#customizing admin site

customizing Django’s automatically-generated admin site

#Customize the admin form
By registering the Question model with admin.site.register(Question), Django was able to construct a default form representation

to change the admin options for a model,
create a model admin class, then pass it as the second argument to admin.site.register()

Replace the admin.site.register(Question) with,

```
class QuestionAdmin(admin.ModelAdmin):
    fields = ['pub_date', 'question_text']

admin.site.register(Question, QuestionAdmin)
```

- it makes the “Publication date” come before the “Question” field in admin panel

Can split the form using fields as follows,

```
class QuestionAdmin(admin.ModelAdmin):
    fieldsets = [
        (None,               {'fields': ['question_text']}),
        ('Date information', {'fields': ['pub_date']}),
    ]

admin.site.register(Question, QuestionAdmin)
```

The first element of each tuple in fieldsets is the title of the fieldset.

#Adding related objects

#Customize the admin change list

“change list” page – the one that displays all the questions in the system (all values in table)

```
list_display = ('question_text', 'pub_date', 'was_published_recently')      # display the respective fields
list_filter = ['pub_date']                                 # add filter option in this model
search_fields = ['question_text']                          # add search option
```

display() decorator on that method (in polls/models.py) to improve

change lists give free pagination, default is to display 100 items per page

In Change list - pagination, search boxes, filters, date-hierarchies, and column-header-ordering all work together

#pagination
https://docs.djangoproject.com/en/4.1/ref/contrib/admin/#django.contrib.admin.ModelAdmin.list_per_page

ModelAdmin.list_per_page

> Set list_per_page to control how many items appear on each paginated admin change list page. By default, this is set to 100.

#search boxes
https://docs.djangoproject.com/en/4.1/ref/contrib/admin/#django.contrib.admin.ModelAdmin.search_fields

ModelAdmin.search_fields

```
    search_fields = ['foreign_key__related_fieldname']
    search_fields = ['user__email']
```

#filters
https://docs.djangoproject.com/en/4.1/ref/contrib/admin/#django.contrib.admin.ModelAdmin.list_filter

https://docs.djangoproject.com/en/4.1/ref/contrib/admin/filters/#modeladmin-list-filters

ModelAdmin.list_filter

> Set list_filter to activate filters in the right sidebar of the change list page of the admin

At it’s simplest list_filter takes a list or tuple of field names to activate filtering upon, but several more advanced options as available. See ModelAdmin List Filters for the details.

#date-hierarchies
https://docs.djangoproject.com/en/4.1/ref/contrib/admin/#django.contrib.admin.ModelAdmin.date_hierarchy

```
    date_hierarchy = 'pub_date'
    date_hierarchy = 'author__pub_date'
```

#column-header-ordering

https://docs.djangoproject.com/en/4.1/ref/contrib/admin/#django.contrib.admin.ModelAdmin.list_display

```
	list_display = ('first_name', 'last_name')
```

# Customize the admin look and feel
 using Django’s template system can customize django admin panel

# Customizing your project’s templates
Create a templates directory within project directory (the one that contains manage.py)

templates can live anywhere on a filesystem that Django can access. (Django runs as whatever user your server runs.) However, keeping your templates within the project is a good convention to follow.

in settings file (mysite/settings.py) and add a DIRS option in the TEMPLATES setting:

```
	'DIRS': [BASE_DIR / 'templates'],
```

create a directory called admin inside templates, and copy the template admin/base_site.html (django/contrib/admin/templates)

#Django source files

```
	python -c "import django; print(django.__path__)"
```

edit the file and replace {{ site_header|default:_('Django administration') }} (including the curly braces) with your own site’s name as you see fit

#ref
https://docs.djangoproject.com/en/4.1/intro/tutorial07/