Criando uma REST API com Django

Space Devs - Janeiro de 2020

Camila Maia

Esta apresentação está disponível em:

github.com/camilamaia/talks/tree/master/2020

Público

Já se sente confortável desenvolvendo em Python

Possui ou não conhecimento em Django

Não possui domínio do Django REST Framework

Deseja construir APIs

API?

Application Programming Interface

“É um conjunto de funções estabelecidos por um software para a utilização das suas funcionalidades por aplicativos que não pretendem envolver-se em detalhes da implementação do software, mas apenas usar seus serviços.”

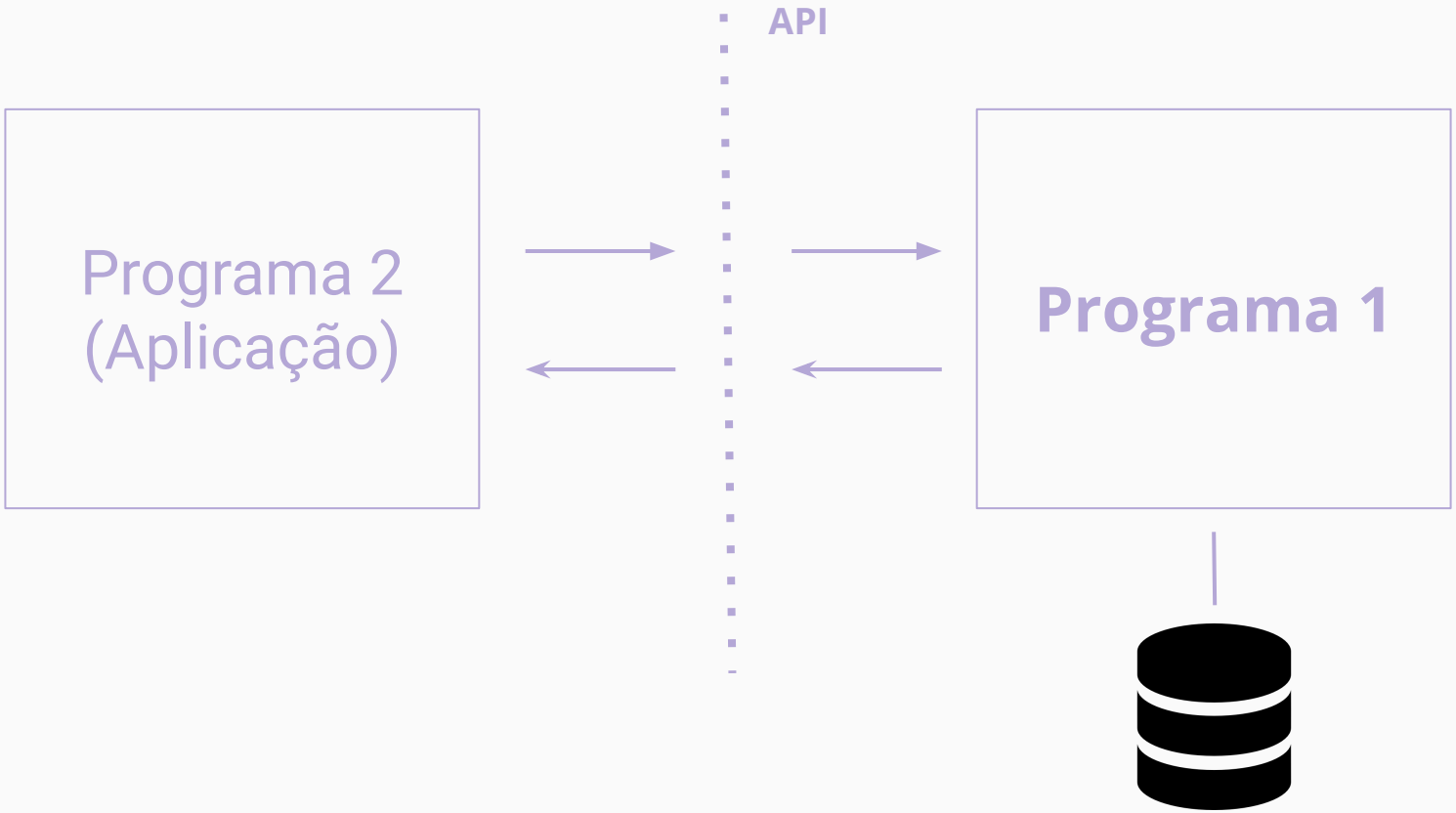
É um **conjunto de funções** estabelecidos por um software

para a **utilização** das suas funcionalidades **por aplicativos**

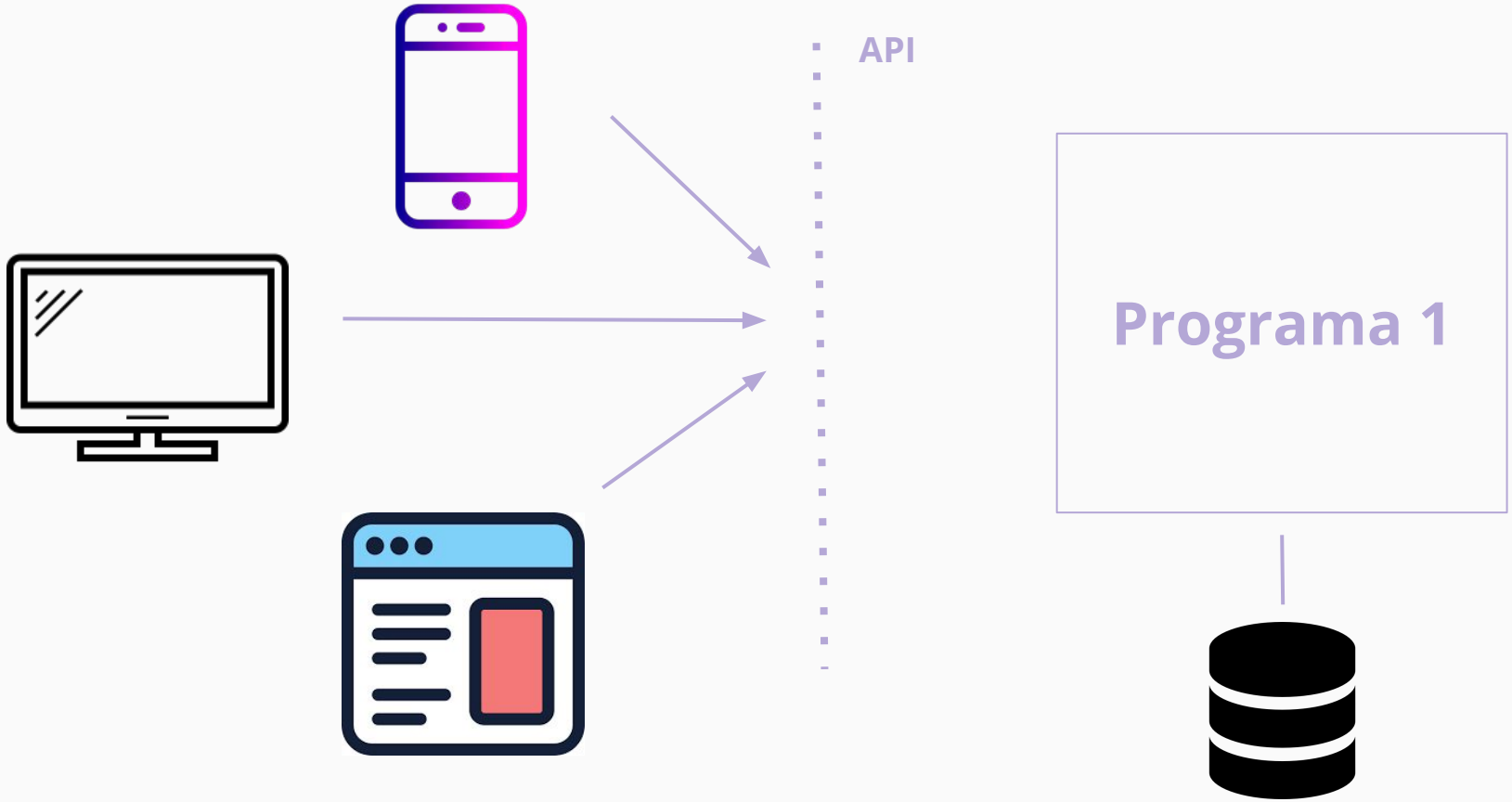
que não pretendem envolver-se em detalhes da implementação do software

mas apenas **usar seus serviços**.

API: Uma interface de comunicação entre dois programas



API: Uma interface de comunicação entre dois programas



API: Uma interface de comunicação entre dois programas

Site de Hotel

Empresa de Logística

Site de delivery

Aplicativo de Carro Particular

API

$$$$

Documentação API Google Maps

API

Serviço

Para utilizar esse serviço, você precisa entender como ele funciona

Documentação

Menu de um restaurante

Por que?

E o quê tem a ver com a economia?

- API é um produto
- Fácil de integrar
- Viabilização de parcerias

“As APIs do Twitter têm facilmente dez vezes mais tráfego do que o site do Twitter.”

APIs THE BUILDING BLOCKS OF THE APP ECONOMY

APIs FOR DUMMIES - IBM

Mais Exemplos

- Twitter API
- Facebook API
- Youtube API
  - github.com/youtube/api-samples

- github.com/toddmotto/public-apis
- blog.rapidapi.com/most-popular-apis/

Métodos HTTP

GET: Retorna recursos do servidor. Não altera o servidor

POST / PUT: Criar e editar recursos no servidor

DELETE: Deletar recursos do servidor

REST APIs

- Design de APIs para web
- É um estilo de arquitetura
- Série de padrões / restrições para construir uma API

Sua API não é RESTful: Entenda por quê.

10 Best Practices for Better RESTful API

Restful API Designing Guidelines

Representação dos Dados

- JSON
- XML
- ...

JSON

- JavaScript Object Notation
- Formato utilizado pelo padrão REST
- Modelo de dados
- O formato JSON é sintaticamente idêntico ao código para criar objetos JavaScript.

Especificação Oficial
JSON em Python

```
{
        "id": "0000234567894",
        "name": "Jane Doe",
        "birthday": "04/18/1978",
        "gender": "female",
        "type": "user",
        "work": [{
           "employer": {
              "id": "106119876543210",
              "name": "Doe Inc."
           },
           "start_date": "2007 - 08"
         },
        {
          "start_date": "2004",
          "end_date": "2007"
        }]
}
```

Exemplo de JSON


django
REST
framework

Django REST Framework (DRF)

Django: Web Framework para Python

DRF permite que uma aplicação Django se comporte com uma REST API.

Django REST Framework é um kit de ferramentas poderoso e flexível para criar Web APIS.

Por que Django / DRF?

- Rápido de fazer uma aplicação
- Possui várias ferramentas
- Versátil
- Escalável

Links:

- Quick Start
- Tutorial
- Web browsable API

Criando um Projeto

```
$ mkdir <project_name>
$ cd <project_name>
$ python -m venv env
$ source env/bin/activate
$ pip install django
$ pip install djangorestframework
```

Criando um Projeto

```
$ django-admin startproject <project_name> .
$ cd <project_name>
$ django-admin startapp <app_name>
$ cd ..
$ python manage.py migrate
```

settings.py

```
INSTALLED_APPS = [
    ...
    'rest_framework',
]
```

```
REST_FRAMEWORK = {
    'DEFAULT_PAGINATION_CLASS': 'rest_framework.pagination.PageNumberPagination',
    'PAGE_SIZE': 10
}
```

Arquitetura

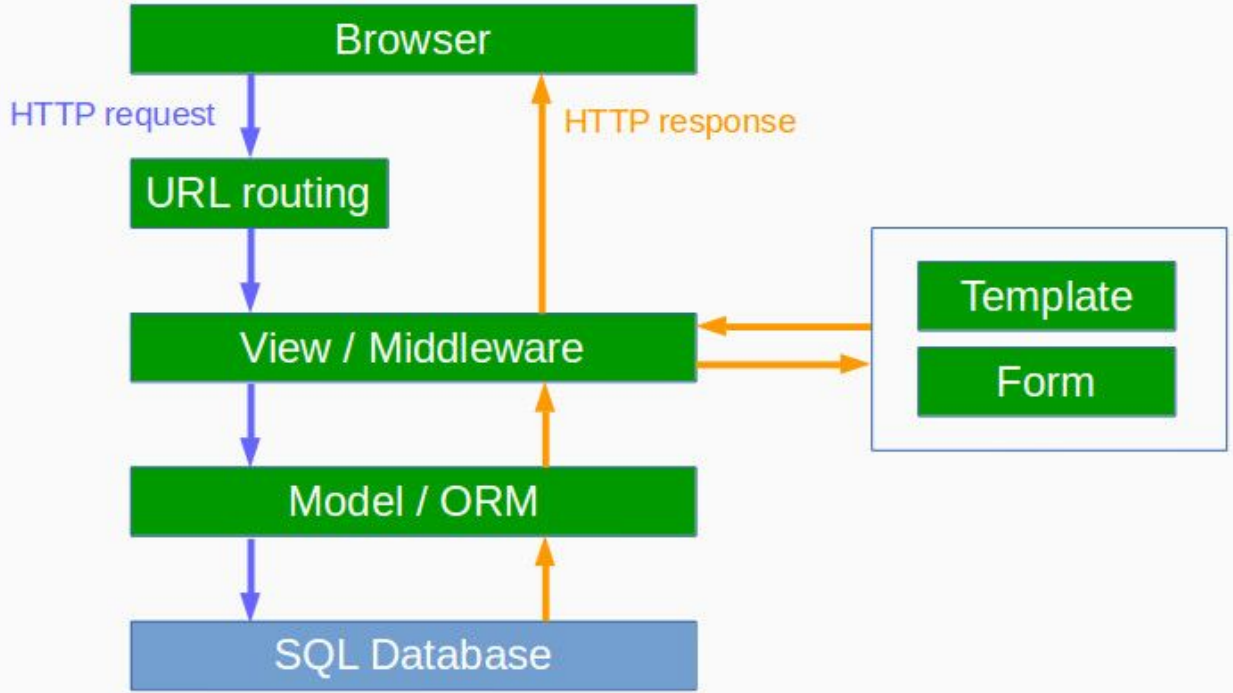
Arquitetura: Model, Serializer e View (MSV?)

View: Descreve qual dado será apresentado

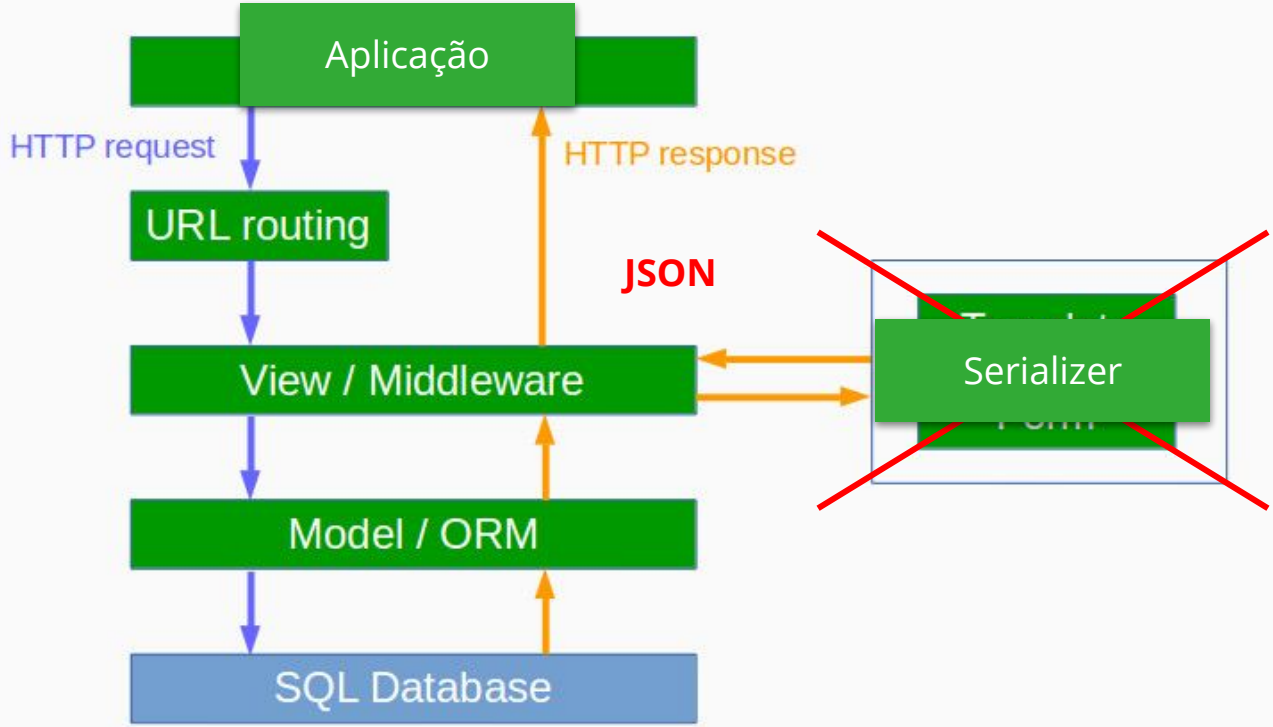
Serializer: Descreve como o dado será apresentado (JSON)

Model: Descreve as entidades da lógica do seu problema

Django's architecture



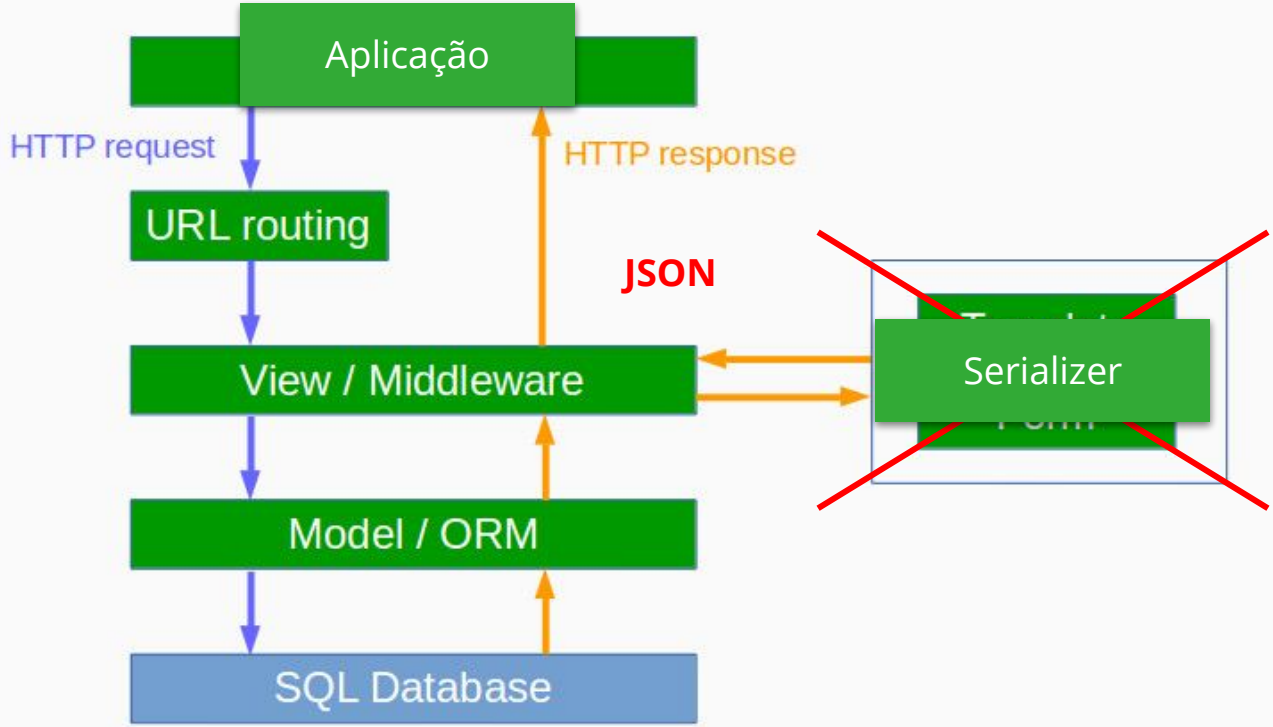
DRF 's architecture



Roteamento das URL

- Rotas do Programa
- Arquivos urls.py

DRF 's architecture



View

Descreve **qual** dados serão apresentados.

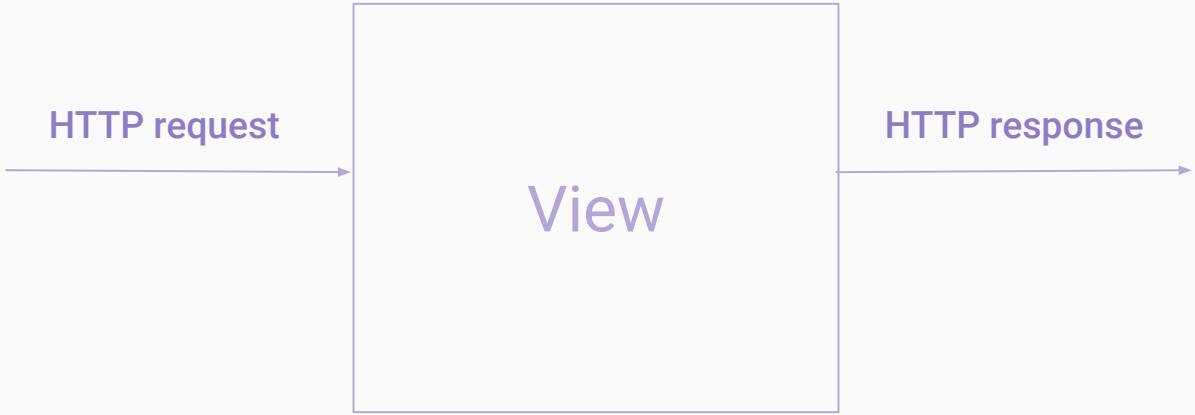
Uma View é simplesmente uma **função** Python que recebe uma **request** e retorna uma **response**.

Essa resposta pode ser um conteúdo HTML de uma página Web, um redirecionamento, um erro 404, um documento XML, uma imagem...

A visualização em si contém qualquer lógica arbitrária necessária para retornar essa resposta.

Como estamos trabalhando com REST APIs, o conteúdo da resposta HTTP deve ser em JSON.

View



Function Based Views (FBV)

```
url(r'^(?P<question_id>[0-9]+)/vote/$',
    views.vote,
    name='vote'
),
```

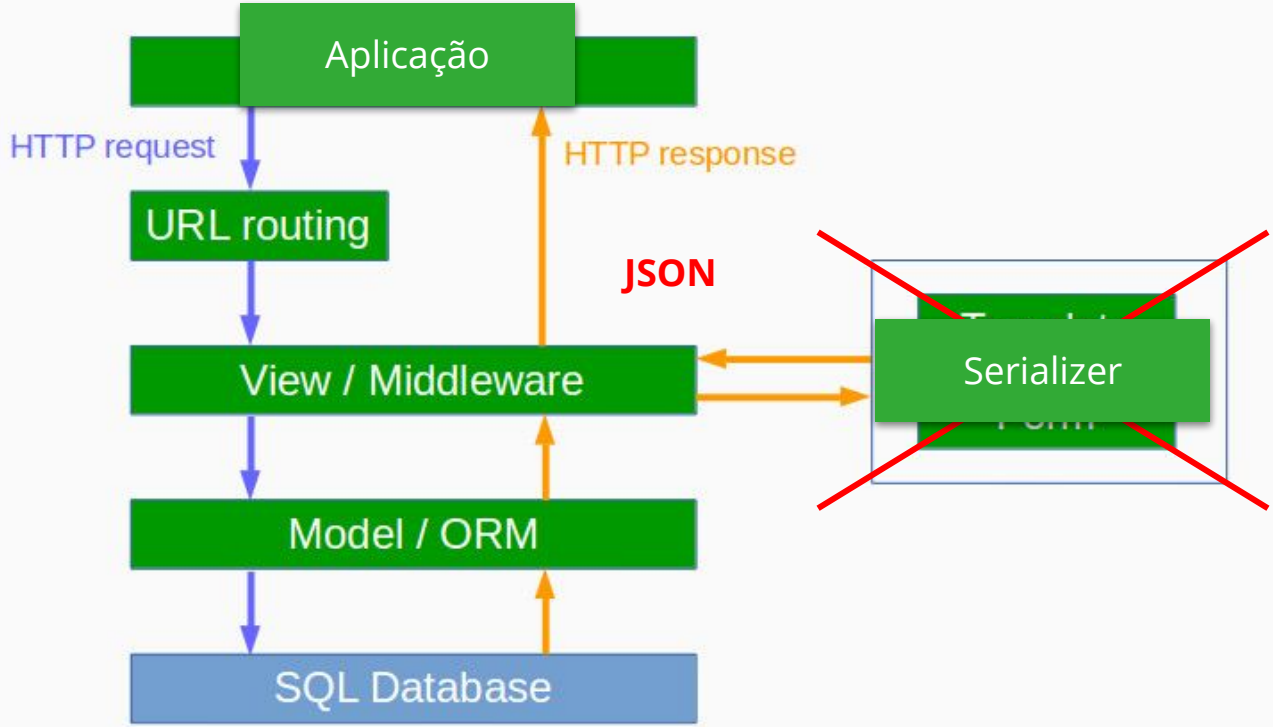
```
def vote(request, question_id):
    if request.method == 'GET':
        pass
    if request.method == 'POST':
        pass
    if request.method == 'PUT':
        pass
    if request.method == 'DELETE':
        pass
```

Sempre retornando uma HTTP Response

Rodando

$ python manage.py runserver

DRF 's architecture

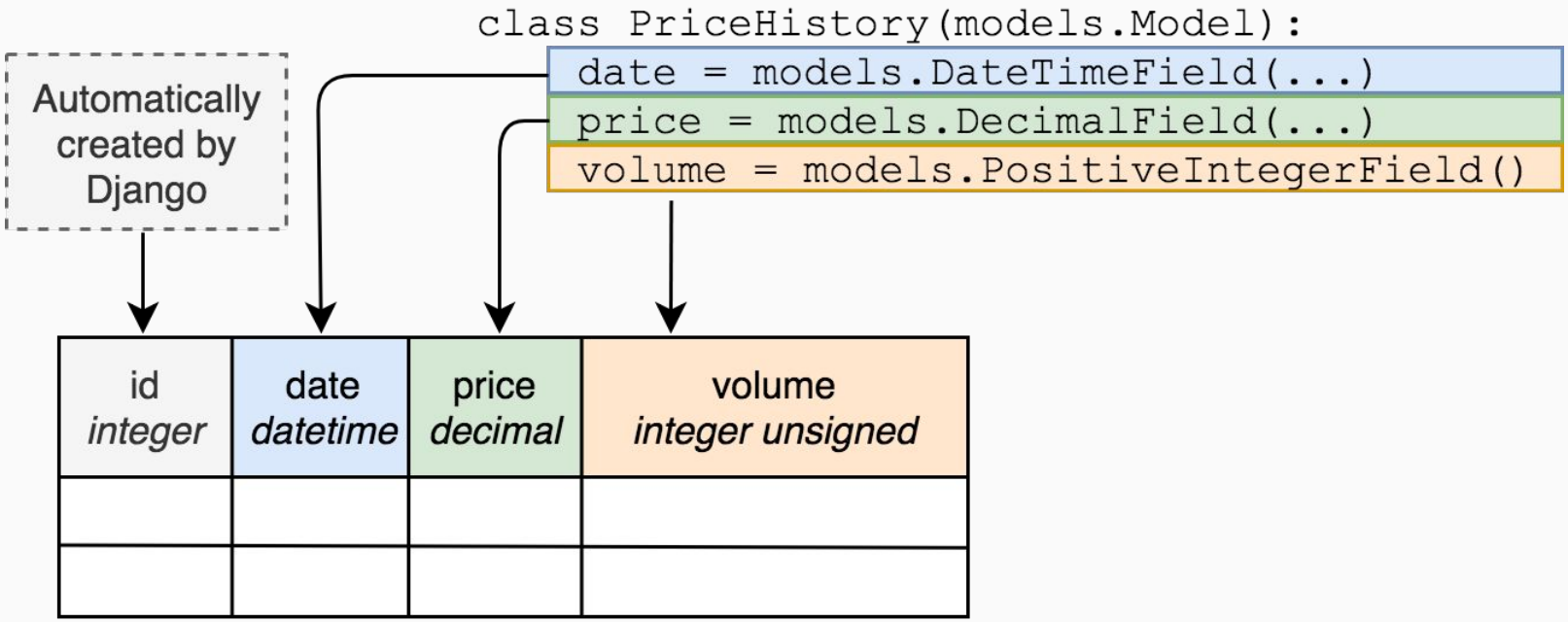


Model

Descreve as entidades lógicas do seu problema.

Podem ou não ser uma tabela no banco de dados

Model


class PriceHistory(models.Model):
date = models.DateTimeField(...)
price = models.DecimalField(...)
volume = models.PositiveIntegerField()
Automatically created by Django
id
integer
date
datetime
price
decimal
volume
integer unsigned

Migrações

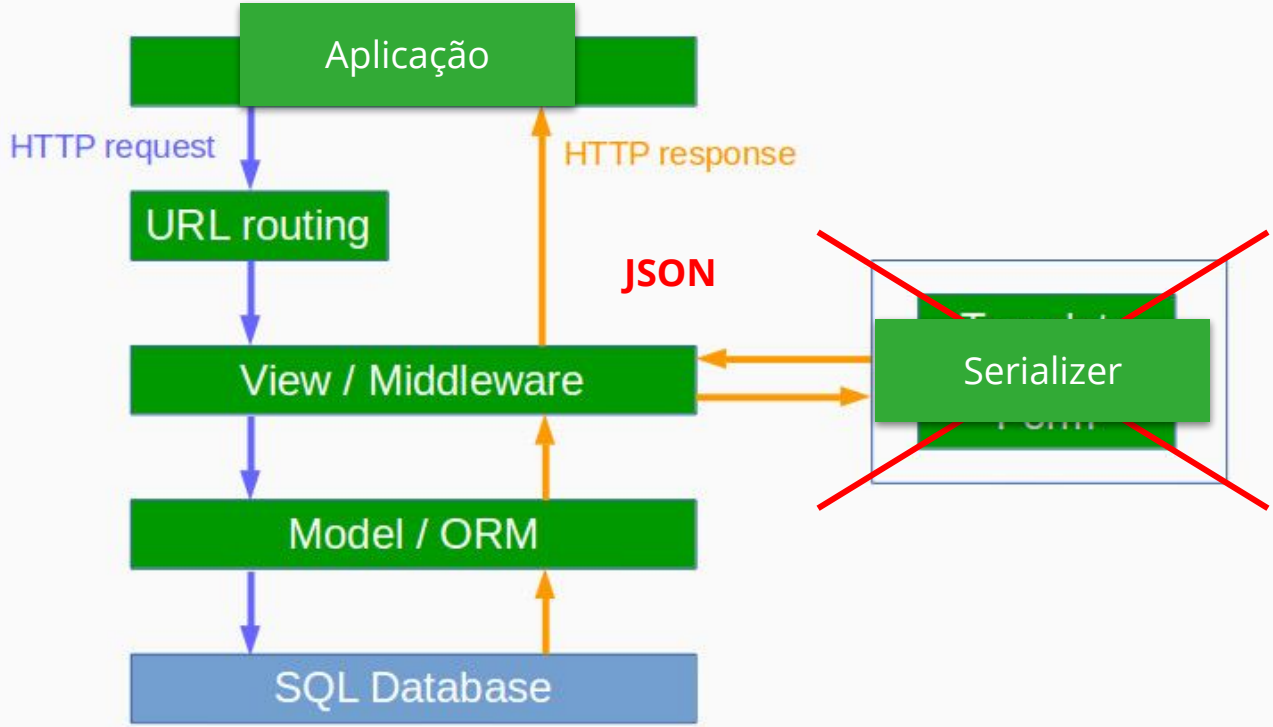
Passo 1: Mudar o seu modelo (arquivo models.py)

Passo 2: Rodar **$ python manage.py makemigrations** para gerar os arquivos de migração

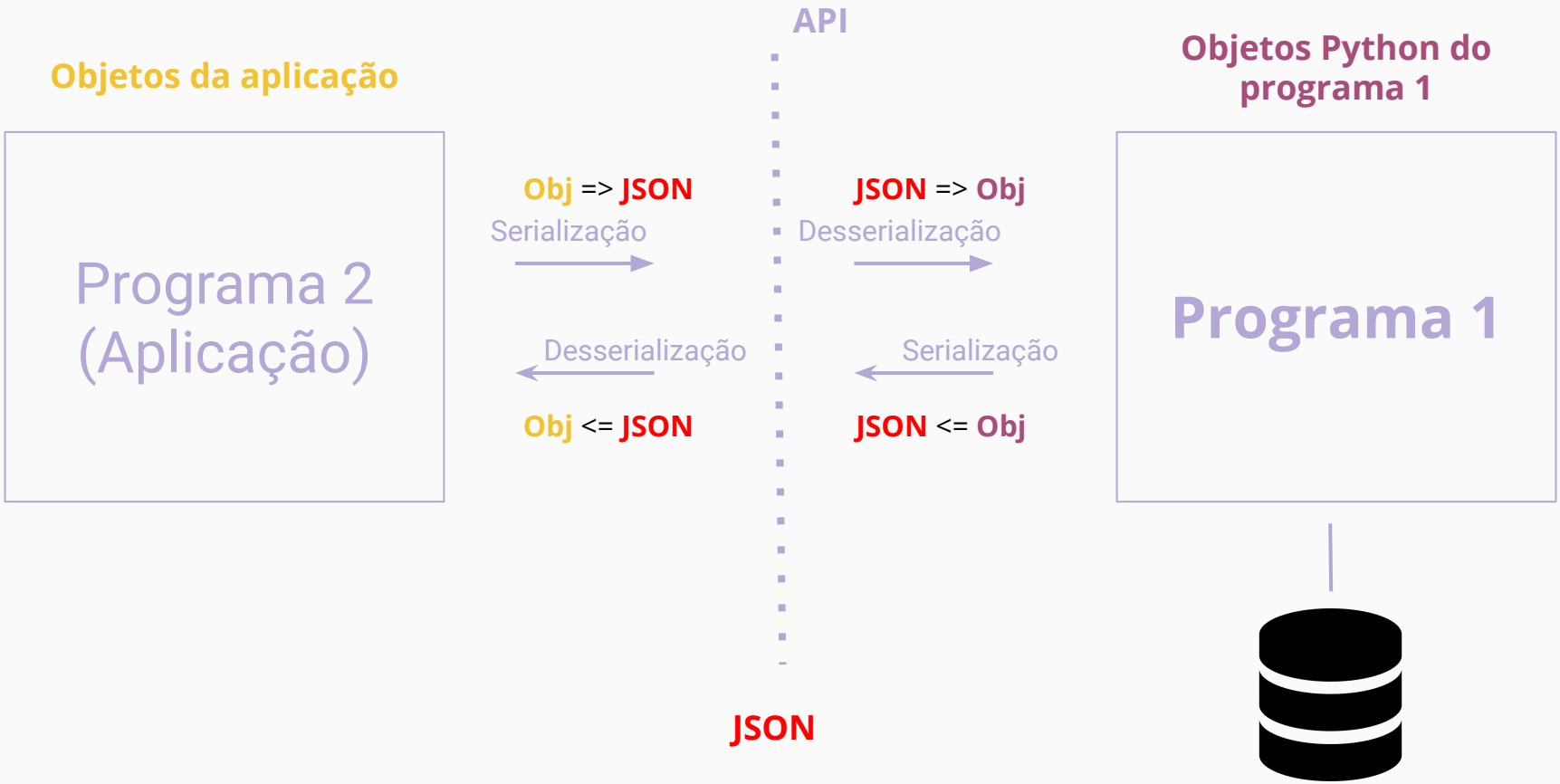
Passo 3: Rodar **$ python manage.py migrate** para aplicar as mudanças no banco de dados

Django Shell **$ python manage.py shell**

DRF 's architecture



Serialização



Class Based Views

Class Based Views (CBV)

Class Based Views fornecem uma maneira **alternativa** de implementar Views com objetos Python, ao invés de funções.

Não substitui as function based views

Reutilizar código através de:

- Heranças
- Mixins

Evita fazer um branch de if’s

Class Based Views (CBV)

Classy Django REST Framework

IDE que possa navegar nas classes do Django e do DRF

Generics, Mixins, ViewSets...

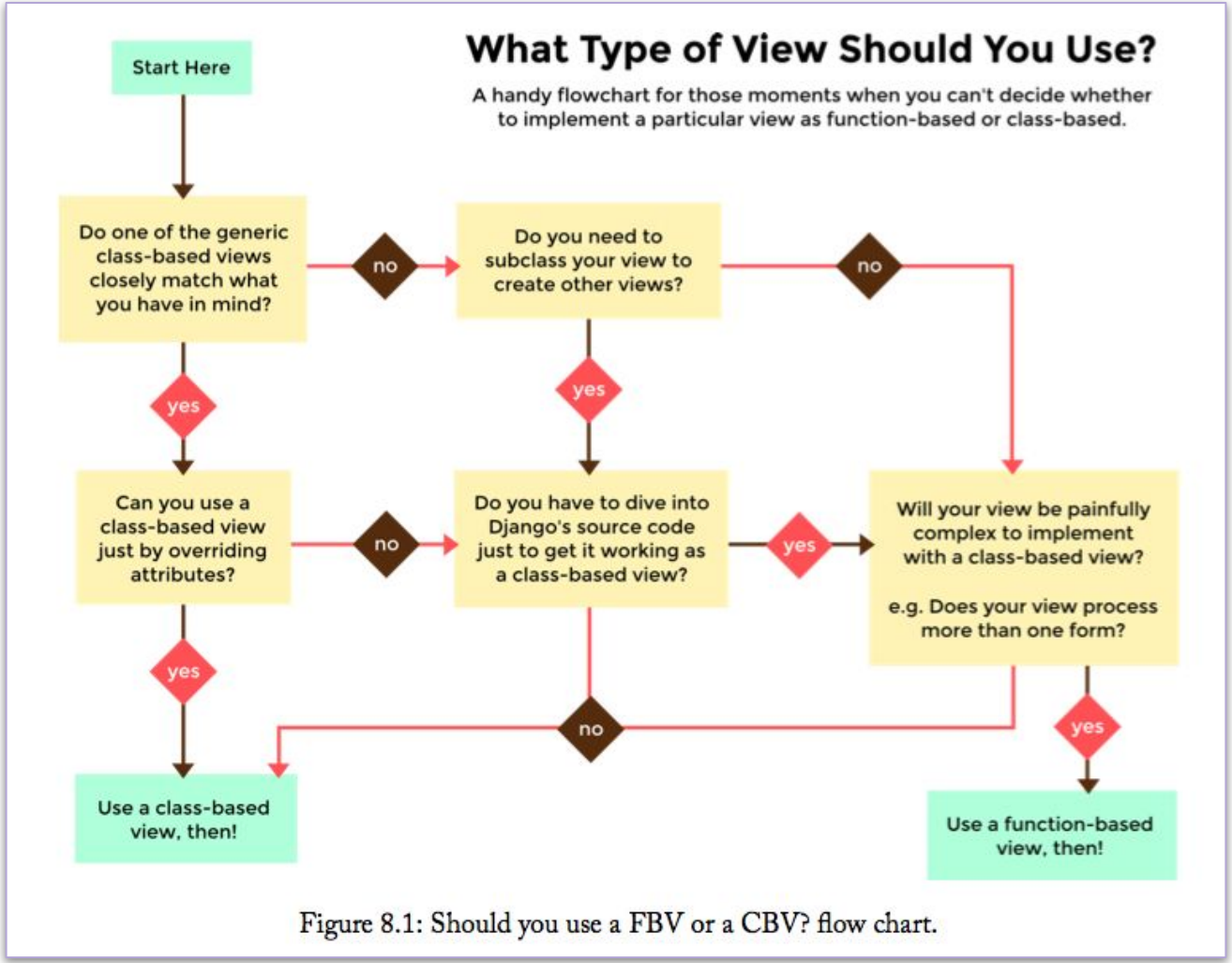


Figure 8.1: Should you use a FBV or a CBV? flow chart.

https://www.twoscoopspress.com/products/two-scoops-of-django-1-11

Django Admin

Django fornece automaticamente uma área de administração


```
$ python manage.py createsuperuser --email admin@example.com --username admin
```

Conteúdos

https://www.nytimes.com/paidpost/ca-technologies/apis-the-building-blocks-of-the-app-economy.html

https://www.ibm.com/downloads/cas/GJ5QVQ7X

https://developer.twitter.com/en/docs.html

https://developers.facebook.com/?locale=pt_BR

https://developers.google.com/youtube/v3/

https://github.com/youtube/api-samples

https://github.com/toddmotto/public-apis

https://blog.rapidapi.com/most-popular-apis/

https://blog.geekhunter.com.br/sua-api-nao-e-restful-entenda-por-que/

https://medium.com/@mwaysolutions/10-best-practices-for-better-restful-api-cbe81b06f291

Conteúdos

https://hackernoon.com/restful-api-designing-guidelines-the-best-practices-60e1d954e7c9

https://www.json.org

https://docs.python.org/3/library/json.html

https://www.django-rest-framework.org/tutorial/quickstart/

https://www.django-rest-framework.org/tutorial/1-serialization/

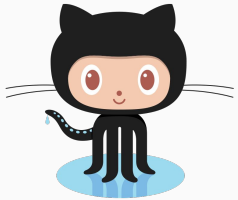
https://restframework.herokuapp.com

http://www.cdrf.co/

https://ccbv.co.uk

MUITO OBRIGADA!

@cmaiacd



camilamaia

Models e Forms

- Null -- Banco de Dados
  - Default is False
- Blank -- Forms
  - Default is False

```
class Question(models.Model):
    question_text = models.CharField(
        max_length=200,
        null=True,
        blank=True
    )
```

null=True
blank=True

Field Type	Setting null=True	Setting blank=True
`CharField`, `TextField`, `SlugField`, `EmailField`, `CommaSeparatedInteger-Field`, `UUIDField`	*Okay* if you also have set both `unique=True` and `blank=True`. In this situation, `null=True` is required to avoid unique constraint violations when saving multiple objects with blank values.	*Okay* if you want the corresponding form widget to accept empty values. If you set this, empty values are stored as `NULL` in the database if `null=True` and `unique=True` are also set. Otherwise, they get stored as empty strings.
`FileField`, `ImageField`	*Don't do this.* Django stores the path from `MEDIA_ROOT` to the file or to the image in a CharField, so the same pattern applies to FileFields.	*Okay.* The same pattern for CharField applies here.

null=True
blank=True

Field Type	Setting null=True	Setting blank=True
`BooleanField`	*Don't do this.* Use `NullBooleanField` instead.	*Don't do this.*
`IntegerField`, `FloatField`, `DecimalField`, `DurationField`, etc	*Okay* if you want to be able to set the value to `NULL` in the database.	*Okay* if you want the corresponding form widget to accept empty values. If so, you will also want to set `null=True`.
`DateTimeField`, `DateField`, `TimeField`, etc.	*Okay* if you want to be able to set the value to `NULL` in the database.	*Okay* if you want the corresponding form widget to accept empty values, or if you are using auto_now or auto_now_add. If it's the former, you will also want to set `null=True`.
`ForeignKey`, `ManyToManyField`, `OneToOneField`	*Okay* if you want to be able to set the value to `NULL` in the database.	*Okay* if you want the corresponding form widget (e.g. the select box) to accept empty values. If so, you will also want to set `null=True`.
`GenericIPAddressField`	*Okay* if you want to be able to set the value to `NULL` in the database.	*Okay* if you want to make the corresponding field widget accept empty values. If so, you will also want to set `null=True`.