



# **Curso completo de Python!**





**Instrutor: Vitor Mazuco**

**<http://facebook.com/vitormazuco>**

**Email: [vitor.mazuco@gmail.com](mailto:vitor.mazuco@gmail.com)**

**WebSite:**

**<http://vmzsolutions.com.br>**



# Flask

web development,  
one drop at a time

mongo  
engine



## MongoEngine com Flask

O MongoEngine é uma ORM para MongoDB. Já que criamos uma API simples e enviamos e recebemos dados pelo formato JSON e o MongoDB guarda esses dados no mesmo formato, podemos aproveitar e juntar essas duas coisas.



## MongoEngine com Flask

Como usamos o Flask, podemos trabalhar com ele usando o MongoEngine, embora ele pode trabalhar só, sem o Flask!.

Para instalar, vamos usar o pip install.

```
# pip install flask-mongoengine
```



## MongoEngine com Flask

Vamos criar uma pasta para a sua aplicação chamado de Model.py. Ao trabalhar nesse projeto, todos os arquivos são mini aplicações, logo é preciso importar sempre o Flask e sua extensão. Ex:

```
from flask import Flask  
from flask.ext.mongoengine import MongoEngine
```



## MongoEngine com Flask

Agora, crie sua aplicação em Flask.

```
app = Flask(__name__)  
app.config["MONGODB_SETTINGS"] = {"db":"dexter-api"}  
db = MongoEngine(app)
```



Já podemos criar os nossos documentos pelas Classes, igual como foi feito no SQLAlchemy, mas os seus tipos não são iguais. Ex:

```
class Usuarios(db.Document):
```

Essa classe Usuários, é um subdocumento de algum outro tipo de documento que podemos criar.





A nossa classe Usuários ficaria da seguinte maneira:

```
class Usuarios(db.Document):  
    nome = db.StringField()  
    email = db.StringField(unique=True)  
    data_cadastro =  
    db.DateTimeField(defaults=datetime.datetime.now())
```



## MongoEngine com Flask

No campo do e-mail, foi colocado um parâmetro `unique=True`. Ele serve para falar ao MongoDB que esse campo jamais poderá ser repetido, já que os dois usuários não podem usar o mesmo e-mail.



## MongoEngine com Flask

Já no `data_cadastro`, foi colocado como parâmetro o:  
`defaults=datetime.datetime.now()`. Isso serve para caso não seja  
passado a data do cadastro de forma correta, ele  
automaticamente funcionará a data e hora local. Precisamos  
importar o módulo `datetime`!



Agora a classe Grupos tem que ser criada da seguinte forma:

```
class Grupos(db.Document):  
    nome = db.StringField(unique=True)  
    integrantes = db.ListField()
```



## MongoEngine com Flask

A classe de grupos ficou um pouco diferente, usamos o `db.StringField` que ele irá receber um valor do tipo `text` e seu parâmetro é o `unique=True`, pois não podemos ter grupos de mesmo nome.



Em Integrantes foram criados do tipo `ListField`, que equivale a uma lista em Python. O seu parâmetro foi `db.ListField()` que indica que será recebido um subdocumento e como parâmetro ele recebe uma classe que contém os valores do subdocumento.



Para podermos testar o nosso Model.py, precisamos do if `__name__ == '__main__':` e dentro dele uma instância da classe Grupos.

```
u = Usuarios()  
u.nome = "Vitor"  
u.email = "vitor@vmzsolutions.com.br"
```



## MongoEngine com Flask

Logo abaixo temos uma instância de Grupos:

```
g = Grupos()  
g.nome = "Comercial"  
g.integrantes.append(u)
```

E depois ele grava no banco de dados:

```
g.save()
```

Observe que o MongoEngine é bem semelhante com o do SQLAlchemy





## MongoEngine com Flask

Agora que já vimos como o MongoEngine funciona, vamos unir ao Flask para fazer persistência dos usuários e seus grupos em nossa API. Agora vamos voltar ao nosso arquivo.



## MongoEngine com Flask

Em nossa programação, vamos ter 2 blueprints que vai ser uma para gerenciar as rotas e suas operações dos usuários e uma dos grupos. Vamos ver o arquivo dos Usuários.



## MongoEngine com Flask

Vamos primeiro fazer uma listagem de todos os nossos usuários, para isso, precisamos importar o model de

Usuários:

```
from Model import Usuarios  
import json
```

Vamos usar o json para fazer a conversão dele.



Dentro da função `def index_usuarios()`: vamos ter esses códigos:

```
lista_usuarios = json.loads(Usuarios.objects.to_json())  
return jsonify({"usuarios":lista_usuarios})
```

Pronto! Agora já está feito a nossa listagem de cadastramento no `mondodb`!



## MongoEngine com Flask

Logo após, vamos salvar o objeto no banco:

```
u.save()
```



Há também a necessidade de retornar uma mensagem ao usuário para mostrar se o usuário foi realmente cadastrado , logo a nossa return ficará assim:

```
return jsonify({"message":"Usuário cadastrado com sucesso!"})
```



## MongoEngine com Flask

Para usar uma seção de busca, precisamos usar a função `def select_usuarios(id):`

```
u = json.loads(Usuarios.objects(id=id).to_json())  
data = {"usuario":u}  
return jsonify(data)
```

Ao trabalhar com o Flask, o MongoEngine fica muito mais fácil e simples de programar.



## MongoEngine com Flask

E para deletar registros, vamos usar a função def

`delete_usuarios(id):`

```
u = Usuarios.objects(id=id)
```

```
u.delete()
```

```
data = {"message": "Deletando usuario cujo o ID e igual a  
%s"%id}
```

```
return jsonify(data)
```





E para terminar, temos a função para atualizar o registro na função `def update_usuarios(id):` e vamos usar o `setattr()` de novo:

```
dados = request.get_json()
u = Usuarios.objects(id=id).first()
for key in dados.keys():
    setattr(u, key, dados[key])
```



## MongoEngine com Flask

Depois de colocar os atributos da classe, temos que usar o método `save()` para gravar no nosso BD.

```
u.save()
```

```
data = {"message": "Atualizando o usuario cujo o ID e igual  
a %s"%id}
```



## MongoEngine com Flask

Agora já temos todas as funções que a nossa API tem para trabalhar com Banco de Dados. Também já programamos os blocos de try/except para assim podermos ver os possíveis erros em nossa programação. Agora é só replicar para a entidade de grupos o mesmo que foi feito com o os usuários.



## MongoEngine com Flask

Agora rode os arquivos `run.py` para subir o Flask e depois o arquivo `model.py` para criar os registros em nosso banco de dados no `mongodb`, e depois faça as operações CRUD em seu RestClient ou veja no próprio MongoDB.