Development - Contributing

First, you might want to see the basic ways to help FastAPI and get help{.internal-link target=_blank}.

Developing

If you already cloned the repository and you know that you need to deep dive in the code, here are some guidelines to set up your environment.

Virtual environment with `venv`

You can create a virtual environment in a directory using Python's `venv` module:

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

That will create a directory `./env/` with the Python binaries and then you will be able to install packages for that isolated environment.

Activate the environment

Activate the new environment with:

=== "Linux, macOS"

~~~
<div class="termy">

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

</div>
~~~

=== "Windows PowerShell"

~~~
<div class="termy">

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

</div>
~~~

=== "Windows Bash"

~~~
Or if you use Bash for Windows (e.g. <a href="https://gitforwindows.org/"
class="external-link" target="_blank">Git Bash</a>):

<div class="termy">

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

````
</div>
````

To check it worked, 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>
~~~

If it shows the `pip` binary at `env/bin/pip` then it worked. 🎉

!!! tip Every time you install a new package with `pip` under that environment, activate the environment again.

```
This makes sure that if you use a terminal program installed by that package (like
`flit`), you use the one from your local environment and not any other that could be
installed globally.
```

Flit

FastAPI uses Flit to build, package and publish the project.

After activating the environment as described above, install `flit`:

```
$ pip install flit

---> 100%
```

Now re-activate the environment to make sure you are using the `flit` you just installed (and not a global one).

And now use `flit` to install the development dependencies:

=== "Linux, macOS"

~~~
<div class="termy">

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

---> 100%
```

</div>
~~~

=== "Windows"

~~~
If you are on Windows, use `--pth-file` instead of `--symlink`:

<div class="termy">

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

---> 100%
```

</div>
~~~

It will install all the dependencies and your local FastAPI in your local environment.

Using your local FastAPI

If you create a Python file that imports and uses FastAPI, and run it with the Python from your local environment, it will use your local FastAPI source code.

And if you update that local FastAPI source code, as it is installed with `--symlink` (or `--pth-file` on Windows), when you run that Python file again, it will use the fresh version of FastAPI you just edited.

That way, you don't have to "install" your local version to be able to test every change.

Format

There is a script that you can run that will format and clean all your code:

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

It will also auto-sort all your imports.

For it to sort them correctly, you need to have FastAPI installed locally in your environment, with the command in the section above using `--symlink` (or `--pth-file` on Windows).

Docs

First, make sure you set up your environment as described above, that will install all the requirements.

The documentation uses MkDocs.

And there are extra tools/scripts in place to handle translations in `./scripts/docs.py`.

!!! tip You don't need to see the code in `./scripts/docs.py`, you just use it in the command line.

All the documentation is in Markdown format in the directory `./docs/en/`.

Many of the tutorials have blocks of code.

In most of the cases, these blocks of code are actual complete applications that can be run as is.

In fact, those blocks of code are not written inside the Markdown, they are Python files in the `./docs_src/` directory.

And those Python files are included/injected in the documentation when generating the site.

Docs for tests

Most of the tests actually run against the example source files in the documentation.

This helps making sure that:

- The documentation is up to date.
- The documentation examples can be run as is.
- Most of the features are covered by the documentation, ensured by test coverage.

During local development, there is a script that builds the site and checks for any changes, live-reloading:

```
$ 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
```

It will serve the documentation on `http://127.0.0.1:8008`.

That way, you can edit the documentation/source files and see the changes live.

Typer CLI (optional)

The instructions here show you how to use the script at `./scripts/docs.py` with the `python` program directly.

But you can also use Typer CLI, and you will get autocompletion in your terminal for the commands after installing completion.

If you install Typer CLI, you can install completion with:

```
$ typer --install-completion

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

Apps and docs at the same time

If you run the examples with, e.g.:

```
$ uvicorn tutorial001:app --reload

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

as Uvicorn by default will use the port `8000`, the documentation on port `8008` won't clash.

Translations

Help with translations is VERY MUCH appreciated! And it can't be done without the help from the community. 🌎 🚀

Here are the steps to help with translations.

Tips and guidelines

- Check the currently existing pull requests for your language and add reviews requesting changes or approving them.

!!! tip You can add comments with change suggestions to existing pull requests.

```
Check the docs about <a href="https://help.github.com/en/github/collaborating-with-
issues-and-pull-requests/about-pull-request-reviews" class="external-link"
target="_blank">adding a pull request review</a> to approve it or request changes.
```

- Check in the issues to see if there's one coordinating translations for your language.

- Add a single pull request per page translated. That will make it much easier for others to review it.

For the languages I don't speak, I'll wait for several others to review the translation before merging.

- You can also check if there are translations for your language and add a review to them, that will help me know that the translation is correct and I can merge it.

- Use the same Python examples and only translate the text in the docs. You don't have to change anything for this to work.

- Use the same images, file names, and links. You don't have to change anything for it to work.

- To check the 2-letter code for the language you want to translate you can use the table List of ISO 639-1 codes.

Existing language

Let's say you want to translate a page for a language that already has translations for some pages, like Spanish.

In the case of Spanish, the 2-letter code is `es`. So, the directory for Spanish translations is located at `docs/es/`.

!!! tip The main ("official") language is English, located at `docs/en/`.

Now run the live server for the docs in Spanish:

```
// Use the command "live" and pass the language code as a CLI argument
$ 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
```

Now you can go to http://127.0.0.1:8008 and see your changes live.

If you look at the FastAPI docs website, you will see that every language has all the pages. But some pages are not translated and have a notification about the missing translation.

But when you run it locally like this, you will only see the pages that are already translated.

Now let's say that you want to add a translation for the section Features{.internal-link target=_blank}.

- Copy the file at:

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

- Paste it in exactly the same location but for the language you want to translate, e.g.:

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

!!! tip Notice that the only change in the path and file name is the language code, from `en` to `es`.

- Now open the MkDocs config file for English at:

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

- Find the place where that `docs/features.md` is located in the config file. Somewhere like:

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

- Open the MkDocs config file for the language you are editing, e.g.:

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

- Add it there at the exact same location it was for English, e.g.:

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

Make sure that if there are other entries, the new entry with your translation is exactly in the same order as in the English version.

If you go to your browser you will see that now the docs show your new section. 🎉

Now you can translate it all and see how it looks as you save the file.

New Language

Let's say that you want to add translations for a language that is not yet translated, not even some pages.

Let's say you want to add translations for Creole, and it's not yet there in the docs.

Checking the link from above, the code for "Creole" is `ht`.

The next step is to run the script to generate a new translation directory:

```
// Use the command new-lang, pass the language code as a CLI argument
$ python ./scripts/docs.py new-lang ht

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

Now you can check in your code editor the newly created directory `docs/ht/`.

!!! tip Create a first pull request with just this, to set up the configuration for the new language, before adding translations.

```
That way others can help with other pages while you work on the first one. 🚀
```

Start by translating the main page, `docs/ht/index.md`.

Then you can continue with the previous instructions, for an "Existing Language".

New Language not supported

If when running the live server script you get an error about the language not being supported, something like:

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

That means that the theme doesn't support that language (in this case, with a fake 2-letter code of `xx`).

But don't worry, you can set the theme language to English and then translate the content of the docs.

If you need to do that, edit the `mkdocs.yml` for your new language, it will have something like:

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

Change that language from `xx` (from your language code) to `en`.

Then you can start the live server again.

Preview the result

When you use the script at `./scripts/docs.py` with the `live` command it only shows the files and translations available for the current language.

But once you are done, you can test it all as it would look online.

To do that, first build all the docs:

```
// Use the command "build-all", this will take a bit
$ 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
```

That generates all the docs at `./docs_build/` for each language. This includes adding any files with missing translations, with a note saying that "this file doesn't have a translation yet". But you don't have to do anything with that directory.

Then it builds all those independent MkDocs sites for each language, combines them, and generates the final output at `./site/`.

Then you can serve that with the command `serve`:

```
// Use the command "serve" after running "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
```

Tests

There is a script that you can run locally to test all the code and generate coverage reports in HTML:

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

This command generates a directory `./htmlcov/`, if you open the file `./htmlcov/index.html` in your browser, you can explore interactively the regions of code that are covered by the tests, and notice if there is any region missing.