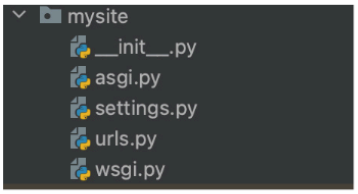
Repaso -Django

1.- Crear Proyecto en Pychar, cuidando que se cree con un entorno virtual venv

2.- Abrir la terminal pychar e instalar Django: ***pip install django***

3.- Crear un proyecto con Django: ***django-admin startproject mysite .*** (de esta manera se craer le proyecto dentro del proyecto creado en el punto 1)

Una ves creado el proyecto en Django analizaremos los archivos y para que se usan



settings.py: archivo de configuración central de todo el projecto Django

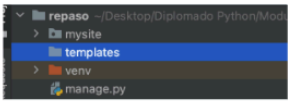
urls.py: archivo de rutas o urls principal del proyecto, acá van las rutas de tu landing y de cada app creada

4.- Una vez creado tu proyecto puedes agregar todas las vistas necesarias para hacer que tu landing page funcione adecuadamente, siguiendo los siguiente conceptos

A.- Templates, para crear tus templates debe ir a settings.py, modificar el arreglo TEMPLATES, para modificar la linea de DIRS y debe quedar de la siguiente manera

```
56  TEMPLATES = [
57      {
58          'BACKEND': 'django.template.backends.django.DjangoTemplates',
59          'DIRS': [BASE_DIR / "templates"],
60          'APP_DIRS': True,
61          'OPTIONS': {
62              'context_processors': [
63                  'django.template.context_processors.debug',
64                  'django.template.context_processors.request',
65                  'django.contrib.auth.context_processors.auth',
66                  'django.contrib.messages.context_processors.messages'
67              ],
68          },
69      },
70  ]
```

B.- Crear en el mismo nivel de la app mysite una carpeta llamada templates



C.- Crear dentro de templates un archivo llamado base.html, acá vas a declarar todo el html principal que servirá de como template padre para aplicar la herencia, (acá colocar tus archivos de bootstrap y todos aquellos archivos generales a tu landing, recuerda que todo lo va acá se cargará en todos los templates que hereden de el.

No olvides declara todos los bloques necesarios para que puedas implementar la herencia.

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>{% block title %} Mysite {% endblock %}</title>
    {% load static %}
    <link rel="stylesheet" href="{% static 'css/bootstrap/bootstrap.css' %}">
    <link rel="stylesheet" href="{% static 'css/styles.css' %}">
</head>
<body>
<nav class="navbar navbar-expand-lg bg-body-tertiary"...>
{% block content %}
{% endblock %}
<script src="{% static 'js/bootstrap/bootstrap.js' %}"></script>
</body>
</html>
```

D.- Crear todos los templates necesario para las rutas que vas a declarar: home, contact, about, projects etc etc

```
{% extends "base.html" %}

{% block title %} Home {% endblock %}

{% block content %}
    <h1>About</h1>
{% endblock %}
```

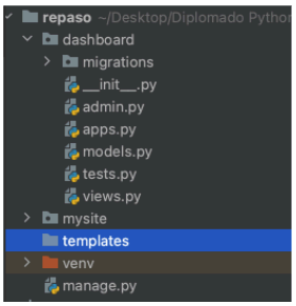
E.- Ir al archivo views.py dentro de tu app mysite, y crear todas las vitas necesarias para que puedan ser llamadas desde la urls.py

```
from django.shortcuts import render

Richar
def home(request):
    return render(request, 'home.html')


Richar
def contact(request):
    return render(request, 'contact.html',)


Richar
def about(request):
    return render(request, 'about.html')
```

F.- Ir al archivo urls.py dentro de tu app mysite, y crear todas las rutas necesarias y que hagan coherencia con tus templates o vistas de tu landing

```python
from django.contrib import admin
from django.urls import path
from . import views

urlpatterns = [
    path('admin/', admin.site.urls),
    path('', views.home, name='home'),
    path('contact/', views.contact, name='contact'),
    path('about/', views.about, name='about'),
]
```

5.- Para comenzar a agregar apps a tu sistema web, lo debes hacer con el siguiente comando: ***python manage.py startapp dashboard***, esto creará un app a nivel de mysite con la siguiente estructura



admin.py: permite declarar los modelos y sus configuraciones para que sean administrados desde el panel de administrador

models.py: permite declarar todas la configuraciones de tus modelos

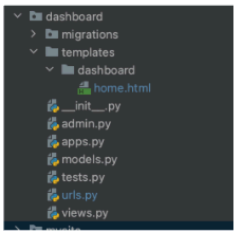
views.py: permite declara todas funciones que reciben las peticiones desde el front y que llaman desde el urls.py

6.- Una vez creado tu app puedes agregar todas las vistas necesarias para hacer que tu app funcione adecuadamente, siguiendo los siguiente pasos

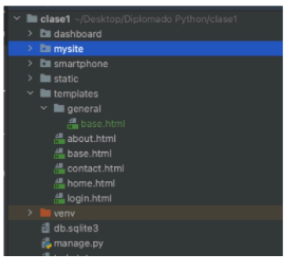
A.- Ir a mysite/settings.py, y agregar la app recién creada al arreglo INSTALLED_APPS

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

B.- Crear dentro de tu app una carpeta llamada templates y dentro de esta, otra carpeta con el mismo nombre de tu app



C.- Crear dentro de la carpeta templates que esta a nivel de tu proyecto una carpeta llamada general y alli crear tu archivo base.html para que sirva de template padre para la herencia de tus apps



D.- Crear todos los templates necesario para las rutas que vas a declarar

E.- Ir al archivo views.py dentro de tu app, y crear todas las vitas necesarias para que puedan ser llamadas desde la urls.py

F.- Crear en tu app el archivo urls.py en donde deberás crear todas las rutas necesarias y que hagan coherencia con tus templates o vistas de tu app

G.- ir al archivo mysite/urls.py, debes agregar las url declaradas en tu app dashboard

```python
from django.contrib import admin
from django.urls import path, include
from . import views
from django.contrib.auth import views as auth_views

urlpatterns = [
    path('admin/', admin.site.urls),
    path('', views.home, name='home'),
    path('contact/', views.contact, name='contact'),
    path('about/', views.about, name='about'),
    path('dashboard/', include('dashboard.urls')),
    path('smartphone/', include('smartphone.urls'))
]
```

7.- Para agregar postgres a tu proyecto debes ir al archivo mysite/settings.py y agregar el driver respectivo, mas la credenciales de tu base de datos

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.postgresql',
        'NAME': 'factory',
        'USER': 'postgres',
        'PASSWORD': '******',
        'HOST': 'localhost',
        'PORT': '5432',
    }
}
```

8.- Para realizar tu migraciones incluyendo las iniciales siempre debes ejecutar estos comandos:

python manage.py makemigration *(esto buscara todos lo modelos modificados o agregados y creará los archivos de migración)*
python manage.py migrate *(esto ejecutará las migraciones del comando anterior y llevara los cambios a la base de datos)*

Search tabs

9.- Para agregar autenticación a nuestro proyecto debemos ir al mysite/settings.py y agregar lo siguiente

```
LOGIN_REDIRECT_URL = 'dashboard:home'
LOGOUT_REDIRECT_URL = 'login'
LOGIN_URL = '/login/'
```

LOGIN_REDIRECT_URL, url a donde sera redireccionado la aplicación cuando alguien haga login efectivamente.
LOGOUT_REDIRECT_URL, url a donde será redireccionado la aplicación cuando se haga un logout efectivamente.
LOGIN_URL, url donde esta el inicio de sesión

10.- Ir a mysite/settings.py y agregar las rutas de autenticación login y logout

```
urlpatterns = [
    path('admin/', admin.site.urls),
    path('', views.home, name='home'),
    path('contact/', views.contact, name='contact'),
    path('about/', views.about, name='about'),
    path('login/', auth_views.LoginView.as_view(template_name='login.html'), name='login'),
    path('logout', auth_views.LogoutView.as_view(), name='logout'),
    path('dashboard/', include('dashboard.urls')),
    path('smartphone/', include('smartphone.urls'))
]
```

11.- para proteger rutas y que estas solo puedan ser accedidas cuando un persona esta logueada, se debe ocupar el decorador @login_required

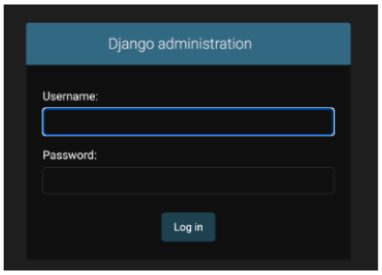
```
Richar
@login_required
def home(request):
    return render(request, 'dashboard/home.html')
```

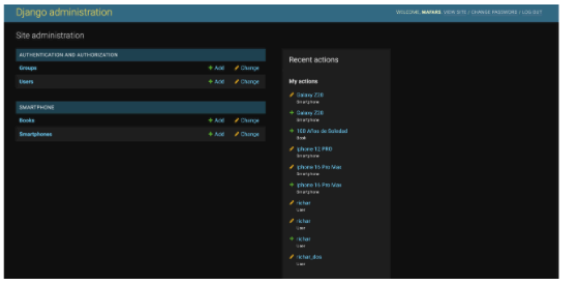
12.- Para navegar entre paginas a través de un menu se debe utilizar el milddleware url: **ir a Home**

13.- Accediendo al sitio administrador de Django

A.- Antes de acceder debes crear un super usuario, con los permisos suficientes para poder acceder al sitio administrativo, ejecutando en siguiente comendo en la consola de tu proyecto: **python manage.py createsuperuser**, y seguir las instrucciones

B.- Para acceder al sitio administrativo, una ves el proyecto este andando, si aun no has ejecutado las migracione siniciales puedes ir al punto 8 y correrlas, luego irl a 127.0.0.1:8000/admin

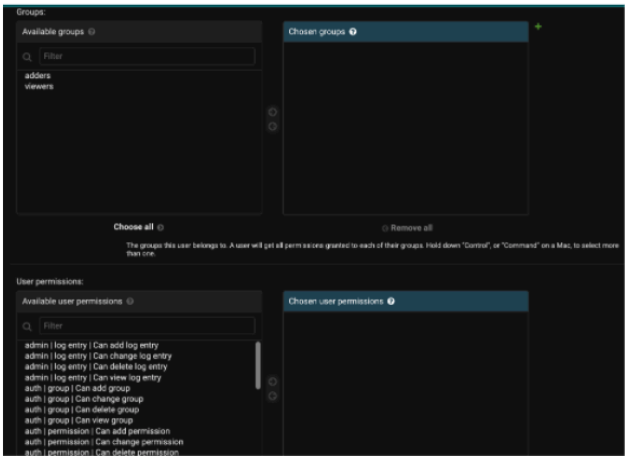


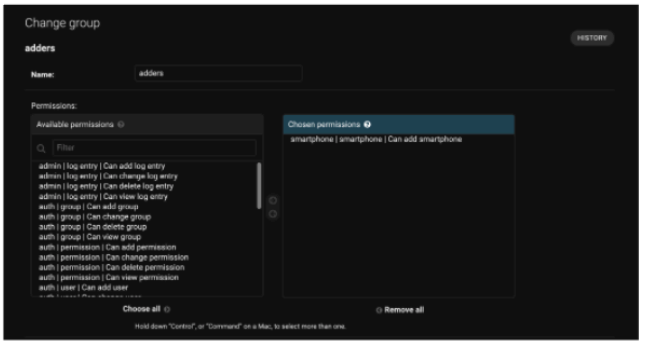


C.- debes proporcionar el usuario y password proporcionado cuando se creo el super usuario

D.- Una vez hecho un login encontraras al menos dos menus, "Groups" y "Users", en el primero, se crean grupos y permisos de acceso, para poder manejarlo desde el tu aplicación, también podrás configurar los permisos sobre tus propios modelos.

En el segundo podrás crear usuarios y asignarlos los permisos y/o grupos necesarios para su funcionamiento, incluso darle permisos como administrador





12.- Agregando models al sitio administrativo

A.- Crear tu(s) modelos y ejecutar la migraciones

B.- Ir a tuapp/admin.py y registra tu modelo

```python
@admin.register(Smartphone)
class SmartphoneAdmin(admin.ModelAdmin):
    list_display = ['model', 'ram', 'height', 'price', 'status']
    search_fields = ['model']
    list_filter = ['height']
    ordering = ['-price', 'model']
    form = SmartphoneForm
```

list_display, es la minima propiedad que debes agregar cuando registras tu modelo, acá van al menos todos los campos obligatorios de tu modelo y que serán visibles en el administrador
search_fields, acá irán los campos en los cuales se hará una búsqueda de tipo texto, a través de un input search en el admin, sino existe no se muestra el input
list_filter, acá irán campo para hacer una búsqueda tipo select, como nota importante ocupa campo que sean de tipo choice para mejor rendimiento
ordering, acá puedes colocar aquellos campo a travez de los cuales quiera que haya un ordenamiento en la tabla cuando se meustre el listado de datos

C.- Modificar el aspecto de formulario en el admin

```python
from django import forms

from smartphone.models import Smartphone


class SmartphoneForm(forms.ModelForm):
    model = Smartphone
    fields = '__all__'

    def clean_price(self):
        price = self.cleaned_data['price']

        if price < 100:
            raise forms.ValidationError("El precio debe ser mayor a 100")

        return price
```

```python
form = SmartphoneForm
```

C.1. agregar un archivo form.py en tu app y para cada modelo de ser necesario, acá puedes colocar validaciones especificas como la que se muestra **clean_price**
C.2 agregar la variable form en el admin.py de tu app, apuntando al form que creaste para tu modelo

D.- Modificar el agrupamiento de campo en tu formulario del sitio administrador

```
fieldsets = (
    ('Basic', {
        'fields': ('model', 'ram')
    }),
    ('Cost', {
        'fields': ('height', 'price')
    })
)
```

para agregar los campos de tu formulario (practica recomendada cuando la cantidad de campo es es grande), debes agregar una nueva variable (**fieldsets**) en ella vas agregar una lista de listas y en cada lista va en la posición 1 el nombre del agrupador y en la posición un diccionario con al key 'fields' y el valor de esta sera una lista con los campos que pertenecen a este agrupador

E.- agregar acciones especificas al tu modelo

```
def marcar_agotado(self, request, queryset):
    actualizados = queryset.update(status='agotado')
    self.message_user(
        request,
        ngettext(
            '%d smartphone was updated',
            '%d smartphones where updated',
            actualizados,
        ) % actualizados,
        messages.SUCCESS
    )

actions=[marcar_agotado]
```

E.1.- para agregar acciones especificas debe declarar una función para cada acción con la modificación en lso datos que necesitas +. un mensaje de éxito para la operación
E.2 .- agregar la variable actions a tu admin, con una lista de tus acciones declaradas, loq ue hara que en el select action del menu tu modelo en el admin, estas acciones, aparezcan

F.- Agregando modelos con relaciones

```python
class Book(models.Model):
    title = models.CharField(max_length=100)
    author = models.CharField(max_length=100)

    def __str__(self):
        return self.title


class Chapter(models.Model):
    book = models.ForeignKey(Book, on_delete=models.CASCADE, related_name='chapters')
    title = models.CharField(max_length=100)
    pages = models.PositiveIntegerField()

    def __str__(self):
        return f"Capitulo {self.title}: Pages{self.pages}"
```

F.1.- agregar lo modelos y relaciones necesarias

```python
class ChapterInline(admin.TabularInline):
    model = Chapter
    extra = 2


@admin.register(Book)
class BookAdmin(admin.ModelAdmin):
    list_display = ['title', 'author']
    inlines = [ChapterInline]
```

F.2.- Agregar al admin.py el modelo principal utilizando TabularInLine model