Desenvolvimento - Contribuindo

Primeiramente, você deveria ver os meios básicos para ajudar FastAPI e pedir ajuda{.internal-link target=_blank}.

Desenvolvendo

Se você já clonou o repositório e precisa mergulhar no código, aqui estão algumas orientações para configurar seu ambiente.

Ambiente virtual com `venv`

Você pode criar um ambiente virtual em um diretório utilizando o módulo `venv` do Python:

```
$ python -m venv env
```

Isso criará o diretório `./env/` com os binários Python e então você será capaz de instalar pacotes nesse ambiente isolado.

Ativar o ambiente

Ative o novo ambiente com:

=== "Linux, macOS"

```
<div class="termy">

```console
$ source ./env/bin/activate
```

</div>
```

=== "Windows PowerShell"

```
<div class="termy">

```console
$ .\env\Scripts\Activate.ps1
```

</div>
```

=== "Windows Bash"

```
Ou se você usa Bash para Windows (por exemplo <a href="https://gitforwindows.org/" class="external-link" target="_blank">Git Bash</a>):

<div class="termy">

```console
$ source ./env/Scripts/activate
```
```

```
</div>
```

Para verificar se funcionou, use:

=== "Linux, macOS, Windows Bash"

```
<div class="termy">

```console
$ which pip

some/directory/fastapi/env/bin/pip
```

</div>
```

=== "Windows PowerShell"

```
<div class="termy">

```console
$ Get-Command pip

some/directory/fastapi/env/bin/pip
```

</div>
```

Se ele exibir o binário `pip` em `env/bin/pip` então funcionou. 🎉

!!! tip Toda vez que você instalar um novo pacote com `pip` nesse ambiente, ative o ambiente novamente.

```
Isso garante que se você usar um programa instalado por aquele pacote (como `flit`), você utilizará aquele de seu ambiente local e não outro que possa estar instalado globalmente.
```

Flit

FastAPI utiliza Flit para construir, empacotar e publicar o projeto.

Após ativar o ambiente como descrito acima, instale o `flit`:

```
$ pip install flit

---> 100%
```

Ative novamente o ambiente para ter certeza que você esteja utilizando o `flit` que você acabou de instalar (e não um global).

E agora use `flit` para instalar as dependências de desenvolvimento:

=== "Linux, macOS"

```
<div class="termy">

```console
$ flit install --deps develop --symlink

---> 100%
```

</div>
```

=== "Windows"

```
Se você está no Windows, use `--pth-file` ao invés de `--symlink`:

<div class="termy">

```console
$ flit install --deps develop --pth-file

---> 100%
```

</div>
```

Isso irá instalar todas as dependências e seu FastAPI local em seu ambiente local.

Usando seu FastAPI local

Se você cria um arquivo Python que importa e usa FastAPI, e roda com Python de seu ambiente local, ele irá utilizar o código fonte de seu FastAPI local.

E se você atualizar o código fonte do FastAPI local, como ele é instalado com `--symlink` (ou `--pth-file` no Windows), quando você rodar aquele arquivo Python novamente, ele irá utilizar a nova versão do FastAPI que você acabou de editar.

Desse modo, você não tem que "instalar" sua versão local para ser capaz de testar cada mudança.

Formato

Tem um arquivo que você pode rodar que irá formatar e limpar todo o seu código:

```
$ bash scripts/format.sh
```

Ele irá organizar também todos os seus imports.

Para que ele organize os imports corretamente, você precisa ter o FastAPI instalado localmente em seu ambiente, com o comando na seção acima usando `--symlink` (ou `--pth-file` no Windows).

Formato dos imports

Tem outro *script* que formata todos os imports e garante que você não tenha imports não utilizados:

```
$ bash scripts/format-imports.sh
```

Como ele roda um comando após o outro, modificando e revertendo muitos arquivos, ele demora um pouco para concluir, então pode ser um pouco mais fácil utilizar `scripts/format.sh` frequentemente e `scripts/format-imports.sh` somente após "commitar uma branch".

Documentação

Primeiro, tenha certeza de configurar seu ambiente como descrito acima, isso irá instalar todas as requisições.

A documentação usa MkDocs.

E existem ferramentas/*scripts* extras para controlar as traduções em `./scripts/docs.py`.

!!! tip Você não precisa ver o código em `./scripts/docs.py`, você apenas o utiliza na linha de comando.

Toda a documentação está no formato Markdown no diretório `./docs/pt/`.

Muitos dos tutoriais tem blocos de código.

Na maioria dos casos, esse blocos de código são aplicações completas que podem ser rodadas do jeito que estão apresentados.

De fato, esses blocos de código não estão escritos dentro do Markdown, eles são arquivos Python dentro do diretório `./docs_src/`.

E esses arquivos Python são incluídos/injetados na documentação quando se gera o site.

Testes para Documentação

A maioria dos testes na verdade rodam encima dos arquivos fonte na documentação.

Isso ajuda a garantir:

- Que a documentação esteja atualizada.
- Que os exemplos da documentação possam ser rodadas do jeito que estão apresentadas.
- A maior parte dos recursos é coberta pela documentação, garantida por cobertura de testes.

Durante o desenvolvimento local, existe um *script* que constrói o site e procura por quaisquer mudanças, carregando na hora:

```
$ python ./scripts/docs.py live

<span style="color: green;">[INFO]</span> Serving on http://127.0.0.1:8008
<span style="color: green;">[INFO]</span> Start watching changes
<span style="color: green;">[INFO]</span> Start detecting changes
```

Isso irá servir a documentação em `http://127.0.0.1:8008`.

Desse jeito, você poderá editar a documentação/arquivos fonte e ver as mudanças na hora.

Typer CLI (opcional)

As instruções aqui mostram como utilizar *scripts* em `./scripts/docs.py` com o programa `python` diretamente.

Mas você pode usar também Typer CLI, e você terá auto-completação para comandos no seu terminal após instalar o *completion*.

Se você instalou Typer CLI, você pode instalar *completion* com:

```
$ typer --install-completion

zsh completion installed in /home/user/.bashrc.
Completion will take effect once you restart the terminal.
```

Aplicações e documentação ao mesmo tempo

Se você rodar os exemplos com, por exemplo:

```
$ uvicorn tutorial001:app --reload

<span style="color: green;">INFO</span>:     Uvicorn running on
http://127.0.0.1:8000 (Press CTRL+C to quit)
```

como Uvicorn utiliza por padrão a porta `8000`, a documentação na porta `8008` não dará conflito.

Traduções

Ajuda com traduções É MUITO apreciada! E essa tarefa não pode ser concluída sem a ajuda da comunidade. 🌎 🚀

Aqui estão os passos para ajudar com as traduções.

Dicas e orientações

- Verifique sempre os *pull requests* existentes para a sua linguagem e faça revisões das alterações e aprove elas.

!!! tip Você pode adicionar comentários com sugestões de alterações para *pull requests* existentes.

```
Verifique as documentações sobre <a
href="https://help.github.com/en/github/collaborating-with-issues-and-pull-
requests/about-pull-request-reviews" class="external-link" target="_blank">adicionar
revisão ao _pull request_</a> para aprovação ou solicitação de alterações.
```

- Verifique em *issues* para ver se existe alguém coordenando traduções para a sua linguagem.
- Adicione um único *pull request* por página traduzida. Isso tornará muito mais fácil a revisão para as outras pessoas.

Para as linguagens que eu não falo, vou esperar por várias pessoas revisarem a tradução antes de *mergear*.

- Você pode verificar também se há traduções para sua linguagem e adicionar revisão para elas, isso irá me ajudar a saber que a tradução está correta e eu possa *mergear*.
- Utilize os mesmos exemplos Python e somente traduza o texto na documentação. Você não tem que alterar nada no código para que funcione.
- Utilize as mesmas imagens, nomes de arquivo e links. Você não tem que alterar nada disso para que funcione.

- Para verificar o código de duas letras para a linguagem que você quer traduzir, você pode usar a Lista de códigos ISO 639-1.

Linguagem existente

Vamos dizer que você queira traduzir uma página para uma linguagem que já tenha traduções para algumas páginas, como o Espanhol.

No caso do Espanhol, o código de duas letras é `es`. Então, o diretório para traduções em Espanhol está localizada em `docs/es/`.

!!! tip A principal ("oficial") linguagem é o Inglês, localizado em `docs/en/`.

Agora rode o *servidor ao vivo* para as documentações em Espanhol:

```
// Use o comando "live" e passe o código da linguagem como um argumento de linha de
comando
$ python ./scripts/docs.py live es

<span style="color: green;">[INFO]</span> Serving on http://127.0.0.1:8008
<span style="color: green;">[INFO]</span> Start watching changes
<span style="color: green;">[INFO]</span> Start detecting changes
```

Agora você pode ir em http://127.0.0.1:8008 e ver suas mudanças ao vivo.

Se você procurar no site da documentação do FastAPI, você verá que toda linguagem tem todas as páginas. Mas algumas páginas não estão traduzidas e tem notificação sobre a falta da tradução.

Mas quando você rodar localmente como descrito acima, você somente verá as páginas que já estão traduzidas.

Agora vamos dizer que você queira adicionar uma tradução para a seção Recursos{.internal-link target=_blank}.

- Copie o arquivo em:

```
docs/en/docs/features.md
```

- Cole ele exatamente no mesmo local mas para a linguagem que você quer traduzir, por exemplo:

```
docs/es/docs/features.md
```

!!! tip Observe que a única mudança na rota é o código da linguagem, de `en` para `es`.

- Agora abra o arquivo de configuração MkDocs para Inglês em:

```
docs/en/docs/mkdocs.yml
```

- Procure o lugar onde `docs/features.md` está localizado no arquivo de configuração. Algum lugar como:

```
site_name: FastAPI
# Mais coisas
nav:
- FastAPI: index.md
```

```
- Languages:
  - en: /
  - es: /es/
- features.md
```

- Abra o arquivo de configuração MkDocs para a linguagem que você está editando, por exemplo:

```
docs/es/docs/mkdocs.yml
```

- Adicione no mesmo local que está no arquivo em Inglês, por exemplo:

```
site_name: FastAPI
# Mais coisas
nav:
- FastAPI: index.md
- Languages:
  - en: /
  - es: /es/
- features.md
```

Tenha certeza que se existem outras entradas, a nova entrada com sua tradução esteja exatamente na mesma ordem como na versão em Inglês.

Se você for no seu navegador verá que agora a documentação mostra sua nova seção. 🎉

Agora você poderá traduzir tudo e ver como está toda vez que salva o arquivo.

Nova linguagem

Vamos dizer que você queira adicionar traduções para uma linguagem que ainda não foi traduzida, nem sequer uma página.

Vamos dizer que você queira adicionar tradução para Haitiano, e ainda não tenha na documentação.

Verificando o link acima, o código para "Haitiano" é `ht`.

O próximo passo é rodar o *script* para gerar um novo diretório de tradução:

```
// Use o comando new-lang, passe o código da linguagem como um argumento de linha de comando
$ python ./scripts/docs.py new-lang ht

Successfully initialized: docs/ht
Updating ht
Updating en
```

Agora você pode verificar em seu editor de código o mais novo diretório criado `docs/ht/`.

!!! tip Crie um primeiro *pull request* com apenas isso, para iniciar a configuração da nova linguagem, antes de adicionar traduções.

```
Desse modo outros poderão ajudar com outras páginas enquanto você trabalha na
primeira. 🚀
```

Inicie traduzindo a página principal, `docs/ht/index.md`.

Então você pode continuar com as instruções anteriores, para uma "Linguagem Existente".

Nova linguagem não suportada

Se quando rodar o *script* do *servidor ao vivo* você pega um erro sobre linguagem não suportada, alguma coisa como:

```
 raise TemplateNotFound(template)
jinja2.exceptions.TemplateNotFound: partials/language/xx.html
```

Isso significa que o tema não suporta essa linguagem (nesse caso, com um código falso de duas letras `xx`).

Mas não se preocupe, você pode configurar o tema de linguagem para Inglês e então traduzir o conteúdo da documentação.

Se você precisar fazer isso, edite o `mkdocs.yml` para sua nova linguagem, teremos algo como:

```yaml
site_name: FastAPI
# Mais coisas
theme:
  # Mais coisas
  language: xx
```

Altere essa linguagem de `xx` (do seu código de linguagem) para `en`.

Então você poderá iniciar novamente o *servidor ao vivo*.

Pré-visualize o resultado

Quando você usa o *script* em `./scripts/docs.py` com o comando `live` ele somente exibe os arquivos e traduções disponíveis para a linguagem atual.

Mas uma vez que você tenha concluído, você poderá testar tudo como se parecesse *online*.

Para fazer isso, primeiro construa toda a documentação:

```
// Use o comando "build-all", isso leverá um tempinho
$ python ./scripts/docs.py build-all

Updating es
Updating en
Building docs for: en
Building docs for: es
Successfully built docs for: es
Copying en index.md to README.md
```

Isso gera toda a documentação em `./docs_build/` para cada linguagem. Isso inclui a adição de quaisquer arquivos com tradução faltando, com uma nota dizendo que "esse arquivo ainda não tem tradução". Mas você não tem que fazer nada com esse diretório.

Então ele constrói todos aqueles *sites* independentes MkDocs para cada linguagem, combina eles, e gera a saída final em `./site/`.

Então você poderá "servir" eles com o comando `serve`:

```
// Use o comando "serve" após rodar "build-all"
$ python ./scripts/docs.py serve

Warning: this is a very simple server. For development, use mkdocs serve instead.
This is here only to preview a site with translations already built.
Make sure you run the build-all command first.
Serving at: http://127.0.0.1:8008
```

Testes

Tem um *script* que você pode rodar localmente para testar todo o código e gerar relatórios de cobertura em HTML:

```
$ bash scripts/test-cov-html.sh
```

Esse comando gera um diretório `./htmlcov/`, se você abrir o arquivo `./htmlcov/index.html` no seu navegador, poderá explorar interativamente as regiões de código que estão cobertas pelos testes, e observar se existe alguma região faltando.

Testes no seu editor

Se você quer usar os testes integrados em seu editor adicione `./docs_src` na sua variável `PYTHONPATH`.

Por exemplo, no VS Code você pode criar um arquivo `.env` com:

```
PYTHONPATH=./docs_src
```