Assignment 4 - FastAPI Implementation

The goal of this assignment is to create a simple API using the FastAPI framework.
In this step, we will only focus on HTTP requests and determine how different components of the program interact with one another using the Flask framework.

Turn in:

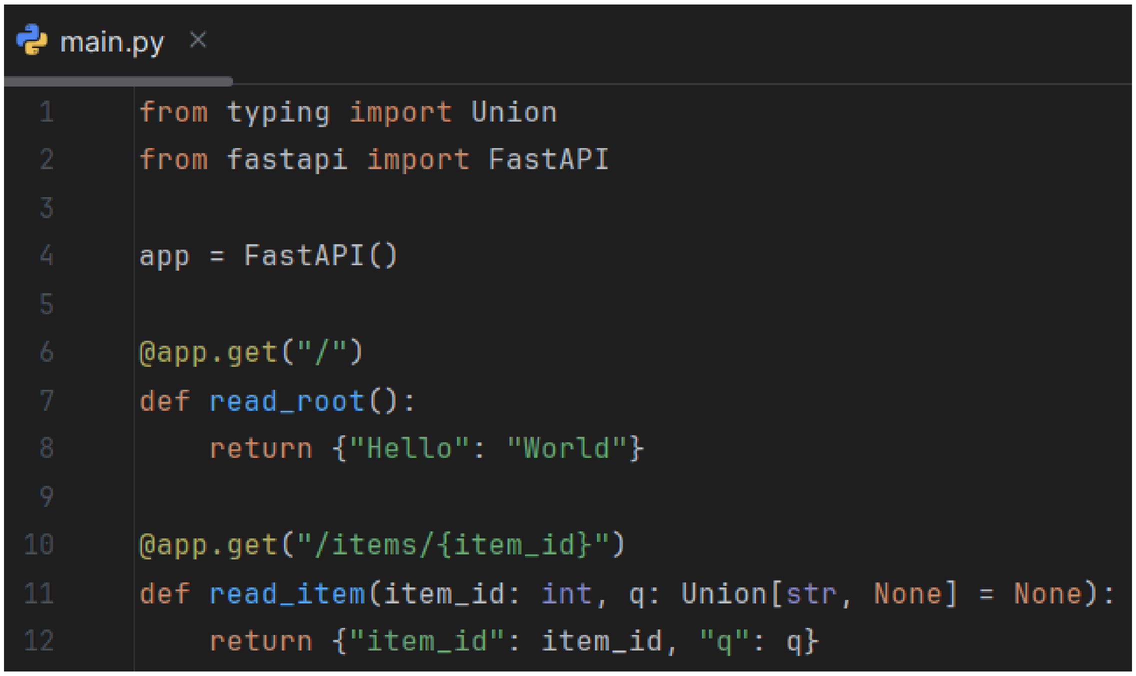
1. Your **GitHub repository** that shows your work (complete source code).
   - main.py file should be included in the assignment folder.
   - Your commit messages **must follow conventional commits**.
   - **At least 3 commit messages** are required.
   - Your last commit time must be before your submission date.
2. **Screenshots of the results** taken from the Interactive API docs
   - **Part 1 - step 5, Part 2 - step 2**
   - Run **ALL** requests in each part.

Part 1 - Setup and run the program:

1. Open your assignment project and install the following packages in the virtual environment using the 'pip' command.

   ```
   pip install fastapi
   pip install "uvicorn[standard]"
   ```

2. Create a file 'main.py' with:



main.py

```python
from typing import Union
from fastapi import FastAPI

app = FastAPI()

@app.get("/")
def read_root():
    return {"Hello": "World"}

@app.get("/items/{item_id}")
def read_item(item_id: int, q: Union[str, None] = None):
    return {"item_id": item_id, "q": q}
```

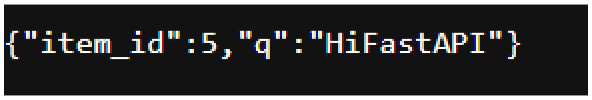
3. Run it using the following command.

   uvicorn main:app --reload

4. Check if the server is running by opening your browser at:

   http://127.0.0.1:8000/items/5?q=HiFastAPI

   Here, you will see the JSON response as:



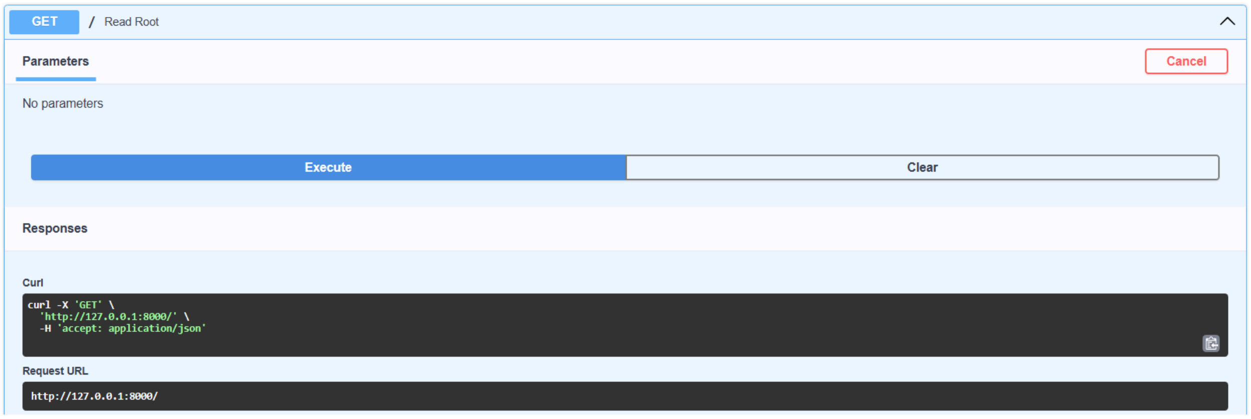
```
{"item_id":5,"q":"HiFastAPI"}
```

   You already created an API that:
   - Receives HTTP requests in the paths `/` and `/items/{item_id}`.
   - Both paths take GET operations (also known as HTTP methods).
   - The path `/items/{item_id}` has a path parameter `item_id` that should be an int.
   - The path `/items/{item_id}` has an optional str query parameter q.

5. Use Interactive API docs to send requests and see the results.

   http://127.0.0.1:8000/docs

   Click the 'Try it out' button for **both endpoints**.
   - You'll need to provide two screenshots in total, one for each endpoint, showing both the parameter tab and the response tab.
   - For the second endpoint, enter an arbitrary value, and send requests.



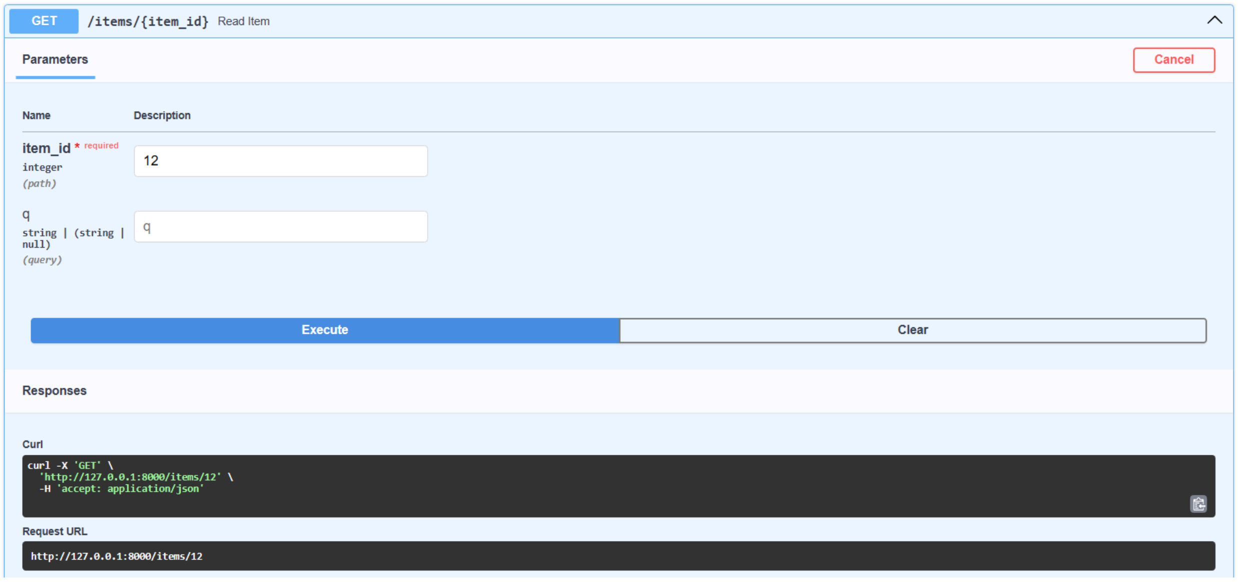
No parameters

Curl

```
curl -X 'GET' \
  'http://127.0.0.1:8000/' \
  -H 'accept: application/json'
```

Request URL

```
http://127.0.0.1:8000/
```



/items/{item_id} Read Item

Parameters

Name	Description
item_id * required integer (path)	12
q string | (string | null) (query)	q

Responses

Curl

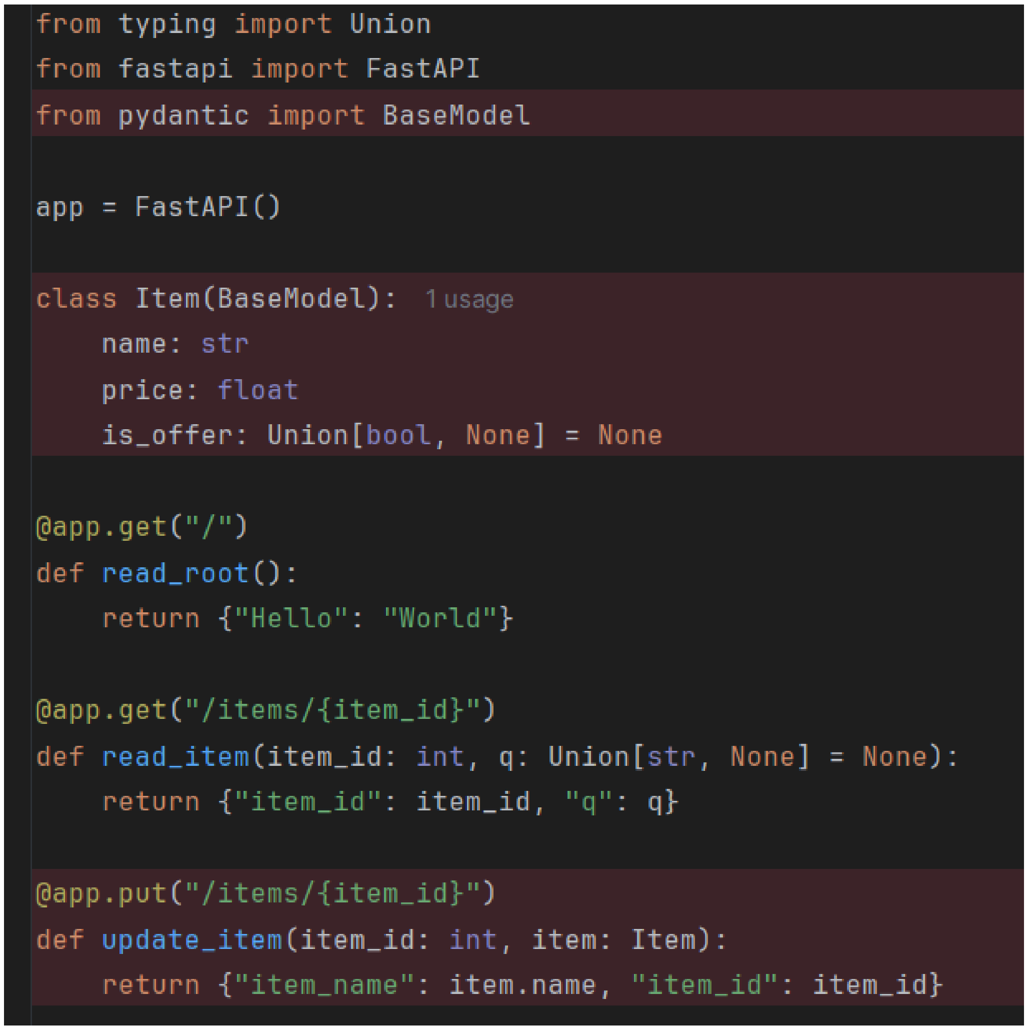
```
curl -X 'GET' \
  'http://127.0.0.1:8000/items/12' \
  -H 'accept: application/json'
```

Request URL

```
http://127.0.0.1:8000/items/12
```

Part 2 - Adding a simple class and PUT request:

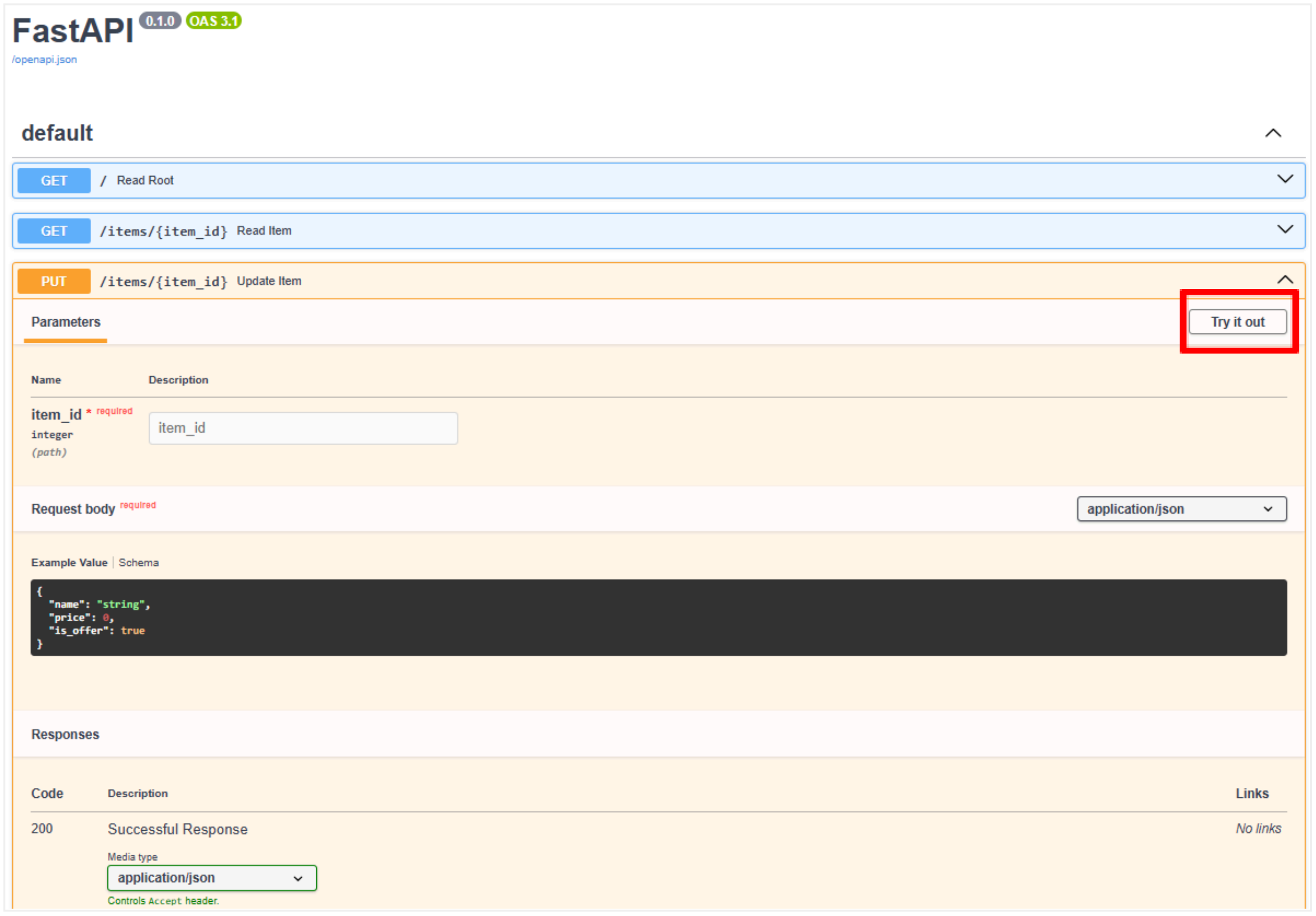
1. Now modify the file main.py to receive a body from a PUT request. Declare the body using standard Python types, thanks to Pydantic:



```python
from typing import Union
from fastapi import FastAPI
from pydantic import BaseModel

app = FastAPI()

class Item(BaseModel):  1 usage
    name: str
    price: float
    is_offer: Union[bool, None] = None

@app.get("/")
def read_root():
    return {"Hello": "World"}

@app.get("/items/{item_id}")
def read_item(item_id: int, q: Union[str, None] = None):
    return {"item_id": item_id, "q": q}

@app.put("/items/{item_id}")
def update_item(item_id: int, item: Item):
    return {"item_name": item.name, "item_id": item_id}
```

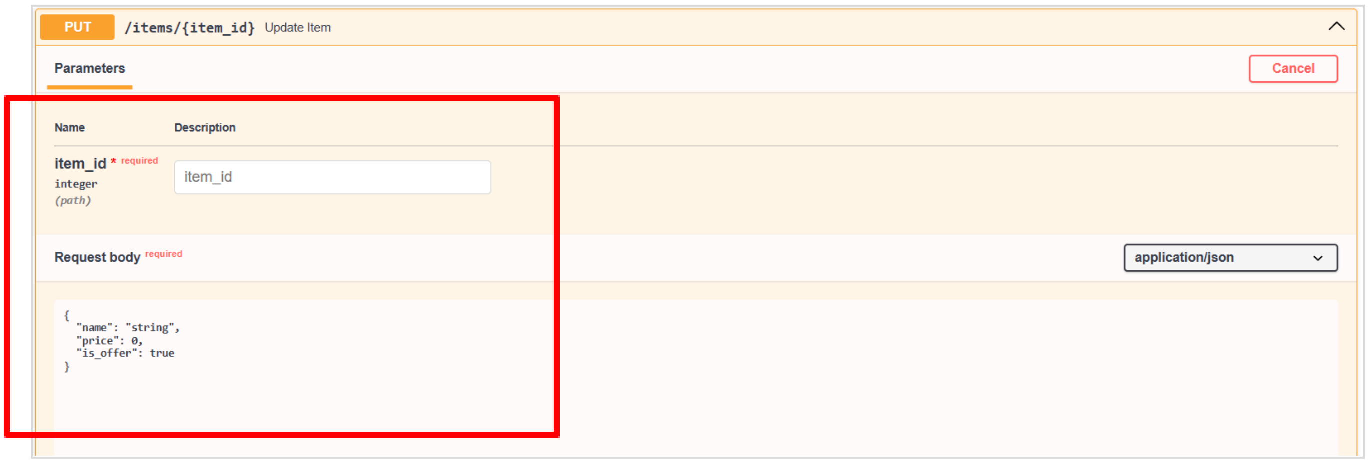
The server should reload automatically
(because you added --reload to the uvicorn command above).

2. Use Interactive API docs to send requests and see the results.
The interactive API documentation will be automatically updated, including the new body:

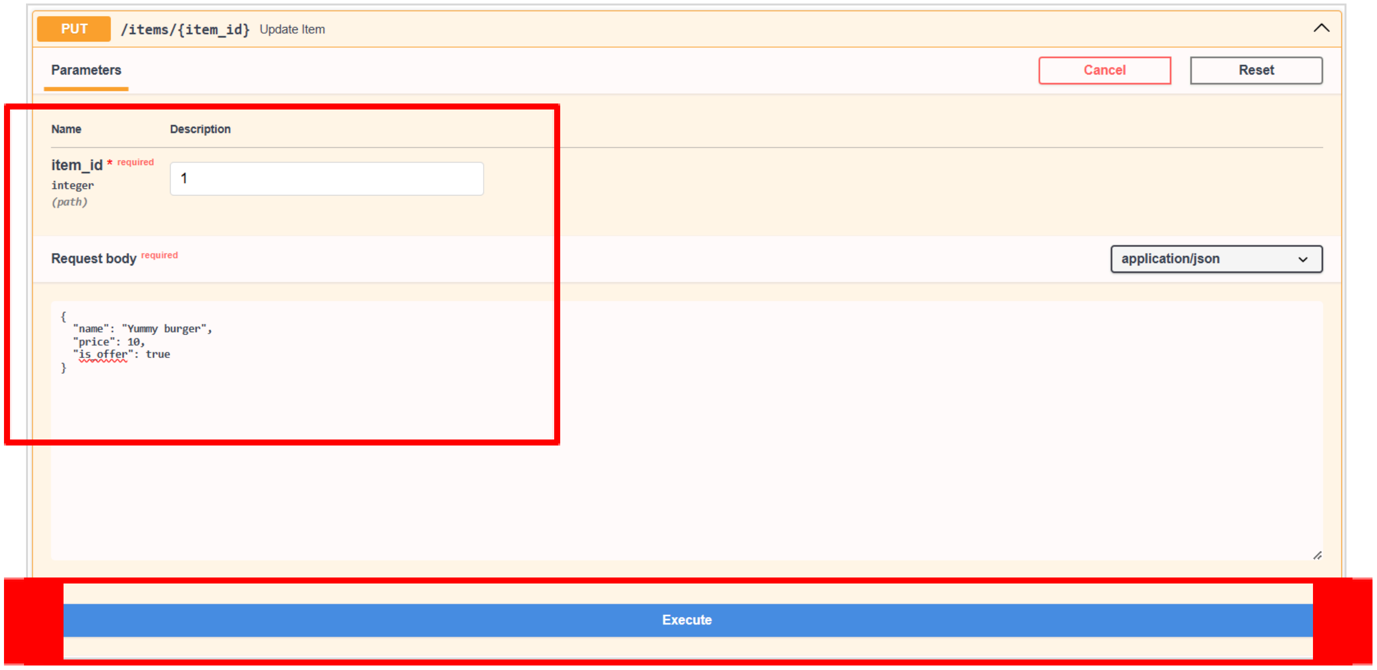


```
{
  "name": "string",
  "price": 0,
  "is_offer": true
}
```

Click on the button "Try it out", it allows you to fill the parameters and directly interact with the API:

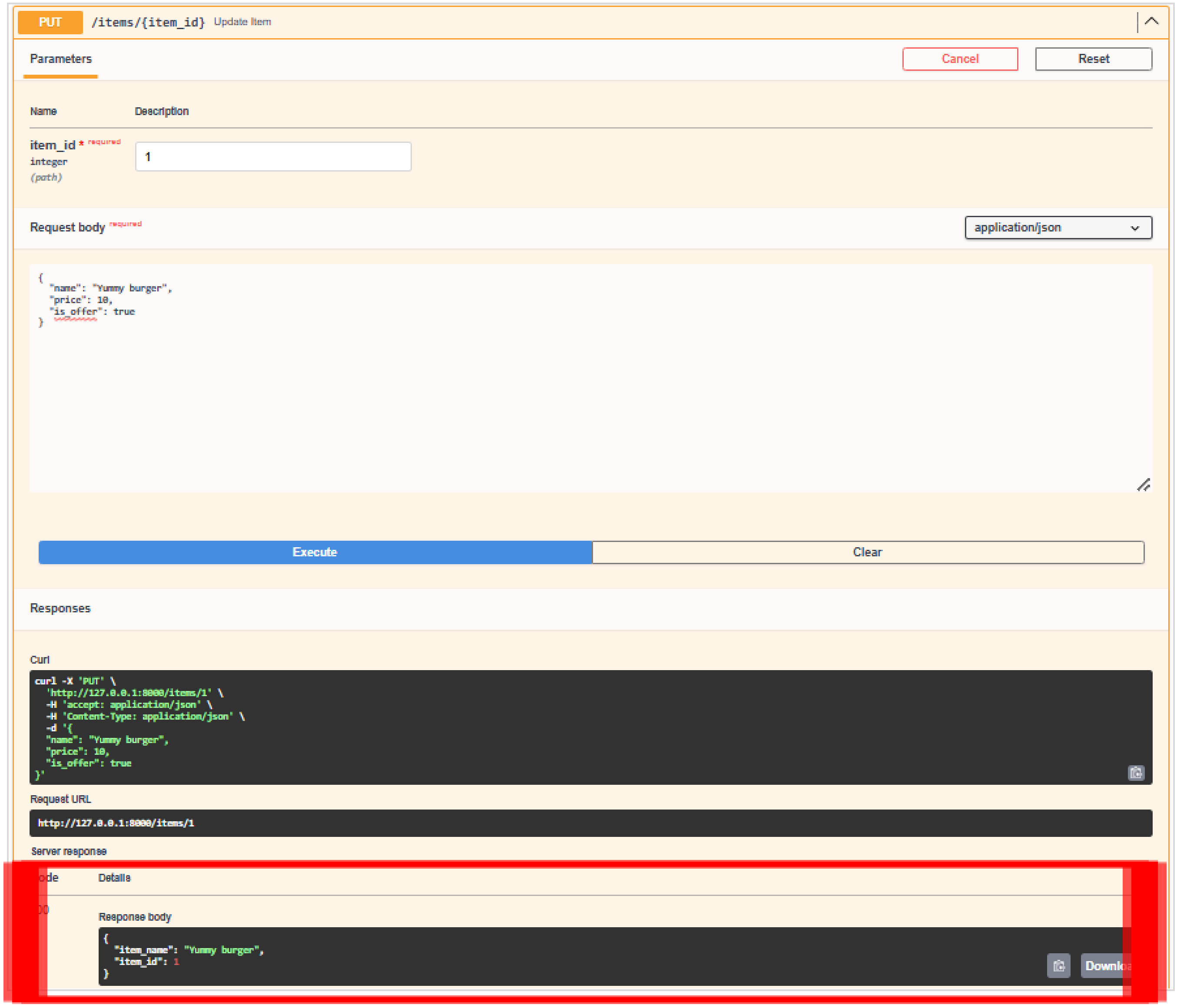


```
{
  "name": "string",
  "price": 0,
  "is_offer": true
}
```



```
{
  "name": "Yummy burger",
  "price": 10,
  "is_offer": true
}
```

Then click on the "Execute" button, the user interface will communicate with your API, send the parameters, get the results and show them on the screen:



`/items/{item_id}`

item_id * (required)
integer
(path)

1

Request body (required)

```
{
  "name": "Yummy burger",
  "price": 10,
  "is_offer": true
}
```

Responses

Curl

```
curl -X 'PUT' \
  'http://127.0.0.1:8000/items/1' \
  -H 'accept: application/json' \
  -H 'Content-Type: application/json' \
  -d '{
  "name": "Yummy burger",
  "price": 10,
  "is_offer": true
}'
```

Request URL

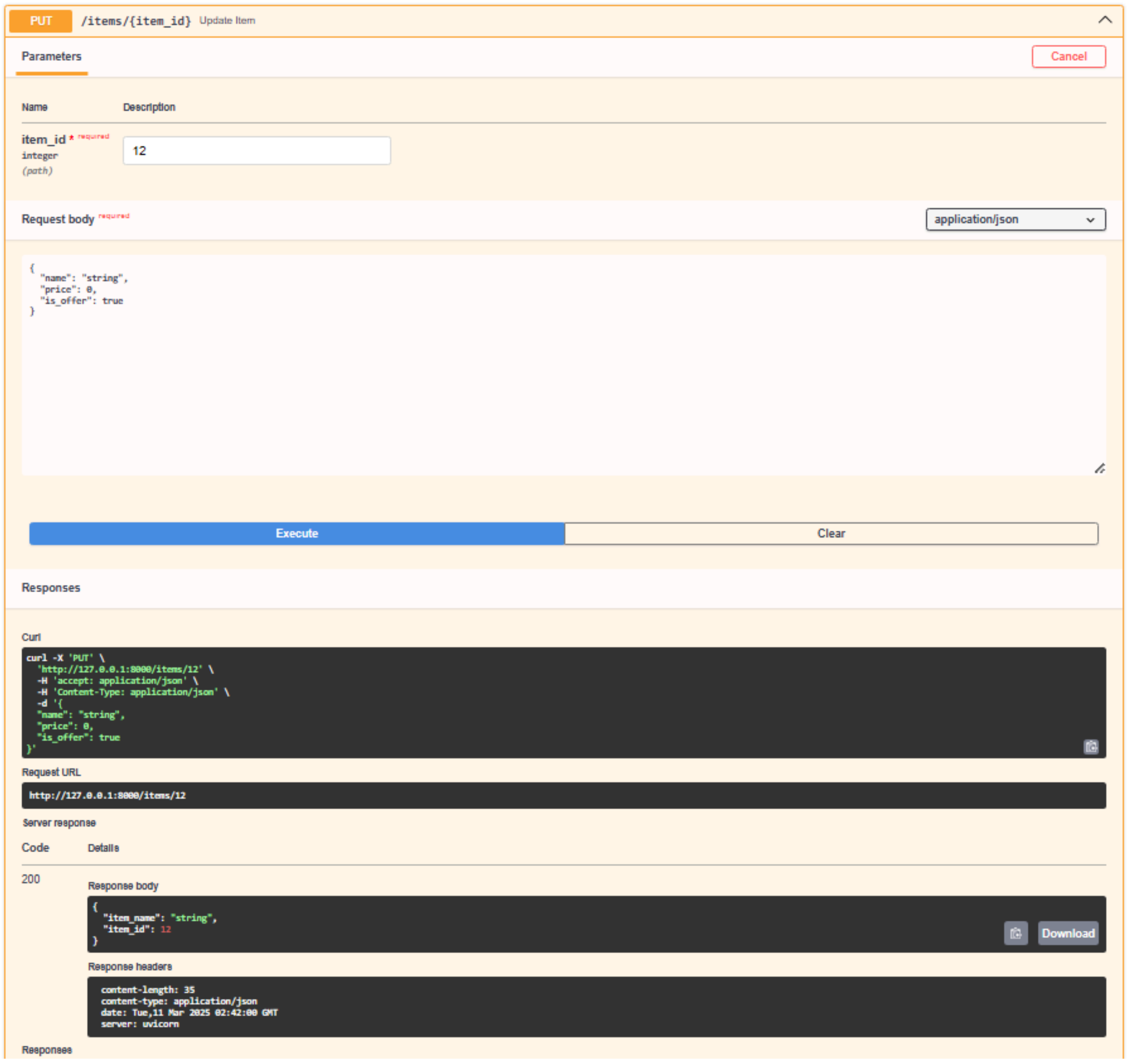
```
http://127.0.0.1:8000/items/1
```

Server response

ode	Details
00	Response body

```
{
  "item_name": "Yummy burger",
  "item_id": 1
}
```

Screenshot of functionality (part 2, put):



item_id * required
integer
(path)

12

Request body required

application/json

```
{
  "name": "string",
  "price": 0,
  "is_offer": true
}
```

Curl

```
curl -X 'PUT' \
  'http://127.0.0.1:8000/items/12' \
  -H 'accept: application/json' \
  -H 'Content-Type: application/json' \
  -d '{
  "name": "string",
  "price": 0,
  "is_offer": true
}'
```

Request URL

```
http://127.0.0.1:8000/items/12
```

Server response

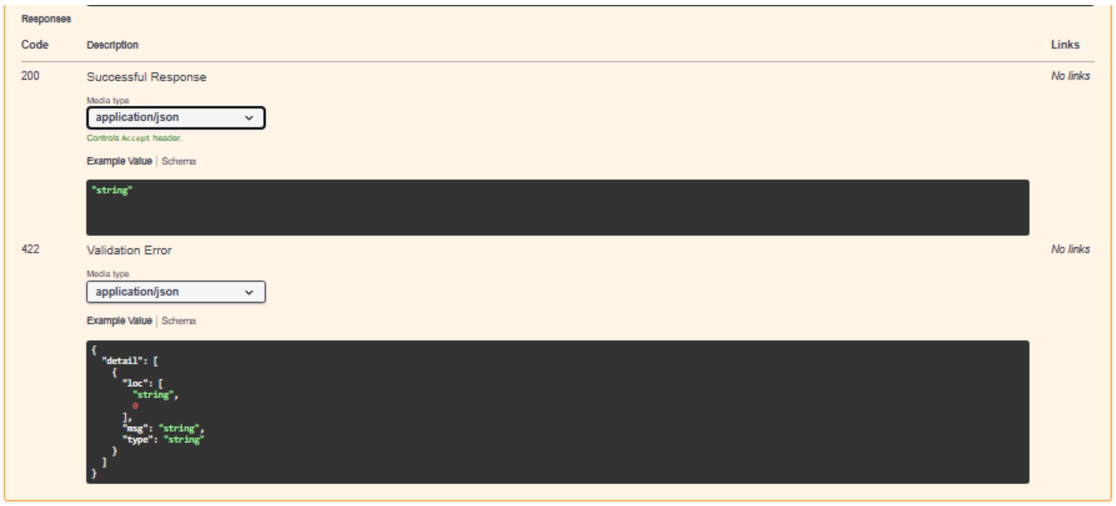
Code	Details
200	Response body

```
{
  "item_name": "string",
  "item_id": 12
}
```

Response headers

```
content-length: 35
content-type: application/json
date: Tue,11 Mar 2025 02:42:00 GMT
server: uvicorn
```

Responses


Responses

Code	Description	Links
200	Successful Response	No links
422	Validation Error	No links

200 Successful Response

Media type
application/json
Controls Accept header.

Example Value | Schema

```
"string"
```

422 Validation Error

Media type
application/json

Example Value | Schema

```
{
  "detail": [
    {
      "loc": [
        "string",
        0
      ],
      "msg": "string",
      "type": "string"
    }
  ]
}
```