FastAPI: The Beginner's Crash Course

High-performance API development with Python.

Based on the tutorial by freeCodeCamp.org

Modern, Fast, and High-Performance

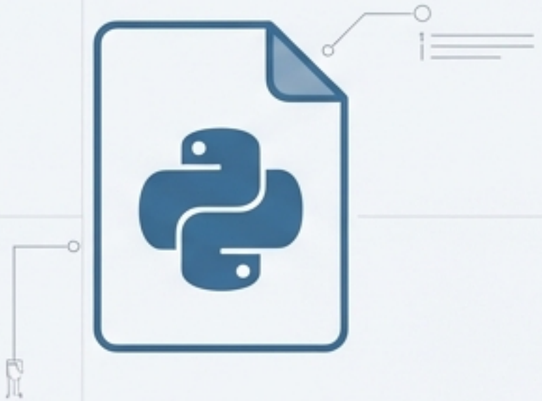
A framework designed for building web applications and APIs quickly.

Speed

One of the fastest Python frameworks available, offering performance on par with NodeJS and Go.

Accessibility

Features automatically generated, interactive documentation using Swagger UI.

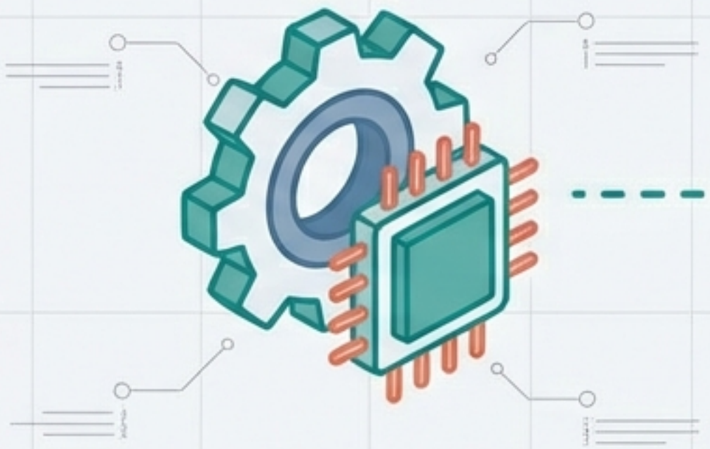


Requirements

Built on modern standards. Python 3.6+ is the only prerequisite.

Setting the Foundation: Installation

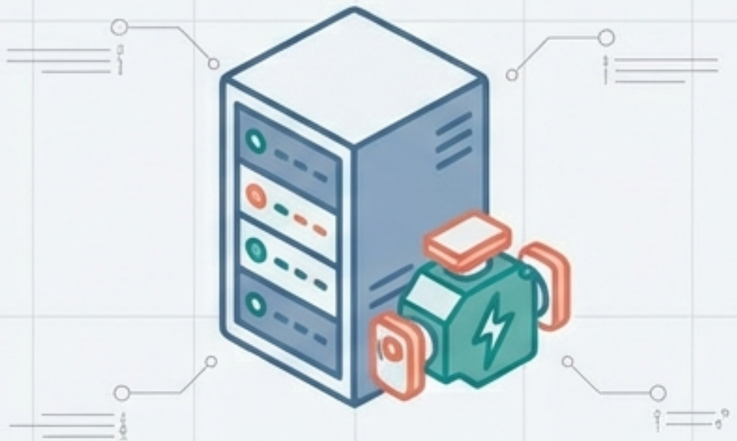
The Framework



```
pip install fastapi
```

Installs the core framework logic.

The Server

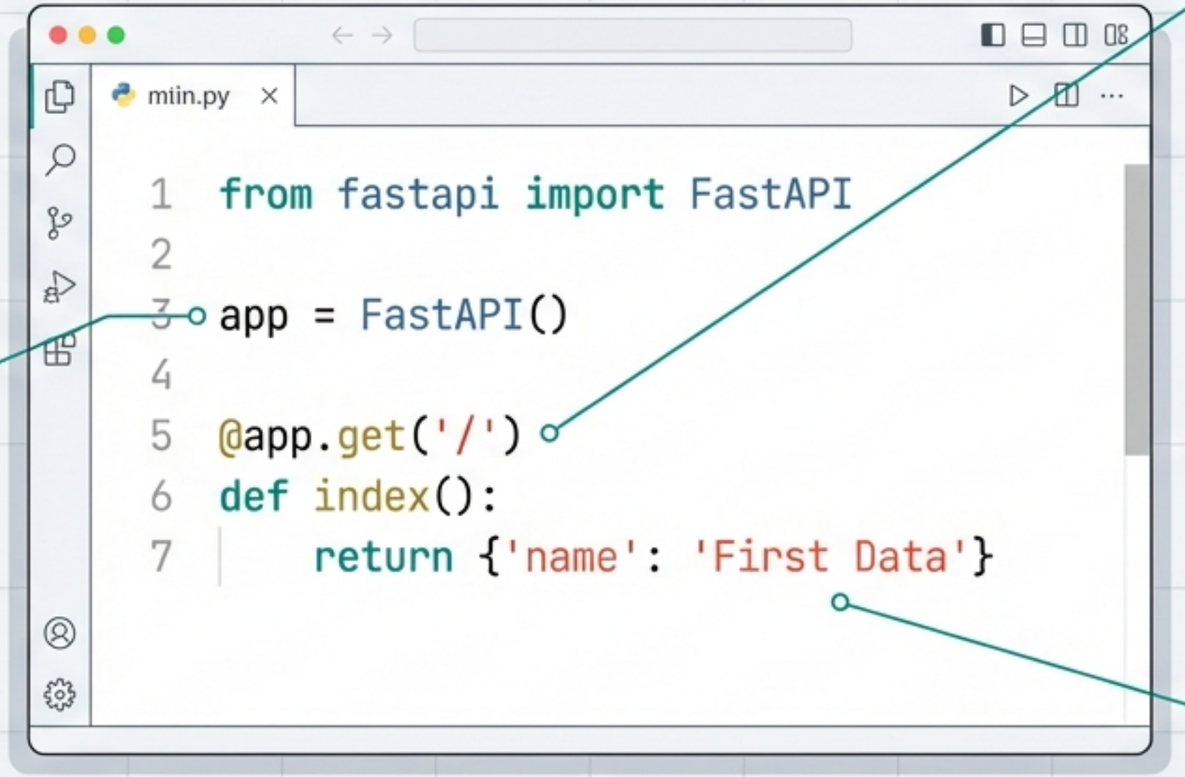


```
pip install uvicorn
```

Installs the ASGI server implementation.

Key Distinction: Unlike Django, which has a built-in server, FastAPI is lightweight and requires an external ASGI server like Uvicorn to run.

The First Blueprint: Hello World



Creating the instance. This 'app' variable is the main entry point.

The Endpoint (Route). Defines the path to the homepage.

FastAPI automatically converts this dictionary to JSON.

Igniting the Server

Anatomy of a Command

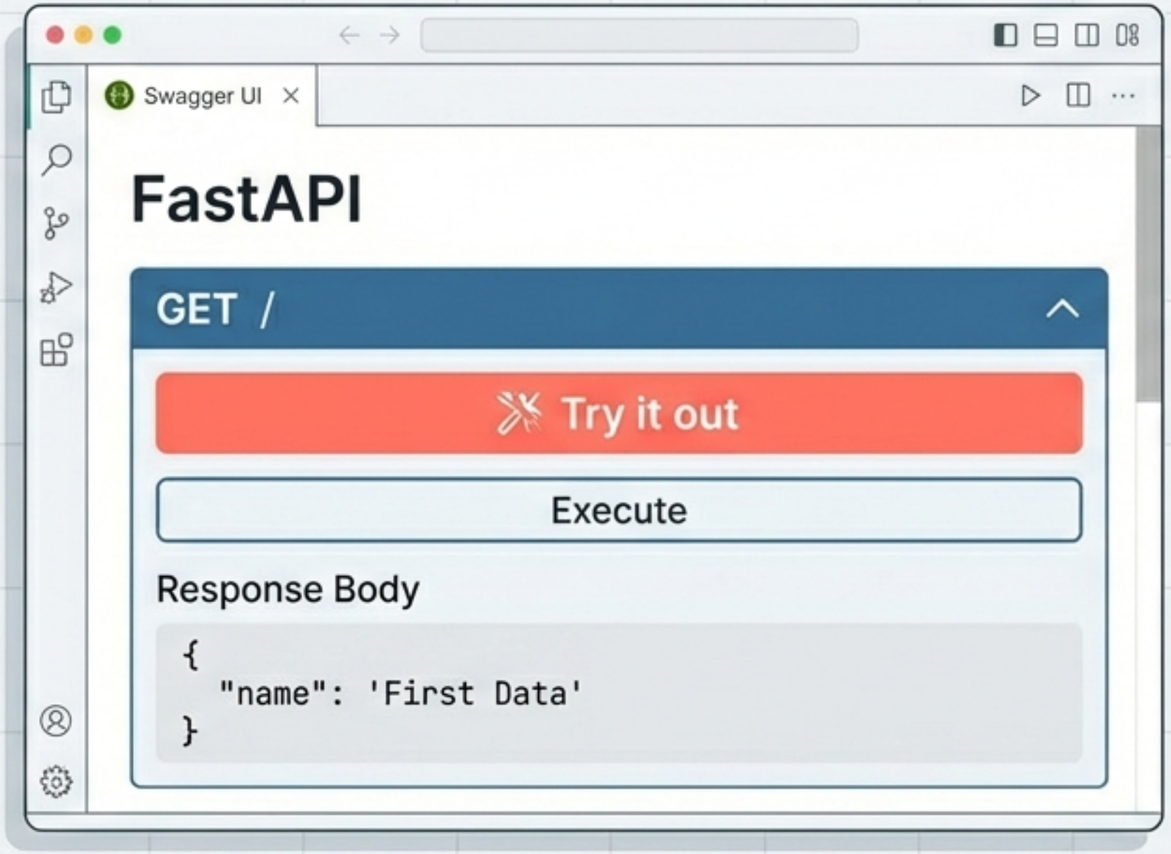
[uvicorn] [my_api] [app] [--reload]

- [uvicorn]: Server Executable
- [my_api]: Filename (my_api.py)
- [app]: FastAPI Instance Name
- [--reload]: Auto-restart on code change

```
INFO: Uvicorn running on http://127.0.0.1:8000 (Press CTRL+C to quit)
```

The ‘Magic’ Feature: Automatic Documentation

Access automatic interactive docs at /docs



Developer Experience (DX) Win:

No need for external tools like Postman initially. Test your API endpoints, parameters, and responses directly in the browser via the auto-generated Swagger UI.

Concepts: Endpoints & Methods

An Endpoint is one end of a communication channel—essentially the URL path.

GET

Retrieve information.

POST

Create something new.

PUT

Update existing data.

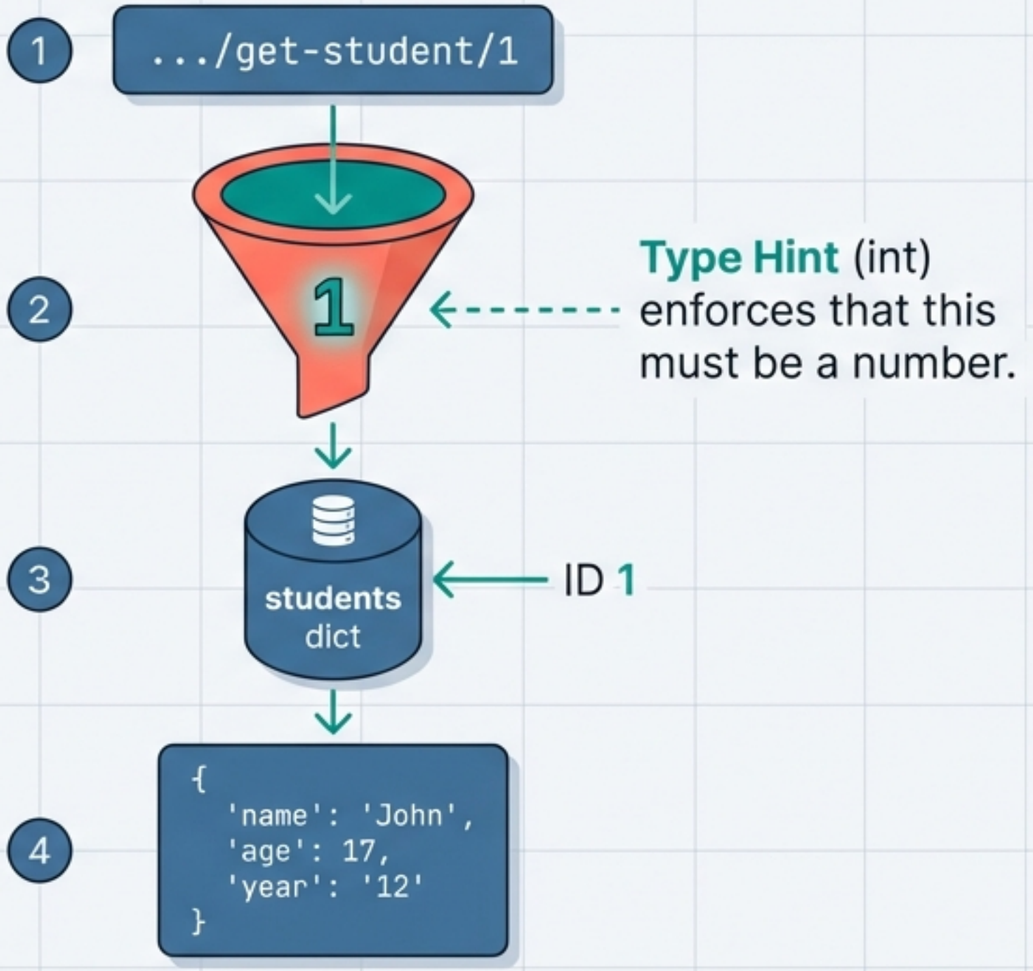
DELETE

Remove data.

Reading Data: Path Parameters

Capturing dynamic values from the URL.

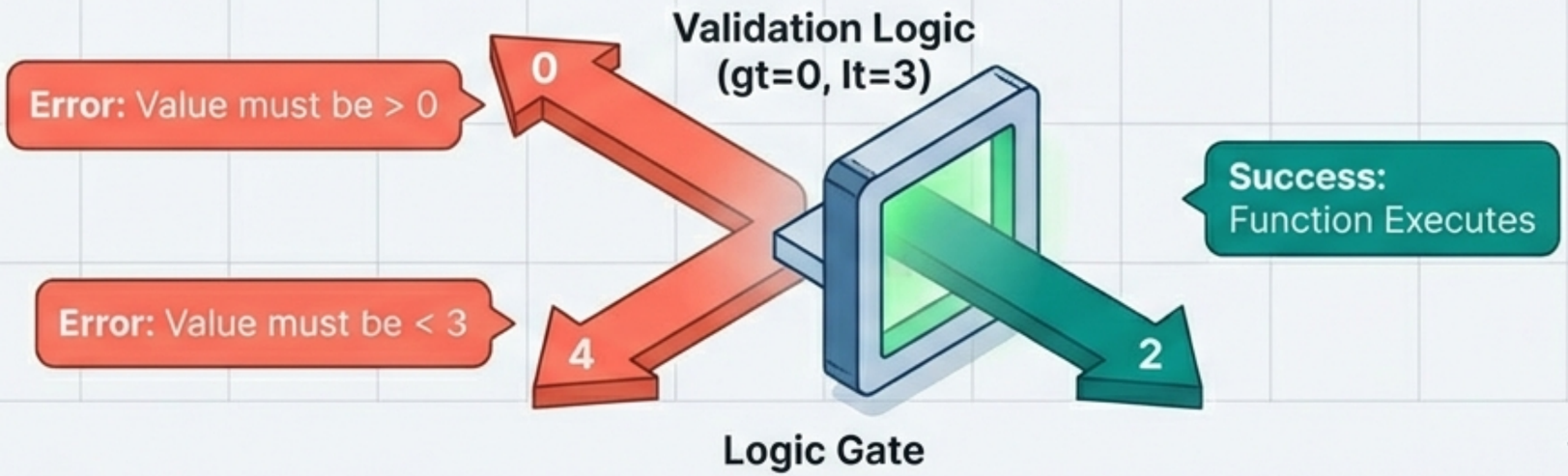
```python
@app.get('/get-student/{student_id}')
def get_student(student_id: int):
    return students[student_id]
```



Path Parameter Validation

```python
from fastapi import Path

@app.get('/student/{student_id}')
def get_student(
    student_id: int = Path(None, description='The ID', gt=0, lt=3)
):
    return students[student_id]
```



Reading Data: Query Parameters

Filtering and searching using key-value pairs.

`/get-student/1`

Path Parameter
(Required Route)

`/get-by-name?name=John`

Query Parameter
(Optional Filter)

```python
@app.get('/get-by-name')
def get_student(name: Optional[str] = None):
    for student_id in students:
        if students[student_id]['name'] == name:
            return students[student_id]
    return {'Data': 'Not found'}
```

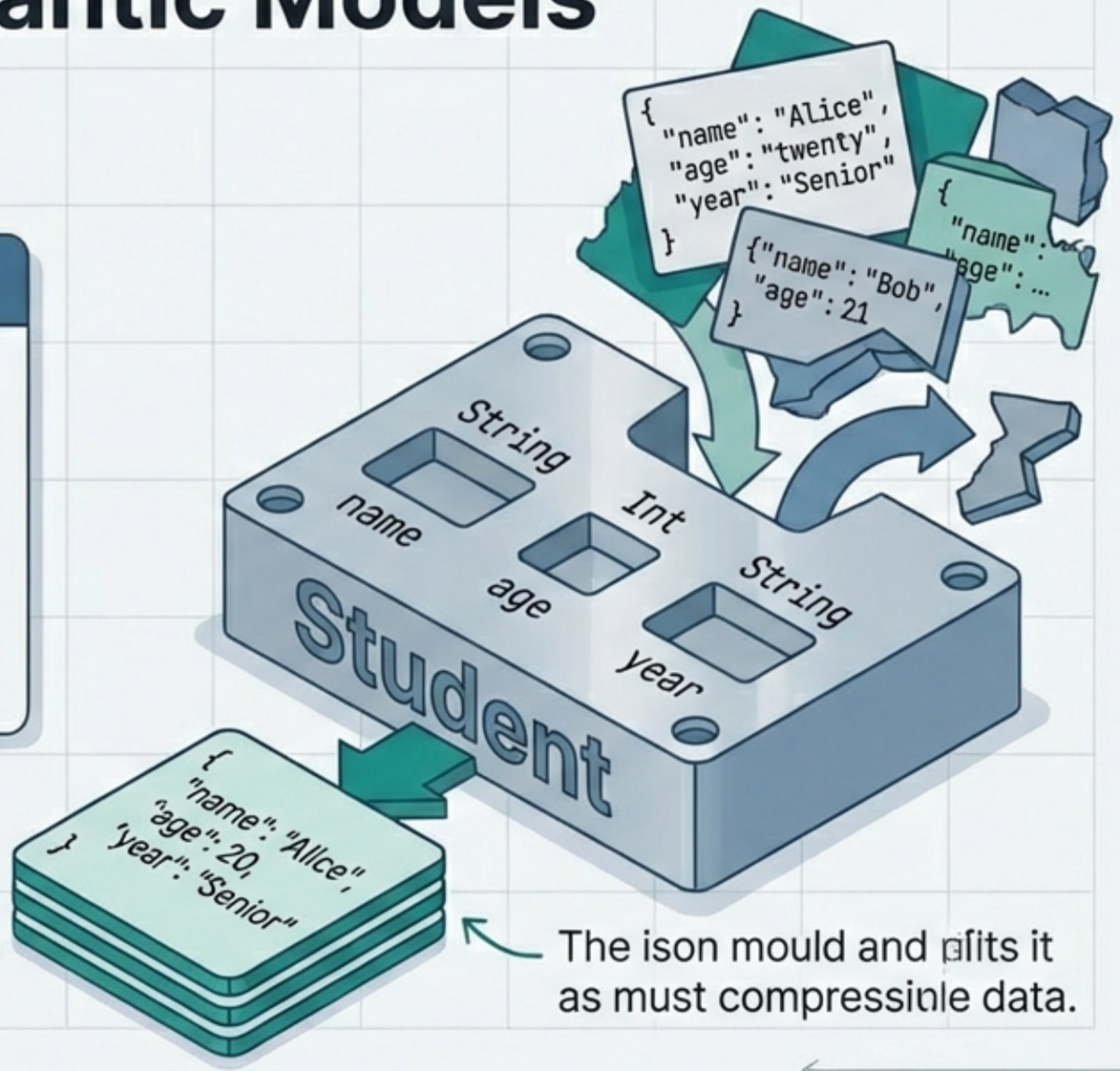
Handles missing parameters gracefully without crashing.

Structuring Data: Pydantic Models

Defining a contract for valid data.

```python
from pydantic import BaseModel

class Student(BaseModel):
    name: str
    age: int
    year: str
```



The Pydantic model acts as a strict blueprint. Data must match this shape to be accepted.

The ison mould and ofits it as must compressinle data.

Create: The POST Method

Creating new resources and handling data intake.

1 Input

Endpoint receives ID and JSON Body.

2 Validation

Pydantic ensures body matches Student model.

```python
@app.post('/create-student/{student_id}')
def create_student(student_id: int, student: Student):
    if student_id in students:
        return {'Error': 'Student exists'}

    students[student_id] = student
    return students[student_id]
```

3 Logic

Check existence to prevent duplicates.

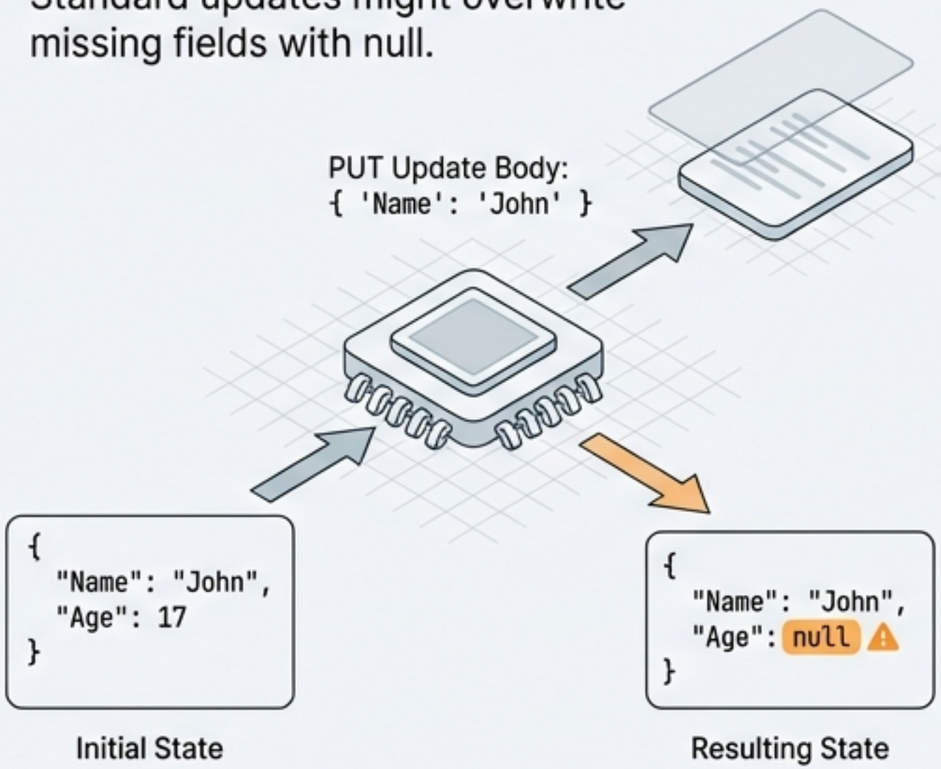
4 Success

Save to dict and return object.

Update: The PUT Method & Data Integrity

The Problem

Standard updates might overwrite missing fields with null.



The Solution

Use a separate model with Optional fields.
Only update fields that are not None.

```python
from typing import Optional
from pydantic import BaseModel

class UpdateStudent(BaseModel):
    name: Optional[str] = None
    age: Optional[int] = None

# In the function:
def update_student(id: int, student: UpdateStudent):
    if student.name != None:
        students[id].name = student.name
    if student.age != None:
        students[id].age = student.age
    return students[id]
```

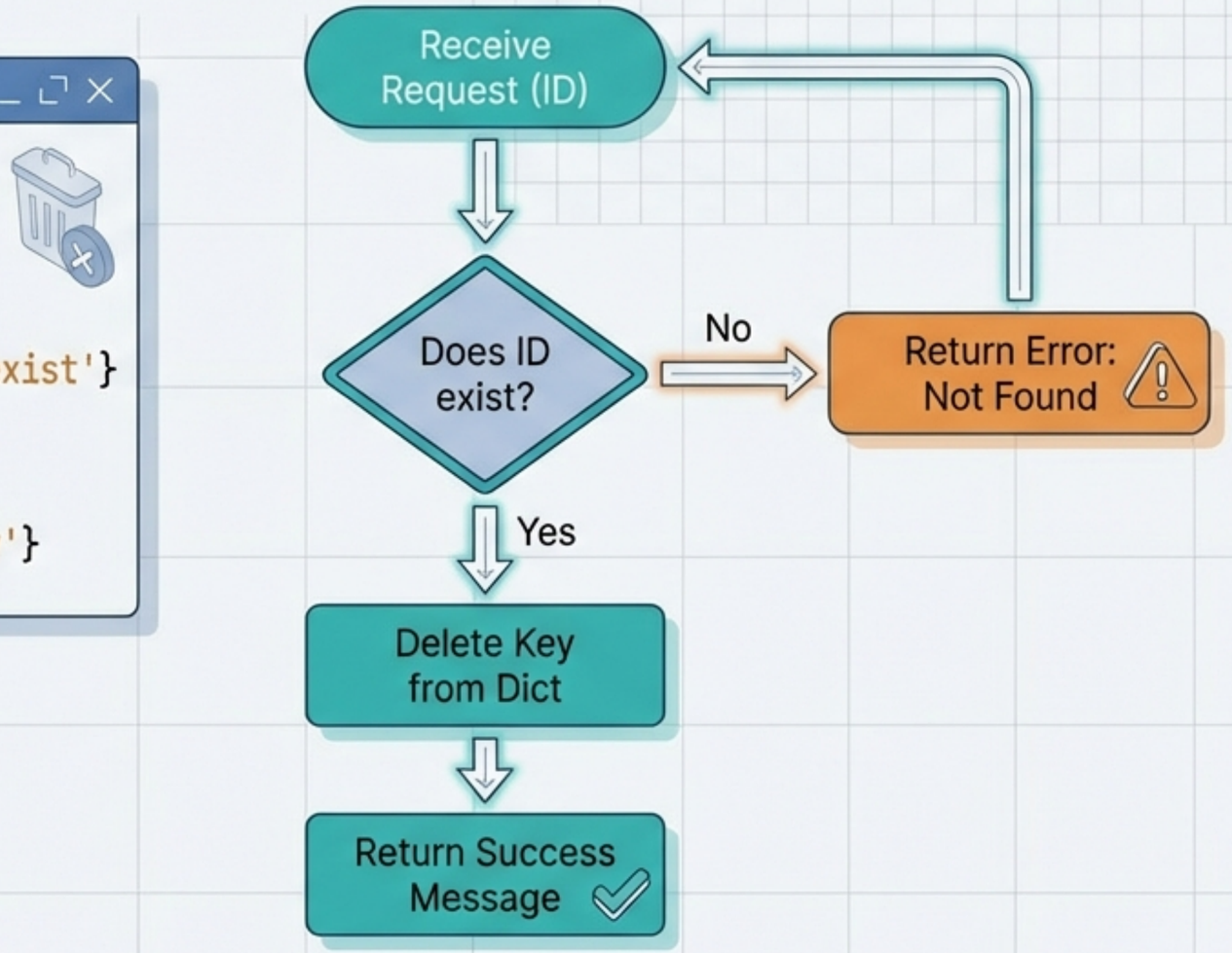
Use a separate model with Optional fields.

Only update fields that are not None.

Delete: The DELETE Method

```python
@app.delete('/delete-student/{student_id}')
def delete_student(student_id: int):
    if student_id not in students:
        return {'Error': 'Student does not exist'}

    del students[student_id]
    return {'Message': 'Deleted successfully'}
```



Summary & Best Practices

FastAPI is the framework; Uvicorn is the ASGI server required to run it.

Use Pydantic Models (BaseModel) to strictly validate request bodies.

GET (Read), POST (Create), PUT (Update), DELETE (Remove).

Visit /docs for instant, interactive Swagger UI testing.

Always handle edge cases—check if an ID exists before creating or deleting.

Download the full cheat sheet from freeCodeCamp to continue your build.