



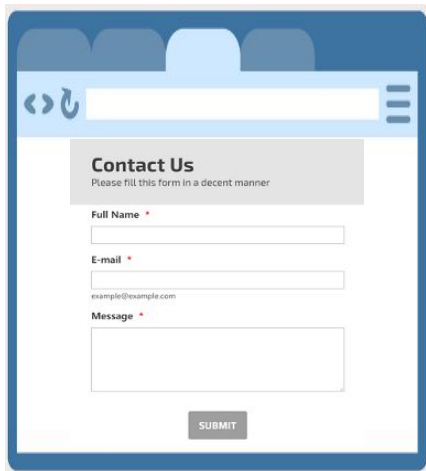
Anatomia do Django & Web

Vimos a
arquitectura
Cliente | Servidor

CLIENT

SERVER

REQUEST →



Contact Us
Please fill this form in a decent manner

Full Name *

E-mail *

Message *

SUBMIT

REQUEST →

← RESPONSE



Ubuntu 20.04 LTS

HTTP SERVER



Front-end
(estático)

```
minha-app/  
css/  
js/  
imgs/  
index.html
```



Backend

```
cursos/  
settings/  
core/  
app.py
```

A forma Frontend/Backend
separados pode ter infinitas
vantagens, mas no mínimo teremos
mais complexidade, mais
configurações e mais devops!

É possível uma
única aplicação
fazer tudo?

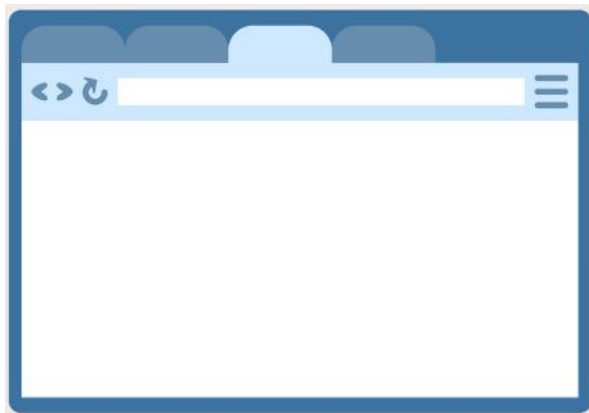
(Front e Back)

Resposta: **SIM**

Existem diversos frameworks
no mercado fazendo isto!

CLIENT

SERVER



/alguma-pagina.html

REQUEST



RESPONSE

HTTP SERVER



Geralmente
configurações de
rotas + simples...

- Pode ser um HTML estático
- Uma imagem ou arquivo CSS
- Um HTML gerado dinamicamente
- Uma lista de clientes no formato JSON
- Um PDF assinado conforme o usuário logado e muito mais



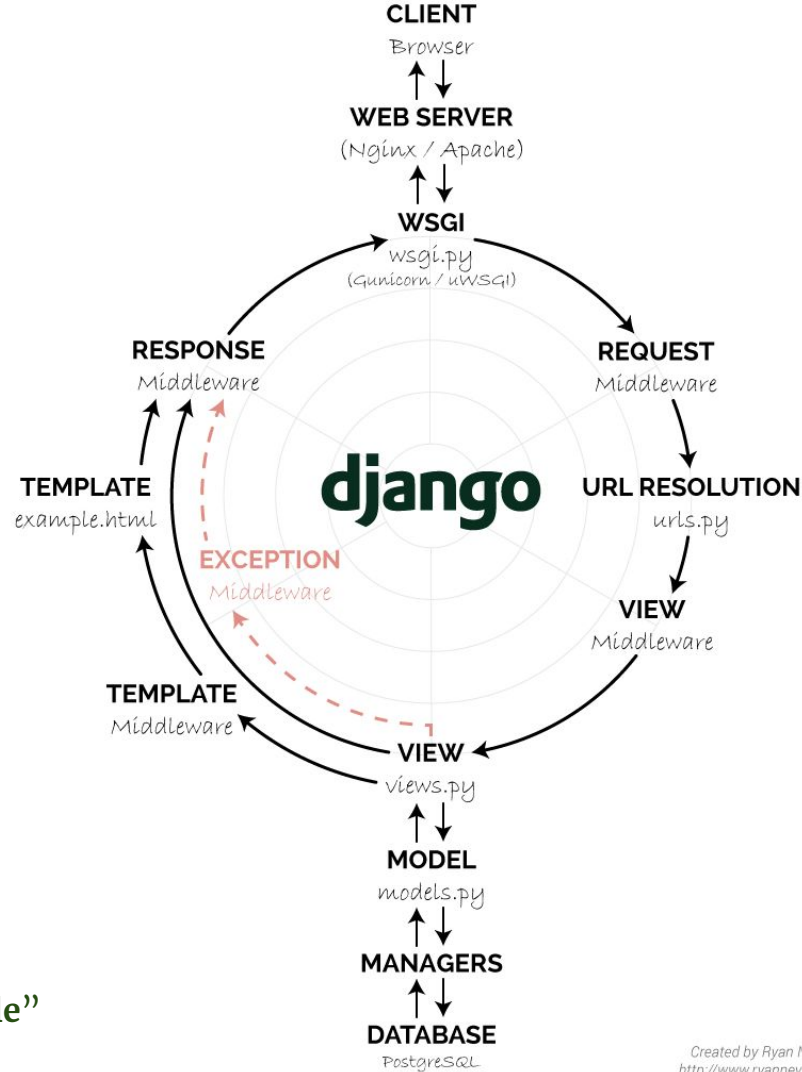


Frontend + Backend
e tudo com esteróides

Okay!

Parece bonito, mas ainda não
está muito vago?

Visão geral



Pesquise por “Django life cycle”

Okay!

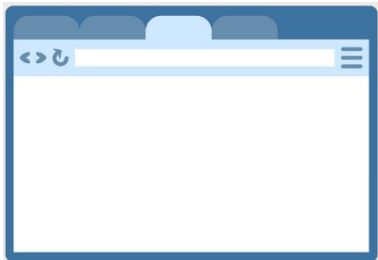
Agora pareceu complicado!

Baby Steps

- Django tem diversas funcionalidades
- Cada módulo.py faz poucas coisas (responsabilidade única)
- Todas partes são conectadas para gerar um resultado
- Tem um fluxo de execução para atender **Request** até gerar **Response**
- Todas partes podem ser configuradas, desabilitadas ou alteradas, depende da necessidade de cada caso/projeto!

CLIENT

Navegador



REQUEST



SERVER

HTTP/ Web
Server



Process rodando em
uma porta do S.O.



Gerenciador de rotas
(**urls.py**)

No **urls.py**
Vamos cadastrar as rotas da
nossa aplicação:

```
principal  
contato  
api/v1/clientes  
api/v1/clientes/<id>  
api/v1/cursos  
api/v1/pontos
```



Cada rota faz referência
para função dentro da
views.py

ATENÇÃO: Estes são alguns pseudos códigos para efeito didático

CLIENT



REQUEST



SERVER

HTTP/ Web
Server



Process rodando em
uma porta do S.O.



Gerenciador de rotas
(urls.py)



Controlador ou parte que
lida com as requests
(views.py)

Na **views.py**
Temos uma função que irá
receber o que desejarmos com
ela...

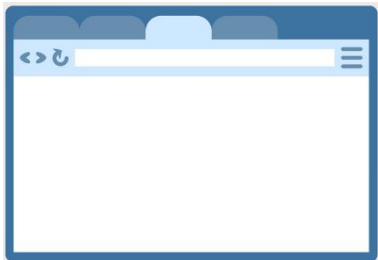
```
def contato(request):  
    return template(html)  
  
def listar_cursos(request):  
    lista = obtem_cursos()  
    return Json(lista)  
  
def pagina_invalida(request):  
    return HttpResponse(404)
```

Em uma função
podemos fazer qualquer
código, desvio
condicional, chamadas
para outras funções,
validar a requisição,
redirecionar para outra
view...

ATENÇÃO: Estes são alguns pseudos códigos para efeito didático

CLIENT

Navegador



REQUEST



SERVER

HTTP/ Web
Server



Process rodando em
uma porta do S.O.



Gerenciador de rotas
(`urls.py`)



Controlador ou parte que
lida com as requests
(`views.py`)

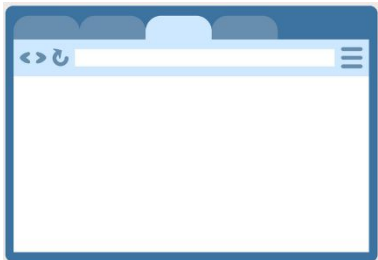
Gerar páginas HTML
(`templates`)



- Dentro da pasta `templates/`
Podemos organizar muitas páginas HTML
- Dentro de um arquivo HTML,
podemos utilizar a linguagem de
template do DJANGO, a qual tem muitos
recursos, como formatar, importar
partes de outros arquivos e infinitas
coisas. Tudo para facilitar a criação de
páginas Web

CLIENT

Navegador



REQUEST



SERVER

HTTP/ Web
Server



Process rodando em
uma porta do S.O.



Gerenciador de rotas
(`urls.py`)



Controlador ou parte que
lida com as requests
(`views.py`)

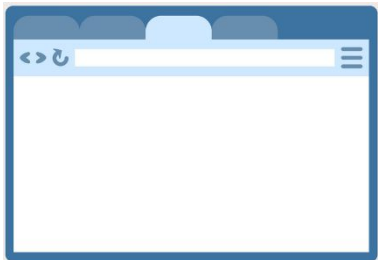
Gerar páginas HTML
(`templates`)



O conteúdo de uma página HTML usando a linguagem de template NÃO é entendida pelo navegador.
Então, o Django sabe ler esta página com recursos extras e converter para um HTML final o qual é aceito pelos browsers.
No template, podemos pegar o usuário logado ou a data do servidor entre muitas outras coisas, tudo isto será convertido pelo Django (processamento)
TEMPLATE.html ->
arquivo-html-final.html

CLIENT

Navegador



REQUEST



SERVER

HTTP/ Web Server



Process rodando em
uma porta do S.O.



Gerenciador de rotas
(urls.py)

Gerar páginas HTML
(templates)

Controlador ou parte que
lida com as requests
(views.py)

```
<h2>Meu nome é {{ first_name }}.</h2>
```

```
{{ my_date|date:"Y-m-d" }}
```

```
{{ meu_dicionario.key }}
```

```
{% if user.is_authenticated %}Olá, {{ user.username }}.{% endif %}
```

```
<p>{{ curso.nome|truncatewords:"100" }}</p>
```

Na **views.py** (controladora), podemos gerar dados e transportá-los para os templates pelo **context** ou **dicionários**.

E como estas estas informações chegam no template?

```
<h2>Meu nome é {{ first_name }}.</h2>
```

```
{{ my_date|date:"Y-m-d" }}
```

```
{{ meu_dicionario.key }}
```

```
{% if user.is_authenticated %}Olá, {{ user.username }}.{% endif %}
```

```
<p>{{ curso.nome|truncatewords:"100" }}</p>
```

Gerar páginas HTML
(templates)



SERVER

Process rodando em uma porta do S.O.

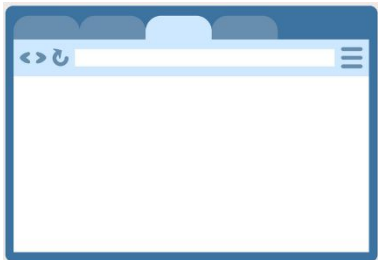


Gerenciador de rotas
(urls.py)

Controlador ou parte que lida com as requests
(views.py)

CLIENT

Navegador



REQUEST



SERVER

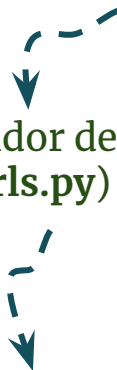
HTTP/ Web
Server



Process rodando em
uma porta do S.O.



Gerenciador de rotas
(urls.py)



Controlador ou parte que
lida com as requests
(views.py)



Obter ou Manipular
dados
(models|managers)



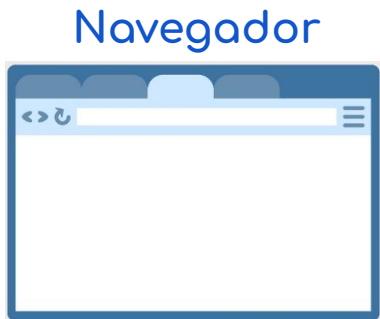
Database

O Django tem internamente um ORM, um componente e padrão de projeto que sabe fazer toda comunicação com o banco de dados.

Também sabe gerar tabelas ou novos campos

Fazer acesso aos dados usando uma linguagem própria ou usando “RAW” SQL

CLIENT



REQUEST



SERVER

HTTP/ Web
Server



Process rodando em
uma porta do S.O.



Managers != manage.py

Gerenciador de rotas
(urls.py)



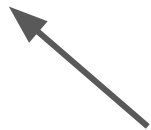
Controlador ou parte que
lida com as requests
(views.py)

Gerar páginas HTML
(templates)



Gera resposta
WEB final
(de qualquer tipo|content-type)

RESPONSE



Obter ou Manipular
dados
(models|managers)



Database

Ops!
Não acabou...

CLIENT



REQUEST



SERVER

HTTP/ Web
Server



Process rodando em
uma porta do S.O.



Gerenciador de rotas
(urls.py)



Controlador ou parte que
lida com as requests
(views.py)



Gerar páginas HTML
(templates)



Obter ou Manipular
dados
(models|managers)



Database

RESPONSE



Gera resposta
WEB final
(de qualquer tipo|content-type)

Precisamos de JS

Para conversar entre
templates e views?

NÃO

Podemos adicionar
JS e fazer firulas no
HTML?

Claro!

Podemos tudo ou
quase tudo!

Estrutura & organização de projeto

```
django-admin startproject cursos_de_django
```

cursos_de_django

```
|— cursos_de_django
|   |— asgi.py
|   |— __init__.py
|   |— settings.py
|   |— urls.py
|   |— wsgi.py
|— manage.py
|— db.sqlite3
```

cursos_de_django

```
├── cursos_de_django
│   ├── asgi.py
│   ├── __init__.py
│   ├── settings.py
│   ├── urls.py
│   └── wsgi.py
├── cursos
│   ├── admin.py
│   ├── apps.py
│   ├── __init__.py
│   ├── migrations
│   │   └── __init__.py
│   ├── models.py
│   ├── templates
│   │   └── cursos
│   │       └── home.html
│   ├── tests.py
│   ├── urls.py
│   └── views.py
├── manage.py
└── db.sqlite
```

`./manage.py start app cursos`

`cursos_de_django`

```
├── cursos_de_django
│   ├── asgi.py
│   ├── __init__.py
│   ├── settings.py
│   ├── urls.py
│   └── wsgi.py
├── cursos
│   ├── admin.py
│   ├── apps.py
│   ├── __init__.py
│   ├── migrations
│   │   └── __init__.py
│   ├── models.py
│   ├── templates
│   │   └── cursos
│   │       └── home.html
│   ├── tests.py
│   ├── urls.py
│   └── views.py
├── manage.py
└── db.sqlite3
```

No `settings.py` temos falar para o projeto que existe uma app chamada **cursos**

Também precisamos incluir as rotas da app **cursos** dentro do arquivos de rotas inicial do projeto (`cursos_de_django/urls.py`)

Assim todas rotas de **cursos** serão reconhecidas pelo projeto

`cursos_de_django`

```
├── cursos_de_django
│   ├── asgi.py
│   ├── __init__.py
│   ├── settings.py
│   ├── urls.py
│   └── wsgi.py
├── cursos
│   ├── admin.py
│   ├── apps.py
│   ├── __init__.py
│   ├── migrations
│   ├── models.py
│   ├── templates
│   ├── tests.py
│   ├── urls.py
│   └── views.py
├── manage.py
└── db.sqlite3
```

aulas!
Onde coloco?

Imagine que temos diversas rotas registradas em `cursos/urls.py`, todas relacionadas ao contexto de cursos...

Mas surgiu a necessidade de ter uma listagem de todas as aulas de um curso específico.

Algo como:

```
# urls.py
cursos/django/aulas    ## -> uma lista com todas as aulas
                        ##      do curso de Django
```

Desta maneira, uma grande dúvida que apareceu foi relacionada se devemos ter uma **nova app chamada aulas** também?
E a app aulas deveria estar **dentro da app cursos**?

Faz sentido isto?

cursos_de_django

```
├── cursos_de_django
│   ├── asgi.py
│   ├── __init__.py
│   ├── settings.py
│   ├── urls.py
│   └── wsgi.py
├── cursos
│   ├── admin.py
│   ├── apps.py
│   ├── __init__.py
│   ├── migrations
│   ├── models.py
│   ├── templates
│   ├── tests.py
│   ├── urls.py
│   └── views.py
├── manage.py
└── db.sqlite3
```

Uma abordagem interessante é pensar que a entidade Aula e Cursos são relacionadas, logo elas podem estar dentro da mesma app curso.

Dentro da app cursos, podemos ter rotas para aulas normalmente! Não um relacionamento algo um para um, ou que temos que ter app para tudo.

E outro ponto é, a ideia é ter apps que são filhas do projeto principal, não faz sentido ter a app aula dentro de outra app.

FIM