**Fazendo o deploy de uma aplicação Django**

Sabemos como o Django, framework web escrito em Python, pode ser bom para aplicações web. Mas ao final de todo o desenvolvimento, o que fazemos com todo o projeto?

Recentemente, desenvolvi um [simples projeto de agenda de contatos](https://github.com/yanorestes/aluraagenda) utilizando o framework **[Django](https://www.djangoproject.com/" \t "_blank)**, e resolvi compartilhar pela Internet. É uma aplicação web, então o ideal seria que fosse possível acessá-la como acessamos qualquer outro site, digitando seu endereço em um navegador qualquer, em um computador qualquer.

Assim, me preparei aluguei um [**VPS**](https://pt.wikipedia.org/wiki/Servidor_virtual_privado) com Linux e registrei um domínio estilo dominio.com.br. O serviço do servidor possuía diversos tutoriais ensinando a colocar projetos em PHP e Java de pé, mas não de Django… E aí, como podemos rodar nossa aplicação?

## Enviando os arquivos locais para o servidor remoto

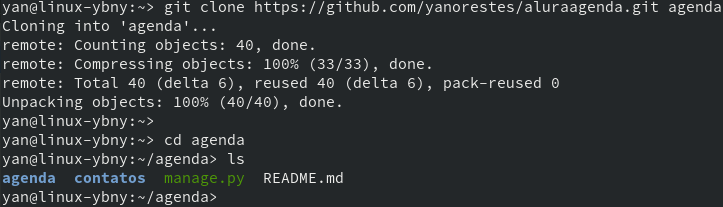
A primeira coisa que precisamos fazer é, realmente, mandar nossa aplicação para o servidor que eu aluguei. Afinal, atualmente tenho todos os meus arquivos aqui no meu computador, mas não vou rodar a aplicação por ele.

Para esse primeiro passo, temos várias opções, como [**FTP**](https://pt.wikipedia.org/wiki/File_Transfer_Protocol), [**SFTP**](https://pt.wikipedia.org/wiki/SSH_File_Transfer_Protocol), ou o **[Git](https://www.alura.com.br/artigos/comecando-com-git-aprendendo-versionar/)**. Por conta da simplicidade, vou optar pelo Git, aproveitando que meu projeto já está no [GitHub](https://github.com/yanorestes/aluraagenda).

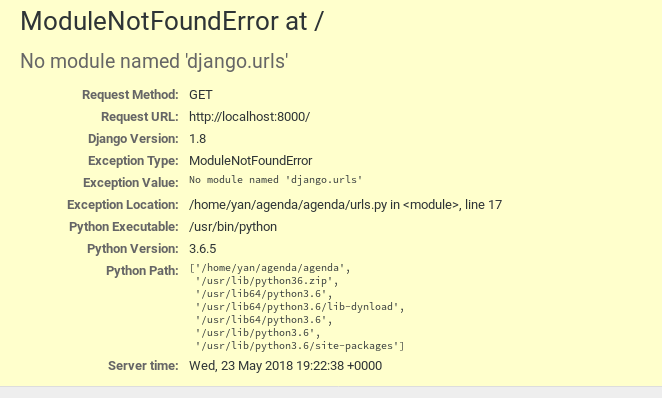
Em minha pasta de usuário (home), quero clonar (clone) o repositório que está lá no GitHub para um diretório chamado agenda. Um simples comando faz o trabalho:

git clone https://github.com/yanorestes/aluraagenda.git agenda

Rapidamente, podemos ver que uma pasta agenda (como especificamos no final do comando) foi criada em nosso diretório home, contendo os arquivos do projeto - a aplicação Django em si:



Como a maioria dos sistemas Linux, o meu já veio com o Python instalado. Além disso, fiz um pedido para que viesse também com o Django. Assim, fui tentar rodar a aplicação, como teste, mas olha o que aconteceu:



Mas, ué, na minha máquina local estava funcionando perfeitamente! O que foi que deu? Com um pouco de atenção, eu percebi - a mensagem de erro indicava que a versão do Django era a 1.8.

O problema é que eu programei me baseando na versão 2.0, que é mais recente e quebra algumas coisas das versões passadas. Será que vou ter que desinstalar o Django e instalar a nova versão?

Isso até poderia resolver… Se bem que eu tenho alguns outros projetos em que uso a versão 1.8 do Django, e estava até pensando em deixar nesse servidor. Mas e agora? Pelo visto, o ideal seria ter isso mais separado, sem cada instalação interferir de um jeito diferente o nosso projeto. E como podemos fazer isso?

## Criando um ambiente virtual Python com o virtualenv

Queremos dividir nossas aplicações em ambientes separados, cada qual com suas especificidades, para não nos embaralharmos com nossos pacotes Python.

Ou seja, queremos criar ambientes virtuais (**virtual environments**), onde cada ambiente terá sua própria versão dos pacotes. Mas como podemos criá-los?

No Python, temos o ***[virtualenv](https://virtualenv.pypa.io/en/stable/" \t "_blank)***, que cria ambientes Python isolados, nos quais podemos aplicar as configurações que desejarmos.

Podemos instalá-lo através do **[pip](https://www.alura.com.br/artigos/como-publicar-seu-codigo-python-no-pypi)**, com o seguinte comando no terminal:

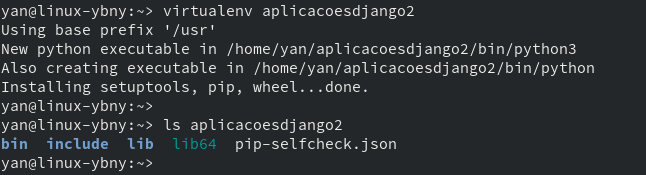
pip install virtualenv

*Dependendo da configuração do seu sistema, será necessária permissão sudo para rodar esse comando*

Com o virtualenv instalado, podemos criar um novo ambiente com um simples comando, especificando o nome que queremos definir:

virtualenv aplicacoesdjango2

Rodando esse comando, um diretório deve ser criado:



Precisamos, agora, dizer para o terminal que queremos entrar nesse ambiente, isto é, ativar esse virtualenv que criamos. Fazemos isso com o comando source, especificando a localização do arquivo de ativação, que é sempre a mesma:

source aplicacoesdjango2/bin/activate

E olha como fica o terminal agora:

O terminal muda quando estamos dentro de uma virtualenv

Podemos, enfim, instalar a versão que queremos do Django para esse ambiente virtual, através do pip:

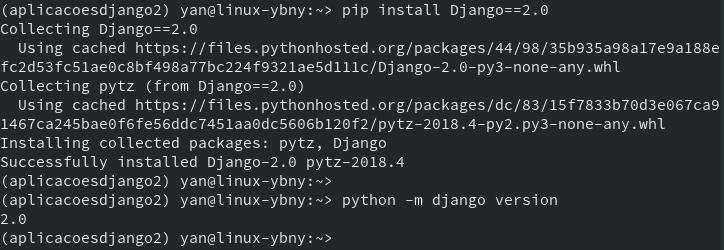
pip install Django==2.0

*É importante que esse comando não seja executado com sudo, ou o pip fará a instalação no Python local da máquina, não do ambiente virtual.*

Podemos verificar se a instalação foi bem sucedida com um comando com o Python:

python -m django version

Se tudo der certo, o resultado deve indicar **2.0**:



Para finalizar essa parte, vamos mover (mv) nossa aplicação para o diretório do ambiente virtual, apenas para manter uma melhor organização:

mv agenda aplicacoesdjango2

Certo! Entrei (cd) no diretório da aplicação para testar:

cd aplicacoesdjango2/agenda

Coloquei o servidor do Django para rodar, com o clássico comando runserver:

python manage.py runserver

Mas… novamente um erro! Dessa vez, direto no terminal:

*# ERRO OMITIDO*

django.core.exceptions.ImproperlyConfigured: Error loading MySQLdb module.

Did you install mysqlclient?

A longa exceção que recebemos indicou um erro de configuração, nos questionando se instalamos o driver conector do MySQL com o Python. De fato, não instalamos. Na verdade, nem ao menos instalamos o MySQL em si! Vamos cuidar disso, então.

## Instalando o MySQL (MariaDB)

Antes de tudo, precisamos apertar **CTRL-C** para sair do runserver falho. Feito isso, agora precisamos instalar o banco de dados.

No caso, usarei o **[MariaDB](https://mariadb.org/" \t "_blank)**, que é um fork do MySQL, porque é como está configurado em minha máquina local. Caso quiséssemos usar outro serviço, como o PostgreSQL, ou o SQLite, essa parte de configuração seria diferente.

A instalação também depende da distribuição Linux e gerenciador de pacotes que estamos usando. No nosso caso, vamos focar na instalação com o **[apt](https://pt.wikipedia.org/wiki/Advanced_Packaging_Tool" \t "_blank)**, padrão de distribuições derivadas do Debian, como o Ubuntu, por ser o mais conhecido.

Apesar disso, é importante que fique claro que não é a única possibilidade! Bastaria um pouco de pesquisa para descobrir os nomes dos pacotes relativos ao que instalaremos pelo apt.

Antes de começar a instalação, vamos sair do nosso ambiente virtual, pois não precisamos dele agora. Para isso, basta usar o comando deactivate no terminal:

Saindo da virtualenv com o deactivate

Partindo para a instalação, temos quatro pacotes no apt que precisamos instalar - mariadb-server, python-dev, libmariadbclient-dev, libssl-dev. Esses pacotes contêm, além do banco de dados, arquivos de cabeçalho importantes.

Instalados os pacotes, vamos garantir que o MariaDB já está rodando:

sudo service mysql start

Agora podemos partir para a instalação do driver de conexão com o Python. Como é um pacote Python, vamos voltar para nosso virtualenv:

source ~/aplicacoesdjango2/bin/activate

O pacote, de nome mysqlclient, pode ser instalado com o pip:

pip install mysqlclient

Tudo ok! Com tudo instalado, precisamos configurar nosso banco, criando a database.

## Configurando o MySQL (MariaDB)

Temos o banco de dados e suas dependências instaladas, mas ele ainda está completamente vazio, sem configuração alguma.

Assim, antes de qualquer coisa, é importante que criemos nossa database (CREATE DATABASE) agenda e nosso usuário (CREATE USER), que vou chamar de yan. Também temos que dar os privilégios necessários (GRANT ALL PRIVILEGES) para nosso usuário.

*Caso SQL ainda seja um tópico confuso, temos uma*[*apostila disponível gratuitamente*](https://www.alura.com.br/apostilas)*que trata da linguagem e de bancos de dados relacionais, além dos diversos*[*cursos na Alura*](https://www.alura.com.br/carreira-iniciando-com-sql-e-mysql)*! ;)*

Note que ainda não criamos a tabela de contatos que meu projeto usa, e é porque fazer isso manualmente é desnecessário! Já já vamos ver como o Django pode nos ajudar nisso.

Com o básico do banco configurado, precisamos passar para o próprio Django essas configurações. Para isso, vamos alterar algumas coisas no arquivo setttings.py, dentro da pasta agenda, de nossa aplicação agenda. Vamos entrar em nosso ambiente virtual e editar o arquivo com o nano:

source ~/aplicacoesdjango2/bin/activate

nano ~/aplicacoesdjango2/agenda/agenda/settings.py

E agora, o que devemos mexer nele?

## Configurando o banco de dados no Django

Com o arquivo settings.py aberto, vamos procurar a parte que trata do banco de dados, localizada no dicionário DATABASES. Dentro dele, há outro dicionário, 'default', com as configurações padrões - é nele que queremos mexer.

A primeira variável, 'ENGINE', diz respeito ao tipo de banco de dados que estamos usando. Como estamos com o MariaDB, que é um fork do MySQL, devemos colocar o valor 'django.db.backends.mysql'. 'NAME' trata do nome da database, e USER e PASSWORD das informações do usuário. Podemos deixar 'HOST' e 'PORT' vazios, já que o banco está rodando no próprio [localhost](https://pt.wikipedia.org/wiki/Localhost" \t "_blank) do servidor. Ao final, esse bloco do código ficou assim:

DATABASES = {

'default': {

'ENGINE': 'django.db.backends.mysql',

'NAME': 'agenda',

'USER': 'yan',

'PASSWORD': 'senha',

'HOST': '',

'PORT': '',

}

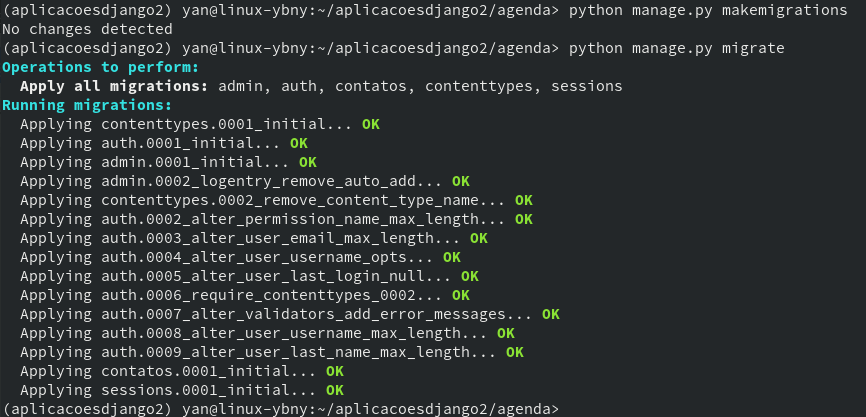
}

Com essa estrutura definida, podemos fazer a migração com o Django:

python manage.py makemigrations

python manage.py migrate

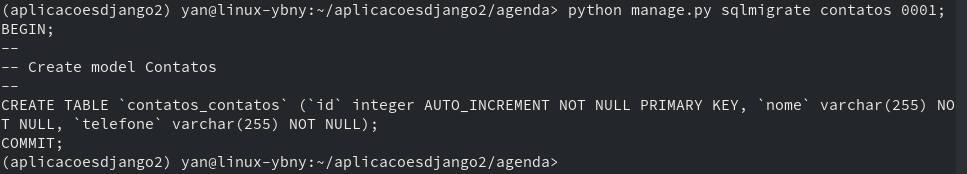
Algumas mensagens devem aparecer indicando que tudo deu certo:



Só falta criar a tabela que configurei no meu projeto Django, a contatos. Aqui, o Django pode nos ajudar muito com um simples comando, no qual só precisamos passar o nome do app e o nome da migração:

python manage.py sqlmigrate contatos 0001

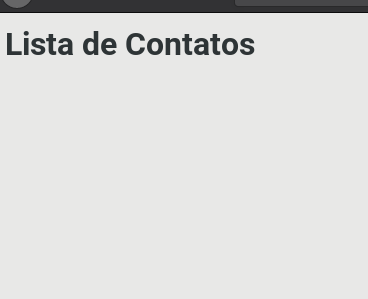
Olha só:



Com a tabela criada, vamos rodar o servidor com o runserver para ver se dessa vez dá certo:

python manage.py runserver

E agora, quando testo no navegador, através da porta **8000** (padrão do Django):



Certo! Bem, já conseguimos até acessar o site, mesmo que por uma porta que não é padrão do HTTP. Então é isso, está tudo certo? Qual seria o problema de deixar dessa forma?

## As questões do servidor de desenvolvimento do Django

Logo quando comecei a me aventurar no mundo do Django, já percebi que era um consenso não usar o servidor de desenvolvimento nativo do framework em ambiente de produção. Na verdade, [**a própria documentação**](https://docs.djangoproject.com/en/2.0/intro/tutorial01/#the-development-server) da ferramenta nos indica isso. Mas por quê, se nos meus testes ele funciona bem?

O que acontece é que o ponto é justamente esse - testes. Esse servidor do Django existe apenas como uma forma rápida e otimizada de rodar nosso código durante o desenvolvimento. Assim, possui algumas características que o tornam impróprios para uso em produção.

Em primeiro lugar, o servidor do Django **não é seguro**. E por quê? Pelo simples fato de não ter sido criado para isso - segurança do servidor não é uma preocupação enquanto queremos testar apenas o código da aplicação; é um setor separado.

Outro ponto está nas features que esse servidor carrega. Além de não suportar algumas que podem ser bastante importantes, como multi-threading, tem outras, como a habilidade de aplicar as mudanças que fazemos sem termos de recarregar o servidor manualmente, que podem forçar bastante o processamento da máquina.

Finalmente, temos a questão da performance em si. O servidor do Django simplesmente não aguenta muitas requisições ao mesmo tempo, o que pode causar lentidão, travamentos e até quedas.

Por conta desses motivos, não devemos utilizar o servidor de desenvolvimento disponível pelo Django para nada além do… desenvolvimento! Ele é incrivelmente útil enquanto programamos, mas não é considerável em ambiente de produção. Mas se é assim, como devemos estruturar nosso servidor remoto?

## O fluxo de conexão de uma aplicação Django

No geral, qualquer aplicação web precisa de um serviço de servidor responsável por atender as requisições [HTTP](https://www.alura.com.br/artigos/qual-e-diferenca-entre-http-e-https), respondendo com conteúdo estático e/ou dinâmico, se integrando com outros serviços. Temos muitos exemplos desse tipo de servidor web, como o [**Apache**](https://httpd.apache.org/), o mais conhecido, e o **[Nginx](https://nginx.org/en/" \t "_blank)**, também muito utilizado. No nosso caso, usaremos o **Nginx**, por ser leve e rápido.

O Nginx vai funcionar muito bem para requisições de imagens e outros arquivos estáticos. Mas e nossa aplicação Django em si? Precisamos de algo para executar todo o código em Python, e esse tipo de servidor web não consegue fazer isso. A resposta mais óbvia talvez seja usar o próprio Python puro para isso, mas ele também não é o melhor para lidar com todos os tipos de requisições. E aí?

Assim, para requisições que precisam ser geradas dinamicamente, usamos, geralmente, um servidor de aplicação. O Nginx, caso necessário, vai passar a requisição para esse servidor, que, lidando com ela, vai devolver uma resposta ao servidor web, que, finalmente, responderá diretamente ao cliente.

Esses dois servidores se comunicam, por padrão, através de uma interface especificada no [**PEP 333**](https://www.python.org/dev/peps/pep-0333/), denominada [**WSGI**](https://pt.wikipedia.org/wiki/Web_Server_Gateway_Interface). Existem diversos serviços de servidores de aplicação que suportam essa interface, como o [mod\_wsgi](http://modwsgi.readthedocs.io/" \t "_blank) e o **[Gunicorn](http://gunicorn.org/" \t "_blank)**, que será o que usaremos, por ser bem rápido e escrito puramente em Python. Vamos começar cuidando do Nginx, para então partir para o Gunicorn.

## Instalando e configurando o Nginx

Antes de tudo, precisamos instalar o Nginx em nossa máquina. Para isso, podemos desativar ( deactivate) nossa virtualenv. Instalar o pacote é fácil com o apt:

sudo apt-get install nginx

*Em sistemas que não utilizam do apt, esse processo também não deve ser nada difícil, provavelmente apenas usando um comando diferente.*

Com o Nginx instalado, podemos logo inicializá-lo, como fizemos com o MySQL:

sudo service nginx start

Agora podemos passar para a parte de configuração. Precisamos explicar para o Nginx como queremos que ele lide com as requisições.

Para isso, vamos criar um arquivo de configuração específica para nosso projeto no diretório sites-available do Nginx. Vamos nos mover para esse diretório com o comando cd:

cd /etc/nginx/sites-available

Utilizando o apt, provavelmente conseguimos entrar no diretório que queríamos, porque o pacote do Nginx mantido por lá automaticamente vem com as pastas sites-available e sites-enabled. Entretanto, em outros sistemas é possível que isso resulte em um erro indicando que essa pasta não existe. Se for o caso, a resolução é simples - podemos criar esses dois diretórios manualmente:

sudo mkdir /etc/nginx/sites-available

sudo mkdir /etc/nginx/sites-enabled

Com as duas pastas criadas, precisamos abrir o arquivo de configuração geral do Nginx:

sudo nano /etc/nginx/nginx.conf

Agora basta adicionar uma linha **dentro do bloco http { }**:

include /etc/nginx/sites-enabled/\*;

Agora sim (caso essas pastas não tenham sido criadas automaticamente), podemos entrar na pasta sem problemas:

cd /etc/nginx/sites-available

Vamos criar nosso arquivo de configuração do projeto, então:

sudo nano agenda

Nosso arquivo de configuração de projeto do Nginx, que será incluído dentro do bloco http { } nas configurações gerais, tem uma estrutura básica, com um bloco server {}. Dentro desse bloco, vamos definir a variável server\_name para especificar a que sites estamos tratando:

server {

server\_name meudominio.com.br www.meudominio.com.br;

}

*meudominio deve ser alterado para o domínio que registramos e/ou o IP do servidor*

Além disso, vamos adicionar um bloco location { } **dentro do nosso bloco server { }**, especificando como o Nginx deve responder para requisições em localizações específicas. No nosso caso, queremos que para qualquer localização a partir da raiz /, a requisição seja encaminhada para o Gunicorn.

Ainda não configuramos o Gunicorn, mas podemos definir agora mesmo que queremos que ele rode no próprio localhost (127.0.0.1) na porta **8000**. Assim, adicionamos o seguinte ao bloco server { }:

location / {

proxy\_pass http://127.0.0.1:8000;

}

Legal! O único problema é que quando o Nginx enviar a requisição para o Gunicorn, o endereço de IP e o hostname específicos dessa requisição se tornarão o localhost e o hostname do nosso servidor. Para resolver isso, basta adicionar algumas linhas no bloco location / { }. Ao final, o arquivo fica assim:

server {

server\_name meudominio.com.br www.meudominio.com.br;

location / {

proxy\_pass http://127.0.0.1:8000;

proxy\_set\_header X-Forwarded-Host $server\_name;

proxy\_set\_header X-Real-IP $remote\_addr;

add\_header P3P 'CP="ALL DSP COR PSAa PSDa OUR NOR ONL UNI COM NAV"';

}

}

Certo! Temos a configuração de nosso site disponível. Precisamos avisar o Nginx que este site não só está disponível, como também ativo, e para isso vamos para o diretório sites-enabled:

cd /etc/nginx/sites-enabled

Dentro da pasta, vamos criar um link com a configuração que acabamos de criar, usando o comando ln no terminal:

sudo ln -s /etc/nginx/sites-available/agenda

Se nosso Nginx já veio com a pasta sites-enabled, provavelmente há um arquivo padrão que não queremos no diretório. Para removê-lo, basta o comando rm:

sudo rm default

Ao final, podemos reiniciar o Nginx, para aplicar as mudanças, com o comando:

sudo service nginx restart

Agora, podemos partir para o Gunicorn.

## Instalando e configurando o Gunicorn

Com o Gunicorn, as coisas ficam bem mais simples. Primeiramente, por ser um pacote Python, podemos ir lá para o diretório de nossa aplicação e ativar o virtualenv:

cd ~/aplicacoesdjango2/agenda

source ../bin/activate

Para instalar o Gunicorn, basta usar o pip:

pip install gunicorn

Agora já podemos iniciá-lo. É importante que lembremos de usar o mesmo endereço de IP e a mesma porta que especificamos lá para o Nginx:

gunicorn -b 127.0.0.1:8000 agenda.wsgi

Finalmente, podemos acessar nosso site pelo navegador, através da porta padrão do HTTP. Para testar nossa aplicação de agenda de contatos, resolvi tentar adicionar alguns contatos através da página de **admin** (http://meudominio.com.br/admin):



Ué! Que página feia! Quando testei no meu próprio computador não ficou assim… Dei uma olhada no meu terminal rodando o Gunicorn e olha o que apareceu:

Not Found: /static/admin/css/base.css

Not Found: /static/admin/css/login.css

Not Found: /static/admin/css/responsive.css

Aparentemente os arquivos estáticos de CSS que personalizavam essa página não foram encontrados na URL /static/admin/css/.... De fato, esses arquivos não existem (ainda!), muito menos nessas URLs. E agora?

## Coletando e configurando os arquivos estáticos

O Django tem, por padrão, alguns arquivos estáticos que melhoram a usabilidade padrão do framework. Coletá-los é simples, podemos fazer com apenas um comando com o manage.py. Vamos fechar o Gunicorn com **CTRL-C** e cuidar disso, então.

Antes, porém, o Django precisa saber onde queremos guardar esses nossos arquivos. Eu decidi deixar em uma pasta static/, fora da aplicação Django, mas dentro do ambiente virtual.

Além disso, quero que qualquer arquivo dentro desta página apareça como na URL meudominio.com.br/static/. Assim, modifiquei as últimas linhas do meu arquivo agenda/settings.py e deixei da seguinte forma:

STATIC\_ROOT = '/home/yan/aplicacoesdjango2/static/'

STATIC\_URL = '/static/'

Agora sim, podemos simplesmente rodar o comando no terminal:

python manage.py collectstatic

Rapidamente, os arquivos vão ser todos copiados:

118 static files copied to '/home/yan/aplicacoesdjango2/static'.

Certo! Agora já temos os arquivos que precisamos. Mas espera… como o Nginx vai saber que quando a requisição for para a URL meudominio.com.br/static/\* ele deve buscar nessa pasta que criamos? Precisamos avisá-lo!

Vamos voltar para a edição de nosso arquivo de configuração do projeto do Nginx:

sudo nano /etc/nginx/sites-available/agenda

Agora, dentro do bloco server { }, vamos precisar de mais um bloco location { } específico para as URLs /static/, e que funcione como um atalho para nossa pasta lá no ambiente virtual. Assim, podemos adicionar isso no arquivo:

location /static/ {

alias /home/yan/aplicacoesdjango2/static/;

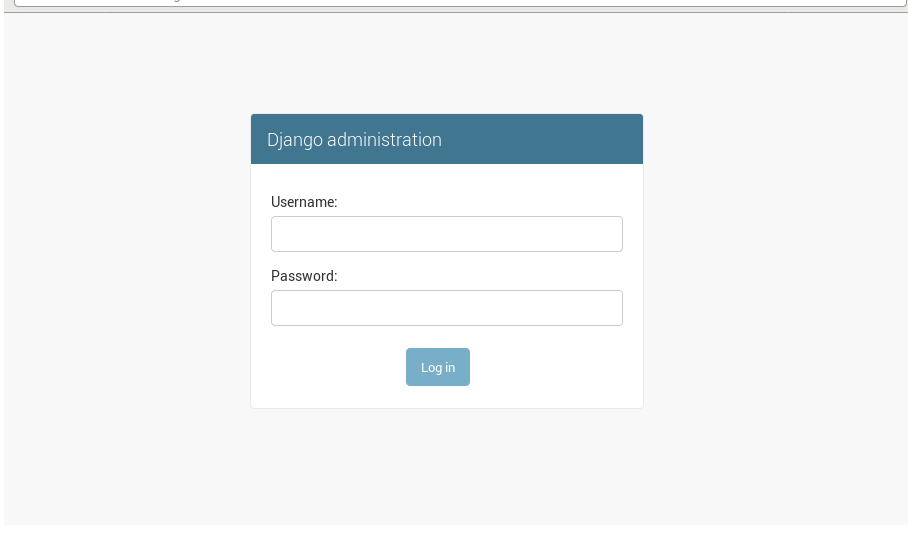
}

Feito! Podemos reiniciar o Nginx e executar novamente o Gunicorn:

sudo service nginx restart

gunicorn -b 127.0.0.1:8000 agenda.wsgi

Dessa vez, olha como ficou a página de admin:



Muito mais bonito, com o CSS carregado! Agora é só digitarmos nosso usuário e senha e… mas quando definimos isso tudo? Ainda não configuramos nosso usuário de administrador da aplicação! Para isso, podemos fechar o Gunicorn e rodar um simples comando:

python manage.py createsuperuser

Seguindo as instruções, tudo deve estar certo!

Apenas para finalizar, a **última** coisa que não podemos esquecer é de **alterar o valor da variável DEBUG para False no agenda/settings.py, agora que estamos a nível de produção**. Assim, podemos rodar novamente o Gunicorn que tudo funcionará como deve!

## Trabalhando com o Django com clareza

Fazer o deploy de uma aplicação Django não precisa ser estressante, como vimos hoje! Apesar do Django ser, em geral, menos utilizado que certas outras ferramentas, ele continua sendo incrivelmente poderoso, como evidenciam sites como o [Instagram](https://thenewstack.io/instagram-makes-smooth-move-python-3/). Assim, usá-lo (e usá-lo bem) pode ser uma ótima opção para o seu projeto!

Após toda a programação, chega sempre um dos momentos mais temidos para os desenvolvedores - o deploy. Apesar dos mitos, hoje aprendemos passo a passo como colocar no ar nossa aplicação Django, de forma simples e eficiente. Daqui para frente não teremos mais esse problema!

Além disso que aprendemos hoje, é bacana saber que temos mais opções que podem nos ajudar com o deploy de aplicações, como o [**Docker**](https://www.alura.com.br/curso-online-docker-e-docker-compose).

Se quiser continuar estudando sobre o assunto, dê uma olhada em nossos cursos na Alura de **[Django](https://www.alura.com.br/curso-online-introducao-ao-django)** e **[Python](https://www.alura.com.br/cursos-online-programacao/python)** e continue aprendendo!

COOKIES

# O que são cookies e como eles funcionam?

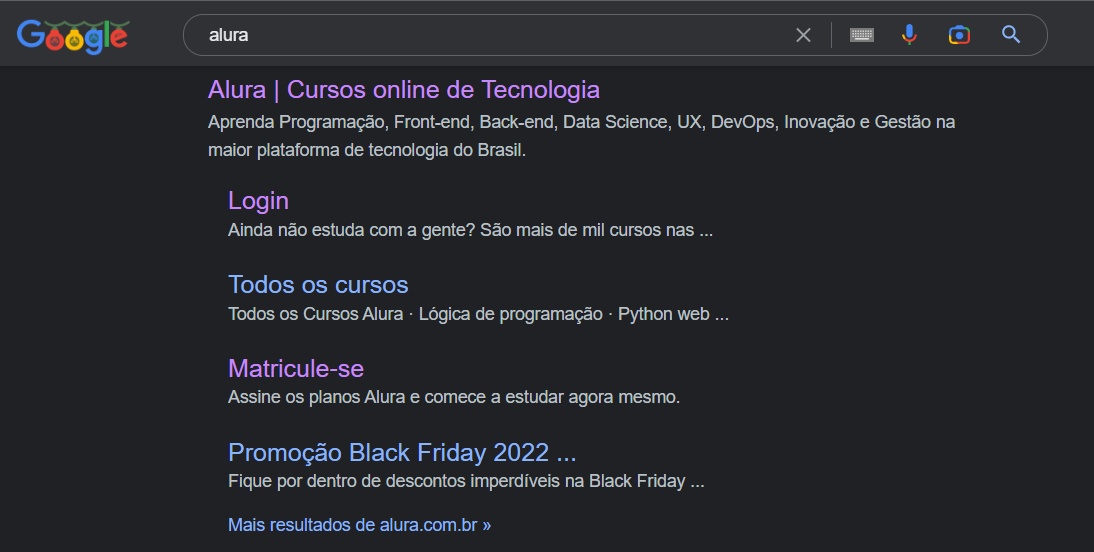
## Introdução

Você certamente já se deparou, ao acessar um site, com uma mensagem desse tipo: “Esse site usa cookies para gerar uma melhor experiência, você concorda com nossa política de uso?”.

É possível que você tenha clicado em “concordo” sem saber muito bem o que isso significa e ainda com pressa para tirar aquele banner do site, correto?

Pois saiba que a ideia de “melhorar sua experiência”, de uma forma simplificada, significa: lembrar de você!

Mas você deve estar se perguntando: “como assim lembrar de mim?”. “Lembrar”, neste contexto, significa guardar informações de sua navegação em um site. Os cookies são pequenos pacotes de **arquivos de texto** que informam ao navegador que você já acessou determinado link. Como na imagem abaixo, que mostra minha busca no Google por “alura” e os links das páginas que já acessei aparecem com uma cor diferente:



Fonte: elaboração própria

## Qual o objetivo dos cookies?

Além de “lembrar” dos links que você acessou, os cookies podem guardar outras informações da sua navegação na web, por exemplo, se você assistiu a um vídeo, quanto tempo passou na página, qual o idioma de sua preferência, ou quais foram suas buscas em um site. Ou seja, tudo o que você fizer durante a navegação gera informações sobre você, que podem ser armazenadas em cookies.

Em um e-commerce, os cookies podem manter os itens selecionados no seu carrinho de compras. Já em páginas web que solicitam login e senha, os dados encriptados de acesso podem ser armazenados em um cookie, sustentando sua sessão e permitindo que suas credenciais de login permaneçam válidas por mais tempo. Confira um breve resumo sobre os usos mais comuns:

* Gerenciamento de sessão:
  + Logins, status de visualização de um vídeo, manter itens em um carrinho de compras de lojas online, buscas realizadas.
* Personalização de preferências:
  + Preferência de idioma, temas de cores ou acessibilidade.
* Rastreamento:
  + Salva informações sobre o comportamento de quem acessa determinada página web, como sua frequência de cliques ou localização.

Por conta dessa funcionalidade, os cookies melhoram a experiência de navegação. Todavia, podem coletar **informações sensíveis** sobre quem está acessando determinado site ou serviço.

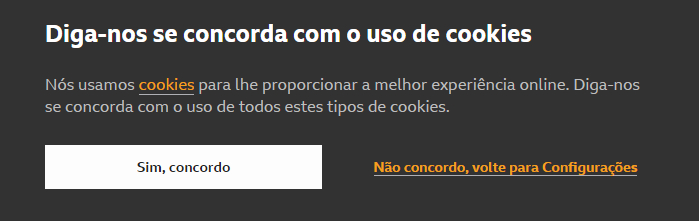


Fonte: [tumblr](https://38.media.tumblr.com/6e34bcded9dce824605275d19f3d1988/tumblr_inline_nr33tmCRwa1rbgh7e_500.gif" \t "_blank)

Sempre que o ponto “informações sensíveis” surge, uma onda de nervosismo ou algumas inquietações aparecem. Mas calma! Os dados armazenados pelo seu navegador são compartilhados com os sites apenas se você permitir. E, ao contrário do que se possa imaginar, os cookies existem desde os primórdios da internet, coletando e armazenando informações de navegação.

Porém, com a ampliação das discussões de privacidade no Brasil e no mundo, na União Européia foi estabelecida a Cookie Law (Lei de Cookies, em tradução livre) e no Brasil a LGPD (Lei Geral de Proteção de Dados), que obriga os sites a informar que utilizam cookies para armazenar suas informações e preferências. É por isso que o alerta sobre a política de uso dos cookies rotineiramente aparece quando você acessa uma página web.

Agora você não vai encarar o alerta da mesma forma, não é?



Fonte: elaboração própria

Agora que já compreendemos o que são cookies da internet, qual a função dessas mensagens de alerta e como os cookies podem ser utilizados, vamos aprofundar o assunto?

## HTTP Cookies para desenvolvimento Web

Já sabemos que os cookies são arquivos de texto que guardam informações referentes à navegação feita em um determinado site. Mas como isso funciona?

Para aprofundarmos a compreensão sobre os cookies, o primeiro ponto é entendermos que as aplicações web são [stateless](http://gabsferreira.com/o-protocolo-http-e-stateless-mas-o-que-isso-significa/" \t "_blank), ou seja, não mantém o estado após o encerramento da conexão do navegador. Cada requisição HTTP é independente e nativamente tem a função de efetuar dois procedimentos: realizar requisições do servidor web e manejar suas respostas.

Isso ocorre porque os servidores web não possuem a responsabilidade de armazenar, autenticar dados do cliente ou manter as sessões, ou seja, manter o usuário logado em determinada página a cada requisição feita. Dessa maneira, um servidor web também não consegue identificar se mais de uma requisição foi executada por um mesmo navegador e, é nesse cenário que os cookies entram em ação.

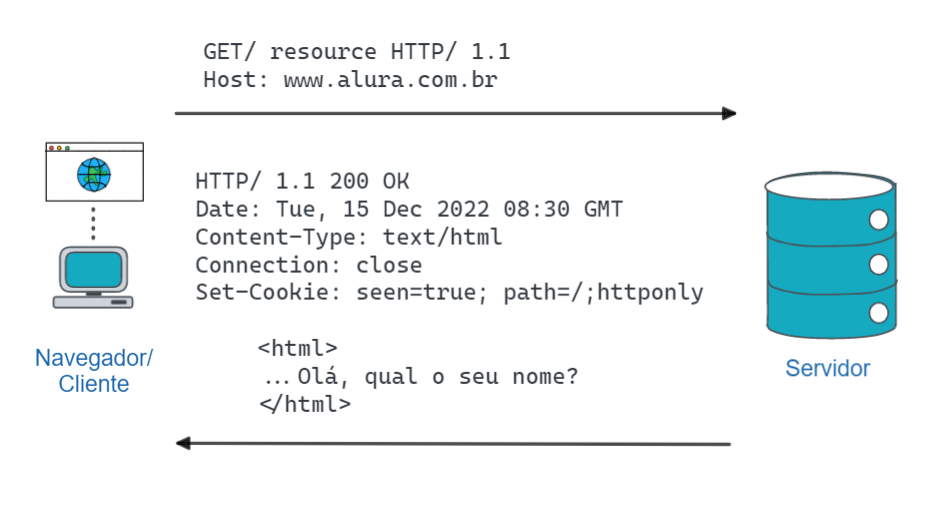


Fonte: [Shutterstock](https://www.shutterstock.com/image-vector/chocolate-chip-cookies-super-hero-260nw-1560703307.jpg" \t "_blank)

As solicitações de cookies podem ser realizadas através de um header (cabeçalho de resposta) chamado Set-Cookie, estabelecido pelo servidor. Ao acessar pela primeira vez uma página web, as informações solicitadas neste cabeçalho de resposta (que variam de acordo com o que é determinado por cada site) ficam armazenadas no seu computador de forma local. Em seguida, na próxima requisição feita pelo usuário naquela mesma página web, o navegador do cliente envia os Cookies de volta ao servidor também através do header (cabeçalho) Cookie.

Vamos analisar o esquema representado na imagem a seguir, para compreendermos melhor.

Figura 1:



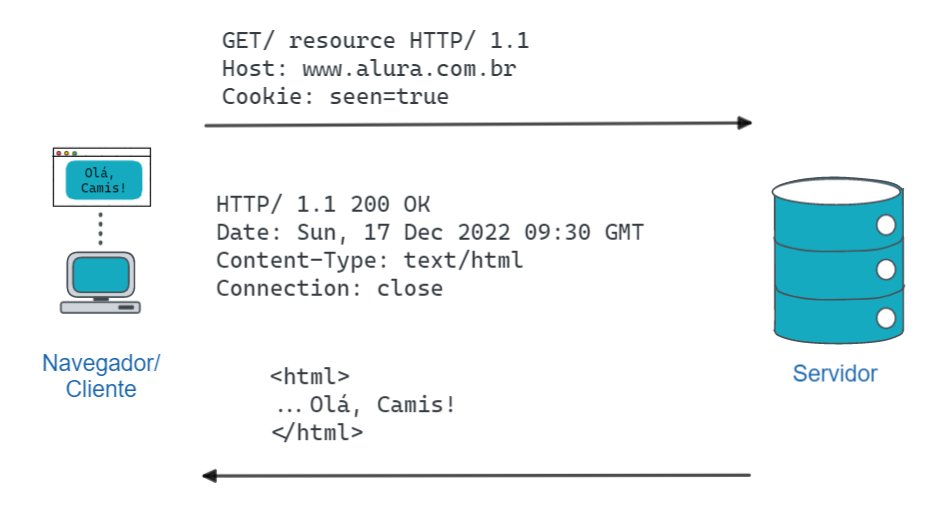
Fonte: elaboração própria, referência: [auth0](https://auth0.com/docs/manage-users/cookies/samesite-cookie-attribute-changes#features-affected)

Você sabe o que acontece quando enviamos uma requisição para o domínio www.alura.com.br?

Ao acessarmos a plataforma o servidor envia o cabeçalho Set-Cookie na resposta HTTP. A sintaxe do Set-Cookie apresenta uma cadeia de caracteres (isso mesmo, uma string!) que possui um par chave/valor e carrega seus atributos obrigatórios e opcionais. O Set-Cookie do nosso exemplo, presente na Figura 1, funciona como um pedido de apresentação e, na próxima vez que o cliente acessar o domínio pelo mesmo navegador, ele irá devolver o HTTP Cookie que será reconhecido pelo servidor.

Agora, confira a Figura 2 que ilustra a situação de um segundo acesso ao mesmo domínio:

Figura 2



Fonte: Elaboração própria. Referência: [auth0](https://auth0.com/docs/manage-users/cookies/samesite-cookie-attribute-changes#features-affected)

Ao acessar o site pela segunda vez, o cliente envia o Cookie e o servidor reconhece o usuário por meio de sua identificação (um ID único), que devolve uma resposta a partir dessas informações. Dessa maneira, todas as vezes que o cliente acessar a página, sua identificação será reconhecida e ele poderá permanecer logado (esse é apenas um exemplo de uso de cookies).

E enquanto pessoa desenvolvedora, quais tipos de informações eu posso requisitar pelo Set-Cookie? Notamos que o Set-Cookie apresenta alguns parâmetros com o formato **nome-valor** e sua estrutura pode ser definida da seguinte maneira:

Set-Cookie: <cookie-name>=<cookie-value>

Mas o que essas informações significam? A tabela abaixo descreve o que cada parâmetro significa em um Set-Cookie:

| **Chave: nome** | **Valor** |
| --- | --- |
| nome do cookie | indica a função do cookie |
| valor do cookie | informações que carrega - em string |

Além disso, também é possível enviar um ou múltiplos **atributos** via et-Cookie para armazenar outros tipos de informações, sua escolha depende do tipo de aplicação e pode estar atrelada a um único domínio. Vamos analisar um formato de Set-Cookie que é enviado no cabeçalho da resposta pelo servidor:

Set-Cookie: <cookie-name>=<cookie-value>; Domain=<domain-value>; Secure; HttpOnly

Aqui, um cookie começa com um par contendo nome-valor, depois temos o atributo opcional Domain com o par domínio-valor, que define um escopo de domínio; o atributo Secure tem como propósito impedir que os cookies sejam observados por partes não autorizadas durante sua transmissão; e o atributo HttpOnly impede o acesso de suas informações por meio da propriedade JavaScript document.cookie. Confira outros atributos:

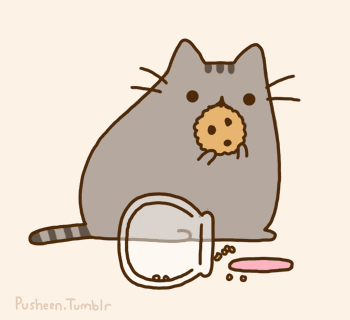
| **Atributos** | **Breve descrição** |
| --- | --- |
| Expires=<date> | Dita a data e hora que o cookie irá expirar |
| Max-Age=<number> | Número de segundos até os cookies expirarem |
| Domain=<domain-value> | Define o host ao qual o cookie será enviado |
| Path=<path-value> | Indica o caminho que deve existir na URL |
| Secure | O cookie só é enviado em requisições HTTPS - boolean |
| HttpOnly | Proíbe o acesso ao cookie com a propriedade document.cookie |

Fonte: Criação própria. Referência: MDN & [Dias de Dev](https://dias.dev/2022-09-27-cookies-e-seguranca/)

Outro atributo opcional que merece destaque, sobretudo devido a questões de segurança e ataques [CSRF](https://www.ibm.com/docs/pt-br/sva/10.0.0?topic=configuration-prevention-cross-site-request-forgery-csrf-attacks), é o SameSite=<samesite-value> que pode receber três parâmetros:

* **SameSite=Strict**: É o mesmo que SameSite (sem valor). Os cookies não serão compartilhados durante a requisição com outros sites, ou seja, é first-party e não third-party;
* **SameSite=Lax**: O parâmetro SameSite=Lax pode ser enviado quando o método HTTP é considerado o mais “seguro”. Por exemplo, quando se trata de um GET e não POST (você pode encontrar a lista dos métodos “seguros” na especificação [RFC7231](https://www.rfc-editor.org/rfc/rfc7231)).
  + Nesse contexto de criação de cookies com SameSite=Lax, devemos utilizar os métodos HTTP que são utilizados para leitura de dados, como o GET, e não os métodos que causam alguma modificação nas informações, como o POST ou PUT. Em segundo plano, a instrução deverá modificar a URL que aparece na barra de endereços do navegador, é o que chamamos de navegação “Top-Level”. De uma forma mais simples, o SameSite=Lax direciona o seu navegador para outra URL através de um link. Essa é a configuração padrão que propõe uma abordagem mais flexível (o termo “lax” tem justamente esse significado de “relaxado”, “ambíguo”), visando garantir proteção contra alguns ataques e tendo como objetivo manter a experiência de navegação para o cliente;
* **SameSite=None**: Se você estabelecer um set-cookie com o SameSite=None, isso significa que os cookies serão enviados pelo navegador tanto para requisições cross-site (sites diferentes, sites cruzados), como para same-site (do mesmo site). Para evitar problemas relacionados à segurança, os navegadores atuais exigem o parâmetro Secure, como: Set-Cookie: <cookie-name>=<cookie-value>; SameSite=None; Secure.

Nosso pote de cookies já está quase cheio, não é?



Fonte: [Giphy](https://giphy.com/gifs/food-kawaii-R52934IAVt4jK" \t "_blank)

Nós já temos uma infinidade de dados e troca de informações. Mas existe a possibilidade de fazer essa troca entre cliente e servidor com os cookies?

É possível especificar quais atributos são relevantes para sua aplicação, como, por exemplo: estipular uma data de duração ou expiração para o cookie, dentre outras informações. Além disso, os tipos de cookies têm a função de nos manter logados (ou não) após inserirmos nossas credenciais em um site. Vamos conhecer agora os tipos de Cookies.

## Tipos de Cookies

Já entendemos o que são os headers Set-Cookie, 'cookie' e como funciona a estrutura para criação de um cookie. No entanto, é essencial aprofundarmos o funcionamento dos chamados **cookies de sessão** e os **cookies permanentes**. Vamos lá?

#### Session Cookies, Cookies de Sessão

Um Cookie de sessão inicia quando o cliente acessa um site e expira (é apagado) quando o cliente encerra a conexão. Isso significa que os cookies de sessão são deletados após o fim da sessão ou quando o navegador é fechado. Também são conhecidos como temporary cookies (cookies temporários) ou non-persistent cookies (cookies não persistentes). Esse tipo de cookie é definido quando não carregamos os atributos Expires ou Max-Age.

E como os cookies de sessão funcionam?

* O servidor cria um ID único para a sessão que é gerado aleatoriamente e armazenado de forma temporária no cookie, como no exemplo: Set-Cookie: sessionId=38afes7a8;
* O cookie de sessão não pode ser compartilhado com nenhum outro cliente, pois armazena informações sobre a interação e movimentação do usuário em determinada página e nada mais;
* Um exemplo comum de cookies de sessão é o **carrinho de compras** em uma loja ou e-commerce. Os cookies vão ajudar o site a memorizar os itens selecionados, o endereço, nome e informação de pagamento. Depois de encerrada a sessão, as informações são deletadas.

#### Permanent Cookies, Cookies permanentes

Os cookies permanentes têm uma data específica para expirar com o atributo Expires. Confira o exemplo:

Set-Cookie: id=a3fWa; Expires=Wed, 25 Jan 2023 07:28:00 GMT

Ou ainda podem expirar após um intervalo de tempo, sem a necessidade do cliente interromper a sessão. Nesse caso, o atributo usado é o Max-Age. Confira a seguir um exemplo de sua aplicação:

Set-Cookie: id=a3fWa; Max-Age=2592000

Temos também o Set-Cookie com o seu nome id e seu valor. Por fim, o atributo Max-Age carregando um valor que define o período **em segundos** que o cookie irá expirar. Dessa forma, o cookie irá expirar em 2592000 segundos, que é o mesmo que 30 dias.

Os cookies permanentes armazenam as informações sobre o comportamento do usuário de forma local (no seu dispositivo) para fornecer uma melhor experiência de navegação.

A diferença entre os dois tipos de cookies encontra-se no fato de que os permanentes ou não expiram, ou têm uma data específica para isso, e são armazenados no seu dispositivo local. Por outro lado, os cookies de sessão estão armazenados na memória de forma temporária, enquanto a sessão estiver ativa ou até você fechar o seu navegador.

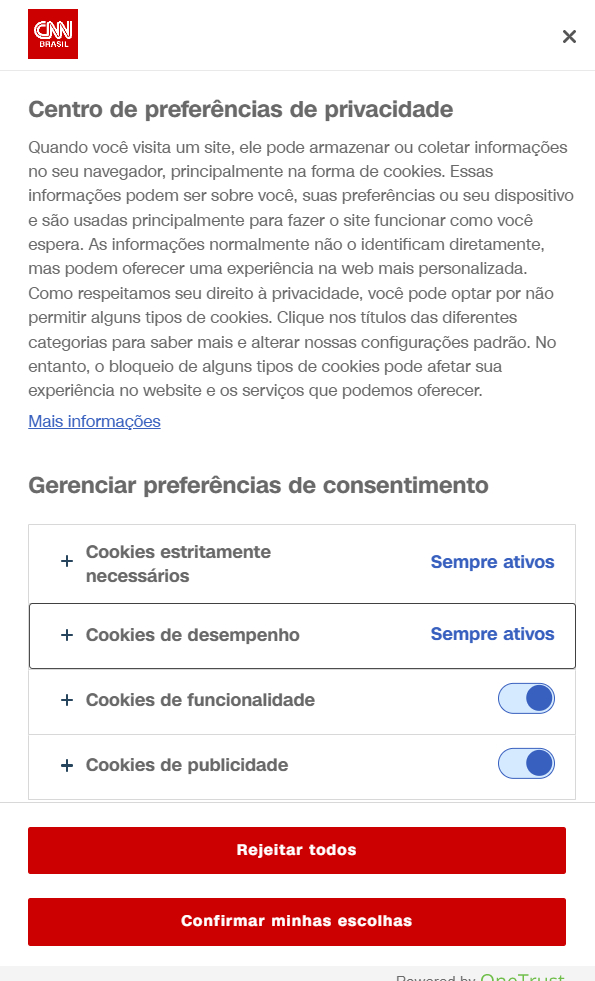
*O fluxo da operação é o mesmo para os dois tipos de cookies: cliente envia a requisição > servidor recebe e envia uma resposta com o Set-Cookie > cliente recebe a resposta e armazena em cookie.*

**A pergunta que fica é: “Devo solicitar e aceitar cookies de sessão e permanentes?”**



Fonte: [pinterest](https://ar.pinterest.com/pin/cookie-gif-find-share-on-giphy--567031409315175070/" \t "_blank)

A resposta para essa pergunta é: “depende”! Imagine que todas as vezes que você queira entrar na plataforma da Alura, ou no seu e-mail, você precise inserir seu login e sua senha. Ou mesmo quando quer assistir videoaulas no Youtube, mas acaba acessando sempre o mesmo vídeo porque não há uma sinalização de que aquele conteúdo já foi visto. Essas tarefas repetitivas podem parecer trabalhosas para muitas pessoas, e se você é uma delas, definitivamente deve aceitar os “cookies essenciais ou necessários” que aparecem nos banners ao acessar alguns sites, como o da imagem a seguir:



Fonte: elaboração própria, retirado do site <https://www.cnnbrasil.com.br/>

#### Cookies First-party e Third-party: Primários e de terceiros

Ser um cookie de sessão ou permanente, em geral, não é o real problema para uma pessoa que acessa um site pela primeira vez, ou para uma pessoa desenvolvedora.

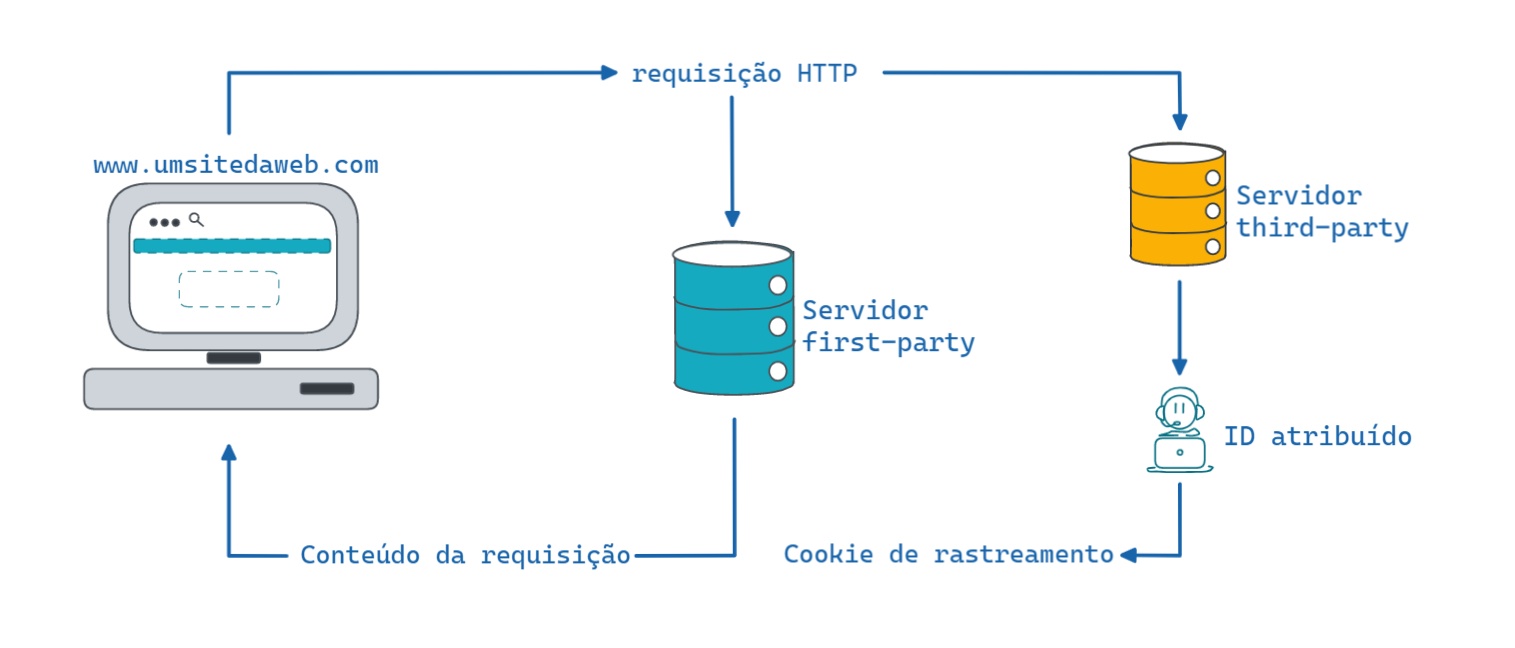
O conflito maior está relacionado ao compartilhamento de informações que envolvem a LGPD, pois essa questão faz referência ao escopo dos cookies **First-party** (Cookie primário) e **Third-party** (Cookie de terceiros). Em alguns casos há também o **Second-party**. Confira o quadro a seguir, que relaciona os tipos de cookies e seus respectivos escopos:

| **Cookies** | **Escopo** |
| --- | --- |
| Cookie primário | Um cookie primário é criado por sites que um usuário visita. Ele é usado para salvar dados, como itens de carrinho de compras e credenciais de entrada. Por exemplo, cookies de autenticação e outras análises. |
| Cookie secundário | Um cookie secundário é tecnicamente o mesmo que um cookie primário. A diferença é que os dados são compartilhados com outra parte por meio de um contrato de parceria de dados. Por exemplo, [Análise e relatórios do Microsoft Teams](https://learn.microsoft.com/pt-br/microsoftteams/teams-analytics-and-reports/teams-reporting-reference). |
| Cookie de terceiros | Um cookie de terceiros é instalado por um domínio diferente do que o usuário visitou explicitamente e, é usado para acompanhamento. Por exemplo, os botões: Curtir, serviço de anúncios e chats ao vivo. |

Fonte: [Atributo de cookie SameSite](https://learn.microsoft.com/pt-br/microsoftteams/platform/resources/samesite-cookie-update)

O quadro acima apresenta um excelente resumo. Porém, os cookies primários e de terceiros apresentam mais características essenciais. Vamos entendê-las melhor a seguir!

Já aprendemos que em termos de validade, nós podemos identificar cookies de sessão e cookies permanentes. Por outro lado, em termos de **Domínio**, podemos diferenciar os cookies como **Primários** e de **Terceiros**.



Fonte: elaboração própria. Referência: <https://devopedia.org/http-cookie>

Os Cookies Primários são enviados diretamente pelo servidor responsável pelo domínio (o site que você acessou). Esse tipo de cookie é utilizado para melhorar a sua experiência neste site específico, é aquela ideia de **memória** que já comentamos anteriormente.

Todavia, os **cookies de terceiros** são oriundos de uma terceira parte , ou seja, de outro servidor. São os botões de curtir de mídias sociais, banners com propagandas (e você nem precisa clicar nas propagandas, os cookies de terceiros já são armazenados pelo seu navegador) e imagens oriundas diretamente de outros sites.

Os **Cookies de Terceiros** podem ser nocivos em relação à segurança e uso de suas informações, pois como o próprio nome sugere, há um terceiro elemento responsável pelas diretivas de cookies e rastreamento de suas informações.

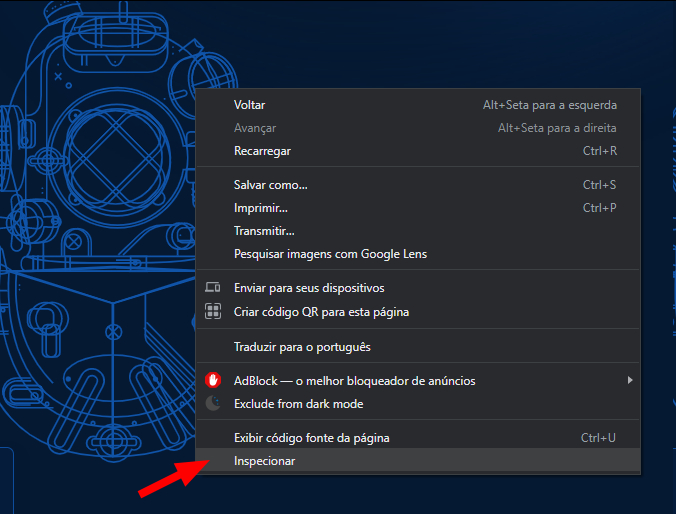
## Encontrando o pote de cookies no seu navegador!

Durante sua leitura, já conversamos sobre os tipos de cookies e acerca de sua função principal, que é armazenar informações. E se até agora você não conhecia o potencial dos cookies em relação à privacidade de informações ou segurança, talvez deseje excluir ou manipular esses pequenos arquivos. Há uma forma direta de executar essa ação, ou seja, sem precisar entrar no site e desabilitar preferências, mas acessar os cookies que foram armazenados no seu computador via navegador.

Mas onde ficam os cookies armazenados pelo meu computador?

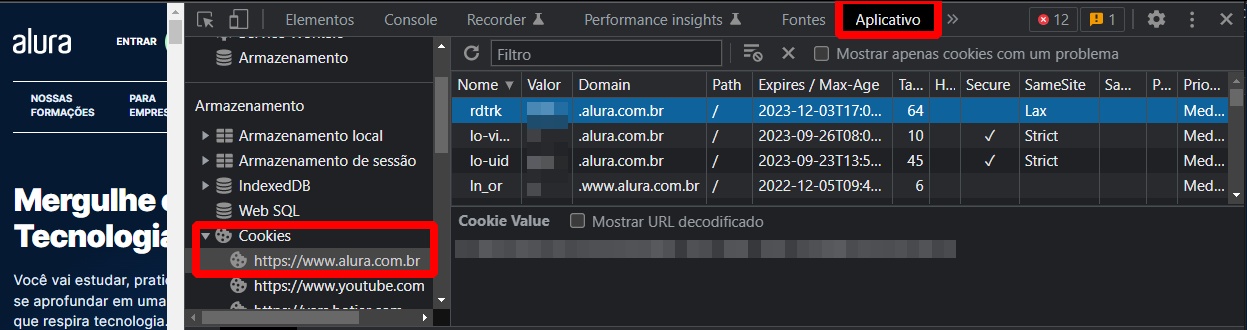
Bom, é possível encontrá-los de forma local, isso mesmo, nos diretórios de sua máquina! Porém, é bem mais rápido com o suporte das ferramentas de dev, como o DevTools do seu navegador. Para isso, siga os passos:

1) No Google Chrome, com o sistema operacional Windows, acesse o DevTools clicando com o botão direito do mouse em qualquer página da web, através da opção “inspecionar”, ou você pode clicar na tecla “F12” do seu teclado. Como exemplo, na imagem abaixo: eu abri o site www.alura.com.br > botão direito do mouse em qualquer área da tela do navegador para abrir o menu > Inspecionar:



Fonte: Elaboração própria

2) Com o DevTools aberto, procure pela opção Aplicativo (Application, em inglês), no menu de opções localizado na parte superior. Após isso, selecione a opção Cookies no menu à esquerda e acesse o domínio desejado. Na área à direita, você conseguirá visualizar os cookies relacionados a este domínio (no caso do nosso teste, alura.com.br) e todos os seus atributos.



Fonte: elaboração própria

Você pode também consultar a documentação de outros navegadores para visualizar os cookies presentes em seu computador:

* [O que é o Developer tools](https://developer.mozilla.org/pt-BR/docs/Learn/Common_questions/What_are_browser_developer_tools);
* [Abrindo as ferramentas do desenvolvedor Firefox - IBM](https://www.ibm.com/docs/pt-br/rpa/21.0?topic=tools-opening-developer-in-mozilla-firefox-browser);
* [Abrindo as ferramentas do desenvolvedor Microsoft Edge - IBM](https://www.ibm.com/docs/pt-br/rpa/21.0?topic=tools-opening-developer-in-microsoft-edge-browser);
* [Documentação Google sobre os Cookies](https://developer.chrome.com/docs/devtools/storage/cookies/?utm_source=devtools).

## Curiosidades

*O que significa cookies em português: Biscoito ou bolacha? Essa discussão não existiu durante a criação do termo cookie, que, na verdade, tem sua origem num período anterior à própria internet. Sua origem está relacionada a outro termo, os “magic cookies” (cookies mágicos, em tradução literal), que faziam referência aos pacotes de dados do sistema UNIX. Esses pacotes não sofriam alterações durante suas transmissões ou recebimentos. Então, para designar os nossos pequenos pacotes de dados na web, o programador Lou Montulli, em 1994, cunhou o termo cookies. Precisamos lembrar que Montulli foi um dos responsáveis pelas primeiras versões do Netscape Navigator e a funcionalidade dos HTTP Cookies que resistiu ao tempo, sendo relevante até hoje.*

*O Google Docs é capaz de salvar alterações em seus arquivos mesmo offline, usando cookies.*

*Cookies*third-party*estão com os dias contados devido às novas normas e reflexões sobre privacidade na internet.*

*Ainda existe o Zombie Cookie (cookie Zumbi)! Eles vêm com os cookies de terceiros e são instalados de forma permanente no dispositivo do cliente sem o seu consentimento. Foram criados quando ainda se usava plug-ins Flash nos sites e sua remoção é muito difícil, já que eles insistem em “voltar à vida”. Os cookies zumbis são utilizados para banir usuários de determinada página ou para rastreá-los. Precisamos lembrar que o suporte ao Flash foi encerrado no final de 2020, mais um motivo para manter seu navegador atualizado!*

## Conclusão

Chegamos ao final desse artigo e agora você aprendeu bastante sobre o que são cookies e para que servem, não é mesmo?!

Até aqui você compreendeu que os cookies são muito mais que pequenos arquivos para armazenar dados. Afinal, com os cookies, é possível obter informações que funcionam como “pegadas” em um caminho, e através dessas “marcas” deixadas pela web é possível garantir uma boa experiência de navegação, manter seus logins ativos, gerenciar suas preferências e rastrear seu comportamento.

Um outro exemplo, considerado por muitos como algo não tão nobre, é o scraping de seus dados **por terceiros, com o intuito de fomentar ou direcionar práticas de consumo e compras online** .

Você também entendeu como funciona a relação entre cliente e servidor no manejo de cookies e identificou o papel de cada entidade nesse processo.

Além disso, conheceu os tipos principais de cookies, que são:

* First-Party: cookies criados e compartilhados sob o mesmo domínio e servidor;
* Third-Party: cookies armazenados por uma terceira parte , ou seja, um cookie é configurado no seu navegador por um domínio diferente do que você acessou em primeiro lugar.

É importante lembrarmos que os cookies **não são usados para executar códigos, programas ou enviar vírus para o seu computador**, mas podem ser utilizados como estratégias de marketing para sugestão de conteúdos ou vendas, que são os third-party.

Agora que descobriu todas essas informações sobre cookies, você pode começar a criar os seus primeiros cookies e ampliar a discussão sobre o seu manejo, pois essa prática é essencial para o desenvolvimento web em áreas como acessibilidade, segurança e LGPD, assim como UI, marketing e publicidade.

E então, vai continuar aceitando todos os cookies? Quais os desafios e vantagens de trabalhar com cookies percebidos ao longo da leitura?

Compartilhe sua opinião em suas redes sociais e marque a gente!

# Preparando o ambiente

[**PRÓXIMA ATIVIDADE**](https://cursos.alura.com.br/course/react-integrando-projeto-react-apis/task/103948/next)

# Olá!

É muito bom receber você neste curso de React: integrando seu projeto React com APIs.

Espero que seja uma experiência de aprendizado incrível e que possamos lado a lado vencer todos os desafios.

## Preparando ambiente

Para conseguir acompanhar este curso, é necessário que você tenha o [Node](https://nodejs.org/en/) instalado.

Durante o treinamento, usarei o [VSCode](https://code.visualstudio.com/" \t "_blank) para editar o código e tornar o desenvolvimento ainda mais fácil. Neste curso, vamos disponibilizar um projeto React inicial e um back-end para disponibilizar os recursos da API.

Para baixar o projeto inicial, [clique aqui](https://github.com/alura-cursos/alfood/archive/refs/heads/main.zip). Você também pode acessar o [repositório](https://github.com/alura-cursos/alfood) se preferir.

Já no caso do back-end, [baixe aqui](https://github.com/alura-cursos/restaurantes_api/archive/refs/heads/master.zip). Mas você também pode acessar o [repositório](https://github.com/alura-cursos/restaurantes_api) se preferir.

Para focarmos 100% no desenvolvimento da aplicação React, vamos utilizar o docker para subir o servidor back-end. Sendo assim, é importante que você tenha o [Docker](https://www.docker.com/products/docker-desktop) instalado.

Caso não tenha o Docker instalado e precise de ajuda:

* [Instalando o Docker no Windows](https://cursos.alura.com.br/course/docker-e-docker-compose/task/29235)
* [Instalando o Docker no Mac/Linux](https://cursos.alura.com.br/course/docker-e-docker-compose/task/29237)

A Alura é formada por pessoas que gostam de tecnologia e acreditam no poder da educação através dela. Somos uma comunidade que ama compartilhar conhecimento. Em caso de dúvida na instalação ou durante o curso, conte sempre com o fórum. Caso não tenha dúvidas, não deixe de participar do fórum para ajudar outras pessoas e fazer da comunidade um lugar ainda melhor!