Table of content

Initialize

To initialize the project you have to instantiate the FastApiBuilder class. It's a singleton class so you can't instantiate it more than once it will always return the same instance.

```
from fast_api_builder.services.fast_api_builder import FastApiBuilder

if __name__ == "__main__":
    api_builder = FastApiBuilder()
```

Create API from functions

Add a function to the api with decorator

You can add a function to the api using the decorator `add_function_to_api_decorator`. The decorator takes 6 parameters:

- `path`: the path of the api
- `method`: the HTTP method of the api
- `function_name`: the name of the function
- `function_description`: the description of the function
- `using_cache`: if the function is using cache
- `max_retries`: the number of retries of a function

```
@api_builder.add_function_to_api_decorator("/square", "GET", max_retries=100)
def square(x1: int) -> int:
    return int(x1) ** 2
```

Add a function to the api with a classic function

The class also provide another way to add a function to the api. You can use the method `_add_function_to_api` takes 7 parameters:

- `function`: the function to add to the api
- `path`: the path of the api
- `method`: the HTTP method of the api
- `function_name`: the name of the function
- `function_description`: the description of the function
- `using_cache`: if the function is using cache
- `max_retries`: the number of retries of a function

```
api_builder._add_function_to_api(
        square,
        "/square",
        "GET",
        max_retries=100,
    )
```

Lambda function supported

You can also add a lambda function instead of a classique function to the api through the method `_add_function_to_api` that takes 7 parameters:

```
api_builder._add_function_to_api(
    lambda x1, x2: {"result": int(x1) + int(x2)},
    "/sum",
    "GET",
    function_name="sum",
    using_cache=True,
    max_retries=100,
)
```

It is strongly recomanded to give a name to the lambda function to avoid any problem with the cache.

Several HTTP methods

The class supports several HTTP methods:

- GET
- POST

The parameters are the same for both methods :

- `path`: the path of the api
- `method`: the HTTP method of the api
- `function_name`: the name of the function

- `function_description`: the description of the function
- `using_cache`: if the function is using cache
- `max_retries`: the number of retries of a function

It is strongly recommended to use the GET method so you do not have to follow a specific format for the parameters. You just have to make you in your function to cast the parameter to the wanted type.

```
@api_builder.add_function_to_api_decorator("/multiply", "GET", max_retries=100)
def multiply(x1: int, x2: int) -> int:
        return int(x1) * int(x2)
```

However if you want to use the POST method you have to follow a specific format for the parameters. The parameters have to be in a dataclass. You can name the function argument to `params` or any value. You must specify the type hinting with the dataclass you created, here `MultiplyPostInput`.

```
class MultiplyPostInput(BaseModel):
    x1: int
    x2: int

@api_builder.add_function_to_api_decorator("/multiply", "POST", max_retries=100)
def multiply_with_post(param: MultiplyPostInput):
    return {"result": param.x1 * param.x2}
```

Launch the API

To finalize the api you have to call the method `start_api` to start the server.

```
api_builder.start_api()
```

It is running at http://localhost:8000 by default.

Utilities

The class provides some utilities to help you check your API while running :

- http://localhost:8000/functions returns the available functions accessible through HTTP :

```
{
  "health_check": {
    "route_path": "/health",
    "function_name": "health_check",
    "http_method": "GET",
    "description": "Perform a health check on the API",
```

```
    "n_calls": 0,
    "max_calls": 1000000
  },
  "square": {
    "route_path": "/square",
    "function_name": "square",
    "http_method": "GET",
    "description": "-",
    "n_calls": 0,
    "max_calls": 100
  },
  "_get_routes": {
    "route_path": "/functions",
    "function_name": "_get_routes",
    "http_method": "GET",
    "description": "Get the list of functions",
    "n_calls": 1,
    "max_calls": 1000000
  }
}
```


- http://localhost:8000/health returns the status of the API :


```
"up & running"
```


- http://localhost:8000/docs in your browser to see the documentation of the API.