



# A CAMADA MODEL DO DJANGO (PYTHON)

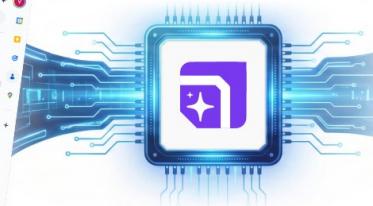
Nesse ebook, vamos tratar da camada Model do Django. Vamos abordar detalhes dessa camada, mapeamento objeto-relacional, a API de acesso a dados provida pelo Django e modelagem de dados.

# Crie Ebooks técnicos incríveis em minutos com IA

Conheça a 1ª IA Especializada na criação de Ebooks **com código!**



Chega de formatar código no Google Docs



Deixe que nossa IA faça o trabalho pesado

 Syntax Highlight

 Adicione Banners Promocionais

 Edite em Markdown em Tempo Real

 Infográficos feitos por IA

**TESTE AGORA** 

Salve salve Pythonista! 🙋

Continuando a nossa série de *posts* sobre Django, nesse artigo vamos tratar sobre a camada *Model* da sua arquitetura!

Se você ainda não leu o *post* introdutório, você pode estar perdendo algumas informações iniciais importantes.

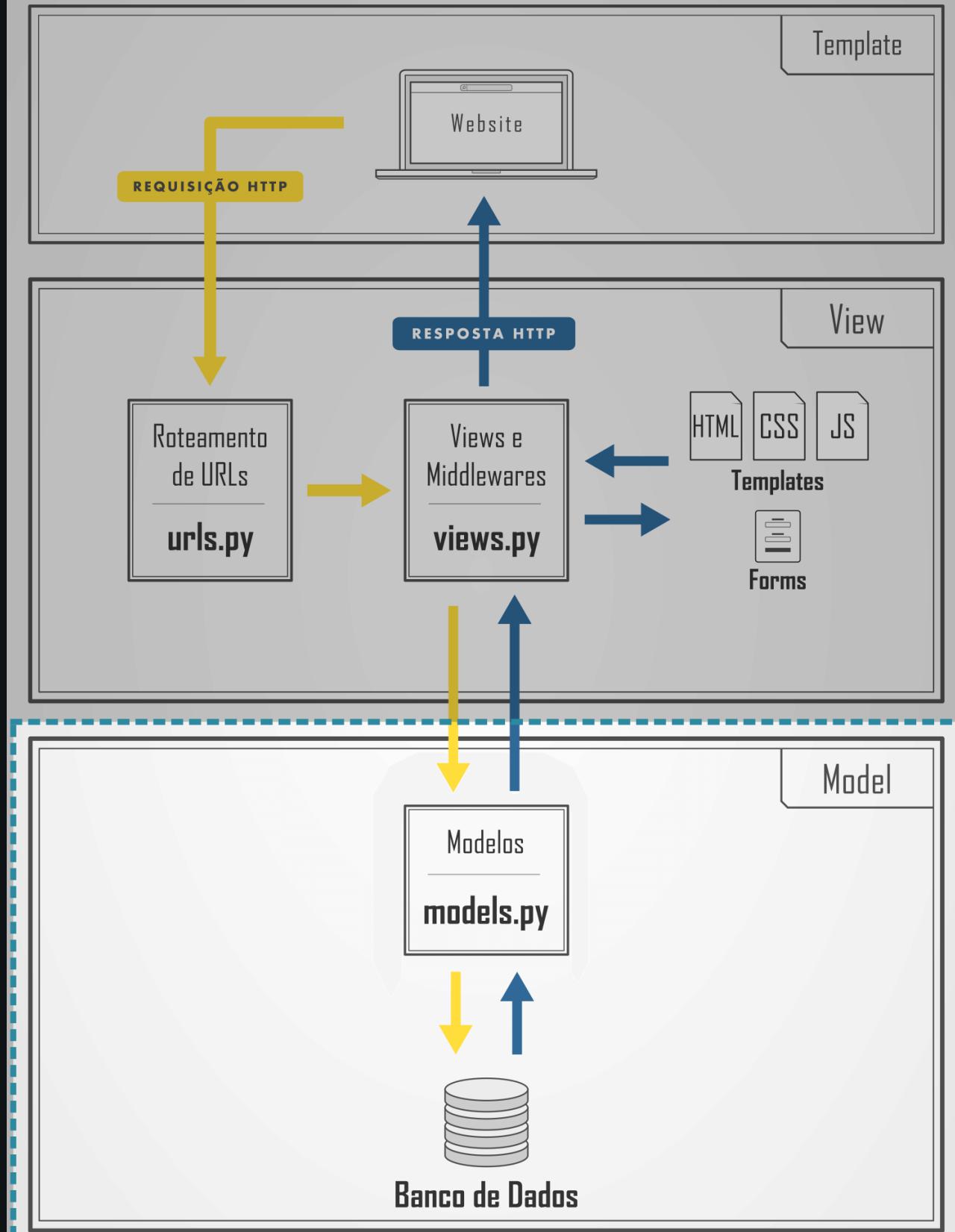
Se esse for o seu caso, então já [clica aqui](#) e **fique por dentro do assunto!**

Dito isso, vamos nessa que o tempo não para!

## Onde estamos...

Primeiramente, vamos nos situar...

# ARQUITETURA DO django



No nosso [primeiro post](#), tratamos de conceitos introdutórios do *framework*, uma visão geral da sua arquitetura, sua instalação e a criação do famoso **Hello World Django-based**.

Agora, vamos mergulhar um pouco mais e vamos conhecer a camada *Model* da arquitetura **MTV** do Django (*Model Template View*).

Nela, vamos descrever, em forma de classes, as **entidade**s do nosso sistema, para que o resto (*Template* e *View*) façam sentido.

Sem mais delongas, apresento-lhes: **A camada Model!**

## **Model Layer (Camada de Modelos)**

Vamos começar pelo básico: pela definição de **modelo**!

Um **modelo** é a descrição do dado que será gerenciado pela sua aplicação.

Ele contém os campos e comportamentos desses dados. No fim, cada modelo vai equivaler à uma tabela no banco de dados.

No Django, um modelo tem basicamente duas características:

- É uma classe que herda de `django.db.models.Model`
- Cada atributo representa um campo da tabela

Com isso, Django gera automaticamente uma API de acesso à dados, isto é: facilita sua vida quando for gerenciar (adicionar, excluir e atualizar) seus dados.

Para entendermos melhor, vejamos como desenvolver um modelo em seu sistema!

Vamos supor que sua empresa está desenvolvendo um sistema de gerenciamento de funcionários e lhe foi dada a tarefa de modelar e desenvolver o acesso aos dados da entidade `Funcionário`.

Pensando calmamente em sua estação de trabalho enquanto seu chefe lhe cobra diversas metas, você pensa nos seguintes atributos para tal classe:

- Nome
- Sobrenome
- CPF
- Tempo de serviço
- Remuneração

Agora, é necessário passar isso para código.

Utilizando seu vasto conhecimento em Python e Django adquirido aqui na Python Academy...

Falando nisso...

Continuando...

Os modelos, por convenção, são descritos no arquivo `models.py`.

Vamos criar, então, esse arquivo na pasta `helloworld/models.py` (\_Você ainda tem o nosso projeto `helloworld` do [post anterior](#)? Se não, então [clique aqui e baixe o arquivo zipado `helloworld.zip`][helloworld-project] para seguirmos do mesmo ponto!).

No arquivo `models.py` você então passa seu rascunho inicial para a classe Python `Funcionário` seguindo as duas características que apresentamos (herdar de `Model` e relacionar os campos com os tipos de campos da tabela) da seguinte forma:

```
from django.db import models

class Funcionario(models.Model):

    nome = models.CharField(
        max_length=255,
        null=False,
        blank=False
    )

    sobrenome = models.CharField(
        max_length=255,
        null=False,
        blank=False
    )

    cpf = models.CharField(
        max_length=14,
        null=False,
        blank=False
    )

    tempo_de_servico = models.IntegerField(
        default=0,
        null=False,
        blank=False
    )

    remuneracao = models.DecimalField(
        max_digits=8,
        decimal_places=2,
        null=False,
        blank=False
    )

    objetos = models.Manager()
```

Explicando brevemente esse modelo:

- Cada campo tem um tipo

- O tipo `CharField` representa uma string
- O tipo `PositiveIntegerField` representa um número inteiro positivo
- O tipo `DecimalField` representa um número decimal com precisão fixa
- Cada tipo tem um conjunto de propriedades, como `max_length` para delimitar o comprimento máximo da string, `decimal_places` para configurar o número de casas decimais, etc (a documentação de cada campo e propriedades pode ser [acessado aqui][django-model-reference]).
- O campo `objetos = models.Manager()` é utilizado para fazer operações de busca e será explicado ali embaixo!
- **Observação:** não precisamos configurar o campo `id` - ele é herdado do objeto `models.Model` (do qual nosso modelo herdou)!

Agora que criamos nosso modelo, é necessário executar a criação das tabelas no banco de dados (*você não achou que seria automático, né?!*).

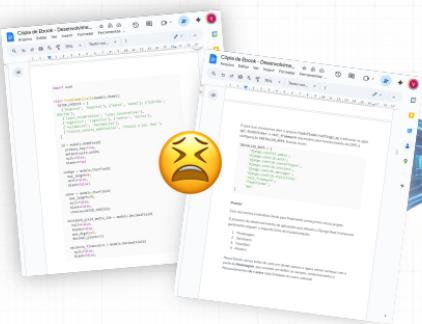
Para isso, o Django possui dois comandos que ajudam **muito**: o `makemigrations` e o `migrate`.



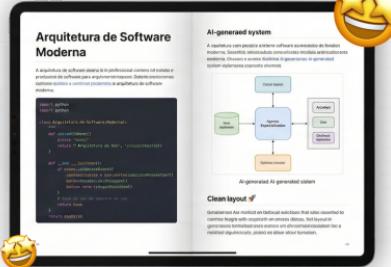
*Estou desenvolvendo o **DevBook**, uma plataforma que usa IA para criar ebooks técnicos — com código formatado e exportação em PDF. Depois de ler, dá uma passada no site!*

## Crie Ebooks técnicos incríveis em minutos com IA

Conheça a 1ª IA Especializada na criação de Ebooks **com código!**



Chega de formatar código no Google Docs



Deixe que nossa IA faça o trabalho pesado

Syntax Highlight

Adicione Banners Promocionais

Edite em Markdown em Tempo Real

Infográficos feitos por IA

**TESTE AGORA! PRIMEIRO CAPÍTULO 100% GRÁTIS**

## O comando `makemigrations`

O comando `makemigrations` analisa se foram feitas mudanças nos modelos e, em caso positivo, cria novas migrações ( `Migrations` ) para alterar a estrutura do seu banco de dados, refletindo as alterações feitas.

Vamos entender o que eu acabei de dizer: toda vez que você faz uma **alteração** em seu modelo, é necessário que ela seja **aplicada** a estrutura presente no banco de dados.

A esse processo é dado o nome de **Migração!**

De acordo com a documentação do Django:

*Migração é a forma do Django de **propagar as alterações** feitas em seu modelo (adição de um novo campo, deleção de um modelo, etc...) ao seu esquema do banco de dados. Elas foram desenvolvidos para serem (a maioria das vezes) **automáticas**, mas cabe a você saber a hora de fazê-las, de executá-las e de resolver os problemas comuns que você possa vir a ser submetidos.*

Portanto, toda vez que você alterar o seu modelo, não se esqueça: execute `python manage.py makemigrations`!

Ao executar esse comando no nosso projeto, devemos ter a seguinte saída:

```
$ python manage.py makemigrations

Migrations for 'helloworld':
  helloworld\migrations\0001_initial.py
    - Create model Funcionario
```

**Observação:** Talvez seja necessário executar o comando referenciando o app onde estão definidos os modelos: `python manage.py makemigrations helloworld`. Daí pra frente, apenas `python manage.py makemigrations` vai bastar!

Agora, podemos ver que foi criada uma pasta chamada `migrations` dentro de `helloworld`.

Dentro dela, você pode ver um arquivo chamado `0001_initial.py`: ele contém a `Migration` que cria o `model Funcionario` no banco de dados (*preste atenção na saída do comando `makemigrations`: `Create model Funcionario` 😊*)

## O comando `migrate`

Quando executamos o `makemigrations`, o Django cria o banco de dados e as *migrations*, mas não as executa, isto é: não aplica as alterações no banco de dados.

Para que o Django as aplique, são necessárias três coisas, basicamente:

- **1.** Que a configuração da interface com o banco de dados esteja descrita no `settings.py`
- **2.** Que os modelos e *migrations* estejam definidos para esse projeto
- **3.** Execução do comando `migrate`

Se você criou o projeto com `django-admin.py createproject helloworld`, a configuração padrão foi aplicada. Procure pela configuração `DATABASES` no `settings.py`. Ela deve ser a seguinte:

```
DATA BASES = {  
    'default': {  
        'ENGINE': 'django.db.backends.sqlite3',  
        'NAME': os.path.join(BASE_DIR, 'db.sqlite3'),  
    }  
}
```

Por padrão, o Django utiliza um banco de dados leve e completo chamado [SQLite](#). Já já vamos falar mais sobre ele.

Sobre os modelos e *migrations*, eles já foram feitos com a definição do **Funcionário** no arquivo `models.py` e com a execução do comando `makemigrations`.

Agora só falta **executar o comando `migrate`**, propriamente dito!

Para isso, vamos para a raíz do projeto e executamos: `python manage.py migrate`. A saída deve ser:

```
$ python manage.py migrate

Operations to perform:
  Apply all migrations: admin, auth, contenttypes, helloworld, 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 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 helloworld.0001_initial... OK
  Applying sessions.0001_initial... OK
```

Calma lá... Havíamos definido apenas **uma** Migration e foram aplicadas 15!!! Por quê???

Se lembra que a configuração `INSTALLED_APPS` continha vários `apps` (e *não* apenas os nossos `helloworld` e `website`)?

Pois então, cada `app` desses contém seus próprios modelos e *migrations*. *Sacou*!?



Com a execução do `migrate`, o Django irá criar diversas tabelas no banco. Uma delas é a nossa, similar à:

```
CREATE TABLE helloworld_funcionario (
    "id" serial NOT NULL PRIMARY KEY,
    "nome" varchar(255) NOT NULL,
    "sobrenome" varchar(255) NOT NULL,
    ...
);
```

*Isso está muito abstrato! Como eu posso ver o banco, as tabelas e os dados na prática?*

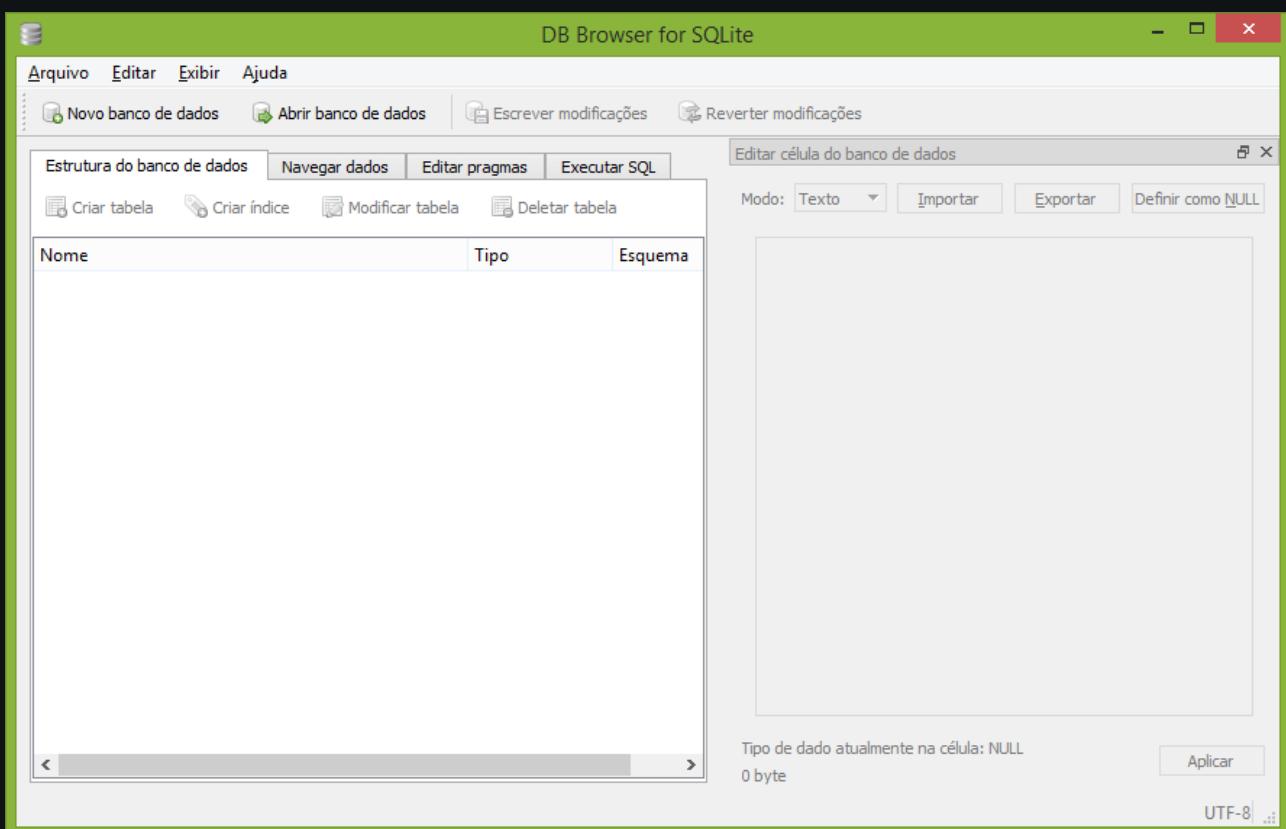
## DB Browser for SQLite

Apresento-lhes uma ferramenta **MUITO** prática: o *DB Browser for SQLite!*

Com ela, podemos ver a estrutura do banco de dados, alterar dados em tempo real, fazer *queries*, e muito mais!

[Clique aqui para fazer download do DB Browser for SQLite](#) e instalá-lo. Ao terminar a instalação, abra o *DB Browser for SQLite*.

Temos:



Nele, podemos clicar em “**Abrir banco de dados**” e procurar pelo banco de dados do nosso projeto `db.sqlite3`.

Ao importá-lo, teremos uma visão geral, mostrando Tabelas, Índices, Views e *Triggers*.

Para ver os dados de cada tabela, vá para a aba “**Navegar dados**”, escolha nossa tabela `helloworld_funcionario` e...

**Voilá!** O que temos? **NADA** 😞

*Calma jovem... Ainda não adicionamos nada 😞*

*Já já vamos criar as **Views** e **Templates** para popular esse BD! 😊*

## API de Acesso a Dados

Com nossa classe `Funcionário` modelada, vamos agora ver a API de acesso à dados provida pelo Django para facilitar **muito** nossa vida!

Vamos testar a adição de um novo funcionário utilizando o *shell* do Django. Para isso, digite o comando:

```
python manage.py shell
```

O *shell* do Django é muito útil para **testar trechos de código** sem ter que executar o servidor inteiro!

Para **adicionar** um novo funcionário, basta criar uma instância do seu modelo e chamar o método `save()` (*lembra que nosso modelo herdou de `Models` ?*).

Podemos fazer isso com o código abaixo (no *shell* do Django):

```
from helloworld.models import Funcionario

funcionario = Funcionario(
    nome='Marcos',
    sobrenome='da Silva',
    cpf='015.458.895-50',
    tempo_de_servico=5,
    remuneracao=10500.00
)

funcionario.save()
```

E.... **Pronto!** O Funcionário `Marcos da Silva` será salvo no seu banco!

**NADA** de código SQL e *queries* enormes!!!

Tudo simples! Tudo limpo! **Tudo Python!**

A API de **busca de dados** é ainda mais completa! Nela, você constrói sua *query* à nível de objeto!

***Mas como assim?!***

Por exemplo, para buscar todos os Funcionários, abra o shell do Django e digite:

```
funcionarios = Funcionario.objects.all()
```

Se lembra do tal `Manager` que falamos lá em cima? Então, um `Manager` é a interface na qual as operações de busca são definidas para o seu modelo.

Ou seja, através do campo `objetos` podemos fazer *queries* incríveis sem uma linha de SQL!

Exemplo de um *query* um pouco mais complexa:

*Busque todos os funcionários que tenham **mais de 3 anos de serviço**, ganhem **menos de R\$ 5.000,00 de remuneração** e que **não tenham Marcos no nome***

Podemos atingir esse objetivo com:

```
funcionarios = Funcionario.objects
    .exclude(name="Marcos")
    .filter(tempo_de_servico__gt=3)
    .filter(remuneracao__lt=5000.00)
    .all()
```

O método `exclude()` retira linhas da pesquisa e `filter()` filtra a busca!

No exemplo, para filtrar por **maior que** concatenamos a string `__gt` (**gt** = *greater than*) ao filtro e `__lt` (**lt** = *less than*) para resultados **menores que**.

O método `.all()` ao final da *query* serve para retornar **todas** as linhas do banco que cumpram os filtros da nossa busca (também temos o `first()` que retorna apenas o primeiro registro, o `last()`, que retorna o último, entre outros).

Agora, vamos ver como é **simples** excluir um `Funcionário`:

```
# Primeiro, encontramos o Funcionário que desejamos deletar (id=1 por
# exemplo)
funcionario = Funcionario.objects.filter(id=1).first()

# Agora, o deletamos!
funcionario.delete()
```

**Legal, né?!**

A atualização também é extremamente simples, bastando buscar a instância desejada, alterar o campo e salvá-lo novamente!

Por exemplo: o funcionário de **id = 13** se casou e alterou seu nome de Marcos da Silva para Marcos da Silva Albuquerque.

Podemos fazer essa alteração da seguinte forma:

```
# Primeiro, buscamos o funcionario desejado
funcionario = Funcionario.objects.filter(id=13).first()

# Alteramos seu sobrenome
funcionario.sobrenome = funcionario.sobrenome + " Albuquerque"

# Salvamos as alterações
funcionario.save()
```

([Clique aqui para baixar o código][django-code] desenvolvido até aqui.)

## Conclusão

Com isso galera:

### ✓ Modelo

Criamos nosso primeiro modelo, nosso banco de dados, vimos como visualizar os dados com o *DB Browser for SQLite* e como a API de acesso a dados do Django é simples e poderosa!

Espero ter aguçado a vontade por aprender mais do Django! O próximo post trata da Camada View ([Já está disponível! Acesse-o AQUI e AGORA](#))! É nela que construímos a lógica de negócio do nosso projeto.

Quer levar esse conteúdo para onde for com nosso **ebook GRÁTIS**?

Então aproveita essa chance 

E não se esqueça: qualquer dúvida, *crash*, *bug* ou sugestão, utilize o box de comentário aqui embaixo que eu responderei ASAP!

É isso pessoal! Aguardo vocês no **próximo artigo**!

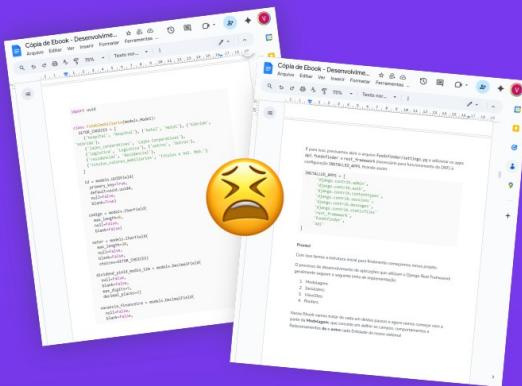
Abraço e bom desenvolvimento!

[django-model-reference]//docs.djangoproject.com/en/4.2/ref/models/fields/



# Crie Ebooks técnicos em minutos com IA

Conheça a 1ª IA Especializada na criação de Ebooks **com código!**



Chega de formatar código no Google Docs



**Arquitetura de Software Moderna**

A arquitetura de software alvo é profissional contendo o e-mail e produções de software para arquiteturas modernas. Oferece recursos como interface gráfica com interface de usuário.

```
import python
import python

class Arquitetura_de_Software_Moderna:
    ...
    def share(self):
        pass
    ...
    return "Arquitetura de NeXt", "arquitetura_moderna"
}

def __init__(self):
    if user_authenticated():
        self.user_authenticated = user_authenticated()
        self.user_email = self.user_authenticated['email']
        self.user_name = self.user_authenticated['name']
    ...
    # Envie AI para gerar um código
    return type
}
resource_available
```

**AI-generated system**

A arquitetura com propósito é a arquitetura moderna. Seus componentes incluem interface de usuário, banco de dados e outros sistemas externos. Chama-se de sistema gerado por IA.

**Clean layout**

O layout é limpo e organizado, facilitando a leitura e compreensão do código gerado.



</> Syntax Highlight

Infográficos feitos por IA

Adicione Banners Promocionais

Deixe que nossa IA faça o trabalho pesado

Edite em Markdown em Tempo Real

**TESTE AGORA**



PRIMEIRO CAPÍTULO 100% GRÁTIS