Pecateni materijali DNICK

CONCEPT for the start of the project creation:

1. Create the application – *python manage.py startapp myapp*
2. We create the static folder and we add the following lines of code inside **setting.py**

STATIC\_URL = "static/"

STATICFILES\_DIRS = [BASE\_DIR / "static"]

MEDIA\_URL = '/media/'

MEDIA\_ROOT = BASE\_DIR / 'media'

1. Inside the **static directory** we add the bootstrap files and we import them with the following links in every html file

**<link rel="stylesheet" href="{% static 'bootstrap/css/bootstrap.css' %}"/>**

**<script src="{% static 'bootstrap/js/bootstrap.bundle.js' %}"></script>**

1. Inside INSTALLED\_APPS = [..], we add the name of the started application so we can register the app.
2. In **urls.py**, we add the media line of code and the imports

***from django.conf.urls.static import static  
from django.conf import settings***

***urlpatterns=[] + static(settings.MEDIA\_URL, document\_root=settings.MEDIA\_ROOT)***

**Admin panel**

For things in the admin panel search for ‘***The Django admin site’*** in the documentation

from django.contrib import admin

class CustomModelAdmin(admin.ModelAdmin):

def **save\_model**(self, request, obj, form, change):

"""Automatically set the user who created the object (only on creation)."""

if not change or not obj.pk:

obj.user\_that\_created = request.user

super().save\_model(request, obj, form, change)

def **has\_add\_permission**(self, request):

"""Control who can add objects."""

return request.user.is\_authenticated # or any custom condition

def **has\_change\_permission**(self, request, obj=None):

"""Only allow change if the user is the creator or is a superuser."""

if request.user.is\_superuser:

return True

if obj is None:

return True # This controls access to the change list page

return obj.user\_that\_created == request.user

def **has\_delete\_permission**(self, request, obj=None):

"""Only allow delete for superusers."""

return request.user.is\_superuser

**Queryset**

def get\_queryset 🡺 retruns a list of objects that will be used to be shown in the view

For example in the following code, we see

*class PilotAdmin(admin.ModelAdmin):*

*def get\_queryset(self, request):*

*qs = super().get\_queryset(request) #it returns all the elements*

*if request.user.is\_superuser:*

*return qs #if the user is superuser, return all of them*

*return qs.filter(user\_that\_created=request.user) #if not, return the ones he created*

The following code is a short version of it

class MyModelAdmin(admin.ModelAdmin):

def get\_queryset(self, request):

return super().get\_queryset(request).filter(user=request.user)

**Signals**

**First of all, I need to register the signals, inside apps.py**

***def ready(self):  
 import myapp.signals***

We have 4 types of signals

1. **pre\_save signal** – *Use when you want to enforce or adjust values before saving to the database.*

@receiver(pre\_save, sender=Pilot)

def assign\_rank\_before\_save(sender, instance, \*\*kwargs):

if instance.total\_hours\_on\_flight > 1000:

instance.rank = 'Expert'

elif instance.total\_hours\_on\_flight > 250:

instance.rank = 'Intermediate'

else:

instance.rank = 'Beginner'

1. **post\_save signal** - *Use when you want to perform side effects like creating logs, sending emails, or related records.*

@receiver(post\_save, sender=Flight)

def create\_flight\_report(sender, instance, created, \*\*kwargs):

if created:

FlightReport.objects.create(

flight\_instance=instance,

description=f"Flight from {instance.departure\_airport} created”

{

)

1. **pre\_delete signal** - *Use when you need to move or protect data before deletion.*

@receiver(pre\_delete, sender=FlightCompany)

def reassign\_pilots(sender, instance, \*\*kwargs):

for rel in AirCompanyPilot.objects.filter(company=instance):

new\_company = FlightCompany.objects.exclude(id=instance.id).first()

if new\_company:

rel.company = new\_company

rel.save()

1. **post\_delete signal** - *Use when you want to log or clean up external systems/files after deletion.*

@receiver(post\_delete, sender=FlightCompany)

def log\_company\_deletion(sender, instance, \*\*kwargs):

FlightCompanyLog.objects.create(

name\_of\_company=instance.company\_name,

description=f"{instance.company\_name} was deleted."

)

**Models**

* **Choices in models**

PILOT\_RANK\_CHOICES = [  
 ("B","Beginner"),  
 ("I","Intermediate"),  
 ("E","Expert"),  
]

rank\_in\_company = models.CharField(max\_length=key\_count, choices=PILOT\_RANK\_CHOICES)

**Queries**

*Q function*

Example 1:

from django.db.models import Q

#OR condition: Books by either Tolkien OR Rowling

books = Book.objects.filter(Q(author="Tolkien") | Q(author="Rowling"))

#AND condition: Books by Tolkien AND with more than 200 pages

books = Book.objects.filter(Q(author="Tolkien") & Q(pages\_\_gt=200))

#NOT condition: Books that are NOT in the Horror genre

books = Book.objects.filter(~Q(genre="Horror"))

#Combined logic: Fantasy books with at least 300 pages

books = Book.objects.filter(Q(genre="Fantasy") & ~Q(pages\_\_lt=300))

Example 2:

Poll**.**objects**.**get(

Q(question\_\_startswith**=**"Who"),

Q(pub\_date**=**date(2005, 5, 2)) **|** Q(pub\_date**=**date(2005, 5, 6)),

)

Analogy to -- **SELECT** **\*** **from** polls **WHERE** question **LIKE** 'Who%' **AND** (pub\_date **=** '2005-05-02' **OR** pub\_date **=** '2005-05-06')

Forms:

GET AND POST FORM, FOR SAVING

def add\_cake(request):

if **request.method == 'POST':**

**#POST PART**

form = CakeForm(request.POST, request.FILES)

if form.is\_valid():

cake = form.save(commit=False)#here we first save the form, but not commit

cake.baker = Baker.objects.filter(user=request.user).first()#we set one attribute to the obj

cake.save()#then we commit to database

return redirect('index')

**#GET PART**

form = CakeForm()

return render(request, 'add\_cake.html', context={'form': form})