PUT & DELETE

Update(PUT)

- Patient ID will be provided
- + We'll get a request body
    - What they'd like to be changed

Steps:

- Build a new pydantic model
- We cannot use the old one because all the fields are required and we can get only 1 or 2 things that should be edited
- We'll make all fields **Optional**
- We'll get : 1. Patient ID & 2. Request body JSON

Code:

- Make a class with all fields **optional**

```
class PatientUpdate(BaseModel):
    name: Annotated[Optional[str], Field(default=None)]
    city: Annotated[Optional[str], Field(default=None)]
    age: Annotated[Optional[int], Field(default=None, gt=0)]
    gender: Annotated[Optional[Literal['male', 'female']], Field(default=None)]
    height: Annotated[Optional[float], Field(default=None, gt=0)]
    weight: Annotated[Optional[float], Field(default=None, gt=0)]
```

- No ID because it will be getting it as a path parameter

Endpoint:

```
@app.put('/edit/{patient_id}')
def update_patient(patient_id: str, patient_update: PatientUpdate):
```

`@app.put('/edit/{ patient_id }')` :

- We will enter patient ID of the patient we want to edit

`def update_patient(patient_id: str, patient_update : PatientUpdate ):`

- `PatientUpdate` is a Pydantic object of `PatientUpdate` **class** that we have created above.

```
    data = load_data()

    if patient_id not in data:
        raise HTTPException(status_code=404, detail="Patient not found")

    existing_patient_info = data[patient_id]
```

- Load the data
- If the patient not found, raise an HTTP exception error
- extract existing info of patient
- `existing_patient_info` **will have all the data of patient in a dictionary like** 👇

```
{"name": "Ananya Verma", "city": "Guwahati", "age": 28, "gender": "female", "height": 1.65, "weight": 90.0, "bmi": 33.06, "verdict": "Obese"}
```

- We'll receive the info in pydantic object
- We have to convert that to a python dictionary → `object.model_dump()`

```
# convert that to a python dictionary
updated_patient_info = patient_update.model_dump(exclude_unset=True)
```

`exclude_unset=True` :

- If we set the value to False(Default), it will include all the fields. But we only want the fields which needed to be updated.
  - eg. If the user sends `city` & `weight`, it will only these contain 2 items' new values.

How FastAPI fills `patient_update` ?

Suppose the client makes this HTTP request:

```
PUT /edit/P001
Content-Type: application/json

{
  "name": "John",
  "age": 40
}
```

What happens:

1. FastAPI sees that `patient_update` must be a `PatientUpdate` object (from your function signature).
2. It takes the incoming JSON body (`{"name":"John","age":40}`) and **creates a Pydantic object**:

   ```
   patient_update = PatientUpdate(name="John", age=40)
   ```

3. That `patient_update` object now exists **before your function body starts running**.

So by the time your code reaches:

```
updated_patient_info = patient_update.model_dump(exclude_unset=True)
```

➡️ `patient_update` already contains data from the request.

```
# Update the new values
for key, value in updated_patient_info.items():
    existing_patient_info[key] = value
```

- This will run a loop in the new dictionary, which contains the new info.

```
{
'city':'mumbai',
'weight': 90
}
```

> 💡 **BUT, it will update the value of existing info dictionary.**

- After running this, we'll have a new dictionary with updated values

> ❗ **There's a problem: If we update weight, we want BMI and verdict auto updated.**

> 💡 **To update BMI & Verdict, we have to make a new pydantic object for `Patient` class with the dictionary `existing_patient_info`**

- In short, **Dict** → **Pydantic Object**
- In this process, all the fields will be recalculated

- We'll again do **Pydantic Object → Dict**
- Save the data to the patient with `data[patient_id] = existing_patient_info`

> 💡 `existing_patient_info` **DOES NOT HAVE** `patient_id` **but** `Patient` **does. So, we have to add the** `patient_id` **to** `existing_patient_info` 👇

```
# Add patient_id in existing_patient_info
existing_patient_info['id'] = patient_id
```

Create pydantic object for Patient class:

```
# Create pydantic object for Patient class
patient_pydantic_obj = Patient(**existing_patient_info)
```

Pydantic obj → Dict:

```
# Pydantic obj → Dict
existing_patient_info = patient_pydantic_obj.model_dump(exclude='id')
```

`exclude='id'` → We do not want id

Add this dictionary to data:

```
# Add this dictionary to data
data[patient_id] = existing_patient_info
```

Save:

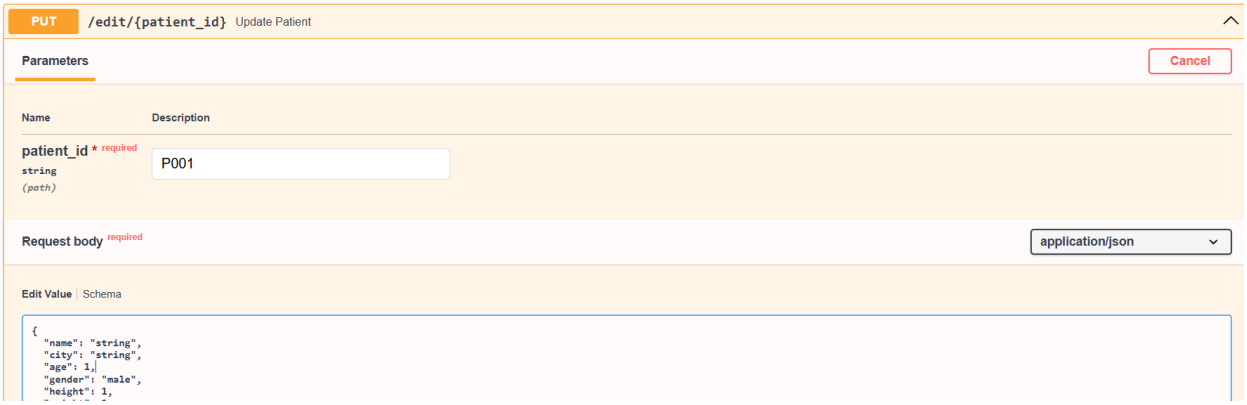
```
    # Save
    save_data(data)
```

Retuen JSON response:

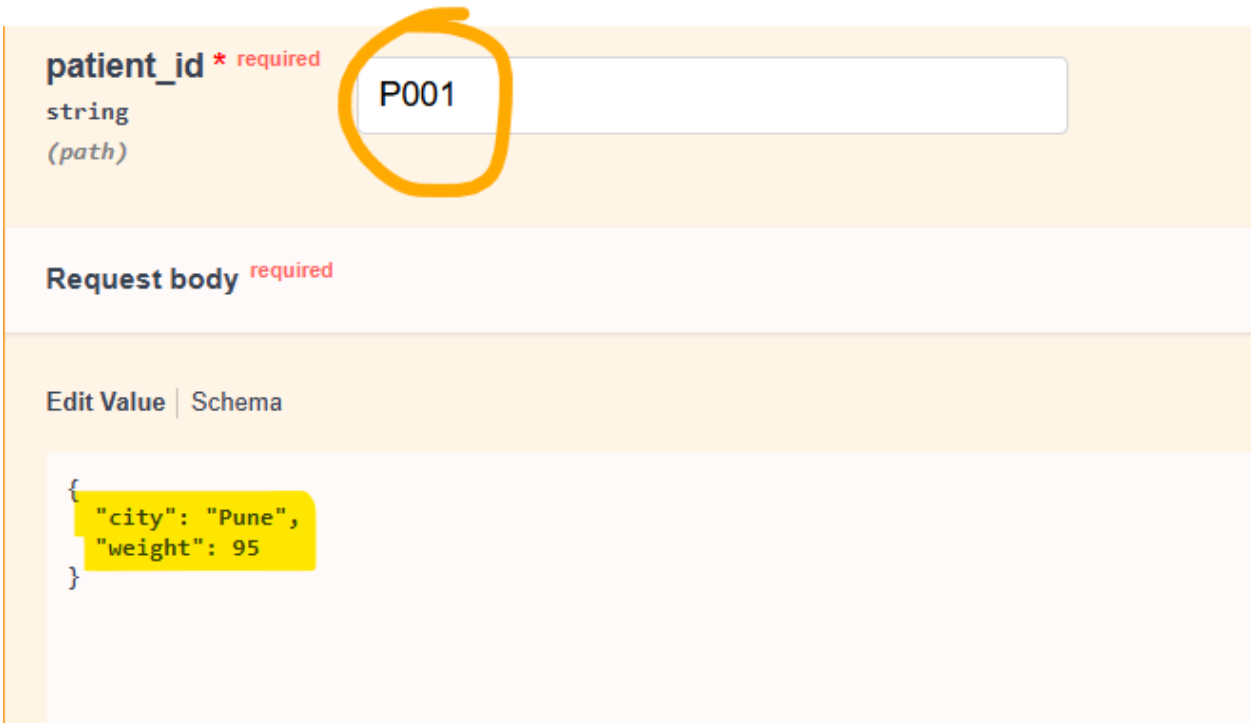
```
return JSONResponse(status_code=200, content={'message':'patient delete
d'})
```

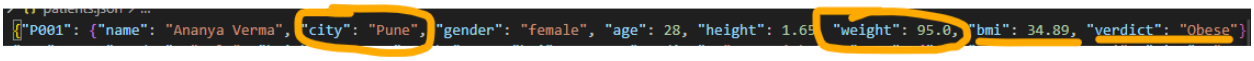
Run:

```
uvicorn FastAPI.Fast:app --reload
```



Update the patient P001:





Previous values:

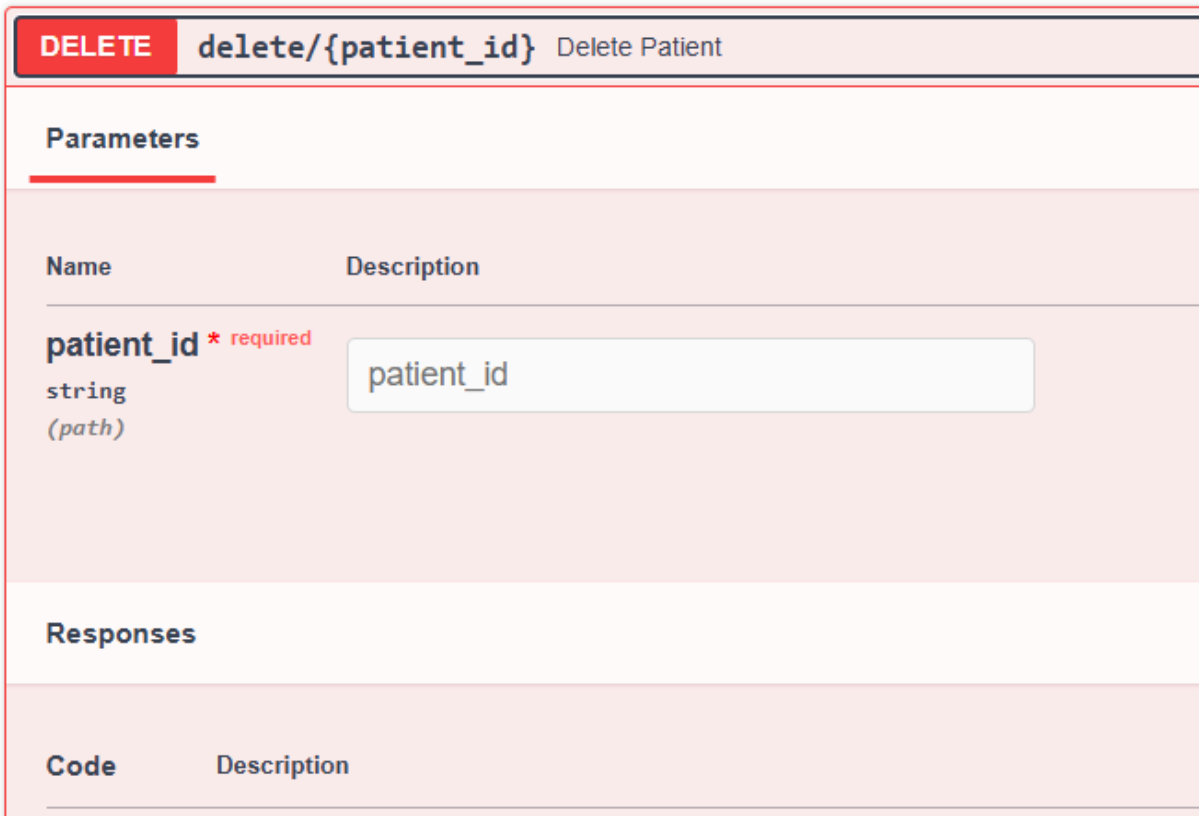
"bmi": 33.06, "verdict": "Obese"

DELETE

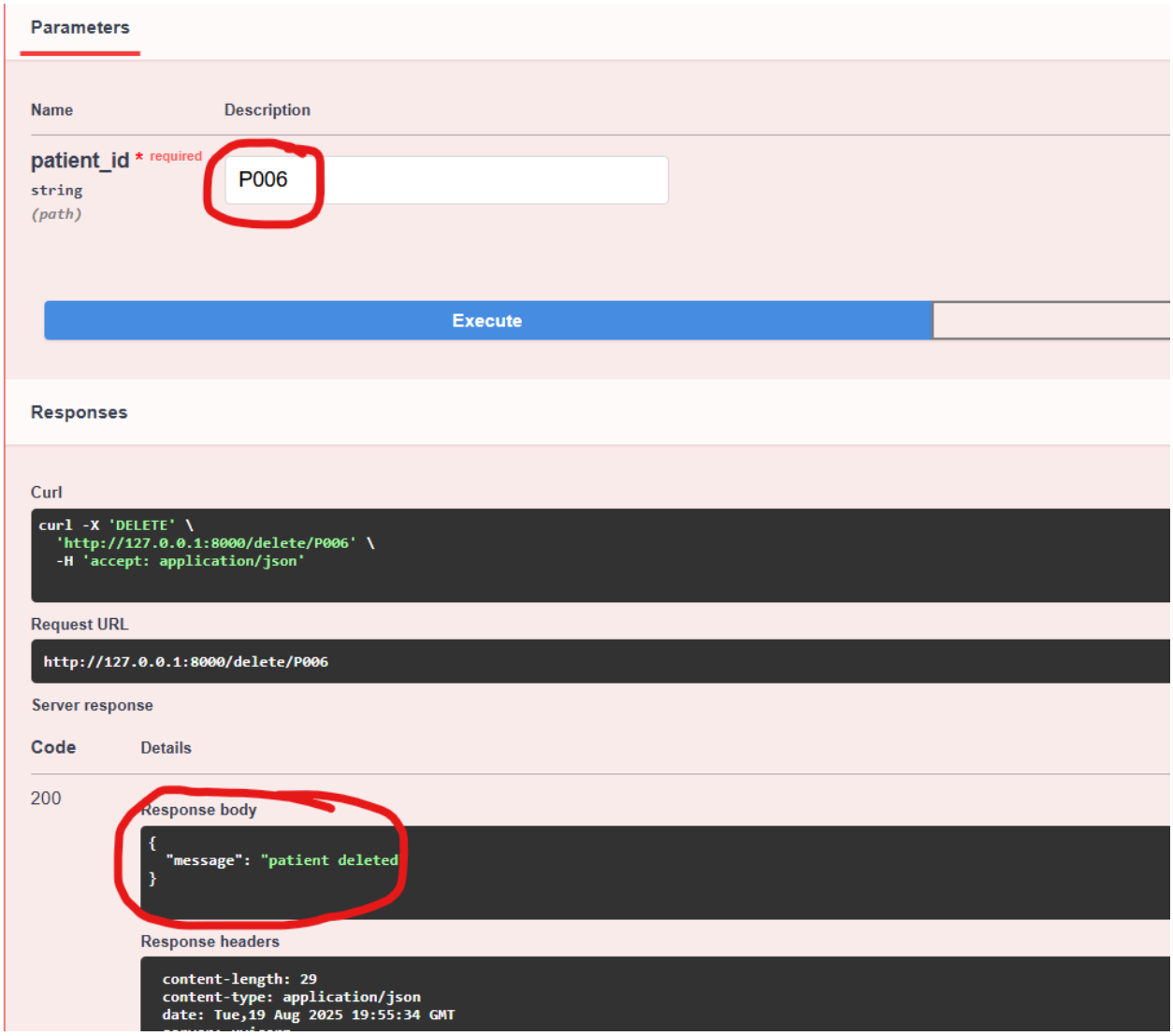
- We will delete a patient
- It will get patient id as path parameter

```
@app.delete('/delete/{patient_id}')
def delete_patient(patient_id: str):

    #load data
    data =load_data()
```

```
    if patient_id not in data:
        raise HTTPException(status_code=400, detail="PATIENT NOT FOUND")

    del data[patient_id]

    save_data(data)

    return JSONResponse(status_code=200, content={'message':'patient delet
ed'})
```



- We will try deleting P006


Parameters
Name
Description
patient_id * required
string
(path)
P006
Execute
Responses
Curl
curl -X 'DELETE' \
'http://127.0.0.1:8000/delete/P006' \
-H 'accept: application/json'
Request URL
http://127.0.0.1:8000/delete/P006
Server response
Code
Details
200
Response body
{
"message": "patient deleted
}
Response headers
content-length: 29
content-type: application/json
date: Tue,19 Aug 2025 19:55:34 GMT