# Curso de Django píldoras informáticas

## Video 11. BBDD

* Crear nuevo proyecto
* Django-admin startproject TeindaOnline
* Python manage.py startapp gestionPedidos

## Video 12

* Se empieza a trabajar con el archivo model.py
* Se crea una clase por cada tabla
* En el archivo settings.py
* Registrar la app
* Para chequear si hay error
* Python manage.py check gestionPedidos
* Para crear la BBDD
* Python manage.py makemigrations
* Para crear las tablas
* Python manage.py sqlmigrate gestionPedidos
* Crear tabla en sqlite
* Python manage.py migrate

## Video 13

* Python manage.py shell
* Con esta instrucción abre la consola de Python para trabajar con BBDD
* From gestionPedidos.models import Articulos
* Art= Articulos(nombre=”mesa”,Sección=”decoracion”,precio=90)
* Art.save()
* Esta es la forma de insertar datos de a uno
* Art2=Artículos.objetcs.create(nombre=”taladro”,sección=”ferreteria”,precio=65)
* Con esta instrucción se agrega a la tabla de una sola vez

**Modificar datos**

usar nombre variable existente

art.precio=300

art.save()

* Acceder a la propiedad del objeto cambiar y guardar

**Eliminar**

Delete : crear una variable y pasar el objeto a eliminar

* Art5=Artículos.objetcs.get.all()
* Lista.query.\_\_str\_\_()

## Video 14

Trabajo con Postgresql

Clave: jorge

* En el editor de la base de datos
* Ventana query
* Ingresar: créate database **prueba**
* **Instalar** pip install psycopg2
* **CAMBIAR EL ARCHIVO SETTING**
* DATABASES = {
* 'default': {
* 'ENGINE': 'django.db.backends.postgresql\_psycopg2',
* 'NAME':'ArticulosClientes',
* 'USER': 'postgres',
* 'PASSWORD':'jorge',
* 'HOST':'127.0.0.1',
* 'DATABASE\_PORT':'5432',
* }
* }
* Python manage.py makemigrations
* Python manage.py migrate (a*plica todos los cambios)*
* Python manage.py Shell abre la consola para agregar datos
* En el Shell
* From gestionPedidos.models import Clientes
* Cli=Clientes(nombre=“jorge”.direccion=”mi casa”,tfno=”124456789”)
* Cli.save()

# Video 15

Postgresql con where

From gestionPedidos.models import Articulos

Articulos.objetcs.filter(seccion=’muebles’)

Se agregar la visualización de datos a la clase de artículos

Se cierra el Shell y se vuelve a cargar de nuevo la migration para que tome los cambios

* Python manage.py makemigrations
* Python manage.py migrate
* Vuelvo a entrar en el Shell
* Python manage.py Shell
* From gestionPedidos.models import Articulos
* Articulos.objects.filter(seccion=’muebles’)
* Articulos.objects.filter(seccion=’muebles’,precio\_\_gte=100) muestra los que valen mas de 100
* Articulos.objects.filter(seccion=’muebles’,precio\_\_lte=100) muestra los que valen menos de 100
* Articulos.objects.filter(seccion=’muebles’,precio\_\_range=(10,100) muestra los que valen entre 10 y 100
* Articulos.objects.filter(seccion=’muebles’,precio\_\_gte=100). order\_\_by(‘precio’) los ordena por precios crecientes de menor a mayor
* Articulos.objects.filter(seccion=’muebles’,precio\_\_gte=100). order\_\_by(‘-precio’) los ordena por precios decrecientes de mayor a menor

# Video 16. Panel de administración I

* Desde la terminal ingresar
* Python manage.py createsuperuser
* Pide crear un nombre de usuario en mi caso **jorge**
* **Email :** [jorgecialm@gmail.com](mailto:jorgecialm@gmail.com)
* **Password :** 123456789

# Video 17. Panel de administración II

* Modificar el archivo admin.py
* from gestionPedidos.models import Clientes
* from django.contrib import admin
* from gestionPedidos.models import Clientes
* # Register your models here.
* admin.site.register(Clientes)
* esto agrega clientes al panel de administración
* también agrego los modelos Artículos y pedidos
* from gestionPedidos.models import Clientes,Articulos,Pedidos
* # Register your models here.
* admin.site.register(Clientes)
* admin.site.register(Articulos)
* admin.site.register(Pedidos)
* si quiero que el campo de mail no lo requiera hago lo siguiente:
* email= models.EmailField(blank=True,null=True)
* Después de esto guardar los cambios
* Pthyon manage.py makemigrations
* Python manage.py migrate
* En el panel de administración la tabla de clientes el Email (ya no aparece en negrita que es lo que indica que es requerido)

# Video 18. Panel administración III

* direccion = models.CharField(max\_length=50,verbose\_name="La direccion")
* Con esta instrucción hacemos que cuando se abre el panel de administración y se muestra la tabla de clientes “El nombre del campo se cambia a: *La dirección* “
* Hacer que cuando quiera ver un cliente me muestre el nombre
* def \_\_str\_\_(self) :
* return self.nombre
* Después de esto guardar los cambios
* Pthyon manage.py makemigrations
* Python manage.py migrate
* Hacer que se muestre distinto la tabla clientes
* class ClientesAdmin(admin.ModelAdmin):
* list\_display=("nombre","direccion","telefono")
* admin.site.register(Clientes,ClientesAdmin)
* se crea la clase model cliente para mostrar de otra forma con todos los datos
* agregar un filtrado de nombre
* search\_fields=("nombre","telefono")
* Se agrega esta instrucción que completa queda así
* class ClientesAdmin(admin.ModelAdmin):
* list\_display=("nombre","direccion","telefono")
* search\_fields=("nombre","telefono")

# Video 19. Panel de administración IV

* Agregar una clase para aplicar un filtro a la tabla de artículos
* class ArticulosAdmin(admin.ModelAdmin):
* list\_filter=("seccion",)
* admin.site.register(Articulos,ArticulosAdmin)
* también se puede trabajar con las fechas en este caso se lo hace con la tabla de pedidos que tiene un campo de fecha.
* class PedidosAdmin(admin.ModelAdmin):
* list\_display=("numero","fecha")
* list\_filter=("entregado","fecha")
* date\_hierarchy="fecha"
* admin.site.register(Pedidos,PedidosAdmin)
* list\_filter muestra opciones de filtrados de fecha
* date\_hierarchy : muestra un filtro tipo miga de pan
* cambiar idioma del panel
* cambiar el archivo settings.py
* LANGUAGE\_CODE = 'en-us'
* LANGUAGE\_CODE = 'es-us'
* Lo cambio a “es-us”

# Video 20. Panel de administración V

* Video sobre como agregar usuarios y grupo para interactúen con el panel de administración
* Con los diferentes permisos

# Video 21. Formularios I

* Creación de formulario (manual) y envio de datos al servidor
* Objetos HttpRequest
* Crear una carpeta templates en gestionPedidos
* Dentro de esta carpeta agregar un archivo “búsqueda\_productos.html”
* <html>
* <head>
* <title>Busqueda de productos</title>
* </head>
* <body>
* <form action="/buscar/" method="get">
* <input type="text" name="prd">
* <input type="button" value="Buscar">
* </form>
* </body>
* </html>
* Dentro del archivo views.py creo una vista
* def busqueda\_productos(request):
* return render(request,"busqueda\_productos.html")
* Registrar la url
* path('busqueda\_productos.html',views.busqueda\_productos),
* Importar el archivo view
* from gestionPedidos import views
* agregamos una nueva función buscar a el archivo views.py
* def buscar(request):
* mensaje="Articulo buscado : %r " %request.get["prd"]
* return HttpResponse(mensaje)
* Agregamos a las referencias
* from django.http import HttpResponse
* registrar la nueva vista en las urls.py
* path("buscar/",views.buscar),

# Video 21. Formularios II

* Búsqueda de información en BBDD.
* producto= request.GET['prd']
* articulos=articulos.objects.filter(nombre\_icontrains=producto)
* Se utiliza para buscar dentro de la base de datos
* def buscar(request):
* if request.GET['prd']:
* #mensaje="Articulo buscado : %r " %request.GET["prd"]
* producto= request.GET['prd']
* articulos=articulos.objects.filter(nombre\_icontrains=producto)
* else:
* mensaje="No introdujo nada"
* return HttpResponse(mensaje)
* Dentro del archivo views.py agregamos una nueva instrucción para mostrar articulo
* return render(request,'resltado\_busqueda.html',{'articulos':articulos,'query':producto})
* Se crea la nueva plantilla = resultado\_busqueda.html
* <html>
* <body>
* <p>Estas buscando : <strong>{{query}}</strong></p>
* {% if articulos %}
* <p>Articulos encontrados : {{articulos|length}} articulos</p>
* <ul>
* {% for articulo in articulos %}
* <li>
* {{articulo.nombre}} &nbsp; {{articulo.seccion}} &nbsp; {{articulo.precio}}
* </li>
* {% endfor %}
* </ul>
* {% else %}
* <p>No existe el articulo que busca</p>
* {% endif %}
* </body>
* </html>

# Video 23: Formularios II

* Trabajo con formularios
  + Limitar n° caracteres a buscar en BBDD
  + creación de formulario de contacto (1° parte)
  + la idea de limitar es que no consuma tantos recursos
* se agrega instrucciones al archivo views.py
* se agrega un if anidado
* if len(producto)>20:
* mensaje="Texto de busqueda demasiado largo"
* else:
* articulos=Articulos.objects.filter(nombre\_\_icontains=producto)
* return render(request,'resultados\_busqueda.html',{'articulos':articulos,'query':producto})
* Se agrega la parte del ir el resto sigue como esta , es decir si se introducen mas de 20 caracteres indica que es muy largo
* Agregamos una función “contacto al archivo views.py”
* def contacto(request):
* return render(request,'contacto.html')
* Modificar el archive urls.py
* path('contacto/',views.contacto),

# Video 24. Envío de mails

* Modificar el archive settings.py
* CONFIGURAR GMAIL PARA QUE TRABAJE CON APLICACIONES DE TERCEROS
* EMAIL\_BACKEND="django.core.mail.backends.smtp.EmailBackend"
* EMAIL\_HOST="smtp.gmail.com"
* EMAIL\_USE\_TLS=True
* EMAIL\_PORT=587
* EMAIL\_HOST\_USER= "jorgecialm@gmail.com"
* EMAIL\_HOST\_PASSWORD="iszs obpo wlhj dafo"
* Lo probamos en la consola
* El password se obtiene desde la configuración de la cuenta de Gmail, que permite que se envíen correos desde aplicaciones
* >>> from django.core.mail import send\_mail
* >>> send\_mail('Asunto','Mensaje','jorgecialm@gmail.com',['jorgecialm@hotmail.com'],fail\_silently=False)
* De esta forma se envía el correo y funciono
* Para realizar esto desde la consola se abre el Shell
* Con py manage.py shell
* Para hacer que funcione con el formulario de correo
* Hay que registrarlo en views.py de gestión pedidos
* Modifico la función contacto
* def contacto*(request)*:
* if request.method=="POST":
* subject= request.POST["asunto"]
* message=request.POST["mensaje"]+" "+ request.POST["email"]
* email\_from=settings.EMAIL\_HOST\_USER
* recipient\_list=["jorgecialm@hotmail.com"]
* send\_mail(subject,message,email\_from,recipient\_list)
* return render(request,"gracias.html")
* return render(request,'contacto.html')

# Video 25. Api Forms

* Crear el archivo forms.py en la carpeta del proyecto
* from django import forms
* class FormularioContacto(forms.Form):
* asunto = forms.CharField()
* email=forms.EmailField()
* mensaje=forms.CharField()
* Agregar al archivo views.py
* from gestionPedidos.forms import FormularioContacto
* modifico la función contacto
* def contacto*(request)*:
* if request.method=="POST":
* miFormulario= FormularioContacto(request.POST)
* if miFormulario.is\_valid():
* infForm=miFormulario.cleaned\_data
* send\_mail(infForm['asunto'],infForm['mensaje'],infForm.get('email',''),['jorgecialm@hotmail.com'],)
* return render(request,'gracias.html')
* else:
* miFormulario=FormularioContacto()
* return render(request,'formulario\_contacto.html',{'form':miFormulario})
* Con esta plantilla de html se crea un nuevo formulario con otra disposición

# Video 26. Proyecto web Completo I

* Proyecto web completo con Django
* Creación de Urls y Vistas
* Creación de plantillas y herencias
* Manejar archivos externos
* Configuración del panel de administración
* Configuración y conexión con BBDD
* CRUD con BBDD
* Trabajo con formularios y envío de mails
* Despliegue

# Video 27. Proyecto web completo II

* Composición de la aplicación
  + Logo
  + Menú
    - Home
    - Servicios
    - Tienda
    - Blog
    - Contacto
  + Zana de carga de contenidos
  + Pie de pagina
* Empiezo creando el proyecto
  + Me situó en la carpeta que quiero alojar el proyecto
  + Invoco el siguiente comando:
    - Django-admin startproject **nombre del proyecto**
    - Se crea el proyecto ,me corro a la carpeta que cree
    - Creo la app con el siguiente comando:
      * Py manage.py startapp **nombre app**
* pruebo si funciona el proyecto lo arranco con la siguiente instrucción
  + py manage.py runserver
  + y aplico los cambios que hacen falta con la siguiente instrucción
    - py manage.py migrate
    - se aplican todos los cambios y se crea la BBDD
  + Después de hacer todo esto empiezo a trabajar con las vistas en el archivo views.py de la aplicación
* from django.shortcuts import render,HttpResponse
* # Create your views here.
* def home*(request)*:
* return HttpResponse("inicio")
* def servicio*(request)*:
* return HttpResponse("servicio")
* def tienda*(request)*:
* return HttpResponse("tienda")
* def blog*(request)*:
* return HttpResponse("blog")
* def contacto*(request)*:
* return HttpResponse("contacto")
  + luego de esto vamos al archivo urls y agregamos las vistas
* from django.contrib import admin
* from django.urls import path
* from ProyectoWebApp import views
* urlpatterns = [
* path('admin/', admin.site.urls),
* path('', views.home,name="Home"),
* path('servicios', views.servicio,name="Servicios"),
* path('tienda', views.tienda,name="Tienda"),
* path('blog', views.blog,name="Blog"),
* path('contacto', views.contacto,name="Contacto"),
* ]

# Video 28. Proyecto web completo III

* Urls y templates
* Un proyecto puede tener varias aplicaciones y estas pueden ser reutilizadas en diferentes proyectos
* Dentro de la aplicación se crea un archivo urls , la que va ser valida solo para la app
* from django.urls import path
* from ProyectoWebApp import views
* urlpatterns = [
* path('', views.home,name="Home"),
* path('servicios', views.servicio,name="Servicios"),
* path('tienda', views.tienda,name="Tienda"),
* path('blog', views.blog,name="Blog"),
* path('contacto', views.contacto,name="Contacto"),
* ]
* Y desde la urls.py del proyecto se enlaza a las urls de las aplicaciones en el caso de que hubiera mas de una
* from django.contrib import admin
* from django.urls import path,include
* from ProyectoWebApp import views
* urlpatterns = [
* path('admin/', admin.site.urls),
* path('ProyectoWebApp/',include('ProyectoWebApp.urls'))
* ]
* Dentro de la carpeta de app creo la carpeta templates
* Creo 5 archivos html
* Acto seguido hay que registrar la app en el archivo settings.py
* En
* INSTALLED\_APPS = [
* 'django.contrib.admin',
* 'django.contrib.auth',
* 'django.contrib.contenttypes',
* 'django.contrib.sessions',
* 'django.contrib.messages',
* 'django.contrib.staticfiles',
* 'ProyectoWebApp',
* ]
* Luego de esto queda hacer que se rendericen las vistas

# Video 29. Proyecto web completo IV

* Formato a sitio web con BootStrap
* Creación de plantillas (templates)
* Herencia de plantillas y estructura del sitio
* ¿Qué es Bootstrap?
  + Framework css creado por Twitter en 2011
* ¿Para que sirve?
  + Permite dar formato a un sitio web utilizando librerías css
* Etc
* A partir de la plantilla de Bootstrap modificar para mi proyecto

# Video 30. Proyecto web completo V

* Modificar
  + Colocar al principio del archivo
* {% load static %}
  + Modificar el link a Bootstrap
* <link href="{% static "ProyectoWebApp/vendor/bootstrap/css/bootstrap.min.css"%}" rel="stylesheet">
  + Agregue **% static/ProyectoWebApp** a la ruta
* <link href="{% static "ProyectoWebApp/css/gestion.css" %}" rel="stylesheet">
  + También se modifica la parte de estilos
  + También modificar el archivo de los js incorporando la carpeta static y la ruta del proyecto
* <!-- Bootstrap -->
* <script src="{% static "ProyectoWebApp//vendor/jquery/jquery.min.js"%}"></script>
* <script src="{% static "ProyectoWebApp//vendor/bootstrap/js/bootstrap.bundle.min.js"%}"></script>
  + También cambiar los enlaces a la imagen
* <img class="intro-img img-fluid mb-3 mb-lg-0 rounded" src="{% static "ProyectoWebApp/img/principal.jpg"%}" alt="">
  + La cabecera y el pie es común para todas las paginas
  + Hacer herencia de plantillas
  + Crear una copia de la plantilla home
  + Le cambiamos el nombre a base y se elimina la parte de “Heading” y la de “message”
  + En su lugar se crea
* <!-- Contenido cambiante -->
* {% block content %}
* {% endblock %}
  + La plantilla home debe heredar de base se elimina todo menos la parte de heading y la de message
  + Y se coloca al principio de la plantilla la siguiente instrucción
* {% extends "ProyectoWebApp/base.html" %}
  + Ahora resta modificar el archivo base.html para que funcionen los enlaces
* {% extends "ProyectoWebApp/base.html" %}
* <a class="navbar-brand text-uppercase text-expanded font-weight-bold d-lg-none" href="{% url "Home" %}">Gestión de Pedidos</a>
* <a class="nav-link text-uppercase text-expanded" href="{% url "Home" %}">Inicio</a>
* <a class="nav-link text-uppercase text-expanded" href="{% url "Servicios" %}">Servicios</a>
* <a class="nav-link text-uppercase text-expanded" href="{% url "Tienda" %}">Tienda</a>
* <a class="nav-link text-uppercase text-expanded" href="{% url "Contacto" %}">Contacto</a>
* <a class="nav-link text-uppercase text-expanded" href="{% url "Blog" %}">Blog</a>
  + Se cambiaron todas las llamadas a las paginas

# Video 31. Proyecto web completo VI

* Completar el resto de paginas con herencia de plantillas
* En cada una de las paginas html coloco lo siguiente
* {% extends "ProyectoWebApp/base.html" %}
* {% load static %}
* {% block content %}
* {% endblock%}
* En la de servicio, tienda, contacto y blog

# Video 32: Proyecto web completo VII

* Creación de app servicio
* Con el comando py manage.py startapp Servicio
* Ahora el siguiente paso es registrar la app en el archivo setting.py en la parte de apps
* Vamos a registrar los modelos clases que luego se transforman en tablas de la base de datos
* from django.db import models
* # Create your models here.
* class Servicio(models.Model):
* titulo=models.CharField(max\_length=50)
* contenido=models.CharField( max\_length=50)
* imagen=models.ImageField()
* created=models.DateTimeField(auto\_now\_add=True)
* updated=models.DateTimeField(auto\_now\_add=True)
* class Meta:
* verbose\_name="servicio"
* verbose\_name\_plural="servicios"
* def \_\_str\_\_*(self)*:
* return self.titulo
* Luego de esto hacer la migración para guardar los datos

# Video 33. Proyecto web completo VIII

* Registro de aplicación en el panel de administración
* Registrar la aplicación en el archivo admin.py
* from django.contrib import admin
* from .models import Servicio
* # Register your models here.
* admin.site.register(Servicio)
* para hacer que el panel se vea en español cambiar el valor de LANGUAGE\_CODE = “es-us”

# Video 34. Proyecto web completo IX

* django no permite mostrar imágenes incorporadas en BBDD
* se agregan instrucciones a settings.py
* MEDIA\_URL='/media/'
* MEDIA\_ROOT = os.path.join(BASE\_DIR,'media')
* LANGUAGE\_CODE = 'es-ar'
* TIME\_ZONE = 'America/Argentina/Buenos\_Aires'
* La parte de lenguaje y cona horaria la cambie para que coincida con el reloj del pc cuando guardo los cambios de la base de datos
* En la urls.py se agrega
* urlpatterns += static(settings.MEDIA\_URL, document\_root=settings.MEDIA\_ROOT)

# Video 35. Proyecto web completo X

* mostrando servicios
* en servicios.html agrego más código
* {% extends "ProyectoWebApp/base.html" %}
* {% load static %}
* {% block content %}
* {% for servicio in servicios%}
* <div><p>
* <h2 style="color: white">{{servicio.titulo}}</h2>
* <p style="color: white">{{servicio.contenido}}</p>
* <p><img src="{{servicio.imagen.url}}"></p>
* </p></div>
* {% endfor %}

# Video 36. Proyecto web completo XI

* Mover los archivos dentro de la clase que vamos a utilizar para que pudieran ser reutilizables
* En principio creamos dentro de la carpeta servicios donde se aloja la app , creamos la carpeta templates y dentro de esta servicios
* Muevo la vista de servicios a views.py de servicios junto con la importación y queda de la siguiente manera
* from django.shortcuts import render
* from Servicios.models import Servicio
* #Se crean las vistas de la app
* def servicio*(request)*:
* servicios=Servicio.objects.all()
* return render(request,"ProyectoWebApp/servicios.html",{"servicios":servicios})
* Creo el archivo de urls.py dentro de servicios y copio todo lo que hay en la uuls.py del proyecto
* Cambio la urls. Py del proyecto
* from django.contrib import admin
* from django.urls import path,include
* from ProyectoWebApp import views
* urlpatterns = [
* path('admin/', admin.site.urls),
* path('servicios/', include('servicios.urls')),
* path('',include('ProyectoWebApp.urls')),
* ]