

Making an API

What is an API?

- An API, which stands for "Application Programming Interface," is a set of rules and protocols that allow different software applications to communicate and exchange data with each other
- Web APIs are typically built using HTTP

Building an API

- There are many tools to do this; for this course, we will use FastAPI
- "FastAPI is a modern, fast (high-performance), web framework for building APIs with Python based on standard Python type hints."
- Decision points for me:
 - Built-in pydantic integration for validation and serialization.
 - OpenAPI integration and automatic interactive documentation.

FastAPI hello world

- create a file server.py

```
#server.py
from fastapi import FastAPI

app = FastAPI()

@app.get("/")
async def root():
    return {"message": "Hello World"}
```

Running the server

- Open a terminal in the directory containing your main.py and run the command below
- Then open the URL in your browser.

```
fastapi dev server.py
```

Making a request to your server

- create another python file client.py
- Make a request to your API

```
# client.py

import requests
import json

url = 'http://127.0.0.1:8000'

### first steps
r = requests.get(f'{url}/')
print(r.status_code)
print(r.text)
```

Dev mode

- Running the server in dev mode means that (in theory) it will automatically restart when you make changes to `server.py`.
- However, I've found that this doesn't work as expected.
- Make changes to the server and run your client - you'll find that the first run won't reflect the changes.
- If you click your mouse in the terminal window with your server after making changes, it will advance the restart process.

Server and client side

- In one terminal, you are running your server.
- In a separate terminal, you are running your client.
- If you get a 500 status code (server error) on the client side, that means that your server crashed and you have to debug on the server side.

Item model

- For the next demo, we need a simple model.
- You can just stick this at the top of server.py (after the imports)

```
# server.py
from pydantic import BaseModel
class Item(BaseModel):
    name:str
    category:str
```

POST requests

- now let's make a POST items endpoint
- this doesn't actually do anything
- we are doing some printing to see what's going on
- no explicit return will return a 200 with null text.

```
@app.post('/items')  
async def create_item(item:Item):  
    print(type(item))  
    print(item)
```

POST requests

- Now make a POST request from the client:

```
request_body = {'name': 'stapler',  
                'category': 'office'}  
r = requests.post(f'{url}/items', json=request_body)  
print(r.status_code)  
print(r.text)
```

Request validation

- What type is request_body on the client side?
- What type is item on the server side?
- What happens if:
 - no body provided?
 - missing field?

Key point

- In our unit_tests, we were manually converting to pydantic objects before passing them to UserManager.
- But we never want to trust our users to pass validated input.
- FastAPI will automatically validate requests against your pydantic models
- FastAPI also handles parameter parsing -
 - extracting data from HTTP URI path, query, and body.

POST responses

- If you need to return another status code
- e.g. 409 for resource conflict

```
from fastapi.responses import JSONResponse
# ---
# e.g. conflict
return JSONResponse(status_code=409,
                    content="duplicate key")
```