fastapi-icontract Documentation

Release 0.0.4

Marko Ristin

Jan 28, 2022

CONTENTS:

CHAPTER

ONE

INTRODUCTION

FastAPI-icontract is a FastAPI extension for design-by-contract which leverages icontract to allow you to specify and enforce code contracts in your FastAPI endpoints.

Depending on how you set it up, FastAPI-icontract will:

- automatically **enforce** the contracts during testing or in production,
- automatically add the contracts to your **OpenAPI specification**, and
- render the **Swagger UI** with a specialized contracts plugin for nicer visualization.

1.1 Benefits of Adding Contracts to Your API

Enforcing code contracts in your FastAPI development opens up new venues for approaches to more systematic design at the API level:

- Contracts are an important **part of the specification**.

  Unlike human language, contracts written in code are unambiguous.

- Contracts are **automatically verifiable**.

  Your clients can rest assured that you actually run them. FastAPI-icontract will specify precisely which contracts are run in production and which were only verified during testing.

- Contracts provide **deeper testing**.

  If you have a mesh of microservices that you need to test in conjunction, turn on all the contracts and test against your *client's* data instead of your own limited unit test data.

- Contracts specified in code allow for automatic **client-side** verification.

  Thus you can signal formally to the client up-front what you expect (using pre-conditions), while the client can verify what to expect back from you (using post-conditions).

- Contracts are **not just for input validation**.

  Though you can use contracts for input validation as well, FastAPI already allows you to specify how you want your input verified. Contracts, on the other hand, really shine when you want to specify relations *between* the endpoints.

- Contracts allow for **automatic test generation**.

  Tools for property-based testing such as Schemathesis can automatically generate test data and verify that your API works as expected. Post-conditions are an easy way to define your properties to be tested.

  There is an ongoing discussion with the authors of the Schemathesis how to integrate it with tools which generate data based on contracts such as icontract-hypothesis.

- Contracts open up **a wider ecosystem for analysis**.

  When you decorate the endpoints with contracts, you can immediately use analysis tools such as CrossHair to analyze your code and find bugs.

CHAPTER

TWO

PRE-CONDITIONS

The pre-conditions are the conditions which must hold *prior* to the execution of the endpoint.

You specify the pre-conditions using the decorator *require*.

Here is a snippet demonstrating how to specify a pre-condition (see the *full example*):

```
from fastapi_icontract import require

@app.get("/books_in_category", response_model=List[Book])
@require(
    has_category,
    status_code=404,
    description="The category must exist."
)
async def books_in_category(category: str) -> Any:
    ...
```

CHAPTER

THREE

POST-CONDITIONS

The post-conditions are the conditions which must hold *after* the execution of the endpoint.

You specify the post-conditions using the decorator *ensure*.

Post-conditions usually involve comparing the state *before* and *after* the request to an endpoint. The state *after* the request is directly available to the post-condition.

3.1 Snapshots

However, you need to specifically instruct fastapi-icontract which state needs to be capture *before* the request. This is done by using snapshots with the decorator *snapshot*.

3.2 Examples

Here is a simple snippet that involves only a post-condition (see the *full example*):

```
import asyncstdlib as a

from fastapi_icontract import ensure

@app.get("/books_in_category", response_model=List[Book])
@ensure(
    lambda result: a.all(a.await_each(has_author(book.author) for book in result)),
    description="One ore more authors of the resulting books do not exist."
)
async def books_in_category(category: str) -> Any:
    """Retrieve the books of the given category from the database."""
    ...
```

The following snippet is a rather sophisticated one that involves both the post-condition and a snapshot (see the *full example*):

```
import asynstdlib as a
from fastapi_icontract import snapshot, ensure

@app.post("/upsert_book")
@snapshot(lambda book: has_book(book.identifier), name="has_book")
@snapshot(lambda: book_count(), name="book_count")
@ensure(lambda book: has_book(book.identifier))
@ensure(
```

(continues on next page)

(continued from previous page)

```
    lambda book, OLD: a.apply(
        lambda a_book_count: (
                OLD.book_count + 1 == a_book_count if not OLD.has_book
                else OLD.book_count == a_book_count),
        book_count()))
async def add_book(book: Book) -> None:
    ...
```

CHAPTER

FOUR

ASYNC

Fastapi-icontract works out-of-the-box with the async functions in conditions and snapshot captures. If a condition or a capture returns a coroutine, the coroutine is awaited first and then tested for truthiness or captured, respectively.

However, Python 3 does not allow async lambdas (see this Python issue), so you need to use a third-party library such as asyncstdlib.

For example, note how we transform each `book` in the result in the following post-condition and verify that the author of the book exists in the system using an async function `has_author`:

```
@app.get("/books_in_category", response_model=List[Book])
@fastapi_icontract.require(
    has_category,
    status_code=404,
    description="The category must exist."
)
@fastapi_icontract.ensure(
    lambda result: a.all(await_each(has_author(book.author) for book in result)),
    description="One ore more authors of the resulting books do not exist."
)
async def books_in_category(category: str) -> Any:
    """Retrieve the books of the given category from the database."""
```

The full example is available at `tests.example.books_in_category()`.

CHAPTER

FIVE

TRANSACTIONS

Fastapi-icontract is not aware of the transactions.

This is especially relevant for concurrent systems where multiple clients modify a shared resource (*e.g.*, a database) at the same time. If you are not careful, you can end up with quite a few time-of-check-time-of-use (TOCTOU) errors.

A possible approach to use transaction both in the contracts *and* the endpoint is to introduce a decorator on top of fastapi-icontract decorators which will start a transaction and pass it down to the contracts and the function, and then close it when the underlying call stack is executed.

For example:

```
from fastapi_icontract import require

@app.get("/books_in_category", response_model=List[Book])
@start_transaction  # This passes ``txn`` to the underlying function and contracts
@require(
    lambda txn, category: has_category(txn, category),
    status_code=404,
    description="The category must exist."
)
async def books_in_category(category: str) -> Any:
    """Retrieve the books of the given category from the database."""
    ...
```

How you implement the `start_transaction` depends on your particular system, and we present its usage here just for illustration.

CHAPTER

SIX

DOCUMENTING CONTRACTS IN OPENAPI

Fastapi-icontract provides the function *fastapi_icontract.wrap_openapi_with_contracts()* so that you can include the contracts in the `openapi.json` endpoint of your FastAPI `app`.

The function is called once the `app` has been fully specified (see the *full example*):

```
fastapi_icontract.wrap_openapi_with_contracts(app=app)
```

The function will override the `app.openapi` method and cache the results for efficiency.

The modified schema is afterwards available at `app.openapi_url` (usually set to the default `"/openapi.json"`).

CHAPTER

SEVEN

VISUALIZING CONTRACTS IN SWAGGER UI

Assuming you include contracts in OpenAPI schema (see *Documenting Contracts in OpenAPI*), they will be available per each path as a `x-contracts` extension field. Unfortunately, Swagger UI does not pretty-prints extension fields, so that the contracts end up barely readable as a long single-line string.

While you might inspect the OpenAPI specification, it is much more convenient to read contracts in a nice structured layout. To that end, we developed swagger-ui-plugin-contracts, a JavaScript plugin that can be readily included in Swagger UI.

There are a couple of options how you can include the plugin in the Swagger UI of your `app`.

7.1 Use `set_up_route_for_docs_with_contracts_plugin`

The most straightforward way is to replace the documentation route (*i.e.*, the route corresponding to Swagger UI) is rendered.

First, you need to explicitly tell your `app` to skip creating the documentation route at setup by setting `docs_url=None`:

```
app = FastAPI(docs_url=None)
```

Fastapi-icontracts gives you the function *`set_up_route_for_docs_with_contracts_plugin()`* which creates the documentation route with the contracts plugin included.

You need to call it explicitly once the `app` has been set up (see the *full example*):

```
fastapi_icontract.set_up_route_for_docs_with_contracts_plugin(
    app=app, path="/docs")
```

From then on, Swagger UI with swagger-ui-plugin-contracts will be available at `"/docs"` path.

7.2 Specify Your Own Documentation Endpoint

`fastapi_icontract.set_up_route_for_docs_with_contracts_plugin()` does not really provide a way to customize the documentation endpoint.

For example, it does not allow you to specify a different URL from where swagger-ui-plugin-contracts should be fetched, or add additional plugins. If you need such more involved customizations, you need to specify the documentation endpoint yourself.

If you only need to change the URLs of the relevant files (*e.g.*, to Swagger UI, swagger-ui-plugin-contracts, *etc.*), you can use the function *`fastapi_icontract.swagger_ui.get_swagger_ui_html()`*:

If you need to include additional Swagger UI plugins or customize otherwise the HTML code of the documentation, you need to re-write your own version of *fastapi_icontract.swagger_ui.get_swagger_ui_html()*.

CHAPTER

EIGHT

API

8.1 require

class fastapi_icontract.**require**

Decorate a FastAPI endpoint with a pre-condition.

__init__(*condition: CallableT*, *status_code: int = 422*, *description: Optional[str] = None*, *enforced: bool = True*, *undocument: bool = False*) → None

Initialize.

Parameters

- **condition** – pre-condition predicate.

  The arguments of the pre-condition are expected to be a subset of the endpoint arguments.

  It can either be a sync function, a lambda, an async function. If the condition returns a coroutine, the coroutine will be awaited first, and then checked for truthiness.

- **status_code** – If the pre-condition is violated, the checker will raise a `fastapi.HTTPException`. This `status_code` will be indicated in the exception.

- **description** – textual description of the pre-condition.

  The `description` will be included in the exception if the pre-condition is violated.

- **enforced** – If set, the pre-condition is enforced.

  Otherwise, the pre-condition is only added to the OpenAPI schema, but is not verified. Usually, you enforce certain slow pre-conditions during testing and then disable them in production. An unenforced pre-condition is however still useful for the client as a formal documentation which is at least verified during testing.

- **undocument** – If set, the pre-condition is not documented in the OpenAPI schema.

__call__(*func: CallableT*) → CallableT

Add the pre-condition to the checker of a FastAPI endpoint.

Parameters **func** – endpoint function to be wrapped

Returns wrapped endpoint

8.2 snapshot

class fastapi_icontract.**snapshot**

Add a snapshot to the checker of an FastAPI endpoint.

This will decorate the endpoint with a snapshot of argument values captured *prior* to the invocation.

A snapshot is defined by a capture function (usually a lambda) that accepts one or more arguments of the function.

The captured values are supplied to post-conditions with the OLD argument of the condition.

__init__(*capture: CallableT*, *name: str*, *enabled: bool = True*, *undocument: bool = False*) → None

Initialize.

Parameters

- **capture** – function to capture the snapshot accepting a one or more arguments of the original function *prior* to the execution.

  The `capture` can either be a lambda, a sync function or an async function. If `capture` returns a coroutine, the coroutine will be first awaited before it is stored into the `OLD` structure.

- **name** – name of the snapshot as will be stored in the OLD structure.

- **enabled** – The snapshot is applied only if `enabled` is set. Otherwise, the snapshot is disabled and there is no run-time overhead.

  Usually the snapshots are enabled and disabled together with their related post-conditions.

- **undocument** – If set, the snapshot is not documented in the OpenAPI schema.

__call__(*func: CallableT*) → CallableT

Add the snapshot to the checker of a FastAPI endpoint `func`.

The function `func` is expected to be decorated with at least one postcondition before the snapshot.

Parameters **func** – function whose arguments we need to snapshot

Returns `func` as given in the input

8.3 ensure

class fastapi_icontract.**ensure**

Decorate a FastAPI endpoint with a post-condition.

__init__(*condition: CallableT*, *status_code: int = 500*, *description: Optional[str] = None*, *enforced: bool = True*, *undocument: bool = False*) → None

Initialize.

Parameters

- **condition** – post-condition predicate.

  The arguments of the post-condition are expected to be a subset of the endpoint arguments.

  It can either be a sync function, a lambda, an async function. If the condition returns a coroutine, the coroutine will be awaited first, and then checked for truthiness.

- **status_code** – If the post-condition is violated, the checker will raise a `fastapi.HTTPException`. This `status_code` will be indicated in the exception.

- **description** – textual description of the post-condition.

  The `description` will be included in the exception if the post-condition is violated.

- **enforced** – If set, the post-condition is enforced.

  Otherwise, the post-condition is only added to the OpenAPI schema, but is not verified. Usually, you enforce post-conditions during testing and then disable them all in production. An unenforced post-condition is however still useful for the client as a formal documentation which is at least verified during testing.

- **undocument** – If set, the post-condition is not documented in the OpenAPI schema.

__call__(*func: CallableT*) → CallableT

Add the postcondition to the checker of a FastAPI endpoint.

If the endpoint has not been already wrapped with a checker, this will wrap it with a checker first.

Parameters **func** – endpoint function to be wrapped

Returns wrapped endpoint

8.4 wrap_openapi_with_contracts

fastapi_icontract.**wrap_openapi_with_contracts**(*app: fastapi.applications.FastAPI*) → None

Wrap the `openapi` method of the `app` to include the contracts in the schema.

8.5 set_up_route_for_docs_with_contracts_plugin

fastapi_icontract.**set_up_route_for_docs_with_contracts_plugin**(*app: fastapi.applications.FastAPI*, *path: str = '/docs'*) → None

Set up the route for Swagger UI with included plugin swagger-ui-plugin-contracts.

The path must not be set before. You must explicitly tell FastAPI to exclude it during initialization with:

```
app = FastAPI(docs_url=None)
```

8.6 get_swagger_ui_html

fastapi_icontract.swagger_ui.**get_swagger_ui_html**(*, *openapi_url: str*, *title: str*, *swagger_js_url: str = 'https://cdn.jsdelivr.net/npm/swagger-ui-dist@3/swagger-ui-bundle.js'*, *swagger_css_url: str = 'https://cdn.jsdelivr.net/npm/swagger-ui-dist@3/swagger-ui.css'*, *swagger_favicon_url: str = 'https://fastapi.tiangolo.com/img/favicon.png'*, *swagger_ui_plugin_contracts_url: str = 'https://unpkg.com/swagger-ui-plugin-contracts'*, *oauth2_redirect_url: Optional[str] = None*, *init_oauth: Optional[Dict[str, Any]] = None*) → starlette.responses.HTMLResponse

Generate the HTML for Swagger UI endpoint.

This is a patched version of the original fastapi.applications.get_swagger_ui_html which includes a separate JavaScript code to display contracts in a pretty format.

CHAPTER

NINE

EXAMPLE

We present here a full example to demonstrate how you can use fastapi-icontract.

CHAPTER

TEN

CONTRIBUTING

10.1 Coordinate First

Before you create a pull request, please create a new issue first to coordinate.

It might be that we are already working on the same or similar feature, but we haven't made our work visible yet.

10.2 Create a Development Environment

We usually develop in a virtual environment. To create one, change to the root directory of the repository and invoke:

```
python -m venv venv
```

You need to activate it. On *nix (Linux, Mac, *etc.*):

```
source venv/bin/activate
```

and on Windows:

```
venv\Scripts\activate
```

10.3 Install Development Dependencies

Once you activated the virtual environment, you can install the development dependencies using `pip`:

```
pip3 install --editable .[dev]
```

The –editable option is necessary so that all the changes made to the repository are automatically reflected in the virtual environment (see also this StackOverflow question).

10.4 Pre-commit Checks

We provide a battery of pre-commit checks to make the code uniform and consistent across the code base.

We use black to format the code and use the default maximum line length of 88 characters.

The docstrings need to conform to PEP 257. We use Sphinx docstring format to mark special fields (such as function arguments, return values *etc.*). Please annotate your function with type annotations instead of writing the types in the docstring.

To run all pre-commit checks, run from the root directory:

```
python precommit.py
```

You can automatically re-format the code with:

```
python precommit.py --overwrite
```

Here is the full manual of the pre-commit script:

```
usage: precommit.py [-h] [--overwrite] [--select  [...]] [--skip  [...]]

Run pre-commit checks on the repository.

optional arguments:
  -h, --help        show this help message and exit
  --overwrite       Try to automatically fix the offending files (e.g., by re-
                    formatting).
  --select  [ ...]  If set, only the selected steps are executed. This is
                    practical if some of the steps failed and you want to fix
                    them in isolation. The steps are given as a space-
                    separated list of: black mypy pylint pydocstyle test
                    doctest check-init-and-setup-coincide check-help-in-doc
  --skip  [ ...]    If set, skips the specified steps. This is practical if
                    some of the steps passed and you want to fix the remainder
                    in isolation. The steps are given as a space-separated
                    list of: black mypy pylint pydocstyle test doctest check-
                    init-and-setup-coincide check-help-in-doc
```

The pre-commit script also runs as part of our continuous integration pipeline.

10.5 Write Commit Message

We follow Chris Beams' guidelines on commit messages:

1) Separate subject from body with a blank line
2) Limit the subject line to 50 characters
3) Capitalize the subject line
4) Do not end the subject line with a period
5) Use the imperative mood in the subject line
6) Wrap the body at 72 characters
7) Use the body to explain *what* and *why* vs. *how*

CHAPTER

ELEVEN

CHANGELOG

11.1 0.0.4

- Add support for Python 3.9 and 3.10 (#15)
- Include Python 3.6 and 3.7 in CI (#14)
- Fix code snippets in `README.rst` (#9)

11.2 0.0.3

- Add FastAPI classifier in `setup.py` (#10)
- Extend message on snapshot missing post-conditions (#7)

11.3 0.0.2

- Fix the teaser code in Readme (#4)

11.4 0.0.1

- This is the first, kick-off version.

CHAPTER

TWELVE

INDICES AND TABLES

- genindex
- modindex
- search

INDEX

Symbols

E

G

R

S

W