



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

Json 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

**FastAPI** 0.1.0 OAS3  
[/openapi.json](#)

default ^

**GET** /user/{user\_id} Get User ^

**PUT** /user Create User ^

**DELETE** /user Delete User ^

Schemas ^

HTTPError >

HTTPValidationError >

User >

ValidationError >

**PUT** /user Create User ^

**Parameters** Cancel Reset

No parameters

**Request body** required application/json ^

```
{
  "last_name": "Peter",
  "first_name": "Higgs",
  "age": 92
}
```

Execute Clear

**Responses**

**Server response**

Code	Details
200	<div><b>Response body</b></div> <div>1 <span>Download</span></div> <div><b>Response headers</b></div> <div>content-length: 1 content-type: application/json date: Fri, 17 Sep 2021 08:07:31 GMT server: uvicorn</div>

**Responses**

# 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
```

```
===== Schemathesis test session starts =====
```

```
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%]
```

```
===== SUMMARY =====
```

## Performed checks:

not_a_server_error	300 / 300 passed	<b>PASSED</b>
status_code_conformance	300 / 300 passed	<b>PASSED</b>
content_type_conformance	300 / 300 passed	<b>PASSED</b>
response_headers_conformance	300 / 300 passed	<b>PASSED</b>
response_schema_conformance	300 / 300 passed	<b>PASSED</b>

```
===== 3 passed in 7.01s =====
```

# 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 fastapi import FastAPI, HTTPException, status
from pydantic import BaseModel
import uvicorn

from example.simple_db import Database

class User(BaseModel):
    last_name: str
    first_name: str
    age: int

class HTTPError(BaseModel):
    detail: str

db: Database[User] = Database()

api = FastAPI()

@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

@api.put("/user", response_model=int)
def create_user(user: User):
    user_id = db.create(user)
    return user_id

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

if __name__ == "__main__":
    uvicorn.run("main:api", host="0.0.0.0", port=9001, reload=True)
```

```
openapi: 3.0.0
info:
  title: Simple API
  version: 1.0.0
paths:
  /user/{user_id}:
    get:
      summary: Get user by ID
      description: Retrieve a user by their ID.
      parameters:
        - name: user_id
          in: path
          required: true
      responses:
        200:
          description: Successful response
          content:
            application/json:
              schema:
                $ref: '#/components/schemas/User'
        404:
          description: User not found
          content:
            application/json:
              schema:
                $ref: '#/components/schemas/HTTPError'
  /user:
    put:
      summary: Create user
      description: Create a new user.
      requestBody:
        required: true
        content:
          application/json:
            schema:
              $ref: '#/components/schemas/User'
      responses:
        200:
          description: Successful response
          content:
            application/json:
              schema:
                type: integer
    delete:
      summary: Delete user
      description: Delete a user by their ID.
      parameters:
        - name: user_id
          in: path
          required: true
      responses:
        200:
          description: Successful response
          content:
            application/json:
              schema:
                $ref: '#/components/schemas/User'
        404:
          description: User not found
          content:
            application/json:
              schema:
                $ref: '#/components/schemas/HTTPError'
```