開発 - 貢献

まず、FastAPIを応援 - ヘルプの入手{.internal-link target=_blank}の基本的な方法を見て、ヘルプを得た方がいいかもしれません。

開発

すでにリポジトリをクローンし、コードを詳しく調べる必要があるとわかっている場合に、環境構築のためのガイドラインをいくつか紹介します。

`venv` を使用した仮想環境

Pythonの `venv` モジュールを使用して、ディレクトリに仮想環境を作成します:

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

これにより、Pythonバイナリを含む `./env/` ディレクトリが作成され、その隔離された環境にパッケージのインストールが可能になります。

仮想環境の有効化

新しい環境を有効化するには:

=== "Linux, macOS"

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

=== "Windows PowerShell"

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

=== "Windows Bash"

もしwindows用のBash（例えば、Git Bash）を使っているなら:

```console
```

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

```

動作の確認には、下記を実行します:

=== "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>

`env/bin/pip` に `pip` バイナリが表示される場合は、正常に機能しています。🎨

!!! tip "豆知識" この環境で `pip` を使って新しいパッケージをインストールするたびに、仮想環境を再度有効化します。

これにより、そのパッケージによってインストールされたターミナルのプログラム（`flit`など）を使用する場合、ローカル環境のものを使用し、グローバルにインストールされたものは使用されなくなります。

Flit

FastAPIはFlitを使って、ビルド、パッケージ化、公開します。

上記のように環境を有効化した後、`flit` をインストールします:

```
$ pip install flit

---> 100%
```

次に、環境を再び有効化して、インストールしたばかりの `flit` (グローバルではない) を使用していることを確認します。
```

そして、`flit`を使用して開発のための依存関係をインストールします:

=== "Linux, macOS"

```
<div class="termy">

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

---> 100%
```

</div>
```

=== "Windows"

```
Windowsユーザーは、`--symlink`のかわりに`--pth-file`を使用します:

<div class="termy">

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

---> 100%
```

</div>
```

これで、すべての依存関係とFastAPIを、ローカル環境にインストールします。

ローカル環境でFastAPIを使う

FastAPIをインポートして使用するPythonファイルを作成し、ローカル環境で実行すると、ローカルのFastAPIソースコードが使用されます。

そして、`--symlink`(Windowsでは`--pth-file`)でインストールされているローカルのFastAPIソースコードを更新した場合、そのPythonファイルを再度実行すると、更新したばかりの新しいバージョンのFastAPIが使用されます。

これにより、ローカルバージョンを「インストール」しなくても、すべての変更をテストできます。

コードの整形

すべてのコードを整形してクリーンにするスクリプトがあります:

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

また、すべてのインポートを自動でソートします。

正しく並べ替えるには、上記セクションのコマンドで`--symlink`(Windowsの場合は`--pth-file`)を使い、FastAPIをローカル環境にインストールしている必要があります。

インポートの整形

他にも、すべてのインポートを整形し、未使用のインポートがないことを確認するスクリプトがあります:

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

多くのファイルを編集したり、リバートした後、これらのコマンドを実行すると、少し時間がかかります。なので `scripts/format.sh` を頻繁に使用し、 `scripts/format-imports.sh` をコミット前に実行する方が楽でしょう。

ドキュメント

まず、上記のように環境をセットアップしてください。すべての必要なパッケージがインストールされます。

ドキュメントは、MkDocsを使っています。

そして、翻訳を処理するためのツール/スクリプトが、 `./scripts/docs.py` に用意されています。

!!! tip "豆知識" `./scripts/docs.py` のコードを見る必要はなく、コマンドラインからただ使うだけです。

すべてのドキュメントが、Markdown形式で `./docs/en/` ディレクトリにあります。

多くのチュートリアルには、コードブロックがあります。

ほとんどの場合、これらのコードブロックは、実際にそのまま実行できる完全なアプリケーションです。

実際、これらのコードブロックはMarkdown内には記述されておらず、 `./docs_src/` ディレクトリのPythonファイルです。

そして、これらのPythonファイルは、サイトの生成時にドキュメントに含まれるか/挿入されます。

ドキュメントのテスト

ほとんどのテストは、実際にドキュメント内のサンプルソースファイルに対して実行されます。

これにより、次のことが確認できます:

- ドキュメントが最新であること。
- ドキュメントの例が、そのまま実行できること。
- ほとんどの機能がドキュメントでカバーされており、テスト範囲で保証されていること。

ローカル開発中に、サイトを構築して変更がないかチェックするスクリプトがあり、ライブリロードされます:

```
$ 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
```

ドキュメントは、 `http://127.0.0.1:8008` で提供します。

そうすることで、ドキュメント/ソースファイルを編集し、変更をライブで見ることができます。

Typer CLI (任意)

ここでは、 `./scripts/docs.py` のスクリプトを `python` プログラムで直接使う方法を説明します。

ですがTyper CLIを使用して、インストール完了後にターミナルでの自動補完もできます。

Typer CLIをインストールする場合、次のコマンドで補完をインストールできます:

```
$ typer --install-completion

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

アプリとドキュメントを同時に

以下の様にサンプルを実行すると:

```
$ uvicorn tutorial001:app --reload

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

Uvicornはデフォルトでポート `8000` を使用するため、ポート `8008` のドキュメントは衝突しません。

翻訳

翻訳のヘルプをとても歓迎しています！これはコミュニティの助けなしでは成し遂げられません。 🌎🚀

翻訳を支援するための手順は次のとおりです。

豆知識とガイドライン

- あなたの言語の今あるプルリクエストを確認し、変更や承認をするレビューを追加します。

!!! tip "豆知識" すでにあるプルリクエストに修正提案つきのコメントを追加できます。

```
修正提案の承認のために<a href="https://help.github.com/en/github/collaborating-with-
issues-and-pull-requests/about-pull-request-reviews" class="external-link"
target="_blank">プルリクエストのレビューの追加</a>のドキュメントを確認してください。
```

- issuesをチェックして、あなたの言語に対応する翻訳があるかどうかを確認してください。
- 翻訳したページごとに1つのプルリクエストを追加します。これにより、他のユーザーがレビューしやすくなります。

私が話さない言語については、他の何人かが翻訳をレビューするのを待って、マージします。

- 自分の言語の翻訳があるかどうか確認し、レビューを追加できます。これにより、翻訳が正しく、マージできることがわかります。
- 同じPythonの例を使用し、ドキュメント内のテキストのみを翻訳してください。何も変更する必要はありません。
- 同じ画像、ファイル名、リンクを使用します。何も変更する必要はありません。
- 翻訳する言語の2文字のコードを確認するには、表 ISO 639-1コードのリストが使用できます。

すでにある言語

スペイン語の様に、既に一部のページが翻訳されている言語の翻訳を追加したいとしましょう。

スペイン語の場合、2文字のコードは `es` です。したがって、スペイン語のディレクトリは `docs/es/` です。

!!! tip "豆知識" メイン(「公式」)言語は英語で、 `docs/en/` にあります。

次に、ドキュメントのライブサーバーをスペイン語で実行します:

```
// コマンド"live"を使用し、言語コードをCLIに引数で渡します。
$ 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
```

これでhttp://127.0.0.1:8008 を開いて、変更を確認できます。

FastAPI docs Webサイトを見ると、すべての言語にすべてのページがあります。しかし、一部のページは翻訳されておらず、翻訳の欠落ページについて通知があります。

しかし、このようにローカルで実行すると、翻訳済みのページのみが表示されます。

ここで、セクションFeatures{.internal-link target=_blank}の翻訳を追加するとします。

- 下記のファイルをコピーします:

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

- 翻訳したい言語のまったく同じところに貼り付けます。例えば:

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

!!! tip "豆知識" パスとファイル名の変更は、 `en` から `es` への言語コードだけであることに注意してください。

- ここで、英語のMkDocs構成ファイルを開きます:

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

- 設定ファイルの中で、 `docs/features.md` が記述されている箇所を見つけます:

```
site_name: FastAPI
# More stuff
nav:
- FastAPI: index.md
- Languages:
  - en: /
  - es: /es/
- features.md
```

- 編集している言語のMkDocs構成ファイルを開きます。例えば:

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

- 英語とまったく同じ場所に追加します。例えば:

```
site_name: FastAPI
# More stuff
nav:
- FastAPI: index.md
- Languages:
  - en: /
  - es: /es/
- features.md
```

他のエントリがある場合は、翻訳を含む新しいエントリが英語版とまったく同じ順序になっていることを確認してください。

ブラウザにアクセスすれば、ドキュメントに新しいセクションが表示されています。 🎉

これですべて翻訳して、ファイルを保存した状態を確認できます。

新しい言語

まだ翻訳されていない言語の翻訳を追加したいとしましょう。

クレオール語の翻訳を追加したいのですが、それはまだドキュメントにありません。

上記のリンクを確認すると、「クレオール語」のコードは `ht` です。

次のステップは、スクリプトを実行して新しい翻訳ディレクトリを生成することです:

```
// コマンド「new-lang」を使用して、言語コードをCLIに引数で渡します
$ python ./scripts/docs.py new-lang ht

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

これで、新しく作成された `docs/ht/` ディレクトリをコードエディターから確認できます。

!!! tip "豆知識" 翻訳を追加する前に、これだけで最初のプルリクエストを作成し、新しい言語の設定をセットアップします。

そうすることで、最初のページで作業している間、誰かの他のページの作業を助けることができます。 🚀

まず、メインページの `docs/ht/index.md` を翻訳します。

その後、「既存の言語」で、さきほどの手順を続行してください。

まだサポートされていない新しい言語

ライブサーバースクリプトを実行するときに、サポートされていない言語に関するエラーが発生した場合は、次のように表示されます:

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

これは、テーマがその言語をサポートしていないことを意味します (この場合は、 `xx` の2文字の偽のコード) 。

ただし、心配しないでください。テーマ言語を英語に設定して、ドキュメントの内容を翻訳できます。

その必要がある場合は、新しい言語の `mkdocs.yml` を次のように編集してください:

```yaml
site_name: FastAPI
# More stuff
theme:
  # More stuff
  language: xx
```

その言語を `xx` (あなたの言語コード) から `en` に変更します。

その後、ライブサーバーを再起動します。

結果のプレビュー

`./scripts/docs.py` のスクリプトを `live` コマンドで使用すると、現在の言語で利用可能なファイルと翻訳のみが表示されます。

しかし一度実行したら、オンラインで表示されるのと同じように、すべてをテストできます。

このために、まずすべてのドキュメントをビルドします:

```
// 「build-all」コマンドは少し時間がかかります。
$ 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
```

これで、言語ごとにすべてのドキュメントが `./docs_build/` に作成されます。

これには、翻訳が欠落しているファイルを追加することと、「このファイルにはまだ翻訳がない」というメモが含まれます。ただし、そのディレクトリで何もする必要はありません。

次に、言語ごとにこれらすべての個別のMkDocsサイトを構築し、それらを組み合わせて、 `./site/` に最終結果を出力します。

これは、コマンド `serve` で提供できます:

```
// 「build-all」コマンドの実行の後に、「serve」コマンドを使います
$ 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
```

テスト

すべてのコードをテストし、HTMLでカバレッジレポートを生成するためにローカルで実行できるスクリプトがあります:

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

このコマンドは `./htmlcov/` ディレクトリを生成します。ブラウザでファイル `./htmlcov/index.html` を開くと、テストでカバーされているコードの領域をインタラクティブに探索できます。それによりテストが不足しているかどうか気付くことができます。