

## Sumário

Ambiente de desenvolvimento.....	1
Virtualenv .....	1
Servidor.....	2
Idioma e Timezone .....	3
Variáveis de ambiente.....	3
Git e Github .....	4

## Ambiente de desenvolvimento

Nesse treinamento, vamos utilizar o [VSCode](#) como editor de código e o [GitHub](#) para o versionamento de código.

Será necessário o instalador de pacotes **pip**. Abra o CMD/prompt ou terminal, conforme o seu sistema operacional e use os seguintes códigos:

Para sistemas Windows:

```
curl https://bootstrap.pypa.io/get-pip.py -o get-pip.py
python get-pip.py
```

Ao decorrer do curso iremos utilizar um ambiente virtualizado chamado [virtualenv](#) para construir os projetos da aula. Você pode instalar a **virtualenv** com o seguinte comando:

```
pip install virtualenv
```

## Virtualenv

Agora vamos isolar todas as dependências do projeto. Essa é uma boa prática de programação. Faremos isso com o comando `virtualenv venv`. Depois disso, será criada uma pasta chamada "venv", com duas pastas dentro dela, "bin" e "lib". Precisaremos, agora, ativar a *venv* para que consigamos continuar a utilizar o projeto.

Na pasta "bin", há o arquivo "activate". No terminal, vamos fazer o caminho até chegar a esse arquivo. No *macOS*, utilizamos o comando `source venv/bin/activate`. Já no *Windows*, o comando, apesar de similar, é diferente: `venv\Scripts\Activate`.

Nota: Se quisermos desativá-lo futuramente, basta executar o comando `deactivate`

Agora precisamos instalar o *Django*. Faremos isso usando com a ajuda do *Pip*, um programa de gerenciamento de pacotes do *Python*.

Basta rodar o comando `pip install django`

## Servidor

# Pip freeze

Agora que o *Django* está instalado, vamos executar uma boa prática em programação.

Existe uma forma de visualizar todas as dependências do projeto e os módulos que precisam ser instalados para que o projeto funcione. Para fazer isso, executamos o comando `pip freeze` no terminal. Ele informa a versão de *Django*, *sqlparse* e *asgireft* necessárias ao funcionamento do projeto. **Uma boa prática é criar um arquivo, chamado "requirements.txt", contendo todos os arquivos necessários.**

Fazemos isso executando o comando `pip freeze > requirements.txt`.

Sempre que instalarmos algo, como o Django e o Flask, por exemplo, precisamos executar o comando `pip freeze > requirements.txt` para manter o arquivo atualizado.

## Carregando o Django

Agora, vamos carregar o *Django* na nossa aplicação. Antes, porém, vamos executar o comando `django-admin help` no terminal. Com isso, poderemos visualizar todos os comando que podemos executar na ferramenta.

Dentre eles, usaremos **startproject** para iniciar nosso projeto. Vamos executar o comando `django-admin startproject setup .`

Fazendo isso, será criada uma pasta chamada "setup", dentro de "alura-space", com as configurações do projeto.

Se não adicionássemos o `.` em `django-admin startproject setup .`, seria criada outra pasta "setup", dentro de "setup", com as configurações.

## Subindo servidor

**Python manage.py runserver**

## Idioma e Timezone

As configurações relacionadas ao idioma da aplicação ficam em "setup > settings.py".

Nesse arquivo, encontramos todas as configurações do projeto, como dependências, *templates* e mais. Portanto, vamos usá-lo muito em nossas operações, porque precisaremos manipular essas configurações. Nas linhas 106 e 108 do código do arquivo, encontramos "LANGUAGE\_CODE" e "TIME\_ZONE", exatamente o que procurávamos.

Ao lado de "LANGUAGE\_CODE", encontraremos 'en-us', que representa o inglês dos Estados Unidos. Vamos substituir para 'pt-br'. Com isso, vamos alterar a linguagem para português brasileiro. Em "TIME\_ZONE", usaremos 'America/Sao Paulo'. Isso será o suficiente para ajustar nosso fuso horário.

## Variáveis de ambiente

Agora que o projeto está configurado com idioma e horário, precisamos versionar o projeto usando *Git* e *Github*.

Porém, não podemos enviar todas as partes do código para essas ferramentas, por questões de segurança. Se enviássemos o código por inteiro, receberíamos uma mensagem do *Github*, avisando que uma *Django Secret Key* está acessível no *Github*. A linha que contém a *Secret Key*, chave secreta, está em "setup > settings.py".

Todo projeto *Django* tem uma chave secreta. Essa *key* não deve ser disponibilizada para outras pessoas.

No terminal, vamos solicitar que o *Pip* instale uma nova dependência, com o comando `pip install python-dotenv`. **Depois da alteração, vamos atualizar o arquivo de *Requirements* com `pip freeze > requirements.txt`.**

Vamos criar um novo arquivo na pasta "Django\_4", chamado ".env". Nele, vamos inserir a *Secret Key*, seguindo o modelo abaixo:

**SECRET\_KEY = sua secret key**

Agora, podemos remover a *Secret Key* do arquivo "settings.py"

Vamos carregar, agora, as variáveis de ambiente.

Na linha 13, onde importamos `Path`, vamos importar também o `os`, utilizando `from dotenv import load_dotenv`.

Abaixo, criaremos a função `load_dotenv()`

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

load_dotenv()
```

Agora, vamos descer até a linha 26 do código, onde antes havia a *Secret Key*. Vamos passar a variável transformada em *string*

`str(os.getenv('SECRET_KEY'))` no lugar da *Secret Key*.

Ex:

```
# SECURITY WARNING: keep the secret key used in production secret!  
SECRET_KEY = str(os.getenv('SECRET_KEY'))
```

Agora, quando enviarmos para o *Github*, ele apresentará a string `str(os.getenv('SECRET_KEY'))` ao invés da *Secret Key*, que está armazenada somente no nosso computador. Isso acontece porque nós não enviamos o arquivo `.env` para nosso repositório.

## Git e Github

Enviando todo o código para o Github

Para isso, criaremos um novo repositório na página do nosso *Github*, clicando em "+ > New repository". Daremos o nome "alura\_space" a ele. Agora já temos o local para onde enviaremos o código.

**Dica:** Diversas pessoas que programam com *Django* têm isso como rotina. Por isso, já existe um padrão para fazer isso. Vamos acessar [gitignore.io](https://gitignore.io). O site nos perguntará em qual linguagem estamos codando. Portanto, vamos digitar "Django". Receberemos a definição de cada arquivo que não podemos enviar para o *Github*. Vamos apertar "Ctrl + A", para selecionar tudo, e "Ctrl + C", para copiar. De volta ao código, apertaremos "Ctrl + V". Com isso, copiaremos todas as informações para o nosso ".gitignore".

Vamos iniciar um repositório local com a ajuda do comando `git init`. Depois, vamos executar `git add .`, que copiará tudo o que precisamos.

Os arquivos que não serão enviados ao *Github* serão exibidos na cor cinza na barra de exibição lateral. Vamos realizar o *commit* com `git commit -m "projeto alura space`. Depois disso o commit será criado.

Vamos acessar o repositório que criamos no *Github*. Nele, na seção "*...or create a new repository on the command line*", copiaremos a linha que traz, além de `git remote add origin`, o link do repositório. No caso do instrutor, o comando era `git remote add origin https://github.com/guilhermeonrails/alura_space.git`, mas o link será diferente para cada repositório.

Vamos copiar esse comando e executá-lo no terminal. Depois, executaremos `git push origin master`.

master


1 branch

0 tags

Go to file

Add file






Code

vivianeads

Projeto Space

ff0b415 1 hour ago

1 commit

 setup	Projeto Space	1 hour ago
 .gitignore	Projeto Space	1 hour ago
 Django_4.docx	Projeto Space	1 hour ago
 manage.py	Projeto Space	1 hour ago
 requirements.txt	Projeto Space	1 hour ago

Help people interested in this repository understand your project by adding a README.

Add a README