Crear repositorio remoto

git clone (con ssh)

crear rama de configuración: **git branch Configuracion_inicial**

git checkout Configuracion_inicial

crear carpeta app (entrar)

crear .gitignore y requirements.txt - con touch

__pycache__/
***.pyc**
***.pyo**
***.pyd**
***.env**
.env
.venv/
venv/
env/

db.sqlite3

/staticfiles/
/media/
***.log**

.vscode/
.idea/
.DS_Store

—--

```
Django~=5.1.1
django-seed~=0.3.1
psycopg2-binary~=2.9.9
django-debug-toolbar~=4.4.6
faker
```

crear entorno: **python3 -m venv myvenv**

activar entorno: **source myvenv/bin/activate**

Instalo Django: **python -m pip install --upgrade pip**

instalo los requerimientos: **pip install -r requirements.txt**

Crear proyecto: **django-admin startproject mysite .**

Configuro dentro de `mysite/settings.py`

Realizamos los settings:

```
TIME_ZONE = 'Europe/Madrid'
```

Europe/Madrid

```
LANGUAGE_CODE = 'es'
```

es

```
USE_TZ = True
```

Poner en false: **USE_TZ = False**

Nos aseguramos de que esto esté así:

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.sqlite3',
        'NAME': BASE_DIR / 'db.sqlite3',
    }
}
```

```
ALLOWED_HOSTS = ['127.0.0.1', 'localhost']
```

‘127.0.0.1’, ‘localhost’

Creamos la base de datos: **python manage.py migrate**

Creo la aplicación con el nombre de protectora: **python manage.py startapp hotel**

Añado en settings, en el apartado de apps instaladas el nombre de la app que acabo de crear:

```
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'protectora'
]
```

git add .

git commit -m "Configuración inicial"

git push origin Configuracion_inicial

git checkout main

git merge Configuracion_inicial

git push

Ahora creamos la RAMA Modelos: **git branch modelos**

git checkout modelos

Creamos los modelos:

```
from django.db import models
from django.core.validators import MinValueValidator, RegexValidator
import uuid

class Hotel(models.Model):
    nombre = models.CharField(max_length=150, unique=True)
    descripcion = models.TextField()
    direccion = models.CharField(max_length=200)
    fecha_fundacion = models.DateField(null=True, blank=True)
    calificacion = models.DecimalField(max_digits=3, decimal_places=2,
default=0.00)
    servicios = models.ManyToManyField('Servicio', blank=True,
related_name='hoteles')

class ContactoHotel(models.Model):
    hotel = models.OneToOneField(Hotel, on_delete=models.CASCADE,
related_name='contacto')
    nombre_contacto = models.CharField(max_length=100)
    telefono = models.CharField(max_length=20,
validators=[RegexValidator(r'^\+?\d{7,15}$')])
    correo = models.EmailField(unique=True)
    sitio_web = models.URLField(null=True, blank=True)

class TipoHabitacion(models.Model):
    nombre = models.CharField(max_length=100)
    descripcion = models.TextField(blank=True)
```

```
    capacidad = models.IntegerField(validators=[MinValueValidator(1)])
    precio_base = models.DecimalField(max_digits=8, decimal_places=2)

class Habitacion(models.Model):
    numero = models.CharField(max_length=10)
    piso = models.IntegerField()
    tipo = models.ForeignKey(TipoHabitacion, on_delete=models.PROTECT,
related_name='habitaciones')
    hotel = models.ForeignKey(Hotel, on_delete=models.CASCADE,
related_name='habitaciones')
    disponible = models.BooleanField(default=True)
    foto = models.ImageField(upload_to='habitaciones/', null=True,
blank=True)
    servicios = models.ManyToManyField('Servicio', blank=True,
related_name='habitaciones')

class Huesped(models.Model):
    nombre = models.CharField(max_length=80)
    apellido = models.CharField(max_length=80)
    correo = models.EmailField(unique=True)
    telefono = models.CharField(max_length=20, blank=True)
    fecha_nacimiento = models.DateField(null=True, blank=True)

class PerfilHuesped(models.Model):
    huesped = models.OneToOneField(Huesped, on_delete=models.CASCADE,
related_name='perfil')
    nacionalidad = models.CharField(max_length=50)
    numero_pasaporte = models.CharField(max_length=30, null=True, blank=True,
unique=True)
    puntos_fidelidad = models.IntegerField(default=0)
    preferencias = models.TextField(blank=True)

class Servicio(models.Model):
    nombre = models.CharField(max_length=100)
    descripcion = models.TextField(blank=True)
    precio = models.DecimalField(max_digits=8, decimal_places=2)
    es_opcional = models.BooleanField(default=True)
    duracion_minutos = models.IntegerField(null=True, blank=True)

class Reserva(models.Model):
    ESTADOS = [
        ('P', 'Pendiente'),
        ('C', 'Confirmada'),
```

```
        ('F', 'Finalizada'),
        ('X', 'Cancelada'),
    ]
    huesped = models.ForeignKey(Huesped, on_delete=models.CASCADE,
related_name='reservas')
    habitacion = models.ForeignKey(Habitacion, on_delete=models.PROTECT,
related_name='reservas')
    fecha_entrada = models.DateField()
    fecha_salida = models.DateField()
    estado = models.CharField(max_length=1, choices=ESTADOS, default='P')
    creada_en = models.DateTimeField(auto_now_add=True)
    servicios = models.ManyToManyField(Servicio, through='ReservaServicio',
blank=True, related_name='reservas')

class Factura(models.Model):
    reserva = models.OneToOneField(Reserva, on_delete=models.CASCADE,
related_name='factura')
    emitida_en = models.DateTimeField(auto_now_add=True)
    numero_factura = models.CharField(max_length=30, unique=True,
default=uuid.uuid4)
    monto_total = models.DecimalField(max_digits=10, decimal_places=2)
    pagada = models.BooleanField(default=False)
    notas = models.TextField(blank=True)

class ReservaServicio(models.Model):
    reserva = models.ForeignKey(Reserva, on_delete=models.CASCADE)
    servicio = models.ForeignKey(Servicio, on_delete=models.CASCADE)
    cantidad = models.IntegerField(default=1,
validators=[MinValueValidator(1)])
    precio_en_momento = models.DecimalField(max_digits=8, decimal_places=2)
    nota = models.CharField(max_length=200, blank=True)
```

Creo superusuario para después loguearse : **python manage.py createsuperuser**

En admin añadimos los registros para que el admin pueda acceder animales, protectora, y colaborador y pueda crearle registros a esas clases.

```python
from django.contrib import admin
from .models import Animal, Protectora, Colaborador

# Register your models here.

admin.site.register(Animal)
admin.site.register(Protectora)
admin.site.register(Colaborador)
```

from .models import
admin.site.register()

Migraciones y registros en la base de datos: **python manage.py makemigrations hotel**

python manage.py migrate

Ejecutamos el servidor: **python manage.py runserver**

git add .

git commit -m "Modelos"

git push origin modelos

git checkout main

git merge modelos

git push

Creamos la nueva rama “Backups”: **git branch Backups**

git checkout Backups

Ahora vamos a rellenar las tablas con **SEED**

pip install -r requirements.txt
En INSTALLED APPS incluimos: `'django_seed',`

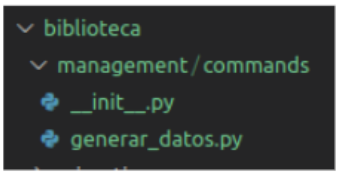
python manage.py seed <nombre_app> --number=10

Ahora vamos a rellenar las tablas con **FAKER**

pip install --upgrade pip

pip install -r requirements.txt

Creamos las siguientes carpetas:



Importante que los archivos **__init__.py** y **generar_datos.py** están dentro de la carpeta **commands**.

```
# region Explicación de los imports.
# ----------------------------------------------------------
# BaseCommand: permite crear comandos personalizados de Django.
# Faker: genera datos falsos (en español si se usa Faker('es_ES')).
# random: funciones aleatorias (choice, sample, randint...).
# Decimal: maneja decimales con precisión (para puntuaciones).
# timedelta: suma/resta días a fechas.
# timezone: obtiene fecha/hora actual con soporte de zona horaria.
# Modelos: importamos las clases necesarias del app Concursos_Online.
# ----------------------------------------------------------
# endregion

from django.core.management.base import BaseCommand
from faker import Faker
import random
from decimal import Decimal
from datetime import timedelta
from django.utils import timezone
from Concursos_Online.models import *
```

```python
class Command(BaseCommand):
   help = 'Generando datos usando Faker'

   def handle(self, *args, **kwargs):
       fake = Faker('es_ES')  # Faker en español

      # region Creación de Usuarios
      # ------------------------------------------------------------
      # 1. Se crea una lista vacía `usuarios` para guardar las instancias creadas.
      # 2. Se generan 30 usuarios falsos con datos de Faker:
      #    - nombre_usuario: nombre de usuario único.
      #    - correo: email único.
      #    - password: contraseña aleatoria de 10 caracteres.
      # 3. Cada usuario creado se guarda en la lista `usuarios` para usarlos después.
      # ------------------------------------------------------------

       self.stdout.write("Generando usuarios...")
       usuarios = []
       for _ in range(30):
           usuarios.append(Usuario.objects.create(
               nombre_usuario=fake.unique.user_name(),
               correo=fake.unique.email(),
               password=fake.password(length=10)
           ))

       # endregion
```

.

cuando rellenes todo lo del archivo generar datos haces:

Lo primero es crear una carpeta fixtures dentro de nuestra aplicación.

python manage.py dumpdata --indent 4 > hotel/fixtures/datos.json

De esta forma guardamos en un archivo todos los datos de nuestra base de datos en .json

A continuación si borramos la base de datos y lanzamos los siguientes comandos:

python manage.py migrate

python manage.py loaddata hotel/fixtures/datos.json

.

Crear un backup de los datos con fixture.

Cuando realices los fixtures haces:

git add .

git commit -m "Backups"

git push origin Backups

git checkout main

git merge Backups

git push

—----------

En el README.MD debe especificarse en que consiste cada modelo, cada atributo y cada parámetro usado. Y el esquema de modelo entidad-relación.

Crear el modelo entidad-relación de la base de datos.

Creación Aplicación Web Parte II - URls, Views y QuerySet

creamos la rama Tarea_url:

```
git branch Tarea_url
git checkout Tarea_url
```

Ahora vamos a crear la URL del index.html antes de comenzar a hacer las 10 url

En la carpeta de nuestra aplicación (*Concursos_Online*, en mi caso)
creamos el archivo ***urls.py*** y incluimos lo siguiente:

```
from django.urls import path
from . import views

urlpatterns = [
   path('',views.index, name='index')
]
```

dentro de ***mysite/urls.py*** le indicamos que existe el archivo urls de nuestra aplicación y que lo use

```
from django.contrib import admin
from django.urls import path, include

urlpatterns = [
    path('admin/', admin.site.urls),
    path('',include('Concursos_Online.urls')),
]
```

en el archivo **Concursos_Online/views.py** incluimos el método de index

```
# Create your views here.

def index(request):
    return render(request, 'Concursos_Online/index.html')
```

Creamos la carpeta **templates** dentro de **Concursos_Online** (Carpeta de la aplicacion)
Creamos la carpeta **Concursos_Online**, dentro de templates
Creamos el archivo **index.html**

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Menu Principal</title>
</head>
<body>
    <h1>Gestion de concursos Online</h1>
    <br>
    <h3>Listado de enlaces</h3>
    <ul>
        <li></li>
    </ul>
</body>
</html>
```

ahora verificamos que funciona con:

`source myvenv/bin/activate` → activar el entorno

`python manage.py runserver`

Guardamos los cambios en github

```
git add .
```

```
git commit -m "crear index"
```

```
git push --set-upstream origin Tarea_url
```

Ahora creamos la rama para la url_1

```
git branch url_1
```

```
git checkout url_1
```

Una url que me muestre todos los < *tu modelo principal* > y sus datos, incluido los relacionados

Concursos_Online/urls.py

```
path('concursos-online/listar', views.concursos_listar, name='lista_concursos')
```

url , función de la vista , nombre de la url

Vamos a views y creamos la función que acabamos de poner en el anterior paso:

```
from .models import *
def concursos_listar(request):
   concursos = Concurso.objects.select_related("creador", "ganador").prefetch_related("participantes")
   concursos.all()
   return render(request,'Concursos_Online/lista.html',{'Concursos_Mostrar':concursos})
```

select_related: Sirve para obtener los datos de las relaciones **OneToOne, ManytoOne**
prefetch_related: Sirve para obtener los datos de las relaciones **ManytoMany**

En mi caso es:

```
select_related("creador", "ganador").prefetch_related("participantes")
```

porque en **Concursos_Online/models.py** tiene estos atributos que son relaciones:

```
# Relacion 1:N con Administrador (Crea)
creador = models.ForeignKey...
# Relacion 1:N con Participante (Gana)
ganador = models.ForeignKey...
# Relacion N:N con Participante (Inscribe)
participantes = models.ManyToManyField...
```

esto es porque hay varios concursos

```
concursos.all()
```

La consulta SQL

```
"""
   concursos = (Concurso.objects.raw(
   "SELECT * FROM Concursos_Online_concurso co "
   + " JOIN Concursos_Online_administrador ad ON co.creador = ad.id "
   + " JOIN Concursos_Online_participante pa ON co.ganador = pa.id "
   + " JOIN Concursos_Online_inscribe ins ON co.id = ins.concurso_id "
   + " JOIN Concursos_Online_participante p ON p.id = ins.participante_id"
   ))
"""
```

Ahora hacemos el html para la lista

```
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8">
   <meta name="viewport" content="width=device-width, initial-scale=1.0">
   <title>Hoteles</title>
</head>
<body>

   <h1>HOTELES</h1>

   {% for hotel in hoteles_mostrar %}
   <div>
       <h2>Nombre: {{ hotel.Nombre }}</h2>
       <h2>Descripcion: {{ hotel.descripcion }}</h2>
       <h2>Direccion: {{ hotel.direccion }}</h2>
       <h2>Fecha_fundacion: {{ hotel.fecha_fundacion }}</h2>
       <h2>Calificacion: {{ hotel.calificacion }}</h2>
       <h2>Servicios:</h2>

       <ul>
           {% for hotel in hoteles_mostrar %}
               <li>{{ Servicios.Nombre }}</li>
           {% endfor %}
       </ul>
   </div>
   {% endfor %}
</body>
</html>
```

Y añadimos al index el enlace

```
<li><a href="{% url 'lista_concursos' %}">Concursos</a></li>
```

el nombre esta en el archivo de urls de la aplicación en mi caso Concursos_Online/urls.py

```
urlpatterns = [
   path('',views.index, name='index'),
   path('concursos-online/listar', views.concursos_listar, name='lista_concursos')
]
```

hacemos un

python manage.py runserver

Nota:
si tienes problemas que no aparecen los datos, comprueba los nombres en vista y el html, si estan bien comprueba que tienes los datos en la base de datos (archivo sql)

si tu base de datos esta vacia lo mas facil y rapido es borrar el archivo sql y hacer un
python manage.py migrate
python manage.py generar_datos
python manage.py loaddata Concursos_Online/fixtures/datos.json

Si los datos que se generan son erróneos tendras que corregir el faker

y por ultimo guardamos cambios en git

git add .
git commit -m “Terminar url_1”
git push --set-upstream origin url_1

ahora tenemos la rama en el remoto, ahora hay que hacer el merge a la rama de Tarea_url

git checkout Tarea_url
git merge url_1
git push

ahora creamos la rama para la url 2:

git branch url_2
git checkout url_2

Ahora vamos a crear una URL para acceder a un < tu modelo principal > en concreto:

Concursos_Online/urls.py

```
path('concursos-online/<int:id_concurso>', views.dame_concurso, name='dame_concurso')
```

Vamos a views y creamos la función que acabamos de poner en el anterior paso:

el metodo de la vista es igual que la anterior pero poniendo un `.get(id=id_concurso)` al final, y quitando el `concursos.all()` porque solo se devuelve un concurso.

Tambien tienes que añadir el id como atributo en el def

```
def dame_concurso(request, id_concurso):
```

```
concurso = (
       Concurso.objects
       .select_related(
           "creador",
           "ganador",
      )
       .prefetch_related("participantes")  # Relacion N:N en participantes
       .get(id=id_concurso) # Obtiene el concurso con el ID especificado
)
```

La sentencia SQL

```
   concurso = (Concurso.objects.raw(
   "SELECT * FROM Concursos_Online_concurso co "
   + " JOIN Concursos_Online_administrador ad ON co.creador_id = ad.id "
   + " JOIN Concursos_Online_participante pa ON co.ganador_id = pa.id "
   + " JOIN Concursos_Online_inscribe ins ON co.id = ins.concurso_id "
   + " JOIN Concursos_Online_participante p ON p.id = ins.participante_id"
   + " WHERE co.id = %s",[id_concurso])[0]
   )
```

y por ultimo guardamos cambios en git

```
git add .
git commit -m “Terminar url_2”
git push --set-upstream origin url_2
```

ahora tenemos la rama en el remoto, ahora hay que hacer el merge a la rama de Tarea_url

```
git checkout Tarea_url
git merge url_2
git push
```

ahora creamos la rama para la url 3:

```
git branch url_3
git checkout url_3
```

Ahora vamos a crear una URL
Una url que me muestre los <tu_modelo> de un año y mes concreto:
EJ de mi caso
Una url que muestre los Concursos que comienzan en un año y mes concreto

Concursos_Online/urls.py

```
path('concursos-online/<int:anyo_concurso>/<int:mes_concurso>', views.dame_concursos_fecha,
name='dame_concursos_fecha')
```

```
# Una url que muestre los Concursos que comienzan en un año y mes concreto
def dame_concursos_fecha(request, anyo_concurso, mes_concurso):

    concursos = (
        Concurso.objects
        .select_related("creador", "ganador")
        .prefetch_related("participantes")  # Relacion N:N en participantes
    )
    concursos = concursos.filter(fecha_inicio__year=anyo_concurso, fecha_inicio__month=mes_concurso)
    concursos.all()

    """
    # Convertir el mes a cadena y asegurar que tenga 2 digitos con relleno de cero
    mes_formato_sql = str(mes_concurso).zfill(2)

    concursos = (Concurso.objects.raw(
    "SELECT * FROM Concursos_Online_concurso co "
    + " JOIN Concursos_Online_administrador ad ON co.creador_id = ad.id "
    + " JOIN Concursos_Online_participante pa ON co.ganador_id = pa.id "
    + " JOIN Concursos_Online_inscribe ins ON co.id = ins.concurso_id "
    + " JOIN Concursos_Online_participante p ON p.id = ins.participante_id"
    + " WHERE strftime('%%Y', co.fecha_inicio) = %s "
    + " AND strftime('%%m', co.fecha_inicio) = %s "
    ,[str(anyo_concurso),mes_formato_sql] # Usamos la variable formateada
    ))
    """

    return render(request,'Concursos_Online/lista_Concurso.html',{'Concursos_Mostrar':concursos})
```

y por ultimo guardamos cambios en git

```
git add .
git commit -m “Terminar url_3”
git push --set-upstream origin url_3
```

ahora tenemos la rama en el remoto, ahora hay que hacer el merge a la rama de Tarea_url

```
git checkout Tarea_url
git merge url_3
git push
```

ahora creamos la rama para la url 4:

```
git branch url_4
git checkout url_4
```

Una url que me muestre los libros que tienen el idioma del libro o español ordenados por fecha de publicación:

EJ de mi caso
Una url que muestre los Concursos que estan activos o inactivos

y por ultimo guardamos cambios en git

```
git add .
git commit -m “Terminar url_4”
git push --set-upstream origin url_4
```

ahora tenemos la rama en el remoto, ahora hay que hacer el merge a la rama de Tarea_url

```
git checkout Tarea_url
git merge url_4
git push
```

ahora creamos la rama para la url 5:

```
git branch url_5
git checkout url_5
```

…

y por ultimo guardamos cambios en git

```
git add .
git commit -m “Terminar url_5”
git push --set-upstream origin url_5
```

ahora tenemos la rama en el remoto, ahora hay que hacer el merge a la rama de Tarea_url

```
git checkout Tarea_url
git merge url_5
git push
```

ahora creamos la rama para la url 5:

```
git branch url_6
git checkout url_6
```

ahora creamos la rama para errores:

```
git branch url_error
git checkout url_error
```

aparte

git add .
git commit -m "Terminar url_4"
git push --set-upstream origin url_4
git checkout Tarea_url
git merge url_4
git push