

DJANGO

CRIANDO
API DE
FORMA
RÁPIDA
E
EFICIENTE

ALISON AGUIAR



django

django
REST
framework

Introdução

Olá eu sou Alison Aguiar desenvolvedor Full Stack apaixonado por programação, ao longo dos últimos 5 anos de experiência em Desenvolvimento Web

Percebi que a maioria dos desenvolvedores tinham dificuldade de criar uma API's de forma rápida com segurança e validação de campos no backend .

Sempre tinha que instalar varias e varias bibliotecas e ainda escrever validações uma a uma. Isso é um saco, eu sei, então um dia navegando na internet conheci o Django.

Depois de algum tempo testando e implementando decidi me aprofundar. Desenvolvi alguns projetos e conheci uma biblioteca chamada Rest Framework. Criei e implementei um projeto.

Foi paixão a primeira vista, a estabilidade e facilidade de escalabilidade. Decidi me aprofundar nesses benditos e até hoje não encontrei algo tão produtivo pra web quanto essas ferramentas.

Django essa ótima plataforma pra levantar aplicações de maneira rápida usado no núcleo do **Instagram** com uma base de mais de 1 bilhões de usuários ativos mensais, então decidi escrever este e-book pra você ter um gostinho da delícia que é essa tecnologia.

Espero que goste.
E mão na massa.

Ferramentas

Aqui vamos instalar as ferramentas essenciais para que possamos dar continuidade que são elas, Python , Pycharm, Insomnia.

- **Python** é a linguagem para executarmos os algoritmos.
- **Pycharm** é uma IDE para auxiliar na escrita do código.
- **Insomnia** é um cliente REST para executarmos os testes



Python



Pycharm Community



Insomnia

- Links Para download

Python - <https://www.python.org/downloads/release/python-370/>

PyCharm Community - <https://www.jetbrains.com/pt-br/pycharm/download/>

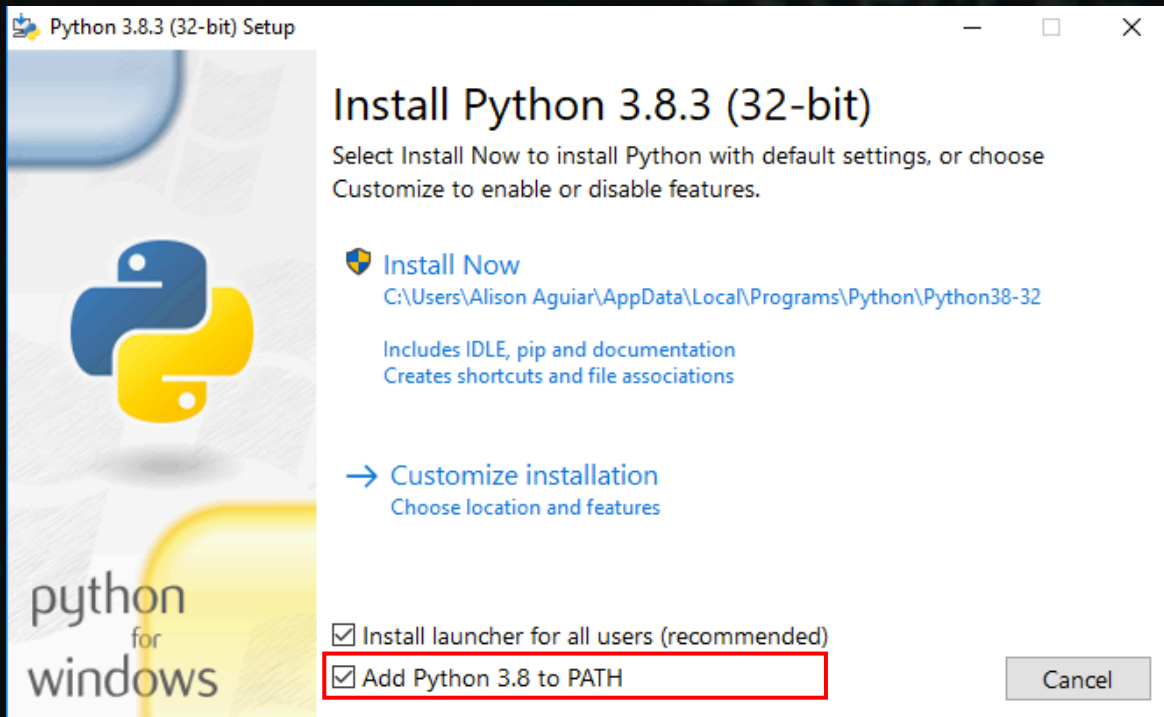
Insomnia - <https://insomnia.rest/download/core/>

Se você é usuário Linux o Python já vem por padrão instalado.

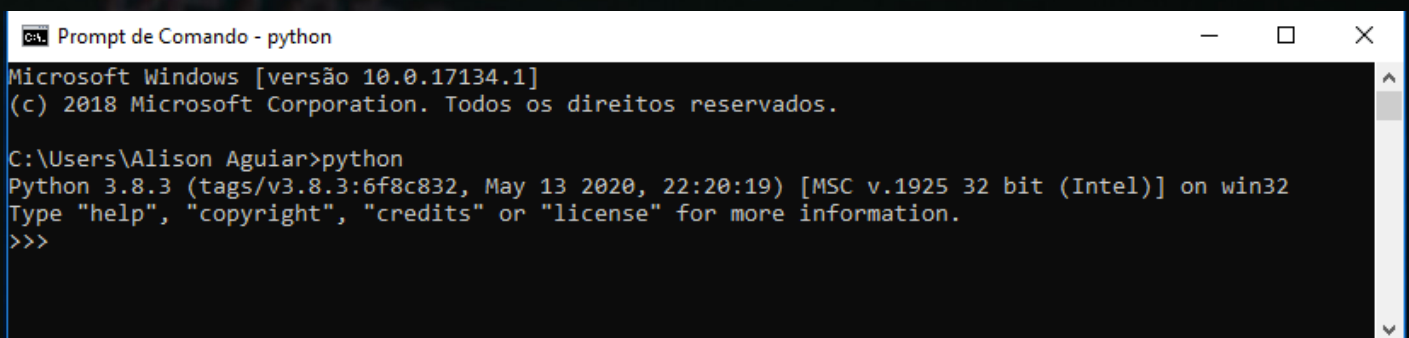
Instalando Python



No momento da instalação do **Python** existe uma parte muito importante que é preciso ficar atento para ter acesso ao python em todo o sistema pelo **Terminal** (Prompt de comando) que é adicionando ao **PATH** marcando como na imagem abaixo.



Ao terminar a instalação para verificar se esta tudo ok abra o **Terminal** e digite python. O resultado é algo similar ao que esta na imagem abaixo.

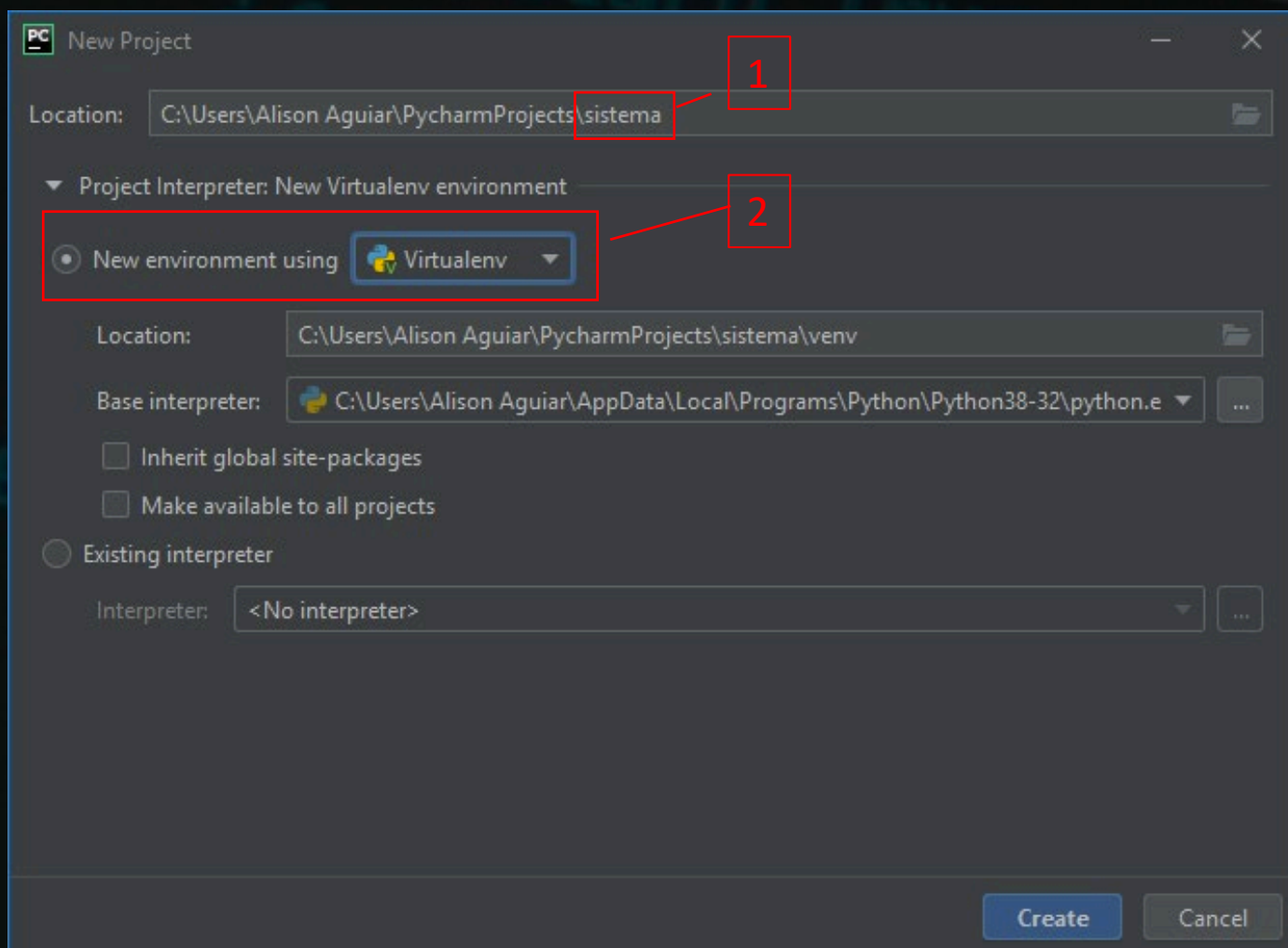


Instalando Pycharm Community e Insomnia



Com **python** devidamente instalado e o **Pycharm** Baixado a instalação é simples como os outros programas Windows (Próximo, Próximo, Próximo...) e o Insomnia não é instalável ele apenas executa.

Abra o **Pycharm** e aceite os termos de uso. Escolha o tema padrão e continue



1 - nome do projeto.

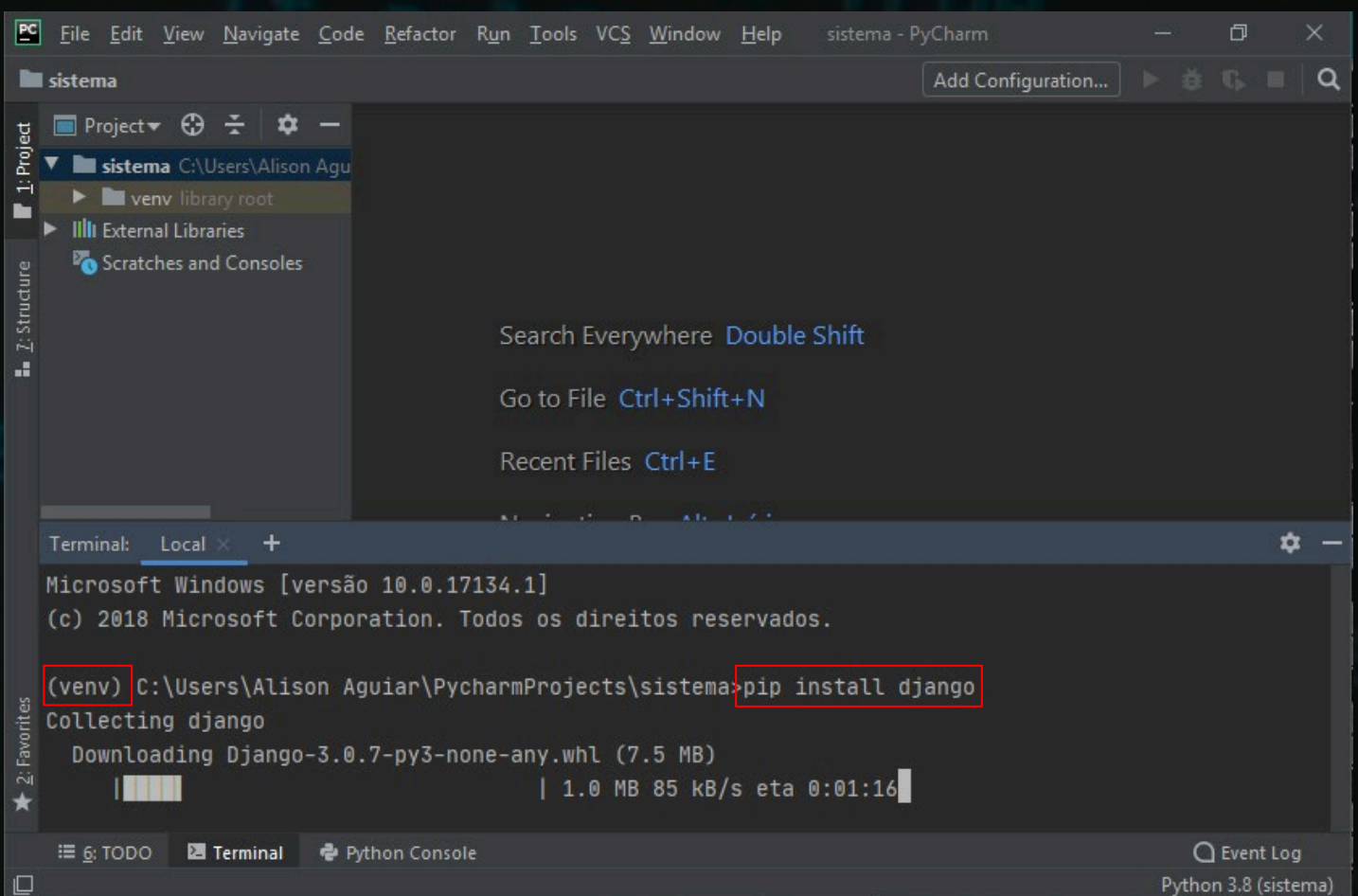
2 - variável de ambiente isola o seu projeto do restante do sistema(tudo que você instalar para o **Python** com este ambiente ativo ficara apenas no nele deixando o python global do sistema sem mudanças).

Clique em Create

Instalando o Django Framework

1 - Com o ambiente python criado vamos instalar o Django no seu ambiente ,certifique que o ambiente esta ativo com o prefixo no terminal como mostra abaixo (**venv**).
venv é o nome que Pycharm da por padrão ao ambiente.

2- instale o **Django** com o comando no terminal : **pip install django**



Criando Projeto Django

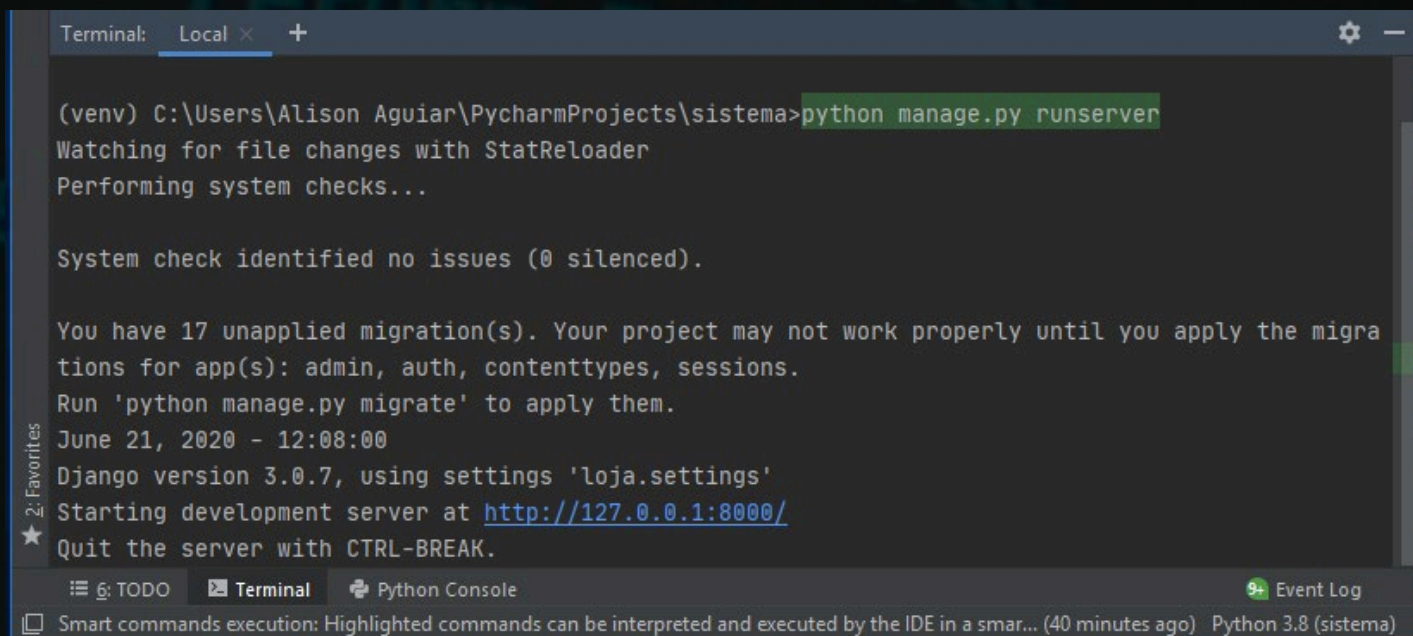
1 - Com o **Django** Instalado digite no terminal: `django-admin startproject <nome do projeto>` .

```
django-admin startproject startproject loja .
```

O ponto(.) ao final do código significa que ele vai criar a estrutura do projeto dentro de uma pasta loja, caso contrario ele criaria uma pasta loja dentro de outra pasta loja

2 – Agora vamos levantar o servidor(colocar o sistema pra executar), para isso o django ao criar o projeto é criado junto um arquivo para auxiliar a gerenciar o projeto este arquivo é o **manage.py**, manda no terminal o seguinte código.

```
Python manage.py runserver
```



```
Terminal: Local x +
(venv) C:\Users\Alison Aguiar\PycharmProjects\sistema>python manage.py runserver
Watching for file changes with StatReloader
Performing system checks...

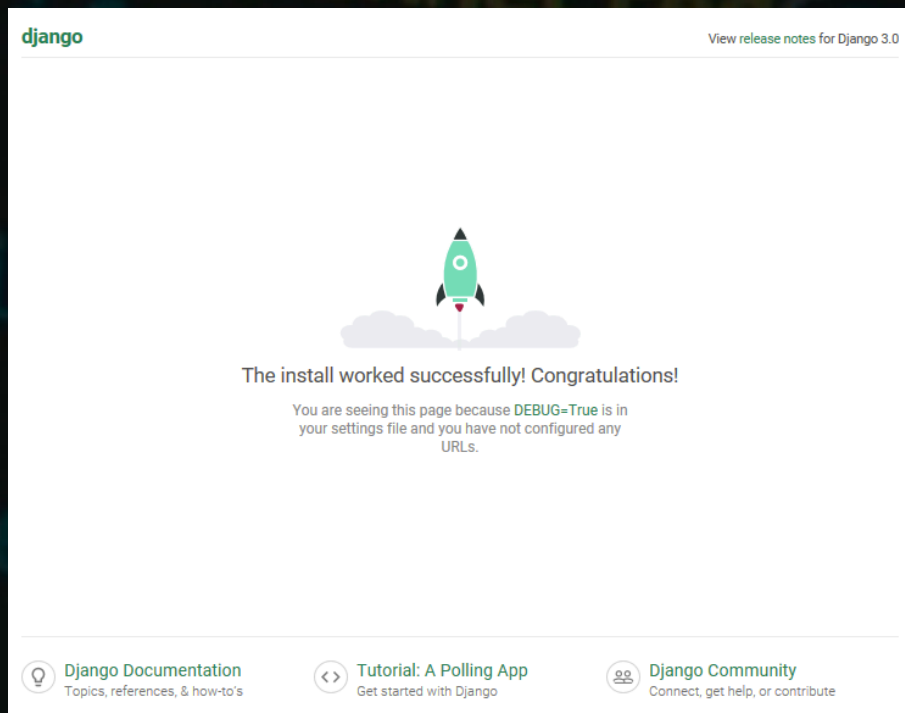
System check identified no issues (0 silenced).

You have 17 unapplied migration(s). Your project may not work properly until you apply the migrations for app(s): admin, auth, contenttypes, sessions.
Run 'python manage.py migrate' to apply them.
June 21, 2020 - 12:08:00
Django version 3.0.7, using settings 'loja.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CTRL-BREAK.
★ 2 Favorites
[+] Event Log
Smart commands execution: Highlighted commands can be interpreted and executed by the IDE in a smar... (40 minutes ago) Python 3.8 (sistema)
```

3 – Agora abra seu navegador e digite o endereço do servidor local:
`http://127.0.0.1:8000`

Servidor Rodando

O Resultado devera ser o seguinte



Agora está chegando a parte divertida. Vamos voltar para a IDE

4- Vamos Criar o primeiro app

O que é um app no Django?

R: Um app é um modulo onde ficarão os todos os recursos relacionados a ele como, model, view, urls,

Models: todos os modelos(entidade) com seus respectivos dados(Atributos) ex: model **Cachorro** que vai possuir nome e raça .

Urls: aqui ficaram as rotas de acesso ex: `http://127.0.0.1:8000/cachorro`

View: aqui ficaram as funções ações que vão ser tratadas quando usuário(cliente) solicitar a rota(url, endpoint).

Manda No terminal

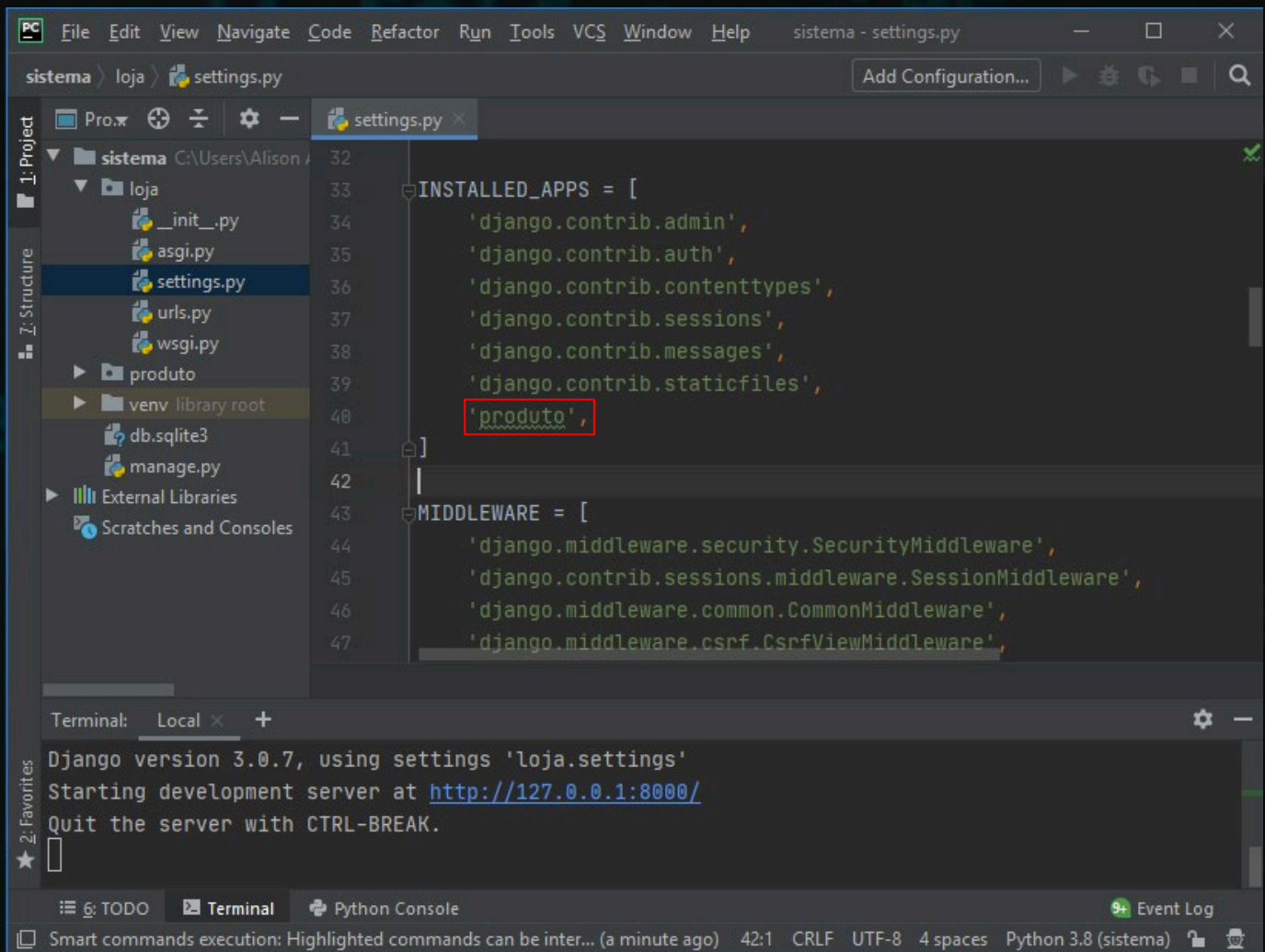
```
python manage.py startapp produto
```


Criando Primeiro App

Para criar o primeiro app a para a pasta raiz do projeto ao mesmo nível de do arquivo **manage.py** e manda no terminal

```
python manage.py startapp produto
```

Após criar o app precisamos declarar ele no **settings.py** do seu projeto, vai no arquivo **settings.py** dentro de **loja** e coloca o nome do app criado dentro do dicionário **INSTALLED_APPS**.



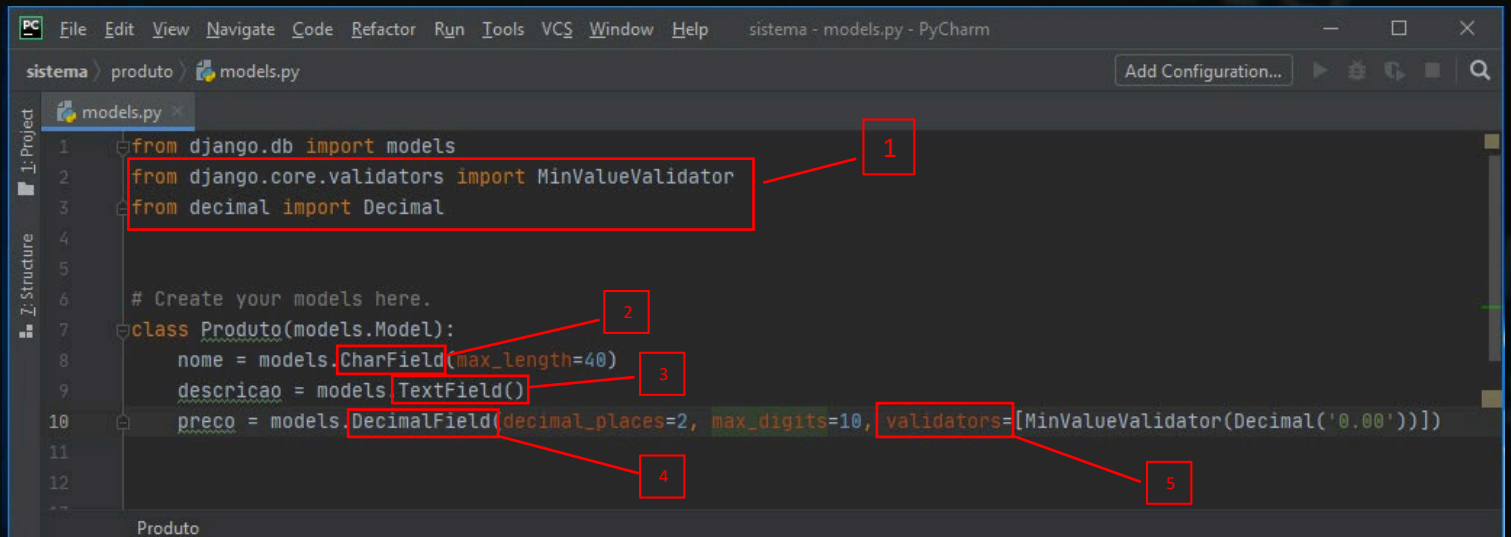
```
File Edit View Navigate Code Refactor Run Tools VCS Window Help sistema - settings.py
sistema > loja > settings.py Add Configuration...
Project
  sistema C:\Users\Alison\...
    loja
      __init__.py
      asgi.py
      settings.py
      urls.py
      wsgi.py
    produto
    venv library root
      db.sqlite3
      manage.py
    External Libraries
    Scratches and Consoles
Structure
  sistema
    loja
      settings.py
Terminal: Local x +
Django version 3.0.7, using settings 'loja.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CTRL-BREAK.
[]
6: TODO Terminal Python Console Event Log
Smart commands execution: Highlighted commands can be inter... (a minute ago) 42:1 CRLF UTF-8 4 spaces Python 3.8 (sistema)
```

```
32
33 INSTALLED_APPS = [
34     'django.contrib.admin',
35     'django.contrib.auth',
36     'django.contrib.contenttypes',
37     'django.contrib.sessions',
38     'django.contrib.messages',
39     'django.contrib.staticfiles',
40     'produto',
41 ]
42
43 MIDDLEWARE = [
44     'django.middleware.security.SecurityMiddleware',
45     'django.contrib.sessions.middleware.SessionMiddleware',
46     'django.middleware.common.CommonMiddleware',
47     'django.middleware.csrf.CsrfViewMiddleware',
```

Sendo Assim vamos criar o nosso primeiro model.

vamos na pasta **produto** no arquivo **models.py** e criar os atributos para o registro de nosso produto .

Obs: lembrando que nunca declare variáveis classes com acentos e símbolos



```
1 from django.db import models
2 from django.core.validators import MinValueValidator
3 from decimal import Decimal
4
5
6 # Create your models here.
7 class Produto(models.Model):
8     nome = models.CharField(max_length=40)
9     descricao = models.TextField()
10    preco = models.DecimalField(decimal_places=2, max_digits=10, validators=[MinValueValidator(Decimal('0.00'))])
```

The screenshot shows a PyCharm IDE window titled 'sistema - models.py - PyCharm'. The code in the editor is for a Django model named 'Produto'. Five red boxes with numbers 1 through 5 are placed over the code, with lines pointing to them. Box 1 points to the import statements. Box 2 points to 'CharField' in line 8. Box 3 points to 'TextField' in line 9. Box 4 points to 'DecimalField' in line 10. Box 5 points to the 'validators' list in line 10.

- 1: Importações necessárias.
- 2: **CharField** é um campo de texto que você precisa definir um tamanho máximo
- 3: **TextField** você não precisa definir o tamanho.
- 4: **DecimalField** você precisa definir casas decimais(decimal_field) e o máximo de dígitos(max_digits)
- 5: **Validators** não é obrigatório mais colocaremos para impedir de um usuário colocar numero negativo.

Fazendo as migrações.

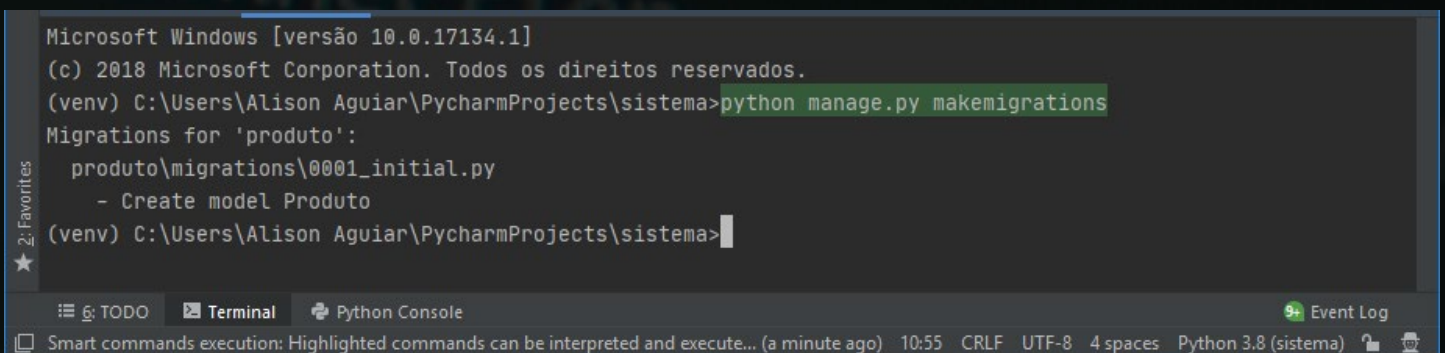
Após criar modificar ou deletar algum model vice precisa fazer as migrações.

O que são as migrações?

R: As **migrations**(migrações) são **arquivos.py** que ficam dentro do app com histórico de mudanças dos modelos. Para aplicar no banco de dados.

Nestes arquivos de migrações contem as etapas para o **Django** criar, deletar, alterar as tabelas e colunas no banco de dados(que por padrão estamos usando o **Sqlite**, porque ele não precisa de configurações e instalação). Para criar as migrations manda no terminal.

```
python manage.py makemigrations
```

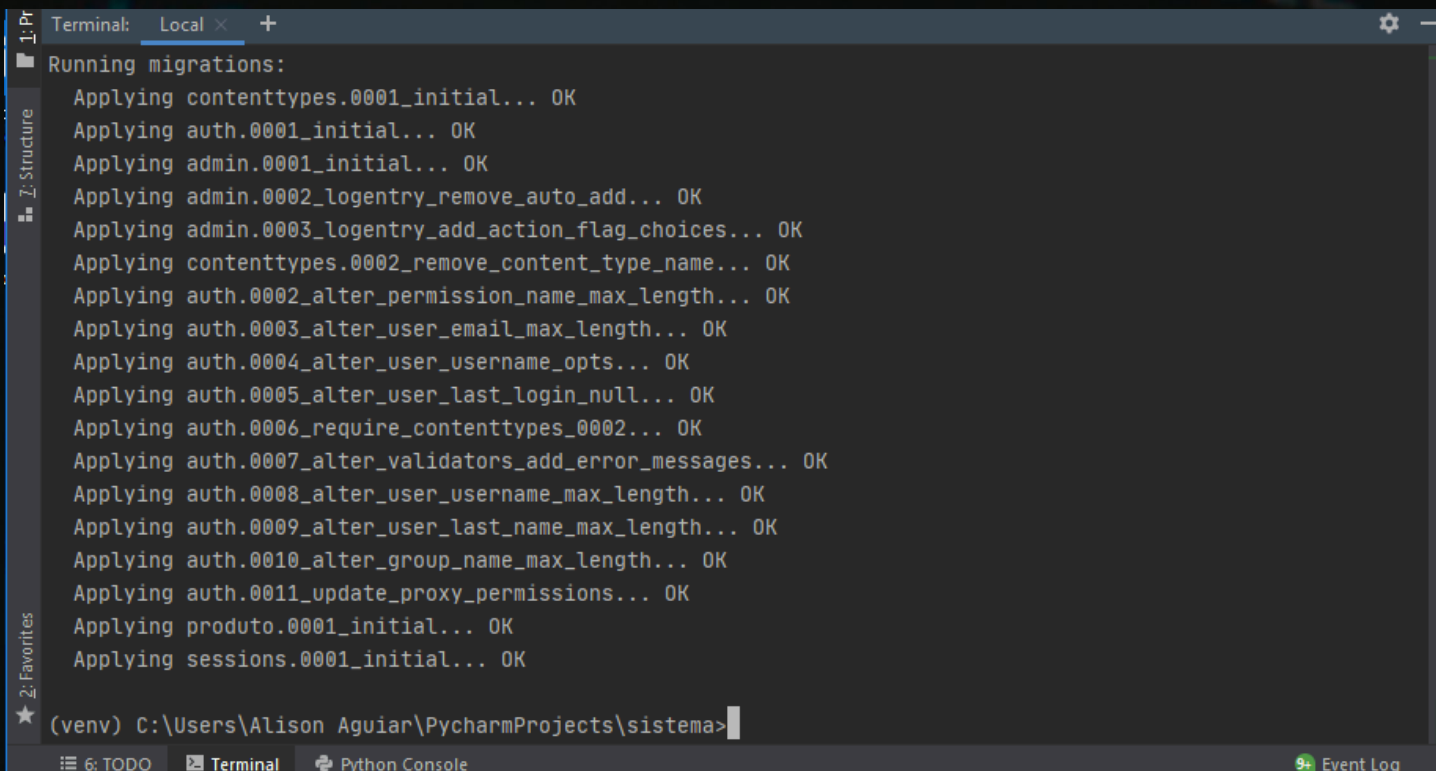


```
Microsoft Windows [versão 10.0.17134.1]
(c) 2018 Microsoft Corporation. Todos os direitos reservados.
(venv) C:\Users\Alison Aguiar\PycharmProjects\sistema>python manage.py makemigrations
Migrations for 'produto':
  produto\migrations\0001_initial.py
    - Create model Produto
(venv) C:\Users\Alison Aguiar\PycharmProjects\sistema>
```

The screenshot shows a terminal window with the command 'python manage.py makemigrations' executed. The output shows the creation of a migration file '0001_initial.py' for the 'produto' app, which includes creating the 'Produto' model. The terminal is running in a virtual environment 'venv' at the path 'C:\Users\Alison Aguiar\PycharmProjects\sistema'.

Agora para fazer as mudanças no banco de dados temos que executar essa migrations no banco de dados..
Manda no terminal

```
python manage.py migrate
```



```
Terminal: Local x +
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 produto.0001_initial... OK
Applying sessions.0001_initial... OK
(venv) C:\Users\Alison Aguiar\PycharmProjects\sistema>
```

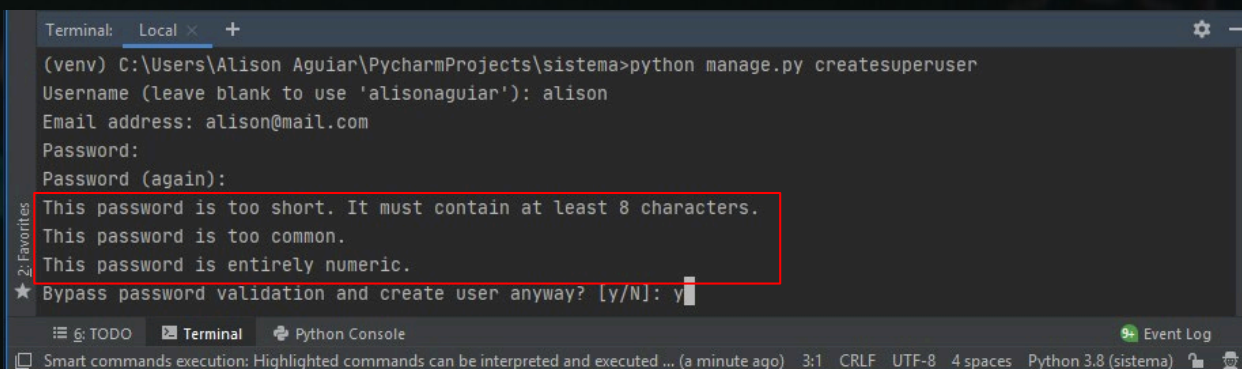
nesse log você percebe que tem dados relacionados a admin, user, etc. Isso porque o django possui por padrão um sistema de usuários, e autenticação, permissões e grupos. Com criptografia de senha login, registro, troca de senha etc. e você não precisa se preocupar com esse trabalho que na maioria dos projetos você vai usar.

Criando um Super User

Para testarmos o modelo criado , vamos aproveitar uma das ferramentas mais incríveis que o Django possui , que é o painel administrativo do Django.

Para isso vamos criar um **superusuário** e colocar nosso modelo no painel.
Manda no terminal.

```
python manage.py createsuperuser
```



```
Terminal: Local x +
(venv) C:\Users\Alison Aguiar\PycharmProjects\sistema>python manage.py createsuperuser
Username (leave blank to use 'alisonaguiar'): alison
Email address: alison@mail.com
Password:
Password (again):
This password is too short. It must contain at least 8 characters.
This password is too common.
This password is entirely numeric.
Bypass password validation and create user anyway? [y/N]: y
```

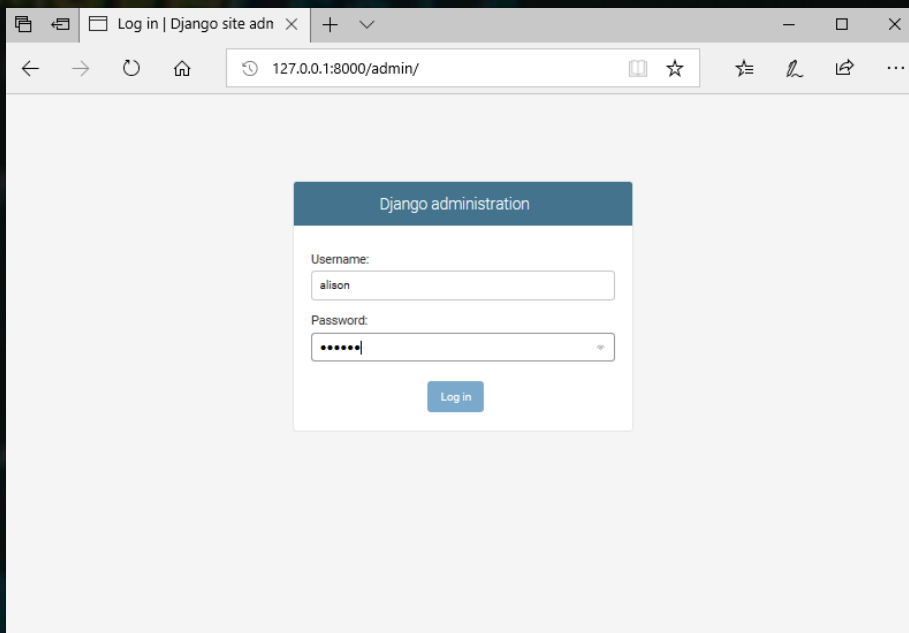
Essas mensagens em vermelho aparecem pra mim porque o django tem um sistema que verifica o nível de senha fraca ou forte e te notifica , eu usei uma senha muito poderosa (123456 : 👁 👁).
Mais coloquem um: usuário, um e-mail, e uma senha.

Django Admin

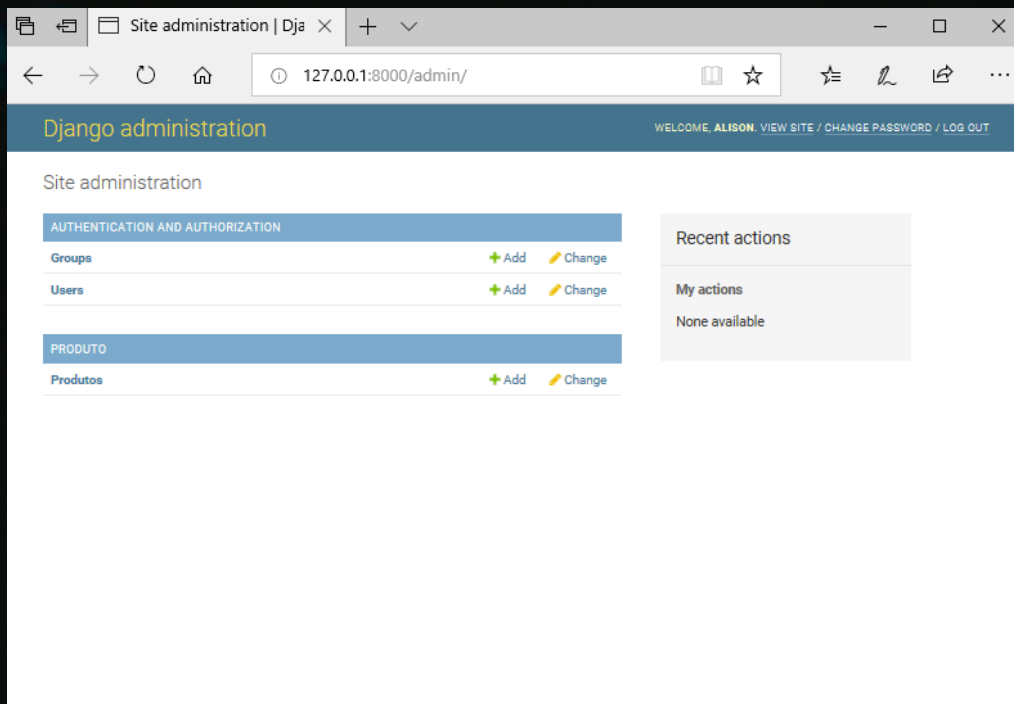
Para colocar nosso modelo no painel admin vamos ao arquivo **admin.py** importar o modelo produto seguindo o seguinte exemplo

```
models.py x views.py x admin.py x
1 from django.contrib import admin
2 from .models import Produto
3
4
5 # Register your models here.
6 admin.site.register(Produto)
7 |
```

Vamos agora ao **painel administrativo** do django no seguinte endereço **127.0.0.1:8000/admin**
Aqui você vai cair na tela de login insira usados no **createsuperuser** para logar.

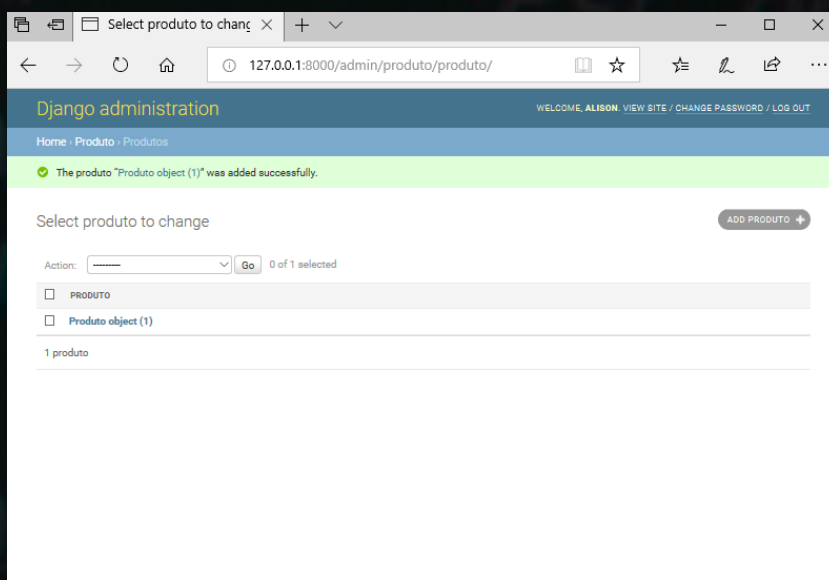
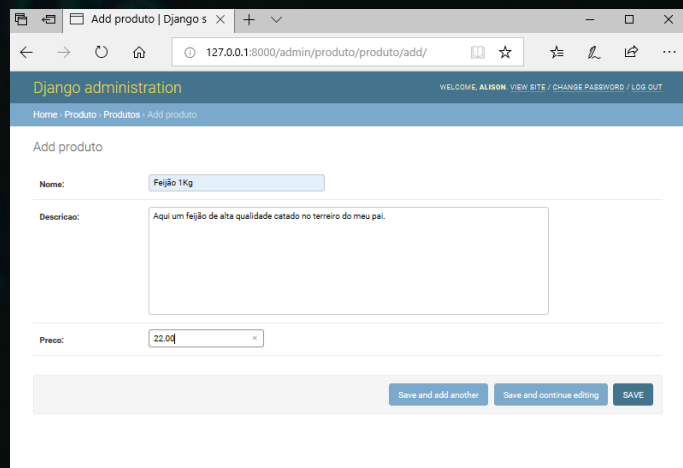


Criando Produto através do admin



↑ Este é o painel do django click em **add** ao lado direito de **Produtos** você ira para o formulário de registro da imagem direita →

Preencha os dados conforme queira e click em **save**



Customizando django admin

Aqui vamos aprender a dar mais vida ao django admin . Vamos no arquivo anterior **admin.py** do app **produto** e segue o exemplo a seguir.

```
models.py x views.py x admin.py x
1 from django.contrib import admin
2 from .models import Produto
3 # Register your models here.
4
5
6 class AdminProduto(admin.ModelAdmin):
7     list_display = ['nome', 'descricao', 'preco']
8     search_fields = ['nome', 'descricao']
9
10
11
12 admin.site.register(Produto, AdminProduto)
13
```

list_display: aqui é quais dados serão mostrados na listagem dos produtos
search_fields: aqui são os campos pesquisáveis.

The screenshot shows the Django Admin interface for the 'produto' app. The browser address bar shows '127.0.0.1:8000/admin/produto/produto/'. The page title is 'Django administration' and the user is 'ALISON'. The breadcrumb trail is 'Home > Produto > Produtos'. The main heading is 'Select produto to change'. There is a search input field with a magnifying glass icon and a 'Search' button. Below the search field, there is an 'Action:' dropdown menu and a 'Go' button. The table below shows the search results:

| <input type="checkbox"/> | NOME | DESCRICAO | PRECO |
|--------------------------|------------|----------------------------------------------------------------|-------|
| <input type="checkbox"/> | Feijão 1Kg | Aqui um feijão de alta qualidade catado no terreno do meu pai. | 22.00 |

At the bottom of the table, it says '1 produto'. Red boxes highlight the search input field and the table columns. Labels 'Search_fields' and 'list_display' point to these elements.

Instalando o Django Rest Framework

Aqui vamos dar início a exposição desse modelo que criamos em uma **API**, para isso precisamos instalar o **django-rest-framework**.
Manda no terminal.

```
pip install djangorestframework
```

```
(venv) C:\Users\Alison Aguiar\PycharmProjects\sisema>pip install djangorestframework
Collecting djangorestframework
  Using cached djangorestframework-3.11.0-py3-none-any.whl (911 kB)
Requirement already satisfied: django>=1.11 in c:\users\alison aguiar\pycharmprojects\sisema\venv\lib\site-packages (from djangorestframework) (3.0.7)
Requirement already satisfied: asgiref<=3.2 in c:\users\alison aguiar\pycharmprojects\sisema\venv\lib\site-packages (from django>=1.11->djangorestframework) (3.2.9)
Requirement already satisfied: sqlparse>=0.2.2 in c:\users\alison aguiar\pycharmprojects\sisema\venv\lib\site-packages (from django>=1.11->djangorestframework) (0.3.1)
Requirement already satisfied: pytz in c:\users\alison aguiar\pycharmprojects\sisema\venv\lib\site-packages (from django>=1.11->djangorestframework) (2020.1)
Installing collected packages: djangorestframework
Successfully installed djangorestframework-3.11.0
```

E também devemos declarar em **INSTALLED_APPS** no arquivo **settings.py**

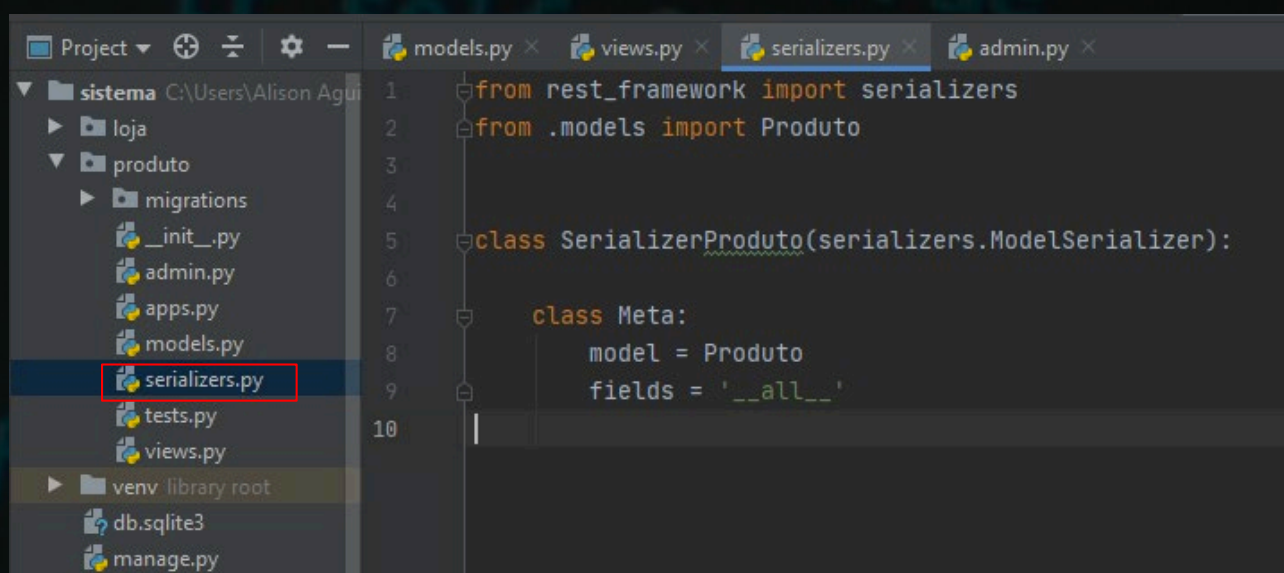
```
32
33 INSTALLED_APPS = [
34     'django.contrib.admin',
35     'django.contrib.auth',
36     'django.contrib.contenttypes',
37     'django.contrib.sessions',
38     'django.contrib.messages',
39     'django.contrib.staticfiles',
40     'rest_framework',
41     'produto',
42 ]
43
```

Criando Um Serializes

Para recebermos um **JSON** da nossa API precisamos de um **Serializer** do modelo.

Resumidamente Um **Serializer** é uma classe que é responsável por transformar um **queryset**(uma busca de dados com o django) em **JSON**(objeto que o cliente vai receber)

vamos ao app **produto** criar um arquivo **serializers.py** e digitar a seguinte.



```
1 from rest_framework import serializers
2 from .models import Produto
3
4
5 class SerializerProduto(serializers.ModelSerializer):
6
7     class Meta:
8         model = Produto
9         fields = '__all__'
10
```

Neste arquivo **serializers.py** estamos importando o modelo **Produto** também herdando a classe **serializers.ModelSerializer** responsável por fazer a transformação do objeto Python (**queryset**) para um **JSON** válido

Na meta class eu declaro qual **modelo** desejo transformar (serializar) em **JSON** e **fields** são os campos que eu desejo ter acesso neste caso declarei **__all__** para mostrar todos os campos, caso eu queira mostrar apenas alguns eu declararia por exemplo: **fields = ['nome', 'descricao']**

Caso eu queira apenas remover os campos que não quero ter acesso é só remover **fields** e declarar **exclude = ['preco']**

Criando Um ViewSet

A view set serve para tratamos os dados que receberemos nas requisições HTTP(**GET,POST,PUT,DELETE**)para a rota (url, endpoint) um **ViewSet** é declarado no arquivo **views.py** Segue o exemplo de uma viewset.

```
models.py × urls.py × serializers.py × views.py × admin.py ×
1  from django.shortcuts import render
2      from rest_framework.viewsets import ModelViewSet
3      from .serializers import SerializerProduto
4  from .models import Produto
5      # Create your views here.
6
7
8  class ProdutoViewSet(ModelViewSet):
9      serializer_class = SerializerProduto
10     queryset = Produto.objects.all()
11
12
```

serializer_class é a classe que transforma a busca do queryset em um **JSON**.

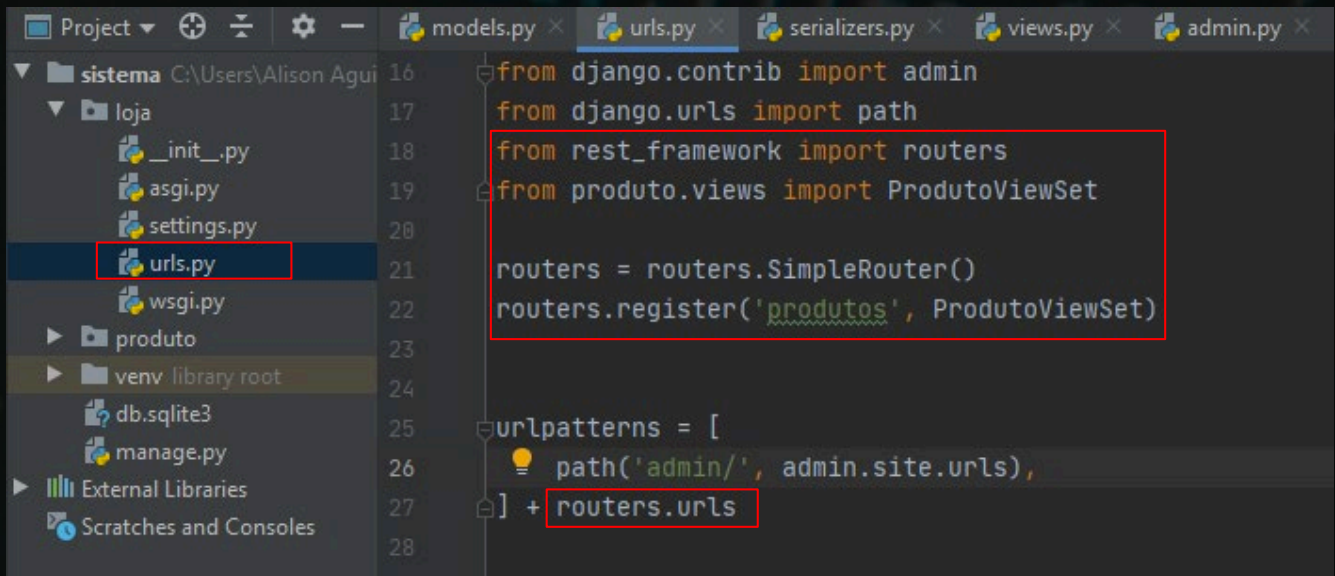
queryset é usado para executar comandos no banco de dados
Ex: **Produto.objects.all()** é equivalente a **SELECT * FROM PRODUTOS**
É isso que um **ORM**(object relational mapping) faz. Sendo assim raramente você executara comandos **SQL** aqui no **Django**.

Saiba mais sobre aqui queryset [Clicando aqui](#)

Criando a rota Para Acessar a API

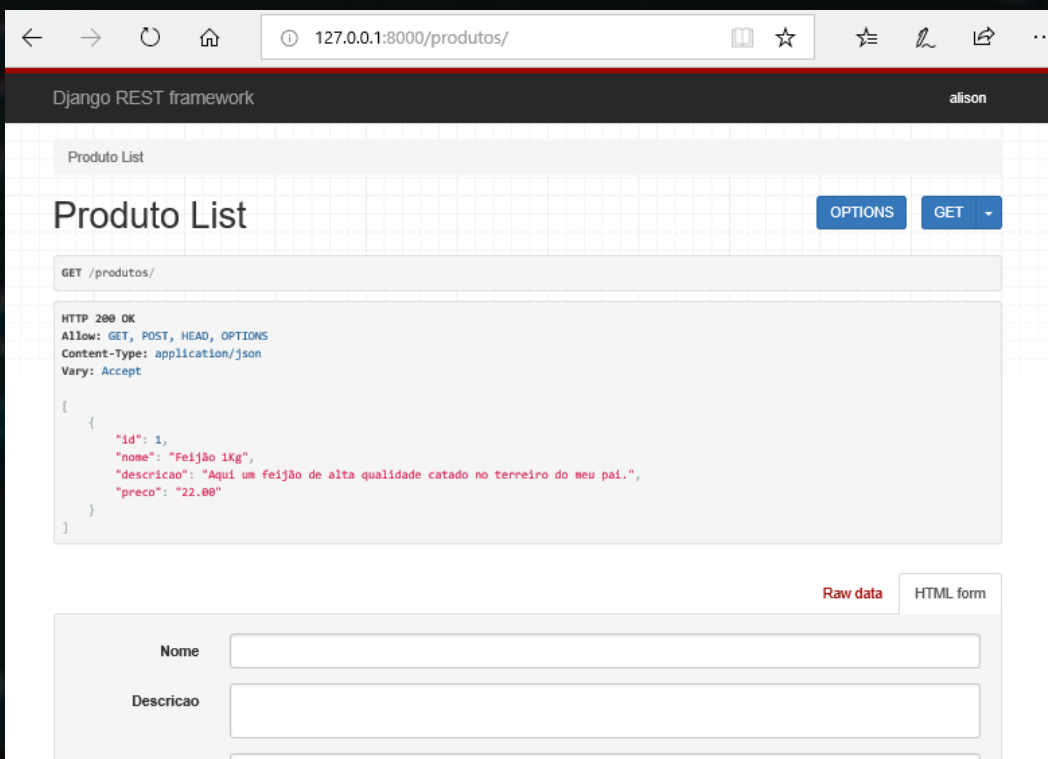
Para isso vamos ao arquivo `urls.py` da pasta raiz do projeto (loja) importe as fontes necessárias destacadas na imagem.

Obs: quando for criado uma rota com nome no singular ou plural mantenham um padrão ou tudo **singular** ou **plural** neste caso deixei plural é uma boa pratica.



```
16 from django.contrib import admin
17 from django.urls import path
18 from rest_framework import routers
19 from produto.views import ProdutoViewSet
20
21 routers = routers.SimpleRouter()
22 routers.register('produtos', ProdutoViewSet)
23
24
25 urlpatterns = [
26     path('admin/', admin.site.urls),
27     routers.urls
28 ]
```

Agora veja uma magia acessando a rota(url, endpoint) 127.0.0.1:8000/produtos



Entendendo o ViewSet

o `ModelViewSet` usado em `views.py` é uma herança de outras **6** classes

`mixins.CreateModelMixin`,
`mixins.RetrieveModelMixin`,
`mixins.UpdateModelMixin`,
`mixins.DestroyModelMixin`,
`mixins.ListModelMixin` e
`GenericViewSet`

Para recuperar essas classes elas estão em:

```
from rest_framework import mixins
from rest_framework.viewsets import GenericViewSet
```

Então o `ModelViewSet` vai criar pra você as seguintes rotas

GET `produtos` → busca todos os produtos.

GET `produtos/pk` → `pk` é a chave primaria de um produto, todo model por padrão já cria `id` como chave primaria para você. é um numero que faz alto incremento começando de 1. ex de requisição: `127.0.0.1:8000/produtos/1/`

POST `produtos/` → o post você precisa passar o **body** no payload contendo os dados que o modelo precisa para ser criado como nome descrição preço, veremos adiante.

DELETE `produtos/pk/` → deleta um produto passando a chave primaria(`id`)

PUT `produtos/pk/` → atualiza um produto passando o `pk(id)` do produto e um body.

PATCH `produtos/pk/` → atualiza um produto passando o `pk(id)` do produto e os demais campos que queira atualizar.

Cada classe é responsável por uma função que são elas `list()`, `retrieve()`, `create()`, `update()`, `partial_update()`, e `destroy()`. você pode facilmente sobrescrever desde que retorne um **Response**.

Mais sobre [Clicando aqui](#)

Testando a API

Você pode testar a **API** pelo formulário mesmo, porem vamos brincar com o cliente REST que baixamos o **Insomnia** para você entender de uma maneira melhor como funciona abra o Insomnia e digite a rota.

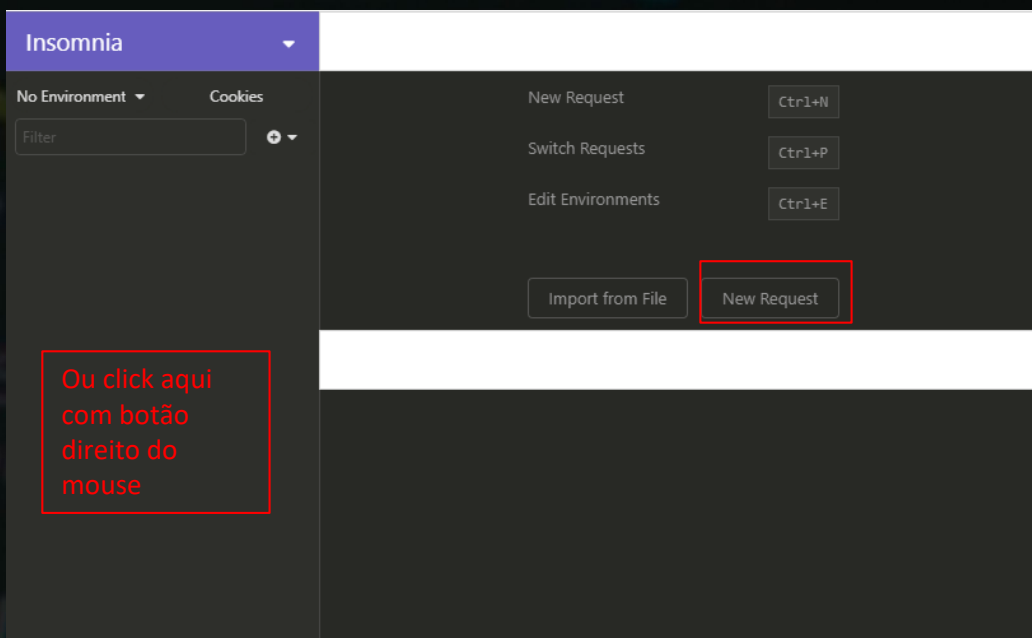
Como citado anteriormente uma aplicação REST é desenvolvida sobre o protocolo http para os métodos mais comuns **GET, POST, PUT, DELETE** e **PATCH**.

GET para buscar, POST para criar PUT para atualizar, DELETE para deletar e patch para atualizar uma parte da entidade(modelo)

Quando você manda um **Request** para o servidor **REST** é enviado junto um **payload** contendo **body** e **header** aqui usaremos apenas body.

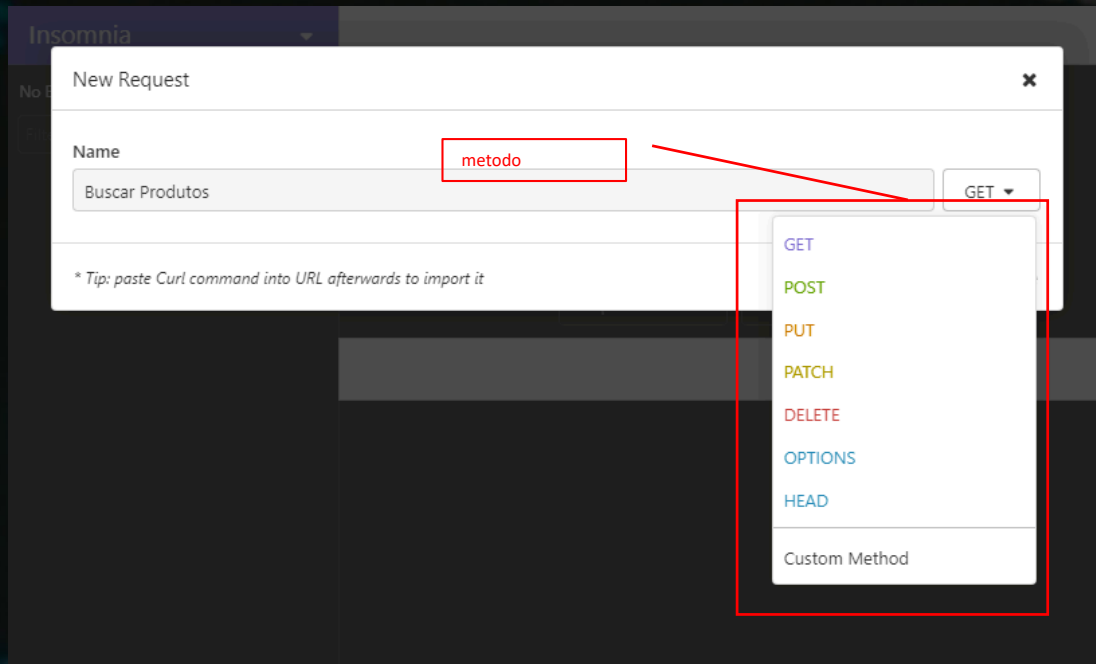
Insomnia

Ao Abrir o Insomnia Crie uma nova requisição em **new request**

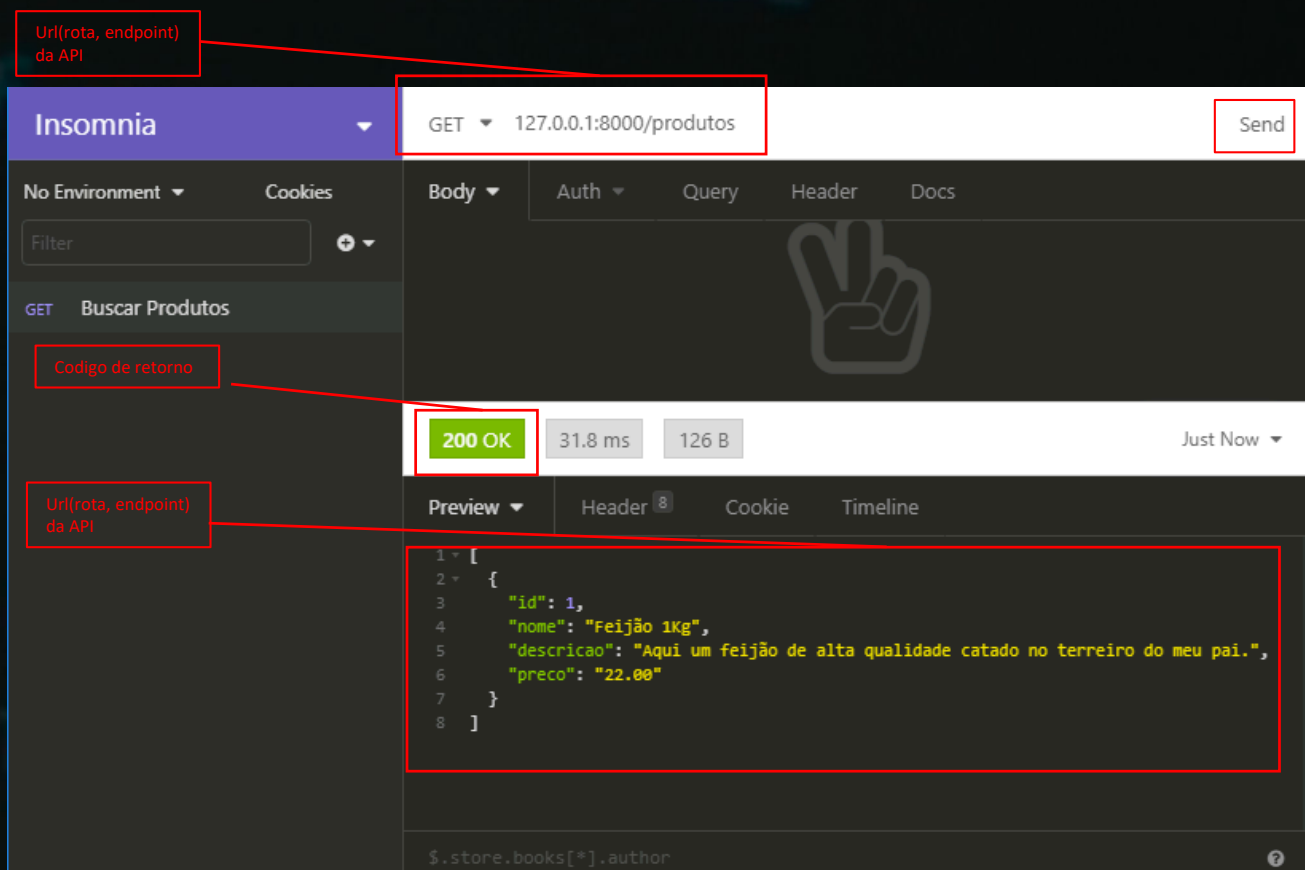


Criando Requisição a API

Escolha um nome para a requisição e um método GET depois create



Preencha os dados de rota e método GET e clique em send



Códigos de retorno mais comuns são: **200** = Ok, **201** = Criado, **204** = sem conteúdo, **400** = Má Requisição, **401** = Não Autorizado e **404** = não encontrado quer conhecer mais sobre pesquisa sobre http status code.

Criando um Produto método **POST**

No Insomnia crie um novo request com nome Registro de Produto com método **POST** Insira a url(endpoint, rota) 127.0.0.1:8000/produtos/

Click na aba body e selecione **multipart** e digite os campos que sua API de produtos precisa para que registre o produto e clique em **send**.

The screenshot shows the Insomnia REST client interface. The request is named "Registrar Produto" and is sent to the endpoint "127.0.0.1:8000/produtos/". The request body is multipart form data with three fields: "nome" (Arroz 1kg), "descricao" (Uma Arroz feito no maranhão), and "preco" (5.50). The response status is "201 Created" with a response time of 183 ms and a body size of 85 B. The response body is a JSON object: { "id": 2, "nome": "Arroz 1kg", "descricao": "Uma Arroz feito no maranhão", "preco": "5.50" }.

Buscando apenas um Produto método **GET**

Crie novo request com nome Buscar um Produto com método **GET** Insira a url(endpoint, rota) 127.0.0.1:8000/produtos/pk pk=id do produto

clique em **send** que será retornado o produto com o id inserido.

Deletando um Produto método **DELETE**

Crie novo request com nome Deletar Produto com método **DELETE** Insira a url(endpoint, rota) 127.0.0.1:8000/produtos/pk pk=id do produto

clique em **send** que será deletado o produto com o id inserido

Atualizando um Produto método **PUT**

No Insomnia crie um novo request com nome Atualizar Produto com método **PUT**
Insira a url(endpoint, rota) 127.0.0.1:8000/produtos/pk/

Click na aba body e selecione **multipart** e digite todos campos que sua API de produtos precisa para que atualize o produto e clique em **send**.

Muito similar ao **POST** só que você precisa passar na rota o **id** do **produto**

Caso não informe o django vai te notificar que os campos em requeridos como na imagem abaixo.

poucos **frameworks** entregam os validadores assim nos seus modelos sem você escrever a regra, outra delicia do **Django**

The screenshot shows the Insomnia REST client interface. On the left, a sidebar lists several API endpoints: 'Buscar um Produto' (GET), 'Buscar Produtos' (GET), 'Atualizar Produto' (PUT), 'Deletar Produto' (DEL), and 'Registrar Produto' (POST). The 'Atualizar Produto' endpoint is selected. The main area shows a PUT request to '127.0.0.1:8000/produtos/1/'. The request body is set to 'Multipart' and contains a single field 'nome' with the value 'Feijão 1Kg'. Below the request body, a status bar indicates a '400 Bad Request' error with a response time of '32.7 ms' and a size of '77 B'. The 'Preview' tab is active, displaying the JSON error response:

```
1 {
2   "descricao": [
3     "This field is required."
4   ],
5   "preco": [
6     "This field is required."
7   ]
8 }
```


Atualizando um Produto método **PATCH**

No Insomnia crie um novo request com nome Atualizar Parte do Produto com método **PUT** Insira a url(endpoint, rota) 127.0.0.1:8000/produtos/pk/. pk=id do produto

Click na aba body e selecione **multipart** e apenas os campos que deseja atualizar clique em **send** o produto será atualizado como na imagem abaixo.

The screenshot shows the Insomnia REST client interface. The top bar indicates the method is PATCH and the URL is 127.0.0.1:8000/produtos/2/. The left sidebar shows a list of requests, with 'Atualizar parte do Produto' selected. The main area shows the request body as multipart form data with two fields: 'nome' with value 'Arroz Maranhão 1Kg' and 'New name' with value 'New value'. Below the request body, the response status is 200 OK, 39.3 ms, 95 B. The response body is a JSON object:

```
1 {
2   "id": 2,
3   "nome": "Arroz Maranhão 1Kg",
4   "descricao": "Uma Arroz feito no maranhão",
5   "preco": "5.50"
6 }
```

Deletando um Produto método **DELETE**

No Insomnia crie um novo request com nome Deletar Produto com método **DELETE** Insira a url(endpoint, rota) 127.0.0.1:8000/produtos/pk/

Ex: 127.0.0.1:8000/produtos/2/

Clique em send e o produto será deletado o status code **204** sem conteúdo.

Aprendizado Extra

Aproveitando a estrutura do projeto aqui vamos aprender algumas mais algumas coisas que o django e o.djangorestframework fornecem.

O que vamos fazer aqui é uma função para para disponibilizar e disponibilizar os produtos registrados através da **API** e do **django admin**.

Vamos no modelo **Produto** e vamos criar mais um atributo chamado **disponivel** do tipo **BooleanField** este novo atributo servira para disponibilizar para o usuário apenas os **produtos** que você definir como disponíveis.

```
1 from django.db import models
2 from django.core.validators import MinValueValidator
3 from decimal import Decimal
4
5 # Create your models here.
6
7 class Produto(models.Model):
8     nome = models.CharField(max_length=40)
9     descricao = models.TextField()
10    preco = models.DecimalField(decimal_places=2, max_digits=10, validators=[MinValueValidator(Decimal('0.00'))])
11    disponivel = models.BooleanField(default=True)
12
13    class Meta:
14        verbose_name = 'produto'
15        verbose_name_plural = 'produtos'
16        ordering = ['-descricao']
```

Nome na interface admin para 1 produto

Default, define um valor padrão caso o usuário não informe

Nome na interface admin para mais de um produto

Ordena a listagem pela descricao no admin caso ['-descricao'] a ordem seria reversa ex: de z-a

Obs: não esqueça de toda vez que você **adicionar, mudar, ou deletar** algo no modelo mandar no terminal: **python manage.py makemigrations** e **python manage.py migrate** para efetuar as mudanças no banco de dados.

```
(venv) C:\Users\Alison Aguiar\PycharmProjects\sisistema>python manage.py makemigrations
Migrations for 'produto':
  produto\migrations\0002_auto_20200623_2227.py
  - Change Meta options on produto
  - Add field disponivel to produto

(venv) C:\Users\Alison Aguiar\PycharmProjects\sisistema>python manage.py migrate
Operations to perform:
  Apply all migrations: admin, auth, contenttypes, produto, sessions
Running migrations:
  Applying produto.0002_auto_20200623_2227... OK

(venv) C:\Users\Alison Aguiar\PycharmProjects\sisistema>
```

Django Admin Custom Actions

Criaremos um arquivo actions.py no diretório do app produto
Com a seguinte estrutura

```
1 def indisponibilizar_produto(adminmodel, request, queryset):
2
3     short_description = 'Disponibilizar Produtos'
4     queryset.update(disponivel=False)
5
6
7
8 def disponibilizar_produto(adminmodel, request, queryset):
9     queryset.update(disponivel=True)
10    short_description = 'Disponibilizar Produtos'
11
```

Agora vamos no arquivo **admin.py** adicionar as actions que acabamos de criar.

```
1 from django.contrib import admin
2 from .models import Produto
3 from .actions import disponibilizar_produto, indisponibilizar_produto
4 # Register your models here.
5
6
7 class AdminProduto(admin.ModelAdmin):
8     list_display = ['nome', 'descricao', 'preco', 'disponivel']
9     search_fields = ['nome', 'descricao']
10    actions = [disponibilizar_produto, indisponibilizar_produto]
11
12
13 admin.site.register(Produto, AdminProduto)
14
```

Vamos no django admin nos produtos testar as novas actions

Usando as Custom Actions no admin

Selecione os produtos e escolha a action de disponibilizar produto e click em go os produtos se tornaram indisponíveis

Django administration WELCOME, ALISON. [VIEW SITE](#) / [CHANGE PASSWORD](#) / [LOG OUT](#)

Home » Produto » Produtos

✔ The produto "Produto object (3)" was added successfully.

Select produto to change ADD PRODUTO +

Q Search

Action: Delete selected produtos
Disponibilizar produto
Indisponibilizar produto 2 of 2 selected

| <input checked="" type="checkbox"/> | NOME | DESCRICAO | PRECO | DISPONIVEL |
|-------------------------------------|-----------------------|------------------------------|-------|------------|
| <input checked="" type="checkbox"/> | Macarrão Gostoso 500g | Macarrão do bom é esse aqui. | 5.50 | ✔ |
| <input checked="" type="checkbox"/> | Arroz Maranhão 1Kg | Uma Arroz feito no maranhão | 5.50 | ✔ |

2 produtos

Resultado.

Django administration WELCOME, ALISON. [VIEW SITE](#) / [CHANGE PASSWORD](#) / [LOG OUT](#)

Home » Produto » Produtos

Select produto to change ADD PRODUTO +

Q Search

Action: 0 of 2 selected

| <input type="checkbox"/> | NOME | DESCRICAO | PRECO | DISPONIVEL |
|--------------------------|-----------------------|------------------------------|-------|------------|
| <input type="checkbox"/> | Macarrão Gostoso 500g | Macarrão do bom é esse aqui. | 5.50 | ✘ |
| <input type="checkbox"/> | Arroz Maranhão 1Kg | Uma Arroz feito no maranhão | 5.50 | ✘ |

2 produtos

Custom Actions no Rest Framework

Faça os importes de action e Response como mostrado abaixo

```
models.py x urls.py x serializers.py x views.py x admin.py x actions.py x
1 from django.shortcuts import render
2 from rest_framework.viewsets import ModelViewSet
3 from .serializers import SerializerProduto
4 from .models import Produto
5 from rest_framework.decorators import action
6 from rest_framework.response import Response
7
8
9 class ProdutoViewSet(ModelViewSet):
10     serializer_class = SerializerProduto
11     queryset = Produto.objects.all()
12     lookup_url_kwarg = 'ids'
13
14     @action(detail=True, methods=['put'])
15     def disponibilizar(self, request, ids):
16         ids = ids.split(',') # separa os ids passados por virgula
17         produtos = Produto.objects.filter(pk__in=ids).update(disponivel=True)
18         # filtra os produtos pelos ids passados
19         return Response('Atualizado')
20
21     @action(detail=True, methods=['put'])
22     def indisponibilizar(self, request, ids):
23         ids = ids.split(',')
24         produtos = Produto.objects.filter(pk__in=ids).update(disponivel=False)
25         print(produtos)
26         return Response('Atualizado')
27
```

Importações necessárias

detail=True define que você pode passar parâmetro na rota ex: os ids dos produtos

Busca os produtos com todos os ids passados no parâmetro

Atualiza o atributo disponivel para True

Abre o Insomnia e cria uma nova **request** nomeada como Disponibilizar Produtos com método **PUT** com exemplo a seguir

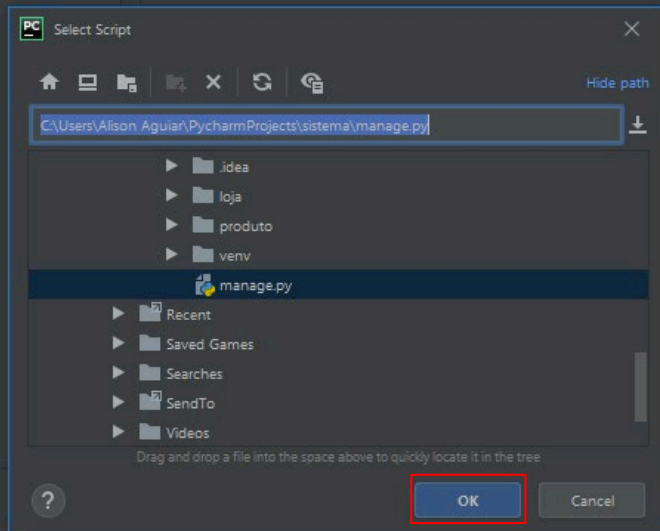
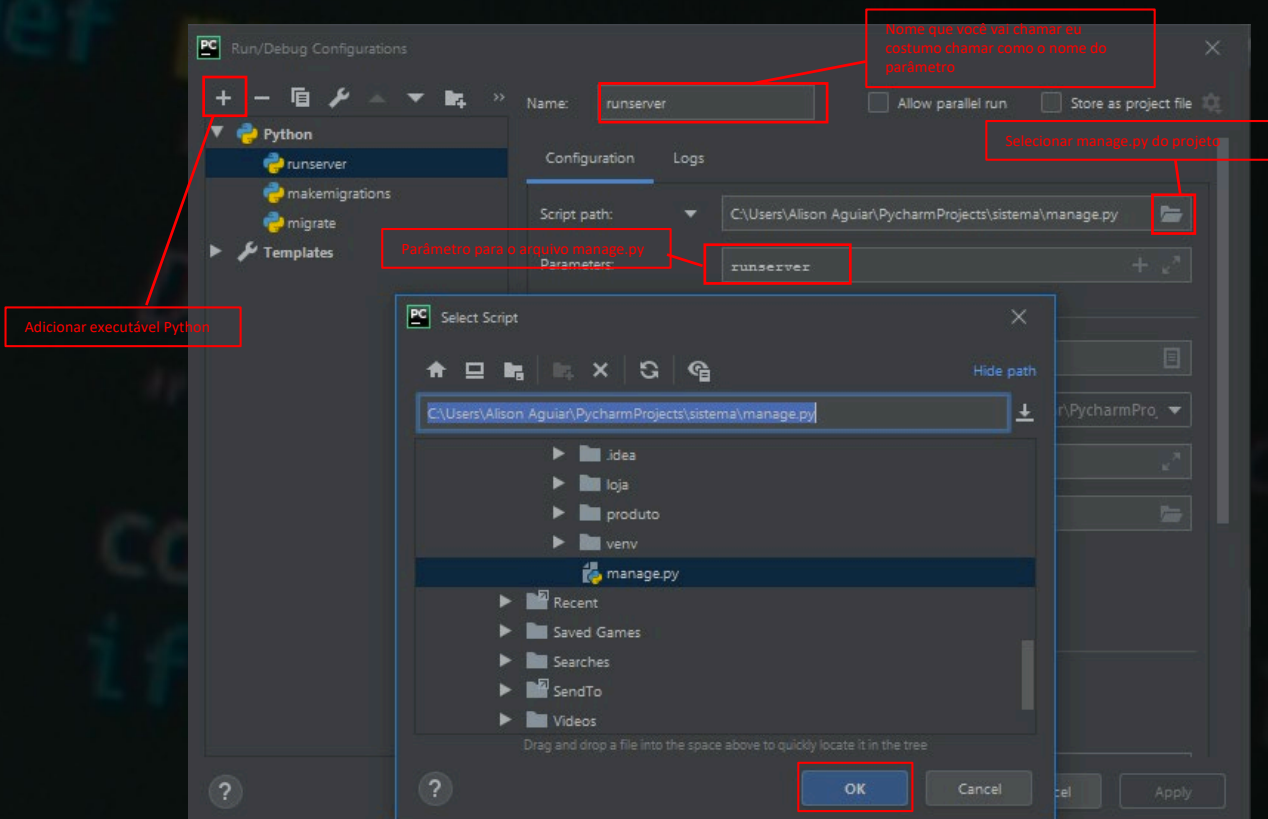
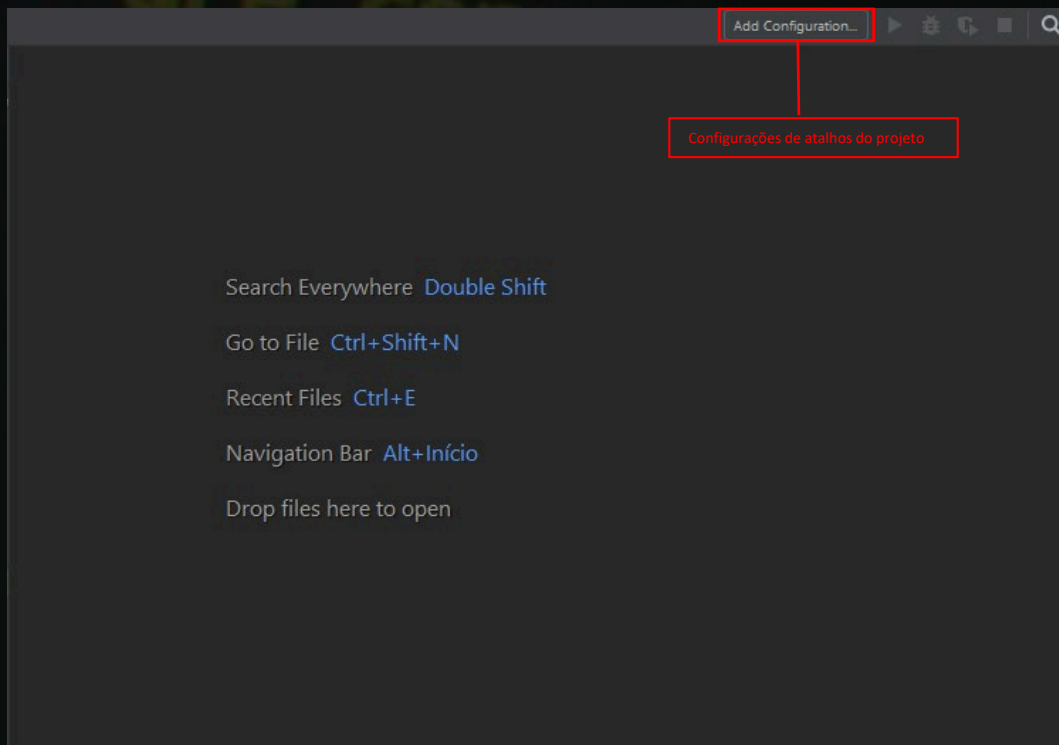
Atualizando Produtos pelo Insomnia

The screenshot displays the Insomnia REST client interface. The top bar shows a PUT request to the endpoint `127.0.0.1:8000/produtos/1,2,3/indisponibilizar/`. The URL path `1,2,3` is highlighted with a red box, and a red arrow points to it with the text "Aqui são os ids dos produtos que você precisa atualizar." (Here are the IDs of the products you need to update).

The left sidebar lists various API actions, with "Indisponibilizar Produtos" (Disable Products) selected. The main area shows the request configuration with tabs for "Multipart", "Auth", "Query", "Header", and "Docs". Below the configuration, the response status is "200 OK" with a response time of "93.1 ms" and a size of "12 B". The response body is shown in the "Preview" tab, displaying the text `"Atualizado"`.

Usando Atalhos do PyCharm

Faça os importes de action e Response como mostrado abaixo



Ficou com alguma duvida?
vai no meu **Instagram** e manda sua
duvida vou te responder

[@ALISON_AGUIAROF](#)

Ou me envia um e-mail

contato@alisonaguiar.com