FastAPI is a modern, fast (high-performance), web framework for building APIs with Python based on standard Python type hints.

The key features are:

* **Fast**: Very high performance, on par with **NodeJS** and **Go** (thanks to Starlette and Pydantic). [One of the fastest Python frameworks available](https://fastapi.tiangolo.com/#performance).
* **Fast to code**: Increase the speed to develop features by about 200% to 300%. \*
* **Fewer bugs**: Reduce about 40% of human (developer) induced errors. \*
* **Intuitive**: Great editor support. Completion everywhere. Less time debugging.
* **Easy**: Designed to be easy to use and learn. Less time reading docs.
* **Short**: Minimize code duplication. Multiple features from each parameter declaration. Fewer bugs.
* **Robust**: Get production-ready code. With automatic interactive documentation.
* **Standards-based**: Based on (and fully compatible with) the open standards for APIs: [OpenAPI](https://github.com/OAI/OpenAPI-Specification" \t "_blank) (previously known as Swagger) and [JSON Schema](https://json-schema.org/).

FastAPI stands on the shoulders of giants:

* [Starlette](https://www.starlette.io/) for the web parts.
* [Pydantic](https://docs.pydantic.dev/) for the data parts.

Dependencies[¶](https://fastapi.tiangolo.com/#dependencies)

FastAPI depends on Pydantic and Starlette.

standard Dependencies[¶](https://fastapi.tiangolo.com/#standard-dependencies)

When you install FastAPI with pip install "fastapi[standard]" it comes the standard group of optional dependencies:

Used by Pydantic:

* [email-validator](https://github.com/JoshData/python-email-validator) - for email validation.

Used by Starlette:

* [httpx](https://www.python-httpx.org/) - Required if you want to use the TestClient.
* [jinja2](https://jinja.palletsprojects.com/) - Required if you want to use the default template configuration.
* [python-multipart](https://github.com/Kludex/python-multipart) - Required if you want to support form "parsing", with request.form().

Used by FastAPI / Starlette:

* [uvicorn](https://www.uvicorn.org/) - for the server that loads and serves your application. This includes uvicorn[standard], which includes some dependencies (e.g. uvloop) needed for high performance serving.
* fastapi-cli - to provide the fastapi command.

Additional Optional Dependencies[¶](https://fastapi.tiangolo.com/#additional-optional-dependencies)

There are some additional dependencies you might want to install.

Additional optional Pydantic dependencies:

* [pydantic-settings](https://docs.pydantic.dev/latest/usage/pydantic_settings/) - for settings management.
* [pydantic-extra-types](https://docs.pydantic.dev/latest/usage/types/extra_types/extra_types/) - for extra types to be used with Pydantic.

Additional optional FastAPI dependencies:

* [orjson](https://github.com/ijl/orjson) - Required if you want to use ORJSONResponse.
* [ujson](https://github.com/esnme/ultrajson) - Required if you want to use UJSONResponse.

Pydantic models[¶](https://fastapi.tiangolo.com/python-types/#pydantic-models)

[Pydantic](https://docs.pydantic.dev/) is a Python library to perform data validation.

You declare the "shape" of the data as classes with attributes.

And each attribute has a type.

Then you create an instance of that class with some values and it will validate the values, convert them to the appropriate type (if that's the case) and give you an object with all the data.

And you get all the editor support with that resulting object.

from datetime import datetime

from pydantic import BaseModel

class User(BaseModel):

id: int

name: str = "John Doe"

signup\_ts: datetime | None = None

friends: list[int] = []

external\_data = {

"id": "123",

"signup\_ts": "2017-06-01 12:22",

"friends": [1, "2", b"3"],

}

user = User(\*\*external\_data)

print(user)

# > User id=123 name='John Doe' signup\_ts=datetime.datetime(2017, 6, 1, 12, 22) friends=[1, 2, 3]

print(user.id)

# > 123