



# FRAMEWORKS PARA DESENVOLVIMENTO WEB

Professor Paulo Honório  
Segundo Dia - 17/02/2024

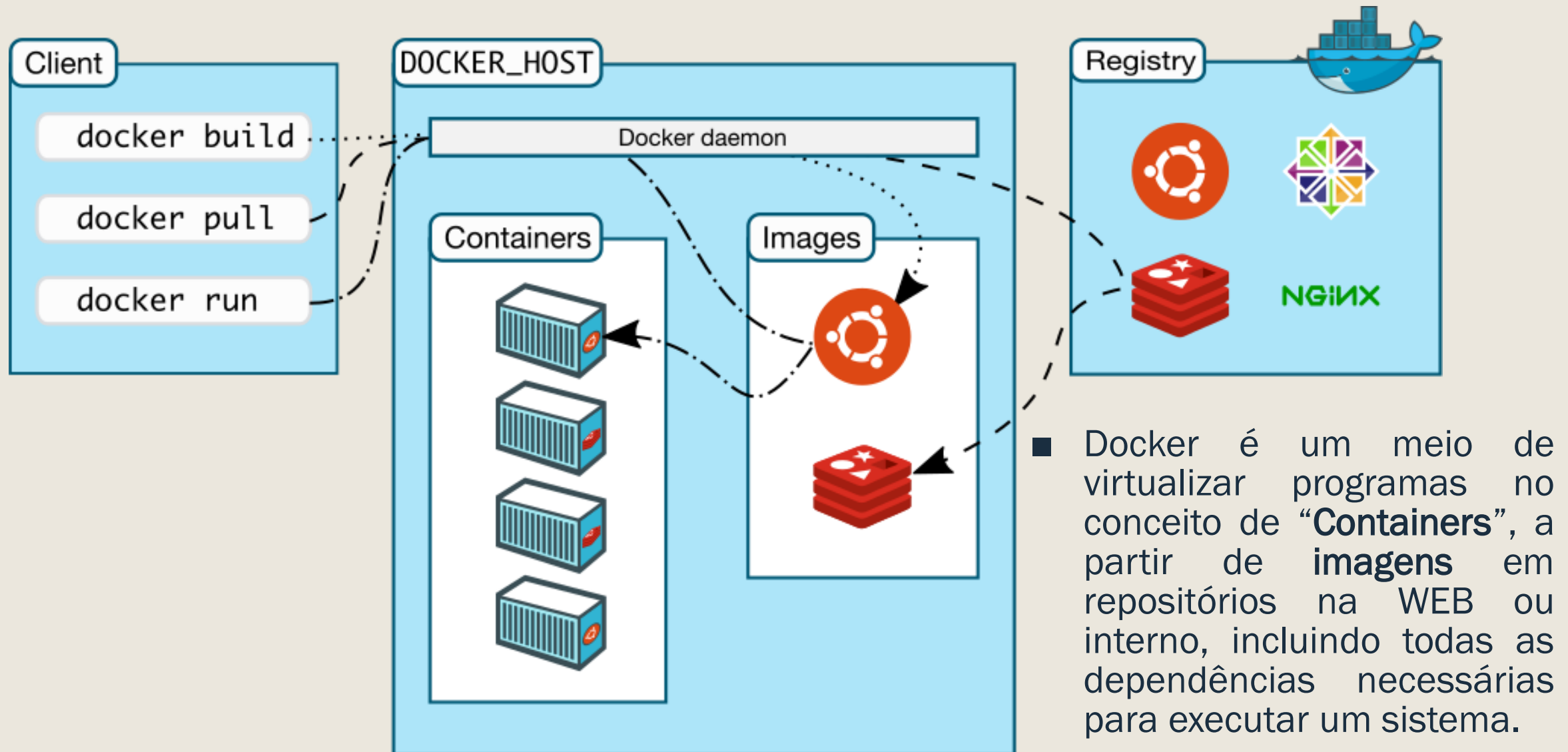
# Agenda

- Revisão
- Container Docker
- Gunicorn & Nginx
- Atividade 4: Construir um Container
- Aplicar arquivos estáticos
- Dependências do frontend
- Estrutura de Template com *Bootstrap* e HTML
- Eventos *JQuery*, *Ajax*
- Atividade 5: Aplicar template Bootstrap
- Django Administrator
- Permissões de acesso Django
- Gerando *Migrations* e *Seed* de para ambiente de homologação ou produção
- Banco de dados relacional PostgreSQL
- Atividade 6: Trocar SGBD para PostgreSQL e aplicar ACL
- Desativar a depuração
- Customizar páginas de Erro
- Construindo uma aplicação REST
- Atividade 7: Gerar path REST com acesso restrito
- Questionário

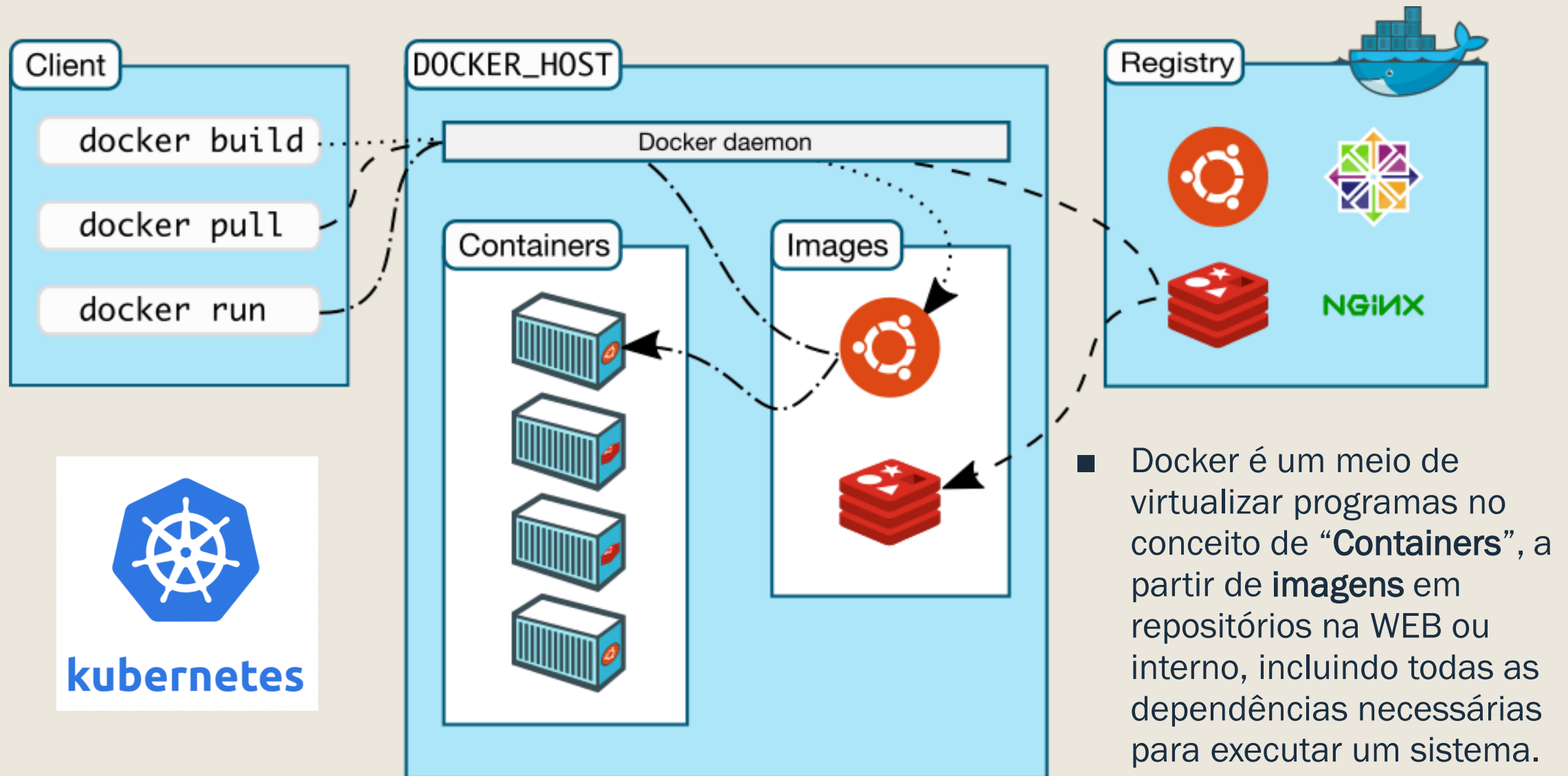
# Revisando a Estrutura do Django

```
i_ HelloWorld
  | db.sqlite3
  | HelloWorld
  |   | __init__.py
  |   | __init__.pyc
  |   | settings.py
  |   | settings.pyc
  |   | urls.py
  |   | urls.pyc
  |   | wsgi.py
  |   | wsgi.pyc
  |   | HelloWorldApp
  |   |   | admin.py
  |   |   | admin.pyc
  |   |   | __init__.py
  |   |   | __init__.pyc
  |   |   | models.py
  |   |   | models.pyc
  |   |   | templates
  |   |   |   | helloDJ
  |   |   |   |   | base.html
  |   |   |   |   | home.html
  |   |   | tests.py
  |   |   | views.py
  |   |   | views.pyc
  |   | manage.py
```

# Container Docker



# Container Docker

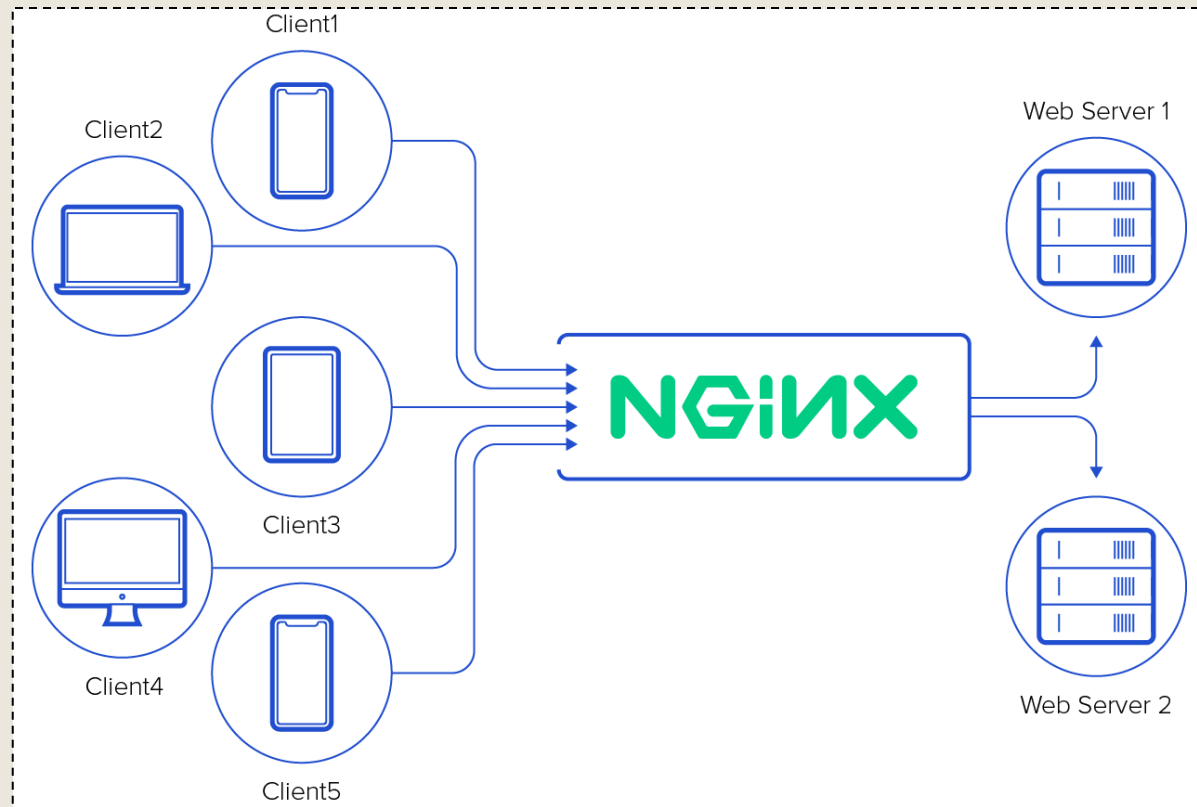


# Gunicorn ou "Green Unicorn"



- é um servidor HTTP Python Web Server Gateway Interface leve nos recursos do servidor e rápido.

# NGINX



- NGINX é um servidor web com estrutura assíncrona e orientada à eventos, possibilitando o processamento de muitas solicitações ao mesmo tempo.

# Setup de Produção





# Docker Compose



- Docker Compose é uma ferramenta para executar várias aplicações containers em um determinado Docker.

# Atividade 4

## Construir um container

1. Efetue fork do [unifametro\\_frameworkweb](#)
2. Clonar o repositório após o *fork*
3. Efetuar Setup do projeto
4. Compilar uma imagem Docker
5. Validá-la
6. Efetuar *commit* da atividade
  - *Procedimento operacionais no documento doc/atv4.md*

# Aplicando Arquivos Estáticos

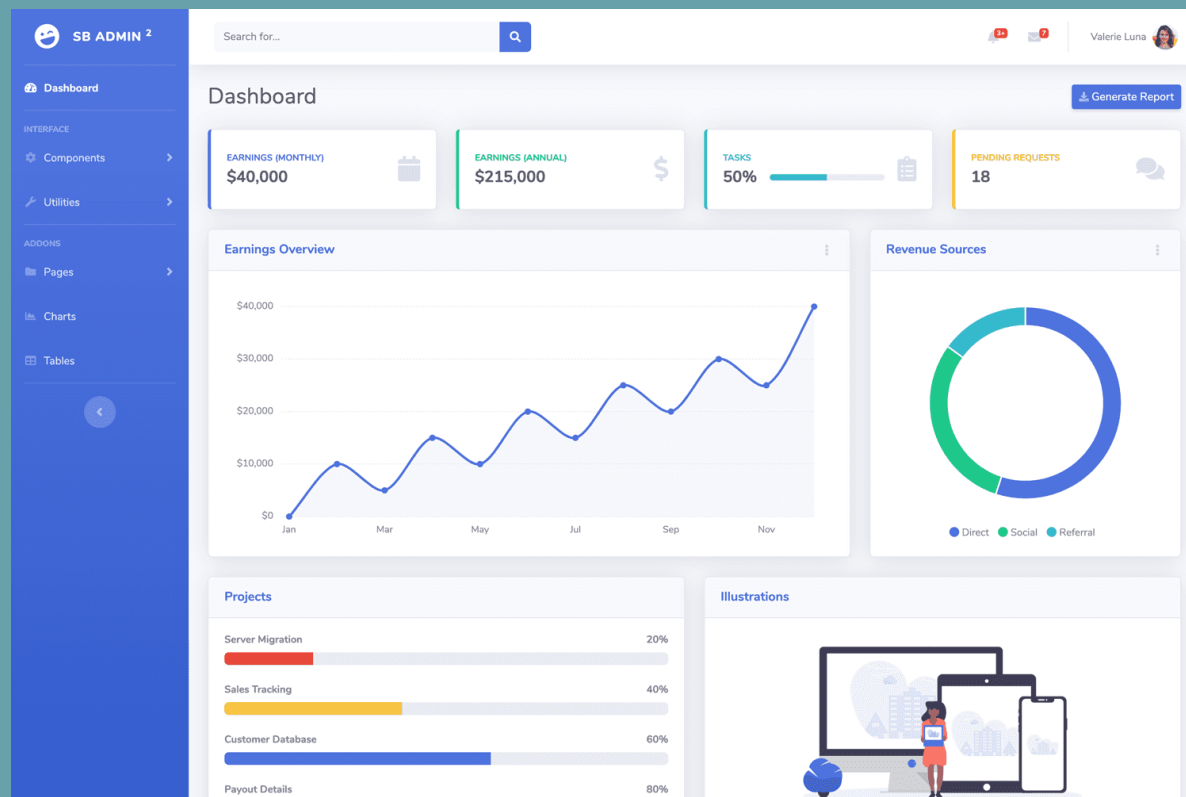


- O Front-end é em geral composto de imagens, estilos e fontes, provendo uma identidade visual adequada a cada aplicação WEB. Tais configurações são obtidas usando **CSS** (folhas de estilo em cascata), códigos **JavaScript** e imagens em suas variadas extensões e resoluções.
- O armazenamento de todas essas imagens, arquivos CSS e JS é realizado em uma pasta chamada estática.

# Dependências do frontend



- O Facebook lança em 2016 o Yarn, um gerenciador de pacotes front-end de código aberto.
- A repercussão foi tão positiva que o criadores do [Bower](#) descontinuaram a ferramenta substituindo pelo Yarn.




# ESTRUTURA DE TEMPLATE COM BOOTSTRAP E HTML

# Eventos JQuery, Ajax

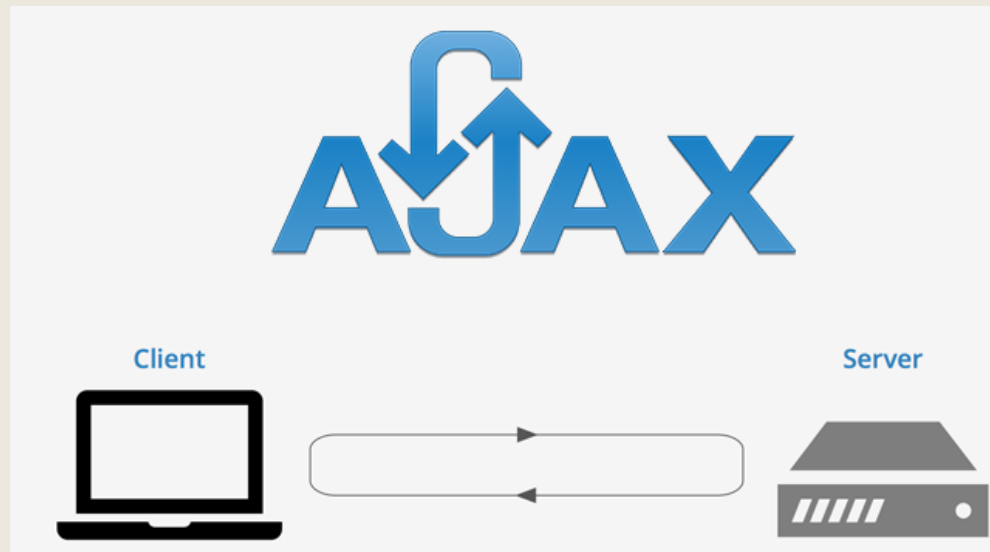
```
$( 'ul' ).children().slice(1,-2).css('background-color','salmon');
$( 'ul' ).children().first().has('b').css('background-color','violet');
$( 'ul' ).children().last().has('em').css('background-color','lime');

$( "ul" ).click(function( event ) {
    var target = $( event.target );

    if ( target.is( "b" ) ) {
        target.css( "background-color", "red" );
    }
});
```



- JQuery é uma ferramenta que permite a tonar as estrutura do código JS mais enxuta com elementos organizacionais bem diretos para manipulação Document Object Model no HTML.
- O Asynchronous JavaScript and XML permite efetuar requisições assíncronas ao servidor de forma imperceptível ao usuário final.



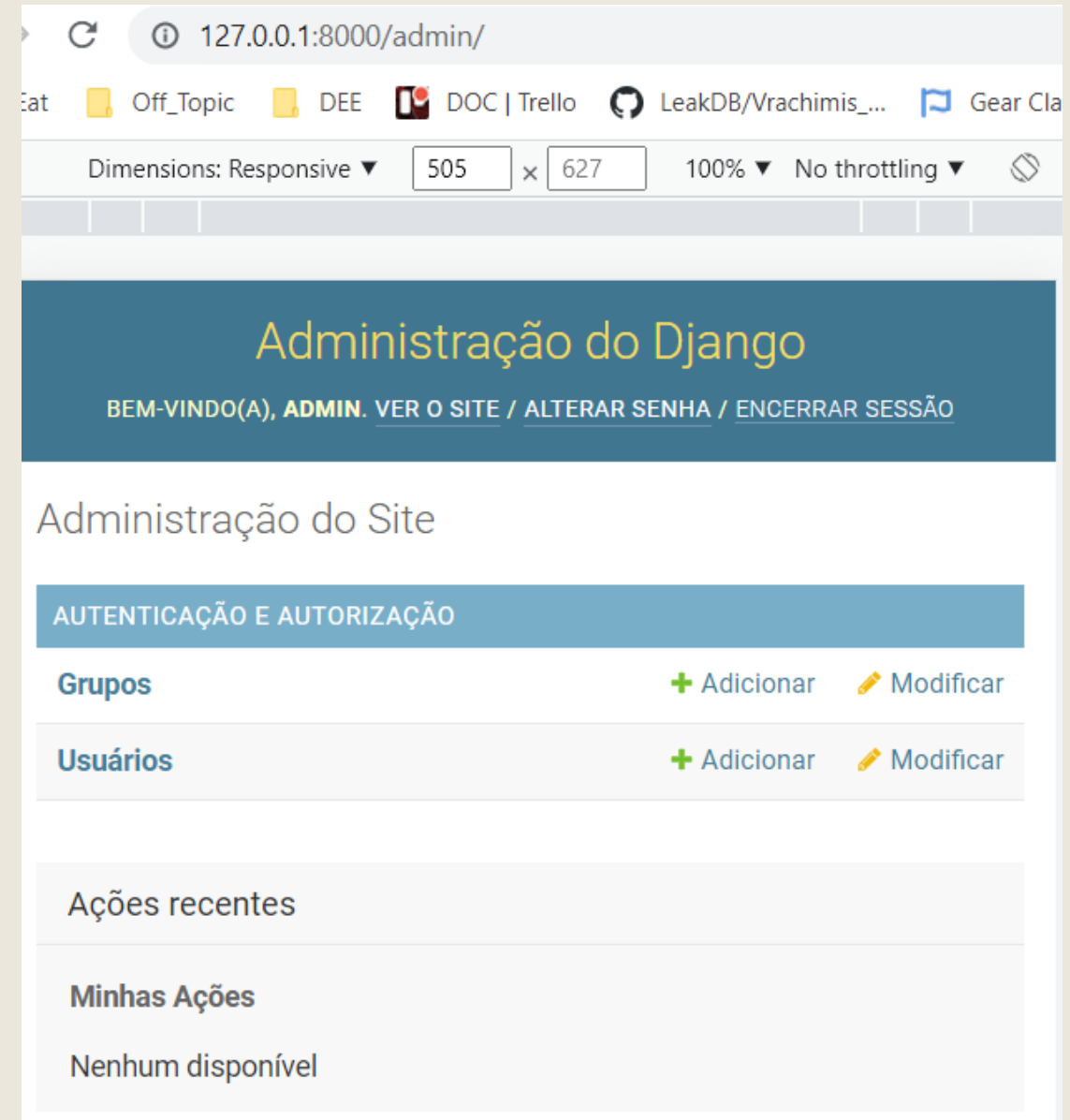
# Atividade 5

## Aplicar template Bootstrap

1. Configurar estrutura template Django:
2. Configurar arquivos estáticos:
3. Configuração de dependências CSS, JavaScript e Imagens:
4. Aplicar template Bootstrap
5. Efetuar *commit* da atividade
  - *Procedimento operacionais no documento doc/atv5.md*
  - *HTML dos template:*
    - <https://1drv.ms/u/s!AuEWHDDPI6pq3Q8wu3cfzMsc48Se?e=fz7f79>
    - **unifametro2023**

# Django Administrator

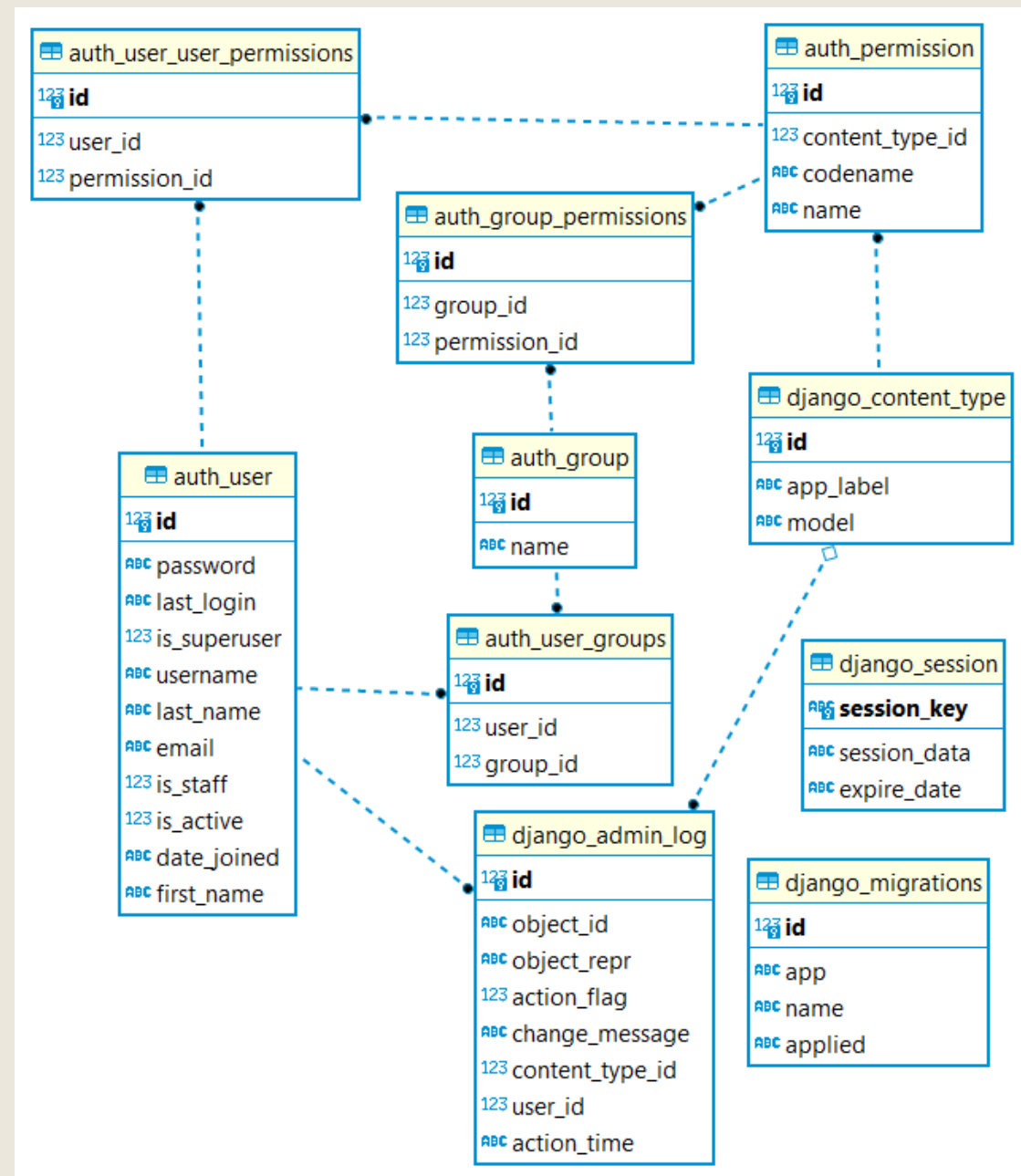
- O Painel de Administração do Django é um front-end que permite mantermos qualquer registro do modelo do projeto seja ele nativo ou dos modelos com o mapeamento do minimundo do projeto.





- O Painel de Administração do Django é um front-end que permite mantermos qualquer registro do modelo do projeto seja ele **nativo** ou dos modelos com o mapeamento do **minimundo do projeto**.

- O Painel de Administração do Django é um front-end que permite mantermos qualquer registro do modelo do projeto seja ele **nativo** ou dos modelos com o mapeamento do **minimundo do projeto**.



# Permissões de acesso Django

Permissões do usuário:

permissões do usuário disponíveis

ovino | ovino | Can add ovino

ovino | ovino | Can change ovino

ovino | ovino | Can delete ovino

ovino | ovino | Can view ovino

Escolher todos

permissões do usuário escolhido(s)

Remover todos

Permissões específicas para este usuário. Pressione "Control", ou "Command" no Mac, para selecionar mais de um.

- O Django suporta nativamente permissões baseadas em modelos.
- Desta forma possibilita atribuir ou remover a permissão Criar/Ver/Alterar/Deletar de um dado modelo.
- Elas são geradas pelo framework na inspeção dos modelos usados na aplicação listadas no `INSTALLED_APPS`.

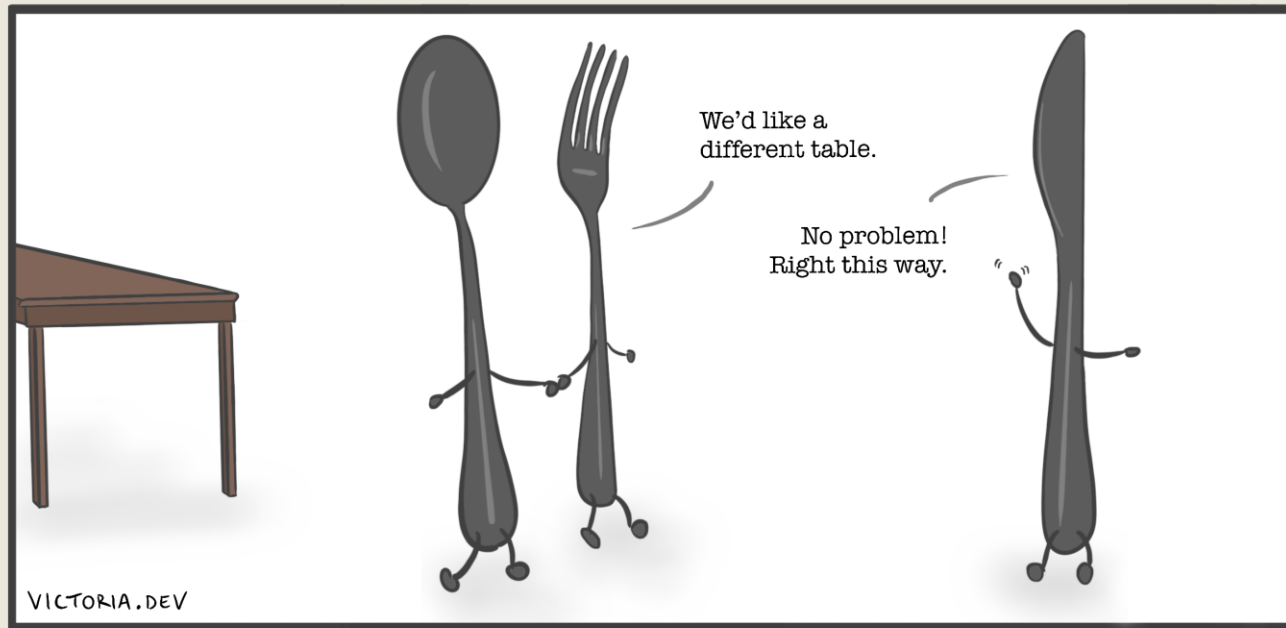
# Django Migrations

```
python manage.py makemigrations
Did you rename dinner.center to dinner.bottom_center (a CharField)? [y/N] y
Did you rename dinner.left_side to dinner.bottom_left (a CharField)? [y/N] y
Did you rename dinner.right_side to dinner.top_center (a CharField)? [y/N] y
Migrations for 'backend':
  backend/migrations/0004_auto_20200914_2345.py
    - Rename field center on dinner to bottom_center
    - Rename field left_side on dinner to bottom_left
    - Rename field right_side on dinner to top_center
```

```
$ python manage.py makemigrations
$ python manage.py migrate
```

- Este módulo permite recuperar toda estrutura ER da base de dados para um script unificado do Django, podendo ser aplicado em Base de dados ou SGBD distintos como SQL Lite e PostgreSQL por exemplo.

# Django Fixtures



- Este módulo permite levar os dados cadastrados em uma instancia do projeto para outra com do mesmo projeto. Desta forma podemos:
  - *Compartilhar uma base de validação.*
  - *Prover registros iniciais do sistema.*
  - *Entre outras situações.*

```
$ python manage.py dumpdata
```

```
$ python manage.py loaddata
```

# Banco de dados relacional PostgreSQL



Características	SQLite	PostgreSQL
Arquitetura	Baseado em arquivo	Cliente/Servidor
OS compatível	Serverless	FreeBSD, Linux, OS X, OpenBSD, HP-UX, Solaris, Unix e Windows
Replicação	n/a	Replicação Mestre-Escravo
Linguagem base	C	C
Suporta	ActionScript, Ada, Basic, C, C#, C++. D. Delphi, Forth, Fortran, Haskell, Java, JavaScript, Lisp, Lua, MATLAB, PHP e PL/SQL	NET, C, C++, Delphi, Java, Perl, PHP, Python e Tcl
Principais aplicações	Webservices de tráficos Baixo-Médio, IoT, Dispositivos embarcados, Testes e Desenvolvimento	Analíticos, Mineração de dados, Data Warehousing, Business Intelligence e Hadoop



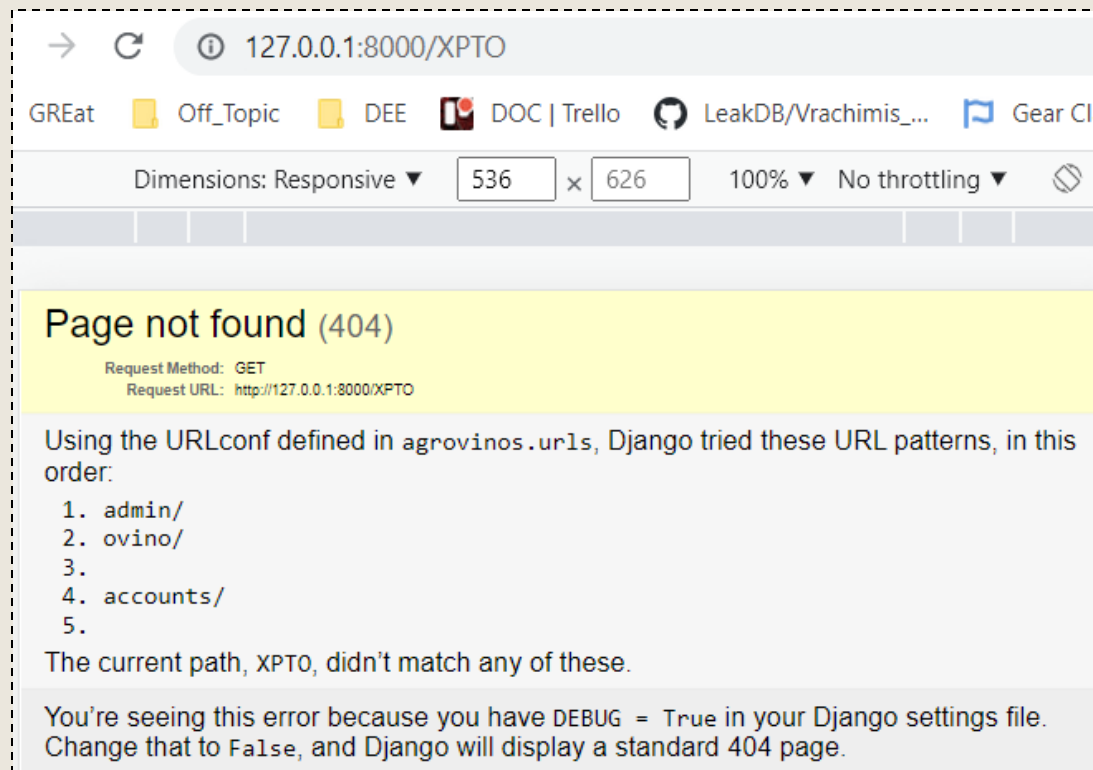
## Atividade 6

### Aplicar SGBD para PostgreSQL

e prover restrição de acesso aos métodos

1. Gerar um acesso sem privilégios de administrador:
2. Criando uma nova aplicação
3. Mapear uma aplicação no Django Administrator
4. Prover carga de dados na base de dados com o Django Fixtures
5. Aplicar SGBD PostgreSQL
6. Efetuar *commit* da atividade
  - *Procedimento operacionais no documento doc/atv6.md*

# Desativar a depuração



- Ao executar um site público, você deve sempre desativar a configuração DEBUG. Isso fará com que seu servidor funcione muito mais rápido e também impedirá que usuários mal-intencionados vejam detalhes de seu aplicativo que podem ser revelados pelas páginas de erro.
- No entanto, executar com DEBUG definido como Falso significa que você nunca verá erros gerados pelo seu site – todos verão suas páginas de erro públicas. Você precisa acompanhar os erros que ocorrem nos sites implantados, para que o Django possa ser configurado para criar relatórios com detalhes sobre esses erros.

# Customizar páginas de Erro

- **django.conf.urls** possui strings que representa o caminho de importação Python completo para exibição que deve ser chamada se o cliente HTTP enviou uma solicitação que causou uma condição de erro e uma resposta com um código de status:

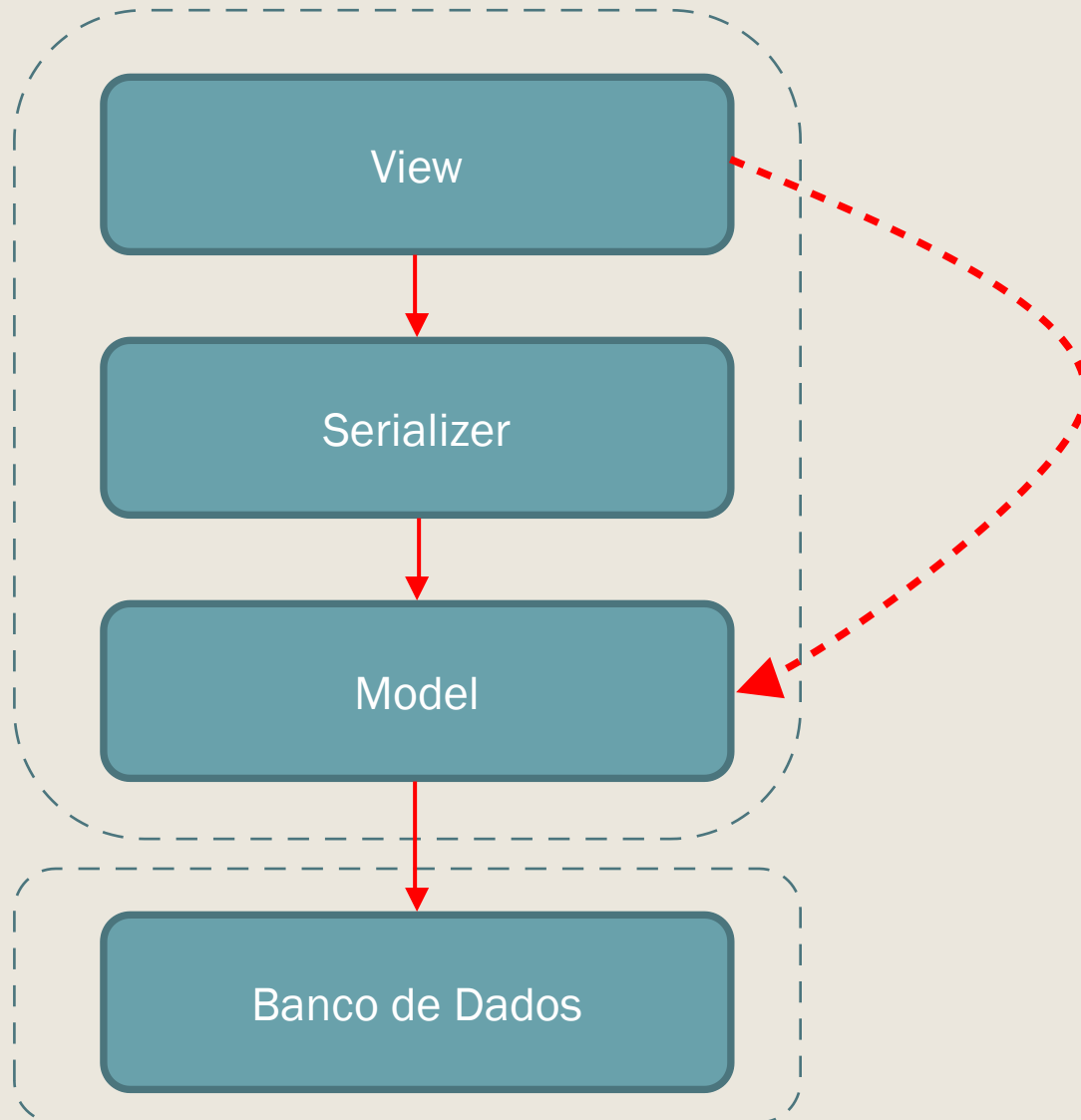
- **Handler400**
  - Template nativo 400.html
- **Handler403**
  - Template nativo 403.html
- **Handler404**
  - Template nativo 404.html
- **Handler500**
  - Template nativo 500.html

```
from django.http import Http404
from django.shortcuts import render
from polls.models import Poll

def detail(request, poll_id):
    try:
        p = Poll.objects.get(pk=poll_id)
    except Poll.DoesNotExist:
        raise Http404("Poll does not exist")
    return render(request, 'polls/detail.html', {'poll': p})
```





# Construindo uma aplicação REST



- O processo de serialização dos dados servem para traduzir entidades complexas, como *querysets* e instâncias de classes em representações simples que podem ser usadas no tráfego da web, como JSON ou XML

# Construindo uma aplicação REST

- Biblioteca Django Rest Framework
  - *Modelo*

```
tag >  models.py >  Tag
1  from django.db import models
2  import uuid
3
4
5  class Tag(models.Model):
6
7      id = models.UUIDField(
8          primary_key=True,
9          default=uuid.uuid4,
10         null=False,
11         blank=True
12     )
```

# Construindo uma aplicação REST

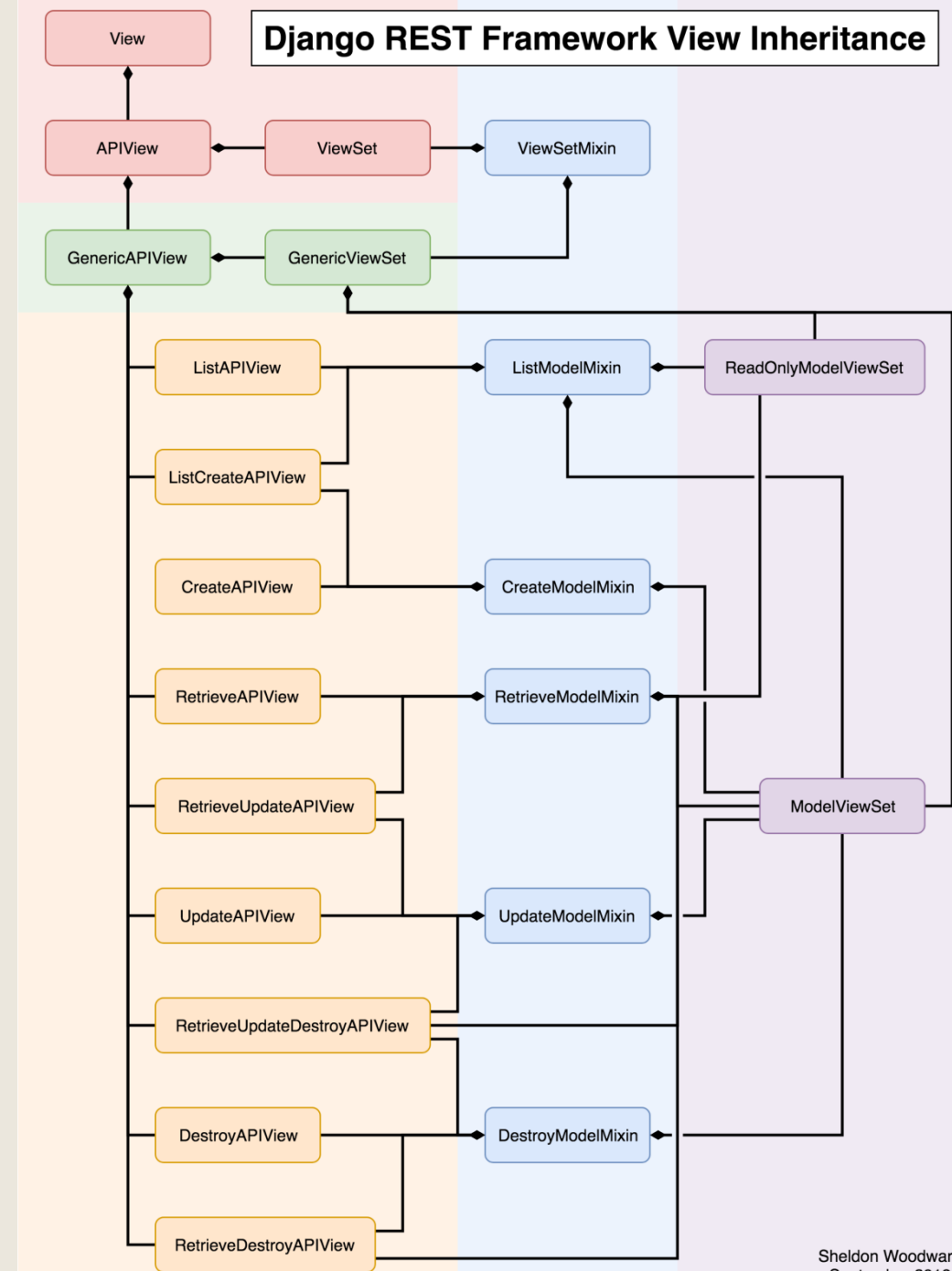
- Biblioteca Django Rest Framework
  - Serializer
    - **BaseSerializer:** Provê instâncias mais genéricas.
    - **ModelSerializer:** Gera serializadores baseados em modelos. Retorna o id da entidade relacionada.
    - **HyperlinkedModelSerializer:** Semelhante ao anterior, mas gera um link para representar o relacionamento entre entidades.

# Construindo uma aplicação REST

- Biblioteca Django Rest Framework
  - ViewSets
    - É o controle onde se define quais operações REST estarão disponíveis e como o sistema vai responder às chamadas para API
    - Elas generalizam e adicionam lógica às Views padrão do Django, responsáveis por:
      - Receber os dados da **Request** (formato JSON ou XML)
      - Validar os dados de acordo com as regras definidas no Modelo e no **Serializer**
      - Desserializar a **Request** e instanciar objetos
      - Processar regras de negócio
      - Formular um Response para cada requisição na API

# CONSTRUINDO UMA APLICAÇÃO REST

Biblioteca Django Rest Framework



# Construindo uma aplicação REST

- Biblioteca Django Rest Framework

- Router

- o REST possui padrões bem definidos de estrutura de URLs, essa biblioteca as gera automaticamente para manter o padrão adequado.
    - **app\_name** é necessário para dar contexto às URLs geradas, especificando o namespace das URLConfs adicionadas.
    - **DefaultRouter** é um dos Router disponíveis no Django para gerar URLs automaticamente. O parâmetro **trailing\_slash** especifica que não é necessário o uso de barras / no final da URL.
    - O método **register** recebe dois parâmetros: o primeiro é o prefixo que será usado na URL (**http://localhost:8000/tag**) e o segundo é a **View** que irá responder as URLs com esse prefixo.
    - E o **urlpatterns** do Django, que expõem as URLs dessa aplicação.

```
tag > urls.py > ...
1  from rest_framework.routers import DefaultRouter
2  from api.views import TagViewSet
3
4
5  app_name = 'api'
6
7  router = DefaultRouter(trailing_slash=False)
8  router.register(r'tag', TagViewSet)
9
10 urlpatterns = router.urls
```

# Construindo uma aplicação REST

- Biblioteca Django Rest Framework
  - Routers
    - É uma boa prática mantenha o prefixo **api/v1/** para simplificar atualizações futuras. Ex.: **api/v2/**

```
agrovinos > ✚ urls.py > ...
1  from django.contrib import admin
2  from django.urls import include, path
3  from ovino import views
4
5  urlpatterns = [
6  |      path('api/v1/', include('api.urls', namespace='api')),
7  |      path('admin/', admin.site.urls),
8  |      path('ovino/', include('ovino.urls')),
9  |      path('', views.index),
10 |  ]
```

URL	HTTP	Ação
/api/v1	GET	Documentação
/api/v1/tag	GET	Listar todos os registros
/api/v1/tag	POST	Criar um novo registro
/api/v1/tag/{lookup}	GET	Recuperar um registro
/api/v1/tag/{lookup}	PUT	Atualizar um registro
/api/v1/tag/{lookup}	PATCH	Atualização parcial
/api/v1/tag/{lookup}	DELETE	Deleção de um registro

# Atividade 7

## Gerar aplicação REST

1. Instalando o Django Rest Framework
2. Configurando o Django Rest Framework
3. Publicar atividade
4. Efetuar *commit* da atividade
  - *Procedimento operacionais no documento doc/atv7.md*



The image features a dark blue background with a subtle pattern of 3D question marks. A large, white, L-shaped frame is positioned on the left and bottom edges, framing the central text. The word "DUVIDAS?" is written in a bold, white, sans-serif font in the center of the image.

DUVIDAS?

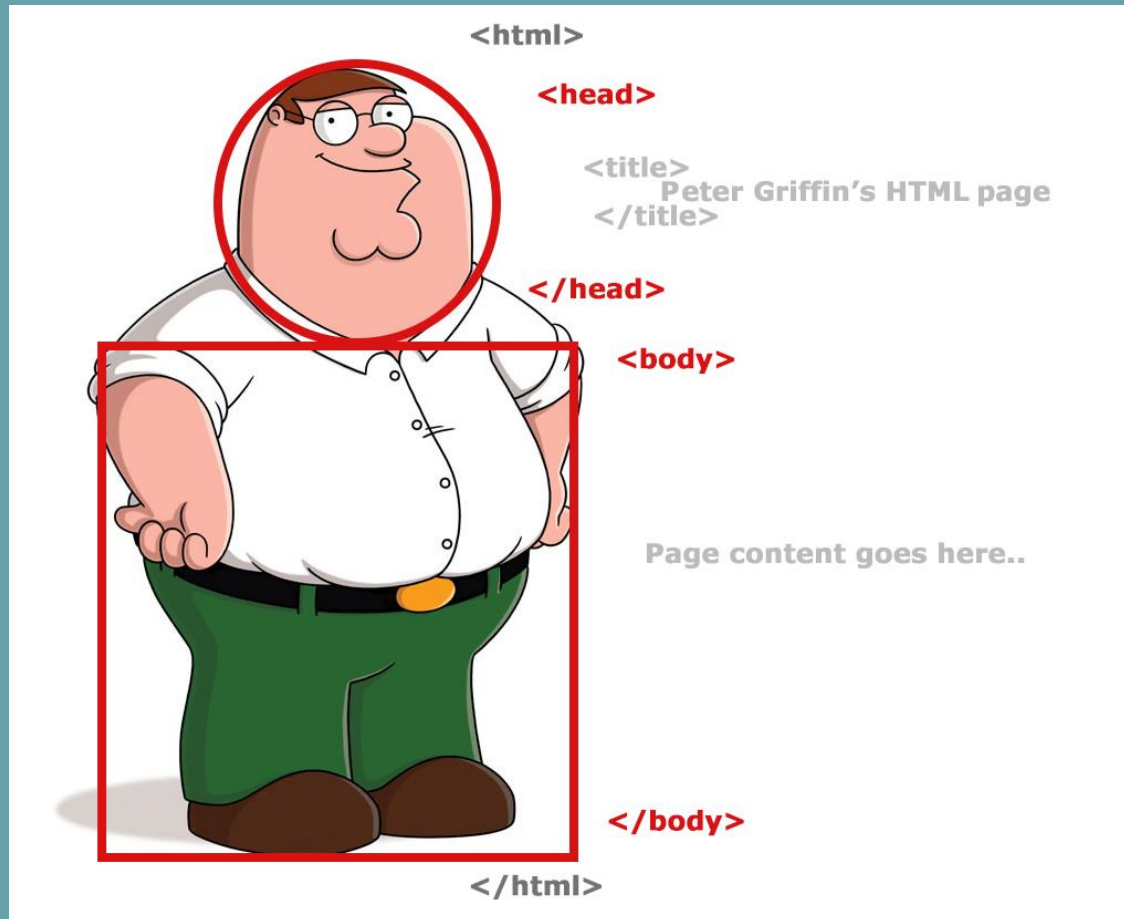
# Questionário

- Prazo  
21/02/2024!!!
- <https://forms.office.com/r/r3uYd7hnUy>



# Referências

- **[Doker-2023]** DOCS, Docker. Overview of Docker Hub. Internet: <https://docs.docker.com/docker-hub/> (Jan. 23, 2024), 2024.
- Foundation, D. S. (2023a). Documentação do Django. [Online; acessada 23-janeiro-2024].



BONS  
ESTUDOS!