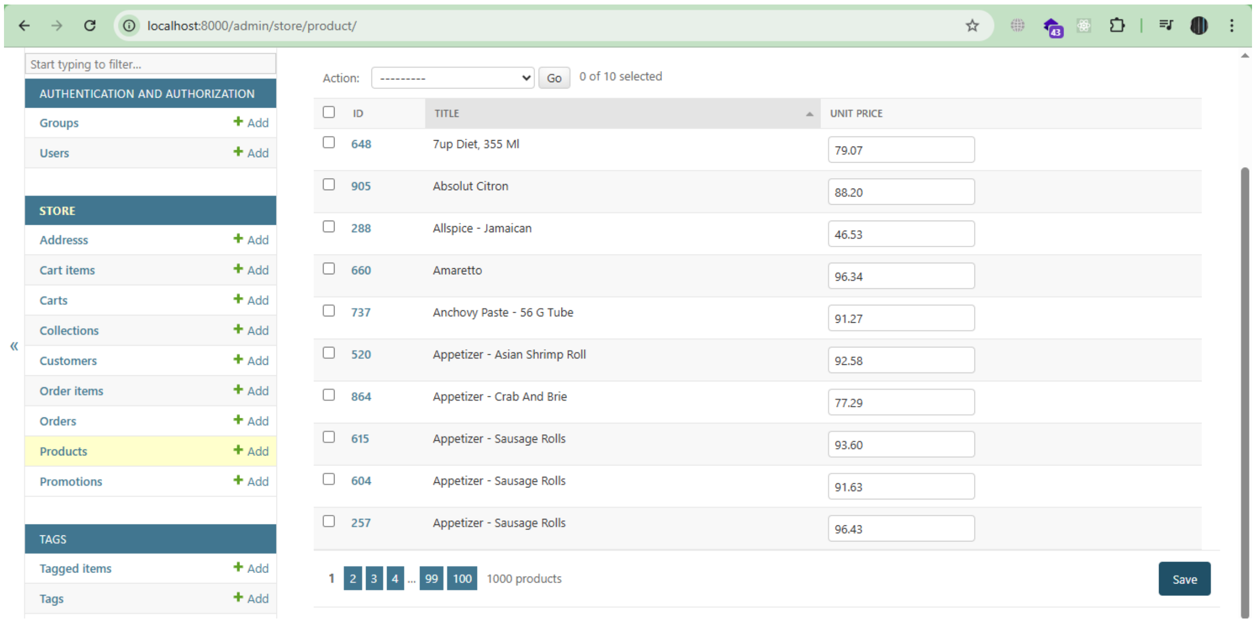
To Change The Number Of Items Per Page For Specific Model:

```
class ProductAdmin(admin.ModelAdmin):
    list_display = [ 'id', 'title', 'unit_price', ]

    list_editable = [ 'unit_price' ]

    list_per_page = 10 # We Use This
```
*******************************************************************



0 of 10 selected

| ID | TITLE | UNIT PRICE |
|---|---|---|
| 648 | 7up Diet, 355 Ml | 79.07 |
| 905 | Absolut Citron | 88.20 |
| 288 | Allspice - Jamaican | 46.53 |
| 660 | Amaretto | 96.34 |
| 737 | Anchovy Paste - 56 G Tube | 91.27 |
| 520 | Appetizer - Asian Shrimp Roll | 92.58 |
| 864 | Appetizer - Crab And Brie | 77.29 |
| 615 | Appetizer - Sausage Rolls | 93.60 |
| 604 | Appetizer - Sausage Rolls | 91.63 |
| 257 | Appetizer - Sausage Rolls | 96.43 |

1 2 3 4 ... 99 100 1000 products

*******************************************************************

To Define Computed Columns:

```
class ProductAdmin(admin.ModelAdmin):
    list_display = [ 'id', 'title', 'unit_price', 'inventory_status']

    list_editable = [ 'unit_price' ]

    list_per_page = 10

    # This Is The Computed Column AS Method
    @admin.display(ordering='inventory')
    def inventory_status(self, product: models.Product):
        if product.inventory < 10:
            return 'Low'

        return 'OK'
```
*******************************************************************

In ModelAdmin-Classes We Can't Use __ To Show Attributes From Related Objects

*************************************************************

To Use The Related Objects With The Current One That Are Displayed To Admin Panel.

First, We Use list_select_realted = ['Objects Are Here']

Then, Define The Method For The Object:

```
class ProductAdmin(admin.ModelAdmin):
    list_display = [ 'id', 'title', 'unit_price', 'inventory_status',
'collection_title']
    list_editable = [ 'unit_price' ]
    list_per_page = 10
    list_select_related = ['collection']

    @admin.display(ordering='inventory')
    def inventory_status(self, product: models.Product):
        if product.inventory < 10:
            return 'Low'

        return 'OK'

    @admin.display(ordering='collection__title')
    def collection_title(self, product: models.Product):
        return product.collection.title
```
*************************************************************

In This Way, We Can Override The Default Query Set Of Specific Model:

```
@admin.register(models.Collection)
class CollectionAdmin(admin.ModelAdmin):
    list_display = [ 'id', 'title', 'products_count' ]
    list_per_page = 5

    @admin.display(ordering='products_count')
    def products_count(self, collection):
        return collection.products_count

    # In This Way We Can Override The Default Query Set
    def get_queryset(self, request):
        return super().get_queryset(request).annotate(
            products_count=Count('product')
        )
```
*************************************************************

To Build URL For Specific End-Point, We Can Use reverse-Function From urls-util:

Note 1: The reverse-Function Parameter Is: admin:app_model_page

- For Product Page In Store App Will Be: admin:store_product_changelist

Note 2: To Create Safe HTML, We Can use format_html-Util

Note 3: To Build The Query Section Of URL, We Can use urlencode-Method.

```python
from django.utils.html import format_html, urlencode

from django.urls import reverse


@admin.register(models.Collection)
class CollectionAdmin(admin.ModelAdmin):
    list_display = [ 'id', 'title', 'products_count' ]

    list_per_page = 5

    @admin.display(ordering='products_count')
    def products_count(self, collection):

        url = (

            reverse('admin:store_product_changelist')
            + '?'
            + urlencode({
                'collection__id': collection.id
            })
        )


        return format_html('<a href="{}">{}</a>', url, collection.products_count)

    # In This Way We Can Override The Default Query Set
    def get_queryset(self, request):
        return super().get_queryset(request).annotate(
            products_count=Count('product')
        )
```
***************************************************************

To Add Search Functionality To Customer Admin Page, We Use *search_fields*:

```
@admin.register(models.Customer)
class CustomerAdmin(admin.ModelAdmin):
    list_display = [ 'first_name', 'last_name', 'membership', 'orders' ]

    list_editable = [ 'membership' ]

    list_per_page = 10

    search_fields = [ 'first_name', 'last_name' ]

    def get_queryset(self, request):
        return super().get_queryset(request).annotate(
            orders=Count('order')
        )

    def orders(self, customer):
        url = (
            reverse('admin:store_order_changelist')
            + '?'
            + urlencode({
                'customer__id': customer.id
            })
        )

        return format_html('<a href="{}">{}</a>', url, customer.orders)
```
******************************************************************

To Add Lookup Fields To Search Fields:

Note (To Remember): This Is Case Sensitive Search

```
search_fields = [ 'first_name__startswith', 'last_name__startswith' ]
```
******************************************************************

To Add Filters, For Specific Admin Model:

```python
class ProductAdmin(admin.ModelAdmin):
    list_display = [ 'id', 'title', 'unit_price', 'inventory_status',
'collection_title']

    list_editable = [ 'unit_price' ]

    list_filter = [ 'collection', 'last_update' ]

    list_per_page = 10

    list_select_related = ['collection']

    @admin.display(ordering='inventory')
    def inventory_status(self, product: models.Product):
        if product.inventory < 10:
            return 'Low'
        return 'OK'

    @admin.display(ordering='collection__title')
    def collection_title(self, product: models.Product):
        return product.collection.title
```
***************************************************************

To Add Custom Filter In Django:

```python
from django.db.models import QuerySet
class InventoryFilter(admin.SimpleListFilter):
    title = "Inventory"
    parameter_name = "Inventory"

    def lookups(self, request, model_admin):
        return [
            ('<10', 'Low'),
            ('>=10', 'OK')
        ]

    def queryset(self, request, queryset: QuerySet):
        if self.value() == '<10':
            return queryset.filter(inventory__lt=10)
        elif self.value() == '>=10':
            return queryset.filter(inventory__gte=10)
```

***************************************************************

And To Register The Custom Filter For Admin Model, We Only Set The Name Of Filter Inside The list_filter-List:

```
class ProductAdmin(admin.ModelAdmin):
    list_display = [ 'id', 'title', 'unit_price', 'inventory_status',
'collection_title']

    list_editable = [ 'unit_price' ]

    list_filter = [ 'collection', 'last_update', InventoryFilter ]

    list_per_page = 10

    list_select_related = ['collection']

    @admin.display(ordering='inventory')
    def inventory_status(self, product: models.Product):
        if product.inventory < 10:
            return 'Low'

        return 'OK'

    @admin.display(ordering='collection__title')
    def collection_title(self, product: models.Product):
        return product.collection.title
```
***************************************************************

To Define Custom Actions For Specific Model, We Need actions-list of ModelAdmin, And We Must Use @admin.action(...):

```
from django.contrib import admin, messages

class ProductAdmin(admin.ModelAdmin):
    actions = ['clear_inventory']

    @admin.action(description="Clear Inventory")
    def clear_inventory(self, request, queryset: QuerySet):
        update_count = queryset.update(inventory=0)
        self.message_user(
            request,
            message=f"{update_count} Has Been Cleared",
            level=messages.SUCCESS
        )
```
***************************************************************

To Populate Fields With Specific Values:

```
class ProductAdmin(admin.ModelAdmin):
    prepopulated_fields = {
        'slug': ['title']
    }
```

*******************************************************************

**Note 1**: The Previous Way, Only Work If We Don't Set Any Value OR Change The Slug Field.

*******************************************************************

To Define Auto Complete Fields (To Avoid Any Problems With Related Relation), We Need:

- First, The ***autocomplete_fields***
- Second, In The Related ModelAdmin-Class, We Need: ***search_fields***

```
class ProductAdmin(admin.ModelAdmin):
    autocomplete_fields = ['collection']

@admin.register(models.Collection)
class CollectionAdmin(admin.ModelAdmin):
    list_display = [ 'id', 'title', 'products_count' ]

    list_per_page = 5

    search_fields = ['title']
```

*******************************************************************

*Note 1*: To Add Nullable Fields For Admin Panel, We Use blank=True

*Note 2*: To Add Validator For Unit price We Can Use Validators Of Django

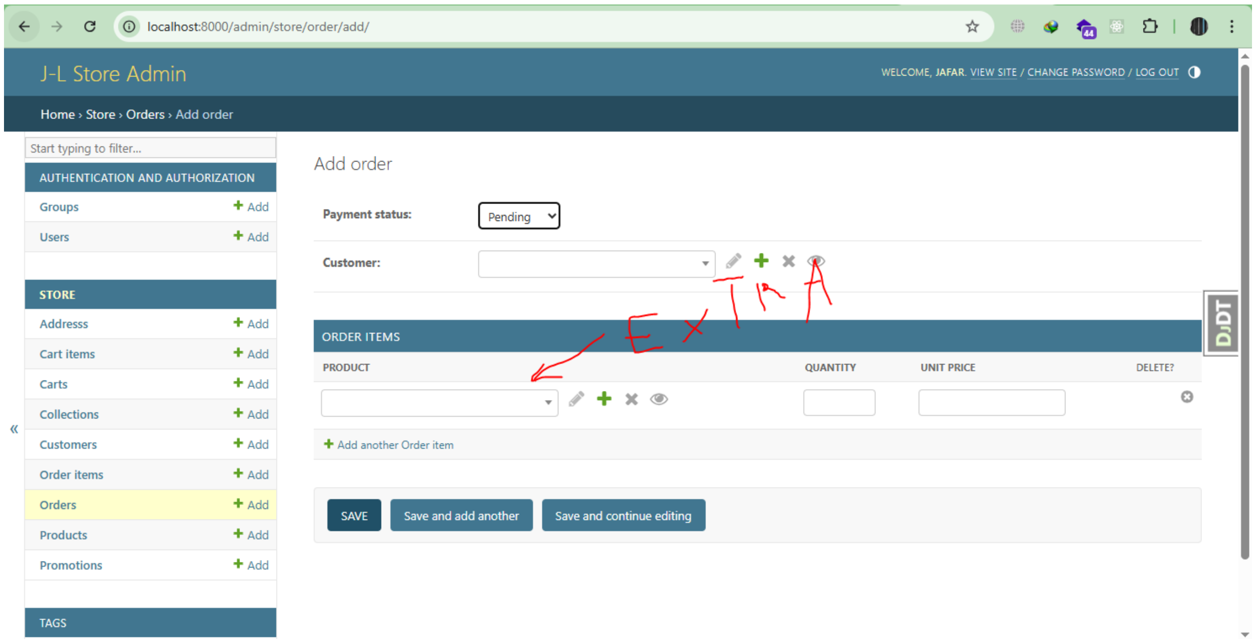
```
from django.core.validators import MinValueValidator
class Product(models.Model):
    title = models.CharField(max_length=255)

    slug  = models.SlugField()

    description = models.TextField(null=True, blank=True)
    unit_price = models.DecimalField(
        max_digits=6,
        decimal_places=2,
        validators=[
            MinValueValidator(limit_value=1, message="Unite Price Must Be Bigger
Than Or Equal 1$")
        ]
    )
    inventory = models.IntegerField()
    last_update = models.DateTimeField(auto_now=True)

    collection = models.ForeignKey(to=Collection, on_delete=models.PROTECT)

    # promotions = models.ManyToManyField(to=Promotion, related_name='products')
    promotions = models.ManyToManyField(to=Promotion)

    def __str__(self):
        return str(self.id) + ' - ' + self.title

    class Meta:
        ordering = ['title']
```

***************************************************************

Note 1: To Add Inline Form For Specific Model With Other One (The Inline One):

Note 2: We Have Also StackedInline That Represent The Children With Form For Each One, Not Row

```
class OrderItemInline(admin.TabularInline):
    autocomplete_fields = ['product']
    model = models.OrderItem

    # The Minimum And Maximum Of OrderItems To Add With Order
    min_num = 1
    max_num = 10

    extra = 1 # The Number Of Extra Fields To Add With Order

@admin.register(models.Order)
class OrderAdmin(admin.ModelAdmin):
    list_display = [ 'id', 'placed_at', 'customer' ]

    list_per_page = 10

    autocomplete_fields = [ 'customer' ]

    inlines = [ OrderItemInline ]
```
*****************************************************************



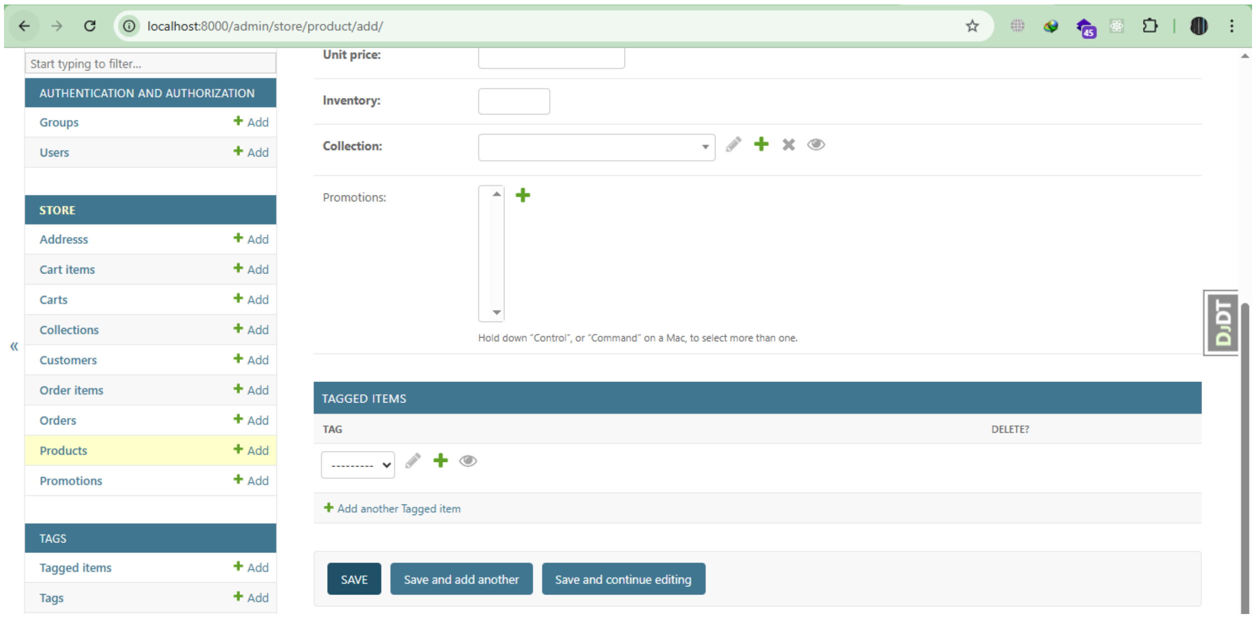
*****************************************************************

To Add Generic Inline Item Like TagItem-Model:

```
admin.site.register(models.Tag)

admin.site.register(models.TaggedItem)
```

Then In Store Admin:

```
from django.contrib.contenttypes.admin import GenericTabularInline


class TagInline(GenericTabularInline):
    model = tags.models.TaggedItem
    min_num = 1
    max_num = 10
    extra = 0

class ProductAdmin(admin.ModelAdmin):
    list_display = [ 'id', 'title', 'unit_price', 'inventory_status',
'collection_title']
    list_editable = [ 'unit_price' ]
    list_filter = [ 'collection', 'last_update', InventoryFilter ]
    list_per_page = 10
    list_select_related = ['collection']
    actions = ['clear_inventory']
    prepopulated_fields = {
        'slug': ['title']
    }
    autocomplete_fields = ['collection']
    search_fields = ['title']
    inlines = [ TagInline ]
```
***********************************************************************



localhost:8000/admin/store/product/add/

Start typing to filter...

AUTHENTICATION AND AUTHORIZATION

| | |
|---|---|
| Groups | + Add |
| Users | + Add |

STORE

| | |
|---|---|
| Addresss | + Add |
| Cart items | + Add |
| Carts | + Add |
| Collections | + Add |
| Customers | + Add |
| Order items | + Add |
| Orders | + Add |
| Products | + Add |
| Promotions | + Add |

TAGS

| | |
|---|---|
| Tagged items | + Add |
| Tags | + Add |

Unit price:

Inventory:

Collection:

Promotions:

Hold down "Control", or "Command" on a Mac, to select more than one.

TAGGED ITEMS

| TAG | DELETE? |
|---|---|
| --------- | |

+ Add another Tagged item

SAVE | Save and add another | Save and continue editing

*****************************************************************