Serving Structured Data Using Pydantic Models

Reindert-Jan Ekker

@rjekker www.codesensei.nl

Overview

Operations that add/change data

Structured data

- Pydantic model classes
- Save/load as JSON file

Input and Output schemas

- Request and Response body

Calling our API using Postman

- Using openapi.json

HTTP Methods

GET `/api/cars/5`

Retrieve data – dont change anything.

POST `/api/cars`

Add a new item. Data in body.

PUT `/api/cars/5`

Replace resource: update a car. Data in body.

DELETE `/api/cars/5`

Remove a car

```
from pydantic import BaseModel

class Car(BaseModel):
    id: int
    fuel: str|None = "electric"
    trips: list[Trip] = []

# BaseModel functionality
car = Car(id=5, fuel="gas")  # __init__

# Convert to json, dict, str
car.json(), car.dict(), str(car)
```

Using pydantic Models

Inherit from pydantic.BaseModel

List fields with types as class attributes

Can include (collections of) other Model objects

Get lots of standard functionality (see https://pydantic-docs.helpmanual.io/)

Separate Input and Output Models

carsharing.py

```python
from schemas import CarInput, CarOutput


@app.post("/api/cars/")
def add_car(car: CarInput) -> CarOutput:
    # Create new_car based on input
    new_car = CarOutput(...)
    # Save new car...
    return new_car
```

schemas.py

```python
from pydantic import BaseModel


class CarInput(BaseModel):
    size: str
    fuel: str|None = "electric"


class CarOutput(CarInput):
    id: int
```

```python
@app.put("/api/cars/{id}", response_model=CarOutput)
def change_car(id: int, new_data: CarInput) -> CarOutput:
    car = find_car_by_id(id)
    if car:
        # update car with data from new_data
        # save, and return
    else:
        raise HTTPException(status_code=404,
                            detail=f"No car with id={id}.")
```

Pydantic Models in Request and Response

`id` **is a path parameter (from URL)**
`new_data` **is a pydantic model so it's read from the request body**
Return type for function is not used by fastapi
To set schema for response, use response_model in the decorator

Setting the Default Status Code

```python
@app.delete("/api/cars/{id}", status_code=204)
def remove_car(id: int) -> None:
    car = find_car_by_id()
    if car:
        # Remove car and save
        # No return needed
    else:
        raise HTTPException(status_code=404,
                            detail=f"No car with id={id}.")
```

Overview

POST, PUT, DELETE Operations

Structured data with Pydantic

- Input and output schemas
- Request/response body
- Nested models

Using openapi.json with Postman

Up Next: Using a Database