




Modelos y Apps con las que trabaja el Proyecto

Por Nombre Por Fecha Por Stok → Notar que se puede ordenar por fecha, stock y nombre

COD	Nombres	Precio Venta	Precio Compra	Stok	Num Ventas	Acciones
750521000050	Yogurt Toni	2,30	1,00	35	27	 
702011075291	Milows	2,30	1,50	21	29	 

View

```
def get_queryset(self):
    kword = self.request.GET.get("kword", '')
    order = self.request.GET.get("order", '')
    queryset = Product.objects.buscar_producto(kword, order)
    return queryset
```

Manager

```
def buscar_producto(self, kword, order):
    consulta = self.filter(
        Q(name__icontains=kword) | Q(barcode=kword)
    )
    # verificamos en que orden se solicita
    if order == 'date':
        # ordenar por fecha
        return consulta.order_by('created')
    elif order == 'name':
        # ordenar por nombre
        return consulta.order_by('name')
    elif order == 'stok':
        return consulta.order_by('count')
    else:
        return consulta.order_by('-created')
```

Registrar Producto - Modulo Almacén

```
class ProductCreateView(AlmacenPermisoMixin, CreateView):
    template_name = "producto/form_producto.html"
    form_class = ProductForm
    success_url = reverse_lazy('producto_app:producto-lista')
```

```
# validations
def clean_barcode(self):
    barcode = self.cleaned_data['barcode']
    if len(barcode) < 11:
        raise forms.ValidationError('Ingrese un codigo de barras correcto')

    return barcode

def clean_purchase_price(self):
    purchase_price = self.cleaned_data['purchase_price']
    if not purchase_price > 0:
        raise forms.ValidationError('Ingrese un precio compra mayor a cero')

    return purchase_price

def clean_sale_price(self):
    sale_price = self.cleaned_data['sale_price']
    purchase_price = self.cleaned_data.get('purchase_price')
    if not sale_price >= purchase_price:
        raise forms.ValidationError('El precio de venta debe ser mayor o igual que el precio de compra')

    return sale_price
```

Registrar Producto - Modulo Almacén

```
class ProductUpdateView(AlmacenPermisoMixin, UpdateView):
    template_name = "producto/form_producto.html"
    model = Product
    form_class = ProductForm
    success_url = reverse_lazy('producto_app:producto-lista')
```

Se maneja igual que el create ya que trabaja con el mismo formulario, sin embargo el HTML se cambia dependiendo si es para crear o actualizar ya que muestra un boton de eliminar en el actualizar

Guardar

Eliminar

```
<div class="cell medium-6">
    <button type="submit" class="success button" style="width: 200px;">Guardar</button>
    {% if product %}
    <a href="{% url 'producto_app:producto-delete' product.id %}" class="alert button clear">Eliminar</a>
    {% endif %}
</div>
```

Recordando que cuando se trabaja con update pues se puede acceder al objeto, entonces he usado éso para saber si hay un objeto y en caso de haberlo mostrar el botón

Eliminar Producto - Módulo Almacén

```
class ProductDeleteView(AlmacenPermisoMixin, DeleteView):
    template_name = "producto/delete.html"
    model = Product
    success_url = reverse_lazy('producto_app:producto-lista')
```

```
<div class="grid-x grid-margin-x align-center">
    <h5 class="cell" style="margin-bottom: 1em; text-align: center;">¿Desea eliminar el producto?</h5>
    <form class="cell medium-9 grid-x grid-margin-x align-center"
        action="{% url 'producto_app:producto-delete' product.id %}" method="POST">
        {% csrf_token %}

        <div class="cell medium-7">
            <div class="card cell">
                <div class="card-divider">
                    Quedan: {{ product.count }}
                </div>
                <div class="card-section">
                    <h4>{{ product.name }}</h4>
                    <p>Proveedor: {{ product.provider }}</p>
                    <p>Descripcion: {{ product.description }}</p>
                    <p>Codigo: {{ product.barcode }}</p>
                </div>
            </div>
        </div>

        <div class="cell medium-6">
            <button type="submit" class="alert button">Eliminar</button>
            <a href="{% url 'producto_app:producto-lista' %}" class="primary button clear">Volver</a>
        </div>
    </form>
</div>
```

¿Desea eliminar el producto?

Quedan: 20

Monster

Proveedor: Hansen Beverage

Descripcion: Bebida Energizante

Codigo: 12345678480

Eliminar

Volver

Detalle del Producto - Módulo Almacén

Precio Compra: 2,00

Precio Venta: 3,50

Numero de Ventas: 2

Descripcion: Bebida Energizante

Ventas en los ultimos 30 dias

Fecha	Producto	Ventas
11 de octubre de 2022	Monster	2

Imprimir

La lista de las ventas de los ultimos 30 dias se la hace por medio de el `get_context_data`

```
class ProductDetailView(AlmacenPermisoMixin, DetailView):
    template_name = "producto/detail.html"
    model = Product

    def get_context_data(self, **kwargs):
        context = super().get_context_data(**kwargs)
        #
        context["ventas_mes"] = SaleDetail.objects.ventas_mes_producto(
            self.kwargs['pk']
        )
        return context
```

```
def ventas_mes_producto(self, id_prod):
    # creamos rango de fecha
    end_date = timezone.now()
    start_date = end_date - timedelta(days=30)

    consulta = self.filter(
        sale_anulate=False,
        created_range=(start_date, end_date),
        product_pk=id_prod,
    ).values('sale_date_sale_date', 'product_name').annotate(
        cantidad_vendida=Sum('count'),
    )
    return consulta
```

```
<tbody>
{% for ventas in ventas_mes %}
<tr>
    <td>{{ ventas.sale_date_sale_date }}</td>
    <td>{{ ventas.product_name }}</td>
    <td>{{ ventas.cantidad_vendida }}</td>
</tr>
{% empty %}
<p style="color: ■green;">No hay ventas para este producto</p>
{% endfor %}
</tbody>
</table>
</div>
```

- si queremos restar una fecha en días usamos el `timedelta`
- si queremos el producto creado en un rango de fechas usamos el `created_range`
- si queremos que por ejemplo que en un dia se ha vendido mas de 1 vez un producto pues que se acumule el numero de veces mas no que me liste una tupla nueva `usamos el values`
- si queremos hacer una operacion aritmetica adicional usamos el `annotate`

Fecha	Producto	Ventas
11 de octubre de 2022	Monster	2

Finalmente en el HTML se itera una vez mandado como contexto el resultado de mi manager

Sin embargo al final tengo un botón imprimir el cual me genera un PDF

<https://www.youtube.com/watch?v=N9iQm4N3H8s> clase extra en youtube

PENDIENTE

Generar PDF con Django

[illegible]

Reportes en Módulo Almacén

Para hacer el reporte en este caso de igual manera se trabaja con un ListView pero se mandan varios **parametros**

Por Fecha de Vencimiento:

Proveedor o Marca:

Filtrar

COD	Nombres	Precio Venta	Precio Compra	Stock	Ventas	Vencimiento	Acciones
12345678480	Monster	3,50	2,00	18	2	25 de octubre de 2022	
750521000050	Yogurt Toni	2,30	1,00	35	27	28 de mayo de 2020	
751271025072	Mermelada de fresa Gloria	2,00	1,30	71	33	16 de septiembre de 2021	

```
class FiltrosProductListView(AlmacenPermisoMixin, ListView):
    template_name = "producto/filtros.html"
    context_object_name = 'productos'

    def get_queryset(self):

        queryset = Product.objects.filter(
            keyword=self.request.GET.get("keyword", ''),
            date_start=self.request.GET.get("date_start", ''),
            date_end=self.request.GET.get("date_end", ''),
            provider=self.request.GET.get("provider", ''),
            marca=self.request.GET.get("marca", ''),
            order=self.request.GET.get("order", ''),
        )
        return queryset
```

```
def Filtrar(self, **filters):
    if not filters['date_start']:
        filters['date_start'] = '2020-01-01'

    if not filters['date_end']:
        filters['date_end'] = timezone.now().date() + timedelta(1080)
    #
    consulta = self.filter(
        due_date_range=(filters['date_start'], filters['date_end'])
    ).filter(
        Q(name__icontains=filters['keyword']) | Q(barcode=filters['keyword'])
    ).filter(
        marca_name__icontains=filters['marca'],
        provider_name__icontains=filters['provider'],
    )

    if filters['order'] == 'name':
        return consulta.order_by('name')
    elif filters['order'] == 'stok':
        return consulta.order_by('count')
    elif filters['order'] == 'num':
        return consulta.order_by('-num_sale')
    else:
        return consulta.order_by('-created')
```

```
{% if request.path == request.get_full_path %}
<a href="{{request.get_full_path}}?order=name"> Nombres</a>
{% else %}
<a href="{{request.get_full_path}}?order=name"> Nombres</a>
{% endif %}
</th>
<th>Precio Venta</th>
<th>Precio Compra</th>
<th>
{% if request.path == request.get_full_path %}
<a href="{{request.get_full_path}}?order=stok">Stock</a>
{% else %}
<a href="{{request.get_full_path}}?order=stok">Stock</a>
{% endif %}
</th>
```

En este caso usa el objeto request dentro del HTML para recuperar toda la URL y le anida el parametro

127.0.0.1:8000/producto/reporte/?keyword=o&date_start=&date_end=&provide=&marca=&order=stok

Análisis Pantalla Proceso Venta- Modulo Venta

Al momento de hacer lo que sería el carrito de compras

En éste caso, este codigo de barras buscará un producto y se añadirá la cantidad, a la final se resumen en un createView (POST) ya que agrega el producto a mi modelo Carshop a fin de que mi carrito se recuerde siempre para cada Usuario..

The screenshot shows a web interface for a shopping cart. At the top, there is a form to add items with a label 'COD/CAN:', a text input containing 'Codigo de barras', a numeric input containing '1', and a button labeled 'Agregar'. Below this is a table titled 'Productos en Venta'. The table has five columns: 'COD', 'Nombres', 'Precio (S/)', 'Cantidad', and 'Acciones'. It contains one row with the following data: COD '12345678480', Nombres 'Monster', Precio '3,50', and Cantidad '1'. The 'Acciones' column for this row contains a blue minus button and a red 'x' button. To the right of the table, there is a green box showing 'Total a Cobrar: S/ 3,5'. Below this are three buttons: a blue 'Pagar Todo' button, a green 'Cobrar e Imprimir' button, and a red-outlined 'Limpiar Todo' button. An orange arrow points from the text 'Se hace una consulta a nuestro modelo Carrito' to the table. A green arrow points from the top left to the 'Agregar' button area.

COD	Nombres	Precio (S/)	Cantidad	Acciones
12345678480	Monster	3,50	1	- x

Se hace una consulta a nuestro modelo Carrito

Total a Cobrar: S/ 3,5

Pagar Todo

Cobrar e Imprimir

Limpiar Todo

Acá lo que hago es listar cada producto guardado dentro de mi modelo carrito a fin de que se recuerde el carrito del usuario

Agregar Producto a Carrito de compra - Modulo Venta

Según la lógica que estuvimos analizando, lo que se tiene que hacer es tomar ese código de barras y enviarlo hacia el servidor a través del método Post. Para que? Para que mediante este código recuperemos un producto, en este caso recuperado este producto, recuperemos ese producto y luego lo registremos dentro de nuestro modelo Carrito de compras o Car Shop.

¿Cómo se Resuelve?

Tenemos 3 formas

- CreateView
- FormView => trabaja con un formulario, toma los datos de ese formulario y hace el registro
- Usar la Vista genérica View

No se puede usar el CreateView debido que para el create debemos estar vinculados a un modelo, tampoco el View debido a que necesitamos un formulario donde mediante el código de barras van a ingresar el producto y su cantidad.. En conclusión solo puedo usar el FormView

Recordando que el formview trabaja siempre con el formvalid, dentro del form_valid iría el código de barras y con éso recuperamos el producto y finalmente agregarlo al carrito

Siempre necesitamos sobrescribir el formvalid al trabajar con el formview a fin de que se valide el formulario

```

class VentaForm(forms.Form):
    barcode = forms.CharField(
        required=True,
        widget=forms.TextInput(
            attrs = {
                'placeholder': 'Codigo de barras',
                'class': 'input-group-field',
            }
        )
    )
    count = forms.IntegerField(
        min_value=1,
        widget=forms.NumberInput(
            attrs = {
                'value': '1',
                'class': 'input-group-field',
            }
        )
    )
    #
    def clean_count(self):
        count = self.cleaned_data['count']
        if count < 1:
            raise forms.ValidationError('Ingresa una cantidad mayor a cero')

        return count

```

Lleva código de barras y la cantidad. *se usa los widgets para el diseño..

*se valida que la cantidad sea mayor a 0

Ahora pasando a mi view y a mi form_valid ¿qué sucede si una vez ya está agregado un producto al carrito y mando de nuevo el mismo código? Pues debería simplemente subir la cantidad, más no añadirse otra tupla, entonces para ello uso el get_or_create.. OjO que no estoy usando el update_or_create debido a que si por ejemplo en cantidad pongo que se añada 5 pues esto debe sumarse mas no actualizarse

En el caso del get_or_create si existe lo recupera y si no existe lo crea

```

def form_valid(self, form):
    barcode = form.cleaned_data['barcode']
    count = form.cleaned_data['count']
    obj, created = CarShop.objects.get_or_create(
        barcode=barcode,
        defaults={
            'product': Product.objects.get(barcode=barcode),
            'count': count
        }
    )
    #
    if not created:
        obj.count = obj.count + count
        obj.save()
    return super(AddCarView, self).form_valid(form)

```

Recordar que para este caso el success_url esta puesto '' lo que significa que me redirecciona a la misma página

```

def get_context_data(self, **kwargs):
    context = super().get_context_data(**kwargs)
    context["productos"] = CarShop.objects.all()
    context["total_cobrar"] = CarShop.objects.total_cobrar()
    # formulario para venta con voucher
    context['form_voucher'] = VentaVoucherForm
    return context

```

Notar que con el get_context_data logro listar los productos del carrito

```

context = super().get_context_data(**kwargs)
context["productos"] = CarShop.objects.all()

```

45	{% for producto in productos %}
----	---------------------------------

Calcular Monto Total de la Venta - Módulo Venta

Productos en Venta

COD	Nombres	Precio (S/)	Cantidad	Acciones
750521000050	Yogurt Toni	2,30	2	<div><div>-</div><div>×</div></div>
12345678480	Monster	3,50	1	<div><div>-</div><div>×</div></div>

Total a Cobrar:

S/ 8,1

Pagar Todo

Para calcular el monto total debemos de igual manera debemos mandar un contexto con el valor procesado, se suman/multiplica los valores y generamos el total

```
context["total_cobrar"] = CarShop.objects.total_cobrar()
```

Recordar que si queremos calcular un valor donde vamos a hacer operaciones aritméticas. ¿qué deberíamos hacer? pues un aggregate

```
def total_cobrar(self):
    consulta = self.aggregate(
        total=Sum(
            F('count')*F('product_sale_price'),
            output_field=FloatField()
        ),
    )
    if consulta['total']:
        return consulta['total']
    else:
        return 0
```

F, específicamente sirve para tomar un valor de un c operaciones aritméticas, en cambio Q sirve para real sin en la orm una coma significa la sentencia condi la condicional "or", pero no muy común, por ello se

Eliminar productos del carrito de venta - Modulo Venta



¿Cómo hacemos para que se disminuya la cantidad cada que damos click en el botón azul y se actualize el total?

Pues partamos de que es un updateview ya que estoy actualizando una cantidad de un registro que ya existe

```
class CarShopUpdateView(VentasPermisoMixin, View):
    """ quita en 1 la cantidad en un carshop """

    def post(self, request, *args, **kwargs):
        car = CarShop.objects.get(id=self.kwargs['pk'])
        if car.count > 1:
            car.count = car.count - 1
            car.save()
        #
        return HttpResponseRedirect(
            reverse(
                'venta_app:venta-index' redirecciona
            )
        )
```

Usamos el View en lugar del updateView ya que el updateV necesita formularios para poder trabajar.. El View no, lo que hace es interceptar un método POST y dentro de esto podemos indicar cualquier proceso


```
path(
    'carshop/update/<pk>/',
    views.CarShopUpdateView.as_view(),
    name='carshop-update',
),
```

```
<form action="{% url 'venta_app:carshop-update' producto.id %}" method="POST">{% csrf_token %}
  <button type="submit" class="button primary"><i class="fi-minus"></i></button>
</form>
```

Y de esta forma haría el update, pero cómo hago con el delete?

```
<form action="{% url 'venta_app:carshop-delete' producto.id %}" method="POST">{% csrf_token %}
  <button type="submit" class="button alert"><i class="fi-x"></i></button>
</form>
```

Pues misma historia

```
class CarShopDeleteView(VentasPermisoMixin, DeleteView):
    model = CarShop
    success_url = reverse_lazy('venta_app:venta-index')
```

```
path(
    'carshop/delete/<pk>/',
    views.CarShopDeleteView.as_view(),
    name='carshop-delete',
),
```

Limpiar carrito de Venta - Modulo Venta

COD/CAN: Código de barras 1

Productos en Venta				
COD	Nombres	Precio (\$/)	Cantidad	Acciones
750521000050	Yogurt Toni	2,30	2	<input type="button" value="-"/> <input type="button" value="x"/>
12345678480	Monster	3,50	1	<input type="button" value="-"/> <input type="button" value="x"/>

Total a Cobrar:

S/ 8,1

COD/CAN: Código de barras 1

Productos en Venta				
COD	Nombres	Precio (\$/)	Cantidad	Acciones

Total a Cobrar:

S/ 0

¿Cómo hago para que al dar click en Limpiar Todo me limpie absolutamente todo?

Pues no debo usar el deleteView debido a que debo mandar un pk por ende voy a usar el View


```
class CarShopDeleteAll(VentasPermisoMixin, View):

    def post(self, request, *args, **kwargs):
        #
        CarShop.objects.all().delete()
        #
        return HttpResponseRedirect(
            reverse(
                'venta_app:venta-index'
            )
        )
```

recordar que se intercepta el método POST

```
<form class="cell" action="{% url 'venta_app:carshop-delete_all' %}"
  method="POST">{% csrf_token %}
  <button type="submit" class="cell hollow button alert">
    <i class="fi-trash"></i>
    <span>Limpiar Todo</span>
  </button>
</form>
```

Vista para Venta Simple

 Pagar Todo

Este botón lo que hace es tomar todo lo que está en mi carrito y registrarlo como una venta sin embargo nunca nos pide un comprobante, qué tipo de pago se hará, etc.

Esto es debido a que cuando no se alcanza un monto mínimo en una compra pues no es necesario emitir un comprobante de pago, no imprime ni na...

Esto es muy usual por lo que se recomienda siempre implementarlo

Entonces si estamos pensando en el proceso de registrar pues qué tipo de vista debemos de registrar?

-No tenemos un formulario estrictamente basado en un modelo (VENTA)

Descartado el formview y el createview...

Ojo que no se debe confundir, los inputs del código son para el carrito

Usaremos el VIEW

- DECLARAR LA VISTA VIEW
- TOMAR PRODUCTOS DEL CARRITO Y AÑADIRLOS A VENTA
- ELIMINAMOS TODOS LOS ELEMENTOS DEL CARRITO DE COMPRAS INDICANDO QUE YA SE VENDIÓ TODO

```
class ProcesoVentaSimpleView(VentasPermisoMixin, View):
    """ Procesa una venta simple """

    def post(self, request, *args, **kwargs):
        #
        procesar_venta(
            self=self,
            type_invoice=Sale.SIN_COMPROBANTE,
            type_payment=Sale.CASH,
            user=self.request.user,
        )
        #
        return HttpResponseRedirect(
            reverse(
                'venta_app:venta-index'
            )
        )
    )
```

```
def procesar_venta(self, **params_venta):
    # recupera la lista de productos en carrito
    productos_en_car = CarShop.objects.all()
    if productos_en_car.count() > 0:
        # crea el objeto venta
        venta = Sale.objects.create(
            date_sale=timezone.now(),
            count=0,
            amount=0,
            type_invoice=params_venta['type_invoice'],
            type_payment=params_venta['type_payment'],
            user=params_venta['user'],
        )
        #
        ventas_detalle = []
        productos_en_venta = []
        for producto_car in productos_en_car:
            venta_detalle = SaleDetail(
                product=producto_car.product,
                sale=venta,
                count=producto_car.count,
                price_purchase=producto_car.product.purchase_price,
                price_sale=producto_car.product.sale_price,
                tax=0.18,
            )
            # actualizamos stock de producto en iteracion
            producto = producto_car.product
            producto.count = producto.count - producto_car.count
            producto.num_sale = producto.num_sale + producto_car.count
            #
            ventas_detalle.append(venta_detalle)
            productos_en_venta.append(producto)
            #
            venta.count = venta.count + producto_car.count
            venta.amount = venta.amount + producto_car.count*producto_car.product.sale_price

        venta.save()
        SaleDetail.objects.bulk_create(ventas_detalle)
        # actualizamos el stock
        Product.objects.bulk_update(productos_en_venta, ['count', 'num_sale'])
        # completada la venta, eliminamos productos del carrito
        productos_en_car.delete()
        return venta
    else:
        return None
```

Para este caso lo que hemos realizado y se recomienda mucho es que si una vista tiene mucho código pues es mejor hacer uso de los functions luego simplemente lo llamo

con el self=self hago referencia a la instancia del post y la igualo al parametro self para el metodo procesar_venta, es siempre recomendable mandar el self en funciones de python

Función para el Proceso Venta - Modulo Venta

```
def procesar_venta(self, **params_venta):
    # recupera la lista de productos en carrito
    productos_en_car = CarShop.objects.all()
    if productos_en_car.count() > 0: verifico si está vacío

    # crea el objeto venta
    venta = Sale.objects.create(
        date_sale=timezone.now(),
        count=0,
        amount=0,
        type_invoice=params_venta['type_invoice'],
        type_payment=params_venta['type_payment'],
        user=params_venta['user'],
    )
    # Registro la venta
    # Como si hay productos pues creo mi venta

    ventas_detalle = []
    productos_en_venta = []
    for producto_car in productos_en_car:
        venta_detalle = SaleDetail(
            product=producto_car.product,
            sale=venta,
            count=producto_car.count,
            price_purchase=producto_car.product.purchase_price,
            price_sale=producto_car.product.sale_price,
            tax=0.18,
        )
        # Creo mi detalle a partir de mi carrito

        # actualizmos stok de producto en iteracion
        producto = producto_car.product
        producto.count = producto.count - producto_car.count
        producto.num_sale = producto.num_sale + producto_car.count
        # disminuyo el stock
        # aqui aumento el numero de ventas de ese producto

        ventas_detalle.append(venta_detalle)
        productos_en_venta.append(producto)
        # añadido mi detalle al arreglo de detalles
        # añadido el producto a mi arreglo de productos a vender

        venta.count = venta.count + producto_car.count
        venta.amount = venta.amount + producto_car.count*producto_car.product.sale_price
        # calculo la cantidad total de productos
        # calculo el monto total de productos por cantidad

    venta.save()
    SaleDetail.objects.bulk_create(ventas_detalle)
    # Crea de golpe mis detalles
    # actualizamos el stok
    Product.objects.bulk_update(productos_en_venta, ['count', 'num_sale'])
    # Actualiza de golpe mis productos
    # completada la venta, eliminamos productos del carrito
    productos_en_car.delete()
    # Eliminamos la lista de productos en el carrito
    return venta
else:
    return None
```



