**How to create FASTAPI with Uvicorn in VS Code**

**Step 1: created a folder named it as fastapi\_CRUD\_app**

**Step 2: created a file named it as main.py in vs code**

**Step 3: wrote python code inside the main.py**

For first time we should install fastapi by using below command in vs code terminal

**pip install fastapi Uvicorn**

later write below code in main.py

Example:

from fastapi import FastAPI, HTTPException

from pydantic import BaseModel

from typing import List

app = FastAPI() #starting webserver

@app.get("/")

def get\_items():

    return {"message": "Hello from GET"}

 You're importing FastAPI to create your app, and HTTPException to handle errors.

 BaseModel from Pydantic is used to validate data.

**Step 4: write the command in vs code terminal / powershell to run the server**

python -m uvicorn main:app --reload

**Step 5:** Now server will open **Uvicorn running on http://127.0.0.1:8000**

**(Press CTRL+click link to open response output page)**

**(Press CTRL+C to quit)**

Great idea! Let’s test the POST /items/ endpoint by adding some data manually and seeing the result.

**🧪 How to Test POST in FastAPI**

FastAPI provides a built-in web interface called **Swagger UI** at:

http://127.0.0.1:8000/docs

Here’s how you can do it:

**✅ Step-by-Step to Add an Item**

1. **Run your FastAPI app** using this command:

Python –m uvicorn main:app --reload

1. **Open your browser** and go to:

http://127.0.0.1:8000/docs

1. **Click on POST /items/**
2. Click "Try it out" and add data like this:

{

"id": 1,

"name": "Laptop",

"price": 1500.99

}

1. Click **Execute**, and you'll see a response like:

{

"id": 1,

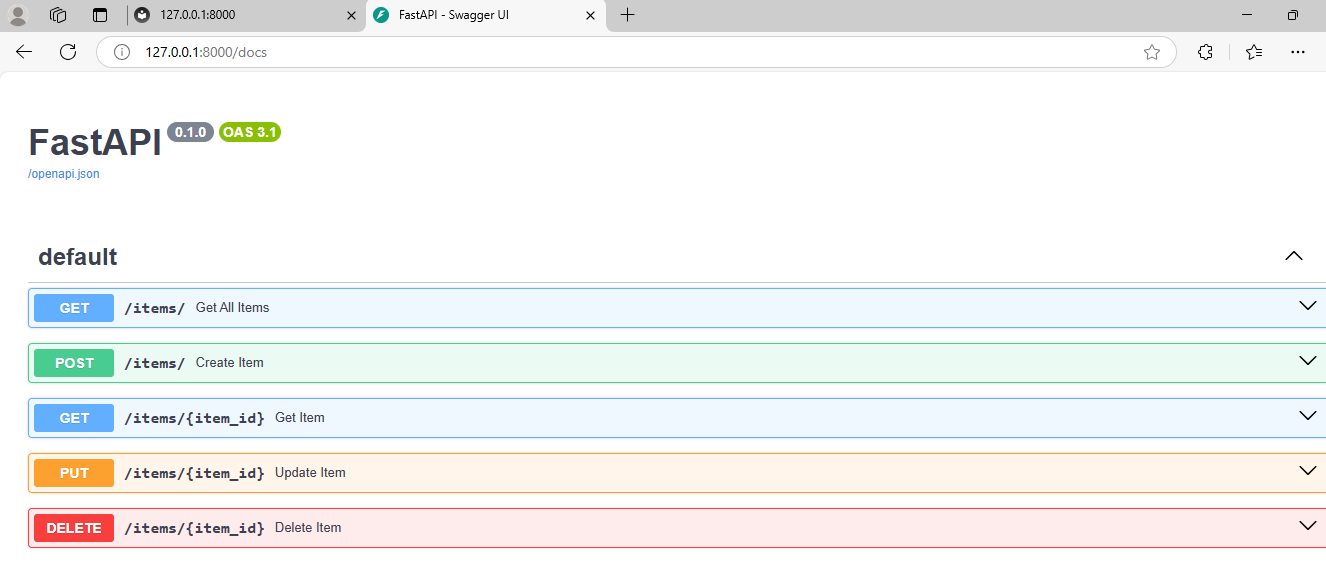
"name": "Laptop",

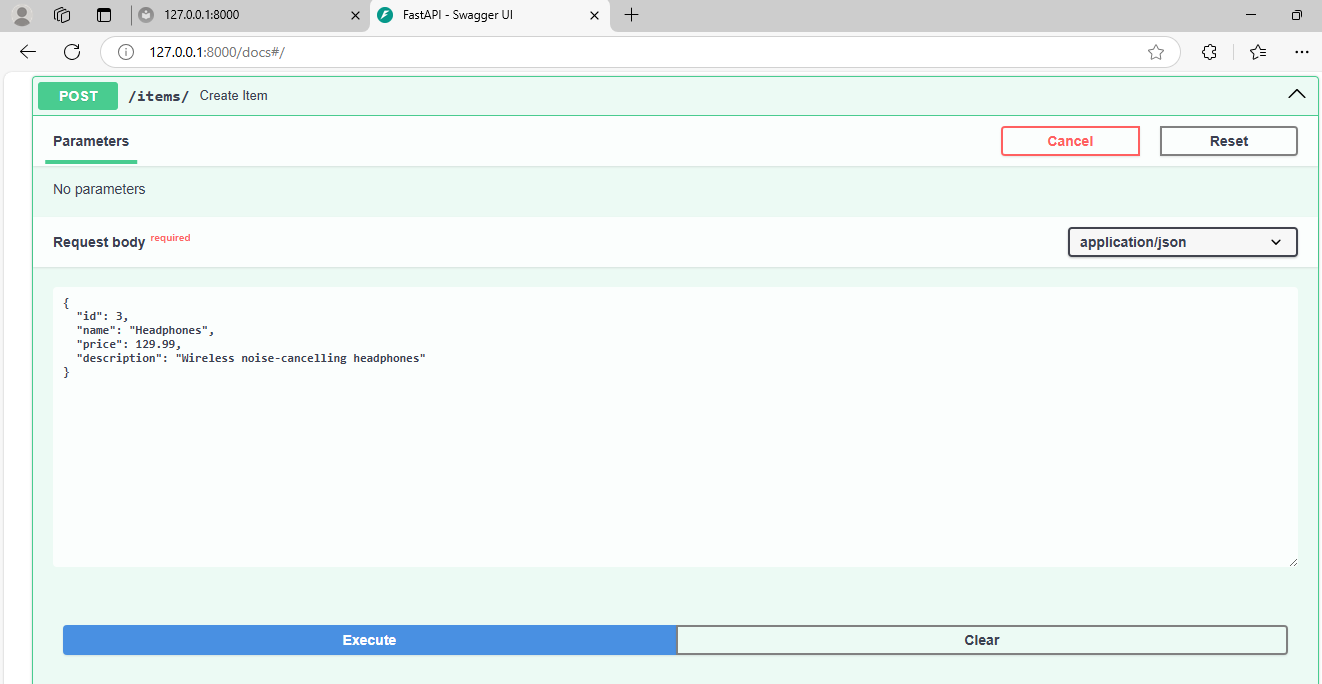
"price": 1500.99

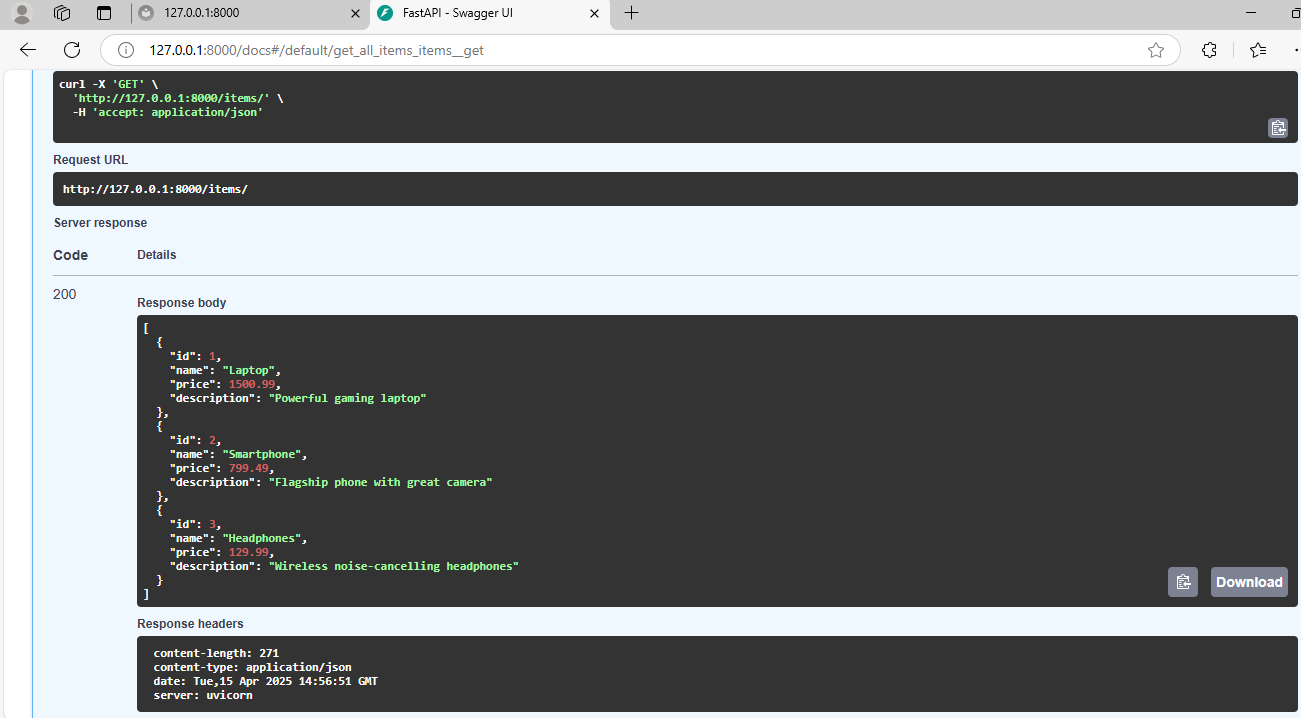
}

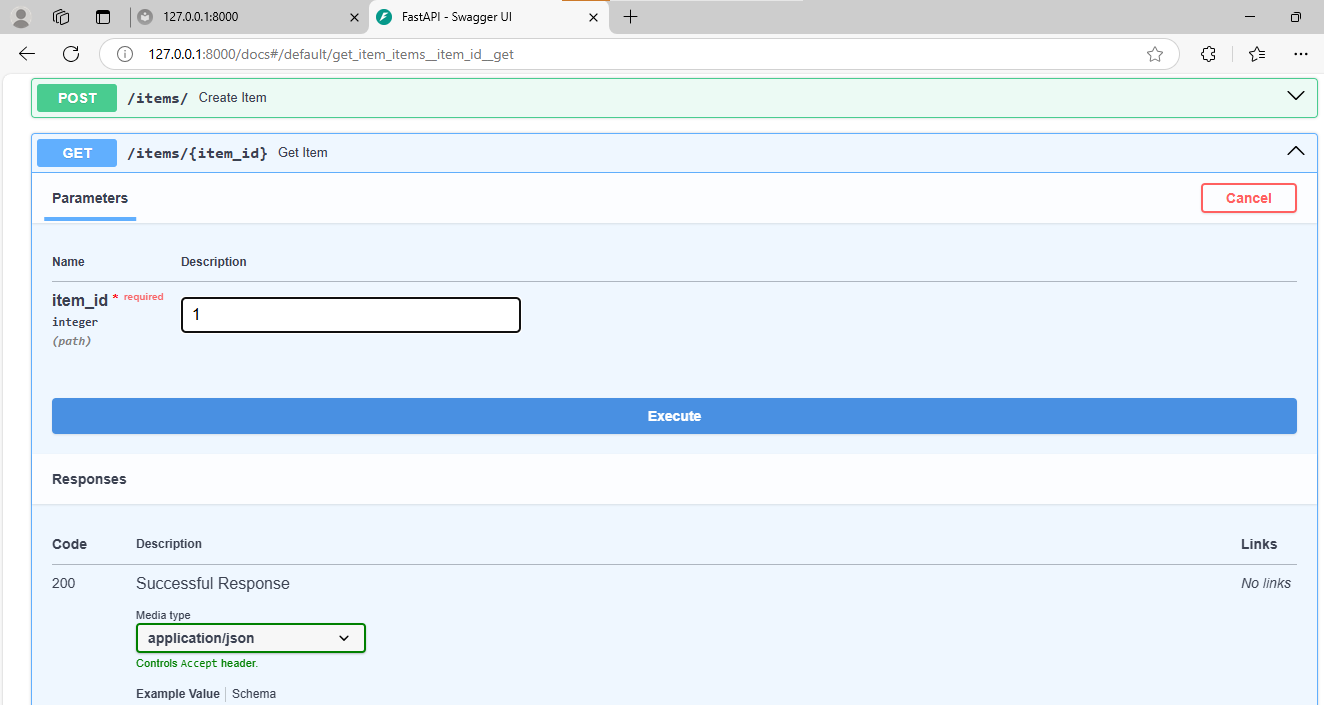
**🧠 What Happened?**

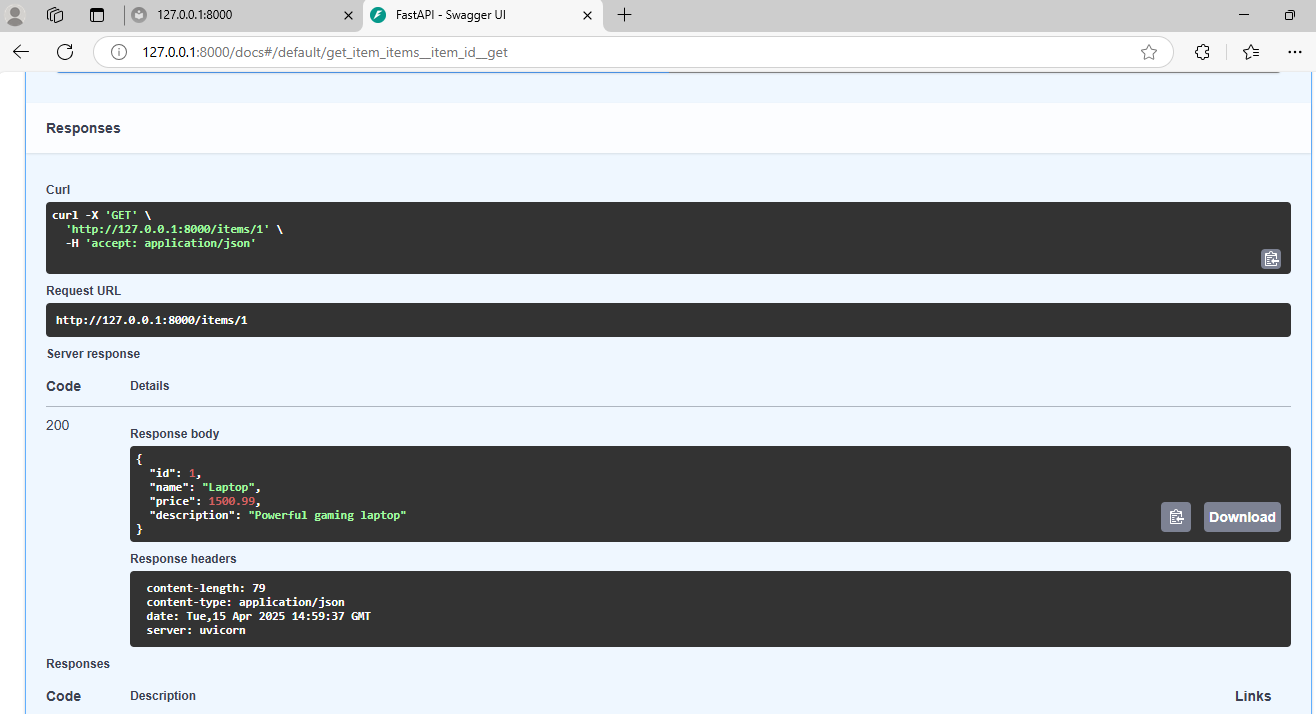
* This sends a POST request to your API.
* The item gets added to the in-memory list items\_db.
* You see the response returned from create\_item().
* **Testing output with Swagger UI**

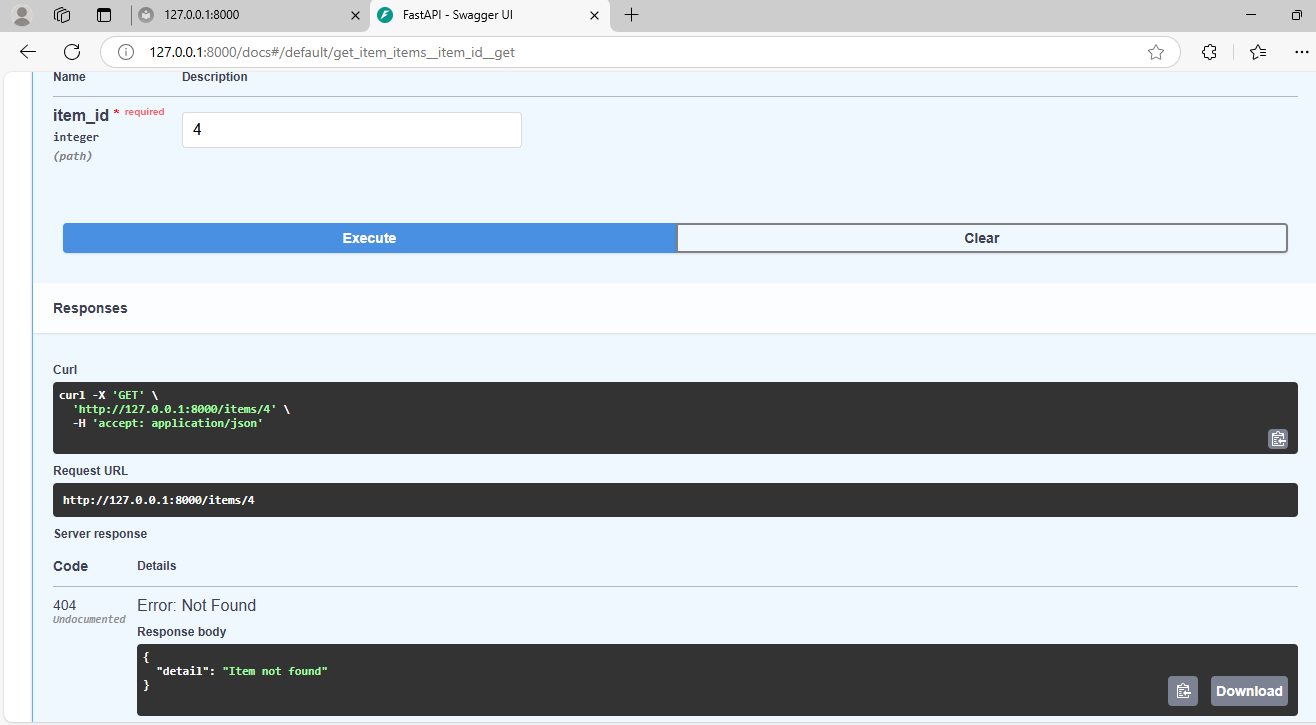
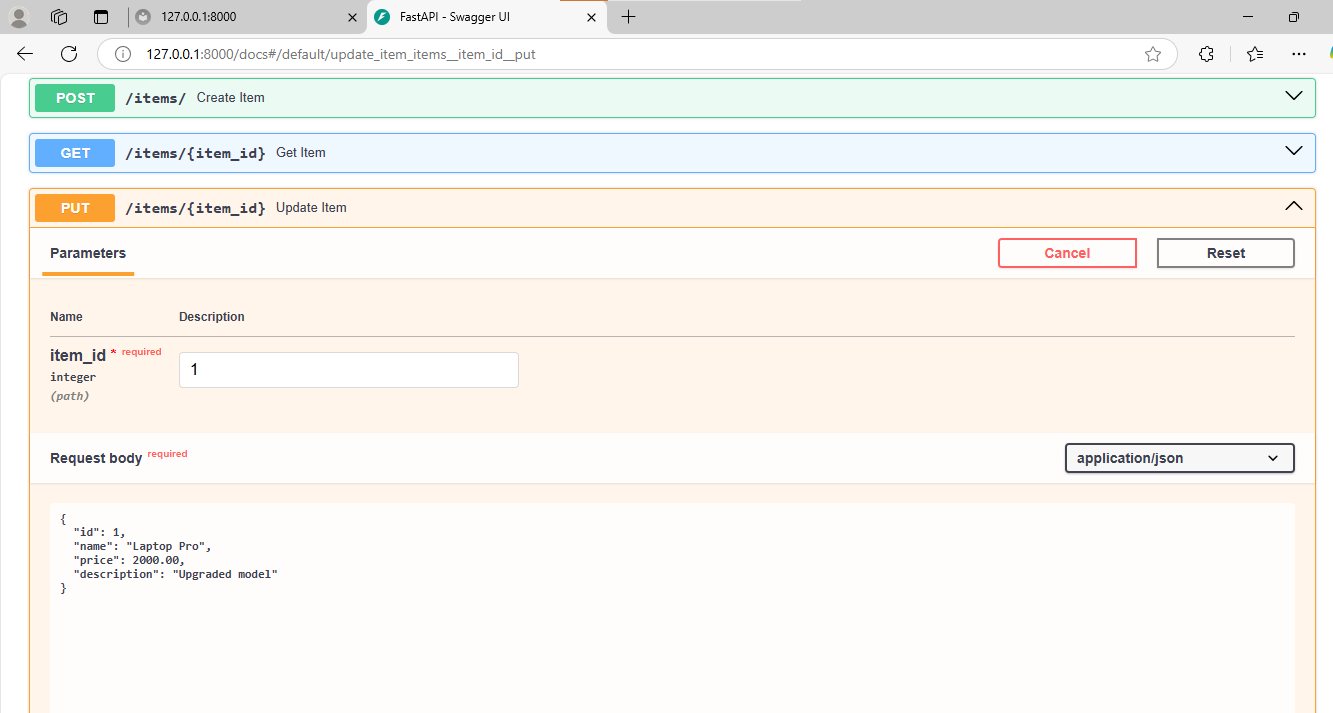
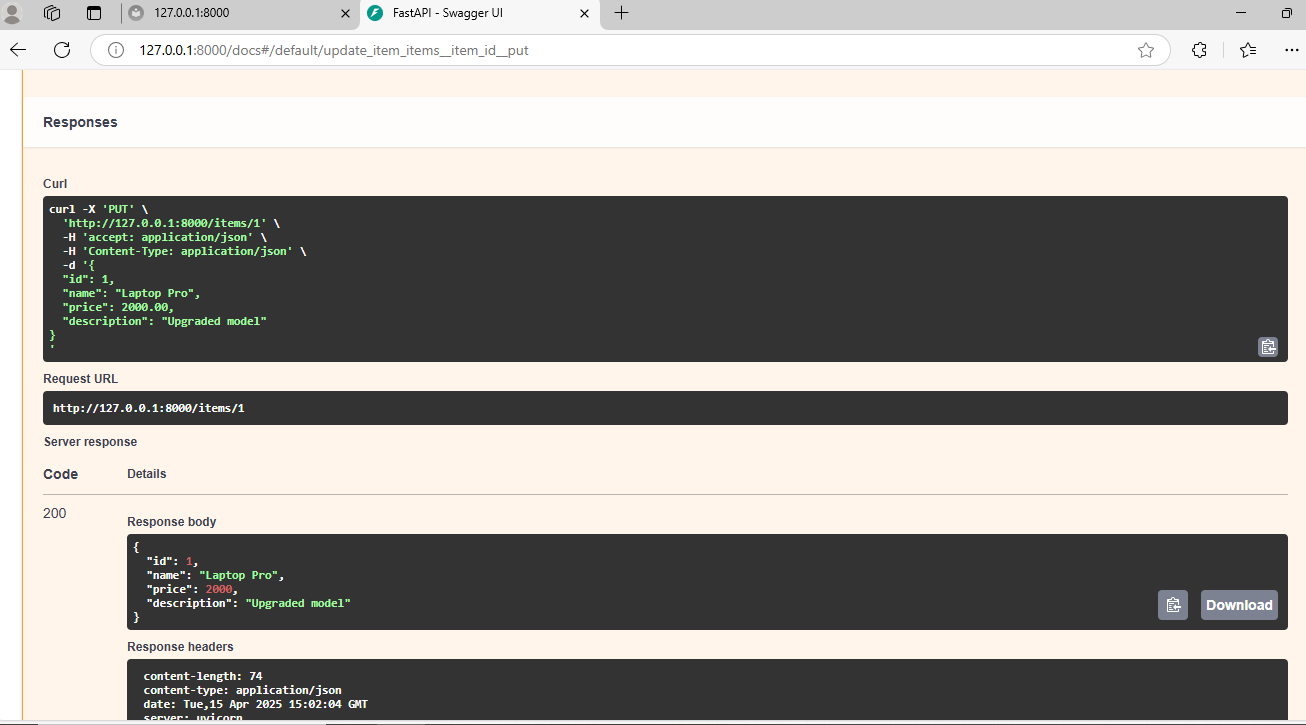
****

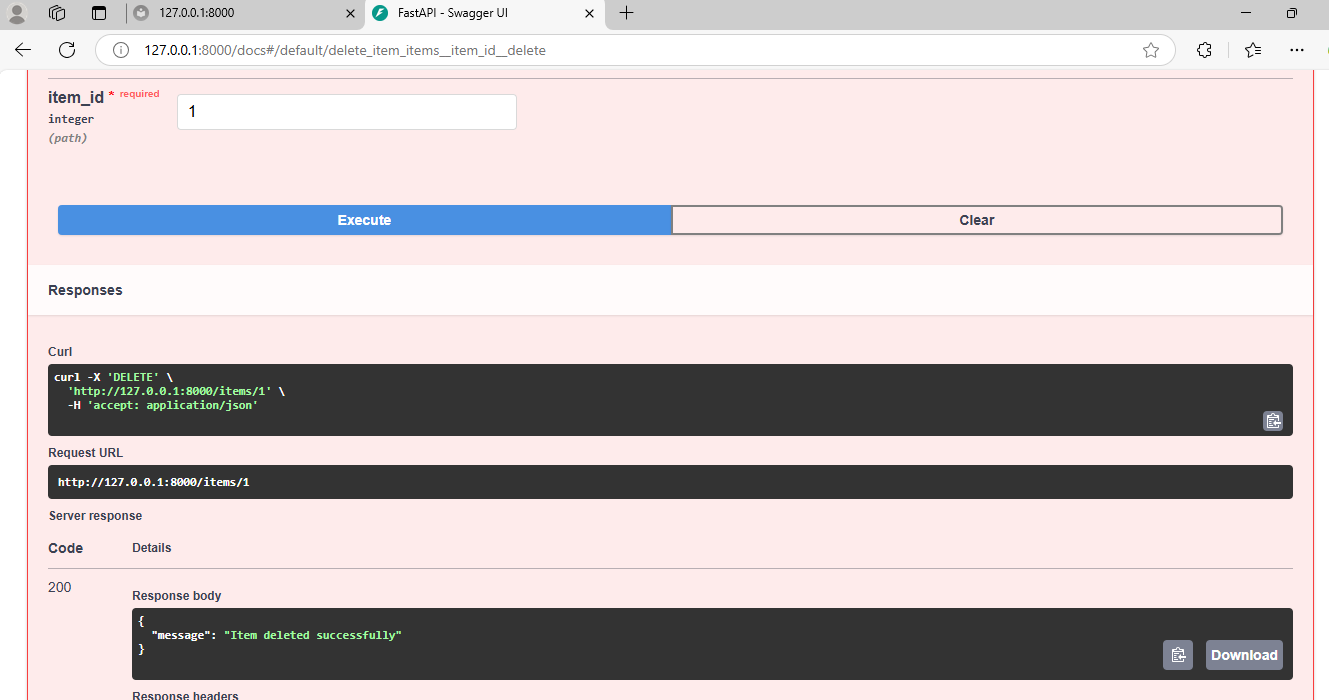
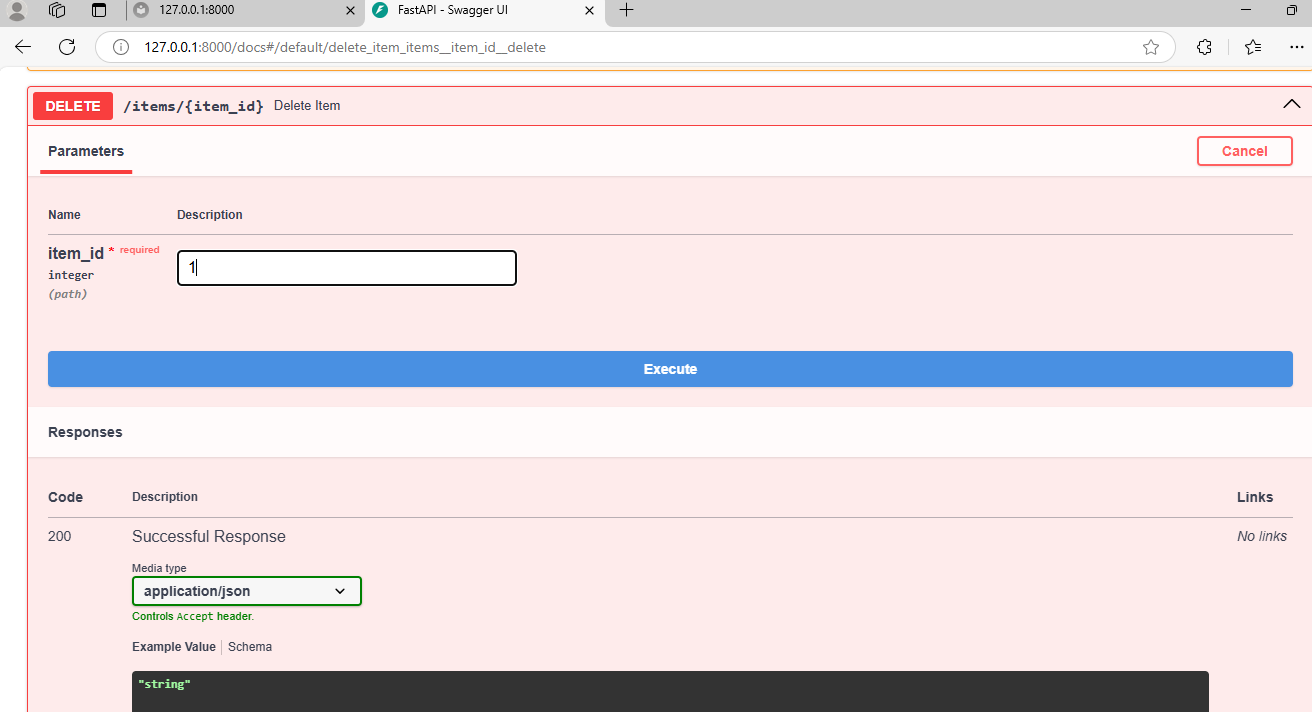


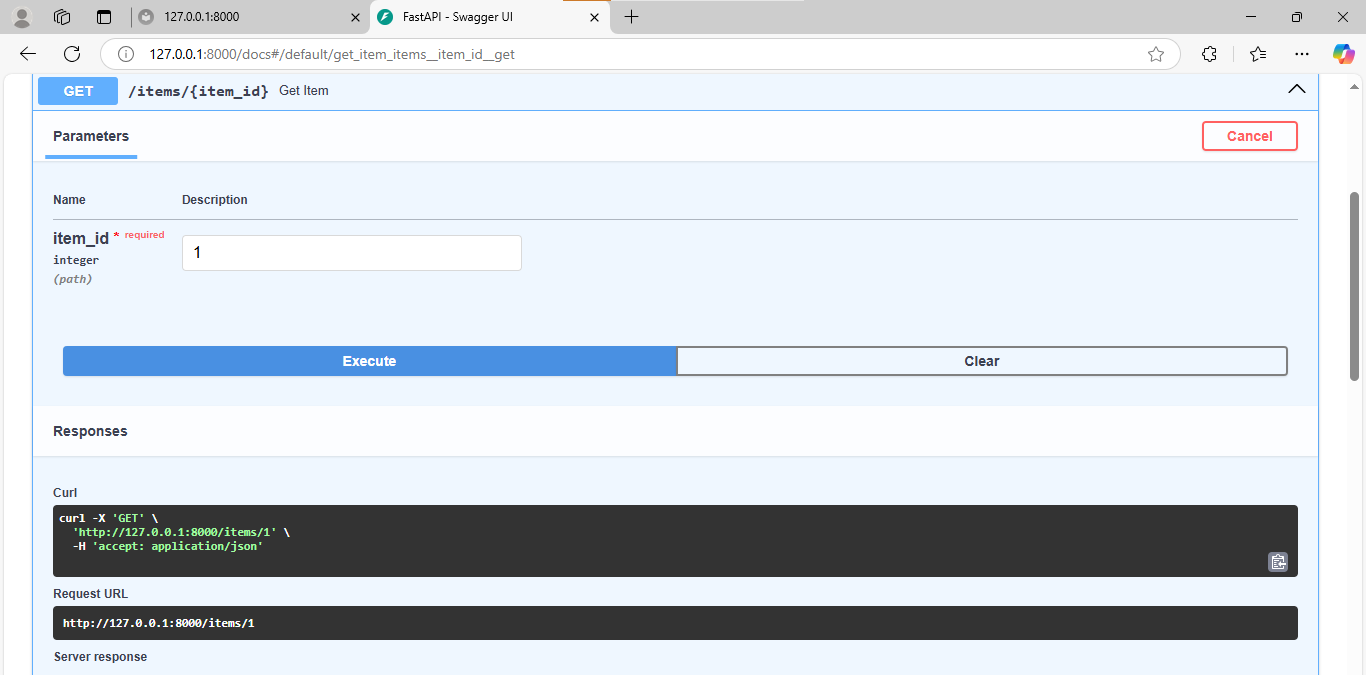


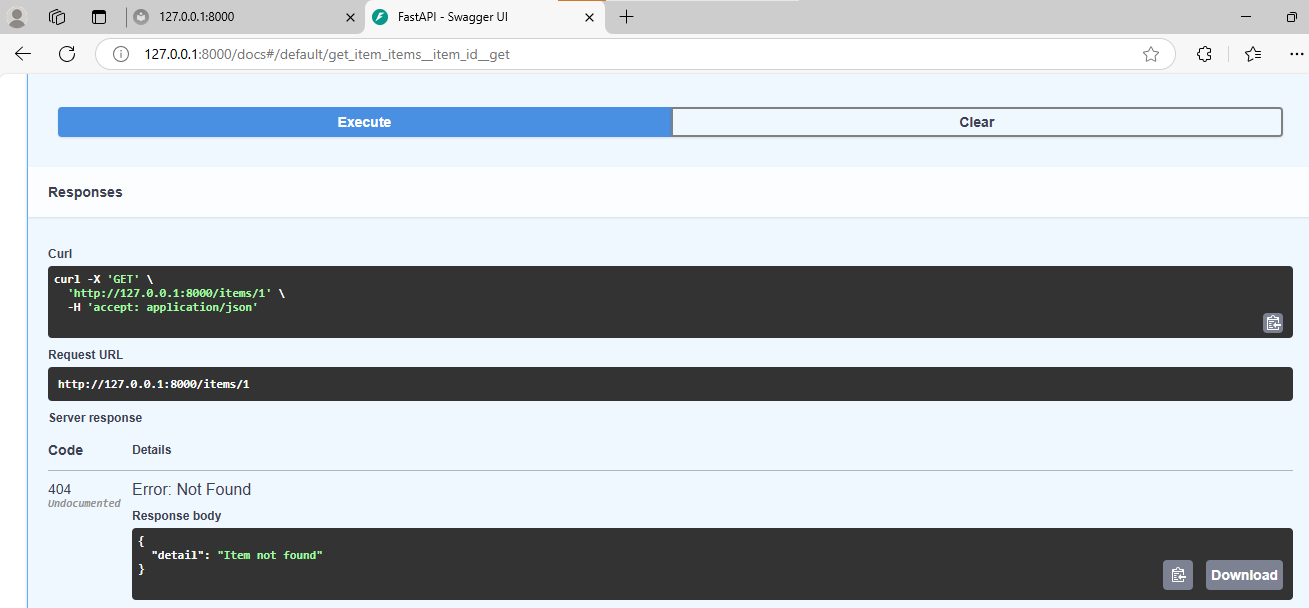




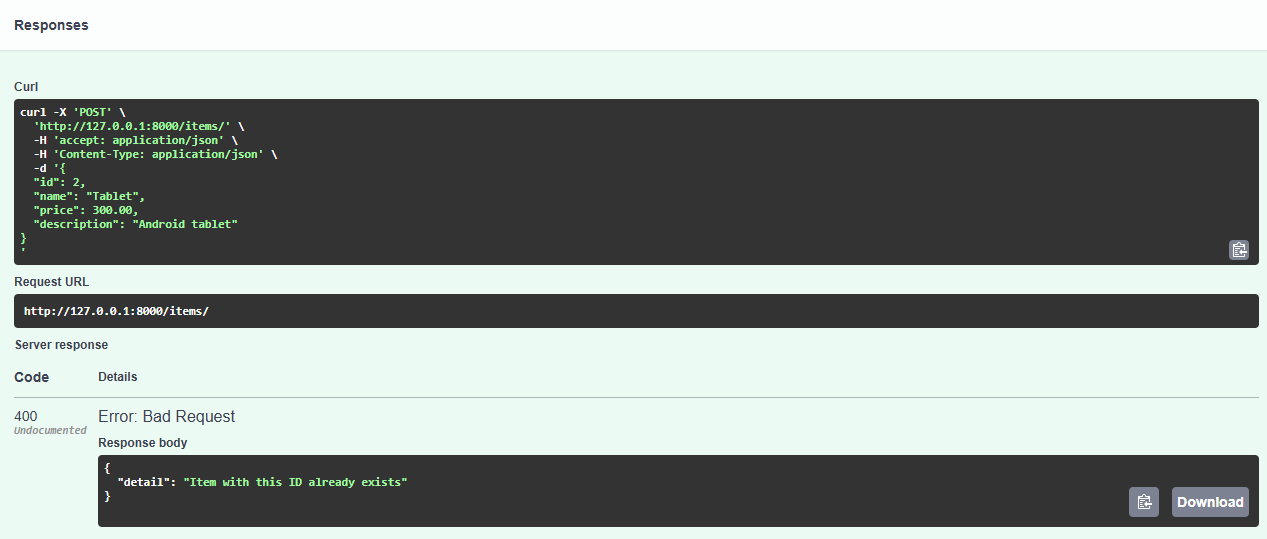




Test **invalid inputs** to check how well your API handles errors

## 🚫 1. **Duplicate ID (POST)**

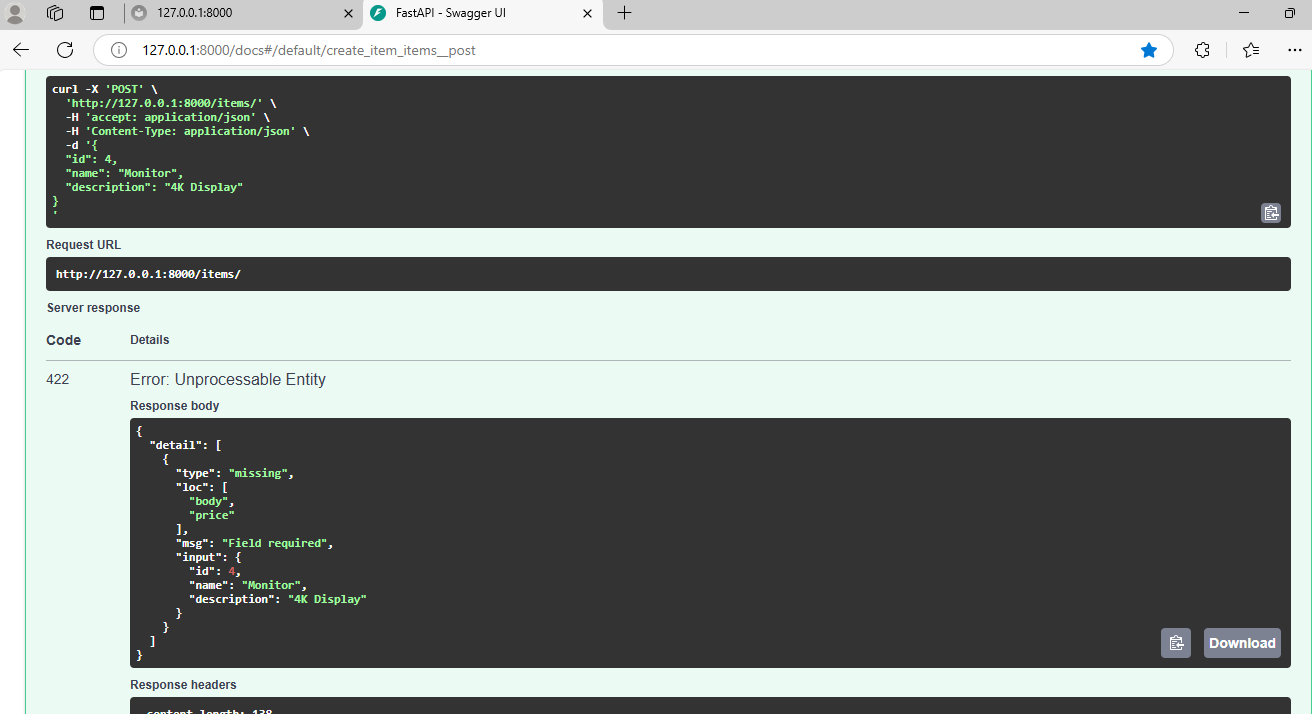
### ➡ Try to create this (again, with id = 1 that already exists):



✅ This shows the API correctly blocks duplicate entries.

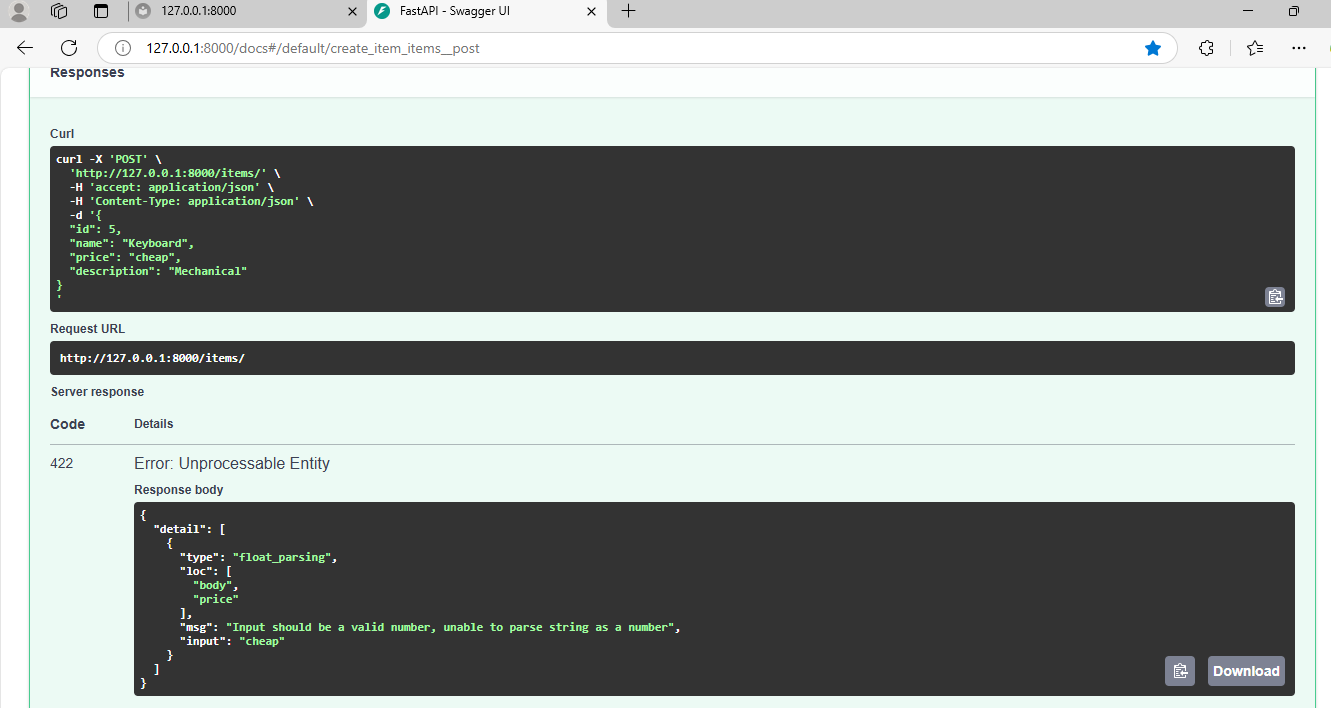
## 🚫 2. **Missing Required Field (POST)**

### ➡ Remove price field and try:



## 🚫 3. **Wrong Data Type (POST)**

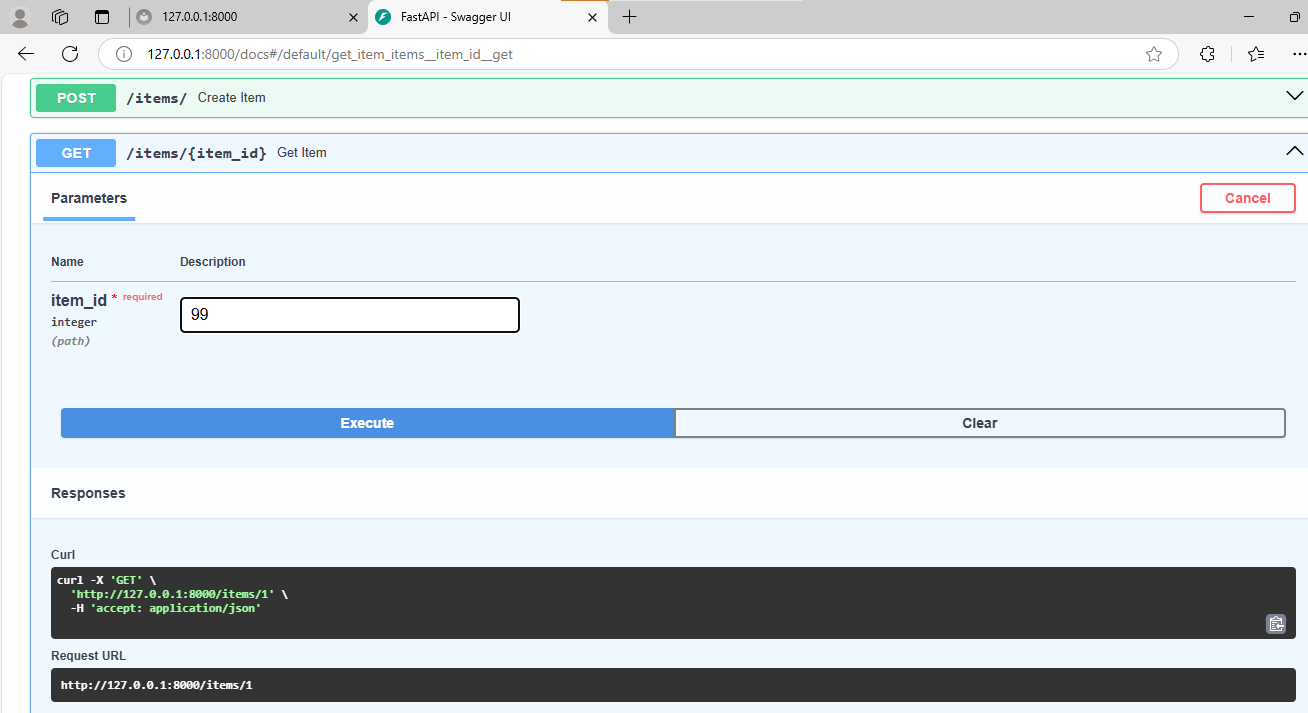
### ➡ Set price to a string (should be a number):

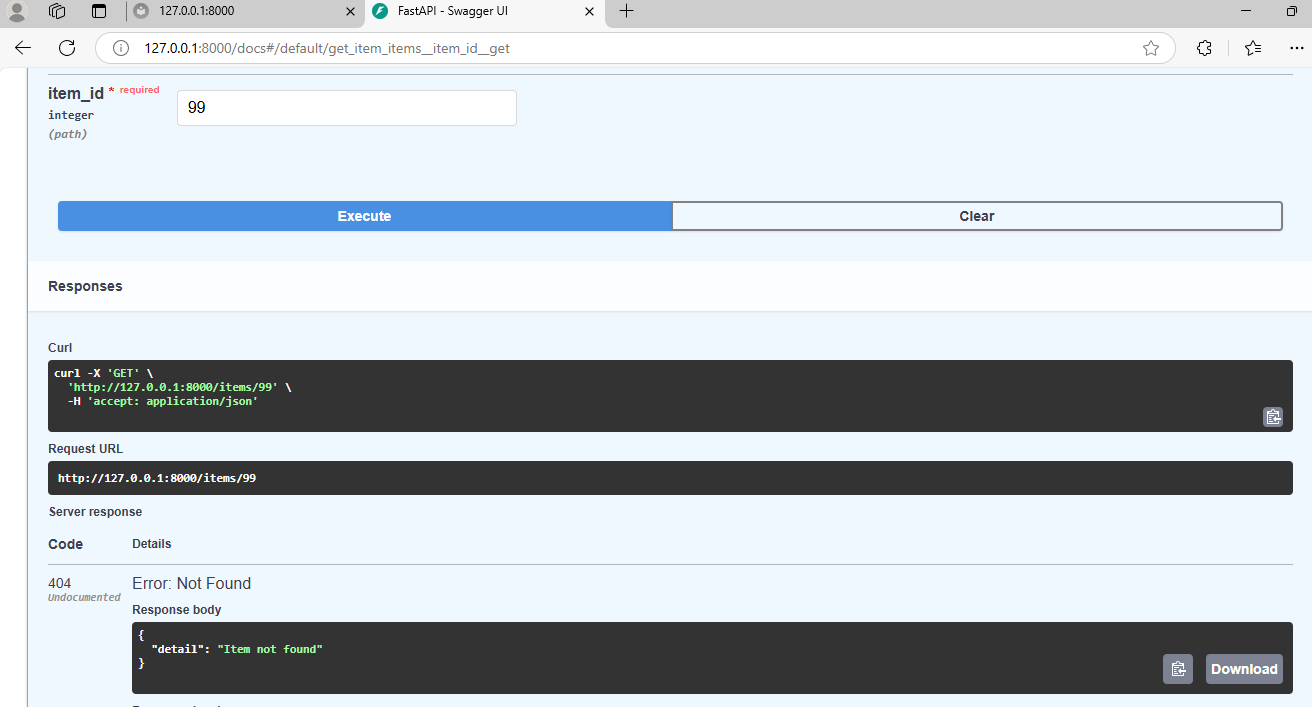


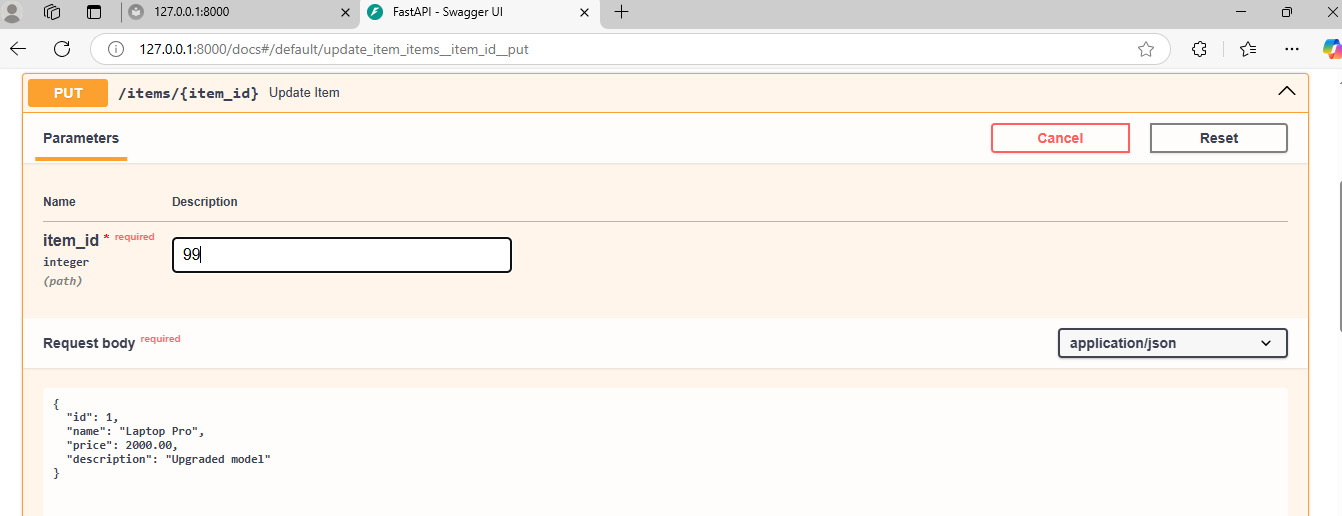
✅ Pydantic ensures data types are validated.

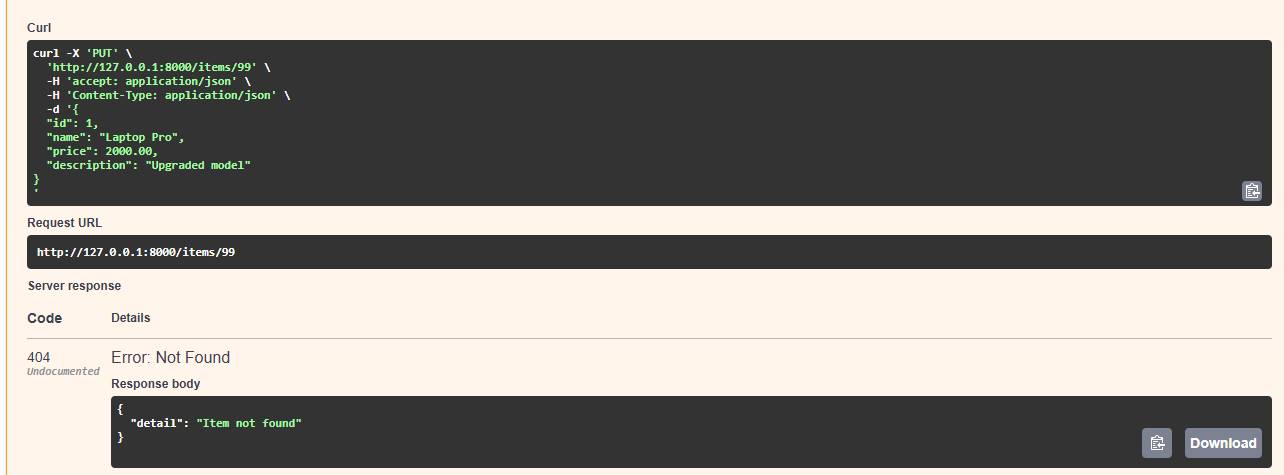
## 🚫 4. **Non-existent ID (GET/PUT/DELETE)**

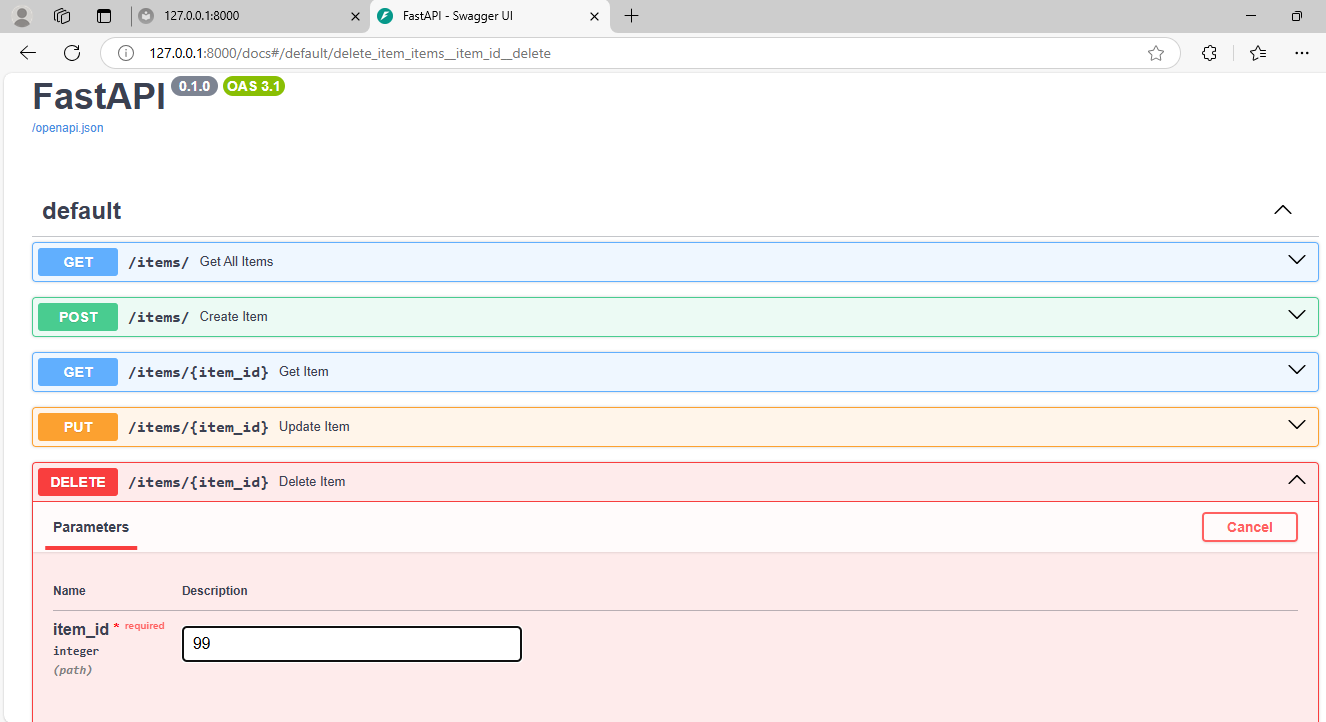
### Try any of these with item\_id = 99 (which doesn’t exist):

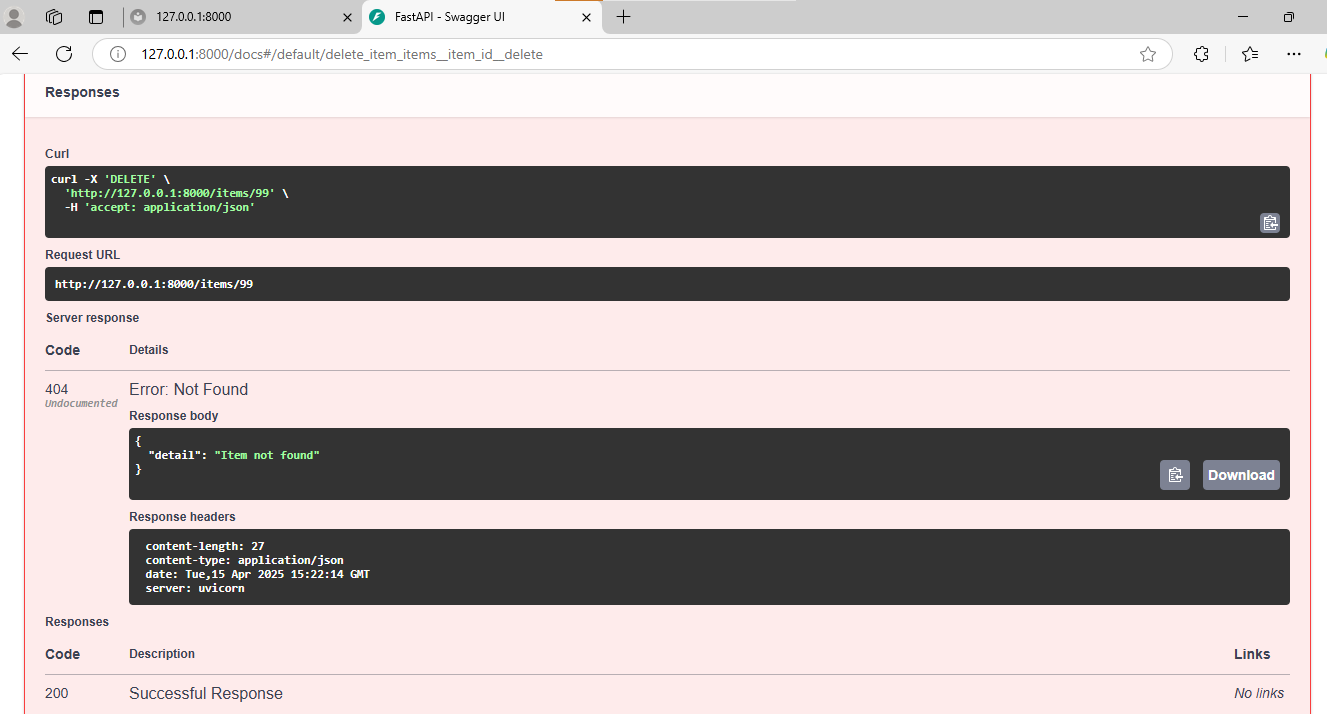












| **Test** | **Triggered?** | **Correct Response?** |
| --- | --- | --- |
| Duplicate ID | ✅ | 400 Bad Request |
| Missing Field | ✅ | 422 Unprocessable Entity |
| Wrong Data Type | ✅ | 422 Type Error |
| GET/PUT/DELETE Non-existent | ✅ | 404 Not Found |