

ESCOLA E FACULDADE DE TECNOLOGIA SENAI ROBERTO MANGE

DESENVOLVIMENTO DE SISTEMAS

JULHO DE 2022 — SENAI-SP

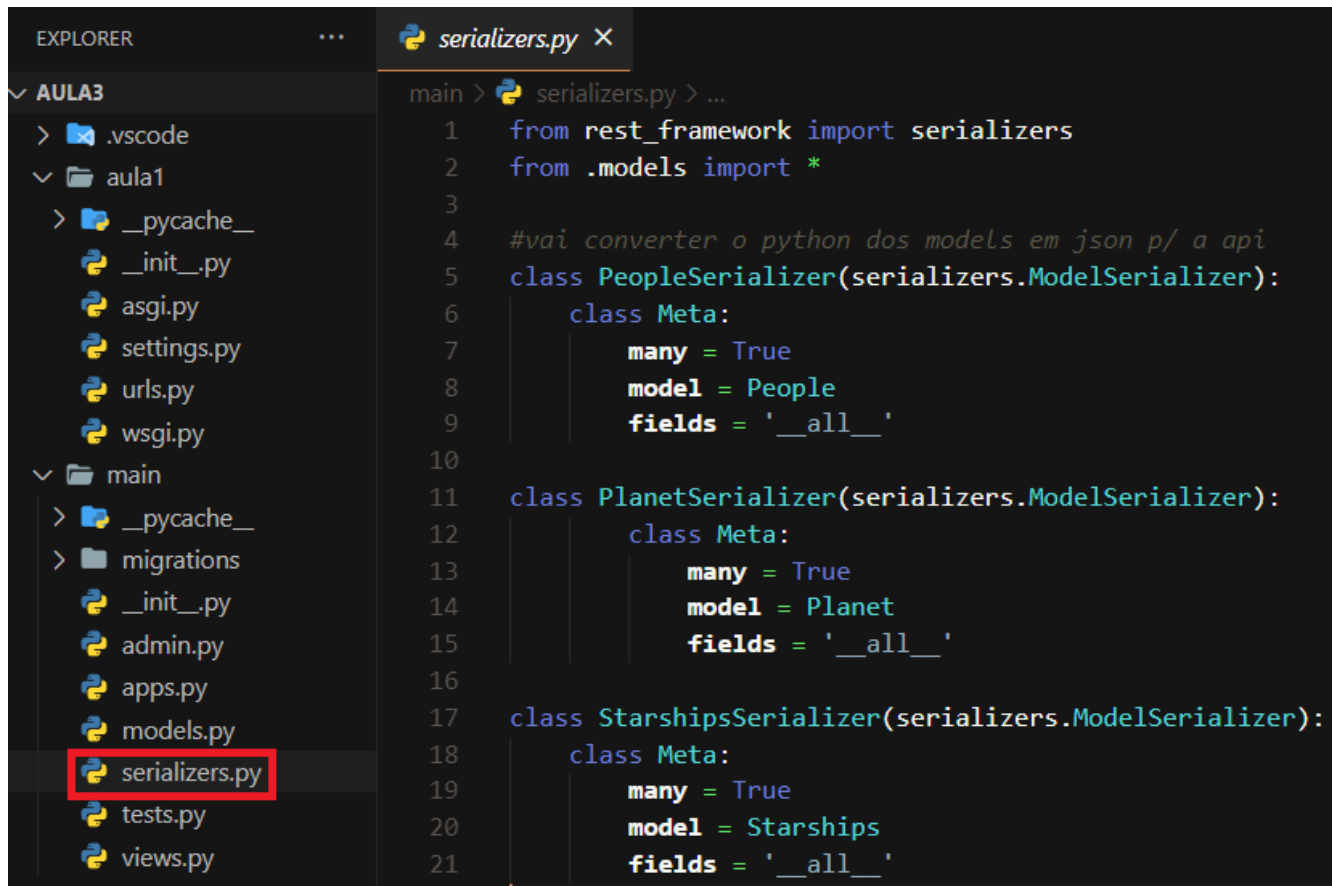


ESCOLA SENAI "ROBERTO MANGE"
UMA REALIZAÇÃO DA INDÚSTRIA

SENAI
SÃO PAULO

PREPARANDO O DJANGO PARA FUNCIONAR COMO API - Serializers

1. Para poder converter o python em JSON e vice-versa, criamos o *serializers.py*



```
EXPLORER
...
serializers.py X

AULA3
├── .vscode
├── aula1
│   ├── __pycache__
│   ├── __init__.py
│   ├── asgi.py
│   ├── settings.py
│   ├── urls.py
│   └── wsgi.py
├── main
│   ├── __pycache__
│   ├── migrations
│   ├── __init__.py
│   ├── admin.py
│   ├── apps.py
│   ├── models.py
│   └── serializers.py
├── tests.py
└── views.py

main > serializers.py > ...
1  from rest_framework import serializers
2  from .models import *
3
4  #vai converter o python dos models em json p/ a api
5  class PeopleSerializer(serializers.ModelSerializer):
6      class Meta:
7          many = True
8          model = People
9          fields = '__all__'
10
11 class PlanetSerializer(serializers.ModelSerializer):
12     class Meta:
13         many = True
14         model = Planet
15         fields = '__all__'
16
17 class StarshipsSerializer(serializers.ModelSerializer):
18     class Meta:
19         many = True
20         model = Starships
21         fields = '__all__'
```

- **many = True** → definimos que o serializer pode converter um conjunto de dados.

- **fields = '__all__'** → definimos que todos os campos da tabela serão serializados (conversão python json e vice versa)

- **model** → definimos qual a tabela do banco

PREPARANDO O DJANGO PARA FUNCIONAR COMO API COM APIVIEW – views.py

1. Agora é necessário criar as views, responsável por definir o que será feito em cada chamada da API:

GET: Chamado quando se deseja obter dados da API

POST: Chamado quando se deseja criar/cadastrar novos dados na API

DELETE: Chamado quando se deseja deletar dados da API

PUT: Chamado quando se deseja alterar dados da API enviando dados completos

PATCH: Chamado quando se deseja alterar dados da API enviando dados parciais

2. Método POST:

Neste código estamos convertendo os dados recebidos em JSON, convertendo em python, verificando se é válido e por último salvando no banco de dados, retornando os dados salvos para o cliente.

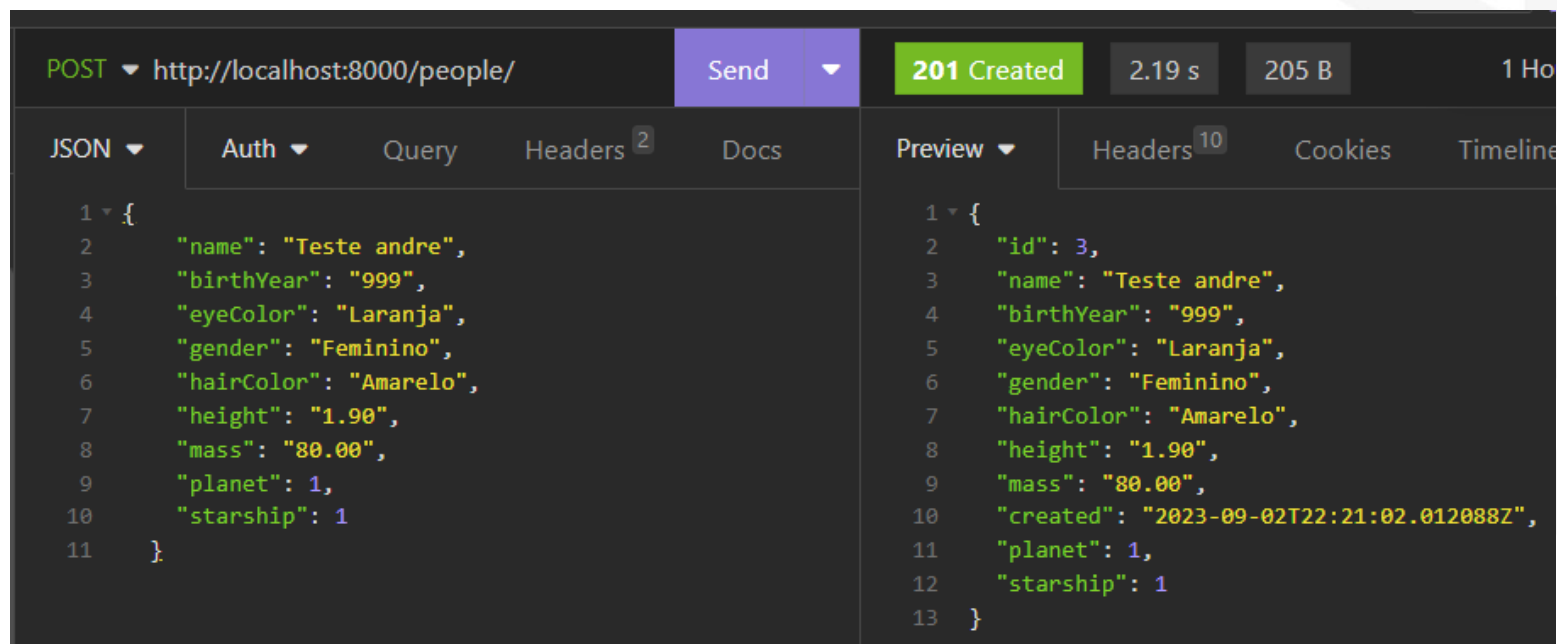
```
class PeopleAPIView(APIView):
    def post(self, request):
        #recebe o json que veio do cliente
        peopleJson = request.data
        #converter json em python!
        peopleSerialized = PeopleSerializer(data=peopleJson)
        #verifica se conversão é válida!
        peopleSerialized.is_valid(raise_exception=True)
        #salva no banco de dados (insert into people ...)
        peopleSerialized.save()
        return Response(status=201, data=peopleSerialized.data)
```

PREPARANDO O DJANGO PARA FUNCIONAR COMO API COM APIVIEW – views.py

2. Método POST:

Requisição do POST no Insomnia, enviando os dados a serem cadastrados em JSON (lado esquerdo) e após serem recebidos pelo backend e cadastrados a API retorna a resposta com os mesmos dados já com o id gerado pelo banco de dados (lado direito)

Vale ressaltar que o status **201** retornado significa que o backend criou o registro com sucesso



PREPARANDO O DJANGO PARA FUNCIONAR COMO API COM APIVIEW – views.py

3. Método GET:

Neste código, temos duas opções:

- caso não seja fornecido o id na requisição, então a API irá retornar todos os dados da tabela usando **objects.all()**
- caso seja fornecido o id na requisição ele será usado para fazer a busca (neste caso é o 'planetId'), usando o **objects.get**

* Outro detalhe interessante é que neste código está sendo usado o **try** e **except** , que colocamos quando queremos tratar algum tipo de erro ou exceção, neste caso estamos usando caso seja informado um planetId que não exista, sendo redirecionado para o 'Except' nesses casos.

```
class PlanetAPIView(APIView):  
    def get(self, request, planetId = ''):  
        if planetId == '':  
            #primeiro vamos fazer um select all do banco:  
            planetFound = Planet.objects.all() #select *from Planet;  
            #agora pegamos os dados em python e mandamos p/ json  
            planetSerialized = PlanetSerializer(planetFound, many=True)  
            #manda a resposta para quem chamou a API:  
            return Response(planetSerialized.data)  
        else:  
            try:  
                planetFound = Planet.objects.get(id=planetId)  
                planetSerialized = PlanetSerializer(planetFound, many=False)  
                return Response(planetSerialized.data)  
            except Planet.DoesNotExist:  
                return Response(status=404, data='Planet Not Found!')
```



PREPARANDO O DJANGO PARA FUNCIONAR COMO API COM APIVIEW – views.py

3. Método GET:

Requisição coletando todos os dados:

GET `http://localhost:8000/people` Send 200 OK 53 ms 423 B Just Now

Body Auth Query Headers 1 Docs

URL PREVIEW

`http://localhost:8000/people`

Add Delete All Toggle Description

height	1.40			
eyeColor	Laranja			

```
1 [
2 {
3   "id": 1,
4   "name": "DJDDJGKKFKF",
5   "birthYear": "ABC1988",
6   "eyeColor": "Vermelho",
7   "gender": "Masculino",
8   "hairColor": "Azul",
9   "height": "1.50",
10  "mass": "68.00",
11  "created": "2023-08-03T01:17:39.680805Z",
12  "planet": 1,
13  "starship": null
14 },
15 {
16   "id": 2,
17   "name": "Aluno TOP Turma",
18   "birthYear": "ABC 78",
19   "eyeColor": "Vermelho",
20   "gender": "Ovni",
21   "hairColor": "Preto",
22   "height": "1.80",
23   "mass": "90.00",
24   "created": "2023-08-08T23:41:28.753366Z",
25   "planet": 1,
26   "starship": 1
27 }
```

Requisição coletando um id específico

GET `http://localhost:8000/people/2` Send 200 OK 10.5 ms 209 B Just Now

Body Auth Query Headers 1 Docs

URL PREVIEW

`http://localhost:8000/people/2`

Add Delete All Toggle Description

height	1.40			
eyeColor	Laranja			

```
1 {
2   "id": 2,
3   "name": "Aluno TOP Turma",
4   "birthYear": "ABC 78",
5   "eyeColor": "Vermelho",
6   "gender": "Ovni",
7   "hairColor": "Preto",
8   "height": "1.80",
9   "mass": "90.00",
10  "created": "2023-08-08T23:41:28.753366Z",
11  "planet": 1,
12  "starship": 1
13 }
```



PREPARANDO O DJANGO PARA FUNCIONAR COMO API COM APIVIEW – views.py

3. Método GET:

Ainda no método GET podemos usá-lo quando queremos filtrar por mais dados:

```
def get(self, request, peopleId = ''):

    if peopleId == '': #se estiver vazio, pega tudo!

        peopleFound = ''

        #se receber filtro de ambos:
        if 'height' in request.GET and 'eyeColor' in request.GET:
            peopleFound = People.objects.filter(height__gt=request.GET['height']) | People.objects.filter(eyeColor__contains=request.GET['eyeColor'])
            # peopleFound = People.objects.filter(height__gt=request.GET['height']).filter(eyeColor__contains=request.GET['eyeColor'])
        elif 'eyeColor' in request.GET:
            peopleFound = People.objects.filter(eyeColor__contains=request.GET['eyeColor'])
        elif 'height' in request.GET:
            peopleFound = People.objects.filter(height__gt=request.GET['height'])
        else:
            #primeiro vamos fazer um select all do banco:
            peopleFound = People.objects.all() #select *from people;
            #agora pegamos os dados em python e mandamos p/ json
            peopleSerialized = PeopleSerializer(peopleFound, many=True)
            #manda a resposta para quem chamou a API:
            #Response(data="ok")
            return Response(peopleSerialized.data)
    else: #coletando people do id solicitado!
        try:
            peopleFound = People.objects.get(id=peopleId)
            #select *from people where id = peopleId
            peopleSerialized = PeopleSerializer(peopleFound, many=False)
            return Response(peopleSerialized.data)
        except People.DoesNotExist:
            return Response(status=404, data='People Not Found!')
```

Neste caso estamos filtrando pelas colunas 'height' ou 'eyeColor', que podem ser enviadas juntas ou individualmente

PREPARANDO O DJANGO PARA FUNCIONAR COMO API COM APIVIEW – views.py

3. Método GET:

Requisição do Insomnia enviando via parâmetros os dois parâmetros configurados

The screenshot shows the Insomnia REST client interface. The request is a GET to `http://localhost:8000/people` with query parameters `height=1.00` and `eyeColor=Preto`. The response is a 200 OK status with a JSON body containing two objects.

URL PREVIEW

`http://localhost:8000/people?height=1.00&eyeColor=Preto`

Query Parameters

Parameter	Value	Required	Delete
height	1.00	<input checked="" type="checkbox"/>	<input type="button" value="Delete"/>
eyeColor	Preto	<input checked="" type="checkbox"/>	<input type="button" value="Delete"/>

Response Body (JSON)

```
[
  {
    "id": 1,
    "name": "DJDJGKKFKF",
    "birthYear": "ABC1988",
    "eyeColor": "Vermelho",
    "gender": "Masculino",
    "hairColor": "Azul",
    "height": "1.50",
    "mass": "68.00",
    "created": "2023-08-03T01:17:39.680805Z",
    "planet": 1,
    "starship": 1
  },
  {
    "id": 2,
    "name": "Aluno TOP Turma A",
    "birthYear": "ABC 78",
    "eyeColor": "Vermelho",
    "gender": "Ovni",
    "hairColor": "Preto",
    "height": "1.80",
    "mass": "90.00",
    "created": "2023-08-03T01:17:39.680805Z",
    "planet": 1,
    "starship": 1
  }
]
```


PREPARANDO O DJANGO PARA FUNCIONAR COMO API COM APIVIEW – views.py

4. Método PUT:

O método PUT serve para alterar um registro já existente, neste caso temos que fazer dois passos: coletar o dado já existente no banco de dados com **objects.get**, e coletar os dados vindos em JSON por meio do **request.data**

Após isso, enviamos os dois dados, antigo e novo, para o serializer.

Caso o serializer verifique que os dados são válidos, salvamos no banco de dados.

```
class PeopleAPIView(APIView):
    def put(self, request, peopleId = ''):
        peopleFound = None

        try:
            #o people já existente no banco:
            peopleFound = People.objects.get(id=peopleId)
        except People.DoesNotExist:
            return Response(status=404, data="People not Found!")

        #o people com os dados novos
        peopleJson = request.data #coletando o json que veio do cliente
        #update
        peopleSerialized = PeopleSerializer(peopleFound, data=peopleJson)
        peopleSerialized.is_valid(raise_exception=True)
        peopleSerialized.save()
        return Response(status=200, data=peopleSerialized.data)
```

PREPARANDO O DJANGO PARA FUNCIONAR COMO API COM APIVIEW – views.py

4. Método PUT:

Requisição alterando as informações cadastradas

The screenshot displays a web browser interface for a REST client. The top bar shows the method **PUT** and the URL `http://localhost:8000/people/1`. The **Send** button is highlighted, and the response status is **200 OK** with a response time of **744 ms** and a body size of **208 B**. The **JSON** tab is selected, showing the request body as a JSON object with the following fields: `"name": "DJDJJGKKFKF", "birthYear": "ABC1988", "eyeColor": "Vermelho", "gender": "Masculino", "hairColor": "Azul", "height": "1.50", "mass": "68.00", "planet": 1, "starship": 1`. The **Preview** tab on the right shows the response body, which is a JSON object with the following fields: `"id": 1, "name": "DJDJJGKKFKF", "birthYear": "ABC1988", "eyeColor": "Vermelho", "gender": "Masculino", "hairColor": "Azul", "height": "1.50", "mass": "68.00", "created": "2023-08-03T01:17:39.680805Z", "planet": 1, "starship": 1`.

```
1 {
2   "name": "DJDJJGKKFKF",
3   "birthYear": "ABC1988",
4   "eyeColor": "Vermelho",
5   "gender": "Masculino",
6   "hairColor": "Azul",
7   "height": "1.50",
8   "mass": "68.00",
9   "planet": 1,
10  "starship": 1
11 }
```

```
1 {
2   "id": 1,
3   "name": "DJDJJGKKFKF",
4   "birthYear": "ABC1988",
5   "eyeColor": "Vermelho",
6   "gender": "Masculino",
7   "hairColor": "Azul",
8   "height": "1.50",
9   "mass": "68.00",
10  "created": "2023-08-03T01:17:39.680805Z",
11  "planet": 1,
12  "starship": 1
13 }
```

PREPARANDO O DJANGO PARA FUNCIONAR COMO API COM APIVIEW – views.py

5. Método DELETE:

Para deletar um registro, fazemos a busca dele no banco de dados usando o id que foi recebido (peopleId) e, se o registro existir procedemos com a deleção.

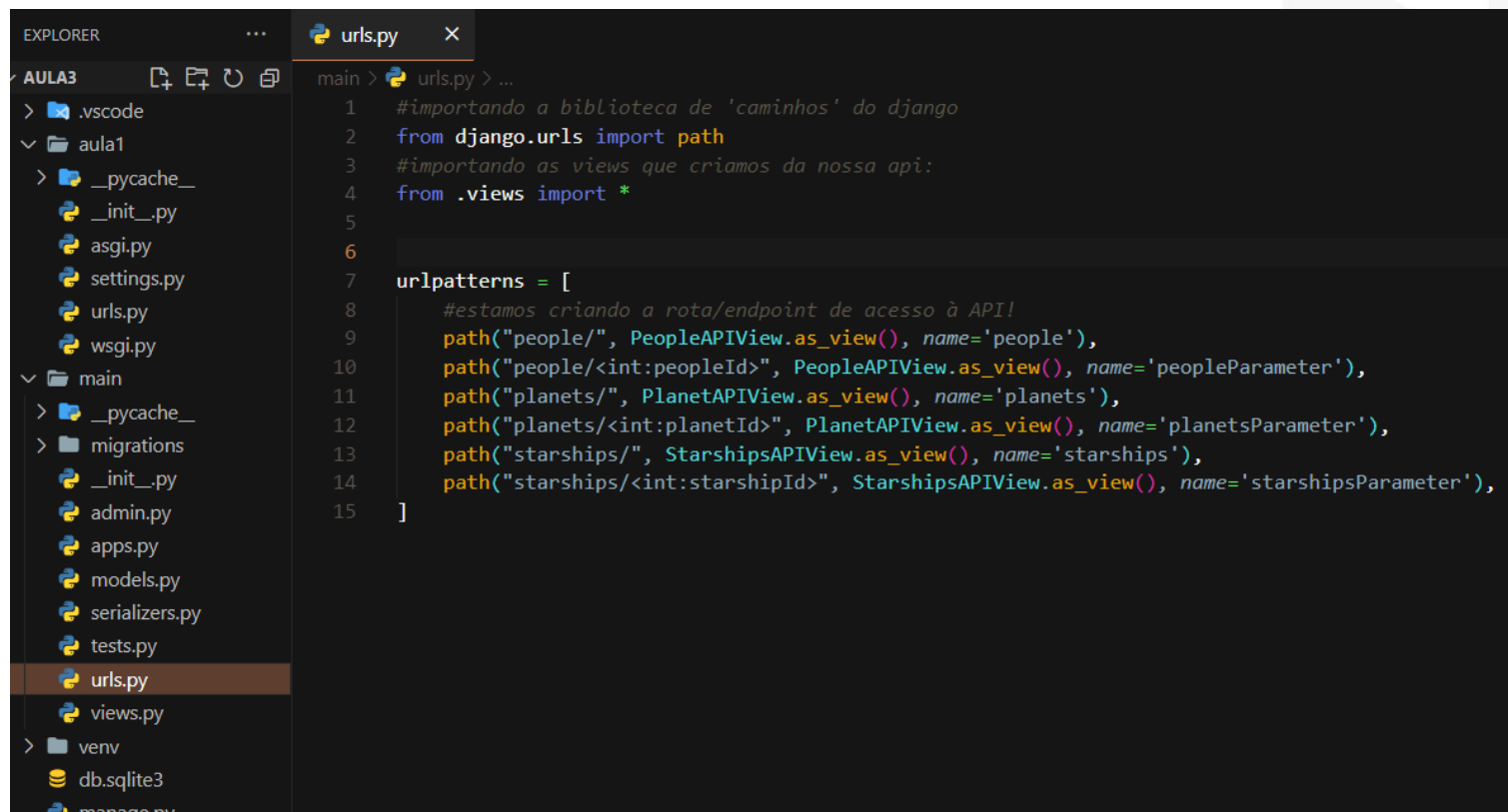
```
class PeopleAPIView(APIView):  
    def delete(self, request, peopleId = ''):  
        peopleFound = None  
        try:  
            #busca o usuario com o id  
            peopleFound = People.objects.get(id=peopleId)  
        except People.DoesNotExist:  
            return Response(status=404, data="People not Found!")  
  
        peopleFound.delete() #deleta o usuario com o id encontrado  
        return Response(status=200, data="People successfully deleted!")
```

PREPARANDO O DJANGO PARA FUNCIONAR COMO API COM APIVIEW – urls.py

1. Criação das URLs / ROTAS da API:

Crie o arquivo urls.py dentro do seu aplicativo

Neste caso, estamos configurando as urls para receberem também os filtros por id



```
EXPLORER
...
aula3
  > .vscode
  > aula1
    > __pycache__
    __init__.py
    asgi.py
    settings.py
    urls.py
    wsgi.py
  > main
    > __pycache__
    > migrations
    __init__.py
    admin.py
    apps.py
    models.py
    serializers.py
    tests.py
    urls.py
    views.py
  > venv
    db.sqlite3
    manage.py

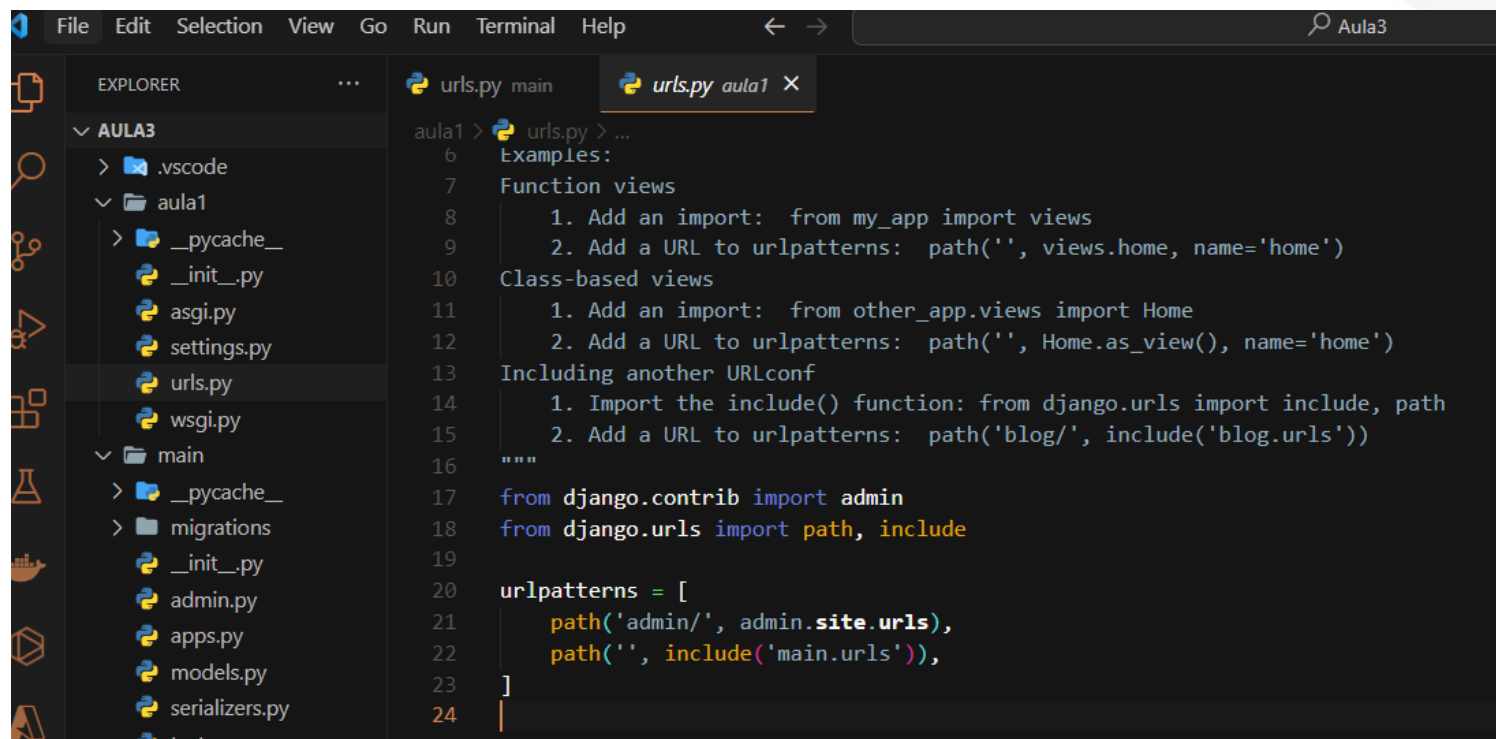
main > urls.py > ...
1  #importando a biblioteca de 'caminhos' do django
2  from django.urls import path
3  #importando as views que criamos da nossa api:
4  from .views import *
5
6
7  urlpatterns = [
8      #estamos criando a rota/endpoint de acesso à API!
9      path("people/", PeopleAPIView.as_view(), name='people'),
10     path("people/<int:peopleId>", PeopleAPIView.as_view(), name='peopleParameter'),
11     path("planets/", PlanetAPIView.as_view(), name='planets'),
12     path("planets/<int:planetId>", PlanetAPIView.as_view(), name='planetsParameter'),
13     path("starships/", StarshipsAPIView.as_view(), name='starships'),
14     path("starships/<int:starshipId>", StarshipsAPIView.as_view(), name='starshipsParameter'),
15 ]
```

PREPARANDO O DJANGO PARA FUNCIONAR COMO API COM APIVIEW – urls.py

2. Registrar as rotas da API nas urls do projeto:

Crie o arquivo urls.py dentro do seu aplicativo

Lembre-se que dentro do **include** é necessário usar o nome do seu aplicativo, neste caso o nome é 'main', porém deverá usar o nome do aplicativo que criou.



The screenshot shows a VS Code editor with the Django project structure in the Explorer on the left and the contents of the `urls.py` file in the main editor. The Explorer shows the project structure with folders `.vscode`, `aula1`, `main`, and `migrations`. The `main` folder contains `__init__.py`, `admin.py`, `apps.py`, `models.py`, `serializers.py`, and `tests.py`. The `urls.py` file in the `main` folder is selected. The main editor shows the contents of `urls.py` with the following code:

```
6 Examples:
7 Function views
8     1. Add an import: from my_app import views
9     2. Add a URL to urlpatterns: path('', views.home, name='home')
10 Class-based views
11     1. Add an import: from other_app.views import Home
12     2. Add a URL to urlpatterns: path('', Home.as_view(), name='home')
13 Including another URLconf
14     1. Import the include() function: from django.urls import include, path
15     2. Add a URL to urlpatterns: path('blog/', include('blog.urls'))
16 """
17 from django.contrib import admin
18 from django.urls import path, include
19
20 urlpatterns = [
21     path('admin/', admin.site.urls),
22     path('', include('main.urls')),
23 ]
24
```