Sumário

[**Ambiente de desenvolvimento** 1](#_Toc118639400)

[**Virtualenv** 1](#_Toc118639401)

[**Servidor** 2](#_Toc118639402)

[**Idioma e Timezone** 3](#_Toc118639403)

[**Variáveis de ambiente** 3](#_Toc118639404)

[**Git e Github** 4](#_Toc118639405)

**Ambiente de desenvolvimento**

Nesse treinamento, vamos utilizar o [**VSCode**](https://code.visualstudio.com/download) como editor de código e o [**GitHub**](https://github.com/) 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](https://bootstrap.pypa.io/get-pip.py%20-o%20get-pip.py)

python get-pip.py

Ao decorrer do curso iremos utilizar um ambiente virtualizado chamado [**virtualenv**](https://github.com/pypa/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, precsamos 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 uma 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()Interface gráfica do usuário, Texto, Aplicativo

Descrição gerada automaticamente

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:

**Texto

Descrição gerada automaticamente**

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