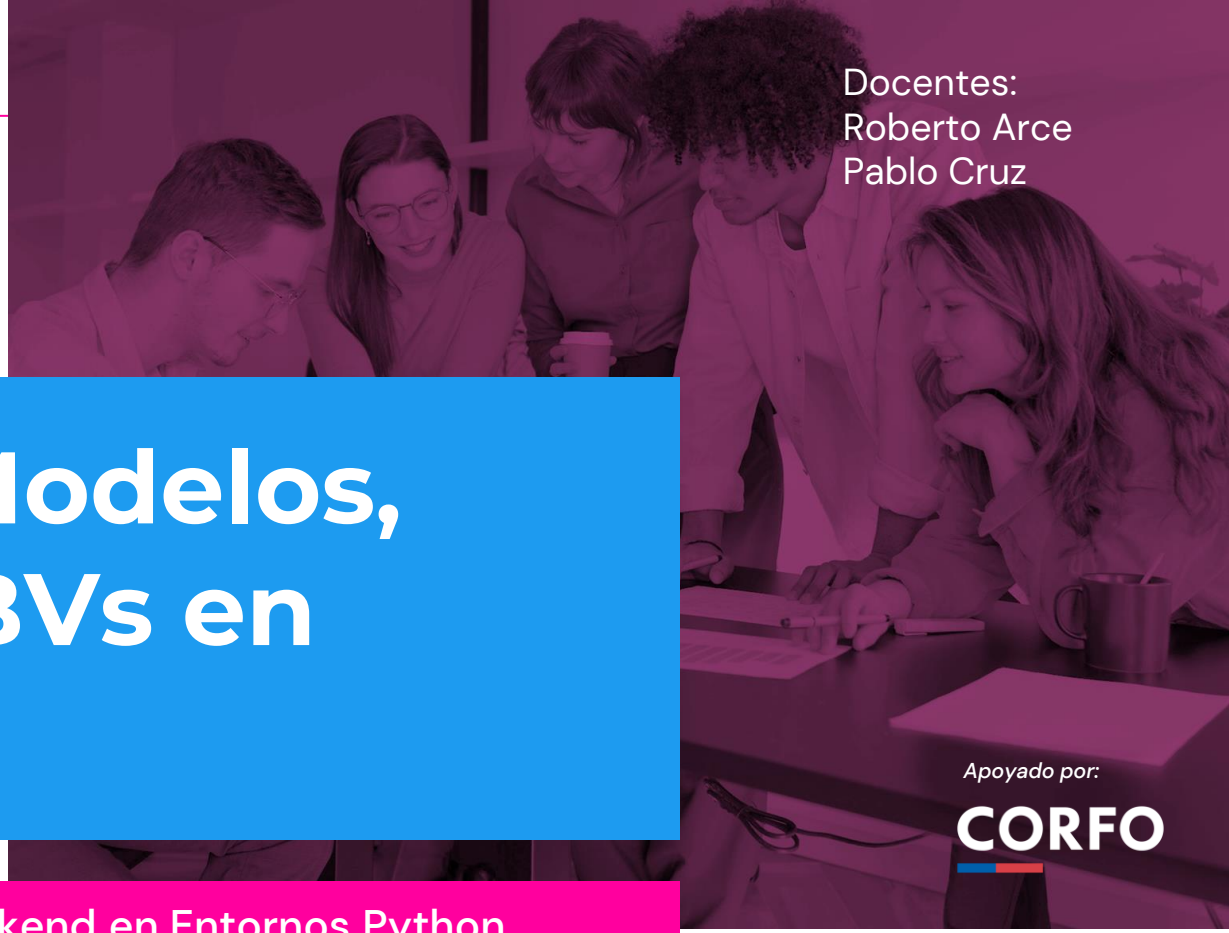
TALENTOFUTURO



Docentes:
Roberto Arce
Pablo Cruz

Clase 3: Modelos, ORM y CBVs en Django

Módulo 4: Desarrollo Backend en Entornos Python

Apoyado por:
CORFO

Clase de hoy

01

Modelos y el ORM de Django

Funciones específicas del ORM

02

Vistas basadas en Clases (CBVs)

Vistas genéricas para controlar métodos HTTP con y sin templates como respuesta.

3A

Modelos y el ORM de Django

Bloque A

Qué veremos en Bloque A

- Propósito de los modelos
- Tipos de campos del ORM
- Relaciones
- Querysets
- Consultas básicas
- Rendimiento
- CRUD (Create, Read, Update, Delete)
- Migraciones

Modelos y ORM en Django

Propósito de los Modelos

- Los **modelos** son clases Python que representan tablas de una base de datos.
- Django traduce automáticamente los modelos a tablas SQL usando migraciones.

Ejemplo básico:

```
class Producto(models.Model):
    nombre = models.CharField(max_length=100)
    precio = models.DecimalField(max_digits=10, decimal_places=2)
```

Modelos y ORM en Django

Campos en los modelos

- Los atributos de un modelo definen las columnas de una tabla en la base de datos.
- Tipos comunes:
  - CharField: Texto corto, requiere max_length.
  - DecimalField: Números decimales, requiere max_digits y decimal_places.
  - DateField: Fechas (opciones: auto_now, auto_now_add).
- Parámetros clave:
  - null: Permite valores nulos en la base de datos.
  - blank: Permite dejar el campo vacío en formularios.

Modelos y ORM en Django

Más campos en los modelos

- Campos de elección

  Choices: Campos con opciones pre-definidas.

```
ESTADOS = [
    ('ACTIVO', 'Activo'),
    ('INACTIVO', 'Inactivo'),
]
estado = models.CharField(max_length=10, choices=ESTADOS, default='ACTIVO')
```

Modelos y ORM en Django

Más campos en los modelos

- Campos específicos
  - EmailField: Valida y almacena direcciones de correo electrónico.
  - URLField: Valida y almacena URLs.

Modelos y ORM en Django

Relaciones entre modelos

- **Relaciones** permiten conectar tablas:
  - ForeignKey: Relación Uno a Mucho. Requiere opción: on_delete (acción al eliminar objeto relacionado).
  - OneToOneField: Relación Uno a Uno.
  - ManyToManyField: Relación Muchos a Muchos.

```
class Etiqueta(models.Model):
    nombre = models.CharField(max_length=100)

class Categoria(models.Model):
    nombre = models.CharField(max_length=100)

class Producto(models.Model):
    nombre = models.CharField(max_length=100)
    precio = models.DecimalField(max_digits=10, decimal_places=2)
    categoria = models.ForeignKey('Categoria', on_delete=models.CASCADE)
    etiquetas = models.ManyToManyField('Etiqueta')
```

Modelos y ORM en Django

Relaciones entre modelos (on_delete)

`on_delete` permite mantener la integridad referencial de los registros en la base de datos.
El desarrollador debe pensar en su comportamiento por cada relación que agregue.

Las opciones más comunes para usar con on_delete son:

1. CASCADE: Borra automáticamente los objetos relacionados cuando se elimina el objeto referenciado.
2. PROTECT: Evita que el objeto referenciado sea eliminado si existen objetos relacionados.
3. SET_NULL: Establece el valor del campo relacionado como NULL cuando el objeto referenciado es eliminado.

Modelos y ORM en Django

Querysets

- Un **Queryset** es una lista de objetos que representa los datos de la base de datos.
- Los querysets son generados por el ORM y permiten realizar consultas complejas.
- Métodos comunes:
    - all(): Devuelve todos los objetos.
    - filter(): Filtra objetos según condiciones.
    - exclude(): Filtra excluyendo objetos.
    - get(): Recupera un único objeto (lanza error si no es único).

```
productos = Producto.objects.all()
print(productos)  # <QuerySet [<Producto: Laptop>, <Producto: Celular>]>
```

Modelos y ORM en Django

Consultas básicas con el ORM usando querysets

- all() Obtener todos los objetos:

```
producto.objects.all()
```

- filter(): Filtrar por categoría:

```
producto.objects.filter(categoria__nombre="Electrónica")
```

- exclude(): Excluir resultados:

```
producto.objects.exclude(precio__lt=100)
```

- first(): Devuelve el primer objeto del Queryset, o None si está vacío.

```
primer_producto = Producto.objects.filter(categoria__nombre="Electrónica").first()
if primer_producto:
    print(primer_producto.nombre)
```

Modelos y ORM en Django

Consultas básicas con el ORM usando querysets

- last(): Devuelve el último objeto del queryset, o None si está vacío
- exists(): Devuelve True si el queryset contiene al menos un objeto; de lo contrario, devuelve False.
- count(): Devuelve el número de objetos en el queryset.
- order_by(): Ordena el queryset por uno o más campos.
- aggregate(): Permite realizar cálculos agregados como Sum, Avg, Max, Min sobre el queryset.

```
from django.db.models import Avg, Max

estadisticas = Producto.objects.filter(categoria__nombre="Electrónica").aggregate(
    precio_promedio=Avg('precio'),
    precio_maximo=Max('precio')
)
print(f"Precio promedio: {estadisticas['precio_promedio']}")
print(f"Precio máximo: {estadisticas['precio_maximo']}")
```

Modelos y ORM en Django

Consejo de rendimiento (importante)

- values():
  - **Descripción**: Devuelve un queryset con diccionarios en lugar de objetos.
  - **Uso común**: Obtener datos específicos sin cargar objetos completos.

```
productos = Producto.objects.filter(categoria__nombre="Electrónica").values('nombre', 'precio')
for producto in productos:
    print(producto['nombre'], producto['precio'])
```

Modelos y ORM en Django

Consejo de rendimiento (importante)

- values_list():
  - **Descripción**: Similar a values(), pero devuelve una lista de tuplas.
  - **Uso común**: Obtener datos específicos en formato simplificado.

```
productos = Producto.objects.filter(categoria__nombre="Electrónica").values_list('nombre', 'precio')
for nombre, precio in productos:
    print(nombre, precio)
```

Modelos y ORM en Django

Inserción de datos

- Crear un nuevo objeto en la base de datos

```
nueva_categoria = Categoria.objects.create(nombre="Deportes")
```

- Guardar un objeto modificado:

```
producto = Producto.objects.get(id=1)
producto.precio = 1200
producto.save()
```

Modelos y ORM en Django

Eliminación de datos

- Eliminar un objeto

```
producto = Producto.objects.get(id=1)
producto.delete()
```

- Eliminar varios objetos

```
Producto.objects.filter(precio__lt=500).delete()
```

Modelos y ORM en Django

Relaciones en Consultas

- Consultar objetos relacionados usando ForeignKey:

```
productos = Producto.objects.filter(categoria__nombre="Electrónica")
```

- Consultar objetos con ManyToManyField:

```
producto = Producto.objects.get(pk=1)
producto.etiquetas.all()
```

- Referencia inversa en ManyToManyField:

```
etiqueta = Etiqueta.objects.get(pk=1)
etiqueta.producto_set.all()
```

Modelos y ORM en Django

Migraciones

- **Django usa migraciones para convertir modelos a tablas SQL.**
- Comandos comunes:
  - python manage.py makemigrations: Crea el archivo de migración.
  - python manage.py migrate: Aplica los cambios a la base de datos.
  - python manage.py showmigrations: Muestra las migraciones aplicadas o no.

Ejemplo: Modifica un modelo y crea un nueva migración:

```
class Producto(models.Model):
    nombre = models.CharField(max_length=100)
    precio = models.DecimalField(max_digits=10, decimal_places=2)
    stock = models.IntegerField(default=0)  # Nuevo campo
```

Trabajo grupal – Bloque A

Paralelo 2

G1	G2	G3	G4
Nicolas Mardones	Víctor Meza Herrera	Daniela Méndez Gándara	Claudia Blanco
Manuel Denis	Estefania Manriquez	Álvaro Pérez	Ariel Inostroza
Bryan Castillo	Patricio Vera	Pedro Nahum	Héctor Aguayo
GERALDY SUAREZ	Oscar Torres	Javier Gajardo	Ruben Sanhueza Ramirez
Scarlett Espinoza	Braulio Quiroz	Angela Proboste Neira	Félix González
Ulises Campodónico	Yerko Gallardo	Nicolás Guzmán	Ariel Mora
Carol Leiva	Rodrigo Araya		

G5	G6	G7	G8
Fabian Díaz	Camila Oyarzún	Mayerlyn Rodriguez	Daniela Porto
Natalia Rivera	Stefanya Pulgar	Sebastian Vega	Cristian Chavez Jara
Juan Salinas	Carlos Emilio Azócar Riquelme	Efrain Duarte Campos	Juan Rodrigo Vega
Rodrigo Pastén Cortés	Nicolas Rojas	Bianel Bianchini	Rodolfo Cantillana
Flavio Jara R.	luis.paillan.cnc@gmail.com	Bastián Gamboa Labbé	Abraham Ruiz
Daniel García	Cristóbal Gajardo	Pablo Uribe	Rodrigo Álvarez

Trabajo grupal – Ejercicio #1

Crear Modelos y Relacionarlos

1. **Crea los siguientes modelos en una aplicación llamada biblioteca:**
    - **Autor**: Almacena el nombre y la fecha de nacimiento.
    - **Libro**: Contiene título, fecha de publicación y una relación Muchos a Muchos con Autor.
    - **Género**: Define un género literario con un campo de nombre único. Relaciona Libro con Género mediante una relación Uno a Muchos (ForeignKey).
2. **Realiza las migraciones necesarias para reflejar estos modelos en la base de datos.**
3. **Registra los modelos en el admin de Django, y realice algunas inserciones para probar consistencia.**

Trabajo grupal – Ejercicio #2

Consultas con el ORM (no con Django admin!)

1. **Carga datos ficticios para los modelos creados:**
    - Inserta al menos 3 géneros (Ficción, No Ficción, Ciencia Ficción).
    - Inserta 2 autores (con sus fechas de nacimiento).
    - Inserta 3 libros con los siguientes detalles: (Libro 1: Relacionado al género Ficción, escrito por ambos autores, Libro 2: Relacionado al género No Ficción, escrito por un autor, Libro 3: Relacionado al género Ciencia Ficción, escrito por un autor).
2. **Realiza las siguientes consultas utilizando el ORM:**
    - Obtener todos los libros escritos por un autor específico.
    - Contar cuántos libros pertenecen al género Ficción.
    - Obtener el título del primer libro ordenado por fecha de publicación.

Break!

3B

Vistas Basadas en Clases (CBVs)

Bloque B

Qué veremos en Bloque B

- ¿Qué son las Vistas basadas en Clases (CBVs)?
- Métodos HTTP
- Vistas genéricas (con y sin template)
- Query String

Vistas basadas en Clases (CBVs) en Django

¿Qué son las CBVs?

- Las **CBVs** son vistas implementadas como clases en lugar de funciones.
- Proporcionan una estructura reutilizable para manejar peticiones HTTP.
- Más flexibles y modulares que las vistas basadas en funciones (FBVs).

Sintáxis básica:

```
from django.views import View

class MiVista(View):
    def get(self, request):
        return HttpResponse("Hola desde una CBV") #sin template
```

Vistas basadas en Clases (CBVs) en Django

¿Qué son los Métodos HTTP?

Métodos HTTP definen cómo un cliente interactúa con el servidor. Los más comunes:

- **GET**: Recuperar datos.
- **POST**: Enviar datos (crear o procesar).
- **PUT**: Actualizar recursos existentes.
- **DELETE**: Eliminar recursos.

Uso de Métodos HTTP en CBVs

- En Django, las vistas basadas en clases (**CBVs**) permiten controlar cada método HTTP sobrescribiendo funciones como get(), post(), etc.
- Esto facilita la lectura y manipulación de datos desde los modelos.

Vistas basadas en Clases (CBVs) en Django

Ejemplo básico de vista genérica con respuesta Json

```
from django.views import View
from django.http import JsonResponse
from .models import Producto

class ProductoView(View):
    def get(self, request):
        # Recuperar todos los productos
        productos = Producto.objects.values("id", "nombre", "precio")
        return JsonResponse(list(productos), safe=False) #sin template
```

Vistas basadas en Clases (CBVs) en Django

Recepción de Parámetros por Query String en Django

- **¿Qué es un Query String?**
  - Es la parte de una URL que incluye parámetros adicionales.
  - Comienza después de ? en la URL.

- **Ejemplo:**

  `/productos/filtrar/?min_precio=1000&max_precio=2000`

- **En la vista:**

```
min_precio = request.GET.get('min_precio')  # Obtiene "1000"
max_precio = request.GET.get('max_precio')  # Obtiene "2000"
```

Vistas basadas en Clases (CBVs) en Django

Vistas genéricas con template: ListView

ListView: Muestra una lista de objetos.

- Atributos Clave:
  - Model: Modelo a listar.
  - Template_name: nombre de la Plantilla HTML.
  - Context_object_name: Nombre del objeto en la Plantilla (opcional).

```
from django.views.generic import ListView
from .models import Producto

class ProductoListView(ListView):
    model = Producto
    template_name = "lista.html"
    context_object_name = "productos"
```

Vistas basadas en Clases (CBVs) en Django

Vistas genéricas con template: DetailView

DetailView: Muestra el detalle de un objeto específico.

- Atributos Clave:
  - Model: Modelo a mostrar.
  - Template_name: Nombre de la Plantilla HTML.
  - Context_object_name: Nombre del objeto en la plantilla (opcional).

```
from django.views.generic import DetailView
from .models import Producto

class ProductoDetailView(DetailView):
    model = Producto
    template_name = "detalle.html"
    context_object_name = "producto"
```

Vistas basadas en Clases (CBVs) en Django

Configurando el enrutamiento de las vistas genéricas

```
from django.urls import path
from myapp.views import ProductoListView, ProductoDetailView

urlpatterns = [
    path('productos/', ProductoListView.as_view(), name='producto-lista'),
    path('productos/<int:pk>/', ProductoDetailView.as_view(), name='producto-detalle'),
]
```

Vistas basadas en Clases (CBVs) en Django

Template para listar los productos

```
<h1>{{ titulo }}</h1>
<ul>
    {% for producto in productos %}
        <li>
            <a href="{% url 'producto-detalle' producto.id %}">{{ producto.nombre }}</a>
        </li>
    {% endfor %}
</ul>
```

Vistas basadas en Clases (CBVs) en Django

Template para detallar un producto

```
<h1>{{ producto.nombre }}</h1>
<p>Precio: {{ producto.precio }}</p>
<p>Stock: {{ producto.stock }}</p>
```

Trabajo grupal – Bloque B

Paralelo 2

G1	G2	G3	G4
Nicolas Mardones	Víctor Meza Herrera	Daniela Méndez Gándara	Claudia Blanco
Manuel Denis	Estefania Manriquez	Álvaro Pérez	Ariel Inostroza
Bryan Castillo	Patricio Vera	Pedro Nahum	Héctor Aguayo
GERALDY SUAREZ	Oscar Torres	Javier Gajardo	Ruben Sanhueza Ramirez
Scarlett Espinoza	Braulio Quiroz	Angela Proboste Neira	Félix González
Ulises Campodónico	Yerko Gallardo	Nicolás Guzmán	Ariel Mora
Carol Leiva	Rodrigo Araya		

G5	G6	G7	G8
Fabian Díaz	Camila Oyarzún	Mayerlyn Rodriguez	Daniela Porto
Natalia Rivera	Stefanya Pulgar	Sebastian Vega	Cristian Chavez Jara
Juan Salinas	Carlos Emilio Azócar Riquelme	Efrain Duarte Campos	Juan Rodrigo Vega
Rodrigo Pastén Cortés	Nicolas Rojas	Bianel Bianchini	Rodolfo Cantillana
Flavio Jara R.	luis.paillan.cnc@gmail.com	Bastián Gamboa Labbé	Abraham Ruiz
Daniel García	Cristóbal Gajardo	Pablo Uribe	Rodrigo Álvarez

Trabajo grupal – Ejercicio #1

Construir una Vista para Listar Productos en JSON

1. Crea una vista LibroListView que:
    - Use la clase base View.
    - Maneje solicitudes HTTP GET.
    - Recupere datos del modelo Libro, incluyendo; título, género, autores asociados.
    - Devuelva los datos en formato JSON.

2. Configura la URL para la vista.

Extra: Personaliza la respuesta JSON para que incluya un mensaje como "total": <número de productos>.

Pistas:

- Usa un queryset para obtener todos los libros.
- **Itera sobre los libros y utiliza sus relaciones para construir el JSON.**
- Utiliza JsonResponse para retornar la respuesta.

Trabajo grupal – Ejercicio #2

Filtrar Libros por Género y Mostrar Autores

1. Crea una vista LibroFiltrarView que:
    - Use la clase base View.
    - Maneje solicitudes HTTP GET.
    - Recibe un parámetro genero desde la URL (query string).
    - Filtra los libros que pertenecen al género especificado.
    - Devuelve los datos en formato JSON, incluyendo; título, género, y autores asociados.
2. Configura la URL para la vista.

Pistas:

- Usa request.GET.get('genero') para obtener el valor del parámetro género.
- Filtra los libros por género.
- Itera sobre los libros para construir la respuesta JSON.
- Si no se proporciona el parámetro genero, devuelve un error JSON con un código de estado 400.

Referencias

1. Documentación Oficial:

Object-Relational Mapping (ORM): Explica los querysets, filtrados y las interacciones con la base de datos.
URL: https://docs.djangoproject.com/en/stable/topics/db/queries/

HTTP Methods in Django Views: Explica cómo manejar métodos HTTP tanto en vistas basadas en funciones como en vistas basadas en clases.
URL: https://docs.djangoproject.com/en/stable/topics/http/views/

2. Libro:

"Two Scoops of Django 3.x" by Audrey Roy Greenfeld and Daniel Roy Greenfeld
Cubre buenas prácticas en Django, incluyendo modelos, ORM y CBVs.

¿Preguntas?

¡Hemos llegado al final de la clase!

TALENTOFUTURO