Tu primera app CRUD en Django con Class Based Views

PabloTrinidadPa 0 Puntos hace 2 años

Una vista es un elemento que se encarga de tomar una petición y regresar una respuesta HTTP. Éstas no necesariamente tienen que ser una función de Python, ¡también es posible usar clases en su lugar!

El uso de clases como vistas en Django es también conocido como **Clases basadas en vistas** o **Class Based Views** y son útiles para mantener el código más legible y reutilizable. Cuando comenzamos a usar las clases basadas en vistas surge siempre la siguiente pregunta: *"¿Por qué no simplemente usar funciones y sólo copiar y pegar la lógica cada vez que lo necesite?"* Y la respuesta suele estar en la pregunta: *¿Por qué habríamos de copiar y pegar código a través de todo nuestro proyecto para hacer tareas tan cotidianas, cuando* ***la filosofía de Django siempre ha sido Don't repeat yourself (DRY)****?* Las vistas basadas en clases fueron creadas con el mismo objetivo que las vistas basadas en funciones: **hacer el desarrollo de nuestro proyecto más fácil**. Aunque no nos guste admitirlo, el desarrollo web puede volverse monótono y aburrido en ciertos casos. Las **vistas genéricas** o **generic views** fueron creadas para aliviar ese dolor (pereza); tomando tareas comunes como mostrar una lista de objetos, crear, borrar o editar un objeto de manera simple al alcance de una clase. En este artículo iremos paso a paso a través de la creación de un sitio web que nos da control completo de los cursos que Platzi tiene mediante un **CRUD** (*create*, *read*, *update and delete*).

Creación del proyecto

Comencemos creando un proyecto llamado ***platzi.***

```
django-admin startproject platzi
```

Desde la raíz de nuestro proyecto crearemos una app llamada ***courses*** para almacenar toda la información de los cursos.

```
./manage.py startapp courses
```

Agregamos *'courses'* a la variable **INSTALLED_APPS** dentro de ***platzi/settings.py***:

```
[python]
INSTALLED_APPS = (
  :
  'courses',
  :
)
[/python]
```

Creación del modelo

Para este ejemplo, consideraremos que todos los cursos constan de un *nombre*, *fecha de inicio*, *fecha de fin* y una *imagen*. Aunque sabemos que en un caso real, el modelo tendría algunas relaciones extra como a los profesores, edición, carrera, etc. El arhivo del modelo dentro de ***courses/models.py*** quedará de la siguiente forma:

```
[python]
from django.db import models


class Course(models.Model):
    name = models.CharField(max_length=140)
    start_date = models.DateTimeField()
    end_date = models.DateTimeField()
    picture = models.ImageField(upload_to='media/courses/pictures')
[/python]
```

Después de definir el modelo creamos las migraciones:

```
./manage.py makemigrations
```

Y aplicamos las migraciones para crear la tabla en la base de datos.

```
./manage.py migrate
```

Las vistas

Para listar todos los cursos en una página usaremos la vista genérica **ListView.** Esta requiere que definamos el modelo que usará:

```
[python]
from django.views.generic import ListView
from .models import Course

class CourseList(ListView):
    model = Course
[/python]
```

Para mostrar el detalle de un curso en específico usaremos **DetailView,** que también sólo requiere que sea definido el modelo:

```
[python]
from django.views.generic.detail import DetailView

class CourseDetail(DetailView):
    model = Course
[/python]
```

Para la creación de un nuevo curso usaremos **CreateView,** que requiere que sea definido el modelo, los campos que usará para la creación y la URL a la que debe redireccionar cuando la creación haya sido concluida:

```
[python]
from django.views.generic.edit import CreateView

from django.core.urlresolvers import reverse_lazy

class CourseCreation(CreateView):
    model = Course
    success_url = reverse_lazy('courses:list')
    fields = ['name', 'start_date', 'end_date', 'picture']
[/python]
```

Para editar un objeto ya existente mediante un formulario usaremos **UpdateView** que requiere el modelo, los campos y la URL de éxito:

```
[python]
from django.views.generic.edit import UpdateView
from django.core.urlresolvers import reverse_lazy

class CourseUpdate(UpdateView):
    model = Course
    success_url = reverse_lazy('courses:list')
    fields = ['name', 'start_date', 'end_date', 'picture']
[/python]
```

Por último, para borrar un objeto usaremos **DeleteView** que requiere del modelo y la URL de éxito:

```
[python]
```

```
[python]
from django.views.generic.edit import DeleteView
from django.core.urlresolvers import reverse_lazy

class CourseDelete(DeleteView):
    model = Course
    success_url = reverse_lazy('courses:list')
[/python]
```

Nuestro archivo final ***courses/views.py*** será:

```
[python]
from django.core.urlresolvers import reverse_lazy

from django.views.generic import ListView
from django.views.generic.detail import DetailView
from django.views.generic.edit import (
    CreateView,
    UpdateView,
    DeleteView
)

from .models import Course


class CourseList(ListView):
    model = Course


class CourseDetail(DetailView):
    model = Course


class CourseCreation(CreateView):
    model = Course
    success_url = reverse_lazy('courses:list')
```

```
    fields = ['name', 'start_date', 'end_date', 'picture']


class CourseUpdate(UpdateView):
    model = Course
    success_url = reverse_lazy('courses:list')
    fields = ['name', 'start_date', 'end_date', 'picture']



class CourseDelete(DeleteView):
    model = Course
    success_url = reverse_lazy('courses:list')
[/python]
```

Las URLs

Ahora debemos tener las URLs donde nuestras vistas serán llamadas. Comencemos extendiendo el módulo principal de URLs dentro de ***platzi/urls.py*** de la siguiente forma:

```
[python]
from django.conf.urls import url, include
from django.contrib import admin

urlpatterns = [
 url(r'^cursos/', include('courses.urls', namespace='courses')),
 url(r'^admin/', admin.site.urls),
]
[/python]
```

Necesitamos agregar un archivo **urls.py** dentro de **courses** de manera que el archivo ***courses/urls.py*** luzca así:

```
[python]
from django.conf.urls import url

from .views import (
    CourseList,
    CourseDetail,
    CourseCreation,
    CourseUpdate,
    CourseDelete
)

urlpatterns = [

    url(r'^$', CourseList.as_view(), name='list'),
    url(r'^(?P<pk>\d+)$', CourseDetail.as_view(), name='detail'),
    url(r'^nuevo$', CourseCreation.as_view(), name='new'),
    url(r'^editar/(?P<pk>\d+)$', CourseUpdate.as_view(), name='edit'),
    url(r'^borrar/(?P<pk>\d+)$', CourseDelete.as_view(), name='delete'),

]
[/python]
```

Templates

Por último, necesitamos crear los templates HTML. Estos siguen una convención para se nombrados y así poder ser usados de manera automática por las clases genéricas (cosa que puede ser modificada). Para listar todos los cursos usaremos la convención ***model_list.html*** por lo que el nuestro se llamará ***course_list.html*** y vivirá dentro de ***courses/templates/courses/course_list.html***:

```
[html]
<h1>Cursos de Platzi</h1>
```

```
<p>
    <a href="{% url "courses:new" %}">Agregar curso</a>
</p>

<ul>
    {% for course in object_list %}
        <li>
            <p>{{ course.name }}</p>
            <p>
                <a href="{% url "courses:detail" course.id %}">Ver</a> |
                <a href="{% url "courses:edit" course.id %}">Editar</a> |
                <a href="{% url "courses:delete" course.id %}">Borrar</a>
            </p>
        </li>
    {% endfor %}
</ul>
[/html]
```

Para agregar y editar necesitamos crear ***courses/templates/courses/course_form.html*** de la siguiente manera:

```
[html]
<form method="post">
    {% csrf_token %}
    {{ form.as_p }}
    <input type="submit" value="Submit" />
</form>
[/html]
```

Para ver el detalle necesitamos crear ***courses/templates/courses/course_detail.html***:

```
[html]
<h1>{{ course.name }}</h1>
```

```
<p>Fecha de inicio: <i>{{ course.start_date }}</i></p>
<p>Fecha de fin: <i>{{ course.end_date }}</i></p>
<p>
    <a href="{% url "courses:list" %}">Regresar</a>
</p>
[/html]
```

Por último necesitamos la pantalla de confirmación para borrar un curso que vivirá en ***courses/templates/courses/courseconfirmdelete***:

```
[html]
<form method="post">{% csrf_token %}
    ¿Estás seguro que deseas borrar el curso "{{ object }}"?
    <input type="submit" value="Submit" />
</form>
[/html]
```

Es necesario comentar que estas clases pueden ser modificadas con mucho mayor libertad para lograr cosas a la medida y más completas. ¡Toda esta *"magia"* es posible con simple herencia de clases! Si deseas consultar el código fuente da clic aquí. Y si estás en busca de una referencia completa a las clases genéricas de Django, te recomiendo mucho leer la documentación y apoyarte en este sitio web. Por otro lado, si buscas aprender más a fondo a cerca de Django en boca de los profesionales recuerda tomar el **Curso de Python y Django** de Platzi.

Entrar al curso

Pablo
@PabloTrinidadPa 0 Puntos hace 2 años **Todas sus entradas**

B I U | Insertar código | Enlace | Imagen | | |

Suma tu comentario