Cookiecutter Fastapi

Tobi DEGNON

May 13, 2024

CONTENTS

A Cookiecutter template for fastapi projects, inspired by cookiecutter-django.

Read the full documentation

CHAPTER

ONE

FEATURES

- fastapi-users for users authentication and management
- Pydantic for settings management
- Include a cli tool built with typer to simplify project management
- Pre-commit integration included by default
- Tortoise-orm and aerich database setup by default but switchable
- Limit-offset pagination helpers included
- Sending emails using aiosmtplib or Amazon SES
- Optional integration with sentry for error logging
- Production Dockerfile included
- Integration with saq for background tasks
- Optional setup of HTML templates rendering using jinja2
- Procfile for deploying to heroku
- Implement the Health Check API patterns on your fastapi application

1.1 ORM/ODM options

- Tortoise ORM
- Beanie

CHAPTER

TWO

USAGE

Install the cookiecutter package:

```
pip install cookiecutter black isort
```

Note: `Black` and `isort` are used to format your project right after it has been generated.

Now run it against this repo:

```
cookiecutter https://github.com/Tobi-De/cookiecutter-fastapi
```

You'll be prompted for some values. Provide them, then a fastapi project will be created for you.

You could also use cruft to be able to keep your project up to date with the latest changes:

```
pip install cruft
cruft create https://github.com/Tobi-De/cookiecutter-fastapi
```

It work pretty much the same way as cookiecutter but you'll be able to update your project with the latest changes by running `cruft update`.

CHAPTER

THREE

CONTRIBUTING

Contributions are very welcome. To learn more, see the *Contributor Guide*.

CHAPTER

FOUR

LICENSE

Distributed under the terms of the *MIT license*, *Cookiecutter Fastapi* is free and open source software.

CHAPTER

FIVE

ISSUES

If you encounter any problems, please file an issue along with a detailed description.

5.1 User Guide

5.1.1 Introduction: Structuring of API

- `app`: Contains all the API related code base.
  - `core`: Contains core modules of your application.
    - `auth.py`: Contains authentication configuration based on fastapi-users.
    - `config.py`: Contains main global settings of your application.
    - `logger.py`: Logging module for application.
    - `pagination.py`: Contains simple helpers to apply limit-offset based pagination on your api routes.
  - `db`: Main package for your database / orm configuration.
    - `config.py`: Main module for your database configuration.
    - `models.py`: Contains some common abstract base models for your **apps** database models. Read the *WHY* section bellow to understand what **apps** are.
  - `emails`: Contains html templates for your emails.
    - `base.html`: Base html template from which all your other email templates must inherit.
    - `welcome.html`: Welcome email template, sent when a new user registers.
  - `frontend`: Contains all the python code for your frontend. This directory is only present when you enter **y** with the **render_html** option. More details about this directory are available in the *WHY* section.
    - `routers`: All your frontend API routers.
      - `home.py`: This is an example, containing a single route to render the html of the home page.
    - `app.py`: The frontend is in fact a separate fastapi application defined in this module and mounted on the main application in the `main.py` file.
    - `utils.py`: Contains python utilities specific to frontend stuff.
  - `services`: Contains classes and functions intended to connect to external resources that your application could use.
    - `email`: Contains helper modules to connect to an email provider to send emails.
      - `errors.py`: Errors related to sending an email.

            - `null.py`: A dummy email provider when you don't want to send real emails.
            - `ses.py`: Email provider class to send email with amazon ses using aioaws. This file is only included if you choose **AMAZON SES** as **mail_service**.
            - `smtp.py`: Email provider class to send email via smtp using aiosmtplib. This file is only included if you choose **SMTP** as **mail_service**.
    - `static`: Folder to store static files, only included if you enter **y** in the **render_html** option.
        - `style.css`: Your project css.
    - `templates`: Html templates directory, only include if you enter **y** to the **render_html** option.
        - `base.html`: Base html for all your html templates.
        - `index.html`: Example html for the index page.
    - `users`: Users management app.
        - `tests`: Tests for your users' app.
            - `factories.py`: Test factories for the users models.
        - `manager.py`: User manager class, used by the fastapi-users package. You can use this concept for your other apps if you like it.
        - `models.py`: Users database models.
        - `routes.py`: Users API router and routes.
        - `schemas.py`: Users pydantic schemas.
        - `tasks.py`: Task queues tasks specific to the user's app.
        - `utils.py`: Utilities specific to the user's app.
    - `health.py`: Health API route.
    - `initial_data.py`: Contains functions that create some initial data for your application.
    - `lifetime.py`: In this module are defined the start and shutdown event handlers for your application.
    - `main.py`: Entry point of your application, the main **FastAPI** application is defined here.
    - `utils.py`: Global application utilities, which can be turned into a module later if there are too many.
    - `worker.py`: Task queue worker configuration file.
- `tests`: Your application tests.
- `.env.template`: A template to create your **.env** file.
- `.gitignore`: List common files and directories of python projects to keep out of git, read here for more details.
- `Dockerfile`: Production dockerfile if you enter **y** to the **use_docker** option.
- `.pre-commit-config.yaml`: pre-commit configuration file for auto formatting of your code on each commit.
- `manage.py`: Cli app to simplify project management, run `python manage.py --help` for all available commands.
- `Procfile`: Heroku Procfile configuration file, only present when you choose **y** to the **user_heroku** option.
- `pyproject.toml`: Application dependencies, packaging data and metadata, for more details read this.
- `README.md`: Details and setup guide for your application.
- `runtime.txt`: Heroku runtime configuration file, only present when you choose **y** to the **user_heroku** option.

- `setup.cfg`: A python configuration file for external tools like flake8, mypy etc., but I strongly recommend to put them in the `pyproject.toml` file if the tool supports it. This file will probably be removed in future versions when all tools used here add support for the `pyproject.toml` file.
- `gunicorn.conf.py`: gunicorn configuration file for deployment.

Most of the ideas and patterns that this template follows were inspired by OSS (open source software) projects and tools. I took the most interesting (from my point of view) patterns and applied them when creating this cookiecutter, and you are by no means obliged to follow 100% of the structure described here. I'm always learning new things and will improve this project over time. If you think something doesn't make sense in the files and folder structures, the code base, or if you have suggestions or improvements or anything else really, please feel free to open an issue or a discussion, I will be glad to discuss with you. This cookiecutter can't cover every use case, so here are some alternatives if this template doesn't fit your needs:

- fastapi-template
- manage-fastapi
- fastapi-nano

5.1.2 Design Decisions

n this section, we'll explore some of the design decisions made in the creation of the cookicutter-fastapi project template. We'll discuss the reasoning behind the choices made in terms of project structure, code organization, and other key aspects of the template. This section is intended to provide insight into the thought process behind the template, as well as to help users understand why certain design decisions were made.

Use of Django-style "apps" in the Template

> **TL;DR**: Features bound to the same domain in the business logic are encapsulated in an application. Applications should be small, simple and focus on a single task. E.g. the **users** app for users management.

One of the core design decisions of the `cookicutter-fastapi` project template is the use of the Django-style "apps" structure. As a Django user, I appreciate the concept of applications which is basically the clear and distinct separation of functionality into reusable packages. With this in mind, I set out to emulate this idea with FastAPI.

FastAPI does not impose any kind of structure on users, which is great for simple projects, but for more complex projects, a well-structured and organized base is necessary. If you search for FastAPI projects on GitHub, you will find a wide variety of styles and structures, each reflecting the experience and preferences of the individual developer. In the `cookicutter-fastapi` template, I have implemented a structure that represents my own experience as a Django developer, and I believe it will be familiar and intuitive to many other django developers.

The idea behind the `apps` structure is to encapsulate features that are bound to the same domain in the business logic within a single application. Each application should be small, simple, and focused on a single task. For example in the context on an ecommerce web app, a `users` application would be responsible for managing users, while an `orders` application would be responsible for managing orders. In my opinion, this structure allows a better organization and a better maintainability of the project as it grows and becomes more complex.

The frontend application

When I use fastapi, it's usually for small backend services and not for full fledge server side rendered (SSR) projects (think of an e-commerce site with django or laravel for example). I mainly use fastapi for relatively simple backend APIs and I rarely need to serve html and when I do it's for very few pages. If I needed to build a big monolith that serves html, I would probably choose django. The idea here is that the `frontend` directory is an application in your project, it will serve the few html pages you need. If you build a public weather API for example, but you need a few html pages to present the project, a home page, a contact page and maybe some user account pages, you can use the frontend app for that instead of creating a separate html/css/js or **<place your favorite js framework here>** project for that. Spreading the routes for those pages in your API routes is not a good idea in my opinion. The `frontend` application is a bit special in that it is a full Fastapi application that is mounted on top of your main application. I don't think this approach can scale to hundreds of html pages, so think twice before you consider using it for a huge project (frankly I don't know, I haven't tried it, I could be wrong).

> **Note**: Checkout htmx if you to improve your frontend development and user experience without relying on a complex SPA javascript framework.

5.1.3 Deployment

For deployment the official fastapi documentation has an excellent guide on the subject here. If you want some platform recommendation check this page.

> **Note**: This page is a work in progress, new content will be added with new release of `cookiecutter-fastapi`.

5.2 Contributor Guide

Thank you for your interest in improving cookiecutter fastapi. This project is open-source under the MIT license and welcomes contributions in the form of bug reports, feature requests, and pull requests.

Here is a list of important resources for contributors:

- Source Code
- Documentation
- Issue Tracker
- *Code of Conduct*

5.2.1 How to report a bug

Report bugs on the Issue Tracker.

When filing an issue, make sure to answer these questions:

- Which operating system and Python version are you using?
- Which version of this project are you using?
- What did you do?
- What did you expect to see?
- What did you see instead?

The best way to get your bug fixed is to provide a test case, and/or steps to reproduce the issue.

5.2.2 How to request a feature

Request features on the Issue Tracker.

5.2.3 How to set up your development environment

You need Python 3.7+ and the following tools:

- Poetry
- Nox
- nox-poetry

Install the package with development requirements:

```
$ poetry install
```

You can now run an interactive Python session, or the command-line interface:

```
$ poetry run python
$ poetry run cookiecutter-fastapi
```

5.2.4 How to test the project

Run the full test suite:

```
$ pytest
```

Unit tests are located in the *tests* directory, and are written using the pytest testing framework.

5.2.5 How to submit changes

Open a pull request to submit changes to this project.

Your pull request needs to meet the following guidelines for acceptance:

- The Nox test suite must pass without errors and warnings.
- Include unit tests. This project maintains 100% code coverage.
- If your changes add functionality, update the documentation accordingly.

Feel free to submit early, though—we can always iterate on this.

It is recommended to open an issue before starting work on anything. This will allow a chance to talk it over with the owners and validate your approach.

5.3 Contributor Covenant Code of Conduct

5.3.1 Our Pledge

We as members, contributors, and leaders pledge to make participation in our community a harassment-free experience for everyone, regardless of age, body size, visible or invisible disability, ethnicity, sex characteristics, gender identity and expression, level of experience, education, socio-economic status, nationality, personal appearance, race, caste, color, religion, or sexual identity and orientation.

We pledge to act and interact in ways that contribute to an open, welcoming, diverse, inclusive, and healthy community.

5.3.2 Our Standards

Examples of behavior that contributes to a positive environment for our community include:

- Demonstrating empathy and kindness toward other people
- Being respectful of differing opinions, viewpoints, and experiences
- Giving and gracefully accepting constructive feedback
- Accepting responsibility and apologizing to those affected by our mistakes, and learning from the experience
- Focusing on what is best not just for us as individuals, but for the overall community

Examples of unacceptable behavior include:

- The use of sexualized language or imagery, and sexual attention or advances of any kind
- Trolling, insulting or derogatory comments, and personal or political attacks
- Public or private harassment
- Publishing others' private information, such as a physical or email address, without their explicit permission
- Other conduct which could reasonably be considered inappropriate in a professional setting

5.3.3 Enforcement Responsibilities

Community leaders are responsible for clarifying and enforcing our standards of acceptable behavior and will take appropriate and fair corrective action in response to any behavior that they deem inappropriate, threatening, offensive, or harmful.

Community leaders have the right and responsibility to remove, edit, or reject comments, commits, code, wiki edits, issues, and other contributions that are not aligned to this Code of Conduct, and will communicate reasons for moderation decisions when appropriate.

5.3.4 Scope

This Code of Conduct applies within all community spaces, and also applies when an individual is officially representing the community in public spaces. Examples of representing our community include using an official e-mail address, posting via an official social media account, or acting as an appointed representative at an online or offline event.

5.3.5 Enforcement

Instances of abusive, harassing, or otherwise unacceptable behavior may be reported to the community leaders responsible for enforcement at tobidegnon@proton.me. All complaints will be reviewed and investigated promptly and fairly.

All community leaders are obligated to respect the privacy and security of the reporter of any incident.

5.3.6 Enforcement Guidelines

Community leaders will follow these Community Impact Guidelines in determining the consequences for any action they deem in violation of this Code of Conduct:

1. Correction

Community Impact: Use of inappropriate language or other behavior deemed unprofessional or unwelcome in the community.

Consequence: A private, written warning from community leaders, providing clarity around the nature of the violation and an explanation of why the behavior was inappropriate. A public apology may be requested.

2. Warning

Community Impact: A violation through a single incident or series of actions.

Consequence: A warning with consequences for continued behavior. No interaction with the people involved, including unsolicited interaction with those enforcing the Code of Conduct, for a specified period of time. This includes avoiding interactions in community spaces as well as external channels like social media. Violating these terms may lead to a temporary or permanent ban.

3. Temporary Ban

Community Impact: A serious violation of community standards, including sustained inappropriate behavior.

Consequence: A temporary ban from any sort of interaction or public communication with the community for a specified period of time. No public or private interaction with the people involved, including unsolicited interaction with those enforcing the Code of Conduct, is allowed during this period. Violating these terms may lead to a permanent ban.

4. Permanent Ban

Community Impact: Demonstrating a pattern of violation of community standards, including sustained inappropriate behavior, harassment of an individual, or aggression toward or disparagement of classes of individuals.

Consequence: A permanent ban from any sort of public interaction within the community.

5.3.7 Attribution

This Code of Conduct is adapted from the Contributor Covenant, version 2.1, available at https://www.contributor-covenant.org/version/2/1/code_of_conduct.html.

Community Impact Guidelines were inspired by Mozilla's code of conduct enforcement ladder.

For answers to common questions about this code of conduct, see the FAQ at https://www.contributor-covenant.org/faq. Translations are available at https://www.contributor-covenant.org/translations.

5.4 License