4) FastAPI

Documentation: https://fastapi.tiangolo.com

Source Code: https://github.com/tiangolo/fastapi

Design goals

- Fast
- Good IDE support (autocompletion)
- Minimal boilerplate
- Standards based
- Easily testable

FastAPI

Starlette

Uvicorn

Pydantic

Ison Schema

OpenAPI

OAuth2

Modern Python 3.6+ (asyncio + type hints)

Open Standards

Features

- Fully async
- Good documentation
- Full featured: WebSockets, GraphQL, CORS, GZip, Static Files, Templating, Streaming responses, Background Tasks, Startup and shutdown events, ...
- Automatic data validation
- Test client built on requests API.
- 100% test coverage.
- 100% type annotated codebase.

Leverages standard Python type hints

- Type checking
- IDE auto-completion
- Data validation
- Data serialization
- Automatic API documentation
- Automatic OpenAPI schema generation

Basic example (no data validation)

```
from fastapi import FastAPI
api = FastAPI()
@api.get("/user/{user_id}")
def get_user(user_id):
    user = db.find(user_id)
    return user
@api.put("/user")
def create_user(user):
    user_id = db.create(user)
    return user_id
```

Automatic data validation (Pydantic)

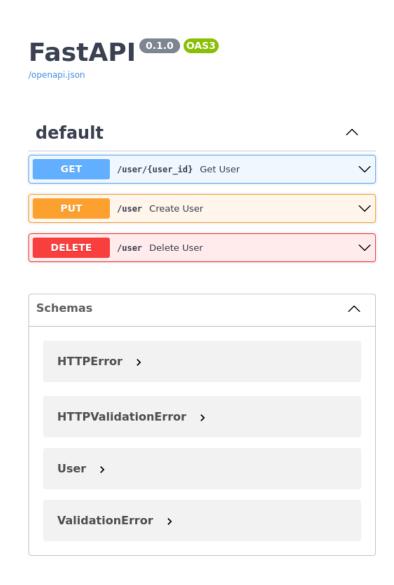
```
from fastapi import FastAPI
from pydantic import BaseModel
class User(BaseModel):
    last_name: str
    first name: str
    age: int
api = FastAPI()
@api.get("/user/{user_id}", response_model=Optional[User])
def get_user(user_id: int):
    user = db.find(user_id)
    return user
@api.put("/user", response_model=int)
def create_user(user: User):
    user_id = db.create(user)
    return user id
```

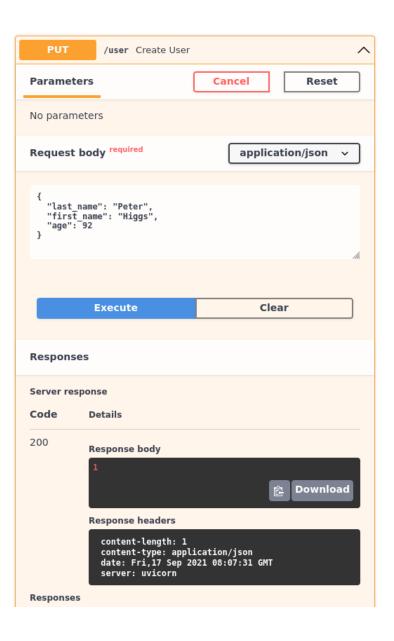
Error handling

- Automatic handling of data validation errors
- HTTPException for custom error handling

```
@api.get(
    "/user/{user_id}",
    response_model=User,
    responses={status.HTTP_404_NOT_FOUND: {"model": HTTPError}},
def get_user(user_id: int):
    user = db.find(user_id)
    if user is None:
        raise HTTPException(
            status_code=status.HTTP_404_NOT_FOUND,
            detail=f"User '{user_id}' does not exist.",
    return user
```

Automatic interactive documentation





Testing

```
from http import HTTPStatus
from fastapi.testclient import TestClient
from example.main import api
demo_user = {"first_name": "Peter", "last_name": "Higgs", "age": 92}
def test_create_user():
    client = TestClient(api)
    response = client.put("/user", json=demo_user)
    assert response.status_code == HTTPStatus.OK
    assert response.text == str(0)
    response = client.get("/user/0")
    assert response.status_code == HTTPStatus.OK
    assert response.json() == demo_user
    response = client.get("/user/1")
    assert response.status_code == HTTPStatus.NOT_FOUND
    assert response.json() == {"detail": "User '1' does not exist."}
```

Fuzzing with Hypothesis

```
from http import HTTPStatus
from fastapi.testclient import TestClient
from hypothesis import given, strategies as st
from example.main import api, User
def test_inserting_random_users():
    client = TestClient(api)
    @given(st.builds(User, age=st.integers(1, 99)))
    def insert_user(random_user: User):
        response = client.put("/user", json=random_user.dict())
        assert response.status_code == HTTPStatus.OK
        user_id = int(response.text)
        response = client.get(f"/user/{user_id}")
        assert response.status_code == HTTPStatus.OK
        assert response.json() == random_user.dict()
    insert_user()
```

Schema compliance with Schemathesis

> schemathesis run --checks all --app=example.main:api /openapi.json

```
platform Linux -- Python 3.8.10, schemathesis-3.10.0, hypothesis-6.21.4, hypothesis jsonschema-0.20.1, jsonschema-3.2.0
rootdir: /fastapi slides/src/example
Schema location: /openapi.json
Base URL: /
Specification version: Open API 3.0.2
Workers: 1
Collected API operations: 3
GET /user/{user id} .
                                                                                 33%]
PUT /user .
                                                                                66%]
DELETE /user .
                                                                                [100%]
                  Performed checks:
                                  300 / 300 passed
  not a server error
                                                    PASSED
  status_code_conformance
                                  300 / 300 passed
                                                   PASSED
  content_type_conformance
                                  300 / 300 passed
                                                   PASSED
  response headers conformance
                                  300 / 300 passed
                                                    PASSED
  response schema conformance
                                  300 / 300 passed
                                                   PASSED
```

Schema compliance as a pytest test

```
import schemathesis
from schemathesis.checks import ALL_CHECKS
from example.main import api
schema = schemathesis.from_dict(
    api.openapi(),
    data generation methods=[
        schemathesis.DataGenerationMethod.positive, # generates valid data
        schemathesis.DataGenerationMethod.negative, # generates invalid data
@schema.parametrize()
def test_schema_compliance(case):
    response = case.call_asgi(api)
    case.validate_response(response, checks=ALL_CHECKS)
```

Comparison (Code ⇔ OpenAPI Spec)

```
from http import HTTPStatus
from typing import Optional
from fastapi import FastAPI, HTTPException
from pydantic import BaseModel
import uvicorn
from example.simple_db import Database
class User(BaseModel):
    last name: str
    first name: str
    age: int
db: Database[User] = Database()
api = FastAPI()
@api.get("/user/{user_id}", response_model=User)
def get_user(user_id: int):
    user = db.find(user_id)
    if user is None:
       raise HTTPException(
            status code=HTTPStatus.NOT FOUND,
            detail=f"User '{user_id}' does not exist.",
    return user
@api.put("/user", response_model=int)
def create_user(user: User):
    user_id = db.create(user)
    return user id
@api.delete("/user", response_model=Optional[User])
def delete_user(user_id: int):
    deleted_user = db.delete(user_id)
    return deleted user
if name == " main ":
    uvicorn.run("example2:api", host="0.0.0.0", port=9001, reload=True)
```

