# Lembretes de Inicialização

## Iniciar o Projeto:

para iniciar o Projeto precisa rodar o código com o "." no final indicando para criar a pasta setup no mesmo diretório, caso contrário vai ficar "setup/setup"

```
django-admin startproject setup .
```

## Comandos de Adim

**lista dos comandos :**

```
django-admin  #
```

**subir servidor: basta rodar o manage.py com a opcao runserver**

```
python manage.py runserver
```

*não esqueça de tirar a SECRET_KEY das settings*

## Variaveis de ambiente e SECRET_KEY:

Na linha das settings, apague o conteudo da variavel SECRET_KEY, mantendo `SECRET_KEY = ''`

crie um arquivo na pasta Raiz chamado .env que carregará as variáveis de ambiente e nele inclua

```
SECRET_KEY = chave(sem aspas)
```

De volta ao arquivo settings.py na linha 13 começam as importações de módulos, aqui inclua o os,e o dotenv :

depois chame a função chamada `load_dotenv()`

```
from pathlib import Path, os
from dotenv import load_dotenv

load_dotenv()
```

para que sejam carregadas as variáveis de ambiente quando iniciado o servidor

No arquivo settings na linha 26 a Variável SECRET_KEY que estava vazia deve ser alterada:

```
SECRET_KEY = str(os.getenv('SECRET_KEY'))
```

para que usemos o "os" e chamar o .env com o conteúdo da variável 'SECRET_KEY', lembrando que dentro do .env a chave não deve ser uma string, já na função os.getenv('') ele irá buscar o conteudo da string 'SECRET_KEY' e depois percisamos transformar o retorno em string

# Iniciar um app:

## iniciando o app

O app seria basicamente o aplicativo/site que rodará em cima do Django.
Para iniciar um app precisamos rodar um código

```
python manage.py startapp [nome]
```

Isso cria uma pasta com o nome e com alguns arquivos como models views entre outros para controlarmos o app que funcionará no Django.

No arquivo settings.py aparecem na linha *36* aparecem os apps instalados como uma lista:

```
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
]
```

Note que ao iniciar um app com o comando anterior precisamos incluir o nome do app na lista, apenas o nome.
Como exemplo rodamos:

```
python manage.py startapp galeria
```

o app e pasta com nome de *galeria* foi criado, e vamos alterar o settings para incluir esse app nas configurações, ficando o settings.py assim:

```
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'galeria',
]
```

Nesse momento se rodarmos o servidor ainda estará igual

# Views

## Criar view

Na pasta do app galeria, existe um arquivo chamado views.py que tratará as requisições do que deveria ser mostrado em tela

No arquivo view.py precisamos importar uma forma de responder requisição que no caso seria o Módulo `HttpResponse` do próprio Django

```
from django.shortcuts import render
from django.http import HttpResponse
```

Precisamos agora criar uma página para ser exibida, e pra isso precisamos criar uma função, que nesse caso chamarei de index que recebe um argumento request, e o retorno seria um `HttpResponse` com argumento o código html da página .

```
from django.shortcuts import render
from django.http import HttpResponse
# Create your views here. {#create-your-views-here }


def index(request):
    return HttpResponse('Alura Space')
```

Criamos assim a primeira View simples mas ela ainda não aparecerá pois precisamos configurar a rota de url para que seja apresentada essa função *index* com essa view

## Rota para o Index

Dentro da pasta Setup no módulo url.py, importe a view no inicio:

```
from galeria.views import index
```

Lembrando que no exemplo o nome do app é galeria e a view está na pasta 'galeria/views'

Inclua a rota para o index dentro da lista urlpatterns com a sintaxe:

```
urlpatterns = [
    path('admin/', admin.site.urls),
    path('', index),
]
```

Lembrando que por padrão apenas o path do admin estava configurado, e que mantivemos zerado o valor do argumento '', pois seria a primeira requisição no site

# urlpatterns

## Boas Praticas:

O arquivo setup/urls deve ficar responsável por listar todas as urls de todos os aplicativos criados, logo é uma boa prática criar na página de cada aplicativo um arquivo "igual" chamado urls.py com os dados das rotas daquele aplicativo, para que o setup/urls quando acionado pela rota chame do módulo urls.py de cada aplicativo usando o metodo include().

Anteriormente o arquivo setup/urls.py estava dessa forma:

```
from django.contrib import admin
from django.urls import path

urlpatterns = [
    path('admin/', admin.site.urls),
    path('', index),
]
```

Agora ficaria asssim, importando o metodo include de django.urls, e adicionando include('url_http','caminho_urls_aplicativo'), para que quando o usuario entrar naquela url_http que no caso está em branco, o setup vai importar do aplicativo(galeria.urls).

O setup/urls fica assim:

```
from django.contrib import admin
from django.urls import path,include


urlpatterns = [
    path('admin/', admin.site.urls),
    path('', include('galeria.urls')),
]
```

E criamos o arquivo urls.py dentro da pagina inicial do aplicativo, que no caso é 'galeria', e lembrando que ele precisa importar as views para que seja renderizado pelo sistema quando for chamado. deixando o arquivo assim:

```
from django.urls import path
from galeria.views import index


urlpatterns = [
    path('', index),
]
```

### Resumo de Boas Práticas

No item anterior vale lembrar que a ideia é isolar a responsabilidade do modulo urls.py da pasta setup de manter as configurações de admin necessárias e requisitar as views diretamente dos aplicativos, levando em consideração que podemos ter mais de um, evitando assim do modulo ficar longo e bagunçado.

Cada aplicativo lida com suas views isoladamente e o setup.urls, lida como controlador de cada aplicativo, assim mantendo fácil o desenvolvimento de um aplicativo novo enquanto que na página do setup basta incluir a chamada de rotas com o include()

## Templates

### Incluir no settings

No arquivo setup/settings.py existe uma linha (58) que inclui as configurações dos templates.
A variável DIRS que por padrão é uma lista vazia, deve ser preenchida com o acminho da pasta dos nossos templates.

Crie uma pasta chamada templates fora da pasta do app, onde ficarão os templates html da aplicação.

No modulo settings.py da pasta setup, precisamos indicar a pasta dos templates, na linha 58 existe uma lista de dicionários com o nome **TEMPLATES**, a Chave 'DIRS' tem como valor uma lista vazia que devemos incluir nossa pasta de templates usando o os.path.join() ficando assim:

```
'DIRS': [os.path.join(BASE_DIR, 'templates')],
```

Após isso precisamos informar a View para renderizar esse HTML
alterando a views.py;

Onde estáva o paragrafo de retorno:

```
def index(request):
    return HttpResponse('Alura Space')
```

Ficará assim:

```
from django.shortcuts import render
# Create your views here. {#create-your-views-here-1 }


def index(request):
    return render(request, 'index.html')
```

# Arquivos Estaticos

Com o index.html criado, faltam carregar os arquivos estáticos para a página ficar com o front coerente.

Precisamos criar uma pasta chamada "static" dentro da pasta setup, de depois criar no settings.py uma linha de código de que indique essa pasta, apesar da linha não ter sido criada para que possamos apenas preencher como foi o caso do Templates DIRS, logo na linha 124(não obrigatótio) vamos criar uma variável chamada STATICFILES_DIRS que recebe uma lista, e usando o os.path.join() iremos indicar onde está a pasta.
ficando assim:

```
STATICFILES_DIRS = [
    os.path.join(BASE_DIR, 'setup/static'),
]
```

ainda após isso nas linhas abaixos iremos passar o caminho absoluto para o diretório static, que é receita de bolo e ficará assim por padrão:

```
STATICFILES_DIRS = [
    os.path.join(BASE_DIR, 'setup/static'),
]

STATIC_ROOT = os.path.join(BASE_DIR, 'static')
```

# Estilizando front

## Pastas e diretorios do front

as pastas padrões "assests" que contem imagens/icones do front e "style" que contem a formatação css, ficarão dentro da pasta static, como static/assets, e static/style

## Manipular o Static no Django

com os arquivos na pasta de imagens/icones setup/static/assets e css na pasta setup/static/style precisamos rodar um comando no terminal para que o django colete os arquivos e implemente.

como o comando de ajuda `python manage.py help` , note que existem 3 comandos na aba 'staticfiles', que seriam:

- collectstatic
- findstatic
- runserver

vamos rodar no terminal o primeiro comando, para que o Django colete todos os staticos para que o deploy quando feito o Django saiba onde achar os estaticos

No Terminal:

```
python manage.py collectstatic
```

Assim que rodar o código note que será criada uma pasta fora de Setup com o nome de static relacionada ao STATIC_ROOT e a partir de agora conseguimos manipular os arquivos estaticos

**Esse comando não faz alteração na página**

**Alterações futuras no estatico**

Deve-se sempre fazer as alterações no arquivo setup/static/

Lembre-se que a pasta static criada pelo collectstatic é apenas para armazenar os arquivos estáticos que serão servidos pelo Django.

Essa pasta funciona como se fosse uma maneira mais fácil para o Django achar os estaticos, mas para o DEV permanece como a pasta de static dentro de setup

**Toda alteração feita nos estaticos, precisam ser atualizadas pelo comando do terminal collectstatic**

## Implmentar o Static 'style'

Na página do index.html que está no templates precisamos incluir na primeira linha uma indicação que esse html possui arquivos estáticos.

```
{% load static %}
```

Posteriormente precisamos incluir um codigo python no href do css para " embedar {% %} ", nesse exemplo na linha 13 do index.html está assim:

```
link rel="stylesheet" href="/styles/style.css"
```

Ficará assim:

```
link rel="stylesheet" href="{% static '/styles/style.css' %}"
```

Nesse momento, o style.css já estilizou o index.html faltando fazer o mesmo com os assets para que as imagens e ícones apareçam

## Implementar o Static 'asset'

Nessa hora para carretagar as imagens corretamente precisamos incluir "{% static 'caminho asset' %}" em cada linha do index que possua asset manualmente.

Exemplo na linha 19 temos o código:

```
img src="/assets/logo/Logo(2).png" alt="Logo da Alura Space"
```

Deverá ser alterado com o prefixo "{ static ' bem como trocar as aspar duplas por aspas simples no inicio de src=' e no final incluir '% }", ficando dessa forma :

```
img src="{% static '/assets/logo/Logo(2).png' %}" alt="Logo da Alura Space"
```