Labs 8-9: Django Tutorial

Fast notes on setting up Django and the guitars example application
Courses: Database Systems, Databases and Information Analysis
Author: Pedro Furtado, professor at University of Coimbra, Portugal.
(The first version of the guitars application was developed by Andre Perrota for this course. It was then changed by Pedro Furtado and motogp was added, together with installation and the modifications notes.)

1. (Lab 8) Setup do django e do python

Instale o python e o django:

Install Python

Install Django

Mais instruções de instalação em:

https://docs.djangoproject.com/en/4.2/topics/install/

Quando estiver correctamente instalado, deverá mostrar o seguinte ecran quando corre o seguinte:

```
django-admin startproject firsttestdjango
```

```
cd firsttestdjango

python manage.py runserver
```

Assim que este teste funcione, parabéns, conseguiu instalar o django.

PARE O SERVIDOR FECHANDO A LINHA DE COMANDO.

2. (Lab 8) Setup to run guitars

1. Descomprime o zip do tutorial de guitars. coloque-o numa directoria bem definida:

   MacOS: pode criar uma subdirectoria dentro da directoria Documentos que se chame guitars
   Windows: pode criar uma subdirectoria dentro da directoria Desktop ou Users chamada guitars

2. Edita ficheiro settings.py para alterar a secção DATABASES

   a. Procure a localização do texto:

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.postgresql',
        'NAME': 'brGuitars',
        'USER': 'perrotta',
        'PASSWORD': 'perrotta',
        'HOST': 'localhost',
        'PORT': '5432',
    }
}
```

   b. Modifique o username e password para coincidir com um username e password que tenha no pgadmin (postgres). Normalmente temos usado o username postgres, e você definiu a password desse username (por exemplo postgres).

No meu caso ficou:

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.postgresql',
        'NAME': 'brGuitars',
        'USER': 'postgres',
        'PASSWORD': 'postgres',
        'HOST': 'localhost',
        'PORT': '5432',
    }
}
```

3. Note que a base de dados que é indicada no DATABASES, em 'NAME' é brGuitars. Crie uma base de dados com esse nome, uma vez que será a base de dados que vai usar.

4. Abra uma linha de comandos.

MacOS:

a. Corra o comando terminal no spotlight, ou clique no icon terminal

b. Mude para a directoria onde esta o guitars usando o comando cd. Por exemplo, se o guitars estiver na directoria home/pedro/Documents/guitars, basta por o comando:

cd home/pedro/Documents/guitars

Windows:

a. Va para a directoria guitars usando o explorador de directorias

b. No explorador de directorias toque duas vezes no espaço que mostra o caminho em que está e digite cmd nesse espaço. Deverá abrir-se uma janela com a linha de comandos na directoria em que está, que deverá ser a guitars.

5. Corra o servidor web do django: DENTRO DA DIRECTORIA DO PROJECTO (guitars), i.e. na directoria que tem o manage.py:

python manage.py runserver

nota: No MAC OS pode ter de usar (sempre) pip3 e python3 em vez de pip e python, devido a ser a versão 3. Por isso onde a instrução diz pip ou python, se necessário use pip3 e python3.

6. Corrija os "fatal errors" se estes aparecerem

Poderão faltar dois pacotes, o que dará erro quando corre o servidor usando o comando acima. Como o guitars esta a usar postgres, um dos erros terá a ver com falta do pacote para ligar ao postgres (psycopg2). O outro erro poderá ser a falta de módulo import_export. Para os instalar, corra:

Windows:

pip install psycopg2

pip install django-import_export

MAC OS: (onde esta pip poderá ser pip3)

pip install psycopg2-binary

pip install django-import_export

Poderá ainda ter erros na ligação à base de dados. Para já, terá de ter criado a base de dados brGuitars no postgres; em segundo lugar, terá de ter configurado correctamente o utilizador (nome e password) no ficheiro settings.py.

7. Observe o ficheiro models.py que define as relações (tabelas) da base de dados. Estas são definidas como classes, e as classes não são uteis apenas como tabelas na base de dados mas também como estruturas da aplicação para conter os dados enquanto estão a ser usados (daí serem definidos como classes). No modelo um luthier é uma pessoa que constrói guitarras. De acordo com a wikipedia, o luthier "é um profissional especializado na construção e no reparo de instrumentos de cordas, com caixa de ressonância. Isto inclui o violão, violinos, violas, violoncelos, contrabaixos, violas da gamba e todo tipo de guitarras (acústica, elétrica, clássica), alaúdes, archilaúdes, tiorbas, e bandolins." (https://pt.wikipedia.org/wiki/Violeiro_(Luthier))

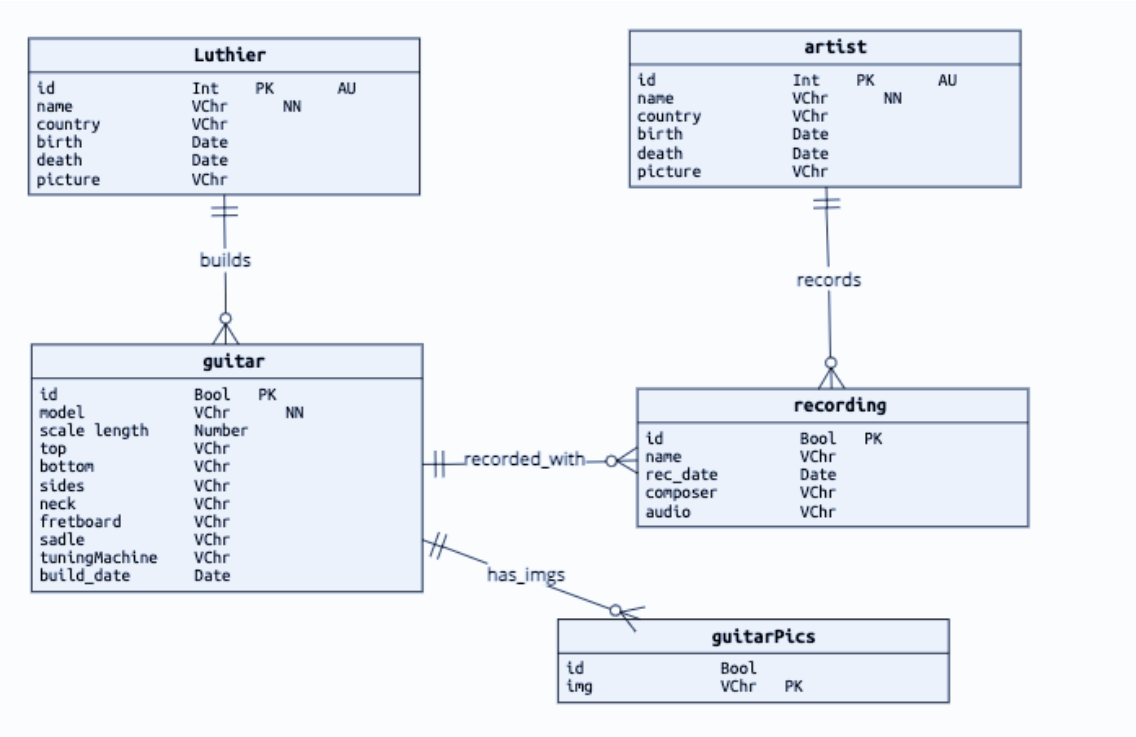


Figura 1. Modelo entidade-relacionamento das guitarras

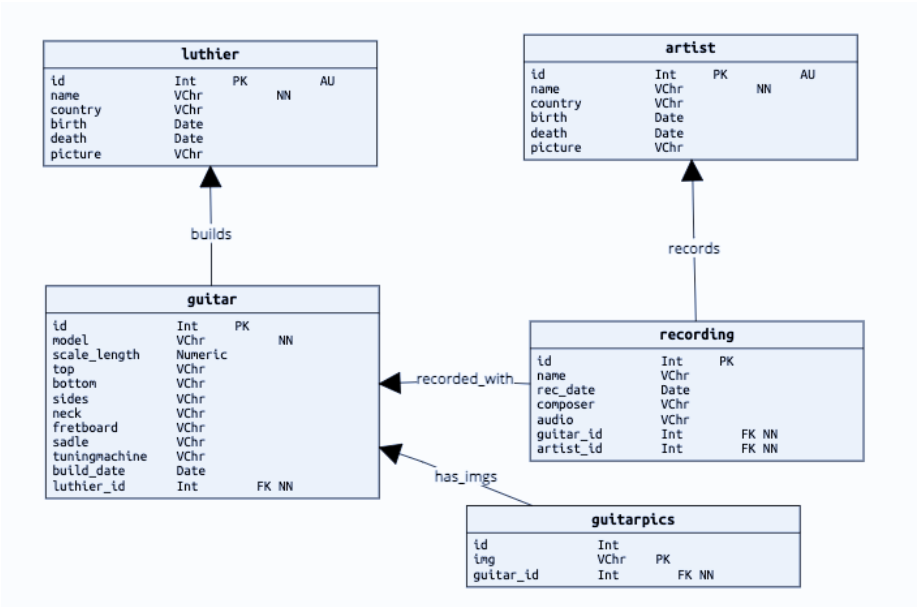


Figura 2. Modelo físico das guitarras

O modelo correspondente que definimos em models.py é:

```
from django.db import models

class Luthier(models.Model):
    name = models.CharField(max_length=20, null=False, blank=False)
    country = models.CharField(max_length=200, null=True, blank=True)
    birth = models.IntegerField(null=True, blank=True)
    death = models.IntegerField(null=True, blank=True)
    pic = models.URLField(max_length=500, null=True, blank=True)
    def __str__(self):
        return self.name
    def isAlive(self):
        return death==False
```

```
class Guitar(models.Model):
    luthier = models.ForeignKey(Luthier, on_delete=models.CASCADE)
    model = models.CharField(max_length = 200, null=False, blank=False)
    year = models.IntegerField(null=True, blank=True)
    top = models.CharField(max_length=200, null=True, blank=True)
    bottom = models.CharField(max_length=200, null=True, blank=True)
    sides = models.CharField(max_length=200, null=True, blank=True)
    neck = models.CharField(max_length=200, null=True, blank=True)
    fretboard = models.CharField(max_length=200, null=True, blank=True)
    sadle = models.CharField(max_length=200, null=True, blank=True)
    tuningMachines = models.CharField(max_length=200, null=True, blank=True)
    def __str__(self):
        return self.luthier.name+"-"+self.model

class GuitarPic(models.Model):
    img = models.URLField(max_length=500, null=False, blank=False)
    guitar = models.ForeignKey(Guitar, on_delete=models.CASCADE)
    def __str__(self):
        return self.img
```

```python
class Artist(models.Model):
    name = models.CharField(max_length=200, null=False, blank=False)
    country = models.CharField(max_length=200, null=True, blank=True)
    birth = models.IntegerField(null=True, blank=True)
    death = models.IntegerField(null=True, blank=True)
    pic = models.URLField(max_length=500, null=True, blank=True)
    def __str__(self):
        return self.name
    def isAlive(self):
        return death==False

class Recording(models.Model):
    name = models.CharField(max_length=200, null=False, blank=False)
    year = models.IntegerField(null=True, blank=True)
    composer = models.CharField(max_length=200)
    arrangement = models.CharField(max_length=200, null=True, blank=True)
    audio = models.URLField(max_length=500, null=False, blank=False)
    artist = models.ForeignKey(Artist, on_delete=models.CASCADE)
    guitar = models.ForeignKey(Guitar, on_delete=models.CASCADE, null=True, blank=True)
    def __str__(self):
        return self.name
```

Figura 3. Models.py

No models.py repare como são definidos os tipos de dados e como são definidas as chaves estrangeiras (**foreign key**). Um ponto importante é que neste models.py não definimos chaves primárias, sendo que nesse caso o **django cria automaticamente chaves primárias** para cada entidade com o nome id. Poderiamos ter optado por definirmos chaves primárias nossas, possibilidade que não descrevemos aqui.

A linguagem usada pelo django é o python, onde a identação (tabs/espaços colocados desde a borda esquerda do ficheiro) são importantes (o código tem de estar adequadamente identado para não dar erro).

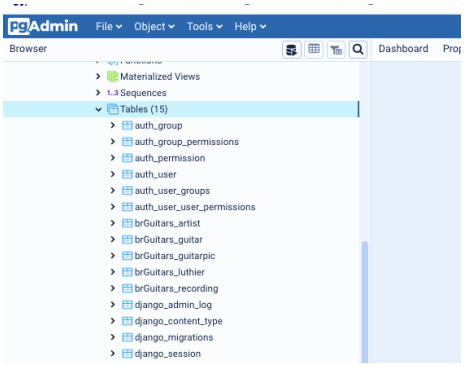
Uma **"class" contém atributos**, tal como uma tabela, e pode conter também funções. Cada luthier especifico (linha de uma tabela) vai corresponder a um **objecto da classe**. Note a palavra chave def. São definições (**def**) de funções. Uma função recebe parâmetros, faz qualquer coisa necessária e devolve resultados. No caso do models temos funções muito básicas. Em particular, **a função __str__** é a função que é usada sempre que alguma acção no django pretende mostrar o conteúdo de um objecto da classe. Por exemplo, **na Figura 3 definimos que quando o django pretende mostrar o conteúdo de uma guitarra, deve mostrar o nome do luthier e o modelo da guitarra**.

8. Corra os seguintes comandos que permitem criar SQL e corrê-lo sobre a base de dados para "migrar" as tabelas para a base de dados criada. Estes comandos pegam no models.py e criam as tabelas correspondentes às figuras 1 e 2 acima.

   python manage.py makemigrations brGuitars
   python manage.py migrate

   Deve ver algo como mostrado a seguir na linha de comandos e deve também ver que as tabelas brGuitars terão sido criadas na base de dados correspondente.

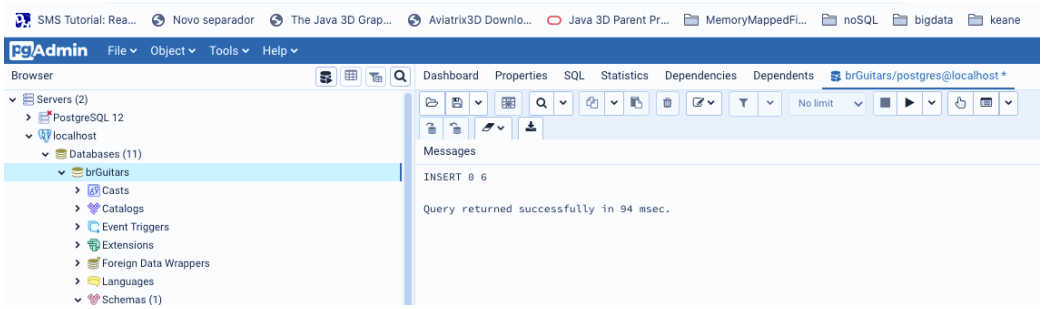
```
Macs-MacBook-Pro-4:guitars pedro$ python manage.py migrate
Operations to perform:
  Apply all migrations: admin, auth, brGuitars, contenttypes, sessions
Running migrations:
  Applying contenttypes.0001_initial... OK
  Applying auth.0001_initial... OK
  Applying admin.0001_initial... OK
  Applying admin.0002_logentry_remove_auto_add... OK
  Applying admin.0003_logentry_add_action_flag_choices... OK
  Applying contenttypes.0002_remove_content_type_name... OK
  Applying auth.0002_alter_permission_name_max_length... OK
  Applying auth.0003_alter_user_email_max_length... OK
  Applying auth.0004_alter_user_username_opts... OK
  Applying auth.0005_alter_user_last_login_null... OK
  Applying auth.0006_require_contenttypes_0002... OK
  Applying auth.0007_alter_validators_add_error_messages... OK
  Applying auth.0008_alter_user_username_max_length... OK
  Applying auth.0009_alter_user_last_name_max_length... OK
  Applying auth.0010_alter_group_name_max_length... OK
  Applying auth.0011_update_proxy_permissions... OK
  Applying auth.0012_alter_user_first_name_max_length... OK
  Applying brGuitars.0001_initial... OK
  Applying sessions.0001_initial... OK
Macs-MacBook-Pro-4:guitars pedro$
```



9. Insira dados na base de dados.

   O que fizémos antes foi criar a estrutura das tabelas. No passo seguinte pretendemos inserir dados de exemplo directamente na base de dados para podermos ver as guitarras e os luthiers.

   Usando o Query Tool do pgAdmin na base de dados brGuitars criada, corra os inserts do script **postgreData.sql** que vem dentro da dir guitars. Já aprendemos a correr SQL em aulas anteriores da cadeira, é só repetir esse processo mas agora para criar a base de dados das guitarras.



10. Arranque o servidor web do django como fez nos tutoriais anteriores. Para tal, dentro da directoria guitars, arranque usando: **python manage.py runserver**

11. Veja no browser a aplicação: **localhost:8000/brGuitars**

12. Finalmente, vamos examinar e o professor vai descrever os três principais ficheiros e os templates para você perceber bem o que foi feito nesta aplicação: **urls.py, views.py, models.py** e por fim os Templates.

Ficheiro urls.py:

O ficheiro seguinte é chamado **urls.py** e tem uma função muito relevante: **quando se escreve um endereço (URL) no browser que corresponda a um determinado padrão, chama uma função presente no ficheiro views.py.**

Por exemplo, **o primeiro path** indica que se não se puser nada no URL (‘’) chama a função índex do views.py. Note-se que “não pôr nada” no URL corresponde a pôr http://localhost:8000/brGuitars/, uma vez que a aplicação se chama brGuitars.

O segundo path indica que se o URL for http://localhost:8000/brGuitars/luthiers/, será chamada a função views.luthiers.

O terceiro path indica que se o URL for http://localhost:8000/brGuitars/luthiers/1/ será chamada a função views.luthierDetails. A ideia nesse caso será a função luthierDetails receber o valor 1 como parâmetro e por isso ir buscar o luthier com id 1 e mostrar os detalhes desse luthier.

Os restantes path têm uma lógica semelhante a estes acima.

urls.py

```
from django.urls import path

from . import views

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

    #Luthiers
    path('luthiers', views.luthiers, name='luthiers'),
    path('luthiers/<int:luthier_id>/', views.luthierDetails, name='luthierDetails'),

    #artists
    path('artists', views.artists, name='artists'),
    path('artists/<int:artist_id>/', views.artistDetails, name='artistDetails'),

    #recordings
    path('recordings', views.recordings, name='recordings'),
    path('recordings/<int:rec_id>/', views.recordingDetails, name='recordingDetails'),

    #guitars
    path('guitars', views.guitars, name='guitars'),
    path('guitars/<int:guitar_id>/', views.guitarDetails, name='guitarDetails'),

    #guitarPics
    path('guitarPics', views.guitarPics, name='guitarPics'),
]
```

Figure 4. urls.py

Ficheiro views.py:

O ficheiro views.py tem as funções que são chamadas quando se põe um dado URL.

Função index:

A **função índex abaixo**, que como vimos atrás é chamada quando pomos http://localhost:8000/brGuitars/, **carrega um “template” chamado índex.html** e **cria 5 variaveis** (tais como numLuthiers), cada uma com o número de elementos (linhas) de cada uma das tabelas definidas na base de dados, **enviando essas 5 variáveis para o template processar**. Para sabermos quantas linhas tem cada tabela chamamos a função count(), tais como em **Luthier.objects.all().count().** Luthier significa que nos referimos à **tabela de luthiers** (através da classe Luthier definida em models.py), objects significa que queremos os objectos, isto é, as linhas da tabela, all() significa que queremos todas as linhas (em vez de alguma condição que buscaria só algumas linhas), e por fim count() é a função que ordena que sejam contados os números de linhas devolvidos.

```
views.py
from django.http import HttpResponse
from django.shortcuts import render
from django.template import loader

from .models import Luthier
from .models import Guitar
from .models import Artist
from .models import GuitarPic
from .models import Recording

def index(request):
    template = loader.get_template('brGuitars/index.html')
    context = {
        'numLuthiers':Luthier.objects.all().count(),
        'numGuitars':Guitar.objects.all().count(),
        'numArtists':Artist.objects.all().count(),
        'numRecordings':Recording.objects.all().count(),
        'numGuitarPics':GuitarPic.objects.all().count(),
    }
    return HttpResponse(template.render(context, request))
```

Função luthiers:

A função luthiers, que como vimos antes é chamada quando pomos http://localhost:8000/brGuitars/luthiers/, vai carregar o template brGuitars/luthiers.html e envia para esse template a variável luthiers que contém todos os luthiers ordenados por nome. Como é que eu sei que tem todos os luthiers? Porque a variável é preenchida com items, e items é Luthier.objects.order_by('name')[0:]. Luthier.objects vai buscar todos os objectos do tipo luthier, isto é, as linhas da tabela luthier; order_by('name') vai ordenar essas linhas buscadas por nome; [0:] significa todos os elementos (linhas) começando na primeira (linha de índice 0) e acabando na ultima, já que em python um : sem nada a seguir quer dizer até ao final do conjunto.

Função luthiersDetails:

Não vamos descrever a função luthierDetails porque é semelhante, a única diferença que vale a pena referir é o paramtero de entrada luthier_id, que corresponde por exemplo ao 1 de http://localhost:8000/brGuitars/luthiers/1/, e dessa forma o facto de eu ir buscar um so luthier baseado na chave primaria (id) ser esse valor (myLuthier = Luthier.objects.get(pk=luthier_id). Também interessante é o facto de que vou buscar as guitarras desse luthier através de uma função filter sobre a chave estrangeira luthier da tabela guitar (myLuthierGuitars=Guitar,objects.filter(luthier = luthier_id)). Mais uma vez, as variáveis assim criadas vao ser enviadas para o template (brGuitars/luthierDetails.html) processar.

```python
def luthiers(request):
    template = loader.get_template('brGuitars/luthiers.html')
    items = Luthier.objects.order_by('name')[0:]
    context = {
        'luthiers':items
    }
    return HttpResponse(template.render(context, request))

def luthierDetails(request, luthier_id):
    template = loader.get_template('brGuitars/luthierDetails.html')
    try:
        myLuthier = Luthier.objects.get(pk=luthier_id)
        myLuthierGuitars = Guitar.objects.filter(luthier = luthier_id)
        context = {'luthier' : myLuthier, 'guitars' : myLuthierGuitars}
    except Luthier.DoesNotExist:
        raise Http404("Luthier does not exist")

    return HttpResponse(template.render(context, request))
```

Template index.html:

The template índex.html receives as parameter the variables prepared in the index function of views.py (numLuthiers, numArtists, numGuitars, …). It is an html file which can use the typical html elements, but it can also use any of the variables it received as parameters, in that case enclosed by {{ to start and }} to end the reference to it. Django will automatically process those references by the contents of the variables.

In this case we have the typical html tags <html>, <head>,<title> and <body>, and then we have anchors to allow the user to click to link to each page containing lists of luthiers, artistics, guitars and so on. For instance, the anchor link {{numLuthiers}} Luthiers enclosed as it is between <a> and </<> will link to the page /brGuitars/luthiers, which means it will open that page if clicked upon. The link will show the text "x Luthiers", where x is the number of luthiers in the database. For instance, if there exist 5 luthiers in the database, then it will show "5 luthiers". The value x is obtained using the variable numLuthiers that was received as a parameter. That number is referenced in the template using {{numLuthiers}}.

index.html

```html
<html>
<head>
<meta charset="UTF-8">
<title>brGuitars Database</title>
</head>
<body>
<h1>brGuitars Database</h1>
<h2><a href="/brGuitars/luthiers"> {{numLuthiers}} Luthiers</a></h2>
<h2><a href="/brGuitars/artists"> {{numArtists}} Artists</a></h2>
<h2><a href="/brGuitars/guitars"> {{numGuitars}} Guitars</a></h2>
<h2><a href="/brGuitars/recordings"> {{numRecordings}} Recordings</a></h2>
<h2><a href="/brGuitars/guitarPics"> {{numGuitarPics}} Guitar Pictures</a></h2>
```

Template luthiers.html:

The next example is for the template /brGuitars/luthiers, which lists the names of all luthiers. First of all, it includes an anchor (a link) to the homepage Home</p>, which allows the user to go back to the index page. We have seen before that the URL /brGuitars/ (empty suffix ‘ ’)calls the index function in views.py and will open the index template, which is the homepage in this case.

The next part of the file is html code to show a list of something. The tag <ul> means the start of an unordered list, and the tag closes that list. The tag <li> means list item (closed by) and it shows an item of the list. Code for a for loop that iterates through the variable that was input to the template by the caller function in views.py (function luthiers) as a parameter is added using the {% and %} delimiters. The code is easy to understand. The variable luthiers contains a set of luthiers (all luthiers), so the for line {% for luthierROW in luthiers %} is simply iterating through the rows of luthiers. In each iteration the row is placed in the variable luthierROW. The end of the for loop is the line {% end for %}.

Inside the <li> we have a link to a specific luthier. The link is whatever is inside the <a> tag, and the text shown is whatever is between the <a> and the tags. The href="/brGuitars/luthiers/{{luthierROW.id}}/" code creates a URL to the specific luthier, based on luthierROW.id. This will be replaced by ="/brGuitars/luthiers/1, ="/brGuitars/luthiers/2, ="/brGuitars/luthiers/3 and so on for the iterations of the for loop.

Based on what you can see in the urls.py file shown in Figure x, these references will trigger a call to the function views.luthierDetails, which will load another template to show the details of that luthier (luthierDetails.html). The text shown in the link will be the name of the luthier (that’s the code {{ luthierROW.name }}).

luthiers.html

```
<a href="/brGuitars/">Home</a></p>
{% if luthiers %}
    <ul>
    {% for luthierROW in luthiers %}
        <li><a href="/brGuitars/luthiers/{{luthierROW.id}}/">{{ luthierROW.name }}</a></li>
    {% endfor %}
    </ul>
{% else %}
    <p>No luthiers to show.</p>
{% endif %}
```

Template luthierDetails.html:

Try to understand by yourself what luthierDetails.html is doing:

luthierDetails.html

```html
<a href="/brGuitars/">Home</a></p>
<h1>{{luthier.name}} details</h1>
<picture>
 <source media="(min-width:320px)" srcset={{luthier.pic}}>
 <source media="(min-width:240px)" srcset={{luthier.pic}}>
 <img src={{luthier.pic}} alt={{luthier.name}}-{{luthier.pic}} style="width:auto;">
</picture>
<h3>Country: {{luthier.country}}</h3>
<h3>Birth: {{luthier.birth}}</h3>
<h3>Death: {{luthier.death}}</h3>
<h3>Guitars:</h3>
{% if guitars %}
    <ul>
    {% for guitarROW in guitars %}
        <li><a href="/brGuitars/guitars/{{guitarROW.id}}/">{{guitarROW.model}}</a></li>
    {% endfor %}
    </ul>
{% else %}
    <p>No guitar to show.</p>
{% endif %}
```

3. Try to setup and use admin UI to add data manually

There is also a predefined user interface to allow you to add new data without coding. You can try to setup this UI and use it. Follow the following instructions (pasted from https://docs.djangoproject.com/en/1.8/intro/tutorial02/), but replace by your case:

Creating an admin user

First we'll need to create a user who can login to the admin site. Run the following command:

```
$ python manage.py createsuperuser
```

Enter your desired username and press enter.

```
Username: admin
```

You will then be prompted for your desired email address:

```
Email address: admin@example.com
```

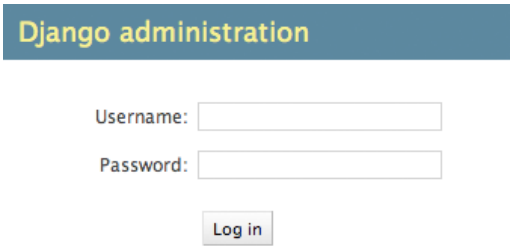
The final step is to enter your password. You will be asked to enter your password twice, the second time as a confirmation of the first.

```
Password: **********

Password (again): *********

Superuser created successfully.
```

Now, open a Web browser and go to "/admin/" on your local domain – e.g., http://127.0.0.1:8000/admin/. You should see the admin's login screen:



Explore the free admin functionality¶

No ficheiro admin.py verifique se estão já registadas as classes Luthier, Guitar, Artist, etc:

`brGuitars/admin.py`

```
from django.contrib import admin


from .models import Luthier
from .models import Guitar
…
admin.site.register(Luthier)
admin.site.register(Guitar)
…
```

1. Tente adicione uma instância de cada coisa (1 luthier, 1 guitar, 1 guitarpic, 1 artist) usando o inteface admin. Veja depois na web se mostra com os novos elementos.
2. Tente agora adicionar mais um de cada mas através de sql no pgadmin. Para tal consulte o script SQL e adicione os novos. Verifique depois novamente no interface web se ficaram adicionados.

4. (Lab 9) Create motogp application from scratch

Step 1. In the cmd line run:

```
django-admin startproject motogp
```

Step 2. Get into the new motogp folder and run the cmd line run:

```
python manage.py startapp brMotogp
```

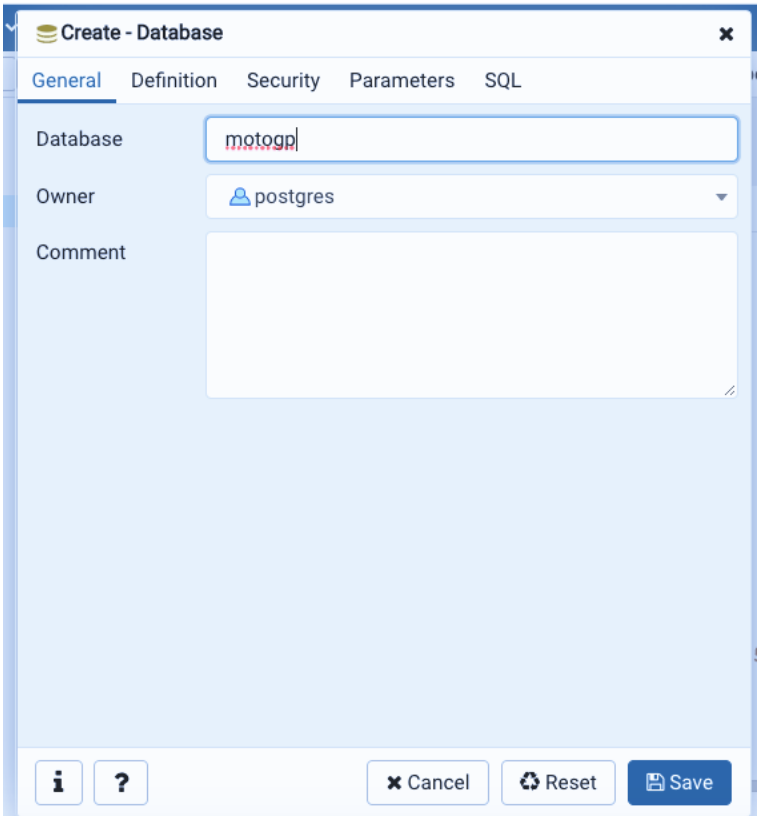
The new application should run by running Django application server:

```
python manage.py runserver
```

go to http://127.0.0.1:8000/, you should see a page running there

If it does not run, you need help from the teacher, and/or try fixes from anexo 10.

Step 3. Create new database motogp in postgres (pgadmin) and modify the database to be used by the Django motogp application.

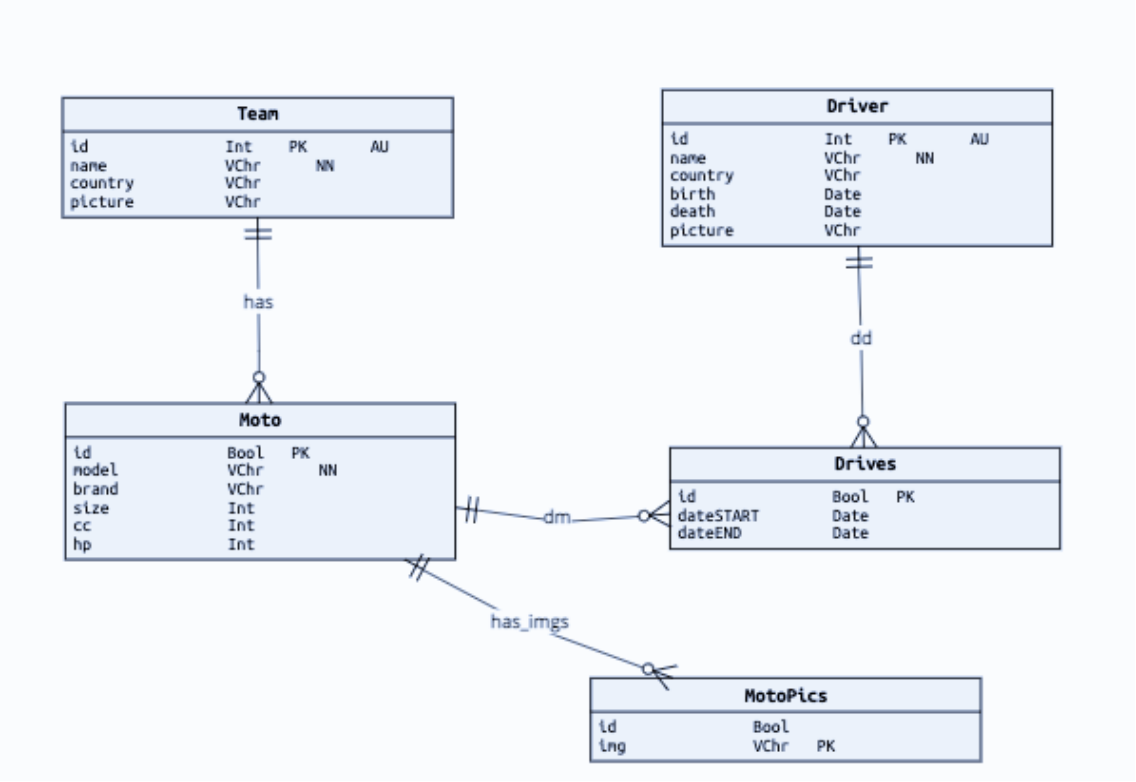


Change DATABASES section in settings.py:

```
settings.py
82  # }
83  DATABASES = {
84      'default': {
85          'ENGINE': 'django.db.backends.postgresql',
86          'NAME': 'motogp',
87          'USER': 'postgres',
88          'PASSWORD': 'postgres',
89          'HOST': 'localhost',
90          'PORT': '5432',
91      }
92  }
93
```

Step 4. Now we are ready to create our new database. The motogp schema is shown next, as well as the previous Guitars schema.

Motogp:



Step 5. Since we are using Django, we will now create the model for the intended database. Just add the correct model classes to the existing classes in models.py.

I did this by copy-pasting the model for guitars, and modifying the new classes to become as shown next.

(after the previous guitars model classes☺):

```
models.py
class Team(models.Model):
    name = models.CharField(max_length=20, null=False, blank=False)
    country = models.CharField(max_length=200, null=True, blank=True)
    pic = models.URLField(max_length=500, null=True, blank=True)
    def __str__(self):
        return self.name

class Moto(models.Model):
    team = models.ForeignKey(Team, on_delete=models.CASCADE)
    model = models.CharField(max_length = 200, null=False, blank=False)
    brand = models.CharField(max_length = 200, null=False, blank=False)
    size = models.IntegerField(null=True, blank=True)
    cc = models.IntegerField(null=True, blank=True)
    hp = models.IntegerField(null=True, blank=True)
    def __str__(self):
        return self.team.name+"-"+self.model

class MotoPic(models.Model):
    img = models.URLField(max_length=500, null=False, blank=False)
    moto = models.ForeignKey(Moto, on_delete=models.CASCADE)
    def __str__(self):
        return self.img

class Driver(models.Model):
    name = models.CharField(max_length=200, null=False, blank=False)
    country = models.CharField(max_length=200, null=True, blank=True)
    birth = models.IntegerField(null=True, blank=True)
    death = models.IntegerField(null=True, blank=True)
    pic = models.URLField(max_length=500, null=True, blank=True)
    def __str__(self):
        return self.name
    def isAlive(self):
        return death==False

class Drives(models.Model):
    dateSTART = models.IntegerField(null=True, blank=True)
    dateEND = models.IntegerField(null=True, blank=True)
    driver = models.ForeignKey(Driver, on_delete=models.CASCADE)
    moto = models.ForeignKey(Moto, on_delete=models.CASCADE, null=True, blank=True)
    def __str__(self):
        return self.name
```

Code available in appendix 12.

Step 6. Register the brMotogp app in motogp project. In order to do that, add the following to the beginning of INSTALLED_APPS in settings.py file:

```
INSTALLED_APPS = [
    'brMotogp.apps.BrMotogpConfig',
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
]
```

Now go to brMotogp folder and in apps.py make sure you have:

class BrMotogpConfig(AppConfig):

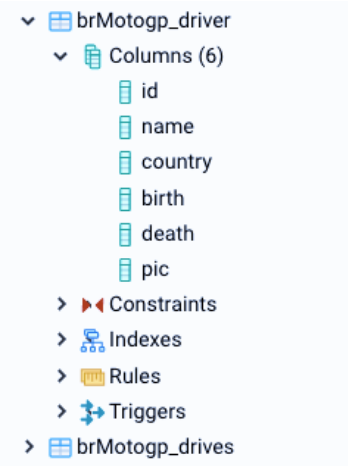
default_auto_field = 'django.db.models.BigAutoField'
name = 'brMotogp'

note that the capital letters must be correct in all parts.

Step 7. Now migrate the updated models into the new database:

```
python manage.py makemigrations brMotogp
python manage.py migrate
```

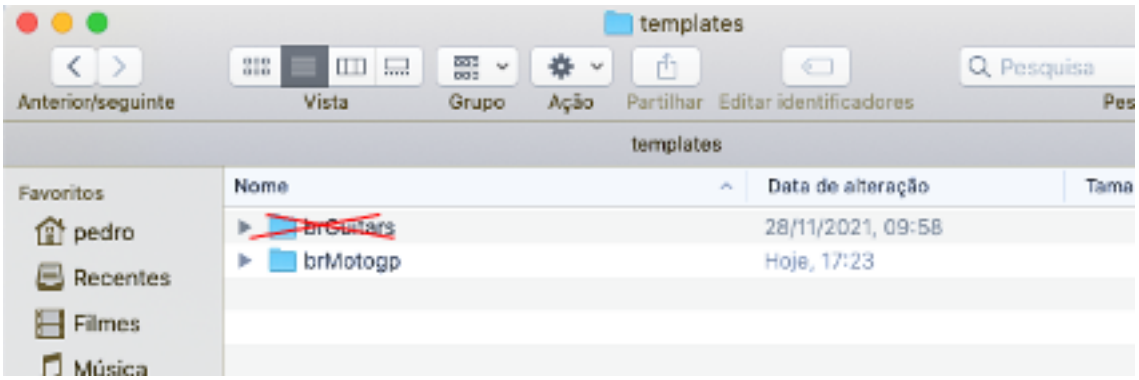
Go to pgadmin and look for the motogp tables. You should see things as shown next:



Step 8. Create an SQL script inserting new data into motogp tables. Look at motogp.sql file and modify it if necessary to insert into motogp tables instead of guitars. Please, instead of running the whole file at once, try to run insert commands separately so that you will detect any error and try to correct the errors. Error may appear if you have something different in the models.py file, for instance.

Anexo 9. Anexo insert SQL for motogp

Step 9. Create the folder templates inside brMotogp folder. Create a new folder inside templates for brMotogp. AT the end you will have the path: motogp/brMotogp/templates/brMotogp.



Step 10. Add imports and Modify function index in views.py to add also motogp data to the context variable:

```
views.py

from django.http import HttpResponse
from django.shortcuts import render
from django.template import loader

from .models import Luthier
from .models import Guitar
from .models import Artist
from .models import GuitarPic
from .models import Recording

from .models import Team
from .models import Moto
from .models import Driver
from .models import MotoPic
from .models import Drives

def index(request):
    template = loader.get_template('brMotogp/index.html')
    context = {
        'numLuthiers':Luthier.objects.all().count(),
        'numGuitars':Guitar.objects.all().count(),
        'numArtists':Artist.objects.all().count(),
        'numRecordings':Recording.objects.all().count(),
        'numGuitarPics':GuitarPic.objects.all().count(),

        'numTeams':Team.objects.all().count(),
        'numMotos':Moto.objects.all().count(),
        'numDrivers':Driver.objects.all().count(),
        'numDrives':Drives.objects.all().count(),
        'numMotoPics':MotoPic.objects.all().count(),

    }
    return HttpResponse(template.render(context, request))
```

Code available in appendix 12.

Step 11. Copy index.html from brGuitars to brMotogp and modify it (the copy) to show motogp data from the context variable:

(note that any misspelling in any of the steps may result in errors)

```html
index.html
<html>
<head>
<meta charset="UTF-8">
<title>MotoGP Database</title>
</head>
<body>
<h1>MotoGP Database</h1>


<h2><a href="/brMotogp/teams"> {{numTeams}} Teams</a></h2>
<h2><a href="/brMotogp/drivers"> {{numDrivers}} Drivers</a></h2>
<h2><a href="/brMotogp/motos"> {{numMotos}} Motos</a></h2>
<h2><a href="/brMotogp/drives"> {{numDrives}} Drives</a></h2>
<h2><a href="/brMotogp/motoPics"> {{numMotoPics}} Moto Pictures</a></h2>

```

Step 12. Create the urls for motogp. You will need to have two urls.py files: one in the project folder including the urls.py of the application and the other of the application:

urls.py of the motogp project:
you must have it in the motogp/motogp directory

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

urlpatterns = [
    path('brMotogp/', include('brMotogp.urls')),
    path('admin/', admin.site.urls),
]
```

urls.py of the application (brMotogp.urls**):**
you must place it in the motogp/brMotogp directory:

```
from django.urls import path

from . import views

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

    #Teams
    path('teams', views.teams, name='teams'),
    path('teams/<int:team_id>/', views.teamDetails, name='teamDetails'),

]
```

Step 13. Now test the new page: http://localhost:8000/brMotogp/

Step 14. Now add functions to the views and templates for showing the list of teams and the team details. You can copy-paste the luthier functions and templates and then change the details there:

views.py

```
    }
    return Http[illegible]se(templa[illegible] render(context, request))

def recordingDetai[illegible]_id):
    template = loade[illegible]ate('brGuitars/recordingDetails.html')
    try:
        rec = Rec[illegible]s.get(id = rec_id)
        contex[illegible]
    except Reco[illegible].DoesNo[illegible]
        raise Htt[illegible]("Recorc[illegible] does not exist")
    return HttpResponse(template.render(context, request))


def teams(request):
    template = loader.get_template('brMotogp/teams.html')
    items = Team.objects.order_by('name')[0:]
    context = {
        'teams':items
    }
    return HttpResponse(template.render(context, request))

def teamDetails(request, team_id):
    template = loader.get_template('brMotogp/teamDetails.html')
    try:
        myTeam = Team.objects.get(pk=team_id)
        myTeamMotos = Moto.objects.filter(team = team_id)
        context = {'team' : myTeam, 'motos' : myTeamMotos}
    except Team.DoesNotExist:
        raise Http404("Team does not exist")

    return HttpResponse(template.render(context, request))
```

teams.html

```
<a href="/brMotogp/">Home</a></p>
{% if teams %}
    <ul>
    {% for teamROW in teams %}
        <li><a href="/brMotogp/teams/{{teamROW.id}}/">{{ teamROW.name }}</a></li>
    {% endfor %}
    </ul>
{% else %}
    <p>No teams to show.</p>
{% endif %}
```

```html
teamDetails.html
<a href="/brMotogp/">Home</a></p>
<h1>{{team.name}} details</h1>
<picture>
 <source media="(min-width:320px)" srcset={{team.pic}}>
 <source media="(min-width:240px)" srcset={{team.pic}}>
 <img src={{team.pic}} alt={{team.name}}-{{team.pic}} style="max-width:600px;">
</picture>
<h3>Country: {{team.country}}</h3>
<h3>Motos:</h3>
{% if motos %}
    <ul>
    {% for motoROW in motos %}
        <li><a href="/brMotogp/motos/{{motoROW.id}}/">{{motoROW.model}}</a></li>
    {% endfor %}
    </ul>
{% else %}
    <p>No moto to show.</p>
{% endif %}
```

```python
urls.py
from django.urls import path

from . import views

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

    #luthiers
    path('luthiers', views.luthiers, name='luthiers'),
    path('luthiers/<int:luthier_id>/', views.luthierDetails, name='luthierDetails'),

    #artists
    path('artists', views.artists, name='artists'),
    path('artists/<int:artist_id>/', views.artistDetails, name='artistDetails'),

    #recordings
    path('recordings', views.recordings, name='recordings'),
    path('recordings/<int:rec_id>/', views.recordingDetails, name='recordingDetails'),

    #guitars
    path('guitars', views.guitars, name='guitars'),
    path('guitars/<int:guitar_id>/', views.guitarDetails, name='guitarDetails'),

    #guitarPics
    path('guitarPics', views.guitarPics, name='guitarPics'),

    #Teams
    path('teams', views.teams, name='teams'),
    path('teams/<int:team_id>/', views.teamDetails, name='teamDetails'),

]
```

Step 11. In the index page, touch the link to see the list of teams and the detail of one team. See if your changes worked.

5. (Lab 9- extra, homework) Do the rest for motogp

Step 15 (lab 13..). Now you can try to add the remaining views and templates to be able to see the drivers, motos, moto pictures...

6. Try to setup and use admin UI to add data manually

There is also a predefined user interface to allow you to add new data without coding. You can try to setup this UI and use it. Follow the following instructions (pasted from https://docs.djangoproject.com/en/1.8/intro/tutorial02/), but replace by your case:

Creating an admin user

First we'll need to create a user who can login to the admin site. Run the following command:

```
$ python manage.py createsuperuser
```

Enter your desired username and press enter.

```
Username: admin
```

You will then be prompted for your desired email address:

```
Email address: admin@example.com
```

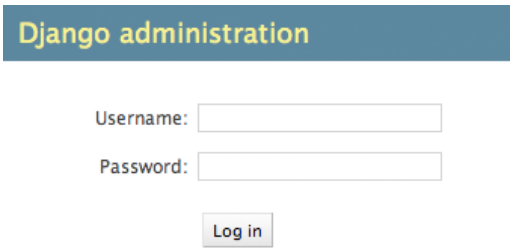
The final step is to enter your password. You will be asked to enter your password twice, the second time as a confirmation of the first.

```
Password: **********

Password (again): *********

Superuser created successfully.
```

Now, open a Web browser and go to "/admin/" on your local domain – e.g., http://127.0.0.1:8000/admin/. You should see the admin's login screen:



Make the app modifiable in the admin¶

But where's our app? It's not displayed on the admin index page.

Just one thing to do: we need to tell the admin that **`Team, Moto, Driver,etc`** objects have an admin interface. To do this, open the **`polls/admin.py`** file, and edit it to look like this:

`polls/admin.py`

```
from django.contrib import admin
```

```
from .models import Team

from .models import Moto

…

admin.site.register(Team)

admin.site.register(Moto)

…
```

Explore the free admin functionality¶

Now that we've registered **`your classes`**, Django knows that it should be displayed on the admin index page. Click "Teams". Now you're at the "change list" page for teams. This page displays all the teams in the database and lets you choose one to change it.

7. (Project) Now you can do similarly for your course project

Given your course project (a database application choosen by you), defined by your group, do the same as you did with motogp for your project. Define the database model (do it only with a few tables, do not try to do a huge project), and implement the equivalent to the parts that we implemented for guitars and motogp (you do not need to implement a huge application, if it gets big you can restrict what you implement. Also, you do not need to do anything besides what you saw in guitars or motogp, unless you want to explore more by yourself = extra 2 points extracted from exam).

8. Anexo SQL creation scripts motogp

```
CREATE TABLE team (
	id	SERIAL,
	name	VARCHAR(512) NOT NULL,
	country VARCHAR(512),
	picture VARCHAR(512),
	PRIMARY KEY(id)
);

CREATE TABLE moto (
	id	BOOL,
	model	VARCHAR(512) NOT NULL,
	brand	VARCHAR(512),
	size	INTEGER,
	cc	INTEGER,
	hp	INTEGER,
	team_id INTEGER NOT NULL,
	PRIMARY KEY(id)
);

CREATE TABLE driver (
	id	SERIAL,
	name	VARCHAR(512) NOT NULL,
	country VARCHAR(512),
	birth	DATE,
	death	DATE,
	picture VARCHAR(512),
	PRIMARY KEY(id)
);

CREATE TABLE drives (
	id	BOOL,
	datestart DATE,
	dateend	DATE,
	moto_id	BOOL NOT NULL,
	driver_id INTEGER NOT NULL,
	PRIMARY KEY(id)
);

CREATE TABLE motopics (
	id	BOOL,
	img	VARCHAR(512),
	moto_id BOOL NOT NULL,
	PRIMARY KEY(img)
);

ALTER TABLE moto ADD CONSTRAINT moto_fk1 FOREIGN KEY (team_id) REFERENCES team(id);
ALTER TABLE drives ADD CONSTRAINT drives_fk1 FOREIGN KEY (moto_id) REFERENCES moto(id);
ALTER TABLE drives ADD CONSTRAINT drives_fk2 FOREIGN KEY (driver_id) REFERENCES driver(id);
ALTER TABLE motopics ADD CONSTRAINT motopics_fk1 FOREIGN KEY (moto_id) REFERENCES moto(id);
```

9. Anexo insert SQL for motogp

```
insert into "brMotogp_team" (name, country, pic)
values
('Ducati', 'Brasil',
 'https://upload.wikimedia.org/wikipedia/commons/3/3a/Ducati_MotoGP_04_%2810760413416%29.jpg'),
('Aprillia', 'Brasil',

'https://upload.wikimedia.org/wikipedia/commons/0/06/Aleix_Espargar%C3%B3_leads_the_pack_2021_Sachsenrin
g_%28cropped%29.jpg'),
('KTM', 'Canada',
```

```
 'https://upload.wikimedia.org/wikipedia/commons/thumb/b/bf/KTM_MotoGP_Bike_RC16.jpg/1024px-KTM_MotoGP_Bike_RC16.jpg');


insert into "brMotogp_driver" (name, country, birth, death, pic)
values
('Ulisses Rocha', 'Brasil', 1960, null,
'https://img.discogs.com/z2q6ST1LdvB9Urv_UZG4mBlTxWM=/600x399/smart/filters:strip_icc():format(jpeg):mode_rgb():quality(90)/discogs-images/A-399042-1441106248-4288.jpeg.jpg'),
('Daniel Murray', 'Brasil', 1981, null, 'http://www.kleberpatricio.com.br/wp-content/uploads/2019/04/dm-1024x683.jpg'),
('Yamandu Costa', 'Brasil', 1980, null, 'https://www.rbsdirect.com.br/imagesrc/25666109.jpg'),
('Paulo Bellinati', 'Brasil', 1950, null, 'http://www.guitarplayer.com.br/materias/1225.jpg'),
('Sebastião Tapajós', 'Brasil', 1943, 2021,
'https://s2.glbimg.com/zjsvHa83gl3o_JvUA_EaElWZuvg=/0x0:660x494/984x0/smart/filters:strip_icc()/i.s3.glbimg.com/v1/AUTH_59edd422c0c84a879bd37670ae4f538a/internal_photos/bs/2018/O/V/AzLMI0ROSMDgZg1pyd2Q/sebastiao-tapajos.jpg');

insert into "brMotogp_moto" (team_id, model, brand,size,cc,hp)
values
((select id from "brMotogp_team" where name = 'Ducati'), 'D1', 'Ducati',2,1000,150),
((select id from "brMotogp_team" where name = 'Ducati'), 'D2', 'Ducati',2,1000,150),
((select id from "brMotogp_team" where name = 'Aprillia'), 'A1', 'Aprillia',2,1100,155),
((select id from "brMotogp_team" where name = 'KTM'), 'K1', 'KTM',2,1200,160);

insert into "brMotogp_motopic" (moto_id, img)
values
((select id from "brMotogp_moto" m where m.model='D1'),
'https://upload.wikimedia.org/wikipedia/commons/3/3a/Ducati_MotoGP_04_%2810760413416%29.jpg'),
((select id from "brMotogp_moto" m where m.model='D2'),
'https://upload.wikimedia.org/wikipedia/commons/3/3a/Ducati_MotoGP_04_%2810760413416%29.jpg'),
((select id from "brMotogp_moto" m where m.model='A1'),
'https://upload.wikimedia.org/wikipedia/commons/0/06/Aleix_Espargar%C3%B3_leads_the_pack_2021_Sachsenring_%28cropped%29.jpg'),
((select id from "brMotogp_moto" m where m.model='K1'),
'https://upload.wikimedia.org/wikipedia/commons/thumb/b/bf/KTM_MotoGP_Bike_RC16.jpg/1024px-KTM_MotoGP_Bike_RC16.jpg');

insert into "brMotogp_drives" (driver_id, moto_id, dateSTART, dateEND)
values
(
	(select id from "brMotogp_driver" where name='Ulisses Rocha'),
	(select id from "brMotogp_moto" m where m.model='D1'),
	2020,2021
);
```

10. Anexo possible fixes for motogp runserver

```
ImportError: Module 'brMotogp.apps' does not contain a 'BrMotogpConfig' class. Choices are: 'BrmotogpConfig'.
```

Go to apps.py:

```python
apps.py
from django.apps import AppConfig


class BrMotogpConfig(AppConfig):
    default_auto_field = 'django.db.models.BigAutoField'
    name = 'brMotogp'

```

Add to urls.py in motogp (not the one in brMotogp):

```python
urls.py
"""motogp URL Configuration

The `urlpatterns` list routes URLs to views. For more information please see:
    https://docs.djangoproject.com/en/3.2/topics/http/urls/
Examples:
Function views
    1. Add an import:  from my_app import views
    2. Add a URL to urlpatterns:  path('', views.home, name='home')
Class-based views
    1. Add an import:  from other_app.views import Home
    2. Add a URL to urlpatterns:  path('', Home.as_view(), name='home')
Including another URLconf
    1. Import the include() function: from django.urls import include, path
    2. Add a URL to urlpatterns:  path('blog/', include('blog.urls'))
"""
from django.contrib import admin
from django.urls import include,path

urlpatterns = [
    path('brMotogp/', include('brMotogp.urls')),
    path('admin/', admin.site.urls),
]

```

11. Anexo: parts of possible code for motogp

Models.py:

```
from django.db import models

class Team(models.Model):
    name = models.CharField(max_length=20, null=False, blank=False)
    country = models.CharField(max_length=200, null=True, blank=True)
    pic = models.URLField(max_length=500, null=True, blank=True)
    def __str__(self):
        return self.name

class Moto(models.Model):
    team = models.ForeignKey(Team, on_delete=models.CASCADE)
    model = models.CharField(max_length = 200, null=False, blank=False)
    brand = models.CharField(max_length = 200, null=False, blank=False)
    size = models.IntegerField(null=True, blank=True)
    cc = models.IntegerField(null=True, blank=True)
    hp = models.IntegerField(null=True, blank=True)
    def __str__(self):
        return self.team.name+"-"+self.model

class MotoPic(models.Model):
```

```
    img = models.URLField(max_length=500, null=False, blank=False)
    moto = models.ForeignKey(Moto, on_delete=models.CASCADE)
    def __str__(self):
        return self.img

class Driver(models.Model):
    name = models.CharField(max_length=200, null=False, blank=False)
    country = models.CharField(max_length=200, null=True, blank=True)
    birth = models.IntegerField(null=True, blank=True)
    death = models.IntegerField(null=True, blank=True)
    pic = models.URLField(max_length=500, null=True, blank=True)
    def __str__(self):
        return self.name
    def isAlive(self):
        return death==False

class Drives(models.Model):
    dateSTART = models.IntegerField(null=True, blank=True)
    dateEND = models.IntegerField(null=True, blank=True)
    driver = models.ForeignKey(Driver, on_delete=models.CASCADE)
    moto = models.ForeignKey(Moto, on_delete=models.CASCADE, null=True, blank=True)
    def __str__(self):
        return self.name
```

views.py:

```
from django.http import HttpResponse
from django.shortcuts import render
from django.template import loader

from .models import Team
from .models import Moto

def index(request):
    template = loader.get_template('brMotogp/index.html')
    context = {
    }
    return HttpResponse(template.render(context, request))

def teams(request):
    template = loader.get_template('brMotogp/teams.html')
    teams = Team.objects.order_by('name')[0:]
    print(teams)
    context = { 'teams':teams }
    return HttpResponse(template.render(context, request))

def teamDetails(request, team_id):
    template = loader.get_template('brMotogp/teamDetails.html')
    try:
        myTeam = Team.objects.get(pk=team_id)
        myTeamMotos = Moto.objects.filter(team = team_id)
        context = {'team' : myTeam, 'motos' : myTeamMotos}
    except Team.DoesNotExist:
        raise Http404("Team does not exist")

    return HttpResponse(template.render(context, request))
```