



Instituto Politécnico de Viseu
Escola Superior de Tecnologia e Gestão de Viseu
Departamento de Informática

Unidade Curricular: Base de Dados 2

Relatório Relativo ao Projeto

Tema: FutebolStats

Realizado por: Marco Rodrigues – 25001

Miguel Silva – 25187

Tiago Figueiredo – 17185

Joel Aparício - 19908

Viseu, 2025

Instituto Politécnico de Viseu
Escola Superior de Tecnologia e Gestão de Viseu
Departamento de Informática

Relatório relativo ao Projeto

Curso de Licenciatura em Engenharia Informática

Unidade Curricular de Base de Dados 2

FutebolStats

Ano Letivo 2024/25

Viseu, 2025

RESUMO

Este programa tem a função de simular um Website de estatísticas de Futebol, onde qualquer utilizador da Internet poderá visitar e ver informações sobre Jogos, Jogadores, Competições, etc.

Palavras-Chave: Base de Dados, MongoDB, Postgres, Django, Manipulação, Dados.

ÍNDICE

1. Introdução	1
2. Base de Dados	2
2.1. Criação da Base de Dados	2
2.2. Configuração no Django	3
2.2.1. Routers	4
3. Framework Django	6
3.1. Models	6
3.1.1. ObjectId	6
3.1.2. Relações	7
3.1.3. Nome Tabela	7
3.1.4. Devolver Ele Próprio	7
3.2. Forms	7
3.2.1. Campos Dropdown, Date e Time	8
3.2.2. Relações	9
3.2.3. Forms Utils	10
3.2.4. Valores Iniciais	10
3.2.5. Verificação	10
3.3. Views	11
3.3.1. Mostrar Valores	11
3.3.2. Adicionar e Editar Valor	12
3.3.3. Apagar Valor	13
3.4. Templates	14
3.4.1. Adicionar e Editar	14
3.4.2. Apagar	14
3.4.3. Erro ao Apagar	15
3.5. Urls	15
4. Funcionamento	17
4.1. Script	17
4.2. Programa	18
4.2.1. Administradores VS Utilizadores	18
4.2.2. Estádios e Associações	20
4.2.3. Clubes e Equipas	21
4.2.4. Jogadores	23
4.2.5. Jogos e Estatísticas	24
5. Conclusões	26
6. Referências	27
7. Bibliografia	28

Índice de tabelas

Tabela 3-1 on_delete das Relações - Django.....	7
---	---

Índice de Figuras

Figura 2-1 Modelo Conceptual.....	2
Figura 2-2 Modelo Físico	3
Figura 2-3 Configuração SGBD - Django.....	4
Figura 2-4 Configuração Router – Django	5
Figura 2-5 Router no Model - Django	5
Figura 3-1 Exemplo Model – Django.....	6
Figura 3-2 Biblioteca ObjectId – Django	6
Figura 3-3 Formulário – Django.....	8
Figura 3-4 Dropdown no Formulário – Django.....	8
Figura 3-5 Dia e Hora no Formulário - Django.....	9
Figura 3-6 Relações no Formulário – Django	9
Figura 3-7 Relações no Formulário 2 - Django.....	9
Figura 3-8 Funções de Conversão – Django	10
Figura 3-9 Valor Inicial Formulário - Django	10
Figura 3-10 Verificação Final do Formulário - Django	11
Figura 3-11 View para Listar – Django.....	11
Figura 3-12 View para Mostrar Detalhes - Django	12
Figura 3-13 View para Adicionar – Django	12
Figura 3-14 View para Editar - Django	13
Figura 3-15 View para Apagar – Django	13
Figura 3-16 View para Apagar com Atualizações - Django	13
Figura 3-17 Adicionar e Editar – Template Django.....	14
Figura 3-18 Modal Apagar - Template Django	15
Figura 3-19 Mostrar Erro ao Eliminar - Template Django.....	15
Figura 3-20 Urls - Django	16
Figura 4-1 Script de Dados – Django	17
Figura 4-2 Inserção de Dados Com Relações - Django.....	18
Figura 4-3 Perfil – Administrador	18
Figura 4-4 Homepage – Administrado	19
Figura 4-5 Perfil – Utilizador	19
Figura 4-6 Homepage - Utilizador.....	20
Figura 4-7 Listagem de Estádios – Administrador.....	20
Figura 4-8 Editar Estádio – Administrador	21
Figura 4-9 Detalhes do Estádio	21
Figura 4-10 Associações no Clube – Administrador.....	22
Figura 4-11 Ver Equipas – Administrador	22
Figura 4-12 Adicionar Equipa – Administrador.....	22
Figura 4-13 Detalhes Clube.....	23
Figura 4-14 Editar Jogador – Administrador.....	23
Figura 4-15 Detalhes Jogador.....	24
Figura 4-16 Estatísticas de um Jogo – Administrador.....	24
Figura 4-17 Detalhes Jogo.....	25
Figura 4-18 Detalhes Estatísticas de um Jogo	25

1. Introdução

Foi-nos pedido pelos professores de Base de Dados 2 a realizar um projeto prático com a utilização da framework Django e, para a base de dados, o MongoDB e Postgres com o objetivo de avaliarem as nossas capacidades, utilizando o que nos foi ensinado ao longo do semestre.

Neste programa, uma das dificuldades com que nos deparámos foi a configuração para a utilização de duas bases de dados diferentes, sendo estes o MongoDB e o Postgres.

Este documento está organizado em quatro capítulos que se seguem a esta introdução.

No segundo capítulo propõe-se a Base de Dados.

Segue-se o capítulo três, onde falamos da Framework Django.

O capítulo quatro aborda o Funcionamento do programa.

Termina-se com o capítulo cinco, onde se apresentam as respetivas conclusões obtidas sobre este trabalho.

2. Base de Dados

Neste capítulo será falado da criação da Base de Dados e a sua configuração no Django.

2.1. Criação da Base de Dados

Inicialmente, como qualquer projeto que envolve dados, é necessário criar a respectiva base de dados e, para isso, foi utilizado o PowerDesigner para a organizar. Esta base de dados consiste em guardar informações relativas a futebol, sendo elas os Jogadores, Clubes, Jogos, etc.

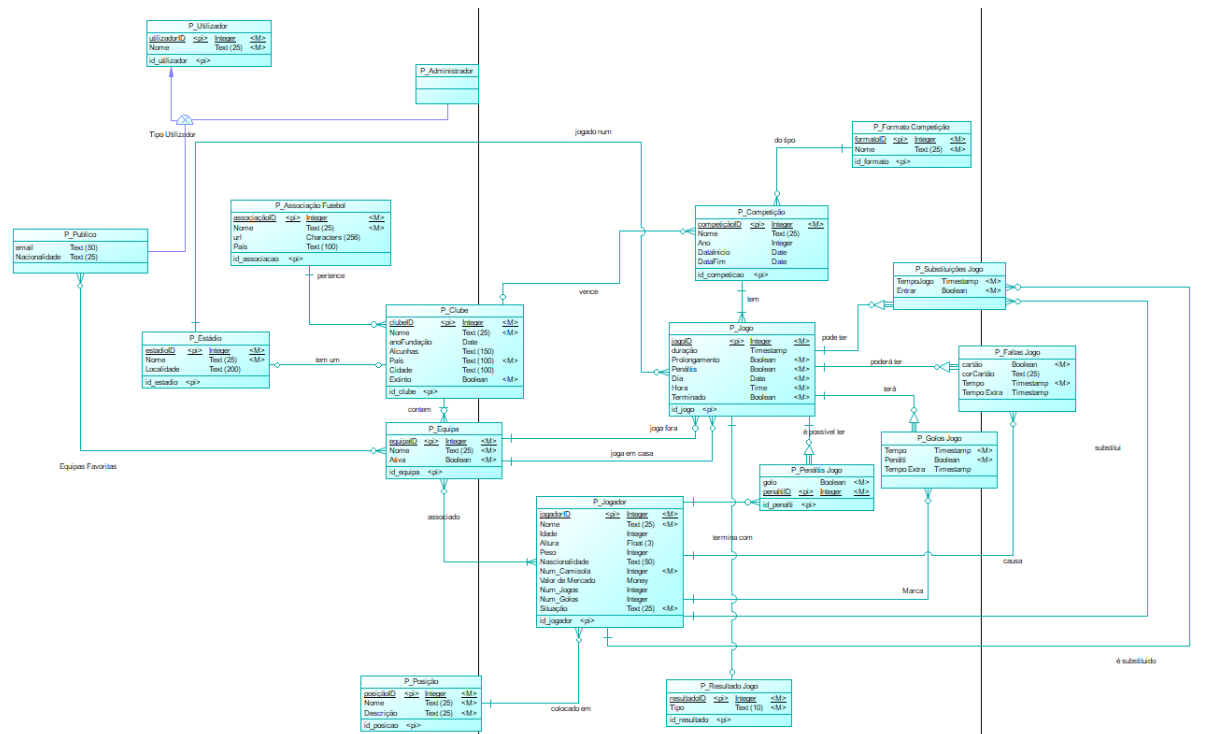


Figura 2-1 Modelo Conceptual


```
DATABASES = {
    # postgreSQL => Utilizadores
    'default': {
        'ENGINE': 'django.db.backends.postgresql',
        'NAME': 'projeto_BD2',
        'USER': 'postgres',
        'PASSWORD': '12345',
        'HOST': 'localhost',
        'PORT': '5432',
    },
    # mongoDB => Resto da Base de Dados
    'mongodb': {
        'ENGINE': 'djongo',
        'NAME': 'projeto_BD2',
        'ENFORCE_SCHEMA': False,
        'CLIENT': {
            'host': 'localhost',
            'port': 27017,
        },
        'LOGGING': {
            'version': 1,
            'loggers': {
                'djongo': {
                    'level': 'DEBUG',
                    'propagate': False,
                }
            }
        },
    },
},
}
```

Figura 2-3 Configuração SGBD - Django

2.2.1. Routers

Para a configurar o Django de forma que os dados sejam guardados nas respetivas SGBD foi necessário criar um ficheiro chamado “routers.py” onde seria feito a respetiva configuração. Como é possível ver na Figura 2-4, foi configurado para cada uma das ações a respetiva Base de Dados dependendo da “app_label”.

```

class MultiDBRouter:
    def db_for_read(self, model, **hints):
        """Define qual base de dados usar para leitura."""
        if model._meta.app_label == 'BD2_Trabalhofinal.App': #Nome app
            return 'mongodb'
        return 'default'

    def db_for_write(self, model, **hints):
        """Define qual base de dados usar para escrita."""
        if model._meta.app_label == 'BD2_Trabalhofinal.App': #Nome app
            return 'mongodb'
        return 'default'

    def allow_relation(self, obj1, obj2, **hints):
        """Permite relações entre objetos de base de dados diferentes."""
        db_list = ('default', 'mongodb')
        if obj1._state.db in db_list and obj2._state.db in db_list:
            return True
        return None

    def allow_migrate(self, db, app_label, model_name=None, **hints):
        """Define em qual base de dados as migrações podem ser executadas."""
        if app_label == 'BD2_Trabalhofinal.App': #Nome app
            return db == 'mongodb'
        return db == 'default'

```

Figura 2-4 Configuração Router – Django

Além do router, foi necessário ir aos respectivos Modelos/Objetos (Models, que será falado no próximo capítulo) e adicionar a “app_label” nos que serão guardados no MongoDB.

```

class Meta:
    db_table = "p_associacoes"
    app_label = 'BD2_Trabalhofinal.App'

```

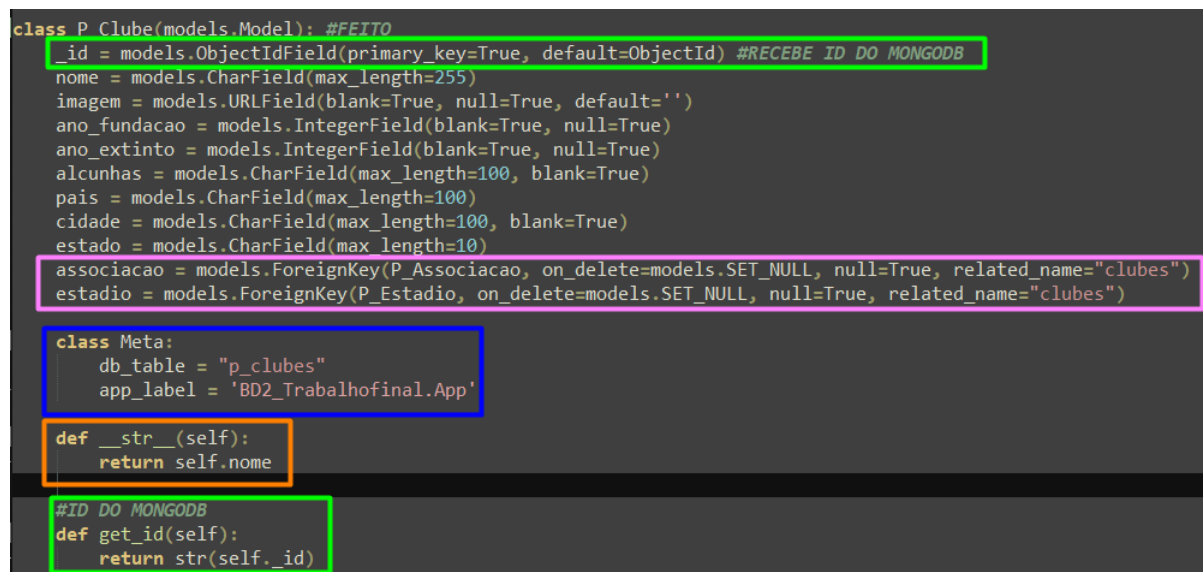
Figura 2-5 Router no Model - Django

3. Framework Django

Como já foi mencionado, um dos requisitos principais era a utilização da Framework Django no desenvolvimento deste projeto.

3.1. Models

Uma das partes mais importantes do trabalho são os Models. Estes são as “tabelas” de base de dados da Framework do Django, que permitem a configuração e o controlo destas visto que é a partir destes modelos que as tabelas são criadas nos SGBD (através das migrações).



```
class P_Clube(models.Model): #FEITO
    _id = models.ObjectIdField(primary_key=True, default=ObjectId) #RECEBE ID DO MONGODB
    nome = models.CharField(max_length=255)
    imagem = models.URLField(blank=True, null=True, default='')
    ano_fundacao = models.IntegerField(blank=True, null=True)
    ano_extinto = models.IntegerField(blank=True, null=True)
    alcunhas = models.CharField(max_length=100, blank=True)
    pais = models.CharField(max_length=100)
    cidade = models.CharField(max_length=100, blank=True)
    estado = models.CharField(max_length=10)
    associacao = models.ForeignKey(P_Associacao, on_delete=models.SET_NULL, null=True, related_name="clubes")
    estadio = models.ForeignKey(P_Estadio, on_delete=models.SET_NULL, null=True, related_name="clubes")

    class Meta:
        db_table = "p_clubes"
        app_label = 'BD2_Trabalhofinal.App'

    def __str__(self):
        return self.nome

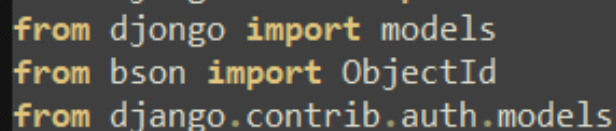
    #ID DO MONGODB
    def get_id(self):
        return str(self._id)
```

Figura 3-1 Exemplo Model – Django

Estes Models contêm várias definições marcadas pelas diferentes cores na imagem acima (Fig. 3-1), sendo a parte não assinalada as variáveis/colunas da tabela de base de dados “normais”/simples.

3.1.1. ObjectId

Este Model em específico representa um conjunto de dados que será guardado no SGBD MongoDB e, por isso, o ID não pode ser um número inteiro, mas sim um ObjectId. Para a sua utilização é necessário importar a respetiva biblioteca.



```
from django import models
from bson import ObjectId
from django.contrib.auth.models
```

Figura 3-2 Biblioteca ObjectId – Django

Além disso, foi decidido criar a função “get_id” (assinalado a verde) para devolver o respetivo ObjectId, permitindo assim a utilização do identificador de forma prática e eficiente.

3.1.2. Relações

As relações, também conhecidas como chaves estrangeiras, permitem a ligação entre tabelas de dados. No modelo apresentado acima é possível ver duas relações assinaladas a rosa, que relacionam um elemento da tabela “P_Associacao” e “P_Estadio”, respetivamente.

As associações contêm uma função muito importante: “on_delete”. Esta garante que, caso o objeto a que está associado seja apagado, uma das seguintes situações acontecerá:

Ação	Consequência
SET_NULL	Altera o valor para Null (ou seja, nada)
CASCADE	Apaga os objetos relacionados (ou seja, apagaria o Clube)
PROTECT	Impede que seja apagado se existir objetos relacionados (ou seja, não será possível apagar a Associação se um Clube estiver associado)
SET_DEFAULT	Altera para o valor padrão
DO_NOTHING	Não faz nada (pode causar erros)

Tabela 3-1 on_delete das Relações - Django

3.1.3. Nome Tabela

A classe Meta, assinalada a azul, contém o “db_table” que indica o nome da tabela na respetiva base de dados, existindo em todos eles. Já o “app_label” existe naqueles que serão guardados no MongoDB, como foi falado anteriormente.

3.1.4. Devolver Ele Próprio

Como é possível ver na Figura 3-1, existe uma outra função assinalada a laranja. Esta serve para devolver, neste caso, o nome do objeto para o caso de este ser chamado, por exemplo, num formulário.

3.2. Forms

Para criar os formulários para a adição e edição de dados do programa, utiliza-se o ficheiro “forms.py” e cria-se um semelhante ao que se encontra na Figura a seguir.

```

class P_PosicaoForm(forms.ModelForm):
    class Meta:
        model = P_Posicao
        fields = ['nome', 'descricao', 'desig']
        widgets = {
            'nome': forms.TextInput(attrs={'class': 'form-control', 'placeholder': 'Insira o nome da posição'}),
            'descricao': forms.Textarea(attrs={'class': 'form-control', 'placeholder': 'Insira a descrição'}),
            'desig': forms.TextInput(attrs={'class': 'form-control', 'placeholder': 'Insira a designação da posição'})
        }
        labels = {
            'nome': 'Nome da Posição',
            'descricao': 'Descrição',
            'desig': 'Designação',
        }

```

Figura 3-3 Formulário – Django

Em primeiro lugar, seleciona-se o Model (azul) e os seguintes itens/colunas (laranja) a inserir/editar. Na parte dos *widgets* (verde) seleciona-se um dos seguintes tipos de campo, de acordo com o respetivo tipo de dado:

- ➔ TextInput
- ➔ Textarea
- ➔ CheckboxInput
- ➔ NumberInput

No caso dos tipos dropdown, como é o caso das relações que serão faladas mais à frente, data e hora não se coloca nesta parte. Por fim, o *labels* servem para ajudar o utilizador a identificar que dado inserir.

3.2.1. Campos Dropdown, Date e Time

Como foi falado, também é possível criar campos de Dropdown (Fig. 3-4), ou seja, uma lista de opções desde que o tipo de dado esteja configurado como texto, senão esta função irá certamente causar erros indesejados.

O Django também disponibiliza campos do tipo data e hora (Fig. 3-5), oferecendo ao utilizador um seletor destes, melhorando assim a experiência de utilização por ele.

```

estado = forms.ChoiceField(
    choices=[
        ('', 'Selecione o Estado'),
        ('Em Breve', 'Em Breve'),
        ('A Decorrer', 'A Decorrer'),
        ('Terminado', 'Terminado'),
        ('Cancelado', 'Cancelado'),
    ],
    widget=forms.Select(attrs={'class': 'form-control'}),
    required=True
)

```

Figura 3-4 Dropdown no Formulário – Django

```

# DIA e HORA
dia = forms.DateField(
    widget=forms.DateInput(attrs={'type': 'date', 'class': 'form-control'}),
    label="Dia do Jogo"
)
hora = forms.TimeField(
    widget=forms.TimeInput(
        attrs={
            'type': 'time',
            'class': 'form-control'
        }
    ),
    label="Hora do Jogo"
)

```

Figura 3-5 Dia e Hora no Formulário - Django

3.2.2. Relações

Para as respectivas relações no formulário, basta criar um campo do tipo “dropdown”, fora da área dos *widgets*, sendo necessário apenas escolher o Model a referenciar e, se necessário, adicionar o filtro como é possível ver assinalado a azul (Fig. 3-6).

```

# COMPETICAO E ESTADIO
competicao = forms.ModelChoiceField(
    queryset=P_Competicao.objects.all(),
    widget=forms.Select(attrs={'class': 'form-control'}),
    empty_label="Escolha a Competicao"
)
estadio = forms.ModelChoiceField(
    queryset=P_Estadio.objects.filter(estado="Ativo"),
    widget=forms.Select(attrs={'class': 'form-control'}),
    empty_label="Escolha o Estádio"
)

```

Figura 3-6 Relações no Formulário – Django

Além do campo, é necessário escolher como que o objeto irá aparecer na respetiva lista ao utilizador (azul) e garantir a conversão do seu identificador (verde) através das funções criadas no ficheiro “form_utils.py”, dentro da função “__init__”.

```

# COMPETICAO
self.fields['competicao'].label_from_instance = lambda obj: f"{obj.nome}"
self.fields['competicao'].to_python = convert_to_competicao

# ESTADIO
self.fields['estadio'].label_from_instance = lambda obj: f"{obj.nome}"
self.fields['estadio'].to_python = convert_to_estadio

```

Figura 3-7 Relações no Formulário 2 - Django

3.2.3. Forms Utils

Um dos problemas encontrado foi na seleção do objeto para a relação de tabelas, ou seja, a chave estrangeira. Para isso foram criadas funções de conversão dos identificadores (IDs) para os relacionamentos de cada objeto.

```
def convert_to_clube(value):
    if not value:
        return None
    try:
        if isinstance(value, str):
            return P_Clube.objects.get(_id=ObjectId(value))
        return value
    except Exception as e:
        raise ValidationError('Clube inválido')
```

Figura 3-8 Funções de Conversão – Django

3.2.4. Valores Iniciais

A função “__init__”, além de permitir a configuração das relações, permite adicionar valores iniciais a certos valores do Model.

```
def __init__(self, *args, **kwargs):
    super().__init__(*args, **kwargs)
    self.fields['duracao'].initial = 90
```

Figura 3-9 Valor Inicial Formulário - Django

3.2.5. Verificação

Em certas ocasiões, foi necessário criar a função “clean” para verificação de dados e impedir erros. Na imagem abaixo (Fig. 3-10) é possível ver o exemplo utilizado no formulário do Clube, onde verifica se foi inserido o ano de extinção do clube se este estiver extinto. A outra parte, assinalada a rosa, faz a conversão da imagem deste para que, caso o utilizador não insira uma, o valor seja “None” (null).


```
def clean(self):
    cleaned_data = super().clean()
    if 'imagem' in cleaned_data and cleaned_data['imagem'] == '':
        cleaned_data['imagem'] = None

    estado = cleaned_data.get('estado')
    ano_extinto = cleaned_data.get('ano_extinto')

    if estado == 'Extinto' and (not ano_extinto or ano_extinto == 0):
        raise forms.ValidationError('Para clubes extintos, é necessário informar o ano de extinção.')

    if estado != 'Extinto' and ano_extinto and ano_extinto != 0:
        cleaned_data['ano_extinto'] = 0

    return cleaned_data
```

Figura 3-10 Verificação Final do Formulário - Django

3.3. Views

As *views* são indispensáveis para o funcionamento do programa pois são estas que indicam o que será feito quando o utilizador abre um *url* (caminho) / página do website.

3.3.1. Mostrar Valores

Neste programa existem duas *views* para mostrar os dados, sendo estas o *listar_X* e o *detalhes_X*, sendo X o respetivo objeto. A primeira tem o objetivo de listar todos os dados do respetivo modelo enquanto a segunda serve para mostrar apenas um.

```
## --- Jogadores ---
def listar_jogadores(request):
    jogadores = P_Jogador.objects.all().order_by('nome', 'num_camisola') # Ordenar por Nome e Número de Camisola para melhor organização
    return render(request, 'jogadores/listar_jogadores.html', {'jogadores': jogadores})
```

Figura 3-11 View para Listar – Django

Em algumas *views* do tipo “detalhes”, foi necessário obter outros valores disponíveis através das relações e, para isso, depois de obter o respetivo jogador (no caso da Figura 3-12), é obtido todos os golos e jogos e estes serão guardados num “context” (contexto). Este contexto será retornado ao renderizar o respetivo template, permitindo assim mostrar os respetivos dados neste.

```
def detalhes_jogador(request, id):
    jogador = get_object_or_404(P_Jogador, _id=ObjectId(id))

    # Contar o número de gols do jogador
    num_golos = P_Golo.objects.filter(jogador=jogador).count()

    # Contar o número de jogos do jogador (verifica se a equipa do jogador é equipa_casa ou equipa_fora)
    num_jogos = P_Jogo.objects.filter(
        equipa_casa=jogador.equipa
    ).count() + P_Jogo.objects.filter(
        equipa_fora=jogador.equipa
    ).count()

    # Buscar todos os jogos onde o jogador participou
    jogos = P_Jogo.objects.filter(
        equipa_casa=jogador.equipa
    ) | P_Jogo.objects.filter(
        equipa_fora=jogador.equipa
    )

    context = {
        'jogador': jogador,
        'num_golos': num_golos,
        'num_jogos': num_jogos,
        'jogos': jogos.filter(estados="Terminado") #Só mostra os jogos terminados
    }

    return render(request, 'jogadores/detalhes_jogador.html', context)
```

Figura 3-12 View para Mostrar Detalhes - Django

3.3.2. Adicionar e Editar Valor

A *view* de adicionar, simplesmente verifica o método de requisição e, caso não seja “POST”, este chama o respetivo template e formulário, mas, caso seja, este guarda os valores introduzidos pelo utilizador e renderiza o template da respetiva lista.

```
def adicionar_competicao(request):
    if request.method == 'POST':
        form = P_CompeticaoForm(request.POST)
        if form.is_valid():
            competicao = form.save(commit=False)
            # Tratar de strings vazias para campos opcionais
            if not competicao.imagem:
                competicao.imagem = None
            competicao.save()
            return redirect('listar_competicoes')
        else:
            form = P_CompeticaoForm()
            return render(request, 'competicoes/adicionar_competicao.html', {'form': form})
```

Figura 3-13 View para Adicionar – Django

A *view* de editar é bastante semelhante à de adicionar, com a diferença que esta vai buscar os dados já introduzidos através do identificador (ID) e mostra-os ao utilizador.

```
def editar_competicao(request, id):
    competicao = get_object_or_404(P_Competicao, _id=ObjectId(id))
    if request.method == 'POST':
        form = P_CompeticaoForm(request.POST, instance=competicao)
        if form.is_valid():
            form.save()
            return redirect('listar_competicoes')
    else:
        form = P_CompeticaoForm(instance=competicao)
    return render(request, 'competicoes/editar_competicao.html', {'form': form})
```

Figura 3-14 View para Editar - Django

3.3.3. Apagar Valor

Por fim, a view de eliminação irá procurar o objeto através do identificador e apagá-lo.

```
def apagar_competicao(request, id):
    competicao = get_object_or_404(P_Competicao, _id=ObjectId(id))
    if request.method == 'POST':
        competicao.delete()
        return redirect('listar_competicoes')
```

Figura 3-15 View para Apagar – Django

Em certos casos, a view será mais complexa (Fig. 3-16), com função de atualizações ou até impedindo a sua eliminação caso existam relações.

A zona assinalada azul verifica se existem equipas associadas ao clube e, se existirem, avisa o utilizador sem processar a eliminação. Já a parte assinalada a verde, verifica se existem jogadores, competições e jogos com o respetivo clube associado, altera o valor deste para “None” (null) e processa a eliminação deste.

```
def apagar_clube(request, id):
    clube = get_object_or_404(P_Clube, _id=ObjectId(id))

    # Verificar se há equipas associadas
    if P_Equipa.objects.filter(clube=clube).exists():
        messages.error(request, "Não é possível apagar este Clube porque ele tem equipas associadas.")
        return redirect('listar_clubes')

    if request.method == 'POST':
        # Atualiza os jogadores para terem clube = None
        P_Jogador.objects.filter(clube=clube).update(clube=None)
        # Atualiza o vencedor da competição
        P_Competicao.objects.filter(vencedor=clube).update(vencedor=None)
        # Atualiza o jogo
        P_Jogo.objects.filter(clube_casa=clube).update(clube_casa=None)
        P_Jogo.objects.filter(clube_fora=clube).update(clube_fora=None)
        P_Jogo.objects.filter(vencedor=clube).update(vencedor=None)

        clube.delete()
        return redirect('listar_clubes')
```

Figura 3-16 View para Apagar com Atualizações - Django

3.4. Templates

Os templates são ficheiros HTML, com o objetivo de criar uma aparência simples, mas intuitiva através do Bootstrap CSS e JS.

3.4.1. Adicionar e Editar

Os templates para adicionar e editar são semelhantes, podendo alterar apenas a sua aparência, visto que o importante destes é o *form* com o método “POST” e o botão para submeter a operação.

```
<form method="post">
  {% csrf_token %}
  {% for field in form %}
    <div class="mb-3">
      {% if field.name == 'ano_extinto' %}
        <div id="ano-extinto-field" style="display: none;">
          {{ field.label_tag }}
          {{ field }}
          {% if field.help_text %}
            <small class="form-text text-muted">{{ field.help_text }}</small>
          {% endif %}
        </div>
      {% else %}
        {{ field.label_tag }}
        {{ field }}
        {% if field.help_text %}
          <small class="form-text text-muted">{{ field.help_text }}</small>
        {% endif %}
      {% endif %}
    </div>
  {% endfor %}
  <button type="submit" class="btn btn-primary">Salvar</button>
  <a href="{% url 'listar_clubes' %}" class="btn btn-secondary">Cancelar</a>
</form>
```

Figura 3-17 Adicionar e Editar – Template Django

3.4.2. Apagar

Em vez de criar um template para a eliminação, foi decidido que fosse utilizado um modal visto que este não precisa de mostrar nada ao utilizador. Como acontece no adicionar e editar, é necessário garantir que o *form* tenha o método “POST” para funcionar e, além disso, recebam o identificador (ID) do objeto a eliminar.

```

<!-- Botão para abrir o modal -->
<button type="button" class="btn btn-danger btn-sm" data-bs-toggle="modal" data-bs-target="#deleteModal{{ clube.get_id }}">
  Apagar
</button>

<!-- Modal -->
<div class="modal fade" id="deleteModal{{ clube.get_id }}" tabindex="-1" aria-labelledby="deleteModalLabel{{ clube.get_id }}" aria-hidden="true">
  <div class="modal-dialog">
    <div class="modal-content">
      <div class="modal-header bg-danger text-white">
        <h5 class="modal-title" id="deleteModalLabel{{ clube.get_id }}">Confirmar Exclusão</h5>
        <button type="button" class="btn-close" data-bs-dismiss="modal" aria-label="Close"></button>
      </div>
      <div class="modal-body">
        Tem certeza de que deseja apagar o clube <strong>{{ clube.nome }}</strong> ({{ clube.ano_fundacao }})?
      </div>
      <div class="modal-footer">
        <form method="post" action="{% url 'apagar_clube' clube.get_id %}">
          {% csrf_token %}
          <button type="submit" class="btn btn-danger">Apagar</button>
          <button type="button" class="btn btn-secondary" data-bs-dismiss="modal">Cancelar</button>
        </form>
      </div>
    </div>
  </div>
</div>

```

Figura 3-18 Modal Apagar - Template Django

3.4.3. Erro ao Apagar

Como foi falado no subcapítulo 3.3.3, caso uma *view* de eliminação encontrar uma relação que não pode ser alterada para “None”, esta mostrará o erro ao utilizador.

```

{% if clubes %}
{% if messages %}
  {% for message in messages %}
    <div class="alert" {% if message.tags == 'error' %}alert-danger{% else %}alert-success{% endif %} alert-dismissible fade show" role="alert">
      {{ message }}
      <button type="button" class="btn-close" data-bs-dismiss="alert" aria-label="Close"></button>
    </div>
  {% endfor %}
{% endif %}
<div class="table-responsive">
  <table class="table table-bordered table-hover" id="clubesTabela">
    <thead class="table-dark">
      <tr>

```

Figura 3-19 Mostrar Erro ao Eliminar - Template Django

3.5. Urls

Para as *views* funcionarem corretamente é necessário criar os respetivos *urls*. Estes servem para indicar o caminho (selecionado a vermelho) e, ao serem abertos, indicaram qual a *view* (selecionado a rosa) a chamar, permitindo dar-lhe um respetivamente (selecionado a azul). Um excerto do código é possível ver na imagem abaixo.

```
urlpatterns = [
    path('', views.home, name='home'),
    path('bd/conectividade', views.lista_utilizadores, name='lista_utilizadores'),
    path('login/', views.login_view, name='login'),
    path('logout/', views.logout_view, name='logout'),
    path('register/', views.register, name='register'),

    # PERFIL
    path('perfil/', views.ver_perfil, name='perfil'),
    path('perfil/editar/dados', views.editar_perfil, name='editar_perfil'),
    path("perfil/editar/senha/", views.editar_senha, name="editar_senha"),

    # POSIÇÕES DE CAMPO DOS JOGADORES
    path('admin/posicoes/', views.listar_posicoes, name='listar_posicoes'),
    path('admin/posicoes/adicionar/', views.adicionar_posicao, name='adicionar_posicao'),
    path('admin/posicoes/editar/<str:id>/', views.editar_posicao, name='editar_posicao'),
    path('admin/posicoes/apagar/<str:id>/', views.apagar_posicao, name='apagar_posicao'),
```

Figura 3-20 Urls - Django

4. Funcionamento

Neste capítulo iremos falar resumidamente do funcionamento da aplicação Web desenvolvida.

4.1. Script

Um dos requisitos deste trabalho era a existência de um *script* de dados e, para isso, foi criado o ficheiro “inserir_dados.py” onde foram colocados alguns exemplos de dados.

De forma geral, os dados são inseridos de forma sequencial, sendo guardados em vetores (ou *arrays*) para facilitar a inserção de dados de tabelas com chaves estrangeiras, como é possível ver na Figura 4-2.

```
# Usa-se o comando python manage.py inserir_dados
class Command(BaseCommand):
    help = 'Inserir dados iniciais para todos os Models'

    def handle(self, *args, **options):
        try:
            with transaction.atomic():

                # Inserir Associações
                associacoes_data = [
                    {
                        'nome': "Associação de Futebol de Viseu",
                        'url': "https://afviseu.fpf.pt/",
                        'pais': "Portugal",
                        'imagem': "https://www.zerozero.pt/img/logos/associacoes/17_af_viseu_imgbank.png"
                    },
                    {
                        'nome': "Associação de Futebol de Aveiro",
                        'url': "http://afaveiro.pt",
                        'pais': "Portugal",
                        'imagem': "https://afaveiro.fpf.pt/Portals/19/Logo_AFA3.gif?ver=2019-01-24-120420-670"
                    }
                ]
```

Figura 4-1 Script de Dados – Django

```

# Inserir Clubes
## ano_extinto não pode ser null senão dá erro
## Estado do Clube: Ativo / Suspenso / Extinto
clubes_data = [
    {
        'nome': "CD Santacruzense",
        'imagem': "https://www.zerozero.pt/img/logos/equipas/15069_imgbank 1734620394.png",
        'ano_fundacao': 1931,
        'ano_extinto': 0,
        'alculhas': "Santa",
        'pais': "Portugal",
        'cidade': "Santa Cruz da Trapa - São Pedro do Sul",
        'estado': "Ativo",
        'associacao': associacoes[0], # Associação de Futebol de Viseu
        'estadio': estadios[25]
    },
    {
        'nome': "Naval",
        'imagem': "https://www.zerozero.pt/img/logos/equipas/28_imgbank.png",
        'ano_fundacao': 1893,
        'ano_extinto': 2017,
        'alculhas': "Velhinha Senhora",
        'pais': "Portugal",
        'cidade': "Figueira da Foz",
        'estado': "Extinto",
        'associacao': associacoes[2], # Associação de Futebol de Coimbra
        'estadio': estadios[3] # Estádio Municipal José Bento Pessoa
    },
]

```

Figura 4-2 Inserção de Dados Com Relações - Django

4.2. Programa

Aqui será mostrado, de forma muito resumida, o funcionamento do programa criado.

4.2.1. Administradores VS Utilizadores

Neste sistema, existem os Administradores que criam, editam e apagam os dados do website e existem, também, os utilizadores que podem ver todas as informações disponíveis e, além disso, selecionar os seus clubes favoritos.

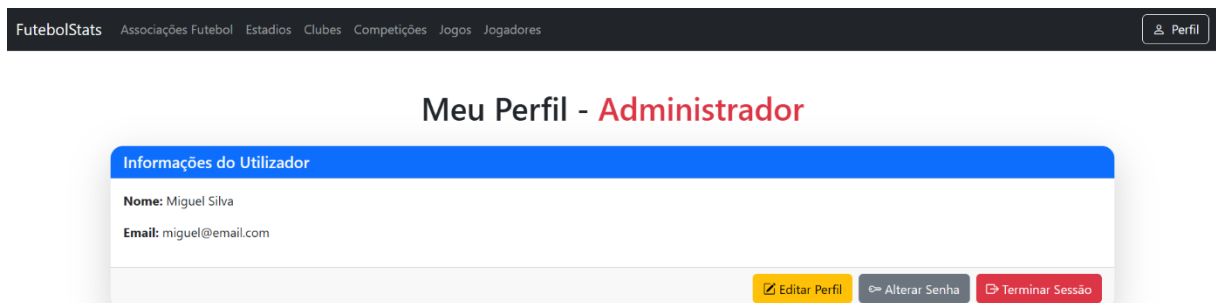


Figura 4-3 Perfil – Administrador



Figura 4-4 Homepage – Administrado

Como foi mencionado, o utilizador que esteja autenticado no website pode adicionar clubes favoritos (Fig. 4-5) e, assim, ter os jogos destes em destaque na página inicial (Fig. 4-6).

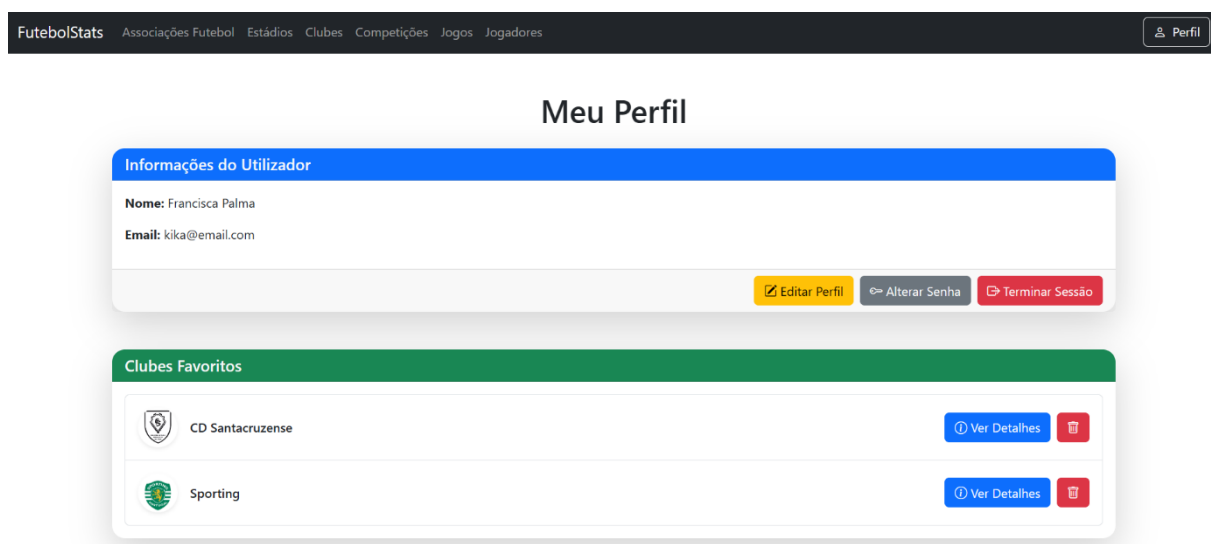




Figura 4-5 Perfil – Utilizador

Próximos 3 Jogos

Benfica


Em Breve
20/02/2025 14:00
[Ver Detalhes](#)

Sporting 

Jogos A Decorrer

Nenhum jogo a decorrer.

Últimos 3 Jogos

Porto

Sporting
25/12/2024
[Ver Detalhes](#)


Sporting 

Figura 4-6 Homepage - Utilizador

4.2.2. Estádios e Associações

O administrador terá uma listagem semelhante a estas em todas as páginas, permitindo assim o controlo total sobre os dados. A inserção e edição de dados é simples e intuitiva, como é possível ver na figura 4-8.

FutebolStats Associações Futebol Estádios Clubes Competições Jogos Jogadores [Perfil](#)

Lista de Estádios

[Adicionar Estádio](#)

Nome	Localidade	Dados	Estado	Ações
Campo de jogos Almeida Sobrinho	Portugal, S. Cruz da Trapa	Inauguração: None Lotação: 600	Em Obras	Ver Editar Apagar
Estádio Algarve	Portugal, Algarve	Inauguração: 2003 Lotação: 30305	Ativo	Ver Editar Apagar
Estádio António Coimbra da Mota	Portugal, Estoril	Inauguração: 1939 Lotação: 8000	Ativo	Ver Editar Apagar
Estádio Cidade de Barcelos	Portugal, Barcelos	Inauguração: 2004 Lotação: 12046	Ativo	Ver Editar Apagar

Figura 4-7 Listagem de Estádios – Administrador


Editar Estádio

Nome do Estádio:	<input type="text" value="Campo de jogos Almeida Sobrinho"/>
Imagem do Estádio:	<input type="text" value="https://www.zerozero.pt/img/estadios/602/660602_ori_20200308174404_campo_almeida_sobrinho.jpg"/>
País:	<input type="text" value="Portugal"/>
Cidade:	<input type="text" value="S. Cruz da Trapa"/>
Ano de Inauguração:	<input type="text" value="Ano de Inauguração"/>
Lotação do Estádio:	<input type="text" value="600"/>
Estado:	<input type="text" value="Em Obras"/>
<input type="button" value="Atualizar"/> <input type="button" value="Cancelar"/>	

Figura 4-8 Editar Estádio – Administrador

Ambos os tipos de utilizador podem visualizar a pagina de detalhes do respetivo estádio (e nos outros itens), sendo possível ver toda a sua informação e os Clubes associados, que serão falados no próximo subcapítulo.

Detalhes do Estádio




Estádio Municipal José Bento Pessoa

País: Portugal
Cidade: Figueira da Foz


Ficha do Estádio

Nome: Estádio Municipal José Bento Pessoa
País: Portugal
Cidade: Figueira da Foz
Inauguração: 1953
Lotação: 9000
Estado: Ativo

Clubes



Naval



Naval 1893

Figura 4-9 Detalhes do Estádio

4.2.3. Clubes e Equipas

Ao criar e/ou editar um clube, é possível associá-lo a um respetivo estádio e/ou associação de futebol. Estas associações irão permitir que os clubes apareçam na respetiva página dos estádios.

País:
Portugal

Cidade:
Vila das Aves

Associação:
Associação de Futebol do Porto (Portugal)

Estádio:
Estádio do Clube Desportivo das Aves

Salvar Cancelar

Figura 4-10 Associações no Clube – Administrador

Além disso, os clubes precisam de ter equipas e, para isso, existe o botão “Ver Equipas” (Fig. 4-11) que redireciona o Administrador para a páginas destas onde as poderá criar e associar a um clube (Fig. 4-12).

FutebolStats Associação Futebol Estádios Clubes Competições Jogos Jogadores Perfil

Lista de Clubes

Pesquisar por nome do clube...

Adicionar Clube Ver Equipas

Nome	Ano Fundação	Localidade	Estado	Associação & Estádio	Ações
AFS	2023	Portugal, Vila das Aves	Ativo	Associação de Futebol do Porto Estádio do Clube Desportivo das Aves	Ver Editar Apagar
Arouca	1952	Portugal, Arouca	Ativo	Associação de Futebol de Aveiro	Ver Editar Apagar

Figura 4-11 Ver Equipas – Administrador

Adicionar Equipa

Clube:
Escolha o clube

Nome da Equipa:
Escreva o nome da Equipa

Estado:
Selecione a Situação


Salvar Cancelar

Figura 4-12 Adicionar Equipa – Administrador

Na página do clube, é possível adicioná-lo e removê-lo dos favoritos, caso o utilizador não seja um administrador. Aqui, é possível escolher uma das respetivas equipas e ver os jogadores destas separados por posições gerais (Guarda-Redes, Defesa, etc).

FutebolStats
Associações Futebol
Estádios
Clubes
Competições
Jogos
Jogadores
Perfil





Detalhes do Jogador



9. Viktor Gyökeres
Nacionalidade: Suécia
Clube: Sporting
Valor de Mercado: 77.0 M€

Dados Pessoais
Idade: 26 anos
Altura: 189 cm
Peso: 90.0 kg

Jogos Jogados

 Porto	25/12/2024 Ver Detalhes	Sporting 
 Porto	29/08/2024 Ver Detalhes	Sporting 

Informações Adicionais
Posição: Avançado (PL)
Número de Camisola: 9
Número de Jogos: 3
Número de Golos: 3
Equipa: Equipa Principal
Situação: Ativo

Figura 4-15 Detalhes Jogador

4.2.5. Jogos e Estatísticas

No controlo das informações dos jogos, não é feito através de associações escolhidas pelo administrador. Em vez disso, o administrador escolhe a opção “Editar Estatísticas” e será levado para a página onde poderá adicionar/editar/apagar estas, sem ter de associar um jogo manualmente.

FutebolStats
Associações Futebol
Estádios
Clubes
Competições
Jogos
Jogadores
Perfil

Estatísticas dos Jogos

Golos
Adicionar Golo

Clube	Jogador	Minuto	Pênalti	Ações
Benfica	Ángel Di María	36	Sim	Editar Apagar
Sporting	Viktor Gyökeres	45 + 3	Não	Editar Apagar
Sporting	Viktor Gyökeres	63	Não	Editar Apagar
Benfica	Andreas Schjelderup	88	Não	Editar Apagar

Penaltis
Adicionar Penalti

Nº do Pênalti	Clube	Jogador	Golo	Ações
1	Benfica	Ángel Di María	Sim	Editar Apagar
2	Sporting	Viktor Gyökeres	Não	Editar Apagar
3	Benfica	Nicolás Otamendi	Sim	Editar Apagar

Figura 4-16 Estatísticas de um Jogo – Administrador

A página de administração (Fig.4-16) e de visualização (Fig. 4-18) estão separadas em 4 setores sendo, cada um destes, um dos respetivos tipos de estatística: golos, penáltis, faltas e substituições.

Detalhes do Jogo



Benfica

2 - 2
4 - 1



Sporting

Ficha do Jogo

Competição:

Alianz Cup 2024/2025

Data:

Feb. 20, 2025, 14:00

Local:

Portugal, Leiria

Estádio:

Estádio Municipal de Leiria

Estado:

Terminado

Vencedor:

Benfica

Figura 4-17 Detalhes Jogo

Estatísticas do Jogo

Golos

Faltas

Substituições

Penáltis

Clube Casa		Clube Fora	
11. Ángel Di María	Golo	9. Viktor Gyökeres	Falhado
30. Nicolás Otamendi	Golo	21. Ricardo Pereira	Golo
3. Álvaro Carreras	Golo	17. Nuno Costa	Falhado
7. Zeki Amdouni	Golo		

Figura 4-18 Detalhes Estatísticas de um Jogo

5. Conclusões

Este projeto permitiu-nos melhorar as nossas habilidades na framework Django como software MongoDB. Para o desenvolvimento deste projeto foi utilizado a matéria dada na cadeira de Base de Dados 2 assim como pesquisas adicionais. Este permitiu produzir código de maneira eficaz e organizada, sendo dividido nas respectivas funcionalidades (Models, Forms, etc).

A utilização do MongoDB e pgAdmin (postgres) é uma ótima ferramenta para visualizar os dados do sistema, de forma intuitiva e fácil de compreender.

6. Referências

7. Bibliografia

Aulas de Base de Dados 2 2024-2025

PDFs disponibilizados no Moodle 2024-2025 (moodle.estgv.ipv.pt) , Curso Base de Dados 2

