

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](#_Toc190041424)

[2. Base de Dados 2](#_Toc190041425)

[2.1. Criação da Base de Dados 2](#_Toc190041426)

[2.2. Configuração no Django 3](#_Toc190041427)

[2.2.1. Routers 4](#_Toc190041428)

[3. Framework Django 6](#_Toc190041429)

[3.1. Models 6](#_Toc190041430)

[3.1.1. ObjectId 6](#_Toc190041431)

[3.1.2. Relações 7](#_Toc190041432)

[3.1.3. Nome Tabela 7](#_Toc190041433)

[3.1.4. Devolver Ele Próprio 7](#_Toc190041434)

[3.2. Forms 7](#_Toc190041435)

[3.2.1. Campos Dropdown, Date e Time 8](#_Toc190041436)

[3.2.2. Relações 9](#_Toc190041437)

[3.2.3. Forms Utils 10](#_Toc190041438)

[3.2.4. Valores Iniciais 10](#_Toc190041439)

[3.2.5. Verificação 10](#_Toc190041440)

[3.3. Views 11](#_Toc190041441)

[3.3.1. Mostrar Valores 11](#_Toc190041442)

[3.3.2. Adicionar e Editar Valor 12](#_Toc190041443)

[3.3.3. Apagar Valor 13](#_Toc190041444)

[3.4. Templates 14](#_Toc190041445)

[3.4.1. Adicionar e Editar 14](#_Toc190041446)

[3.4.2. Apagar 14](#_Toc190041447)

[3.4.3. Erro ao Apagar 15](#_Toc190041448)

[3.5. Urls 15](#_Toc190041449)

[4. Funcionamento 17](#_Toc190041450)

[4.1. Script 17](#_Toc190041451)

[4.2. Programa 18](#_Toc190041452)

[5. Conclusões 19](#_Toc190041453)

[6. Referências 20](#_Toc190041454)

[7. Bibliografia 21](#_Toc190041455)

**Índice de tabelas**

[Tabela 3‑1 on\_delete das Relações - Django 7](#_Toc189949199)

**Índice de Figuras**

[Figura 2‑1 Modelo Conceptual 2](#_Toc190041456)

[Figura 2‑2 Modelo Físico 3](#_Toc190041457)

[Figura 2‑3 Configuração SGBD - Django 4](#_Toc190041458)

[Figura 2‑4 Configuração Router – Django 5](#_Toc190041459)

[Figura 2‑5 Router no Model - Django 5](#_Toc190041460)

[Figura 3‑1 Exemplo Model – Django 6](#_Toc190041461)

[Figura 3‑2 Biblioteca ObjectId – Django 6](#_Toc190041462)

[Figura 3‑3 Formulário – Django 8](#_Toc190041463)

[Figura 3‑4 Dropdown no Formulário – Django 8](#_Toc190041464)

[Figura 3‑5 Dia e Hora no Formulário - Django 9](#_Toc190041465)

[Figura 3‑6 Relações no Formulário – Django 9](#_Toc190041466)

[Figura 3‑7 Relações no Formulário 2 - Django 9](#_Toc190041467)

[Figura 3‑8 Funções de Conversão – Django 10](#_Toc190041468)

[Figura 3‑9 Valor Inicial Formulário - Django 10](#_Toc190041469)

[Figura 3‑10 Verificação Final do Formulário - Django 11](#_Toc190041470)

[Figura 3‑11 View para Listar – Django 11](#_Toc190041471)

[Figura 3‑12 View para Mostrar Detalhes - Django 12](#_Toc190041472)

[Figura 3‑13 View para Adicionar – Django 12](#_Toc190041473)

[Figura 3‑14 View para Editar - Django 13](#_Toc190041474)

[Figura 3‑15 View para Apagar – Django 13](#_Toc190041475)

[Figura 3‑16 View para Apagar com Atualizações - Django 13](#_Toc190041476)

[Figura 3‑17 Adicionar e Editar – Template Django 14](#_Toc190041477)

[Figura 3‑18 Modal Apagar - Template Django 15](#_Toc190041478)

[Figura 3‑19 Mostrar Erro ao Eliminar - Template Django 15](#_Toc190041479)

[Figura 3‑20 Urls - Django 16](#_Toc190041480)

[Figura 4‑1 Script de Dados – Django 17](#_Toc190041481)

[Figura 4‑2 Inserção de Dados Com Relações - Django 18](#_Toc190041482)

# 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.

# Base de Dados

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

## Criação da Base de Dados

Inicialmente, como qualquer projeto que envolve dados, é necessário criar a respetiva 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.

Uma imagem com texto, diagrama, Esquema, esquemático

Descrição gerada automaticamente

Figura ‑ Modelo Conceptual

Uma imagem com texto, diagrama, Esquema, Desenho técnico

Descrição gerada automaticamente

Figura ‑ Modelo Físico

## Configuração no Django

Um dos requisitos não funcionais era a utilização de duas SGBD (Sistema de Gerenciamento de Base de Dados) em que, cada um dos dados só podia ser armazenado numa destas. Depois de discussões, chegou-se à conclusão que seria utilizado o postgres para os utilizadores e o MongoDB para o resto dos dados.

No ficheiro “settings.py” foram adicionadas as respetivas bases de dados a ser utilizadas pelo programa, deixando o postgres como “default” (Fig. 2-3).

Uma imagem com texto, captura de ecrã, Tipo de letra

Descrição gerada automaticamente

Figura ‑ Configuração SGBD - Django

### 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”.

Uma imagem com texto, captura de ecrã, Tipo de letra

Descrição gerada automaticamente

Figura ‑ Configuração Router – Django

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

Uma imagem com texto, Tipo de letra, captura de ecrã, número

Descrição gerada automaticamente

Figura ‑ Router no Model - Django

# Framework Django

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

## 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).

Uma imagem com texto, captura de ecrã, Tipo de letra, software

Os conteúdos gerados por IA poderão estar incorretos.

Figura ‑ 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.

### 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.

Uma imagem com texto, Tipo de letra, captura de ecrã, número

Os conteúdos gerados por IA poderão estar incorretos.

Figura ‑ 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.

### 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 ‑ on\_delete das Relações - Django

### 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.

### 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.

## 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.

Uma imagem com texto, captura de ecrã, Tipo de letra, file

Os conteúdos gerados por IA poderão estar incorretos.

Figura ‑ 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.

### 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.

Uma imagem com texto, Tipo de letra, captura de ecrã

Os conteúdos gerados por IA poderão estar incorretos.

Figura ‑ Dropdown no Formulário – Django

Uma imagem com texto, captura de ecrã, Tipo de letra

Os conteúdos gerados por IA poderão estar incorretos.

Figura ‑ Dia e Hora no Formulário - Django

### Relações

Para as respetivas 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).

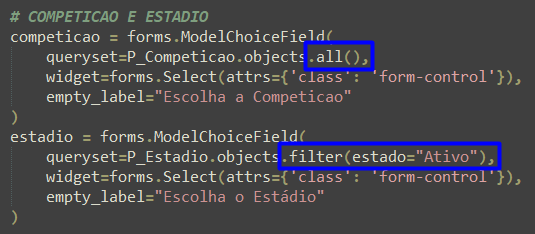


Figura ‑ Relações no Formulário – Django

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\_\_”.

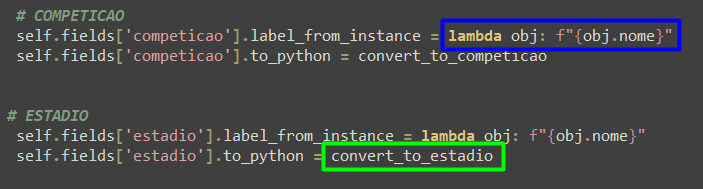


Figura ‑ Relações no Formulário 2 - Django

### 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.

Uma imagem com texto, captura de ecrã, Tipo de letra

Os conteúdos gerados por IA poderão estar incorretos.

Figura ‑ Funções de Conversão – Django

### Valores Iniciais

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

Uma imagem com texto, Tipo de letra, captura de ecrã, file

Os conteúdos gerados por IA poderão estar incorretos.

Figura ‑ Valor Inicial Formulário - Django

### 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).

Uma imagem com texto, captura de ecrã, Tipo de letra

Os conteúdos gerados por IA poderão estar incorretos.

Figura ‑ Verificação Final do Formulário - Django

## 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.

### 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.

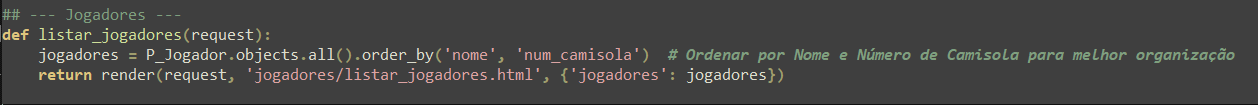


Figura ‑ View para Listar – Django

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.

Uma imagem com texto, captura de ecrã, Tipo de letra, software

Os conteúdos gerados por IA poderão estar incorretos.

Figura ‑ View para Mostrar Detalhes - Django

### 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.

Uma imagem com texto, captura de ecrã, Tipo de letra

Os conteúdos gerados por IA poderão estar incorretos.

Figura ‑ 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-los ao utilizador.

Uma imagem com texto, captura de ecrã, Tipo de letra

Os conteúdos gerados por IA poderão estar incorretos.

Figura ‑ View para Editar - Django

### Apagar Valor

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

Uma imagem com texto, captura de ecrã, Tipo de letra, file

Os conteúdos gerados por IA poderão estar incorretos.

Figura ‑ 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.

Uma imagem com texto, captura de ecrã, Tipo de letra, software

Os conteúdos gerados por IA poderão estar incorretos.

Figura ‑ View para Apagar com Atualizações - Django

## Templates

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

### 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.

Uma imagem com texto, captura de ecrã, Tipo de letra

Os conteúdos gerados por IA poderão estar incorretos.

Figura ‑ Adicionar e Editar – Template Django

### 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.

Uma imagem com texto, captura de ecrã, Tipo de letra

Os conteúdos gerados por IA poderão estar incorretos.

Figura ‑ Modal Apagar - Template Django

### 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.

Uma imagem com texto, captura de ecrã, Tipo de letra

Os conteúdos gerados por IA poderão estar incorretos.

Figura ‑ Mostrar Erro ao Eliminar - Template Django

## 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.

Uma imagem com texto, captura de ecrã, Tipo de letra

Os conteúdos gerados por IA poderão estar incorretos.

Figura ‑ Urls - Django

# Funcionamento

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

## 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.

Uma imagem com texto, captura de ecrã, Tipo de letra

Descrição gerada automaticamente

Figura ‑ Script de Dados – Django

Uma imagem com texto, captura de ecrã, Tipo de letra

Descrição gerada automaticamente

Figura ‑ Inserção de Dados Com Relações - Django

## Programa

XXX

# 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 respetivas 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.

# Referências

# Bibliografia

Aulas de Base de Dados 2 2024-2025

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