents = json.loads(request.POST['vents'])
            sale = Sale()
            sale.date_joined = vents['date_joined']
            sale.cli_id = vents['cli']
            sale.subtotal = float(vents['subtotal'])
            sale.iva = float(vents['iva'])
            sale.total = float(vents['total'])
            sale.save()
            for i in vents['products']:
                det = DetSale()
                det.sale_id = sale.id
                det.prod_id = i['id']
                det.cant = int(i['cant'])
                det.price = float(i['pvp'])
                det.subtotal = float(i['subtotal'])
                det.save()
                det.prod.stock -= det.cant          //DESCONTAMOS LA CANTIDAD DE STOCK
                det.prod.save()                     //Guardamos el detalle de productos
```

Tambien podríamos realizar de la siguiente manera:

```
                prod = Product.objects.get(pk=det.prod_id)
                prod.stock = prod.stock – det.cant
```

Evitar ingresar productos repetidos en la venta

Si queremos evitar que se ingresen productos repetidos en el detalle, sino jugar con la cantidad, debemos agregar una función en la variable vents, para obtener los productos que ya están en el detalle. Hacer lo siguiente:

```
var vents = {
  items: {
    cli: '',
    date_joined: '',
    subtotal: 0.00,
    iva: 0.00,
    total: 0.00,
    products: []
  },
  get_ids: function () {            //Esta función agrega un array con los productos en el detalle
```

```
    var ids = [];
    $.each(this.items.products, function (key, value) {          ///recorro el detalle
      ids.push(value.id);          ///agrego los ID
    });
    return ids;
  },
```

Luego debemos agregar esos datos a la función del select2 y del modal:

```
$('select[name="search"]').select2({                              //SELECT2
    theme: "bootstrap4",
    language: 'es',
    allowClear: true,
    ajax: {
      delay: 250,
      type: 'POST',
      url: window.location.pathname,
      data: function (params) {
        var queryParameters = {
          term: params.term,
          action: 'search_autocomplete',
          ids: JSON.stringify(vents.get_ids())
        }                    //Debemos convertir a string para parsear y agregar el array de ids para capturar en la VISTA

  $('.btnSearchProducts').on('click', function () {                    //MODAL
    tblSearchProducts = $('#tblSearchProducts').DataTable({
      responsive: true,
      autoWidth: false,
      destroy: true,
      deferRender: true,
      ajax: {
        url: window.location.pathname,
        type: 'POST',
        data: {
          'action': 'search_products',
          'ids': JSON.stringify(vents.get_ids()),
//Debemos convertir a string para parsear y agregar el array de ids para capturar en la VISTA
```

Luego en la vista debemos agregar en el método POST en el action **search_products** y **search_autocomplete**

```
  def post(self, request, *args, **kwargs):
    data = {}
    try:
      action = request.POST['action']
      if action == 'search_products':
        data = []
        ids_exclude = json.loads(request.POST['ids'])
//Se crea una variable con lo que llega del POST, se convierte a un Diccionario porque lo que llega es un STRING
        term = request.POST['term'].strip()
        products = Product.objects.filter(stock__gt=0)
        if len(term):
          products = products.filter(name__icontains=term)
        for i in products.exclude(id__in=ids_exclude)[0:10]:     //Se excluyen los productos contra la variable creada
          item = i.toJSON()
          item['value'] = i.name
          # item['text'] = i.name
          data.append(item)
```

Dentro del Modal tenemos que agregar una validación para que no agregue los productos repetidos, para ello debemos eliminar el producto del listado una vez que se lo agrega al detalle de productos. Entonces debemos modificar la función en la acción del botón ADD:

```
$('#tblSearchProducts tbody')
    .on('click', 'a[rel="add"]', function () {
        var tr = tblSearchProducts.cell($(this).closest('td, li')).index();
        var product = tblSearchProducts.row(tr.row).data();
        product.cant = 1;
        product.subtotal = 0.00;
        vents.add(product);
        tblSearchProducts.row($(this).parents('tr')).remove().draw();
                    //Quitamos la fila del datatables del modal al hacer click
    });
```

Reingresar Stock al eliminar una venta

Para ello debemos sobreescribir el método delete del modelo SALE para poder reingresar el stock una vez que se elimina la venta.

```
def delete(self, using=None, keep_parents=False):
    for det in self.detsale_set.all():      ///RECORREMOS POR CADA PRODUCTO EN LA VENTA
        det.prod.stock += det.cant          ///SUMAMOS AL PRODUCTO LA CANTIDAD
        det.prod.save()                     ///GUARDAMOS EL DETALLE DEL PRODUCTO
    super(Sale, self).delete()              ///Se esta sobreescribiendo pero de igual manera se esta ejecutando el delete
```

Plugins y Librerias para campos Date, Time, Subir y Bajar Numeros, Etc

Datetimepicker

Para los campos que sean de tipo Date, Datetime y Time, podemos usar el plugin Tempus Dominus Bootstrap4. Este plugin nos permite poder agregar componentes para usar el datetimepicker y el timepicker. Para instalar en Django: https://getdatepicker.com/5-4/Installing/#django
Tambien tenemos que utilizar el moment.js. Ambos hay que descargar y colocar dentro de la carpeta general static/libs

Para utilizar hay que agregar al block head_list en el siguiente orden. Reemplazando según corresponda:

```
<script src="{% static 'lib/moment-2.25.3/moment.js' %}"></script> # para que implente los idiomas usar el de abajo.
<script src="{% static 'lib/moment-2.25.3/moment-with-locales.js' %}"></script> # para que implente los idiomas
<script src="{% static 'lib/tempusdominus-bootstrap-4/tempusdominus-bootstrap-4.min.js' %}"></script>
<link href="{% static 'lib/tempusdominus-bootstrap-4/tempusdominus-bootstrap-4.min.css' %}" rel="stylesheet"/>
```

Tambien debemos cargar al archivo form.js static dentro de Functions el tipo de componente, para inicializarlo

```
$(function () {
    $('#date_joined').datetimepicker({              #donde #date_joined es el nombre del componente
        format: 'DD-MM-YYYY',       ##Seteamos como queremos visualizar porque por defecto es: YYYY-MM-DD
        date: moment().format("DD-MM-YYYY "),               ##Toma la fecha por defecto NOW a través del momento
        locale: 'es',           #Colocamos en ESPAÑOL el DATETIMEPICKER
        //minDate: moment().format("DD-MM-YYYY ") #a partir de la fecha hacia adelante
       //maxDate: moment().format("DD-MM-YYYY ") #como máximo la fecha y hacia atras
    });
```

Para que esto funcione en la definición del componente en el forms.py hay que agregar los widgets necesarios dentro de la clase Meta, para que la librería funcione correctamente:

```
class Meta:
    model = Sale
    fields = '__all__'
    widgets = {
        'date_joined': DateInput(
            format='%Y-%m-%d',
            attrs={
                'value': datetime.now().strftime('%d-%m-%Y'),
                'autocomplete': 'off',                      #para que no interfiera con la seleccion
                'class': 'form-control datetimepicker-input',
                'id': 'date_joined',                ##se coloca el nombre del componente
```

```
            'data-target': '#date_joined',
            'data-toggle': 'datetimepicker'
        }
    ),
```

Touchspin

Para los campos que sean de numericos, podemos usar la librería de Bootstrap Touchspin. Este plugin nos permite poder agregar componentes para incrementar o decrementar valores numéricos y asignar funciones.

https://www.virtuosoft.eu/code/bootstrap-touchspin/

Debemos descargarla y copiarla a la carpeta general static/libs.

Para utilizarla, hay que agregar al block head_list en el siguiente orden. Reemplazando según corresponda:

```
<link href="{% static 'lib/bootstrap-touchspin-4.3.0/jquery.bootstrap-touchspin.css' %}" rel="stylesheet"/>
<script src="{% static 'lib/bootstrap-touchspin-4.3.0/jquery.bootstrap-touchspin.js' %}"></script>
```

Tambien debemos cargar al archivo form.js dentro de Functions el tipo de componente, para inicializarlo

```
$(function () {
  $("input[name='iva']").TouchSpin({    #'iva' nombre del componente
    min: 0,       #valor minimo
    max: 100,   #valor maximo
    step: 0.01, #incremento
    decimals: 2,
    boostat: 5,
    maxboostedstep: 10,
    postfix: '%'            #Agregar el icono
  }).on('change', function () {
    vents.calculate_invoice();   #para calcular nuevamente cuando se modifique el %
  })
  .val(0.12);      #para que inicie con ese valor como minimo
```

Si el campo declarado en el model es numérico, hay que modificar en el forms.py en los widgets para que sea de tipo texto, para que no aparezca las opciones por defecto de incremento, sino lo de la librería utilizada. Esto se realiza en la clase Meta:

```
class Meta:
    model = Sale
    fields = '__all__'
    widgets = {
        'iva': TextInput(attrs={
            'class': 'form-control',
        }),
```

En la librería, en el archivo JS hay que configurar un par de líneas para que el color de los botones dentro del touchspin coincida con el tema de boostrap:

```
Buttondown_class: 'btn btn-primary',    Buttondown_class: 'btn btn-secondary',
Buttonup_class: 'btn btn-primary',        Buttonup_class: 'btn btn-secondary',
```

Listar Ventas

Video 64

Se crea un archivo list.html dentro de los templates de la aplicación y se hereda lo del list.html general. Tambien se lo extiende con los elementos necesarios para poder listar los componentes dentro del datatables:

```
{% extends 'list.html' %}
{% load static %}
{% block head_list %}
  <script src="{% static 'sale/js/list.js' %}"></script>
{% endblock %}
{% block columns %}
  <tr>
```

```
      <th scope="col">Nro</th>
      <th scope="col">Cliente</th>
      <th scope="col">Fecha de registro</th>
      <th scope="col">Subtotal</th>
      <th scope="col">Iva</th>
      <th scope="col">Total</th>
      <th scope="col">Opciones</th>
    </tr>
{% endblock %}
{% block rows %}

{% endblock %}
{% block javascript %}                    //BLOQUE MODAL PARA EL CONSULTAR VENTA
  <div class="modal fade" id="myModelDet" tabindex="-1" role="dialog" aria-labelledby="exampleModalLabel"
       aria-hidden="true">
      <div class="modal-dialog modal-lg" role="document">       //modal-lg es un tamaño mas grande de MODAL
        <div class="modal-content">
          <div class="modal-header">
            <h5 class="modal-title" id="exampleModalLabel"><b><i class="fas fa-shopping-cart"></i> Consulta de
              detalle de venta</b></h5>
            <button type="button" class="close" data-dismiss="modal" aria-label="Close">
              <span aria-hidden="true">&times;</span>
            </button>
          </div>
          <div class="modal-body">
            <table class="table table-bordered" id="tblDet">
              <thead>
              <tr>
                <th>Producto</th>
                <th>Categoría</th>
                <th>PVP</th>
                <th>Cantidad</th>
                <th>Subtotal</th>
              </tr>
              </thead>
              <tbody>
              </tbody>
            </table>
          </div>
        </div>
      </div>
    </div>
{% endblock %}
```

A su vez, se necesita crear un **list.js** dentro de la carpeta static de la aplicación para poder cargar mediante AJAX el datatables y agregar la funcionalidad de los botones en las opciones:

Al inicio del archivo, fuera de la función debemos crear una variable: var tblSale;

```
Var tblSale;
$(function () {
  tblSale = $('#data').DataTable({        //Inicializamos la variable, esto servirá también para el consultar mediante modal
    responsive: true,
    autoWidth: false,
    destroy: true,
    deferRender: true,
    ajax: {
      url: window.location.pathname,
      type: 'POST',
```

```
        data: {
            'action': 'searchdata'
        },
        dataSrc: ""
    },
    columns: [
        {"data": "id"},
        {"data": "cli.names"},
        {"data": "date_joined"},
        {"data": "subtotal"},
        {"data": "iva"},
        {"data": "total"},
        {"data": "id"},
    ],
    columnDefs: [
        {
            targets: [-2, -3, -4],
            class: 'text-center',
            orderable: false,
            render: function (data, type, row) {
                return '$' + parseFloat(data).toFixed(2);
            }
        },
        {
            targets: [-1],
            class: 'text-center',
            orderable: false,
            render: function (data, type, row) {
                //Boton para eliminar una venta
                var buttons = '<a href="/erp/sale/delete/' + row.id + '/" class="btn btn-danger btn-xs btn-flat"><i class="fas fa-trash-alt"></i></a> ';
                //Boton para ver un detalle de VENTA, implementado con MODAL, se agrega la referencia relativa para ello.
                buttons += '<a rel="details" class="btn btn-success btn-xs btn-flat"><i class="fas fa-search"></i></a> ';
                //Boton para UPDATE de la venta
                var buttons = '<a href="/erp/sale/update/' + row.id + '/" class="btn btn-warning btn-xs btn-flat"><i class="fas fa-edit"></i></a> ';
                return buttons;
            }
        },
    ],
    initComplete: function (settings, json) {

    }
});
```

Consultar Detalle Venta mediante Modal

Video 64

Dentro del list.js debemos crear una función para poder referenciar a la acción del botón creado para la consulta

Al inicio del archivo, fuera de la función debe estar una variable: var tblSale; y debe ser inicializada con el datatables:

```
tblSale = $('#data').DataTable({  ..... })    //esto no seria necesario si ya se inicializa para listar.
$('#data tbody')            //dentro de la variable data estaría lo necesario para generar la consulta
    .on('click', 'a[rel="details"]', function () {
        var tr = tblSale.cell($(this).closest('td, li')).index();     //con esto traemos el objeto completo de la fila seleccionada
        var data = tblSale.row(tr.row).data();                        //asignamos a data el valor del tr.row para obtener ID de sale
        $('#tblDet').DataTable({
            responsive: true,
```

```
            autoWidth: false,
            destroy: true,
            deferRender: true,
            //data: data.det,
            ajax: {
                url: window.location.pathname,
                type: 'POST',
                data: {
                    'action': 'search_details_prod',
                    'id': data.id
                },
                dataSrc: ""
            },
            columns: [
                {"data": "prod.name"},
                {"data": "prod.cat.name"},
                {"data": "price"},
                {"data": "cant"},
                {"data": "subtotal"},
            ],
            columnDefs: [
                {
                    targets: [-1, -3],
                    class: 'text-center',
                    render: function (data, type, row) {
                        return '$' + parseFloat(data).toFixed(2);
                    }
                },
                {
                    targets: [-2],
                    class: 'text-center',
                    render: function (data, type, row) {
                        return data;
                    }
                },
            ],
            initComplete: function (settings, json) {
            }
        });
        $('#myModelDet').modal('show');          //Para mostrar el MODAL
    });
```

Para que esto funcione en la vista se debe agregar en el método POST la acción necesaria “search_details”

```
elif action == 'search_details_prod':
            data = []
            for i in DetSale.objects.filter(sale_id=request.POST['id']):          //filtramos con el ID de la Venta del tr.row
                data.append(i.toJSON())                  //se va llenando la data con los diccionarios del detalle correspondiente
```

Los modelos utilizados para sale y det_sale son:

```
class Sale(models.Model):
    cli = models.ForeignKey(Client, on_delete=models.CASCADE)
    date_joined = models.DateField(default=datetime.now)
    subtotal = models.DecimalField(default=0.00, max_digits=9, decimal_places=2)
    iva = models.DecimalField(default=0.00, max_digits=9, decimal_places=2)
    total = models.DecimalField(default=0.00, max_digits=9, decimal_places=2)

    def __str__(self):
        return self.cli.names
```

```
    def toJSON(self):
        item = model_to_dict(self)
        item['cli'] = self.cli.toJSON()
        item['subtotal'] = format(self.subtotal, '.2f')
        item['iva'] = format(self.iva, '.2f')
        item['total'] = format(self.total, '.2f')
        item['date_joined'] = self.date_joined.strftime('%Y-%m-%d')
        item['det'] = [i.toJSON() for i in self.detsale_set.all()]
        return item

    class Meta:
        verbose_name = 'Venta'
        verbose_name_plural = 'Ventas'
        ordering = ['id']

class DetSale(models.Model):
    sale = models.ForeignKey(Sale, on_delete=models.CASCADE)
    prod = models.ForeignKey(Product, on_delete=models.CASCADE)
    price = models.DecimalField(default=0.00, max_digits=9, decimal_places=2)
    cant = models.IntegerField(default=0)
    subtotal = models.DecimalField(default=0.00, max_digits=9, decimal_places=2)

    def __str__(self):
        return self.prod.name

    def toJSON(self):
        item = model_to_dict(self, exclude=['sale'])        //se excluye porque no hay que hacer la conversión del ID
        item['prod'] = self.prod.toJSON()
        item['price'] = format(self.price, '.2f')           //se tiene que convertir el decimal de esta manera para que no
        item['subtotal'] = format(self.subtotal, '.2f')     //genere error
        return item

    class Meta:
        verbose_name = 'Detalle de Venta'
        verbose_name_plural = 'Detalle de Ventas'
        ordering = ['id']
```

En el list.html de la aplicación se debe agregar en el bloque javascript el modal para consultar el detalle, como se especifico en:Listar Ventas BLOQUE MODAL CONSULTAR VENTA

Consultar Detalle Venta mediante Child Rows Datatables

Video 65.

Esto nos permite agregar información extra en el datatables por cada fila del mismo. Para ello debemos agregar a las columnas del datatables lo siguiente:

```
columns: [
        {
            "className": 'details-control',
            "orderable": false,
            "data": null,
            "defaultContent": ''
        },
```

Tambien tenemos que agregar unas imágenes de open y close a la carpeta /static/libs/datatables/images

Luego hay que agregar el código CSS a la hoja de estilo del proyecto en /static/css/mystyle.css

```
td.details-control {
    background: url('../lib/datatables-1.10.20/images/details_open.png') no-repeat center center;
```

```
    cursor: pointer;
}
tr.shown td.details-control {
    background: url('../lib/datatables-1.10.20/images/details_close.png') no-repeat center center;
}
```

Despues hay que agregar la función format el evento de javascript al list.js de la aplicación. Agregamos la función:

```
function format(d) {
    var html = '<table class="table">';                 //creamos una var y armamos la cabecera del datatable detalle
    html += '<thead class="thead-dark">';               //se incorpora el tema oscuro en la cabecera
    html += '<tr><th scope="col">Producto</th>';
    html += '<th scope="col">Categoría</th>';
    html += '<th scope="col">PVP</th>';
    html += '<th scope="col">Cantidad</th>';
    html += '<th scope="col">Subtotal</th></tr>';
    html += '</thead>';
    html += '<tbody>';
    $.each(d.det, function (key, value) {               //armamos la iteración por cada componente del detalle
        html+='<tr>'
        html+='<td>'+value.prod.name+'</td>'
        html+='<td>'+value.prod.cat.name+'</td>'
        html+='<td>'+value.price+'</td>'
        html+='<td>'+value.cant+'</td>'
        html+='<td>'+value.subtotal+'</td>'
        html+='</tr>';
    });
    html += '</tbody>';
    return html;                                        //retornamos el html completo
}
```

Luego agregamos el evento

```
$('#data tbody')            //Dentro del data tbody
.on('click', 'td.details-control', function () {
        var tr = $(this).closest('tr');
        var row = tblSale.row(tr);
        if (row.child.isShown()) {
            row.child.hide();
            tr.removeClass('shown');
        } else {
            row.child(format(row.data())).show();
            tr.addClass('shown');
        }
    });
});
```

Esto pierde la funcionalidad al hacer datatables responsive. Para ello se puede implementar desplazamientos horizontales para poder recorrer hacia los costados, agregando dicha funcionalidad de la siguiente manera:

```
$(function () {
    tblSale = $('#data').DataTable({
        //responsive: true,                             //desactivamos el responsive
        scrollX: true,                                  //habilitamos el scroll en el eje X, también se puede hacer en el eje Y.
```

Update de una Venta

Esto en la realidad no se aplica, pero a fines de aprendizaje en un abm compuesto se ve como realizar.

Creamos la vista correspondiente, la url de la misma y el botón en el list.js para que nos dirija hacia el template de la venta. Como parte de los datos corresponden a una estructura creada en javascript, casi todos los datos no va a traer, sino únicamente los datos del modelo de venta, y no de detalle de venta. Para ello vamos a tener que traer los datos

correspondientes a la venta que pasemos el detalle. Para ello creamos un método en la vista para que traiga los productos de la venta. get_details_product(). La vista quedaría de la siguiente manera:

```
class SaleUpdateView(LoginRequiredMixin, ValidatePermissionRequiredMixin, UpdateView):
    model = Sale
    form_class = SaleForm
    template_name = 'sale/create.html'
    success_url = reverse_lazy('erp:sale_list')
    permission_required = 'erp.change_sale'
    url_redirect = success_url

    @method_decorator(csrf_exempt)
    def dispatch(self, request, *args, **kwargs):
        return super().dispatch(request, *args, **kwargs)

    def post(self, request, *args, **kwargs):
        data = {}
        try:
            action = request.POST['action']
            if action == 'search_products':
                data = []
                prods = Product.objects.filter(name__icontains=request.POST['term'])[0:10]
                for i in prods:
                    item = i.toJSON()
                    item['value'] = i.name
                    data.append(item)
            elif action == 'edit':
                with transaction.atomic():
                    vents = json.loads(request.POST['vents'])
                    #sale = Sale.objects.get(pk=self.get_object().id)        # de esta manera también se puede
                    sale = self.get_object()                                # pero el get_object ya me devuelve la instancia actual.
                    sale.date_joined = vents['date_joined']
                    sale.cli_id = vents['cli']
                    sale.subtotal = float(vents['subtotal'])
                    sale.iva = float(vents['iva'])
                    sale.total = float(vents['total'])
                    sale.save()
                    sale.detsale_set.all().delete()     //como se modifica se debe borrar el detalle para que se itere nuevamente
                    for i in vents['products']:
                        det = DetSale()
                        det.sale_id = sale.id
                        det.prod_id = i['id']
                        det.cant = int(i['cant'])
                        det.price = float(i['pvp'])
                        det.subtotal = float(i['subtotal'])
                        det.save()
            else:
                data['error'] = 'No ha ingresado a ninguna opción'
        except Exception as e:
            data['error'] = str(e)
        return JsonResponse(data, safe=False)
```

Este método es para obtener el detalle de los productos que corresponden a la venta que le pasamos como ID desde el template. Recordar que esta carga esta relacionada al form.js que ya tiene funciones para la carga de los subtotales.

```
    def get_details_product(self):
        data = []
        try:
            for i in DetSale.objects.filter(sale_id=self.get_object().id):          //iteramos los productos filtrando por ID venta
```

```
        // for i in DetSale.objects.filter(sale=self.get_object()):    //también se puede traer de esta manera
            item = i.prod.toJSON()     //utilizamos con el método toJSON() para que funcione, en vez de llamar solo a "i"
            item['cant'] = i.cant     //cargamos la cantidad
            data.append(item)      //agregamos
        except:
            pass
        return data

    def get_context_data(self, **kwargs):
        context = super().get_context_data(**kwargs)
        context['title'] = 'Edición de una Venta'
        context['entity'] = 'Ventas'
        context['list_url'] = self.success_url
        context['action'] = 'edit'
        context['det'] = json.dumps(self.get_details_product())
        return context
```

//enviamos al método como parámetro, en formato JSON. Esta variable tiene que estar en todas las vistas, pero vacias.

En el create.html de la aplicación en el bloque de contenido en SCRIPT agrego lo siguiente:

```
<script>
    vents.items.products = {{ det|safe }};          //se agrega el |safe para que tome el parseo de los componentes "&"
</script>
```

Generacion de PDF's Mediante xhtml2pdf en Django

Video 68 y 69

Para poder utilizar esta librería debemos instalarla en el entorno virtual, con el comando **pip install xhtml2pdf.** Luego, procedemos a crear la vista que va a devolver el PDF. Siguiendo el ejemplo de CRUM complejo, generamos el PDF de la venta realizada.

IMPORTANTE:

Se debe declarar en el settings.py del proyecto el STATIC_ROOT para utilizar en los PDF's la incorporación de imágenes. De la siguiente manera:

STATIC_ROOT = os.path.join(BASE_DIR, 'staticfiles/')

Se crea una carpeta nueva que recolecta en una carpeta todos los archivos para el proyecto. Esto es necesario al momento de hacer el DEPLOY. Se realiza mediante el comando **manage.py collectstatic**

La vista esta basada en la clase genérica View de Django y debe estar compuesta de la siguiente manera:

class SaleInvoicePdfView(View):

```
    def link_callback(self, uri, rel):
        # use short variable names
        sUrl = settings.STATIC_URL
        sRoot = settings.STATIC_ROOT  /
        mUrl = settings.MEDIA_URL
        mRoot = settings.MEDIA_ROOT

        # convert URIs to absolute system paths
        if uri.startswith(mUrl):
            path = os.path.join(mRoot, uri.replace(mUrl, ""))
        elif uri.startswith(sUrl):
            path = os.path.join(sRoot, uri.replace(sUrl, ""))
        else:
            return uri  # handle absolute uri (ie: http://some.tld/foo.png)

        # make sure that file exists
        if not os.path.isfile(path):
            raise Exception(
```

```
            'media URI must start with %s or %s' % (sUrl, mUrl)
        )
    return path

  def get(self, request, *args, **kwargs):
    try:
      template = get_template('sale/invoice.html')       //devuelve un objeto de tipo template, pasar el html
      context = {
        'sale': Sale.objects.get(pk=self.kwargs['pk']),
//se utiliza el self.kwargs'pk' para obtener el objeto para enviar los demás parámetros necesarios para la factura asi
//como también el detalle de la factura. Además Se pasan los datos de la Empresa.
//Despues hay que editar esto para pasar como parámetro. En el HTML esta como parámetro, falta crear tabla Empresa.
        'comp': {'name': 'ALGORISOFT S.A.', 'ruc': '9999999999999', 'address': 'Milagro, Ecuador'},
        'icon': '{}{}'.format(settings.MEDIA_URL, 'logo.png')
      }
      html = template.render(context) //permite incrustrar parámetros, en este caso el diccionario creado.
      response = HttpResponse(content_type='application/pdf')          //genera el pdf y lo descarga
      //Se deshabilita la opción debajo para no descargar el PDF sino visualizarlo en el NAVEGADOR
      #response['Content-Disposition'] = 'attachment; filename="report.pdf"'
       //La función para crear el pdf es la que sigue
      pisaStatus = pisa.CreatePDF(
        html, dest=response,
        link_callback=self.link_callback)
      return response
    except:
      pass
    return HttpResponseRedirect(reverse_lazy('erp:sale_list'))
//Si hay un error, no imprimir, retornar al listado de ventas
```

Una vez creada la vista debemos crear la url correspondiente dentro del urls.py de la aplicación con el parámetro del ID de la venta solicitada.

```
  path('sale/invoice/pdf/<int:pk>/', SaleInvoicePdfView.as_view(), name='sale_invoice_pdf'),
```

El html debe ser como el siguiente:

```
<!DOCTYPE html>
<html>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8"/>
<style>
  .img-invoice {
    width: 50px;
    text-align: center;
    margin: 0 auto;
  }
  .head {
    text-align: center;
    text-transform: uppercase;
  }
  #invoice thead tr th {
    text-align: left;
    border-bottom: 1px solid black;
    border-top: 1px solid black;
    padding-top: 4px;
  }
  #invoice thead tr {
    margin-bottom: 0;
    padding-bottom: 0;
  }
  #invoice tbody tr {
    padding: 0;
```

```
    }
    .text-center{
        text-align: center;
    }
</style>
<body>
<img src="{{ icon }}" class="img-invoice">
<p class="head">
    {{ comp.name|upper }}<br>                    //estos datos hay que crear la tabla con los datos de la compañía para que se cargue de
    {{ comp.ruc }}<br>                           //manera dinámica dichos campos
    {{ comp.address|upper }}                     //se utiliza la propiedad |upper para que se visualizen los campos en MAYUSCULA
</p>
<p>
    <b>FACTURA:</b> {{ sale.id }}<br>
    <b>FECHA DE VENTA:</b> {{ sale.date_joined }}<br>
    <b>CLIENTE:</b> {{ sale.cli.names|upper }}<br>
    <b>DNI:</b> {{ sale.cli.dni }}<br>
</p>
<table id="invoice">
    <thead>
    <tr>
        <th style="width: 20%;">CATEGORIA</th>
        <th style="width: 40%;">PRODUCTO</th>
        <th style="width: 10%;">CANT</th>
        <th style="width: 15%;">P.UNITARIO</th>
        <th style="width: 15%;">TOTAL</th>
    </tr>
    </thead>
    <tbody>
    {% for d in sale.detsale_set.all %}
        <tr {% if forloop.first %}style="padding-top: 3px;" {% endif %}>
            <td>{{ d.prod.cat.name }}</td>
            <td>{{ d.prod.name }}</td>
            <td class="text-center">{{ d.cant }}</td>
            <td class="text-center">${{ d.price }}</td>
            <td style="text-align: right;">${{ d.subtotal }}</td>
        </tr>
    {% endfor %}
    <tr style="border-top: 1px solid black; padding-top: 4px;">
        <td colspan="4"><b>SUBTOTAL</b></td>
        <td style="text-align: right">${{ sale.subtotal }}</td>
    </tr>
    <tr style="padding-top: 0px;">
        <td colspan="4"><b>IVA 12%</b></td>
        <td style="text-align: right">${{ sale.iva }}</td>
    </tr>
    <tr style="padding-top: 0px;">
        <td colspan="4"><b>TOTAL A PAGAR</b></td>
        <td style="text-align: right">${{ sale.total }}</td>
    </tr>
    </tbody>
</table>
<table style="margin-top: 250px;">
    <thead>
    <tr>
        <th>****GRACIAS POR SU COMPRA****</th>
    </tr>
    <tr>
        <th>NOSE ACEPTAN CAMBIOS NI DEVOLUCIONES</th>
    </tr>
    </thead>
</table>
</body>
```

</html>

Agregar PDF a la operación de Venta y al Listado

Video 70

Para esto, necesitamos agregar un botón al list.js en la carpeta static de la aplicación.

buttons += ' ';

target="_blank" nos permite colocar una acción que permita abrir en una ventana o pestaña nueva.

Tambien desde la creación de la venta, tenemos que agregar un evento al hacer submit que pregunte si deseamos generar el PDF de la venta. Para ello podemos utilizar la función desarrollada **alert_action**. La invocamos en el submit de la siguiente manera:

```
//para hacer la venta
submit_with_ajax(window.location.pathname, 'Notificación',
        '¿Estas seguro de realizar la siguiente acción?', parameters, function (response) {
//se agrega el response para obtener el ID de la venta y pasar en la generación del PDF
          //para preguntar si queremos generar el PDF de la venta
           alert_action('Notificación', '¿Desea imprimir la boleta de venta?', function () {
              window.open('/erp/sale/invoice/pdf/' + response.id + '/', '_blank');
              location.href = '/erp/sale/list/';        //Realiza la venta y genera el PDF en una pestaña o ventana nueva
           }, function () {
              location.href = '/erp/sale/list/';        //Solo realiza la venta sin generar el pdf
           });
        });
```

Para que la función **alert_action** funcione, tenemos que agregar un campo mas para el evento CANCEL, si no se quiere generar el PDF para que siga con la operación, por lo tanto la función debería ser como en: function alert_action

En la vista del SaleCreateView y el SaleUdpateView, en el método POST se debe agregar al final del: **for i in vents…..**

Debajo del det.save()

data = {'id': sale.id}

Generacion de PDF's Mediante weasyprint en Django

Para instalar la librería debemos instalar la librería en el entorno virtual:

```
sudo apt-get install build-essential python3-dev python3-pip python3-setuptools python3-wheel python3-cffi libcairo2 libpango-1.0-0 libpangocairo-1.0-0 libgdk-pixbuf2.0-0 libffi-dev shared-mime-info
```

Una vez instalado probamos que funcione la librería con lo siguiente:

```
python -m weasyprint http://weasyprint.org weasyprint.pdf
```

Esto genera un pdf en la carpeta para visualizar que se haya generado correctamente.

A modo de ejemplo creamos un archivo printTicket.py en el raíz de las aplicaciones con lo siguiente:

```
from config.wsgi import *                    //debemos importar para que django pueda interpretar la config en el archivo
from django.template.loader import get_template
from weasyprint import HTML, CSS
from config import settings

def printTicket():
   template = get_template("ticket.html")        //creamos una var y le asignamos una instancia del template
   context = {"name": "Martin Lacheski"}         //pasamos argumentos asignando el nombre
   html_template = template.render(context)
         //creamos una nueva variable y utilizamos el render para pasar como parámetro el contexto
   css_url = os.path.join(settings.BASE_DIR, 'static/lib/bootstrap-4.4.1-dist/css/bootstrap.min.css')
         //pasamos la url del CCS del proyecto uniendo la ruta con os.path.join(settings.BASE_DIR, 'URL DEL
         //bootstrap.min.css')
   HTML(string=html_template).write_pdf(target="ticket.pdf", stylesheets=[CSS(css_url)])
         //HTML es una clase de weasyprint, recibe como parámetros un string con el template
         //write_pdf genera el pdf mediante weasyprint con la propiedad con el nombre del pdf y
         // el estilo CCS asignado a ccs_url a manera de listado de los archivos necesarios de CSS []
         //CSS es una clase de weasyprint que recibe como parámetros la ruta del css que se le va a mandar
```

printTicket()

Luego creamos el archivo ticket.html en la carpeta principal de templates con lo siguiente:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <title>Bootstrap Example</title>
  <meta charset="utf-8">
  <meta name="viewport" content="width=device-width, initial-scale=1">
</head>
<body>

<div class="container">
  <h2 class="text-uppercase">{{ name }}</h2>
  <p>The .table class adds basic styling (light padding and horizontal dividers) to a table:</p>
  <table class="table">
    <thead>
      <tr>
        <th>Firstname</th>
        <th>Lastname</th>
        <th>Email</th>
      </tr>
    </thead>
    <tbody>
      <tr>
        <td>John</td>
        <td>Doe</td>
        <td>john@example.com</td>
      </tr>
      <tr>
        <td>Mary</td>
        <td>Moe</td>
        <td>mary@example.com</td>
      </tr>
      <tr>
        <td>July</td>
        <td>Dooley</td>
        <td>july@example.com</td>
      </tr>
    </tbody>
  </table>
</div>

</body>
</html>
```

Agregar PDF a la operación de Venta y al Listado

Para ello modificamos la clase SaleInvoicePdfView del archivo views.py de la aplicación de venta.

```
class SaleInvoicePdfView(View):

    def get(self, request, *args, **kwargs):
        try:
            template = get_template('sale/invoice.html')                    //Instancia del template
            context = {                                                     //Contenido del CONTEXTO
                'sale': Sale.objects.get(pk=self.kwargs['pk']),
                'comp': {'name': 'ALGORISOFT S.A.', 'ruc': '9999999999999', 'address': 'Milagro, Ecuador'},
                'icon': '{}{}'.format(settings.MEDIA_URL, 'logo.png')
            }
            html = template.render(context)              //AGREGAMOS EL css_url con la ruta del CSS
            css_url = os.path.join(settings.BASE_DIR, 'static/lib/bootstrap-4.4.1-dist/css/bootstrap.min.css')
                        // y la variable pdf con el HTML para poder retornar
            pdf = HTML(string=html, base_url=request.build_absolute_uri()).write_pdf(stylesheets=[CSS(css_url)])
              //el argumento base_url=request.build_absolute_uri() devuelve la ruta absoluta para poder visualizar imagenes
```

```
            return HttpResponse(pdf, content_type='application/pdf')
        except:
            pass
        return HttpResponseRedirect(reverse_lazy('erp:sale_list'))
```

Luego debemos modificar el archivo invoice.html dentro de la carpeta templates de la venta con lo siguiente:

```
<!DOCTYPE html>
{% load static %}
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
    <style>

        * {
            color: black;
        }

        body {
            font-family: "Roboto", "Lucida Grande", Verdana, Arial, sans-serif;
            padding: 0;
            margin: 0;
            color: black;
        }

        .name-company {
            font-size: 30px;
            padding: 0;
            margin: 0;
            font-weight: bold;
            text-transform: uppercase;
            text-align: center;
        }

        table thead tr th {
            border: 1px solid black !important;
            padding: 3px;
        }

        table tbody tr td {
            border: 1px solid black;
            padding: 3px;
        }

        .img-logo {
            margin-top: 10px;
            width: 75px;
            height: 75px;
            margin-bottom: 10px;
        }

    </style>
</head>
<body>
<div class="container-fluid">
    <img src="{% static 'img/logo.png' %}" class="img-fluid d-block mx-auto img-logo">
    <h1 class="name-company">{{ company.name }}</h1>
    <p class="text-center">
        Cdla. Dager, Calle Rio Zamora entre Av.Tumbes y Av. Tarqui<br>
        Celular: +593 0979014551<br>
        Teléfono: 297541221<br>
        Web: algorisoft.com<br>
    </p>
    <p>
        <b>Cliente:</b> {{ sale.cli.get_full_name }}<br>
        <b>Número de cedula:</b> {{ sale.cli.dni }}<br>
        <b>Fecha de venta:</b> {{ sale.date_joined|date:'c' }}<br>
    </p>
    <table class="table" style="width: 100%;">
        <thead>
        <tr style="border: 1px solid black;">
            <th style="width: 20%;">Cantidad</th>
            <th style="width: 40%;">Producto</th>
            <th style="width: 20%;">P.Unitario</th>
            <th style="width: 20%;" class="text-right">Subtotal</th>
        </tr>
```

```
    </thead>
    <tbody>
    {% for det in sale.detsale_set.all %}
      <tr>
        <td class="text-center">{{ det.cant }}</td>
        <td>{{ det.prod.name }}</td>
        <td class="text-center">{{ det.price|floatformat:2 }}</td>
        <td class="text-right">{{ det.subtotal|floatformat:2 }}</td>
      </tr>
    {% endfor %}
    <tr>
      <td colspan="3" class="text-right"><b>Subtotal</b></td>
      <td class="text-right">{{ sale.subtotal|floatformat:2 }}</td>
    </tr>
    <tr>
      <td colspan="3" class="text-right"><b>Iva %</b></td>
      <td class="text-right">{{ sale.iva|floatformat:2 }}</td>
    </tr>
    <tr>
      <td colspan="3" class="text-right"><b>Total a pagar</b></td>
      <td class="text-right">{{ sale.total|floatformat:2 }}</td>
    </tr>
    <tr>
      <td colspan="4">
        <p class="text-uppercase font-weight-bold text-center">
          Total de productos {{ sale.detsale_set.all.count }}<br>
          !Gracias por su preferencia!<br>
          Una vez que usted a recibido el pedido<br>
          No hay derecho a reclamo ni devolución del producto<br>
          Que tenga un excelente día
        </p>
      </td>
    </tr>
    </tbody>
  </table>
</div>
</body>
</html>
```

Reportes en Django

Para poder realizar los reportes es necesario aplicar filtros para poder delimitarlos dinamicamente. Para ello se incorpora el plugin **DATERANGEPICKER**. Lo descargamos y lo copiamos a la carpeta static/lib del proyecto. Este plugin requiere también la librería **MOMENTS** para poder trabajar correctamente. Tambien se incorpora la funcionalidad de agregar botones para exportar a EXCEL y PDF.
Procedemos a crear una nueva aplicación en el proyecto Django con manage.py startapp reports por ejemplo.
Luego registramos en el archivo settings.py la aplicación.

Despues, debemos crear y configurar el archivo urls.py de la aplicación:

```
urlpatterns = [
  # reports
  path('sale/', ReportSaleView.as_view(), name='sale_report'),
]          //Agregamos al URLS general
```

El archivo forms.py con una clase para el reporte, para el componente adicional de Rangos.

```
class ReportForm(Form):
  date_range = CharField(widget=TextInput(attrs={
    'class': 'form-control',
    'autocomplete': 'off'
  }))
```

El archivo views.py creamos una vista para el reporte de ventas:

```
class ReportSaleView(TemplateView):
  template_name = 'sale/report.html'

  @method_decorator(csrf_exempt)
  def dispatch(self, request, *args, **kwargs):
    return super().dispatch(request, *args, **kwargs)
```

```
    def post(self, request, *args, **kwargs):
        data = {}
        try:
            action = request.POST['action']
            if action == 'search_report':                       ###relacionada con el action del JS
                data = []
                start_date = request.POST.get('start_date', '')         ###Tomamos los valores aplicados en el filtro Inicio y Fin
                end_date = request.POST.get('end_date', '')
                search = Sale.objects.all()
                if len(start_date) and len(end_date):    ###El filtro se aplica si fecha de inicio y fin tienen algún valor
                    search = search.filter(date_joined__range=[start_date, end_date])   #Filtramos reporte según rango
                for s in search:
                    data.append([                   ###Se agregan los datos a manera de ARRAY
                        s.id,
                        s.cli.names,
                        s.date_joined.strftime('%d-%m-%Y'),     ###Configuramos el formato de la fecha que necesitamos
                        format(s.subtotal, '.2f'),              ###Seteamos el tipo de dato y la cantidad de decimales
                        format(s.iva, '.2f'),
                        format(s.total, '.2f'),
                    ])
###Para poder tener un resumen debajo del reporte, una vez que aplicamos los filtros, debemos incorporar el
###AGGREGATE para poder realizar subconsultas, entonces se debe hacer lo siguiente:
###donde: R es una variable para obtener el resultado de la subconsulta.
###COALESCE es una función de Django que permite reemplazar si encuentra NULL le pasamos el valor CERO (0)
###Creamos las variables necesarias para el resumen del reporte:
                subtotal = search.aggregate(r=Coalesce(Sum('subtotal'), 0)).get('r')
                iva = search.aggregate(r=Coalesce(Sum('iva'), 0)).get('r')
                total = search.aggregate(r=Coalesce(Sum('total'), 0)).get('r')
                data.append([
                    '---',                  ###Por cada columna en la posición dada que no necesitemos contabilizar
                    '---',
                    '---',
                    format(subtotal, '.2f'),
                    format(iva, '.2f'),
                    format(total, '.2f'),
                ])
            else:
                data['error'] = 'Ha ocurrido un error'
        except Exception as e:
            data['error'] = str(e)
        return JsonResponse(data, safe=False)

    def get_context_data(self, **kwargs):                   //Se sobreescribe para enviar información adicional al template
        context = super().get_context_data(**kwargs)
        context['title'] = 'Reporte de Ventas'              //Titulo de la vista para visualizar en el template
        context['entity'] = 'Reportes'                      //Entidad de la vista para visualizar en el template
        context['list_url'] = reverse_lazy('sale_report')   //redireccion de la vista al refrescar la pagina
        context['form'] = ReportForm()                      //Se llama al Form para incorporar el campo de Rangos
        return context
```

En la aplicación creamos la carpeta de Templates, dentro de ella creamos la carpeta sale y ahí creamos un report.html

```
{% extends 'list.html' %}           //Extendemos del listado que ya tiene las librerías del datatable
{% load static %}
{% block head_list %}
                    ////Librerias necesarias para usar el daterangepicker
    <script src="{% static 'lib/moment-2.25.3/moment.js' %}"></script>
    <script src="{% static 'lib/daterangepicker-3.1/spanish.js' %}"></script>
```

```
  <script src="{% static 'lib/daterangepicker-3.1/daterangepicker.js' %}"></script>
  <link rel="stylesheet" href="{% static 'lib/daterangepicker-3.1/daterangepicker.css' %}"/>
                ////Librerias necesarias para incorporar los botones al DATATABLES
  <link rel="stylesheet" href="{% static 'lib/datatables-1.10.20/plugins/buttons-1.6.1/css/buttons.bootstrap.min.css'
%}"/>
  <script src="{% static 'lib/datatables-1.10.20/plugins/buttons-1.6.1/js/dataTables.buttons.min.js' %}"
type="text/javascript"></script>
  <script src="{% static 'lib/datatables-1.10.20/plugins/jszip-2.5.0/jszip.min.js' %}" type="text/javascript"></script>
  <script src="{% static 'lib/datatables-1.10.20/plugins/pdfmake-0.1.36/pdfmake.min.js' %}"
type="text/javascript"></script>
  <script src="{% static 'lib/datatables-1.10.20/plugins/pdfmake-0.1.36/vfs_fonts.js' %}"
type="text/javascript"></script>
  <script src="{% static 'lib/datatables-1.10.20/plugins/buttons-1.6.1/js/buttons.html5.min.js' %}"
type="text/javascript"></script>
                ////llamamos al javascripts con las funciones para el reporte.
  <script src="{% static 'sale/js/report.js' %}"></script>
{% endblock %}
{% block content %}              //sobreescribimos el bloque de contenido con lo necesario para el reporte
  <div class="card card-primary">
    <div class="card-header">
      <h3 class="card-title">
        <i class="fas fa-chart-bar"></i>          //cambiamos el tipo de icono para el reporte
        {{ title }}
      </h3>
    </div>
    <div class="card-body">
      <div class="row">
        <div class="col-lg-4">
          <div class="form-group">                    ///Creamos el componente para aplicar el DATERANGEPICKER
            <label>Rango de fechas:</label>
            {{ form.date_range }}
          </div>
        </div>
      </div>
      <hr>      //una LINEA para separar de la tabla
      <table class="table table-bordered" id="data">
        <thead>
        <tr>                    //DEFINIMOS las columnas para el reporte con la base del LIST de ventas sin OPCIONES
          <th scope="col">Nro</th>
          <th scope="col">Cliente</th>
          <th scope="col">Fecha de registro</th>
          <th scope="col">Subtotal</th>
          <th scope="col">Iva</th>
          <th scope="col">Total</th>
        </tr>
        </thead>
        <tbody>
        </tbody>
      </table>
    </div>
    <div class="card-footer">
      <a href="{{ list_url }}" class="btn btn-success btn-flat">
        <i class="fas fa-sync"></i> Actualizar
      </a>
    </div>
  </div>
{% endblock %}
```

```
{% block javascript %}
        ///Se sobreescribe para que inicialice VACIO para que no cargue el datatables como el list.html general
{% endblock %}
```

Este plugin requiere la creación de un archivo report.js dentro de la carpeta static de la aplicación.

```
var date_range = null;              //Inicializamos la variable para asociarla al picker
var date_now = new moment().format('DD-MM-YYYY');          //Esta variable la utilizamos para el evento CANCEL
function generate_report() {                ///Reporte de las ventas
  var parameters = {                        ///Necesario para el envio de datos hacia la vista
    'action': 'search_report',              ///Se define la acción para tomar desde la vista
    'start_date': date_now,                 ///Inicializamos las variables de inicio y fin con el dia actual NOW
    'end_date': date_now,
  };
  if (date_range !== null) {                ///Si se aplica el Filtro, asignamos los valores a INICIO y FIN
    parameters['start_date'] = date_range.startDate.format(DD-MM-YYYY);
    parameters['end_date'] = date_range.endDate.format('DD-MM-YYYY');
  }
  $('#data').DataTable({
    responsive: true,
    autoWidth: false,
    destroy: true,
    deferRender: true,
    ajax: {
      url: window.location.pathname,
      type: 'POST',
      data: parameters,
      dataSrc: ""
    },
    order: false,                   ///para que el resumen figure debajo
    paging: false,                  ///para que no haya paginados, sino que figure todo en una sola pagina
    ordering: false,                ///para quitar el ordenamiento
    info: false,                    ///para quitar la información de paginados y registros debajo
    searching: false,               ///para quitar el buscar
    dom: 'Bfrtip',                  ///para implementar los botones de Exportar a Excel y PDF
    buttons: [
      {
        extend: 'excelHtml5',
        text: 'Excel <i class="fas fa-file-excel"></i>',
        titleAttr: 'Excel',
        className: 'btn btn-success btn-flat btn-xs'
      },
      {
        extend: 'pdfHtml5',
        text: 'Pdf <i class="fas fa-file-pdf"></i>',
        titleAttr: 'PDF',
        className: 'btn btn-danger btn-flat btn-xs',
        download: 'open',
        orientation: 'landscape',       //para que salga horizontal en vez de vertical
        pageSize: 'A4',
        //EL PDF es propio de DATATABLES por lo que hay que setearlo para que quede agradable a la vista
        customize: function (doc) {
          doc.styles = {
            header: {
              fontSize: 18,
              bold: true,
              alignment: 'center'
            },
```

```
                subheader: {
                    fontSize: 13,
                    bold: true
                },
                quote: {
                    italics: true
                },
                small: {
                    fontSize: 8
                },
                tableHeader: {
                    bold: true,
                    fontSize: 11,
                    color: 'white',
                    fillColor: '#2d4154',
                    alignment: 'center'
                }
            };
            doc.content[1].table.widths = ['20%', '20%', '15%', '15%', '15%', '15%'];
            doc.content[1].margin = [0, 35, 0, 0];
            doc.content[1].layout = {};
            doc['footer'] = (function (page, pages) {
                return {
                    columns: [
                        {
                            alignment: 'left',
                            text: ['Fecha de creación: ', {text: date_now}]
                        },
                        {
                            alignment: 'right',
                            text: ['página ', {text: page.toString()}, ' de ', {text: pages.toString()}]
                        }
                    ],
                    margin: 20
                }
            });
        }
    }
    ],
    columnDefs: [                   ///DEFINIMOS LOS PARAMETROS PARA que devuelva formato $ y decimal con 2 posic.
    {
        targets: [-1, -2, -3],
        class: 'text-center',
        orderable: false,
        render: function (data, type, row) {
            return '$' + parseFloat(data).toFixed(2);
        }
    },
    ],
    initComplete: function (settings, json) {
    }
  });
}
$(function () {             ///Inicializamos el componente del DATERANGEPICKER
  $('input[name="date_range"]').daterangepicker({
    ///Esto podría ocuparse si hay que cambiar el estilo de los botones al inicializar el componente
    applyButtonClasses: 'btn-primary',
```

```
        cancelButtonClasses: 'btn-danger',
        locale: {                ///cambiamos el formato de visualización de fecha y label e iconos de los botones
            format: 'DD-MM-YYYY',   ////SI SE REQUIERE
            applyLabel: '<i class="fas fa-chart-pie"></i> Aplicar',        //Agregamos icono al boton
            cancelLabel: '<i class="fas fa-times"></i> Cancelar',        //Agregamos icono al boton
        }
    }).on('apply.daterangepicker', function (ev, picker) {            ///Funcion de aplicar Filtro
        date_range = picker;        //tomamos la variable y le asignamos lo que viene del filtro del RANGEPICKER
        generate_report();        //una vez asignado los valores se genera el reporte
    }).on('cancel.daterangepicker', function (ev, picker) {            ///aplicamos que al cancelar, tome el valor del NOW()
        $(this).data('daterangepicker').setStartDate(date_now);
        $(this).data('daterangepicker').setEndDate(date_now);
        date_range = picker;        //asignamos el reporte al dia de la fecha actual
        generate_report();
    });
    generate_report();
});
```

Graficos en Django

Los graficos se implementan en el dashboard del template. Se utiliza la librería https://www.highcharts.com/
Ver alguna alternativa OPENSOURCE porque es paga para proyectos privados.
Se generaran graficos para ver las ventas de todos los meses, etc.
El template quedaría como lo siguiente:

```
{% extends 'layout.html' %}
{% load static %}
{% block head %}       //SE IMPORTAN LAS LIBRERIAS QUE CORRESPONDEN AL TIPO DE GRAFICO QUE ELEGIMOS COMO EL QUE SIGUE:
    <script src="{% static 'lib/highcharts-8.1.2/highcharts.js' %}"></script>
    <script src="{% static 'lib/highcharts-8.1.2/modules/exporting.js' %}"></script>
    <script src="{% static 'lib/highcharts-8.1.2/modules/export-data.js' %}"></script>
    <script src="{% static 'lib/highcharts-8.1.2/modules/accessibility.js' %}"></script>
{% endblock %}
{% block content %}
    <div class="container-fluid">            //UTILIZAMOS ESTE TIPO DE CONTENEDOR DE BOOSTRAP PARA TODO EL GRAFICO
        <div class="row">                    //SE GENERA LA CLASE FILA Y SE LE DA EL ANCHO DE 12 COLUMNAS PARA GRAFICAR LOS 12 MESES
            <div class="col-lg-12">            //AHÍ SE IMPLEMENTA EL CONTAINER QUE VA A TENER EL GRAFICO, EL CUAL ESTA EN EL APARTADO ABAJO JAVASCRIPT
                <div id="container"></div>        ///GRAFICO 1        ID PARA IDENTIFICAR EN LA FUNICION JS – VENTAS DEL AÑO EN MESES
            </div>
        </div>
        <br>
        <div class="row">
            <div class="col-lg-12">            //GRAFICO 2        ID PARA IDENTIFICAR EN LA FUNICION JS – PRODUCTOS VENDIDOS EN EL MES
                <div id="container-pie"></div>
            </div>
        </div>
        <br>
        <div class="row">                    ///GRAFICO 3        ID PARA IDENTIFICAR EN LA FUNICION JS
            <div class="col-lg-12">
                <div id="container-online"></div>
            </div>
        </div>
        <br>
    </div>
    <script type="application/javascript">
//VARIABLE QUE VA A CONTENER LOS VALORES DE LAS COLUMNAS
        var graphcolumn = Highcharts.chart('container', {            //ESTE ES EL GRAFICO 1 DE VENTAS DE LOS 12 MESES DEL AÑO
            chart: {
                type: 'column'
            },
            title: {
                text: 'Reporte de ventas del año 2020'        //ASIGNAMOS EL TITULO DEL REPORTE      ---- VER PARA QUE TOME EL AÑO EN CURSO EL TITLE
            },
            subtitle: {
                text: 'Reporte de columnas'                    //SUBTITULO, PUEDE SER TIPO DE GRAFICO X EJ
            },
            xAxis: {
                categories: [                                //POR DEFECTO VIENE EN INGLES, DEBEMOS SETEAR A ESPAÑOL
                    'Enero',
                    'Febrero',
                    'Marzo',
```

```
            'Abril',
            'Mayo',
            'Junio',
            'Julio',
            'Agosto',
            'Septiembre',
            'Octubre',
            'Noviembre',
            'Diciembre'
        ],
        crosshair: true
    },
    yAxis: {
        min: 0,
        title: {
            text: 'Total de Ventas por MES'                    //ASIGNAR UN VALOR A UNO DE LOS EJES, en este caso eje “Y”
        }
    },
    tooltip: {
        headerFormat: '<span style="font-size:10px">{point.key}</span><table>',
        pointFormat: '<tr><td style="color:{series.color};padding:0">{series.name}: </td>' +
            '<td style="padding:0"><b>{point.y:.1f} $</b></td></tr>',                    //ACA DEBEMOS COLOCAR EL $ para que figure el tipo de VALOR
        footerFormat: '</table>',
        shared: true,
        useHTML: true
    },
    plotOptions: {
        column: {
            pointPadding: 0.2,
            borderWidth: 0
        }
    },
});
var graphpie = Highcharts.chart('container-pie', {                    //ESTE ES EL GRAFICO 2 DE PRODUCT0S DEL MES EN CURSO (PIE = Grafico Tipo PASTEL)
    chart: {
        plotBackgroundColor: null,
        plotBorderWidth: null,
        plotShadow: false,
        type: 'pie'
    },
    title: {
        text: 'Porcentaje de venta de productos en el mes actual'                    /// ver de pasar como VARIABLE PARA QUE SE ACTUALICE
    },
    tooltip: {
        pointFormat: '{series.name}: <b>{point.percentage:.1f}%</b>'
    },
    accessibility: {
        point: {
            valueSuffix: '%'
        }
    },
    plotOptions: {
        pie: {
            allowPointSelect: true,
            cursor: 'pointer',
            dataLabels: {
                enabled: true,
                format: '<b>{point.name}</b>: {point.percentage:.1f} %'
            }
        }
    },
});
Highcharts.chart('container-online', {                    //ESTE ES EL GRAFICO 3 POR EJ VENTAS EN TIEMPO REAL
    chart: {
        type: 'spline',
        animation: Highcharts.svg, // don't animate in old IE
        marginRight: 10,
        events: {
            load: function () {
                var series = this.series[0];
                setInterval(function () {
                    $.ajax({
                        url: window.location.pathname, //window.location.pathname
                        type: 'POST',
                        data: {
                            'action': 'get_graph_online'                    //PASAMOS EL ACTION PARA LA VISTA AL METODO CORRESPONDIENTE
                        },
                        dataType: 'json',
```

```
                }).done(function (data) {
                    if (!data.hasOwnProperty('error')) {                    //SE ARMA LA SERIE CON LOS DATOS ALEATORIOS Y DATETIME
                        var x = (new Date()).getTime();
                        series.addPoint([x, data.y], true, true);
                        return false;
                    }
                    message_error(data.error);
                }).fail(function (jqXHR, textStatus, errorThrown) {
                    alert(textStatus + ': ' + errorThrown);
                }).always(function (data) {
                });
            }, 1000);
        }
    }
  },
  time: {
    useUTC: false
  },
  title: {
    text: 'Live random data'                            //TITULO DE EJEMPLO DEL GRAFICO
  },
  accessibility: {
    announceNewData: {
      enabled: true,
      minAnnounceInterval: 15000,
      announcementFormatter: function (allSeries, newSeries, newPoint) {
        if (newPoint) {
          return 'New point added. Value: ' + newPoint.y;
        }
        return false;
      }
    }
  },
  xAxis: {
    type: 'datetime',
    tickPixelInterval: 150
  },
  yAxis: {
    title: {
      text: 'Value'
    },
    plotLines: [{
      value: 0,
      width: 1,
      color: '#808080'
    }]
  },
  tooltip: {
    headerFormat: '<b>{series.name}</b><br/>',
    pointFormat: '{point.x:%d-%m-%Y %H:%M:%S}<br/>{point.y:.2f}'                    //datetime actual
  },
  legend: {
    enabled: false
  },
  exporting: {
    enabled: false
  },
  series: [{
    name: 'Random data',
    data: (function () {
      // generate an array of random data
      var data = [],
        time = (new Date()).getTime(),
        i;
      for (i = -19; i <= 0; i += 1) {
        data.push({
          x: time + i * 1000,
          y: Math.random()
        });
      }
      return data;
    }())
  }]
});

///Realizamos la función para enviar los datos del grafico mediante AJAX

        //FUNCION 1 DE VENTAS DE LOS 12 MESES DEL AÑO
```

```
    function get_graph_sales_year_month() {
      $.ajax({
        url: window.location.pathname, //window.location.pathname
        type: 'POST',
        data: {
          'action': 'get_graph_sales_year_month'      //seteamos el ACTION
        },
        dataType: 'json',
      }).done(function (data) {
        if (!data.hasOwnProperty('error')) {
          graphcolumn.addSeries(data);                    //PASAMOS LOS DATOS DE LAS COLUMNAS (SERIES), DATA VIENE DE LA VISTA EN EL METODO POST
          return false;
        }
        message_error(data.error);
      }).fail(function (jqXHR, textStatus, errorThrown) {
        alert(textStatus + ': ' + errorThrown);
      }).always(function (data) {

      });
    }
          //FUNCION 2 DE VENTAS PRODUCTOS EN EL MES EN CURSO
    function get_graph_sales_products_year_month() {
      $.ajax({
        url: window.location.pathname, //window.location.pathname
        type: 'POST',
        data: {
          'action': 'get_graph_sales_products_year_month'
        },
        dataType: 'json',
      }).done(function (data) {
        if (!data.hasOwnProperty('error')) {
          graphpie.addSeries(data);                       //PASAMOS LOS DATOS DE LAS SERIES, DATA VIENE DE LA VISTA EN EL METODO POST
          return false;
        }
        message_error(data.error);
      }).fail(function (jqXHR, textStatus, errorThrown) {
        alert(textStatus + ': ' + errorThrown);
      }).always(function (data) {

      });
    }
    $(function () {
          ///CONFIGURAMOS PARA QUE SE INICIE LAS FUNCIONES DE LOS GRAFICOS CUANDO SE EJECUTA EL TEMPLATE BASE
      get_graph_sales_year_month();
      get_graph_sales_products_year_month();
    });
  </script>
{% endblock %}
```

Los graficos se alimentan de métodos que tenemos que definir en la vista del dashboard. Para este ejemplo queda como lo que sigue:

```
class DashboardView(TemplateView):
  template_name = 'dashboard.html'

  @method_decorator(csrf_exempt)
  def dispatch(self, request, *args, **kwargs):
    return super().dispatch(request, *args, **kwargs)

  def post(self, request, *args, **kwargs):
    data = {}
    try:
      action = request.POST['action']
        //GRAFICO 1 VENTAS POR MES DEL AÑO
      if action == 'get_graph_sales_year_month':          //Se coloca el nombre de la función en javascript para que entre y se ejecute
        data = {                                ///SE PASAN LOS DATOS NECESARIOS PARA LA FUNCION
          'name': 'Porcentaje de venta',
          'showInLegend': False,        //Se desactiva para que no se visualize la parte INFERIOR del EJE X
          'colorByPoint': True,         //Se coloca en TRUE para que tome distintos colores por cada Columna.
          'data': self.get_graph_sales_year_month()       //Retorna en DATA los datos de venta de la funcion
        }
        //GRAFICO 2 VENTA DE PRODUCTOS MES EN CURSO
      elif action == 'get_graph_sales_products_year_month':
        data = {                                ///SE PASAN LOS DATOS NECESARIOS PARA LA FUNCION
          'name': 'Porcentaje',
```

```
            'colorByPoint': True,
            'data': self.get_graph_sales_products_year_month(),
        }
         //GRAFICO 3
    elif action == 'get_graph_online':          //Se coloca el nombre de la función en javascript para que entre y se ejecute
      data = {'y': randint(1, 100)}             ///PARA LA FUNCION DE EJEMPLO SE GENERA UN NUMERO ALEATORIO DE 1 A 100
    else:
      data['error'] = 'Ha ocurrido un error'
  except Exception as e:
    data['error'] = str(e)
  return JsonResponse(data, safe=False)
       //METODO PARA EL GRAFICO 1 VENTAS POR MES DEL AÑO
def get_graph_sales_year_month(self):       //definimos este método para traer las ventas por mes del año en curso
  data = []
  try:
    year = datetime.now().year      //obtenemos el año en curso
    for m in range(1, 13):          //genero el FOR para iterar los 12 meses del año
       //se realiza el filtro con el año y el mes asignando al valor M del FOR sumando el total de ventas en el mes
      total = Sale.objects.filter(date_joined__year=year, date_joined__month=m).aggregate(r=Coalesce(Sum('total'), 0)).get('r')
      data.append(float(total))     //parseamos a FLOAT el total.
  except:
    pass
  return data
       //METODO PARA EL GRAFICO 2 VENTA DE PRODUCTOS MES EN CURSO
def get_graph_sales_products_year_month(self):
  data = []
  year = datetime.now().year        //obtenemos el año en curso
  month = datetime.now().month      //obtenemos el mes en curso
  try:
    for p in Product.objects.all():
       //se realiza el filtro con el año, el mes y se agrupa por producto asignando al valor P del FOR sumando el subtotal de productos
      total = DetSale.objects.filter(sale__date_joined__year=year, sale__date_joined__month=month, prod_id=p.id).aggregate(
        r=Coalesce(Sum('subtotal'), 0)).get('r')
      if total > 0:             //si se vendio el producto en el mes creamos un diccionario con el nombre y el total $ por cada producto
        data.append({
          'name': p.name,
          'y': float(total)
        })
  except:
    pass
  return data

def get_context_data(self, **kwargs):
  context = super().get_context_data(**kwargs)
  context['panel'] = 'Panel de administrador'
  context['graph_sales_year_month'] = self.get_graph_sales_year_month()          //Se pasa como contexto el método de las ventas
  return context
```