Esta clase va a ser

- grabada

Clase 22. PYTHON

Playground Avanzado Parte I

CODERHOUSE

Temario

21

Playground Intermedio Parte III

- ✓ Formularios
- ✓ Creación de formularios
- ✓ Búsquedas con Form

22

Playground Avanzado Parte I

- ✓ CRUD
- ✓ Clases basadas en vistas

23

Playground Avanzado Parte II

- ✓ Login - Registro - Logout
- ✓ Mixin y Decoradores

CODERHOUSE

Objetivos de la clase

- **Implementar** CRUD.
- **Ejecutar** CRUD con CBV.

Repositorio Github

Te dejamos el acceso al Repositorio de Github donde encontrarás todo el material complementario y scripts de la clase.

✓ Repositorio Python

CODERHOUSE

CRUD

¿Qué es CRUD?

¿Qué es CRUD?

En informática, CRUD es el acrónimo de "Crear, Leer, Actualizar y Borrar" (del original en inglés: Create, Read, Update and Delete). También es conocido como ABM en otros entornos (Altas, Bajas y Modificaciones).

Es el acto de agregar, modificar, eliminar o acceder a datos de una base de datos.
Sin saberlo, algo de eso ya hemos realizado.
Ya sabemos insertar datos y buscar datos.

¿Qué es CRUD?

Pero ahora llegó la hora de formalizar un poco esos conceptos y agregar lo que nos falta. Borrar y modificar datos de la BD.

Desde el panel de administración es muy simple. La idea es hacerlo desde nuestras vistas y darle nuestra impronta.

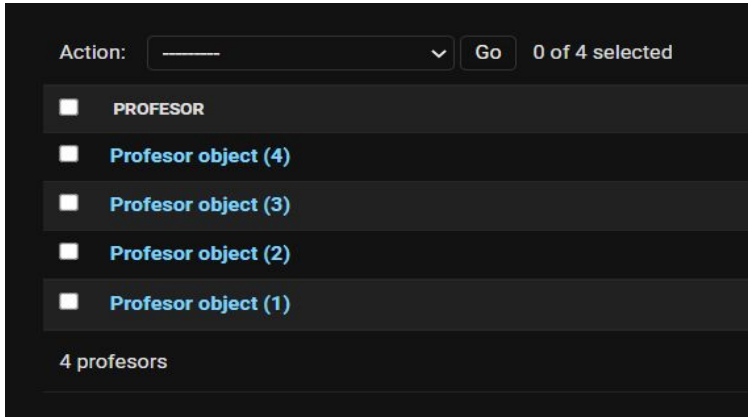
Arrancando con CRUD

CODERHOUSE

Arrancando con CRUD

Primero vamos a agregar datos a alguna de nuestras tablas por medio del panel, así se ven mejor los resultados que obtendremos. Por ejemplo, voy a crear 4 profesores, y trabajaré todo el ejemplo con los profesores.

Primero detalles que no tienen que ver con CRUD, ya agregué 4 profesores por medio del panel de Admin. 👉



Arrancando con CRUD

Esto lo habíamos pasado por alto pero los objetos generados en la BD se ven poco agradables y no se puede entender cuál información tenemos.

Arreglarlo es muy simple 👇

Solo hay que agregar el método `__str__(self):` en la clase que quieres que se vea distinta (o en todas).

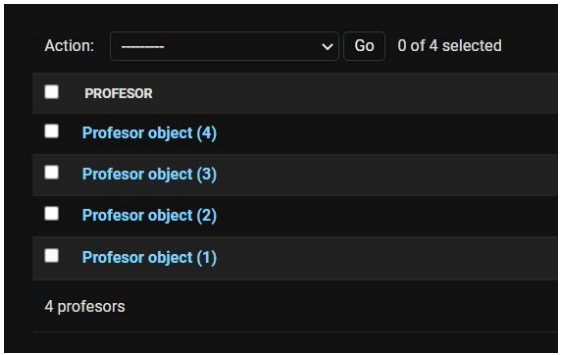
CODERHOUSE

Arrancando con CRUD

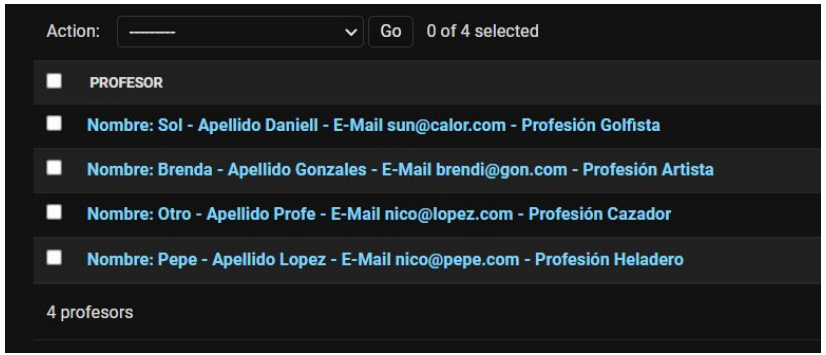
Veamos cómo hacerlo 👇

```
def __str__(self):
        return f"Nombre: {self.nombre} - Apellido {self.apellido} - E-Mail {self.email} - Profesión {self.profesion}"
```

```python
def __str__(self):
    return f"Nombre: {self.nombre} - Apellido {self.apellido} - E-Mail {self.email} - Profesión {self.profesion}"
```



¿Mucho mejor?



CODERHOUSE

CRUD – READ

CODERHOUSE

Arrancando con CRUD

Arrancamos con **READ, lectura.**
Vamos a leer todos los profesores que tenemos en nuestra base de datos.

Primero creamos la vista. 👉

```
def leerProfesores(request):

    profesores = Profesor.objects.all() #trae todos los profesores

    contexto= {"profesores":profesores}

    return render(request, "AppCoder/leerProfesores.html",contexto)
```

Arrancando con CRUD

Script:

```
def leerProfesores(request):

      profesores = Profesor.objects.all() #trae todos los profesores

      contexto= {"profesores":profesores}

      return render(request, "AppCoder/leerProfesores.html",contexto)
```

Arrancando con CRUD

👉 Luego creamos el template y lo asociamos con urls.py a la vista.

```
urlpatterns = [

    path('', views.inicio, name="Inicio"), #esta era nuestra primer view
    path('cursos', views.cursos, name="Cursos"),
    path('profesores', views.profesores, name="Profesores"),
    path('estudiantes', views.estudiantes, name="Estudiantes"),
    path('entregables', views.entregables, name="Entregables"),
    #path('cursoFormulario', views.cursoFormulario, name="CursoFormulario"),
    #path('profesorFormulario', views.profesorFormulario, name="ProfesorFormulario"),
    #path('busquedaCamada',  views.busquedaCamada, name="BusquedaCamada"),
    path('buscar/', views.buscar),
    path('leerProfesores', views.leerProfesores, name="LeerProfesores"),

]
```

URLs

```
<body>

    {% for p in profesores %}

        <li>{{p}}</li>

    {% endfor %}

</body>
```

Template

```
path('leerProfesores', views.leerProfesores,
name = "LeerProfesores")
```

CODERHOUSE

Arrancando con CRUD

Template

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>
<body>
    {% for p in profesores %}
        <li>{{p}}</li>
    {% endfor %}
</body>
</html>
```

Arrancando con CRUD

Nuestro resultado será el siguiente 👀

- Nombre: Pepe - Apellido Lopez - E-Mail nico@pepe.com - Profesión Heladero
- Nombre: Otro - Apellido Profe - E-Mail nico@lopez.com - Profesión Cazador
- Nombre: Brenda - Apellido Gonzales - E-Mail brendi@gon.com - Profesión Artista
- Nombre: Sol - Apellido Daniell - E-Mail sun@calor.com - Profesión Golfista

CRUD – CREATED

```python
def profesores(request):

    if request.method == 'POST':

        miFormulario = ProfesorFormulario(request.POST) #aqui mellega toda la información del html

        print(miFormulario)

        if miFormulario.is_valid:   #Si pasó la validación de Django

            informacion = miFormulario.cleaned_data

            profesor = Profesor (nombre=informacion['nombre'], apellido=informacion['apellido'],
             email=informacion['email'], profesion=informacion['profesion'])

            profesor.save()

            return render(request, "AppCoder/inicio.html") #Vuelvo al inicio o a donde quieran

    else:

        miFormulario= ProfesorFormulario() #Formulario vacio para construir el html

    return render(request, "AppCoder/profesores.html", {"miformulario":miFormulario})
```

```python
class ProfesorFormulario(forms.Form):
    nombre= forms.CharField(max_length=30)
    apellido= forms.CharField(max_length=30)
    email= forms.EmailField()
    profesion= forms.CharField(max_length=30)
```

Formulario - Agregar Profesor

Nombre:

Apellido:

Email:

Profesion:

Enviar

Arrancando con CRUD

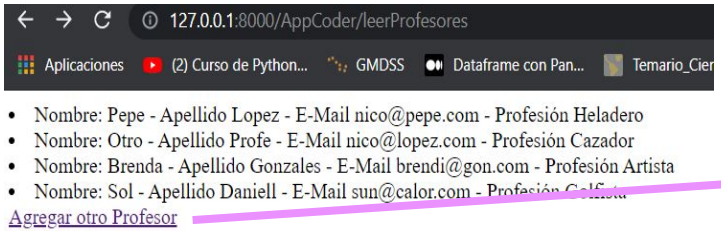
La creación ya la sabíamos hacer, es más ya la tenemos realizada por medio de un FORMS.

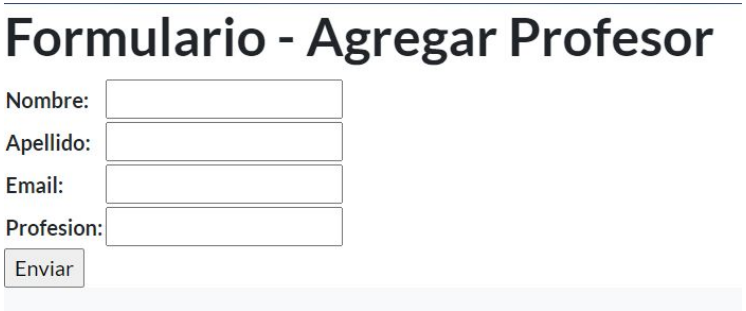
Recordando todo esto, ya lo tenemos 😎

CODERHOUSE

Arrancando con CRUD

Solo agregaremos un vínculo para poder agregar un profesor





CRUD - Delete

CODERHOUSE

Arrancando con CRUD

Pasamos a borrar, es decir DELETE

Primero creamos la vista para buscar el dato que queremos borrar

```python
def eliminarProfesor(request, profesor_nombre):

    profesor = Profesor.objects.get(nombre=profesor_nombre)
    profesor.delete()

    #vuelvo al menú
    profesores = Profesor.objects.all() #trae todos los profesores

    contexto= {"profesores":profesores}

    return render(request, "AppCoder/leerProfesores.html",contexto)
```

```html
<body>

    {% for p in profesores %}

        <li>{{p}}</li>
        <button>
            <a href="{% url 'EliminarProfesor' p.nombre %}">Eliminar</a>
        </button>

    {% endfor %}

    <a href="profesores">Agregar otro Profesor</a>

</body>
```

Arrancando con CRUD

Script

```
def eliminarProfesor(request, profesor_nombre):

    profesor = Profesor.objects.get(nombre=profesor_nombre)
    profesor.delete()

    # vuelvo al menú
    profesores = Profesor.objects.all()  # trae todos los profesores

    contexto = {"profesores": profesores}

    return render(request, "AppCoder/leerProfesores.html", contexto)
```

Arrancando con CRUD

Botón

```
<body>
    {% for p in profesores %}
        <li>{{p}}</li>
        <button>
            <a href="{% url 'EliminarProfesor' p.nombre %}"> Eliminar</a>
        </button>
    {% endfor %}
    <br>
    <a href="profesores">Agregar otro Profesor</a>
</body>
```

Arrancando con CRUD

urls.py

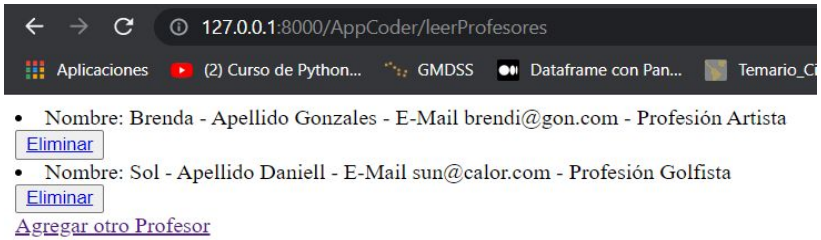
```
path('eliminarProfesor/<profesor_nombre>/', views.eliminarProfesor, name="EliminarProfesor")
```

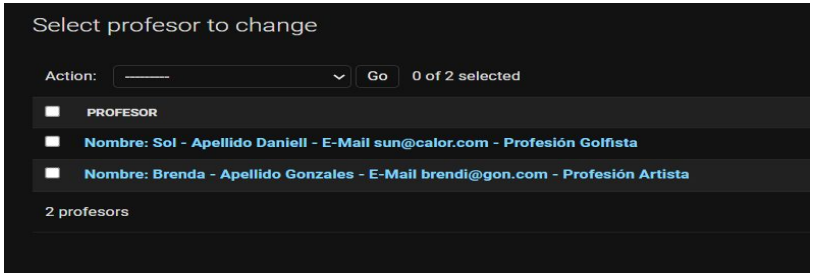
```python
urlpatterns = [

    path('', views.inicio, name="Inicio"), #esta era nuestra primer view
    path('cursos', views.cursos, name="Cursos"),
    path('profesores', views.profesores, name="Profesores"),
    path('estudiantes', views.estudiantes, name="Estudiantes"),
    path('entregables', views.entregables, name="Entregables"),
    #path('cursoFormulario', views.cursoFormulario, name="CursoFormulario"),
    #path('profesorFormulario', views.profesorFormulario, name="ProfesorFormulario"),
    #path('busquedaCamada',  views.busquedaCamada, name="BusquedaCamada"),
    path('buscar/', views.buscar),
    path('leerProfesores', views.leerProfesores, name = "LeerProfesores"),
    path('eliminarProfesor/<profesor_nombre>/', views.eliminarProfesor, name="EliminarProfesor")
]
```

Arrancando con CRUD

Apretando en el botón nuevo podemos ir borrando a los profesores

👇





CODERHOUSE

CRUD – Update

CODERHOUSE

Arrancando con CRUD

```python
def editarProfesor(request, profesor_nombre):

    #Recibe el nombre del profesor que vamos a modificar
    profesor = Profesor.objects.get(nombre=profesor_nombre)

    #Si es metodo POST hago lo mismo que el agregar
    if request.method == 'POST':

        miFormulario = ProfesorFormulario(request.POST) #aquí mellega toda la información del html

        print(miFormulario)

        if miFormulario.is_valid:   #Si pasó la validación de Django

            informacion = miFormulario.cleaned_data

            profesor.nombre = informacion['nombre']
            profesor.apellido = informacion['apellido']
            profesor.email = informacion['email']
            profesor.profesion = informacion['profesion']

            profesor.save()

            return render(request, "AppCoder/inicio.html") #Vuelvo al inicio o a donde quieran
    #En caso que no sea post
    else:
        #Creo el formulario con los datos que voy a modificar
        miFormulario= ProfesorFormulario(initial={'nombre': profesor.nombre, 'apellido':profesor.apellido ,
        'email':profesor.email, 'profesion':profesor.profesion})

    #Voy al html que me permite editar
    return render(request, "AppCoder/editarProfesor.html", {"miFormulario":miFormulario, "profesor_nombre":profesor_nombre})
```

Solo nos queda poder modificar algún dato ya existente.

Creamos la vista, **editarProfesor(request, profesor_nombre)**

Ver: Script_Update.txt

Arrancando con CRUD

Luego la url y el template son muy parecidos al agregar

```
{% for p in profesores %}

    <li>{{p}}</li>
    <button>
        <a href="{% url 'EliminarProfesor' p.nombre %}"> Eliminar</a>
        <a href="{% url 'EditarProfesor' p.nombre %}"> Editar</a>
    </button>


{% endfor %}
```

```
path('buscar/', views.buscar),
path('leerProfesores', views.leerProfesores, name="LeerProfesores"),
path('eliminarProfesor/<profesor_nombre>/', views.eliminarProfesor, name="EliminarProfesor"),
path('editarProfesor/<profesor_nombre>/', views.editarProfesor, name="EditarProfesor"),
```

CODERHOUSE

Arrancando con CRUD

Script: urls.py

```
    path('editarProfesor/<profesor_nombre>/', views.editarProfesor, name="EditarProfesor")
```

editarProfesor.html

Ver: Script_profesores.txt

CODERHOUSE

Arrancando con CRUD

Pantalla 1 💻

Pantalla 2 💻

CODERHOUSE

Arrancando con CRUD

Resultado 💪

CODERHOUSE

Break

¡10 minutos y volvemos!

Vistas basadas en Clases

Vistas basadas en Clases

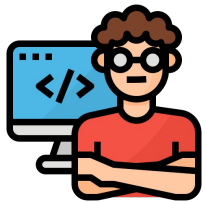
Ya aprendimos lo que es CRUD, crear, leer, modificar y borrar datos de nuestra BD sin utilizar nuestro menú de admin de Django.

👉 Fue relativamente sencillo y pudimos hacer las 4 tareas con los Profesores, por ende podríamos hacerlo con todas las clases de nuestro modelo de la misma forma.

Vistas basadas en Clases

Pero para que ese trabajo sea aún más sencillo y menos repetitivo aparece el concepto de CBV o clases basadas en vistas, o vistas basadas en clases (según traducciones).

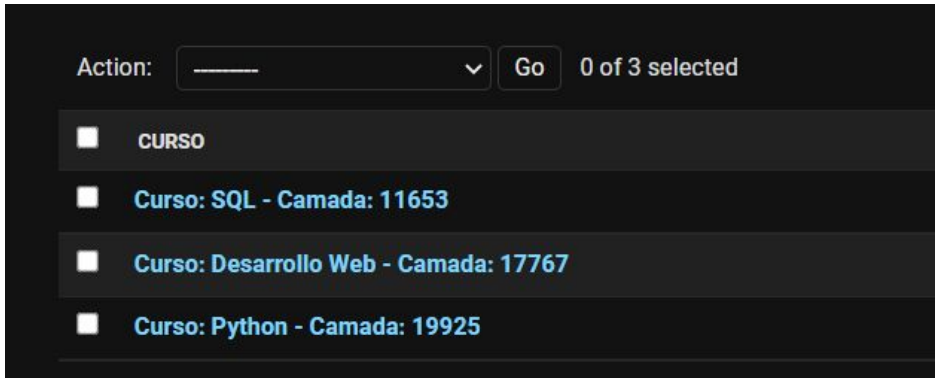
¿Listos para saber de qué se trata?



CODERHOUSE

Vistas basadas en Clases

Para entender este concepto haremos nuevamente CRUD, pero con los Cursos de la BD, así verán cómo se simplifica todo lo que hicimos anteriormente.



👉 Todos los ejemplos van a partir de que en nuestra base de datos ya tenemos cargados estos tres cursos.

CODERHOUSE

Vistas basadas en Clases: ListView

CODERHOUSE

Vistas basadas en Clases

Ahora haremos una vista que nos permita mostrar a todos los cursos.

Ya sabemos hacerlo, pero la idea es hacerlo simplificando el proceso, para eso usaremos las ListView.

Necesitamos 👇

from django.views.generic import ListView

```
class CursoList(ListView):

    model = Curso
    template_name = "AppCoder/cursos_list.html"
```

CODERHOUSE

Vistas basadas en Clases DetailView

CODERHOUSE

Clases basadas en vistas

Luego accedemos al detalle de ese **Curso**, que ya sabemos cómo hacerlo, pero podemos simplificarlo con las DetailView.

Necesitamos 👇

from django.views.generic.detail import DetailView

```
class CursoDetalle(DetailView):

    model = Curso
    template_name = "AppCoder/curso_detalle.html"
```

Clases basadas en vistas createview

CODERHOUSE

Clases basadas en vistas

Ahora pasamos a la creación del curso, usando las CreateView.

Necesitamos 👇

from django.views.generic.edit import CreateView

from django.urls import reverse_lazy

```
class CursoCreacion(CreateView):

    model = Curso
    success_url = "/AppCoder/curso/list"
    fields = ['nombre', 'camada']
```

CODERHOUSE

Clases basadas en vistas updateview

CODERHOUSE

Clases basadas en vistas

Podemos modificar con UpdateView.

Necesitamos 👇

from django.views.generic.edit import UpdateView

```
class CursoUpdate(UpdateView):

    model = Curso
    success_url = "/AppCoder/curso/list"
    fields  = ['nombre', 'camada']
```

CODERHOUSE

Clases basadas en deleteview

CODERHOUSE

Clases basadas en vistas

¿Saben que nos falta?, exacto, el DeleteView.

Necesitamos 👇

from django.views.generic.edit import DeleteView

```
class CursoDelete(DeleteView):

    model = Curso
    success_url = "/AppCoder/curso/list"
```

Clases basadas en vistas

```python
from django.views.generic import ListView
from django.views.generic.detail import DetailView
from django.views.generic.edit import CreateView, UpdateView, DeleteView
```

```python
class CursoList(ListView):

    model = Curso
    template_name = "AppCoder/cursos_list.html"


class CursoDetalle(DetailView):

    model = Curso
    template_name = "AppCoder/curso_detalle.html"


class CursoCreacion(CreateView):

    model = Curso
    success_url = "/AppCoder/curso/list"
    fields = ['nombre', 'camada']
```

Resumiendo en

view.py

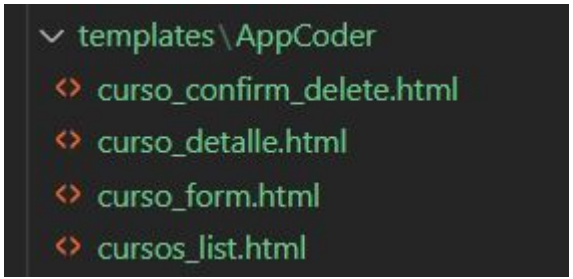
```python
class CursoUpdate(UpdateView):

    model = Curso
    success_url = "/AppCoder/curso/list"
    fields  = ['nombre', 'camada']


class CursoDelete(DeleteView):

    model = Curso
    success_url = "/AppCoder/curso/list"
```

Clases basadas en vistas URLs

CODERHOUSE

CBV - urls - template

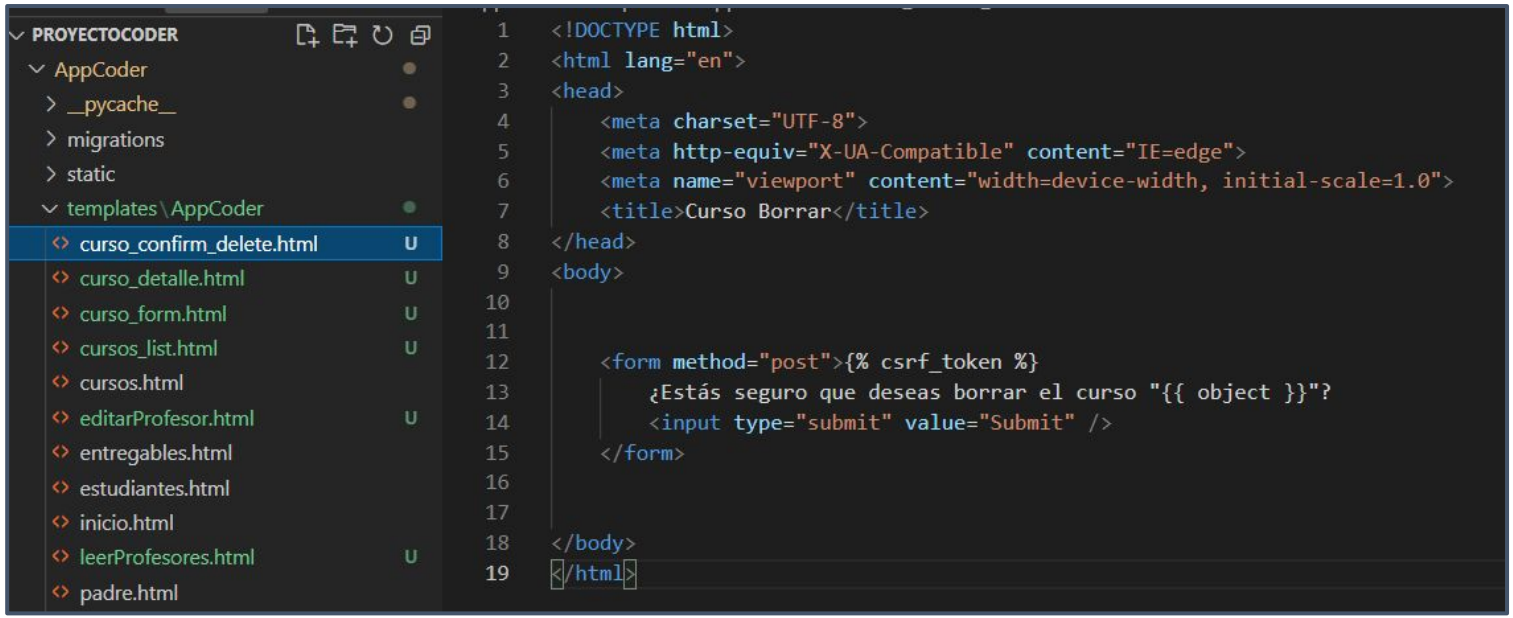
```
path('curso/list', views.CursoList.as_view(), name='List'),
path(r'^(?P<pk>\d+)$', views.CursoDetalle.as_view(), name='Detail'),
path(r'^nuevo$', views.CursoCreacion.as_view(), name='New'),
path(r'^editar/(?P<pk>\d+)$', views.CursoUpdate.as_view(), name='Edit'),
path(r'^borrar/(?P<pk>\d+)$', views.CursoDelete.as_view(), name='Delete'),
```



Clases basadas en vistas
Plantillas

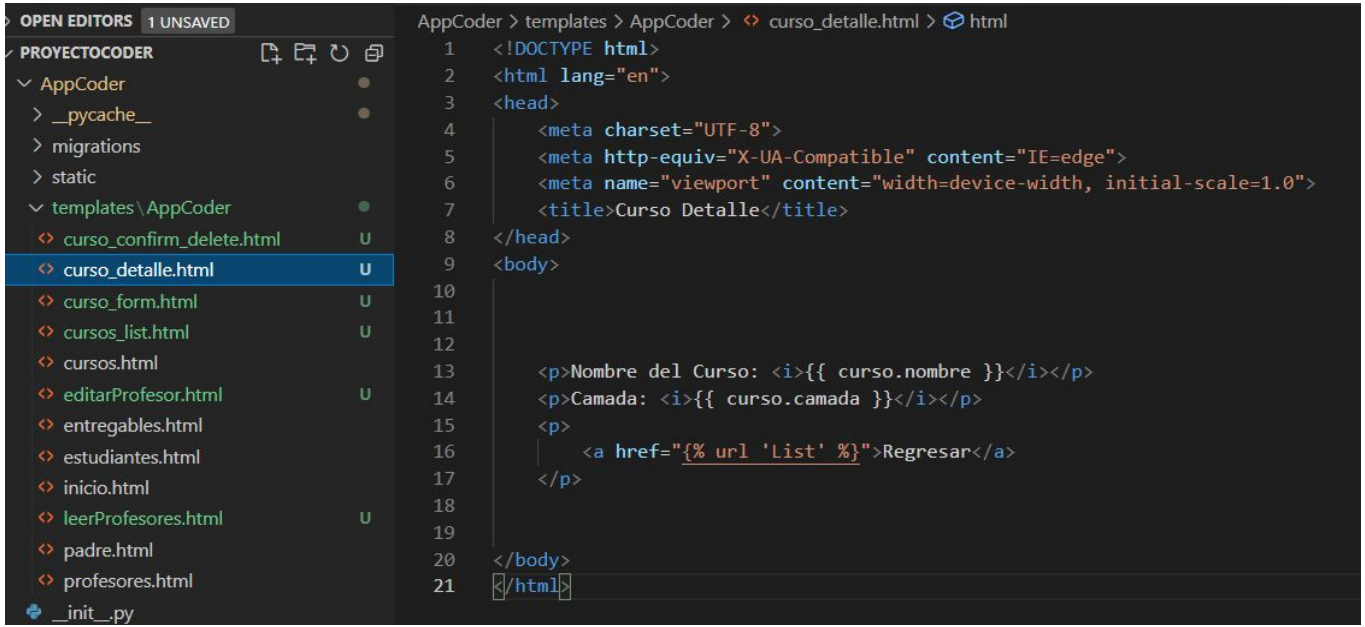
CODERHOUSE

CBV - Delete



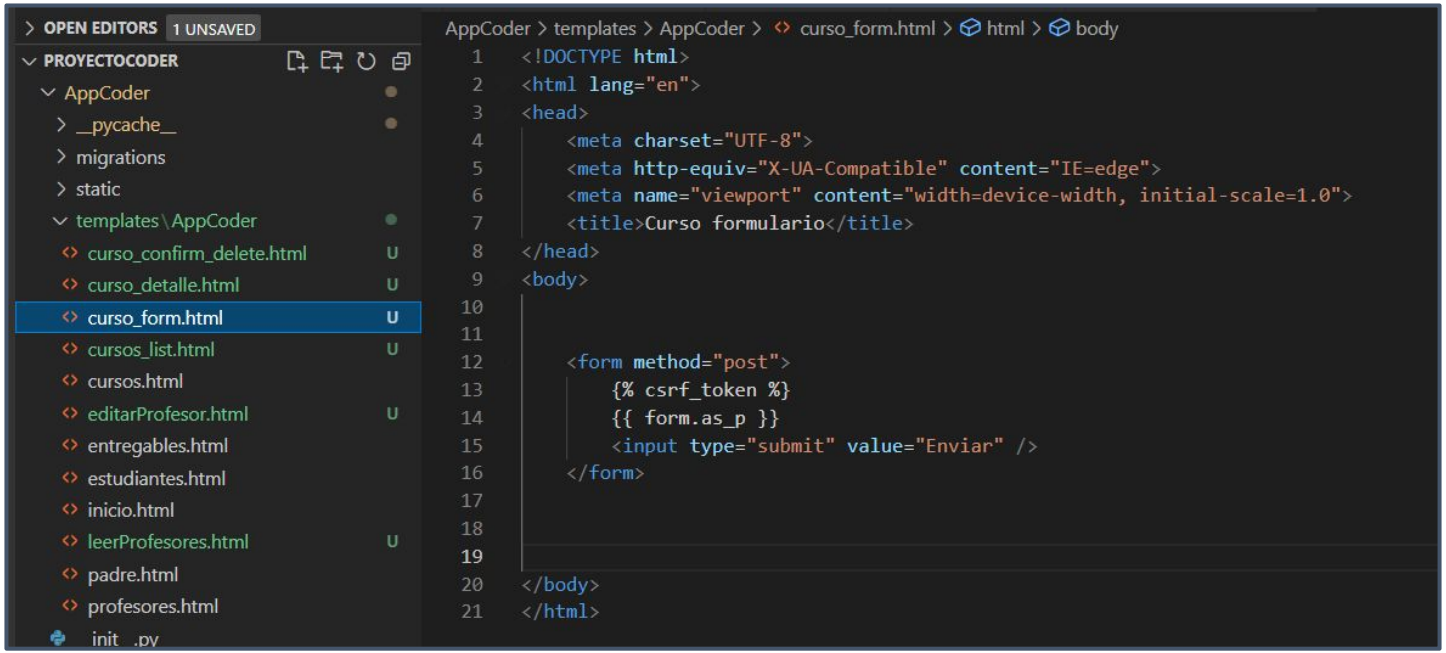
CODERHOUSE

CBV - Detalle



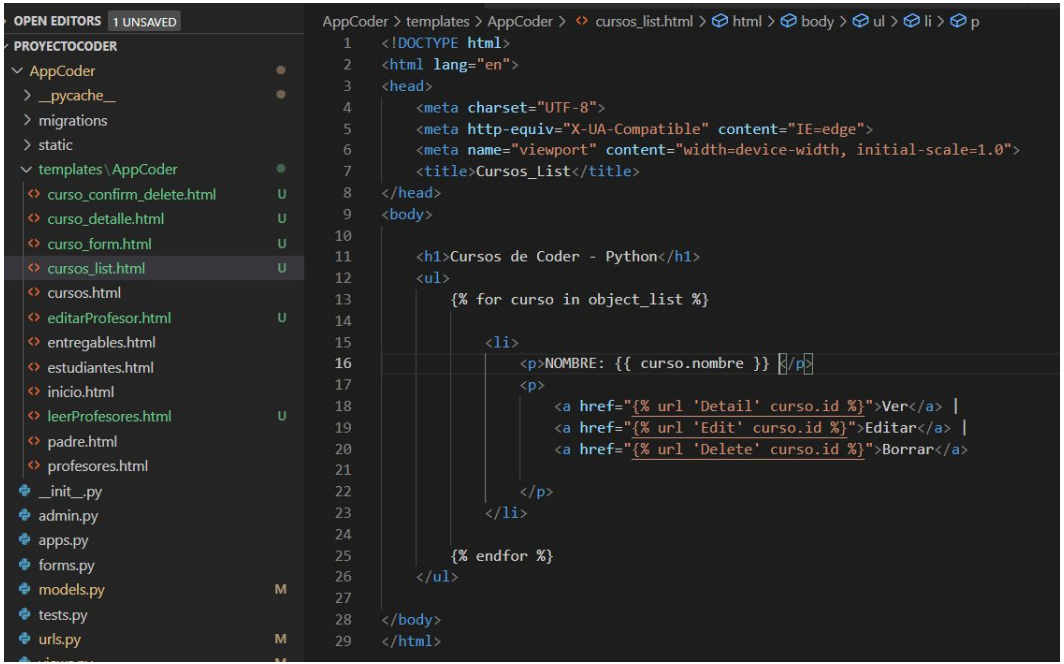
CODERHOUSE

CBV – Formulario



CODERHOUSE

CBV - Lista


OPEN EDITORS 1 UNSAVED
PROYECTOCODER
AppCoder
__pycache__
migrations
static
templates\AppCoder
curso_confirm_delete.html U
curso_detalle.html U
curso_form.html U
cursos_list.html U
cursos.html
editarProfesor.html U
entregables.html
estudiantes.html
inicio.html
leerProfesores.html U
padre.html
profesores.html
__init__.py
admin.py
apps.py
forms.py
models.py M
tests.py
urls.py M
AppCoder > templates > AppCoder > cursos_list.html > html > body > ul > li > p
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<meta http-equiv="X-UA-Compatible" content="IE=edge">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<title>Cursos_List</title>
</head>
<body>
<h1>Cursos de Coder - Python</h1>

{% for curso in object_list %}

NOMBRE: {{ curso.nombre }} {

Ver |
Editar |
Borrar

{% endfor %}

</body>
</html>

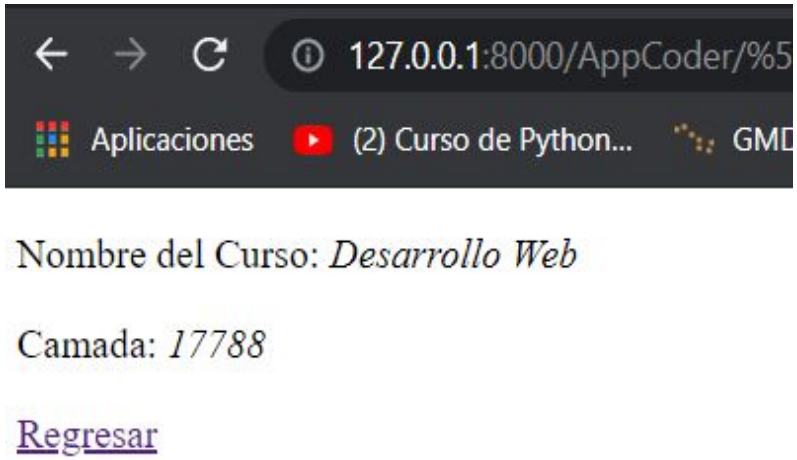
Clases basadas en vistas de acción

CODERHOUSE

Clases basadas en vistas

Vista 1 👀

Vista 2 👀

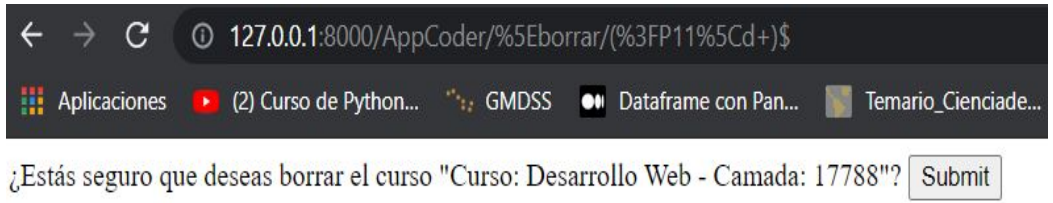


CODERHOUSE

Clases basadas en vistas

Vista 3 👀

Vista 4 👀



CODERHOUSE

Clases basadas en vistas

¡Incre**í**ble! 👋

Hicimos CRUD y FORM de una forma muy simple y abreviada.

Gracias a las vistas basadas en clases, utilizando herramientas View.

Realizar CRUD

Realizar CRUD en alguna de tus clases del trabajo que tienes en mente.

Duración: **20 minutos**

CODERHOUSE

ACTIVIDAD EN CLASE

Realizar CRUD

Pensando en una de las clases que tendrás que usar en la entrega Final, por ejemplo Blog, haz CRUD sobre ella, realizando CBV.

¿Preguntas?

Resumen
de la clase hoy

- ✓ Aprendimos qué es CRUD
- ✓ Hicimos nuestro primer CRUD
- ✓ Mejoramos nuestro CRUD con CBV

Opina y valora esta clase

Muchas gracias.

#DemocratizandoLaEducación