# 01- Setting up the admin site

urls.py in the main folder:
admin.site.site_header = 'Storefront Admin'
admin.site.index_title = 'Admin'

# ########################################################################

# 02- Registering Models

# 1) Rewrite __str__ method of model:

```
class Collection(models.Model):
    title = models.CharField(max_length=255)
    featured_product = models.ForeignKey(
        'Product', on_delete=models.SET_NULL, null=True, related_name='+')

    def __str__(self) -> str:
        return self.title

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

# 2) in admin.py of store app:

```
from django.contrib import admin
from . import models
admin.site.register(models.Collection)
admin.site.register(models.Product)
```

########################################################################

# 03- Customizing the List Page

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

admin.site.register(models.Collection)
admin.site.register(models.Product, ProductAdmin)
```

```
# second way(Using decorators)
@admin.register(models.Product)
class ProductAdmin(admin.ModelAdmin):
    list_display = ['title', 'unit_price']
    # edit
    list_editable = ['unit_price']
    # items in list per page
    list_per_page = 10

admin.site.register(models.Collection)

# Exercise:
@admin.register(models.Customer)
class CustomerAdmin(admin.ModelAdmin):
    list_display = ['first_name', 'last_name', 'membership']
    list_editable = ['membership']
    list_per_page = 10
    ordering = ['first_name', 'last_name']
```

##########################################################################

# 04- Adding Computed Columns

```
@admin.register(models.Product)
class ProductAdmin(admin.ModelAdmin):
    @admin.display(ordering='inventory')
    def inventory_status(self, product) -> str:
        if product.inventory < 10:
            return 'Low'
        return 'Ok'

    list_display = ['title', 'unit_price', 'inventory_status']
    # edit
    list_editable = ['unit_price']
    # items in list per page
    list_per_page = 10
```

```
######################################################################

# 05- Selecting Related Object

@admin.register(models.Product)
class ProductAdmin(admin.ModelAdmin):
    @admin.display(ordering='inventory')
    def inventory_status(self, product) -> str:
        if product.inventory < 10:
            return 'Low'
        return 'Ok'

    def collection_title(self, product) -> str:
        return product.collection.title

    list_display = ['title', 'unit_price', 'inventory_status', 'collection_title']
    # edit
    list_editable = ['unit_price']
    # items in list per page
    list_per_page = 10
    # like select_related in ORM
    list_select_related = ['collection']

# Exercise:
# 1) add ordering and change default __str__ in Customer class:
class Customer(models.Model):
    MEMBERSHIP_BRONZE = 'B'
    MEMBERSHIP_SILVER = 'S'
    MEMBERSHIP_GOLD = 'G'

    MEMBERSHIP_CHOICES = [
        (MEMBERSHIP_BRONZE, 'Bronze'),
        (MEMBERSHIP_SILVER, 'Silver'),
```

```
        (MEMBERSHIP_GOLD, 'Gold'),
    ]
    first_name = models.CharField(max_length=255)
    last_name = models.CharField(max_length=255)
    email = models.EmailField(unique=True)
    phone = models.CharField(max_length=255)
    birth_date = models.DateField(null=True)
    membership = models.CharField(
        max_length=1, choices=MEMBERSHIP_CHOICES, default=MEMBERSHIP_BRONZE)

    def __str__(self) -> str:
        return f'{self.first_name} {self.last_name}'

    class Meta:
        ordering = ['first_name', 'last_name']

# 2) Customize admin page:
@admin.register(models.Order)
class OrderAdmin(admin.ModelAdmin):
    def customer_name(self, order) -> str:
        return f'{order.customer.first_name} {order.customer.last_name}'

    list_display = ['id', 'placed_at', 'customer_name']
    list_per_page = 10
    ordering = ['id']
    list_select_related = ['customer']









########################################################################
# 06- Overriding Base QuerySet

@admin.register(models.Collection)
class CollectionAdmin(admin.ModelAdmin):

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

```
    def get_queryset(self, request) -> QuerySet:
        return super().get_queryset(request).annotate(
            products_count=Count('product')
        )

    list_display = ['title', 'products_count']
```

```
######################################################################
# 07- Providing Links To Other Pages

@admin.register(models.Collection)
class CollectionAdmin(admin.ModelAdmin):
    @admin.display(ordering='products_count')
    def products_count(self, collection):
        url = reverse('admin:store_product_changelist') + '?' \
            + urlencode({'collection__id': str(collection.id)})
        return format_html('<a href="{}">{}</a>', url, collection.products_count)

    def get_queryset(self, request) -> QuerySet:
        return super().get_queryset(request).annotate(
            products_count=Count('product')
        )

    list_display = ['title', 'products_count']

# Exercise:
@admin.register(models.Customer)
```

```
class CustomerAdmin(admin.ModelAdmin):
    def get_queryset(self, request) -> QuerySet:
        return super().get_queryset(request).annotate(
            orders_count=Count('order')
        )

    @admin.display(ordering='orders_count')
    def orders_count(self, customer):
        url = reverse('admin:store_order_changelist') + '?' \
            + urlencode({'customer__id': str(customer.id)})
        return format_html('<a href={}>{}</a>', url, customer.orders_count)

    list_display = ['first_name', 'last_name', 'membership', 'orders_count']
    list_editable = ['membership']
    list_per_page = 10
    ordering = ['first_name', 'last_name']
```

```
#####################################################################
# 08- Adding Search To Page List

@admin.register(models.Customer)
class CustomerAdmin(admin.ModelAdmin):
    def get_queryset(self, request) -> QuerySet:
        return super().get_queryset(request).annotate(
            orders_count=Count('order')
        )

    @admin.display(ordering='orders_count')
    def orders_count(self, customer):
        url = reverse('admin:store_order_changelist') + '?' \
            + urlencode({'customer__id': str(customer.id)})
        return format_html('<a href={}>{}</a>', url, customer.orders_count)

    list_display = ['first_name', 'last_name', 'membership', 'orders_count']
    list_editable = ['membership']
    list_per_page = 10
    ordering = ['first_name', 'last_name']
    search_fields = ['first_name__istartswith', 'last_name__istartswith']
```

```
#####################################################################
# 09-Adding Filtering to the List Page

class InventoryFilter(admin.SimpleListFilter):
    title = 'inventory'
    parameter_name = 'inventory'

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

    def queryset(self, request, queryset: QuerySet):
        if self.value() == '< 10':
            queryset.filter(inventory__lt=10)


@admin.register(models.Product)
class ProductAdmin(admin.ModelAdmin):
    @admin.display(ordering='inventory')
    def inventory_status(self, product) -> str:
        if product.inventory < 10:
            return 'Low'
        return 'Ok'

    def collection_title(self, product) -> str:
        return product.collection.title

    list_display = ['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']


#####################################################################
# 10- Creating Custom Actions

@admin.register(models.Product)
class ProductAdmin(admin.ModelAdmin):
    @admin.display(ordering='inventory')
    def inventory_status(self, product) -> str:
        if product.inventory < 10:
```

```
            return 'Low'
        return 'Ok'

    @admin.action(description='Clear inventory')
    def clear_inventory(self, request, queryset):
        update_count = queryset.update(inventory=0)
        self.message_user(
            request,
            f'{update_count} products were successfully updated.',
            messages.SUCCESS
        )

    def collection_title(self, product) -> str:
        return product.collection.title

    actions = ['clear_inventory']
    list_display = ['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']




########################################################################
# 11- Customizing Forms

@admin.register(models.Product)
class ProductAdmin(admin.ModelAdmin):
    @admin.display(ordering='inventory')
    def inventory_status(self, product) -> str:
        if product.inventory < 10:
            return 'Low'
        return 'Ok'

    @admin.action(description='Clear inventory')
    def clear_inventory(self, request, queryset):
        update_count = queryset.update(inventory=0)
        self.message_user(
            request,
            f'{update_count} products were successfully updated.',
            messages.SUCCESS
        )
```

```
    def collection_title(self, product) -> str:
        return product.collection.title

    # For Forms
    # fields = ['title', 'slug']
    # exclude = ['description']
    # readonly_fields = ['title']
    prepopulated_fields = {'slug': ['title']}
    autocomplete_fields = ['collection']
    # and add search_fields = ['title'] in Collection

    actions = ['clear_inventory']
    list_display = ['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']

# Exercise:
@admin.register(models.Order)
class OrderAdmin(admin.ModelAdmin):
    def customer_name(self, order) -> str:
        return f'{order.customer.first_name} {order.customer.last_name}'

    list_display = ['id', 'placed_at', 'customer_name']
    list_per_page = 10
    ordering = ['id']
    list_select_related = ['customer']
    autocomplete_fields = ['customer']
```

```
######################################################################
# 12- Adding Data Validation
# In models.py:
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(1)])
    inventory = models.IntegerField()
    last_update = models.DateTimeField(auto_now=True)
    collection = models.ForeignKey(Collection, on_delete=models.PROTECT)
    promotions = models.ManyToManyField(Promotion, blank=True)

    def __str__(self) -> str:
        return self.title

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

```
######################################################################
# 13- Edit Children Using Inlines

class OrderItemInline(admin.TabularInline):
# class OrderitemInline(admin.StackedInline):
    min_num = 1
```

```
    max_num = 10
    autocomplete_fields = ['product']
    model = models.OrderItem
    extra = 0


@admin.register(models.Order)
class OrderAdmin(admin.ModelAdmin):
    def customer_name(self, order) -> str:
        return f'{order.customer.first_name} {order.customer.last_name}'

    inlines = [OrderItemInline]
    list_display = ['id', 'placed_at', 'customer_name']
    list_per_page = 10
    ordering = ['id']
    list_select_related = ['customer']
    autocomplete_fields = ['customer']









######################################################################
# 14- Using Generic Relations

store app:
class TagInline(GenericTabularInline):
    autocomplete_fields = ['tag']
    model = TaggedItem
    min_num = 1
    max_num = 4
    extra = 0

@admin.register(models.Product)
class ProductAdmin(admin.ModelAdmin):
    @admin.display(ordering='inventory')
    def inventory_status(self, product) -> str:
        if product.inventory < 10:
            return 'Low'
```

```
        return 'Ok'

    @admin.action(description='Clear inventory')
    def clear_inventory(self, request, queryset):
        update_count = queryset.update(inventory=0)
        self.message_user(
            request,
            f'{update_count} products were successfully updated.',
            messages.SUCCESS
        )

    def collection_title(self, product) -> str:
        return product.collection.title

    # For Forms
    # fields = ['title', 'slug']
    # exclude = ['description']
    # readonly_fields = ['title']
    prepopulated_fields = {'slug': ['title']}
    autocomplete_fields = ['collection']
    # and add search_fields = ['title'] in Collection
    inlines = [TagInline]

    search_fields = ['title__istartswith']
    actions = ['clear_inventory']
    list_display = ['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']
```

tag app:

```
@admin.register(models.Tag)
class TagAdmin(admin.ModelAdmin):
    search_fields = ['label']
```

######################################################################
# 15- Extending Pluggable Apps

For Decouple app we should create new app like below:

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

from store.models import Product
from store.admin import ProductAdmin
from tags.models import TaggedItem


class TagInline(GenericTabularInline):
    autocomplete_fields = ['tag']
    model = TaggedItem
    min_num = 1
    max_num = 4
    extra = 0


class CustomProductAdmin(ProductAdmin):
    inlines = [TagInline]


admin.site.unregister(Product)
admin.site.register(Product, CustomProductAdmin)
```