Introducción a CRUD (Create, Read, Update, Delete) en Django

Paulo Hernández

Objetivos

- **Codificar funciones CRUD en Django sobre una base de datos.**
- **Configurar la conexión a la base de datos en Django.**
- **Implementar validación de acceso mediante sesiones y el manejo del estado de colecciones.**

Introducción a CRUD en Django

- El CRUD es un conjunto de operaciones esenciales para interactuar con bases de datos.
  - **Create**: Crear nuevos registros.
  - **Read**: Leer o visualizar registros.
  - **Update**: Actualizar registros existentes.
  - **Delete**: Eliminar registros.

Configuración de Base de Datos en Django

- Django soporta múltiples bases de datos (SQLite, PostgreSQL, MySQL, etc.).
- La configuración de la base de datos se define en el archivo settings.py.
- Ejemplo (SQLite por defecto):

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.sqlite3',
        'NAME': BASE_DIR / 'db.sqlite3',
    }
}
```

- **Migraciones**: Django utiliza un sistema de migraciones para crear y modificar tablas en la base de datos.

Migraciones en Django

- Las **migraciones** son una forma de sincronizar los modelos de Django con la base de datos.
- Se generan automáticamente a partir de los cambios realizados en los modelos.
- Comandos clave:
  - **python manage.py makemigrations**: Prepara las migraciones en base a los modelos definidos.
  - **python manage.py migrate**: Aplica las migraciones y crea o modifica las tablas en la base de datos.

Migraciones en Django

- Entonces … flujo de migraciones:
  - Definir el modelo en models.py.
  - Ejecutar makemigrations para generar el archivo de migración.
  - Ejecutar migrate para aplicar los cambios en la base de datos.

Definir el Modelo de la Tarea

- Ejemplo del modelo Tarea en models.py

```python
from django.db import models

class Tarea(models.Model):
    titulo = models.CharField(max_length=100)
    descripcion = models.TextField()
    completada = models.BooleanField(default=False)
    fecha_creacion = models.DateTimeField(auto_now_add=True)
```

- titulo: Título de la tarea (máximo 100 caracteres).
- descripcion: Descripción detallada de la tarea.
- completada: Campo booleano para marcar si la tarea está completada.
- fecha_creacion: Guarda la fecha y hora en que la tarea fue creada.

Crear Tareas (Create)

- Se utiliza un formulario para capturar los datos.
- Ejemplo de un formulario básico para crear una tarea:

```
<form method="POST">
    {% csrf_token %}
    <label for="titulo">Título:</label>
    <input type="text" id="titulo" name="titulo" required><br><br>

    <label for="descripcion">Descripción:</label>
    <textarea id="descripcion" name="descripcion" required></textarea><br><br>

    <button type="submit">Crear Tarea</button>
</form>
```

- La vista procesa los datos y los guarda en la base de datos.

Vista para Crear una Nueva Tarea

- Esta vista manejará tanto la presentación del formulario como el procesamiento de los datos del formulario para crear una nueva tarea

```python
from django.shortcuts import render, redirect
from .models import Tarea

# Vista para crear una tarea
def crear_tarea(request):
    if request.method == 'POST':
        titulo = request.POST.get('titulo')  # Obtenemos el valor del formulario
        descripcion = request.POST.get('descripcion')
        # Creamos la tarea en la base de datos
        Tarea.objects.create(titulo=titulo, descripcion=descripcion)
        return redirect('listar_tareas')  # Redirige a la lista de tareas después de crearla
    return render(request, 'crear_tarea.html')  # Mostrar el formulario en caso de GET
```

Mostrar Tareas

- Utiliza una lista para mostrar todas las tareas almacenadas en la base de datos.
- Las tareas se obtienen de la base de datos con Tarea.objects.all()

Ejemplo de la vista y template:

```
<ul>
    {% for tarea in tareas %}
        <li>{{ tarea.titulo }} - {{ tarea.descripcion }}</li>
    {% endfor %}
</ul>
```

Vista para Listar las Tareas

```python
# Vista para listar todas las tareas
def listar_tareas(request):
    tareas = Tarea.objects.all()  # Obtiene todas las tareas de la base de datos
    return render(request, 'listar_tareas.html', {'tareas': tareas})  # Pasamos las tareas al template
```

- Esta vista obtiene todas las tareas de la base de datos y las muestra en una tabla con botones para editar y eliminar.

- **Nota:** El template mostrará las tareas en una tabla, y cada fila incluirá los botones para editar y eliminar.

Actualizar y Eliminar Tareas

- En cada fila de la lista de tareas, se agregan botones para **editar** y **eliminar** la tarea.

```
<table>
    <tr>
        <th>Tarea</th>
        <th>Acciones</th>
    </tr>
    {% for tarea in tareas %}
        <tr>
            <td>{{ tarea.titulo }}</td>
            <td>
                <a href="{% url 'actualizar_tarea' tarea.id %}">Editar</a>
                <a href="{% url 'eliminar_tarea' tarea.id %}">Eliminar</a>
            </td>
        </tr>
    {% endfor %}
</table>
```

Actualizar Tarea (Update)

- Usamos un formulario pre-llenado con los datos actuales de la tarea para actualizarla.
- **Ejemplo del formulario:**

```
<form method="POST">
    {% csrf_token %}
    <input type="text" name="titulo" value="{{ tarea.titulo }}">
    <textarea name="descripcion">{{ tarea.descripcion }}</textarea>
    <button type="submit">Actualizar</button>
</form>
```

Vista para Actualizar una Tarea

- Esta vista muestra un formulario pre-llenado con los datos actuales de la tarea y permite actualizarlos.
- **Nota:** Si la solicitud es de tipo POST, actualiza los datos. Si es GET, muestra el formulario con los valores actuales.

```python
from django.shortcuts import get_object_or_404

# Vista para actualizar una tarea existente
def actualizar_tarea(request, id):
    tarea = get_object_or_404(Tarea, id=id)  # Busca la tarea por ID
    if request.method == 'POST':
        tarea.titulo = request.POST.get('titulo')
        tarea.descripcion = request.POST.get('descripcion')
        tarea.save()  # Guardamos los cambios en la base de datos
        return redirect('listar_tareas')  # Redirige a la lista de tareas después de actualizar
    return render(request, 'actualizar_tarea.html', {'tarea': tarea})  # Mostramos el formulario pre-llenado
```

Eliminar Tareas

Se muestra un mensaje de confirmación
antes de eliminar la tarea.

```
<h2>¿Seguro que deseas eliminar la tarea "{{ tarea.titulo }}"?</h2>
<form method="POST">
    {% csrf_token %}
    <button type="submit">Eliminar</button>
</form>
```

Vista para Eliminar una Tarea

Esta vista muestra un mensaje de confirmación y elimina la tarea si el usuario lo confirma.

```python
# Vista para eliminar una tarea existente
def eliminar_tarea(request, id):
    tarea = get_object_or_404(Tarea, id=id)  # Busca la tarea por ID
    if request.method == 'POST':
        tarea.delete()  # Elimina la tarea de la base de datos
        return redirect('listar_tareas')  # Redirige a la lista de tareas después de eliminar
    return render(request, 'eliminar_tarea.html', {'tarea': tarea})  # Muestra el formulario de confirmación
```

Uso de URL Dinámicas con {% url 'nombre_de_la_vista' %}

¿Por qué usamos {% url 'nombre_de_la_vista' %} y no URLs directas?

Mantiene el código DRY (Don't Repeat Yourself): Si cambias la estructura de las URLs en urls.py, solo tienes que modificar el nombre de la vista en el archivo urls.py y **no en todos los templates** donde se usa la URL.

Mejora la legibilidad y flexibilidad: El uso del tag {% url %} te permite generar URLs dinámicamente con parámetros, como el ID de la tarea en este caso.

¿Que sucede con el ID?

En Django, **no es necesario definir manualmente el campo id** en tus modelos. Django agrega automáticamente un campo id como **clave primaria** para cada modelo si no defines una por tu cuenta.

Django automáticamente crea un campo id para cada modelo, que es un campo de tipo **AutoField**. Este campo es un entero que incrementa automáticamente y sirve como identificador único para cada registro en la base de datos.

El campo id es el **identificador único** de cada instancia del modelo y es utilizado internamente para realizar operaciones como búsquedas, actualizaciones y eliminaciones de registros

¿Qué pasa si quieres usar una clave primaria diferente?

- Si por alguna razón quieres usar un campo diferente como clave primaria (por ejemplo, un campo uuid o cualquier otro campo personalizado), puedes especificarlo con el atributo primary_key

```python
class Tarea(models.Model):
    id = models.UUIDField(primary_key=True, default=uuid.uuid4, editable=False)
    titulo = models.CharField(max_length=100)
    descripcion = models.TextField()
```

Posibles estructuras finales del proyecto en django

```
mi_proyecto/
├── manage.py
├── db.sqlite3                  # Base de datos creada por Django
├── mi_proyecto/                # Carpeta del proyecto principal
│   ├── __init__.py
│   ├── settings.py             # Configuración, incluyendo la base de datos
│   ├── urls.py                 # URLs del proyecto principal
│   └── wsgi.py
├── tareas/                     # Carpeta de la aplicación 'tareas'
│   ├── __init__.py
│   ├── admin.py
│   ├── apps.py
│   ├── models.py               # Modelo de la base de datos (Tarea)
│   ├── views.py                # Vistas CRUD
│   ├── urls.py                 # URLs propias de la aplicación
│   └── templates/              # Templates HTML de la aplicación
│       ├── crear_tarea.html
│       ├── listar_tareas.html
│       ├── actualizar_tarea.html
│       └── eliminar_tarea.html
└── static/                     # Archivos estáticos (CSS, JS)
```

```
mi_proyecto/
├── manage.py
├── db.sqlite3                  # Base de datos SQLite creada por Django
├── mi_proyecto/                # Carpeta del proyecto principal
│   ├── __init__.py
│   ├── settings.py             # Configuración del proyecto
│   ├── urls.py                 # Archivo global de URLs
│   └── wsgi.py
├── tareas/                     # Carpeta de la aplicación 'tareas'
│   ├── __init__.py
│   ├── admin.py
│   ├── apps.py
│   ├── models.py               # Modelo Tarea (definición de la base de datos)
│   ├── views.py                # Vistas para las operaciones CRUD
│   └── templates/              # Templates HTML
│       ├── crear_tarea.html    # Template para crear tareas
│       ├── listar_tareas.html  # Template para listar tareas
│       ├── actualizar_tarea.html# Template para actualizar tareas
│       └── eliminar_tarea.html # Template para eliminar tareas
└── static/                     # Archivos estáticos (CSS, JS)
```