



**INSTITUTO TECNOLÓGICO SUPERIOR
DE APATZINGÁN**



Reporte de Práctica

Taller de Programación Web II

Unidad 3

Profesor:

I.S.C. Javier Cisneros Lucatero

Alumnos:

Pedro Cabello Mondragón 19020085

Tarelo Cerna Ramon Antonio 19020365

Armenta Lopez Eduardo 19020292

Felix Enrique Chavez Navarro 19020270

Fecha: 25/06/2023

Índice:

| | |
|---------------------------------|-----------|
| 1 Introducción..... | 3 |
| 2 Desarrollo..... | 4 |
| 2.1 Backend..... | 4 |
| 2.1.1 Models..... | 4 |
| 2.1.1.1 Explicación Models..... | 8 |
| 2.1.2 Forms..... | 10 |
| 2.1.3 Urls..... | 13 |
| 2.1.4 Views..... | 16 |
| 2.2 Frontend..... | 22 |
| 2.2.1 Templates..... | 22 |
| 2.2.2 Static..... | 32 |
| 2.2.3 Media..... | 33 |
| 3 Conclusión..... | 33 |

1 Introducción

Esta práctica aborda el desarrollo de una aplicación de gestor de equipos de fútbol parecido al desarrollado en la unidad II de programación web II.

Se usaron todos los recursos aprendidos en el tutorial proporcionado en los recursos de la clase para elaborarlos y complementamos el desarrollo de la aplicación indagando en otras fuentes para poder crear la aplicación lo más parecida a la desarrollada en dicha unidad con el marco de trabajo odoo.

Django es un poderosísimo framework para crear aplicaciones web debido a las grandes herramientas que posee además de que te proporciona un buen ambiente de trabajo y separación de archivos para separar la lógica del negocio(backend) así como el diseño y lo visual de la aplicación web(frontend).

2 Desarrollo

El proyecto se divide en 2 partes, en el frontend (la parte visual) y en el backend (la lógica de la aplicación), un proyecto de django está conformado por varias aplicaciones pero en este proyecto solo hicimos uso de una sola aplicación.

El alcance del proyecto es solamente a nivel local, la base de datos es una relacional y también solamente funciona a nivel local.

2.1 Backend

Como lo mencionamos anteriormente el backend es la logica de la aplicación, a lo que el usuario no tiene acceso (conexiones a base de datos y la lógica del negocio), en un proyecto de django este se divide en modelos, el control de las urls y la lógica de los formularios, a continuación se explica cada punto:

2.1.1 Models

En Django, los modelos son clases de Python que representan las tablas de una base de datos relacional. Los modelos en Django se utilizan para definir la estructura y el comportamiento de los datos que serán almacenados en la base de datos.

Los modelos en Django se definen como subclases de la clase `models.Model`, que proporciona una serie de funcionalidades y métodos para interactuar con la base de datos. Cada atributo de la clase modelo representa un campo de la tabla de la base de datos.

Al definir un modelo, se especifican los tipos de campos que contendrá la tabla, como `CharField` para campos de texto, `IntegerField` para campos numéricos enteros, `DateField` para campos de fecha, etc. Además, se pueden agregar relaciones entre modelos utilizando campos como `ForeignKey` o `ManyToManyField`, lo que permite establecer relaciones uno a uno, uno a muchos o muchos a muchos entre tablas.

Models.py

```
from django.db import models

class Directores(models.Model):

    name = models.CharField(max_length=200)

    foto = models.ImageField(upload_to='directores/')

    def __str__(self):

        return self.name

class Ligas(models.Model):

    name = models.CharField(max_length=200)

    foto = models.ImageField(upload_to='ligas/')

    def __str__(self):

        return self.name

class Equipos(models.Model):

    name = models.CharField(max_length=200)

    foto = models.ImageField(upload_to='equipos/')

    directores = models.ForeignKey(Directores,
on_delete=models.SET_NULL,null=True,related_name='equipos_directores')

    ligas = models.ForeignKey(Ligas,
on_delete=models.SET_NULL,null=True,related_name='equipos_ligas')

    def __str__(self):

        return self.name

class Jugadores(models.Model):
```

```

STATE_CHOICES = [

    ('del', 'Delantero'),

    ('def', 'Defensa'),

    ('med', 'Medio Campista'),

    ('ban_der', 'Banda Derecha'),

    ('ban_izq', 'Banda Izquierda'),

    ('por', 'Portero'),

]

name = models.CharField(max_length=200)

posicion = models.CharField(max_length=200,choices=STATE_CHOICES)

num_player = models.CharField(max_length=200)

foto = models.ImageField(upload_to='jugadores/')

equipo_id = models.ForeignKey(Equipos,
on_delete=models.SET_NULL,null=True,related_name='jugadores_equipos')

def __str__(self):

    return self.name

class Arbitros(models.Model):

    name = models.CharField(max_length=200)

    liga_id =
models.ForeignKey(Ligas,on_delete=models.SET_NULL,null=True,related_name='arbitro_l
iga')

    foto = models.ImageField(upload_to='arbitros/')

    def __str__(self):

```

```

        return self.name

class Estadios(models.Model):

    name = models.CharField(max_length=200)

    liga_id = models.ForeignKey(Ligas,
on_delete=models.SET_NULL,null=True,related_name='liga_estadio')

    foto = models.ImageField(upload_to='estadios/')

    def __str__(self):

        return self.name

class Eventos(models.Model):

    player_id = models.ForeignKey(Jugadores, on_delete=models.SET_NULL,null=True)

    tipo = models.CharField(max_length=200)

    minuto = models.CharField(max_length=200)

    tarjeta = models.CharField(max_length=200)

class Juegos(models.Model):

    STATE_CHOICES = [

        ('programado', 'Programado'),

        ('terminado', 'Terminado'),

    ]

    folio = models.CharField(max_length=200)

    fecha = models.DateField()

```

```

    liga_id = models.ForeignKey(Ligas, on_delete=models.SET_NULL, null=True)

    equipo1 = models.ForeignKey(Equipos,
on_delete=models.SET_NULL, null=True, related_name='juegos_equipo1')

    equipo2 = models.ForeignKey(Equipos,
on_delete=models.SET_NULL, null=True, related_name='juegos_equipo2')

    estadio = models.ForeignKey(Estadios, on_delete=models.SET_NULL, null=True)

    arbitro = models.ForeignKey(Arbitros, on_delete=models.SET_NULL, null=True)

    state = models.CharField(max_length=20, choices=STATE_CHOICES,
default='programado')

```

2.1.1.1 Explicación Models

A continuación se explica cada modelo:

Directores:

- **name:** Es un campo de tipo CharField que representa el nombre del director. Tiene una longitud máxima de 200 caracteres.
- **foto:** Es un campo de tipo ImageField que almacena la imagen del director. Las imágenes se suben a la carpeta "directores/" en el sistema de archivos.
- **__str__(self):** Es un método especial que devuelve una representación de cadena del objeto. En este caso, retorna el nombre del director como representación de cadena. Esto se utiliza principalmente para mostrar los objetos de este modelo de manera legible en la interfaz administrativa de Django u otras representaciones de datos.

Ligas:

- **name:** Es un campo de tipo CharField que representa el nombre de la liga. Tiene una longitud máxima de 200 caracteres.
- **foto:** Es un campo de tipo ImageField que almacena la imagen de la liga. Las imágenes se suben a la carpeta "ligas/" en el sistema de archivos.

Equipos:

- **name:** Es un campo de tipo CharField que representa el nombre del equipo. Tiene una longitud máxima de 200 caracteres.
- **foto:** Es un campo de tipo ImageField que almacena la imagen del equipo. Las imágenes se suben a la carpeta "equipos/" en el sistema de archivos.

- **directores:** Es un campo de tipo ForeignKey que establece una relación con el modelo Directores. Permite relacionar un director con el equipo. Si se elimina un director, se establece como nulo (`null=True`) en el campo `directores`.
- **ligas:** Es un campo de tipo ForeignKey que establece una relación con el modelo Ligas. Permite relacionar una liga con el equipo. Si se elimina una liga, se establece como nulo (`null=True`) en el campo `ligas`.

Jugadores:

STATE_CHOICES: Es una lista de tuplas que define las opciones de selección para el campo `posicion`. Cada tupla contiene dos elementos: el valor de la opción y su representación legible.

- **name:** Es un campo de tipo CharField que representa el nombre del jugador. Tiene una longitud máxima de 200 caracteres.
- **posicion:** Es un campo de tipo CharField que almacena la posición del jugador en el campo. Las opciones se definen mediante el atributo `choices`, que utiliza la lista `STATE_CHOICES`.
- **num_player:** Es un campo de tipo CharField que almacena un número asociado al jugador. Tiene una longitud máxima de 200 caracteres.
- **foto:** Es un campo de tipo ImageField que almacena la imagen del jugador. Las imágenes se suben a la carpeta "jugadores/" en el sistema de archivos.
- **equipo_id:** Es un campo de tipo ForeignKey que establece una relación con el modelo Equipos. Permite relacionar un equipo con el jugador. Si se elimina un equipo, se establece como nulo (`null=True`) en el campo `equipo_id`.

Arbitros:

- **name:** Es un campo de tipo CharField que representa el nombre del árbitro. Tiene una longitud máxima de 200 caracteres.
- **liga_id:** Es un campo de tipo ForeignKey que establece una relación con el modelo Ligas. Permite relacionar un árbitro con una liga específica. Si se elimina una liga, se establece como nulo (`null=True`) en el campo `liga_id`.
- **foto:** Es un campo de tipo ImageField que almacena la imagen del árbitro. Las imágenes se suben a la carpeta "arbitros/" en el sistema de archivos.

Eventos:

- **player_id:** Es un campo de tipo ForeignKey que establece una relación con el modelo Jugadores. Permite relacionar un jugador con el evento. Si se elimina un jugador, se establece como nulo (`null=True`) en el campo `player_id`.

- tipo: Es un campo de tipo CharField que almacena el tipo de evento. Tiene una longitud máxima de 200 caracteres.
- minuto: Es un campo de tipo CharField que almacena el minuto en el que ocurrió el evento. Tiene una longitud máxima de 200 caracteres.
- tarjeta: Es un campo de tipo CharField que almacena información sobre una tarjeta relacionada con el evento. Tiene una longitud máxima de 200 caracteres.

Juegos:

- STATE_CHOICES: Es una lista de tuplas que define las opciones de selección para el campo state. Cada tupla contiene dos elementos: el valor de la opción y su representación legible. En este caso, las opciones son "programado" y "terminado".
- folio: Es un campo de tipo CharField que almacena un identificador o número de folio para el juego. Tiene una longitud máxima de 200 caracteres.
- fecha: Es un campo de tipo DateField que almacena la fecha del juego.
- liga_id: Es un campo de tipo ForeignKey que establece una relación con el modelo Ligas. Permite relacionar un juego con una liga específica. Si se elimina una liga, se establece como nulo (null=True) en el campo liga_id.
- equipo1 y equipo2: Son campos de tipo ForeignKey que establecen relaciones con el modelo Equipos. Permiten relacionar el juego con dos equipos participantes. Si se elimina un equipo, se establece como nulo (null=True) en los campos equipo1 y equipo2. El parámetro related_name define el nombre de la relación inversa desde el modelo Equipos.
- estadio: Es un campo de tipo ForeignKey que establece una relación con el modelo Estadios. Permite relacionar el juego con un estadio específico. Si se elimina un estadio, se establece como nulo (null=True) en el campo estadio.
- arbitro: Es un campo de tipo ForeignKey que establece una relación con el modelo Arbitros. Permite relacionar el juego con un árbitro específico. Si se elimina un árbitro, se establece como nulo (null=True) en el campo arbitro.
- state: Es un campo de tipo CharField que almacena el estado del juego. Las opciones se definen mediante el atributo choices, que utiliza la lista STATE_CHOICES. El valor predeterminado (default) es "programado".

2.1.2 Forms

En Django, los formularios son una parte fundamental de la construcción de interfaces de usuario interactivas y la recopilación de datos. Los formularios en Django se definen utilizando clases de formulario que heredan de la clase `django.forms.Form` o `django.forms.ModelForm`.

Un formulario en Django es una representación de una serie de campos y widgets que se utilizan para recopilar y procesar los datos enviados por un usuario. Los formularios permiten validar y limpiar los datos ingresados, así como mostrar mensajes de error en caso de que los datos no cumplan con los requisitos especificados.

Los formularios en Django ofrecen varias funcionalidades, entre las que se incluyen:

1. Generación de HTML: Los formularios pueden generar automáticamente el código HTML necesario para mostrar los campos y los widgets correspondientes.
2. Validación de datos: Los formularios pueden validar los datos ingresados por el usuario según las reglas y restricciones especificadas en la definición del formulario.
3. Procesamiento de datos: Los formularios pueden recibir los datos enviados por el usuario, limpiarlos y realizar cualquier procesamiento adicional necesario.
4. Asociación con modelos: Django ofrece formularios de modelo (ModelForm) que simplifican la creación de formularios basados en modelos de base de datos, donde los campos y las validaciones se generan automáticamente a partir de los campos del modelo.

```
from django import forms

from .models import Jugadores, Directores, Equipos, Ligas, Estadios, Arbitros,
Juegos

class JugadorForm(forms.ModelForm):

    class Meta:

        model = Jugadores

        fields = ('name', 'posicion', 'num_player', 'equipo_id', 'foto')

class DirectorForm(forms.ModelForm):

    class Meta:

        model = Directores

        fields = ('name', 'foto')

class EquiposForm(forms.ModelForm):

    class Meta:

        model = Equipos

        fields = ('name', 'foto', 'directores', 'ligas')

class LigasForm(forms.ModelForm):

    class Meta:

        model = Ligas

        fields = ('name', 'foto')

class EstadiosForm(forms.ModelForm):
```

```

class Meta:

    model = Estadios

    fields = ('name', 'liga_id' , 'foto')

class ArbitrosForm(forms.ModelForm):

    class Meta:

        model = Arbitros

        fields = ('name' , 'foto', 'liga_id')

class JuegosForm(forms.ModelForm):

    fecha = forms.DateField(

        label='Fecha',

        widget=forms.DateInput(attrs={'class': 'datepicker', 'type': 'date'}),

    )

    class Meta:

        model = Juegos

        fields = ('folio' , 'fecha', 'liga_id', 'equipo1', 'equipo2', 'estadio',
'arbitro', 'state')

```

Aquí está la explicación de cada formulario:

1. JugadorForm: Este formulario está relacionado con el modelo Jugadores. Los campos del formulario corresponden a los campos del modelo Jugadores: name, posicion, num_player, equipo_id y foto.
2. DirectorForm: Este formulario está relacionado con el modelo Directores. Los campos del formulario corresponden a los campos del modelo Directores: name y foto.
3. EquiposForm: Este formulario está relacionado con el modelo Equipos. Los campos del formulario corresponden a los campos del modelo Equipos: name, foto, directores y ligas.
4. LigasForm: Este formulario está relacionado con el modelo Ligas. Los campos del formulario corresponden a los campos del modelo Ligas: name y foto.
5. EstadiosForm: Este formulario está relacionado con el modelo Estadios. Los campos del formulario corresponden a los campos del modelo Estadios: name, liga_id y foto.

6. ArbitrosForm: Este formulario está relacionado con el modelo Arbitros. Los campos del formulario corresponden a los campos del modelo Arbitros: name, foto y liga_id.

7. JuegosForm: Este formulario está relacionado con el modelo Juegos. Los campos del formulario corresponden a los campos del modelo Juegos: folio, fecha, liga_id, equipo1, equipo2, estadio, arbitro y state. Además, el campo fecha tiene una configuración personalizada con un widget DateInput que muestra un calendario desplegable.

Cada formulario utiliza la clase ModelForm de Django, que automáticamente genera los campos y validaciones basados en los campos del modelo correspondiente. Esto simplifica la creación de formularios y reduce la cantidad de código necesario.

2.1.3 Urls

Las URLs en Django son manejadas por el enrutador de URL, que es una parte clave del sistema de manejo de solicitudes de Django. El enrutador de URL se encarga de analizar la URL solicitada por el usuario y determinar qué vista o función debe ser llamada para manejar esa solicitud. El enrutador de URL utiliza patrones de URL definidos en el archivo de configuración de URL para realizar esta asignación.

Las URLs en Django se utilizan para:

1. Enlazar direcciones URL a las vistas: El enrutador de URL mapea las URLs a las vistas o funciones que se encargarán de manejar la solicitud y generar una respuesta.
2. Definir la estructura de la aplicación: Las URLs proporcionan la estructura de navegación y acceso a las diferentes páginas y funcionalidades de una aplicación web.
3. Permitir el acceso a diferentes recursos: Las URLs pueden ser utilizadas para acceder a diferentes recursos, como páginas, imágenes, archivos estáticos, servicios web, entre otros.
4. Aceptar parámetros de URL: Las URLs pueden incluir parámetros que se pueden capturar y utilizar en las vistas para realizar acciones específicas o filtrar los resultados.

Las urls que contiene la aplicación son las siguientes:

```
from django.urls import path

from . import views

urlpatterns = [
```

```

path('', views.index, name='index'),

path('create_player/', views.crear_jugador, name='crear_jugador'),

path('create_director/', views.crear_director, name='crear_director'),

path('create_equipo/', views.crear_equipo, name='crear_equipo'),

path('create_liga/', views.crear_liga, name='crear_liga'),

path('create_estadio/', views.crear_estadio, name='crear_estadio'),

path('create_arbitro/', views.crear_arbitro, name='crear_arbitro'),

path('create_juego/', views.crear_juego, name='crear_juego'),

path('jugadores/', views.lista_jugadores, name='lista_jugadores'),

path('directores/', views.lista_directores, name='lista_directores'),

path('equipos/', views.lista_equipos, name='lista_equipos'),

path('ligas/', views.lista_liga, name='lista_liga'),

path('estadios/', views.lista_estadios, name='lista_estadios'),

path('arbitros/', views.lista_arbitros, name='lista_arbitros'),

path('juegos/', views.lista_juegos, name='lista_juegos'),

path('equipos/<int:id>', views.equipo_detail, name='equipo_detail'),

path('ligas/<int:id>', views.liga_detail, name='liga_detail'),

]

```

Cada objeto path especifica una URL y la función de vista que se debe ejecutar cuando se accede a esa URL. También se le asigna un nombre a cada objeto path mediante el parámetro name.

A continuación, se explica cada uno de los objetos path en el código:

1. path("", views.index, name='index'):

Esta URL vacía (") representa la página principal de la aplicación. Cuando se accede a esta URL, se ejecuta la función de vista views.index y se le asigna el nombre 'index'.

2. path('create_player/', views.crear_jugador, name='crear_jugador'):

Esta URL `create_player/` representa la página para crear un nuevo jugador. Cuando se accede a esta URL, se ejecuta la función de vista `views.crear_jugador` y se le asigna el nombre `'crear_jugador'`.

3. `path('create_director/', views.crear_director, name='crear_director'):`

Esta URL `create_director/` representa la página para crear un nuevo director. Cuando se accede a esta URL, se ejecuta la función de vista `views.crear_director` y se le asigna el nombre `'crear_director'`.

4. `path('create_equipo/', views.crear_equipo, name='crear_equipo'):`

Esta URL `create_equipo/` representa la página para crear un nuevo equipo. Cuando se accede a esta URL, se ejecuta la función de vista `views.crear_equipo` y se le asigna el nombre `'crear_equipo'`.

5. `path('create_liga/', views.crear_liga, name='crear_liga'):`

Esta URL `create_liga/` representa la página para crear una nueva liga. Cuando se accede a esta URL, se ejecuta la función de vista `views.crear_liga` y se le asigna el nombre `'crear_liga'`.

6. `path('create_estadio/', views.crear_estadio, name='crear_estadio'):`

Esta URL `create_estadio/` representa la página para crear un nuevo estadio. Cuando se accede a esta URL, se ejecuta la función de vista `views.crear_estadio` y se le asigna el nombre `'crear_estadio'`.

7. `path('create_arbitro/', views.crear_arbitro, name='crear_arbitro'):`

Esta URL `create_arbitro/` representa la página para crear un nuevo árbitro. Cuando se accede a esta URL, se ejecuta la función de vista `views.crear_arbitro` y se le asigna el nombre `'crear_arbitro'`.

8. `path('create_juego/', views.crear_juego, name='crear_juego'):`

Esta URL `create_juego/` representa la página para crear un nuevo juego. Cuando se accede a esta URL, se ejecuta la función de vista `views.crear_juego` y se le asigna el nombre `'crear_juego'`.

9. `path('jugadores/', views.lista_jugadores, name='lista_jugadores'):`

Esta URL `jugadores/` representa la página que muestra la lista de jugadores. Cuando se accede a esta URL, se ejecuta la función de vista `views.lista_jugadores` y se le asigna el nombre `'lista_jugadores'`.

10. `path('directores/', views.lista_directores, name='lista_directores'):`

Esta URL `directores/` representa la página que muestra la lista de directores. Cuando se accede a esta URL, se ejecuta la función de vista `views.lista_directores` y se le asigna el nombre `'lista_directores'`.

11. `path('equipos/', views.lista_equipos, name='lista_equipos'):`

Esta URL `equipos/` representa la página que muestra la lista de equipos. Cuando se accede a esta URL, se ejecuta la función de vista `views.lista_equipos` y se le asigna el nombre `'lista_equipos'`.

12. `path('ligas/', views.lista_liga, name='lista_liga'):`

Esta URL `ligas/` representa la página que muestra la lista de ligas. Cuando se accede a esta URL, se ejecuta la función de vista `views.lista_liga` y se le asigna el nombre `'lista_liga'`.

13. `path('estadios/', views.lista_estadios, name='lista_estadios'):`

Esta URL `estadios/` representa la página que muestra la lista de estadios. Cuando se accede a esta URL, se ejecuta la función de vista `views.lista_estadios` y se le asigna el nombre `'lista_estadios'`.

14. `path('arbitros/', views.lista_arbitros, name='lista_arbitros'):`

Esta URL `arbitros/` representa la página que muestra la lista de árbitros. Cuando se accede a esta URL, se ejecuta la función de vista `views.lista_arbitros` y se le asigna el nombre `'lista_arbitros'`.

15. `path('juegos/', views.lista_juegos, name='lista_juegos'):`

Esta URL `juegos/` representa la página que muestra la lista de juegos. Cuando se accede a esta URL, se ejecuta la función de vista `views.lista_juegos` y se le asigna el nombre `'lista_juegos'`.

16. `path('equipos/<int:id>', views.equipo_detail, name='equipo_detail'):`

Esta URL `equipos/<int:id>` representa la página de detalle de un equipo específico, donde `<int:id>` es un parámetro de URL que representa el ID del equipo. Cuando se accede a esta URL, se ejecuta la función de vista `views.equipo_detail` y se le asigna el nombre `'equipo_detail'`.

17. `path('ligas/<int:id>', views.liga_detail, name='liga_detail'):`

Esta URL `ligas/<int:id>` representa la página de detalle de una liga específica, donde `<int:id>` es un parámetro de URL que representa el ID de la liga. Cuando se accede a esta URL, se ejecuta la función de vista `views.liga_detail` y se le asigna el nombre `'liga_detail'`.

2.1.4 Views

En Django, las vistas (views) son funciones o clases que definen la lógica de cómo procesar una solicitud HTTP y generar una respuesta. Las vistas son el componente central de la capa de control en el patrón de diseño Modelo-Vista-Controlador (MVC) de Django.

La función o clase de vista es responsable de realizar las siguientes tareas:

1. Recibir la solicitud HTTP del cliente, que puede incluir información como parámetros GET o POST, datos de formularios, información de sesión, etc.
2. Realizar cualquier lógica de procesamiento requerida, como interactuar con la base de datos, realizar cálculos, procesar formularios, etc.
3. Devolver una respuesta HTTP al cliente, que puede ser una página HTML renderizada, datos en formato JSON, una redirección a otra URL, un archivo descargable, entre otros.


```

from django.shortcuts import render

from django.http import HttpResponseRedirect, JsonResponse

from django.shortcuts import get_object_or_404, render, redirect

from .forms import JugadorForm, DirectorForm, EquiposForm, LigasForm, EstadiosForm,
ArbitrosForm, JuegosForm

from .models import Jugadores, Directores, Equipos, Ligas, Estadios, Arbitros,
Juegos

def index(request):

    title = "Manager Teams"

    return render(request, 'index.html')

from .forms import JugadorForm

def crear_jugador(request):

    if request.method == 'POST':

        form = JugadorForm(request.POST, request.FILES)

        if form.is_valid():

            form.save()

            # Redirigir o realizar alguna acción adicional

    else:

        form = JugadorForm()

    return render(request, 'crear_jugador.html', {'form': form})

def crear_director(request):

    if request.method == 'POST':

        form = DirectorForm(request.POST, request.FILES)

        if form.is_valid():

            form.save()

            # Redirigir o realizar alguna acción adicional

    else:

        form = DirectorForm()

```

```
    return render(request, 'crear_director.html', {'form': form})

def crear_equipo(request):

    if request.method == 'POST':

        form = EquiposForm(request.POST,request.FILES)

        if form.is_valid():

            form.save()

            # Redirigir o realizar alguna acción adicional

        else:

            form = EquiposForm()

            return render(request, 'crear_equipo.html', {'form': form})

def crear_liga(request):

    if request.method == 'POST':

        form = LigasForm(request.POST,request.FILES)

        if form.is_valid():

            form.save()

            # Redirigir o realizar alguna acción adicional

        else:

            form = LigasForm()

            return render(request, 'crear_liga.html', {'form': form})

def crear_estadio(request):

    if request.method == 'POST':

        form = EstadiosForm(request.POST, request.FILES)

        if form.is_valid():

            form.save()

            # Redirigir o realizar alguna acción adicional

        else:

            form = EstadiosForm()

            return render(request, 'crear_estadio.html', {'form': form})
```

```

def crear_arbitro(request):

    if request.method == 'POST':

        form = ArbitrosForm(request.POST, request.FILES)

        if form.is_valid():

            form.save()

            # Redirigir o realizar alguna acción adicional

        else:

            form = ArbitrosForm()

    return render(request, 'crear_arbitro.html', {'form': form})


def crear_juego(request):

    if request.method == 'POST':

        form = JuegosForm(request.POST, request.FILES)

        if form.is_valid():

            juego = form.save(commit=False) # Obtener instancia del juego sin
guardar en la base de datos

            equipo1 = juego.equipo1

            equipo2 = juego.equipo2

            if equipo1.ligas.id != equipo2.ligas.id:

                form.add_error(None, "Los equipos no pertenecen a la misma liga.")

            else:

                juego.save() # Guardar el juego en la base de datos si pasa la
validación

        else:

            form = JuegosForm()

    return render(request, 'crear_juego.html', {'form': form})


def equipo_detail(request, id):

```

```

equipo = get_object_or_404(Equipos, id=id)

jugadores = Jugadores.objects.filter(equipo_id=id)

return render(request, 'equipo_detail.html', {

    'equipo': equipo,

    'jugadores':jugadores

})

def liga_detail(request,id):

    liga = get_object_or_404(Ligas, id=id)

    equipos = Equipos.objects.filter(ligas_id=id)

    return render(request, 'liga_detail.html', {

        'liga': liga,

        'equipos': equipos

    })

def lista_equipos(request):

    equipos = Equipos.objects.all()

    return render(request, 'equipos_list.html', {'equipos': equipos})

def lista_liga(request):

    ligas = Ligas.objects.all()

    return render(request, 'liga_list.html', {'ligas': ligas})

def lista_estadios(request):

    estadios = Estadios.objects.all()

    return render(request, 'estadios_list.html', {'estadios': estadios})

def lista_arbitros(request):

    arbitros = Arbitros.objects.all()

    return render(request, 'arbitros_list.html', {'arbitros': arbitros})

```

```
def lista_jugadores(request):

    jugadores = Jugadores.objects.all()

    return render(request, 'jugadores_list.html', {'jugadores': jugadores})

def lista_directores(request):

    directores = Directores.objects.all()

    return render(request, 'directores_list.html', {'directores': directores})

def lista_juegos(request):

    juegos = Juegos.objects.all()

    return render(request, 'juegos_list.html', {'juegos': juegos})
```

Aquí está el resumen de las funciones de vista y su funcionalidad:

`index(request):`

Esta función de vista renderiza la plantilla 'index.html' y devuelve la respuesta al cliente. Se utiliza para mostrar la página de inicio del sitio web.

`crear_jugador(request):`

Esta función de vista maneja la creación de un jugador. Si la solicitud es de tipo POST, se valida el formulario JugadorForm y se guarda el jugador en la base de datos. Si la solicitud es de otro tipo, se renderiza el formulario vacío. Se utiliza para mostrar el formulario de creación de jugadores y procesar el envío del formulario.

`crear_director(request), crear_equipo(request), crear_liga(request), crear_estadio(request), crear_arbitro(request):`

Estas funciones de vista siguen el mismo patrón que `crear_jugador(request)`, pero para la creación de directores, equipos, ligas, estadios y árbitros, respectivamente.

`crear_juego(request):`

Esta función de vista maneja la creación de un juego. Si la solicitud es de tipo POST, se valida el formulario JuegosForm y se realiza una validación adicional para asegurarse de que los equipos pertenezcan a la misma liga antes de guardar el juego en la base de datos. Si la solicitud es de otro tipo, se renderiza el formulario vacío. Se utiliza para mostrar el formulario de creación de juegos y procesar el envío del formulario.

`equipo_detail(request, id):`

Esta función de vista recupera un objeto Equipo según el ID proporcionado en la URL y obtiene todos los jugadores asociados a ese equipo. Luego, renderiza la plantilla 'equipo_detail.html' y pasa el equipo y los jugadores como contexto.

`liga_detail(request, id):`

Esta función de vista recupera un objeto Liga según el ID proporcionado en la URL y obtiene todos los equipos asociados a esa liga. Luego, renderiza la plantilla 'liga_detail.html' y pasa la liga y los equipos como contexto.

`lista_equipos(request)`, `lista_liga(request)`, `lista_estadios(request)`, `lista_arbitros(request)`, `lista_jugadores(request)`, `lista_directores(request)`, `lista_juegos(request)`:

Estas funciones de vista recuperan todos los objetos correspondientes a los modelos Equipos, Ligas, Estadios, Arbitros, Jugadores, Directores y Juegos, respectivamente, y los pasan como contexto para ser renderizados en sus respectivas plantillas.

En general, estas funciones de vista se encargan de recibir las solicitudes del cliente, realizar la lógica de procesamiento necesaria, como la validación de formularios y la interacción con la base de datos, y devolver una respuesta al cliente, que puede ser una página HTML renderizada con datos dinámicos o una redirección a otra URL.

2.2 Frontend

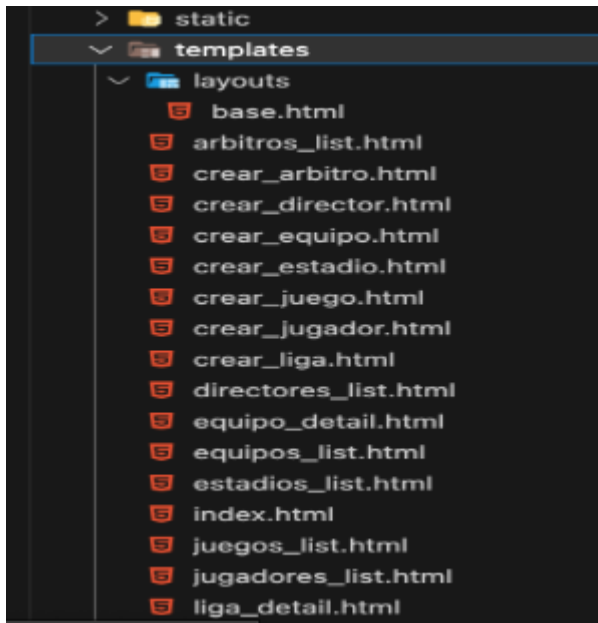
El frontend es la parte visual de la aplicación, la parte con la que el usuario interactúa y en donde se muestra el funcionamiento de la aplicación.

2.2.1 Templates

Los templates en Django son archivos de texto que contienen marcadores y etiquetas especiales que permiten generar contenido dinámico para las respuestas que se envían al cliente. Los templates son una parte fundamental del patrón de diseño Modelo-Vista-Template (MVT) en Django.

En un template, puedes combinar HTML estático con código Python y variables dinámicas. Estas variables son proporcionadas por el contexto que se pasa al renderizar el template. Los templates permiten la separación clara entre el contenido estático y el contenido generado dinámicamente, lo que facilita la creación y el mantenimiento de las interfaces de usuario.

La estructura de los templates son la siguientes:



La lógica de los archivos html es la misma para varios archivos motivo por el cual solo se explicarán algunos para que la información no sea redundante, por ejemplo en los archivos que dicen “crear_algo” se implementa los formularios creados anteriormente y su lógica es la misma, y en los archivos “algo_list” muestra las cosas que sean creado anteriormente (jugadores, estadios, equipos, directores, etc), y en los archivos “algo_detail” se utilizan para algun detalle acerca de la liga o de los equipos.

22En archivos base.html es el esqueleto de la aplicación, ahí se define como se van a mostrar los componentes en el navegador y es un archivo reutilizable que se invoca en varios archivos.

```
{% load static %}

<!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>

    <link rel="stylesheet" href="{% static '/styles/main.css' %}">
```

```

</head>

<body>

  <nav class="navbar">

    <div class="dropdown">

      <button class="dropbtn">Ligas</button>

      <div class="dropdown-content">

        <a href="{% url 'lista_jugadores' %}">Jugadores</a>

        <a href="{% url 'lista_equipos' %}">Equipos</a>

        <a href="{% url 'lista_directores' %}">Directores</a>

        <a href="{% url 'lista_liga' %}">Ver Ligas</a>

      </div>

    </div>

    <div class="dropdown">

      <button class="dropbtn">Estadios</button>

      <div class="dropdown-content">

        <a href="{% url 'lista_juegos' %}">Juegos</a>

        <a href="{% url 'lista_arbitros' %}">Arbitros</a>

        <a href="{% url 'lista_estadios' %}">Ver Estadios</a>

      </div>

    </div>

  </nav>

  <main class="container">

    {% block content %}

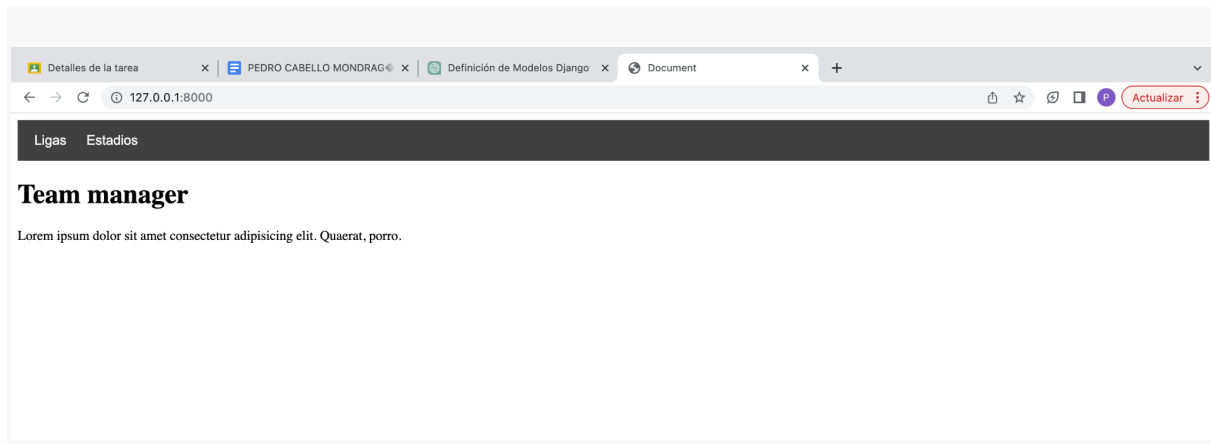
    {% endblock %}

  </main>

</body>

</html>

```

{% load static %}: Esta etiqueta carga la funcionalidad de archivos estáticos en el template, lo cual permite utilizar archivos estáticos como hojas de estilo CSS, archivos JavaScript o imágenes, acerca de los archivos estáticos se hablará más adelante.

<!DOCTYPE html> ... </html>: Esto define la estructura básica de un documento HTML.

<head> ... </head>: Esta sección contiene información y metadatos sobre el documento HTML, como el título de la página y las etiquetas meta.

<link rel="stylesheet" href="{% static '/styles/main.css' %}">: Esta línea enlaza el archivo CSS ubicado en la ruta especificada por {% static ... %}. La función {% static %} es utilizada para generar la URL completa del archivo estático.

<body> ... </body>: Aquí se encuentra el contenido principal de la página.

<nav class="navbar"> ... </nav>: Este bloque representa la barra de navegación de la página.

{% url 'nombre_url' %}: Estas etiquetas generan las URLs correspondientes a las vistas definidas en las urls.py de la aplicación. El valor 'nombre_url' debe coincidir con el nombre definido en la función name de la URL correspondiente.

<main class="container"> ... </main>: Este bloque representa el contenido principal de la página y se encuentra dentro de un contenedor con la clase "container". El contenido dentro de este bloque puede ser personalizado en los templates hija mediante el uso de la etiqueta {% block content %} ... {% endblock %}.

A continuación explicaré unos de los archivos para crear algo:

```
{% extends './layouts/base.html' %}

{% block content%}

{% load static %}
```

```

<head>

    <link rel="stylesheet" type="text/css" href="{% static 'styles/form_styles.css'
%}">

</head>

<h1>Crear Jugador</h1>

<form method="post" enctype="multipart/form-data">

    {% csrf_token %}

    {{ form.as_p }}

    <button type="submit">Guardar</button>

</form>

{% endblock %}

```

{% extends './layouts/base.html' %}:

Esta línea indica que este template está extendiendo o heredando del template "base.html". Esto significa que el contenido de "base.html" se utilizará como el diseño base y se puede personalizar o agregar contenido adicional en este template.

{% block content%} ... {% endblock %}:

Esta etiqueta define un bloque llamado "content" que se puede llenar o sobrescribir en el template "base.html". En este bloque, se coloca el contenido específico de este template.

{% load static %}:

Esta etiqueta carga la funcionalidad de archivos estáticos en el template, lo cual permite utilizar archivos estáticos como hojas de estilo CSS, archivos JavaScript o imágenes.

`<head> ... </head>`:

Esta sección contiene información y metadatos sobre el documento HTML, como enlaces a hojas de estilo.

`<link rel="stylesheet" type="text/css" href="{% static 'styles/form_styles.css' %}">`:

Esta línea enlaza el archivo CSS ubicado en la ruta especificada por `{% static ... %}`. La función `{% static %}` es utilizada para generar la URL completa del archivo estático.

`<h1>Crear Jugador</h1>`:

Esto muestra un encabezado "Crear Jugador" en el contenido del template.

`<form method="post" enctype="multipart/form-data"> ... </form>`:

Aquí se encuentra un formulario HTML que utiliza el método POST para enviar los datos. El atributo `enctype="multipart/form-data"` es necesario cuando se envían archivos en el formulario.

`{% csrf_token %}`:

Esta etiqueta genera un campo de token CSRF que es necesario para proteger el formulario contra ataques CSRF (Cross-Site Request Forgery).

`{{ form.as_p }}`:

Esta línea renderiza el formulario form en el template. El método `.as_p` muestra los campos del formulario como párrafos.

`<button type="submit">Guardar</button>`:

Este es un botón de envío del formulario que se utiliza para enviar los datos del formulario al servidor.

A continuación explicaré uno de los archivos para mostrar la información que se ha creado a través de los formularios:

```
{% extends './layouts/base.html' %}

{% block content%}

{% load static %}

<head>

    <link rel="stylesheet" type="text/css" href="{% static 'styles/list_player.css'
%}">

</head>
```

```

<h1>Lista de Jugadores</h1>

<a class="boton-crear" href="{% url 'crear_jugador' %}">Crear Jugador</a>

<table>

    <thead>

        <tr>

            <th>Nombre</th>

            <th>Posición</th>

            <th>Número de Jugador</th>

            <th>Foto</th>

            <th>Equipo</th>

        </tr>

    </thead>

    <tbody>

        {% for jugador in jugadores %}

        <tr>

            <td>{{ jugador.name }}</td>

            <td>{{ jugador.get_posicion_display }}</td>

            <td>{{ jugador.num_player }}</td>

            <td></td>

            <td><a href="{% url 'equipo_detail' jugador.equipo_id.id %}">{{
jugador.equipo_id }}</a></td>

        </tr>




        {% endfor %}

```

```
</tbody>

</table>

{% endblock %}
```

| Nombre | Posición | Número de Jugador | Foto | Equipo |
|--------------------|----------|-------------------|--|--|
| Miguel Jimenez | Portero | 23 |  | Club Deportivo Guadalajara |
| Jesus Orozco | Defensa | 13 |  | Club Deportivo Guadalajara |
| Gilberto Sepulveda | Defensa | 3 |  | Club Deportivo Guadalajara |

{% extends './layouts/base.html' %}: Esta línea indica que este template está extendiendo o heredando del template "base.html". Esto significa que el contenido de "base.html" se utilizará como el diseño base y se puede personalizar o agregar contenido adicional en este template.

{% block content%} ... {% endblock %}: Esta etiqueta define un bloque llamado "content" que se puede llenar o sobrescribir en el template "base.html". En este bloque, se coloca el contenido específico de este template.

{% load static %}: Esta etiqueta carga la funcionalidad de archivos estáticos en el template, lo cual permite utilizar archivos estáticos como hojas de estilo CSS, archivos JavaScript o imágenes.

<head> ... </head>: Esta sección contiene información y metadatos sobre el documento HTML, como enlaces a hojas de estilo.

<link rel="stylesheet" type="text/css" href="{% static 'styles/list_player.css' %}">: Esta línea enlaza el archivo CSS ubicado en la ruta especificada por {% static ... %}. La función {% static %} es utilizada para generar la URL completa del archivo estático.

<h1>Lista de Jugadores</h1>: Esto muestra un encabezado "Lista de Jugadores" en el contenido del template.

`Crear Jugador`: Este es un enlace que redirige a la URL especificada por `{% url 'crear_jugador' %}`. El texto "Crear Jugador" se muestra como un botón con una clase CSS "boton-crear".

`<table> ... </table>`: Aquí se encuentra una tabla HTML que se utiliza para mostrar la lista de jugadores.

`<thead> ... </thead>`: Esta sección define el encabezado de la tabla.

`<tbody> ... </tbody>`: Esta sección contiene el cuerpo de la tabla donde se mostrarán los datos de los jugadores.

`{% for jugador in jugadores %} ... {% endfor %}`: Este bucle for itera sobre la variable jugadores, que contiene una lista de jugadores. Por cada jugador en la lista, se renderiza el contenido dentro del bucle.

`<tr> ... </tr>`: Esto representa una fila en la tabla.

`<td>{{ jugador.name }}</td>`: Esta celda muestra el nombre del jugador.

`<td>{{ jugador.get_posicion_display }}</td>`: Esta celda muestra la posición del jugador utilizando el método `get_posicion_display`, que devuelve la representación legible por humanos de la opción seleccionada en el campo de posición del modelo.

`<td>{{ jugador.num_player }}</td>`: Esta celda muestra el número de jugador.

`<td></td>`: Esta celda muestra la foto del jugador utilizando la URL de la imagen almacenada en el campo foto del modelo.

`<td>{{ jugador.equipo_id }}</td>`: Esta celda contiene un enlace que redirige a la página de detalle del equipo correspondiente al jugador. El texto del enlace muestra el nombre del equipo.

A continuación explicaré alguno de los archivos para saber más información acerca de los equipos, ligas o estadios:

```
{% extends './layouts/base.html' %}

{% block content%}

{% load static %}

<head>

    <link rel="stylesheet" type="text/css" href="{% static 'styles/list_player.css' %}">

</head>

<h1>{{ equipo.name }}</h1>
```

```



<table>

  <thead>

    <tr>

      <th>Nombre</th>

      <th>Posición</th>

      <th>Número de Jugador</th>

      <th>Foto</th>

    </tr>

  </thead>

  <tbody>

    {% for jugador in jugadores %}

    <tr>

      <td>{{ jugador.name }}</td>

      <td>{{ jugador.posicion }}</td>

      <td>{{ jugador.num_player }}</td>

      <td></td>

    </tr>

    {% endfor %}

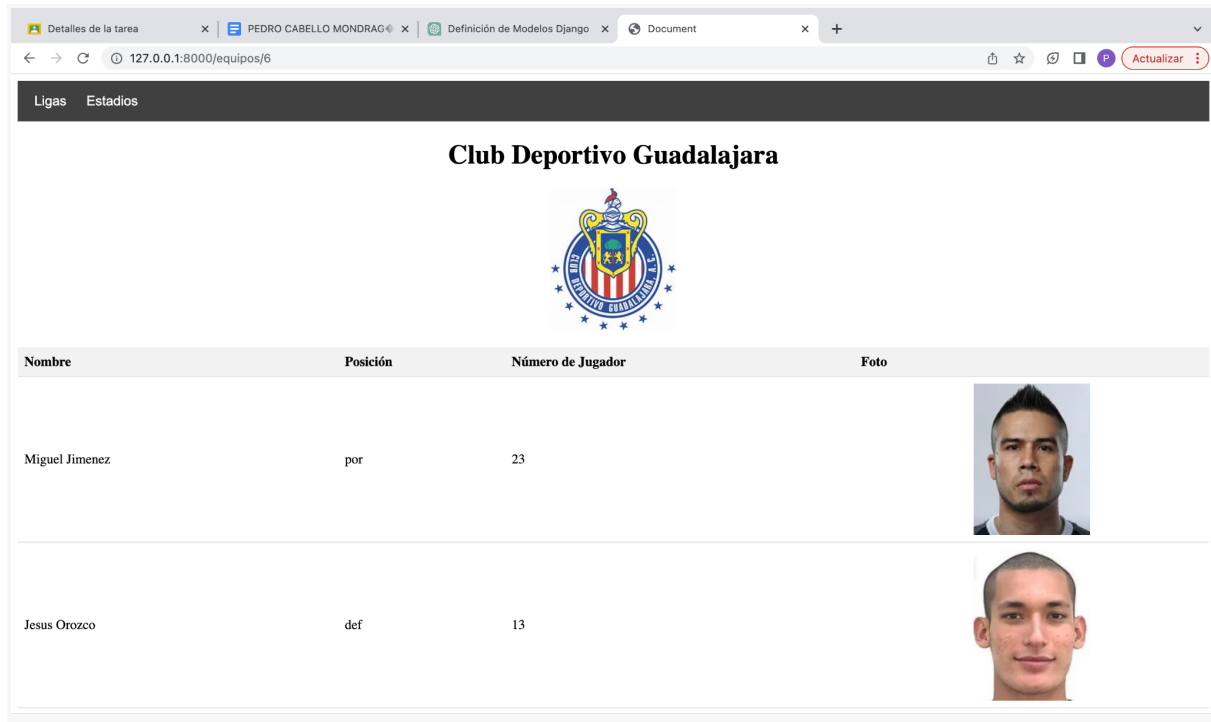
  </tbody>



</table>

{% endblock %}

```

En resumen este código sirve como para redirigir a una página donde muestra información del equipo al que pertenece cada jugador dándole clic al enlace que aparece en la columna equipo.



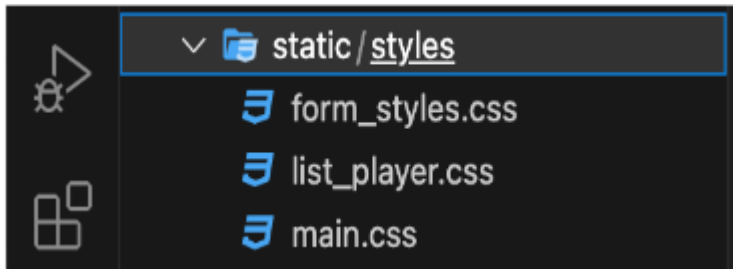
| Nombre | Posición | Número de Jugador | Foto |
|----------------|----------|-------------------|---|
| Miguel Jimenez | por | 23 |  |
| Jesus Orozco | def | 13 |  |

2.2.2 Static

Los archivos estáticos se utilizan para proporcionar estilos visuales, interactividad y recursos multimedia a las páginas web. Los archivos estáticos son servidos directamente por el servidor web sin procesamiento adicional, lo que mejora el rendimiento de la aplicación.

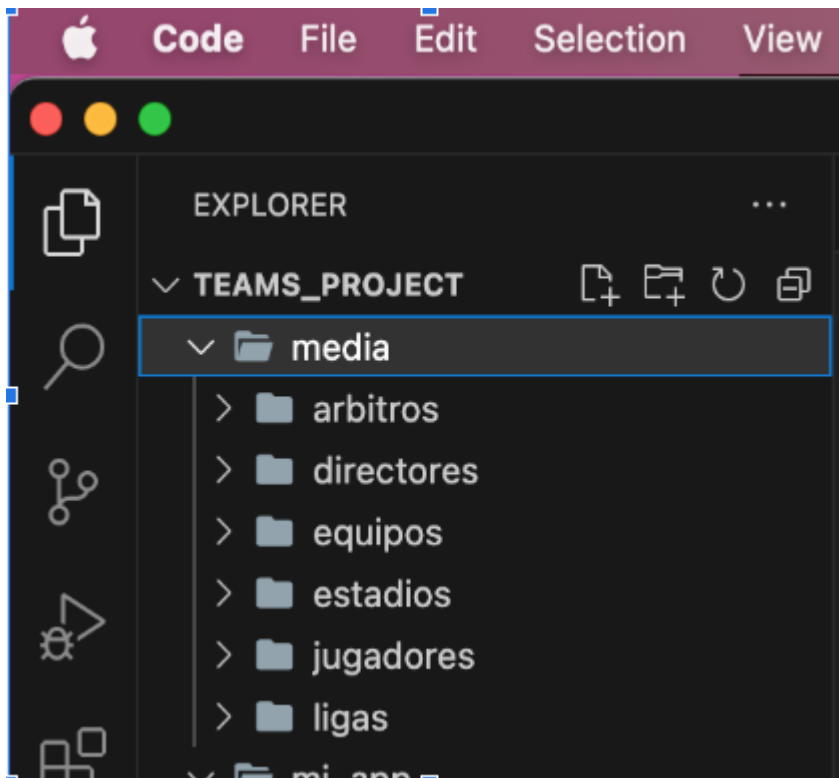
Cuando se trabaja con Django, se utiliza la etiqueta `{% load static %}` en los templates para cargar la funcionalidad de archivos estáticos. Luego, se puede utilizar la función `{% static %}` para generar la URL completa de un archivo estático, especificando la ruta relativa al directorio de archivos estáticos de la aplicación.

En esta carpeta se encuentran los archivos css que le dan los estilos a la pagina web, en este caso para los formularios, para la lista de los jugadores y la barra de navegación



2.2.3 Media

La media es la localidad de los recursos que necesita la pagina o archivos que guarda el usuario, en nuestro caso se guardan imágenes y se almacenan en carpetas dentro de media para hacer que la aplicación cargue más rápido



Aquí se encuentran las imágenes que se muestran en el navegador, tenemos las imágenes separados por carpetas para mayor organización.

3 Conclusión

En general, el proyecto se enfoca en proporcionar una interfaz de usuario para administrar la información relacionada con los equipos de fútbol, las ligas y otros elementos involucrados en el contexto del fútbol. Los formularios, las vistas y los templates trabajan en conjunto para permitir la creación, edición y visualización de datos, brindando una experiencia interactiva y amigable para los usuarios.

Django demuestra ser un framework de altísimo nivel debido a su simplicidad de lenguaje y la rapidez en la que se puede crear una aplicación web, además de diversas cosas, django ya te ofrece herramientas para su desarrollo de aplicaciones, como es el caso de la base de datos lite que cuenta en su sistema.

Para darle un plus a django puedes utilizar otros framework de desarrollo para crear la interfaz gráfica y con django darle la funcionalidad, en mi opinión es muy buen framework para empezar en el mundo de la programación como en el mundo del desarrollo web.