

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:

A screenshot of a code editor window titled 'main.py'. The code is written in Python and defines a FastAPI application. It includes imports for Union from typing and FastAPI from fastapi. The application is instantiated as 'app'. There are two GET endpoints: a root endpoint '/' that returns a JSON object {'Hello': 'World'}, and an endpoint '/items/{item_id}' that returns a JSON object with 'item_id' and 'q' values. The 'q' parameter is optional and defaults to None.

```
1  from typing import Union
2  from fastapi import FastAPI
3
4  app = FastAPI()
5
6  @app.get("/")
7  def read_root():
8      return {"Hello": "World"}
9
10 @app.get("/items/{item_id}")
11 def read_item(item_id: int, q: Union[str, None] = None):
12     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.

The screenshot shows the Interactive API Docs interface for a GET request to the root endpoint. The 'Parameters' tab is active, showing 'No parameters'. Below this is an 'Execute' button and a 'Clear' button. The 'Responses' tab is also visible, showing a 'Curl' command and a 'Request URL' field. The 'Request URL' field contains 'http://127.0.0.1:8000/'.

GET / Read Root

Parameters

No parameters

Execute Clear

Responses

Curl

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

Request URL

http://127.0.0.1:8000/

GET

/items/{item_id} Read Item

^

Parameters

Cancel

Name	Description
item_id * required integer (path)	<input type="text" value="12"/>
q string (string null) (query)	<input type="text" value="q"/>

Execute

Clear

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:

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

app = FastAPI()

class Item(BaseModel):
    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:

FastAPI 0.1.0 OAS 3.1
/openapi.json

default

GET / Read Root

GET /items/{item_id} Read Item

PUT /items/{item_id} Update Item

Parameters

Try it out

Name	Description
item_id * required integer (path)	item_id

Request body required
application/json

Example Value | Schema

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

Responses

Code	Description	Links
200	Successful Response Media type application/json Controls Accept header.	No links

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

PUT /items/{item_id} Update Item

Cancel

Parameters

item_id * required
integer
(path)

Request body required
application/json

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

PUT /items/{item_id} Update Item

Parameters

Cancel

Reset

Name	Description
item_id * required	
integer	
(path)	1

Request body required

application/json

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

Execute

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:

PUT

/items/{item_id} Update Item

Cancel

Reset

Name

Description

item_id * required

integer

(path)

1

Request body * required

application/json

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

Execute

Clear

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

Code

Details

200

Response body

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

Download

Screenshot of functionality (part 2, put):

PUT

/items/{item_id} Update Item

Cancel

Parameters

Name	Description
item_id <small>* required</small>	
integer	12
(path)	

Request body * required

application/json

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

ExecuteClear

Responses

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	<div><div>Response body</div><div><pre>{ "item_name": "string", "item_id": 12 }</pre></div><div>Download</div></div> <div><div>Response headers</div><div><pre>content-length: 35 content-type: application/json date: Tue, 11 Mar 2025 02:42:00 GMT server: unicorn</pre></div></div>

Responses

Code	Description	Links
200	Successful Response	No links
	<div><div>Media type</div><div>application/json</div><div>Controls Accept header</div><div>Example Value Schemas</div><div>"string"</div></div>	
422	Validation Error	No links
	<div><div>Media type</div><div>application/json</div><div>Example Value Schemas</div><div><pre>{ "detail": [{ "loc": ["string", 0], "msg": "string", "type": "string" }] }</pre></div></div>	