

# **PRÁCTICA 4 – PROGRAMACIÓN WEB CON DJANGO**

## **SportsNews**

### **1. Introducción**

En esta práctica se ha desarrollado una aplicación web utilizando el framework Django, con el objetivo de aplicar de forma práctica los conocimientos adquiridos en la asignatura de Programación Web. La aplicación recibe el nombre de SportsNews y está orientada a la consulta de información deportiva organizada por deportes y partidos.

El desarrollo de este proyecto ha permitido comprender de forma más profunda cómo funciona una aplicación web completa, desde la definición de modelos de datos hasta la interacción del usuario final con la interfaz. Además, se ha trabajado con el sistema de autenticación de Django, diferenciando distintos tipos de usuarios y permisos.

La aplicación no se limita únicamente a mostrar información estática, sino que responde de forma dinámica a las acciones del usuario, mostrando contenidos diferentes según el deporte seleccionado o el tipo de usuario que accede a la plataforma.

### **2. Objetivos del proyecto**

El objetivo principal de esta práctica es adquirir experiencia práctica en el desarrollo de aplicaciones web utilizando Django. Para ello, se han definido una serie de objetivos secundarios:

- Aplicar el patrón Modelo–Vista–Template.
- Crear modelos de datos relacionados entre sí.
- Gestionar información almacenada en una base de datos.
- Diseñar vistas dinámicas que respondan a las peticiones del usuario.
- Implementar un sistema de autenticación y control de accesos.
- Diferenciar la experiencia de usuario entre usuarios normales y administradores.
- Mejorar la presentación visual de la aplicación mediante CSS.

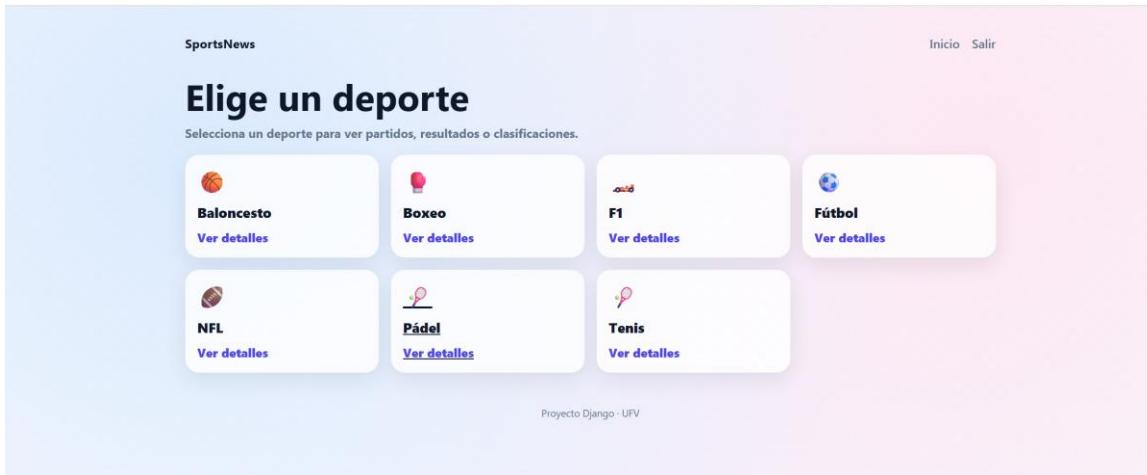
### **3. Motivación y elección de la temática**

La temática deportiva ha sido elegida por ser un ámbito ampliamente conocido y fácil de comprender por cualquier usuario. Esto permite centrarse en los aspectos técnicos del desarrollo sin que la complejidad del dominio dificulte el entendimiento del proyecto.

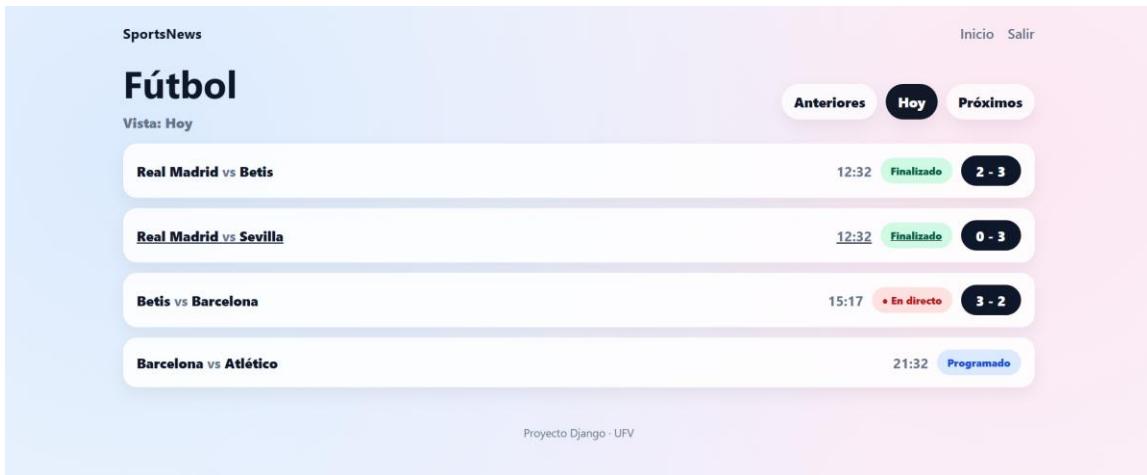
El deporte, además, ofrece múltiples posibilidades de organización de la información, como la clasificación por deportes, partidos, estados de los encuentros y seguimiento de actividad de usuarios, lo cual encaja perfectamente con los objetivos de la práctica.

## 4. Descripción general de la aplicación

SportsNews es una plataforma web que permite al usuario consultar información deportiva de distintos deportes como fútbol, baloncesto, tenis, pádel, NFL, boxeo y Fórmula 1.



Desde la página principal se muestran los deportes disponibles mediante tarjetas visuales. Cada tarjeta permite acceder a un deporte concreto. Una vez dentro, se muestran los partidos asociados, organizados por estado temporal: partidos anteriores, partidos del día y próximos eventos.



La Fórmula 1 se gestiona de forma diferenciada, mostrando una clasificación de pilotos y un calendario de grandes premios, ya que no se adapta al modelo clásico de partidos entre dos equipos.

#	PILOTO	EQUIPO	PUNTOS	GAP
1	<b>Verstappen</b>	Red Bull	<b>250</b>	
2	<b>Hamilton</b>	Ferrari	<b>232</b>	+18 pts
3	<b>Leclerc</b>	Ferrari	<b>210</b>	+40 pts
4	<b>Norris</b>	McLaren	<b>198</b>	+52 pts
5	<b>Sainz</b>	Williams	<b>170</b>	+80 pts
6	<b>Alonso</b>	Aston Martin	<b>160</b>	+90 pts
7	<b>Piastri</b>	McLaren	<b>145</b>	+105 pts
8	<b>Russell</b>	Mercedes	<b>130</b>	+120 pts
9	<b>Perez</b>	Red Bull	<b>120</b>	+130 pts
10	<b>Gasly</b>	Alpine	<b>98</b>	+152 pts

## 5. Tecnologías utilizadas

Para el desarrollo del proyecto se han utilizado las siguientes tecnologías:

- Python 3 como lenguaje principal.
- Django como framework web.
- SQLite como sistema gestor de base de datos.
- HTML5 para la estructura de las páginas.
- CSS3 para la presentación visual.
- Django Templates para el renderizado dinámico.

Estas herramientas permiten desarrollar una aplicación web completa sin necesidad de dependencias externas adicionales.

## 6. Arquitectura del Sistema

The screenshot shows a file explorer window with the following project structure:

- sports\_news** (C:\Users\oscar\Desktop\sports\_news)
  - members**
  - social**
    - .idea**
    - migrations** (highlighted)
    - templates**
      - social**
        - dashboard.html
        - login.html
        - members.html
  - models.py**
  - urls.py**
  - views.py**
- sports**
  - .idea**
  - management**
    - commands**
      - \_\_init\_\_.py
      - seed\_sports.py
      - \_\_init\_\_.py
  - migrations**
  - static**
    - sports**
      - css**
        - style.css
      - img**
  - templates**
    - sports**
      - base.html
      - f1\_ranking.html
      - home.html
      - match\_detail.html
      - sport\_detail.html

```
py __init__.py
py admin.py
py apps.py
py models.py
py urls.py
py views.py
sports_news
py __init__.py
py asgi.py
py settings.py
py urls.py
py wsgi.py
static
templates
db.sqlite3
py manage.py
```

El proyecto sigue la arquitectura Modelo–Vista–Template (MVT), propia del framework Django.

El modelo define la estructura de los datos y las relaciones entre las entidades. La vista se encarga de procesar las peticiones HTTP, aplicar la lógica de negocio y preparar los datos que serán enviados a la plantilla. El template se encarga de mostrar la información al usuario final.

Esta separación de responsabilidades facilita el mantenimiento del código y su escalabilidad.

## 7. Modelado de datos

El sistema cuenta con varios modelos principales:

- Sport: representa un deporte disponible en la plataforma.
- Match: representa un evento o partido asociado a un deporte.
- User: modelo proporcionado por Django para la gestión de usuarios.
- MemberProfile: modelo auxiliar para registrar la actividad del usuario.

## Administración de Django

Sitio administrativo

### AUTENTICACIÓN Y AUTORIZACIÓN

Grupos

 Añadir  Modificar

Usuarios

 Añadir  Modificar

### SPORTS

Matches

 Añadir  Modificar

Sports

 Añadir  Modificar

Teams

 Añadir  Modificar

La relación entre Sport y Match es de tipo uno a muchos, mientras que cada usuario cuenta con un perfil asociado para almacenar información adicional.

## 8. Gestión de usuarios y autenticación

La aplicación utiliza el sistema de autenticación integrado en Django. Los usuarios pueden iniciar sesión mediante un formulario de login.

Existen dos tipos de usuarios:

- Usuarios normales, que pueden navegar por la aplicación.
- Administradores, que además pueden acceder al panel de administración y a la vista de miembros.

<input type="checkbox"/>	NOMBRE DE USUARIO	DIRECCIÓN DE CORREO ELECTRÓNICO	NOMBRE	APELLIDOS	ES STAFF
<input type="checkbox"/>	Admin	admin@gmail.com	-	-	
<input type="checkbox"/>	Marcos	-	-	-	

El control de permisos se realiza mediante atributos del modelo User, como `is_staff`.

## 9. Vista de miembros (Members)

La vista Members es accesible únicamente para usuarios administradores. En esta vista se muestra una lista de usuarios registrados junto con el deporte que están visualizando en ese momento.

Esta funcionalidad permite simular un entorno real donde un responsable del sistema puede supervisar la actividad de los usuarios.

The screenshot shows a web application interface titled "SportsNews". At the top, there is a navigation bar with links for "Inicio", "Admin", "Members" (which is highlighted), and "Salir". The main content area has a title "Miembros" and a subtitle "Solo admins/staff. Muestra qué está viendo cada usuario.". Below this, there is a table-like list of users:

User	Viendo:	Date
Admin	Nada aún	07/01/2026 18:02
Marcos	F1	07/01/2026 20:15
oscar	Baloncesto	07/01/2026 19:44

At the bottom of the page, there is a footer note: "Proyecto Django · UFV".

## 10. Interfaz gráfica y experiencia de usuario

La interfaz ha sido diseñada para ser clara, moderna y fácil de utilizar. Se utilizan tarjetas, etiquetas visuales y colores diferenciados para mejorar la experiencia del usuario.

Los estados de los partidos se distinguen mediante colores y etiquetas, facilitando la comprensión rápida de la información mostrada.

## 11. Problemas encontrados durante el desarrollo

Durante el desarrollo del proyecto se encontraron diversos problemas técnicos, especialmente relacionados con la configuración de rutas, la gestión de templates y el control de permisos.

Algunos errores como rutas no encontradas o templates mal referenciados fueron solucionados revisando la estructura del proyecto y ajustando los nombres de vistas y plantillas.

Estos problemas han servido para comprender mejor el funcionamiento interno de Django.

## 12. Explicación del Código

### Estructura general del proyecto

El proyecto está organizado siguiendo la estructura típica de Django:

- **sports\_news/**: carpeta del proyecto (configuración global).

- settings.py: configuración (apps instaladas, templates, base de datos, etc.).
- urls.py: rutas principales del proyecto (incluye las rutas de las apps).
- **App sports/**: lógica principal de deportes y partidos.
- **App social/**: parte de usuarios y vista de miembros (solo admins).

Esta separación permite tener el proyecto ordenado: una app se centra en “el contenido” (deportes) y otra en “usuarios/actividad”.

---

### URLs (rutas) del proyecto

En Django, las rutas definen a qué vista se llama cuando el usuario entra en una URL.

- En el urls.py del proyecto se incluyen las rutas de las apps:
  - include("sports.urls") para las pantallas del contenido (home, deporte, partido...)
  - include("social.urls") para la parte de members
- Además, se definen las rutas de autenticación:
  - /login/
  - /logout/

Con esto se consigue que el navegador pueda resolver correctamente los enlaces del menú (navbar) y que el login/logout funcione sin errores.

```
from django.contrib import admin
from django.urls import path, include

urlpatterns = [
    path("admin/", admin.site.urls),
    path("", include("sports.urls")),
    path("", include("social.urls")),
]
```



## **Modelos (base de datos)**

Los modelos son las clases que representan tablas en la base de datos. En este proyecto se han utilizado principalmente:

### **1) Sport (deportes)**

Representa un deporte disponible en la plataforma.

- Campos típicos: name, slug, icon, active.
- El slug se utiliza para construir URLs limpias (ejemplo: /sport/futbol/).

### **2) Match (partidos/eventos)**

Representa un partido o evento asociado a un deporte.

- Se relaciona con Sport mediante una relación tipo **ForeignKey** (muchos partidos pertenecen a un deporte).
- Guarda información como:
  - Fecha / hora (date, time o datetime)
  - Equipos (home/away o nombres)
  - Estado (LIVE, FIN, SCH, etc.)
  - Marcadores (si aplica)

Con estos modelos, la web ya no es estática: se lee la información directamente de base de datos y se pinta en pantalla.

The screenshot shows a code editor window with a dark theme. The file is named 'models.py'. At the top, there is a warning message: 'Invalid Python interpreter selected for the project' and a 'Configure Python interpreter' button. The code itself defines three Django models: Sport, Team, and Match. The Sport model has fields for name (CharField), slug (SlugField), icon (CharField), and active (BooleanField). The Team model has a field for name (CharField). The Match model has a foreign key to Sport, two foreign keys to Team (home\_team and away\_team), and fields for date (DateField), time (TimeField), and status (CharField with choices: SCHEDULED, LIVE, FINISHED). A note at the bottom indicates that setting a default value for 'date' prevents makemigrations from failing.

```
from django.db import models
from django.utils import timezone

class Sport(models.Model):
    name = models.CharField(max_length=50)
    slug = models.SlugField(unique=True)
    icon = models.CharField(max_length=10, blank=True, default="")
    active = models.BooleanField(default=True)

    def __str__(self):
        return self.name

class Team(models.Model):
    name = models.CharField(max_length=60)

    def __str__(self):
        return self.name

class Match(models.Model):
    class Status(models.TextChoices):
        SCHEDULED = "scheduled", "Programado"
        LIVE = "live", "En directo"
        FINISHED = "finished", "Finalizado"

    sport = models.ForeignKey(Sport, on_delete=models.CASCADE, related_name="matches")
    home_team = models.ForeignKey(Team, on_delete=models.CASCADE, related_name="home_matches")
    away_team = models.ForeignKey(Team, on_delete=models.CASCADE, related_name="away_matches")

    # Esto evita tu error de makemigrations
    date = models.DateField(default=timezone.localdate)
    from datetime import time as time_obj

    time = models.TimeField(default=time_obj(hour=12, minute=0))

    status = models.CharField(max_length=20, choices=Status.choices, default=Status.SCHEDULED)
```

## Vistas (lógica de la aplicación)

Las vistas en Django son funciones (o clases) que:

1. reciben la petición del usuario,
2. consultan la base de datos,
3. preparan datos,
4. renderizan un template HTML.

### 1) home

- Consulta los deportes activos.
- Los manda al template home.html.

- Resultado: aparece la pantalla principal con tarjetas para elegir deporte.

## 2) sport\_detail

- Recibe un slug del deporte (ej: baloncesto, tenis, etc.).
- Busca el deporte con get\_object\_or\_404.
- Filtra los partidos según pestaña/filtro:
  - **Anteriores** (fechas < hoy)
  - **Hoy** (fecha = hoy)
  - **Próximos** (fechas > hoy)
- Ordena los partidos para que tengan sentido (no mezcla “en directo” con programados de otra hora).

Además, aquí se implementó algo importante:

- Si el usuario está logueado, se guarda “qué deporte está viendo” para que el admin lo pueda ver.

Esto se hace actualizando el perfil del usuario en la app social (MemberProfile).

## 3) match\_detail

- Muestra el detalle de un partido concreto (si se usa).

## 4) Vista especial de Fórmula 1

Fórmula 1 no encaja bien con “dos equipos” y un marcador típico. Por eso se hace como vista especial:

- En lugar de partidos, se muestra:
    - clasificación de pilotos
    - calendario (próximos grandes premios)
- Esto se renderiza en un template propio (por ejemplo f1\_ranking.html).

The screenshot shows a code editor interface with three tabs at the top: 'models.py', 'sport\_detail.html', and 'views.py'. The 'views.py' tab is active, displaying Python code for a sports application. A warning message 'Invalid Python interpreter selected for the project' is visible at the top of the code area. The code itself is a function named 'sport\_detail' that handles requests for sport details. It filters sports by active status, retrieves a specific sport by slug, and then processes a filter parameter ('filtro') to either get previous matches (sorted by date descending), next matches (sorted by date ascending), or today's matches (sorted by time). It also handles the 'Hoy' case. The code uses Django's ORM to query Match objects and includes logic for sorting and filtering. A status bar at the bottom right shows code statistics: 7 errors, 2 warnings, and 13 other items.

```
9 def home(request): 1 usage
10     sports = Sport.objects.filter(active=True).order_by("name")
11     return render(request, "sports/home.html", {"sports": sports})
12
13
14 def sport_detail(request, slug): 1 usage
15     sport = get_object_or_404(Sport, slug=slug, active=True)
16
17     if sport.slug == "f1":
18         return f1_ranking(request)
19
20     filtro = request.GET.get("filtro", "today")
21     today = timezone.localdate()
22
23     qs = Match.objects.filter(sport=sport)
24
25     if filtro == "prev":
26         qs = qs.filter(date__lt=today).order_by("-date", "-time")
27         title = "Anteriores"
28     elif filtro == "next":
29         qs = qs.filter(date__gt=today).order_by("date", "time")
30         title = "Próximos"
31     else:
32         qs = qs.filter(date=today)
33         # orden lógico hoy
34         status_rank = {"finished": 0, "live": 1, "scheduled": 2}
35         matches = list(qs)
36         matches.sort(key=lambda m: (status_rank.get(m.status, 9), m.time))
37         qs = matches
38         title = "Hoy"
39
40     sports_with_text = {"tenis": "tenis", "padel": "padel", "boxeo": "boxeo"}
41
42     return render(
43         request,
44         "sports/sport_detail.html",
45         {"sport": sport, "filtro": filtro, "title": title, "matches": qs, "sports_with_text": sports_with_text},
46     )
47
```

```

55     def f1_ranking(request, slug="f1"):
56         sport = get_object_or_404(Sport, slug=slug, active=True)
57
58         # Datos "mock" coherentes para la UI (como tu foto)
59         standings = [
60             {"pos": 1, "driver": "Verstappen", "team": "Red Bull", "points": 250, "gap": ""},
61             {"pos": 2, "driver": "Hamilton", "team": "Ferrari", "points": 232, "gap": "+18 pts"}, 
62             {"pos": 3, "driver": "Leclerc", "team": "Ferrari", "points": 210, "gap": "+40 pts"}, 
63             {"pos": 4, "driver": "Norris", "team": "McLaren", "points": 198, "gap": "+52 pts"}, 
64             {"pos": 5, "driver": "Sainz", "team": "Williams", "points": 170, "gap": "+80 pts"}, 
65             {"pos": 6, "driver": "Alonso", "team": "Aston Martin", "points": 160, "gap": "+90 pts"}, 
66             {"pos": 7, "driver": "Piastrini", "team": "McLaren", "points": 145, "gap": "+105 pts"}, 
67             {"pos": 8, "driver": "Russell", "team": "Mercedes", "points": 130, "gap": "+120 pts"}, 
68             {"pos": 9, "driver": "Perez", "team": "Red Bull", "points": 120, "gap": "+130 pts"}, 
69             {"pos": 10, "driver": "Gasly", "team": "Alpine", "points": 98, "gap": "+152 pts"}, 
70         ]
71
72         calendar = [
73             {"date": "22 Ene 2026", "gp": "GP Abu Dhabi", "place": "Yas Marina"}, 
74             {"date": "11 Feb 2026", "gp": "GP Japón", "place": "Suzuka"}, 
75             {"date": "02 Mar 2026", "gp": "GP España", "place": "Barcelona"}, 
76             {"date": "20 Mar 2026", "gp": "GP Francia", "place": "Paul Ricard"}, 
77         ]
78
79         return render(
80             request,
81             "sports/f1_ranking.html",
82             {"sport": sport, "standings": standings, "calendar": calendar},
83         )
84

```

## Templates (HTML) y diseño

Django usa templates para mezclar HTML con variables.

Hay un base.html que contiene:

- la estructura general,
- el estilo CSS,
- el menú de navegación.

Luego cada pantalla extiende de base:

- home.html → grid de deportes con tarjetas
- sport\_detail.html → lista de eventos/partidos con etiquetas de estado (en directo, finalizado...)
- login.html → formulario de login
- members.html → lista de usuarios y qué están viendo

Esto permite reutilizar el diseño y que la web se vea consistente.

---

## Autenticación y permisos (admin vs usuario)

Se usa el sistema de autenticación de Django.

- Usuario normal:
  - puede navegar y ver deportes
  - NO ve “Admin” ni “Members” en el menú
- Admin/staff:
  - ve “Admin” y “Members”
  - puede entrar a /members/ y ver actividad

La vista /members/ está protegida para que sólo entren administradores.

The screenshot shows a code editor with several tabs open at the top: models.py, sport\_detail.html, views.py, and base.html (which is currently selected). The base.html file contains the following code:

```
2     <html lang="es">
3     <head>
4         <style>
5             .sportIcon{ font-size:28px; }
6             .sportLink{ color:#4f46e5; font-weight:900; }
7             .sportLink:hover{ text-decoration:none; opacity:.85; }
8         </style>
9     </head>
10
11     <body>
12         <div class="wrap">
13             <div class="topbar">
14                 <div class="brand">SportsNews</div>
15
16                 <div class="nav">
17                     <a href="{% url 'home' %}">Inicio</a>
18
19                     {% if user.is_authenticated %}
20                         {% if user.is_staff %}
21                             <a class="chip" href="/admin/">Admin</a>
22                             <a class="chip" href="{% url 'members_list' %}">Members</a>
23                         {% endif %}
24
25                         <!-- Logout por POST -->
26                         <form action="{% url 'logout' %}" method="post" style="...>
27                             {% csrf_token %}
28                             <button type="submit" style="...>
29                                 Salir
30                             </button>
31                         </form>
32                     {% else %}
33                         <a href="{% url 'login' %}">Entrar</a>
34                     {% endif %}
35                 </div>
36             </div>
37
38             {% block content %}{% endblock %}
39
40             <div class="footer">Proyecto Django · UFV</div>
41         </div>
42     </body>
```

## App social (Members y actividad)

La app social se usa para registrar actividad de usuario:

### MemberProfile

Guarda información extra del usuario que no está en User:

- usuario asociado
- deporte que está viendo
- fecha/hora de última actividad

En el sport\_detail, cada vez que un usuario entra a un deporte, se actualiza el MemberProfile.

En members\_list (solo admin), se muestra:

- listado de usuarios
- deporte que está viendo cada uno
- última actualización

Esto da un extra de funcionalidad y demuestra el uso de modelos adicionales + permisos.

---

### **Seed / Generación de datos**

Para no tener que meter datos manualmente, se implementó un “seed” que crea:

- deportes
- partidos de ejemplo con distintos estados (programado, live, finalizado)

Esto facilita las pruebas y hace que la web se vea “rellena” para la entrega.

The screenshot shows a code editor with several tabs at the top: models.py, sport\_detail.html, views.py, base.html, seed\_sports.py (which is the active tab), and a configuration tab for Python interpreters. The seed\_sports.py file contains Python code for generating scores for various sports based on their slug names. It includes logic for basketball, football, NFL, tennis, padel, boxing, and a general sports section.

```
3     class Command(BaseCommand):
4         def handle(self, *args, **options):
5             def get_team(name):
6                 t, _ = Team.objects.get_or_create(name=name)
7                 return t
8
9             def make_scores(slug):
10                if slug == "baloncesto":
11                    return randint(a: 80, b: 130), randint(a: 80, b: 130), ""
12                if slug == "futbol":
13                    return randint(a: 0, b: 5), randint(a: 0, b: 5), ""
14                if slug == "nfl":
15                    return randint(a: 10, b: 38), randint(a: 10, b: 38), ""
16                if slug == "tenis":
17                    return None, None, choice([
18                        "6-3 3-6 7-6",
19                        "6-4 6-2",
20                        "7-6 6-7 6-4",
21                        "6-1 6-3",
22                    ])
23                if slug == "padel":
24                    return None, None, choice([
25                        "6-4 3-6 6-3",
26                        "6-2 6-4",
27                        "7-6 6-7 6-4",
28                        "6-3 6-3",
29                    ])
30                if slug == "boxeo":
31                    return None, None, choice([
32                        "Gana por KO (R8)",
33                        "Gana por KO (R3)",
34                        "Gana por decisión unánime",
35                        "Gana por TKO (R6)",
36                    ])
37                return randint(a: 0, b: 3), randint(a: 0, b: 3), ""
38
39             # Deportes
40             sports = [
41                 ("Baloncesto", "baloncesto", "🏀"),
42             ]
43
44             for sport in sports:
45                 team_a = get_team(sport[1])
46                 team_b = get_team(sport[1])
47                 score_a, score_b, tie = make_scores(sport[1])
48
49                 if tie:
50                     print(f'{team_a} {score_a} - {score_b} {team_b} (empate)')
51                 else:
52                     print(f'{team_a} {score_a} - {score_b} {team_b}')
53
54             print(" ")
55
56
57
58
59
59             # Deportes
60             sports = [
61                 ("Baloncesto", "baloncesto", "🏀"),
62             ]
```

## 13. Pruebas y validación

Se realizaron pruebas manuales accediendo a la aplicación con distintos tipos de usuarios, comprobando que los permisos se aplicaban correctamente y que la información se mostraba de forma adecuada.

También se verificó que la actividad del usuario se registraba correctamente al navegar por los deportes.

## 14. Conclusiones

El desarrollo de esta aplicación ha permitido consolidar conceptos fundamentales de programación web con Django y adquirir una visión más completa del desarrollo de aplicaciones web.

## **15. Posibles mejoras futuras**

Como posibles mejoras futuras se podrían añadir:

- Integración con APIs deportivas externas.
- Actualización de eventos en tiempo real.
- Sistema de favoritos para usuarios.
- Mejoras de rendimiento y escalabilidad.