RaissaAzevedo

TRABALHANDO COM TRADUÇÃO DE PROJETOS DJANGO

Porque utilizar projetos em diferente idiomas ?

Devido a localização geográfica do Brasil, a probabilidade de fazer negócio com os países mais próximos é maior, e ao redor do país, todos os outros países possuem como língua oficial o Espanhol. Então, para englobar a importância do Projeto, é interessante trabalhar pelo menos com os idiomas inglês e espanhol.

Obs: A própria documentação do Django oferece suporte para tradução. Ou seja, ele permite que o desenvolvimento do site permite ofertar a língua de acordo com o usuário utilizando o website.

O `USE_I18N = True` presente no settings.py, permite que haja o carregamento de todas as ferramentas de internacionalização.

Existe outra ferramenta que ajuda no processo de tradução:

```
pip3 install textblob
```

Esse módulo irá ajudar na tradução do projeto para venda em outros países.

É possível verificar acionando o console python.

```
python3
f>>> from textblob import TextBlob
>>> texto=TextBlob('Evolua seu lado Geek!')
>>> texto.translate(to='es')
TextBlob("¡Desarrolla tu lado friki!")
>>> texto.translate(to='en')
TextBlob("Evolve your Geek side!")
>>> texto.translate(to='fr')
TextBlob("Faites évoluer votre côté Geek!")
>>> texto.translate(to='ru')
TextBlob("Развивайте свою сторону компьютерных фанатов!")
>>> texto.translate(to='zh-CN')
TextBlob(" 进化您的怪胎一面！ ")
>>> texto.translate(to='ar')
TextBlob("تـطور جانبـك المهوس الـخاص بـك!")
>>>
```

É possível imprimir somente o texto também, sem a presença do TextBlob no inicio:

```
>>> print(texto.translate(to='ar'))
تطور جانب المهوس الخاص بك!
```

O TextBlob tem o limite de 5 mil palavras para tradução.

INSTALANDO E CONFIGURANDO O GETTEXT

Abra o terminal

```
sudo apt-get install build-essential curl file git
/bin/bash -c "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/master/install.sh)"
```

Depois da instalação feita, é importante disponibilizar o programa pro sistema, seguindo os passos:

```
test -d ~/.linuxbrew && eval $(~/.linuxbrew/bin/brew shellenv)
test -d /home/linuxbrew/.linuxbrew && eval $(/home/linuxbrew/.linuxbrew/bin/brew shellenv)
test -r ~/.bash_profile && echo "eval" ($(brew --prefix)/bin/brew shellenv)" >>~/.bash_profile
echo "eval $($(brew --prefix)/bin/brew shellenv)" >>~/.profile
```

Após isso, verificar se os sistema está funcionando com:

```
brew install hello
brew upgrade
brew install gettext
brew link gettext --force
```

Em seguida, outra ferramenta será necessária:

o **Poedit** que é utilizado para fazer tradução de textos em desenvolvimento.

```
sudo snap install poedit
```

A tradução de texto ocorrerá de linha a linha no momento da produção.

TRADUZINDO UMA STRING NA VIEW:

No diretório rais do projeto criar um novo diretório chamado “locale”

No settings.py:

```
LOCALE_PATHS = (
    os.path.join(BASE_DIR, 'locale')
)
```

Em MIDDLEWARE adicionar:

```
'django.middleware.locale.LocaleMiddleware',
```

Isso irá auxiliar a automatização da detecção de linguagens através do navegador.

No views.py da aplicação do projeto, fazer os seguintes imports:

```
from django.utils.translation import gettext as _
from django.utils import translation
```

Criar uma variável no context da views:

```
def get_context_data(self, **kwargs):
    context = super(IndexView, self).get_context_data(**kwargs)
    lang = translation.get_language()
    context['servicos'] = Servico.objects.order_by('?').all()
    context['funcionarios'] = Funcionario.objects.order_by('?').all()
    context['resquerdas'] = Aesquerda.objects.order_by('?').all()
    context['rdireitas'] = Adireita.objects.order_by('?').all()
    context['lang'] - lang
    translation.activate(lang)
    return context
```

Porque criar a variável lang?

Porque no template base.html, existe uma parte que é definido o **lang** que por padrão colocamos sempre pt-br. Modificar para lang.

```
{% load static %}
<!DOCTYPE html>
<html lang="{{ lang }}">
```

Na views:

Utilizar **_( )** para todo testo que precisa ser traduzido. Funciona como se fosse uma função.

```
def form_valid(self, form,  *args, **kwargs):
    form.send_mail()
    messages.success(self.request, _('Email enviado com sucesso'))
    return super(IndexView, self).form_valid(self, *args, **kwargs)

def form_invalid(self, form, *args, **kwargs):
    messages.error(self.request, _('Erro ao enviar email'))
    return super(IndexView, self).form_invalid(self, *args, **kwargs)
```

Tradução em Forms ou Models:

Não é feito utilizando gettext, é deito dessa forma:

```
from django.utils.translation import gettext_lazy as _
```

Para formulários e Models o recomentado é utilizar o gettext_lazy.

```
class ContatoForm(forms.Form):
    nome = forms.CharField(label=_('Nome'), max_length=100)
    email = forms.EmailField(label=_('E-mail'), max_length=100)
    assunto = forms.CharField(label=_('Assunto'), max_length=100)
    mensagem = forms.CharField(label=_('Mensagem'), widget=forms.Textarea)

    def send_mail(self):
        nome = self.cleaned_data['nome']
        email = self.cleaned_data['email']
        assunto = self.cleaned_data['assunto']
        mensagem = self.cleaned_data['mensagem']

        n = _(nome)
        e = _(email)
        a = _(assunto)
        m = _(mensagem)

        conteudo = f'{n}: {nome}\n {e}: {email}\n {a}: {assunto}\n {m}: {mensagem}'
```

O forms está preparado para traduções. No models:

```
class Base(models.Model):
    criados = models.DateTimeField(_('Criação'), auto_now_add=True)
    modificado = models.DateTimeField(_('Atualização'), auto_now=True)
    ativo = models.BooleanField(_('Ativo?'), default=True)

    class Meta:
        abstract = True


class Servico(Base):
    ICONE_CHOICES = (
        ('lni-cog', _('Engrenagem')),
        ('lni-stats-up', _('Gráfico')),
        ('lni-users', _('Usuários')),
        ('lni-layers', _('Design')),
        ('lni-mobile', _('Mobile')),
        ('lni-rocket', _('Foguete')),
    )
    POSICAO_CHOICES = (
        ('E', _('a esquerda')),
        ('D', _('a dieita')),
    )
```

E no terminal configurar para qual língua deseja traduzir:

```
python3 manage.py makemessages -l es
processing locale es
```

Esse makemessages vai utilizar o get text e criar um diretório com as mensagens para tradução.

Para fazer a tradução, abrir o **Poedit:**

Clicar em editar uma tradução:

Ir até o diretório onde esta o projeto e abrir o arquivo.po;

Ele vai listar o que deve ser traduzido, fazer a tradução e salvar.

Fazendo a atualização de uma tradução antiga.

Rodar o comando makemessages de novo indicando a língua e:

```
python3 manage.py makemessages -l es
Python3   manage.py compilemessages
```

TRADUZINDO NO TEMPLATE:

A tradução irá ocorrer na página de error404.

Obs: Toda vez que realizar uma tradução, fazer isso no topo da página HTML:

```
{% extends 'base.html' %}
{% load i18n %}
{% load static %}
```

E onde houver texto:

```
<h2 class="''head-title''">Erro 404 <br>{% trans 'Página não encontrada' %}</h2>
```

E fazer esse esquema em todo texto que quiser traduzir.

Após fazer a tradução de todos os templates:

```
python3 manage.py makemessages -l es
python3   manage.py compilemessages
```

Usando o indicador da língua para qual deseja traduzir.

Após a a tradução feita no Poedit:

Usar o Compile messages.