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

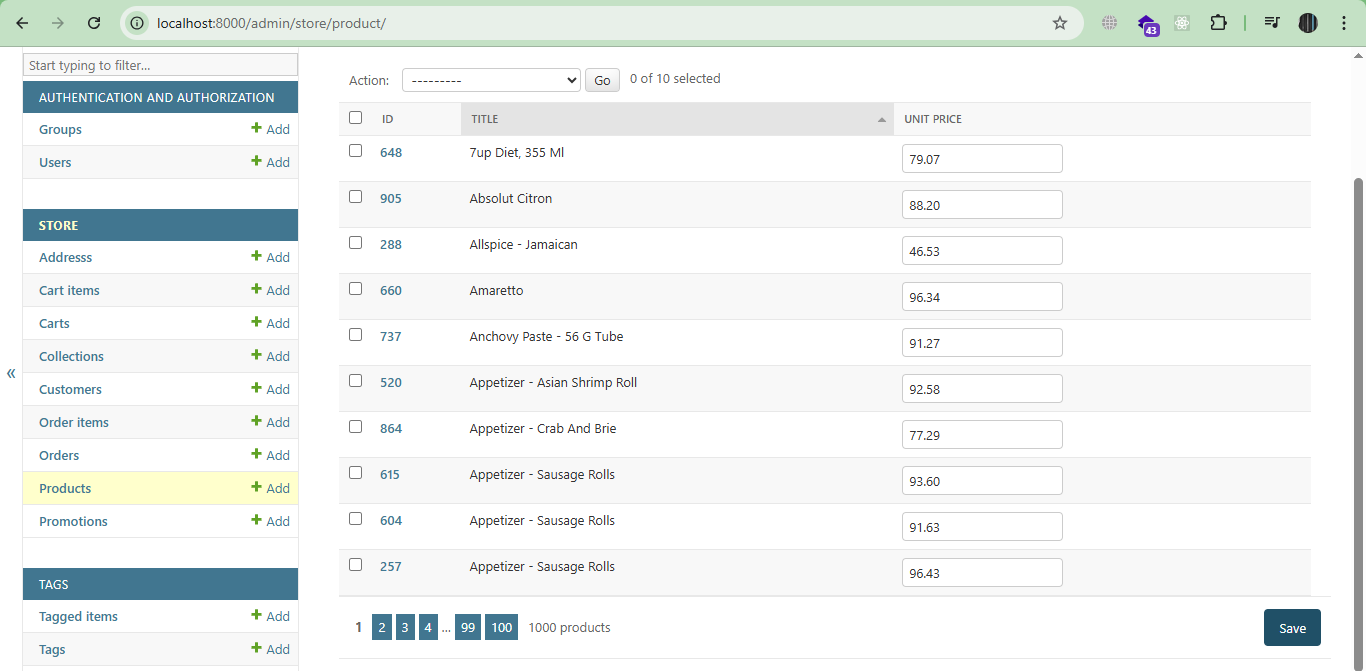
class ProductAdmin(admin.ModelAdmin):

    list\_display = [ 'id', 'title', 'unit\_price', ]

    list\_editable = [ 'unit\_price' ]

    list\_per\_page = 10 # We Use This

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*



\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

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.

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:

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:

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):

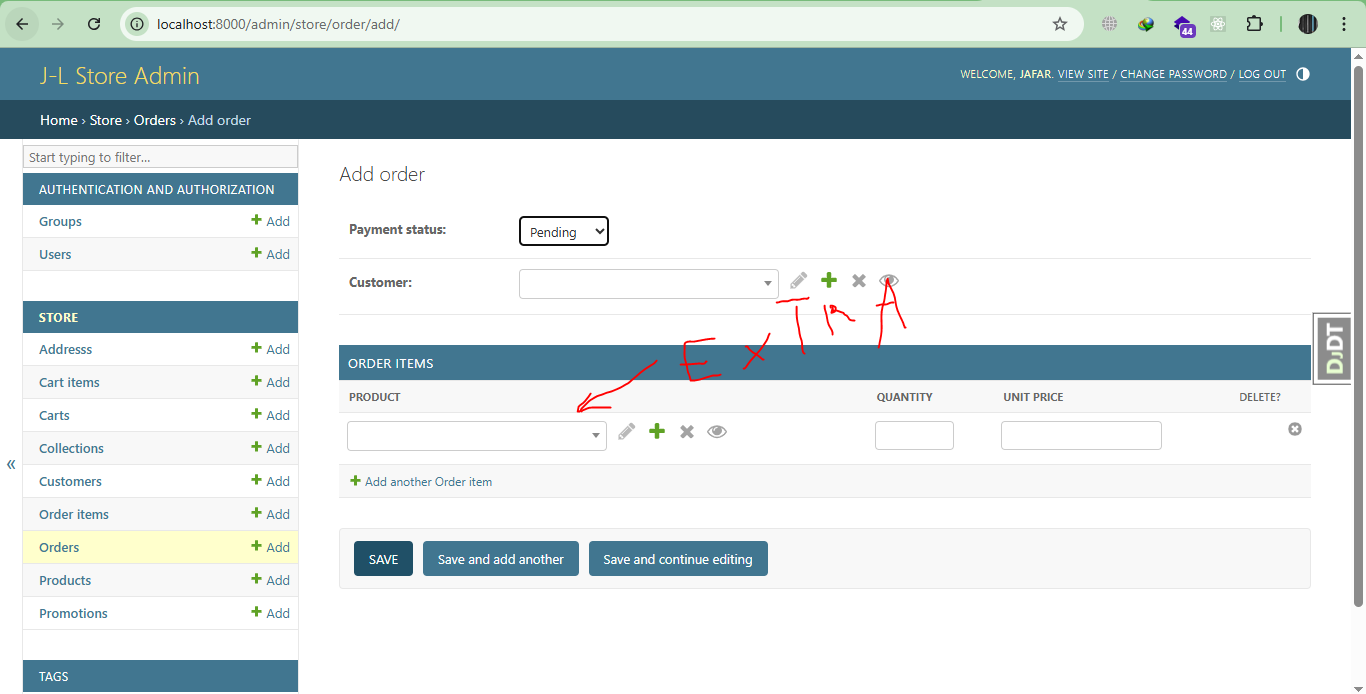
    list\_display = [ 'id', 'placed\_at', 'customer' ]

    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']

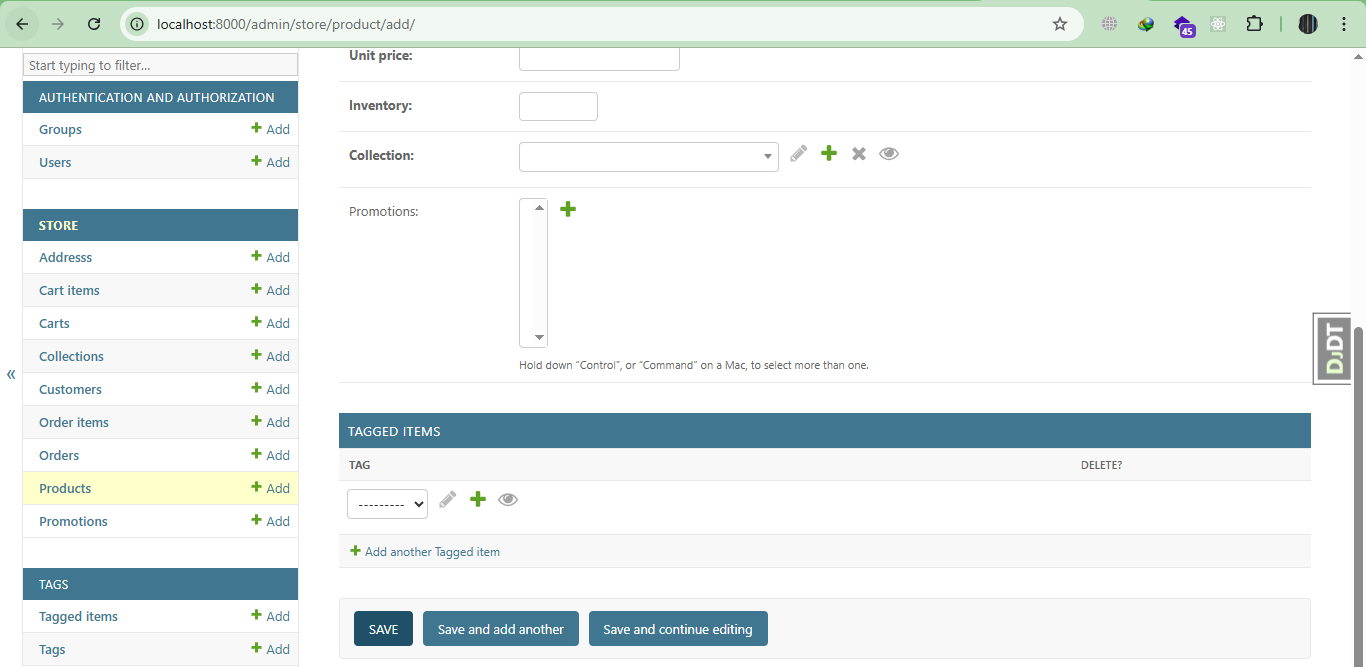
    }

    autocomplete\_fields = ['collection']

    search\_fields = ['title']

    inlines = [ TagInline ]

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*



\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*