0. Topics

Thursday, May 8, 2025 11:03 PM

1. **Creating APIs using FastAPI**
2. **CRUD Operations**
3. **Handling Validations and Errors**
4. **Asynchronous Programming**

1. Creating APIs using FastAPI

Saturday, May 10, 2025 12:50 PM

Route/Endpoint in FastAPI:

- A route/endpoint is a specific URL path which our application reaches out to
- Each route is associated with a function (called a ***path operation function***) that executes when that route is accessed

COMPONENT	ROLE
@app.get("/")	Decorator that defines a GET route at root URL
home()	Function that runs when the route is accessed
return	Response is automatically converted to JSON

Returning Responses in FastAPI:

FastAPI allows returning:

- **Dictionaries** (converted to JSON automatically)
- **Pydantic models** (for validation & serialization)
- **Custom response types** (e.g., HTML, plain text, streaming, etc.)

Returning Pydantic Models:

- **When we call:**

```
@app.get("/user", response_model=User)
```

- **What FastAPI does under the hood:**

```
app.add_api_route(
    path="/user",
    endpoint=get_user,
    methods=["GET"],
    response_model=User
)
```

HTTP Methods in FastAPI:

- FastAPI supports all standard HTTP methods
- Each method has a semantic purpose in RESTful API design

HTTP METHOD	PURPOSE	SYNTAX
GET	Retrieve data	@app.get()
POST	Create new data	@app.post()
PUT	Update or replace data	@app.put()
DELETE	Delete data	@app.delete()

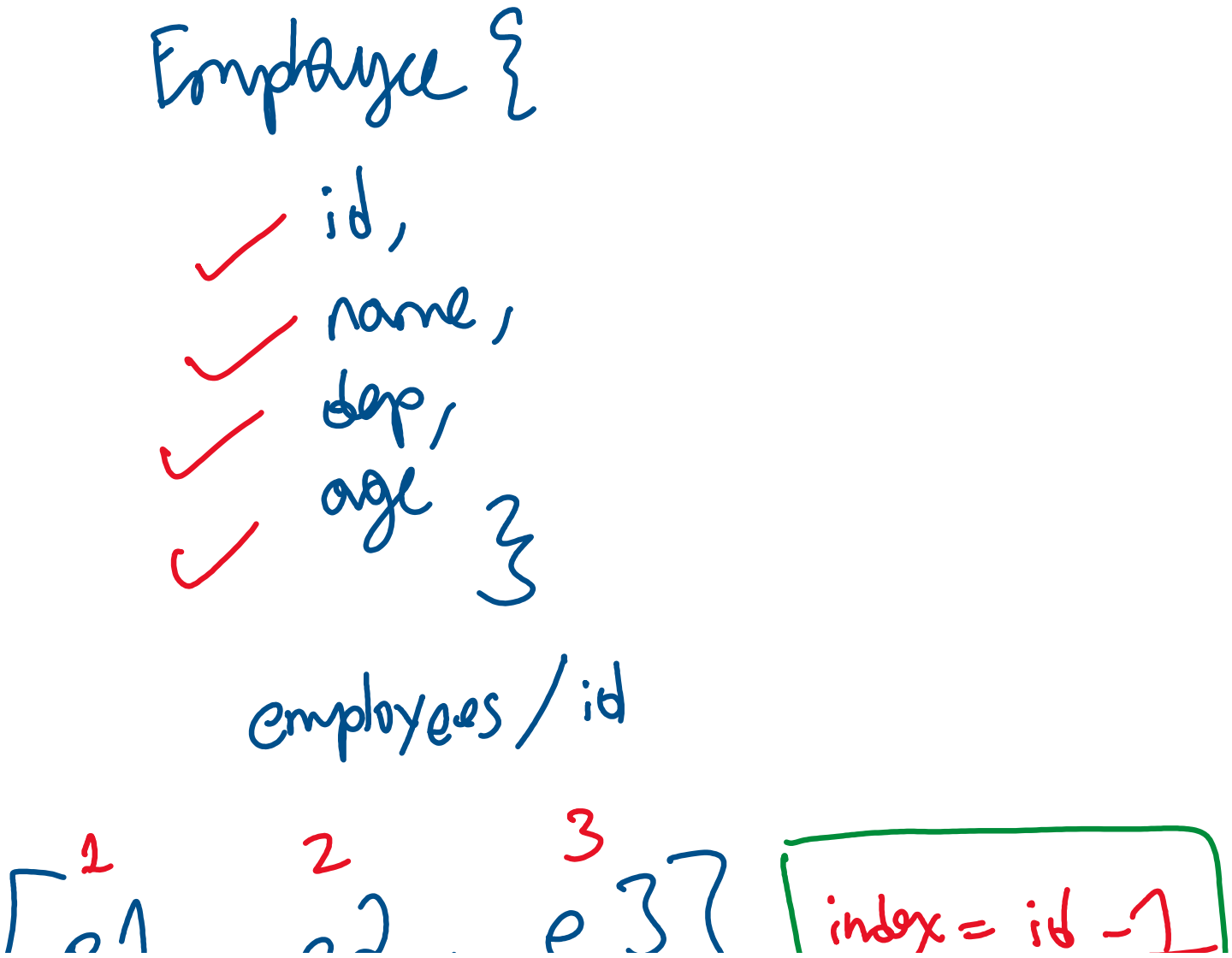
2. CRUD Operations

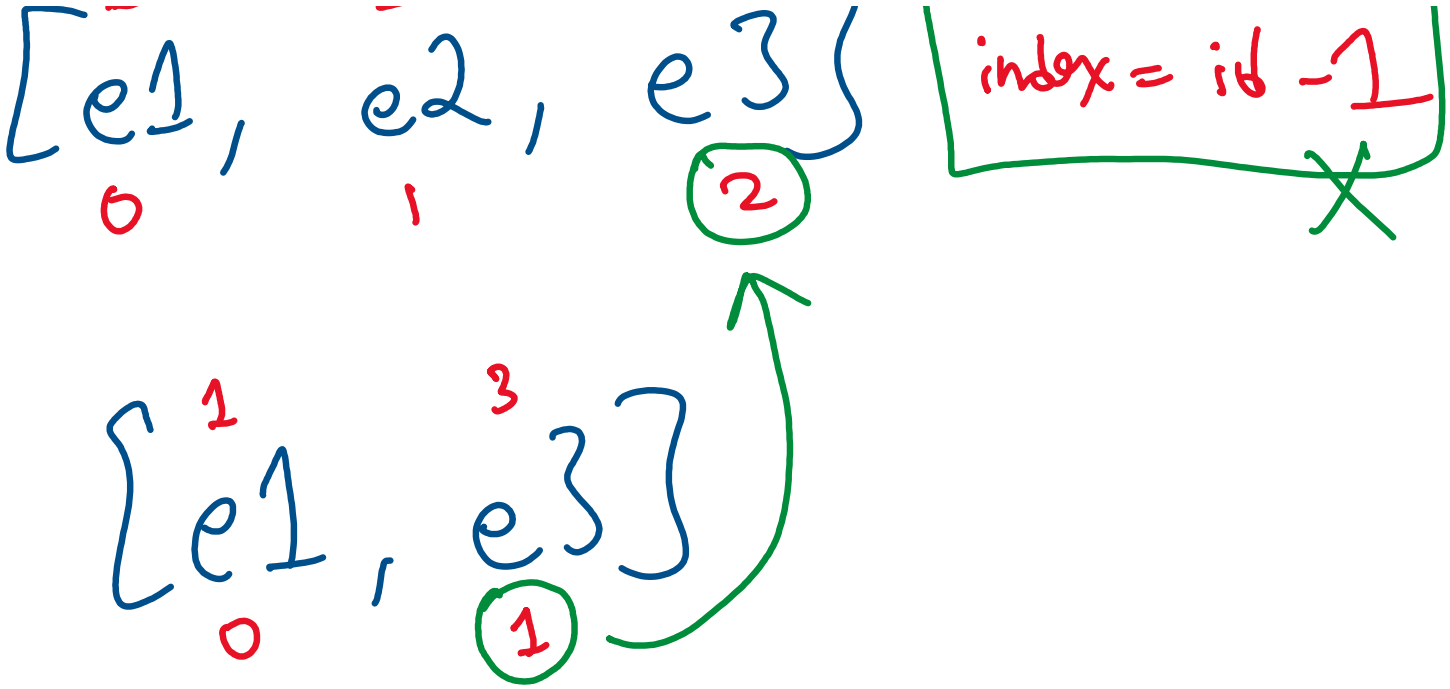
Saturday, May 10, 2025 1:04 PM

Create an app using FastAPI to implement CRUD operations on Employees database

Implement endpoints to:

- Show all employees
- Show particular employee
- Add a new employee
- Update an existing employee
- Delete an existing employee




e1
e2
e3
0
1
2
index = id - 1
1
3
e1
e3
0
1

3. Handling Validations and Errors

Monday, May 12, 2025 8:24 AM

1. Field Validation with Pydantic:

- This can be done using **Field, StrictInt, StrictFloat, etc.**
- In Pydantic, **Field** is used to provide metadata, validations, and default values for fields in a **BaseModel** instance
- Allows for more finer control over input validation and schema generation

Common Parameters of Field:

PARAMETER	DESCRIPTION
default	Default value or ... for required
title	Title for docs/schema
description	Description of the field
example	Example value
gt, ge	Greater than / Greater than or equal (numbers)
lt, le	Less than / Less than or equal (numbers)
min_length	Minimum string length
max_length	Maximum string length
regex	Regex pattern for string validation

2. Optional Fields & Default Values:

- Use **Optional** from the **typing** module

3. Custom Error Responses:

- Can be implemented using **HTTPException** from **FastAPI**
- Helps indicate the status code along with a custom message

4. Asynchronous Programming

Monday, May 12, 2025 9:16 AM

What is Asynchronous Programming?

- Asynchronous programming is a paradigm that allows your program to perform other tasks while waiting for another operation to complete (ex. database query, API call) to complete, without blocking the execution of the rest of the code
- In other words, instead of waiting for a task to finish before moving on (blocking), you can start a task, and then continue doing other things while that task finishes in the background

Synchronous vs Asynchronous:

OPERATION	SYNCHRONOUS	ASYNCHRONOUS
API Call	Waits for response before proceeding	Sends request, does other work, returns later
Database Query	Blocks until data is fetched	Queries DB, resumes when data arrives
File Read	Waits for disk I/O to complete	Reads in background while other code runs

async and await:

In Python, asynchronous code is written using asyncio module:

- **async def**: Declares an asynchronous function (called a *coroutine*)
- **await**: Tells the interpreter to - *pause here and come back when this operation is done*

Internal Working:

1. Python uses an event loop (via **asyncio**) to manage asynchronous tasks
2. Tasks are added to the loop
3. When a task hits **await**, control is yielded back to the loop
4. The loop runs other tasks in the meantime
5. When the awaited task finishes, the loop resumes it

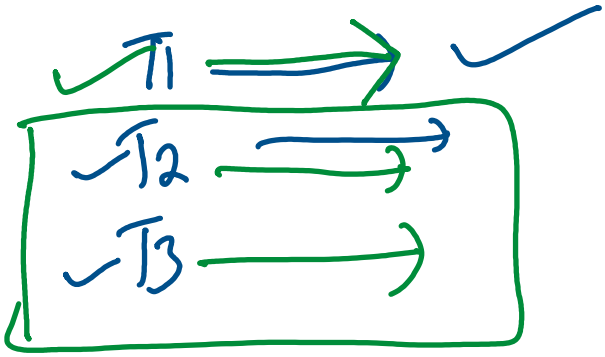
FUNCTION	BLOCKING?	DESCRIPTION
time.sleep(3)	Yes	Pauses the whole thread — blocks everything
await asyncio.sleep(3)	No	Pauses only coroutine — others can run

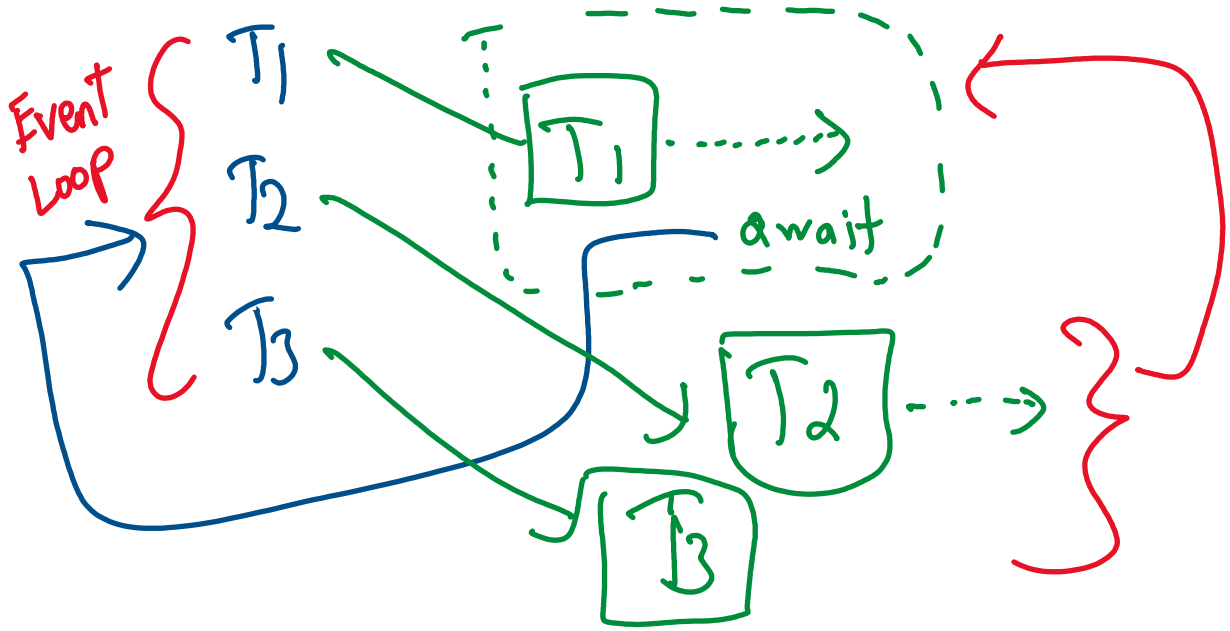
Use async def if your route does:

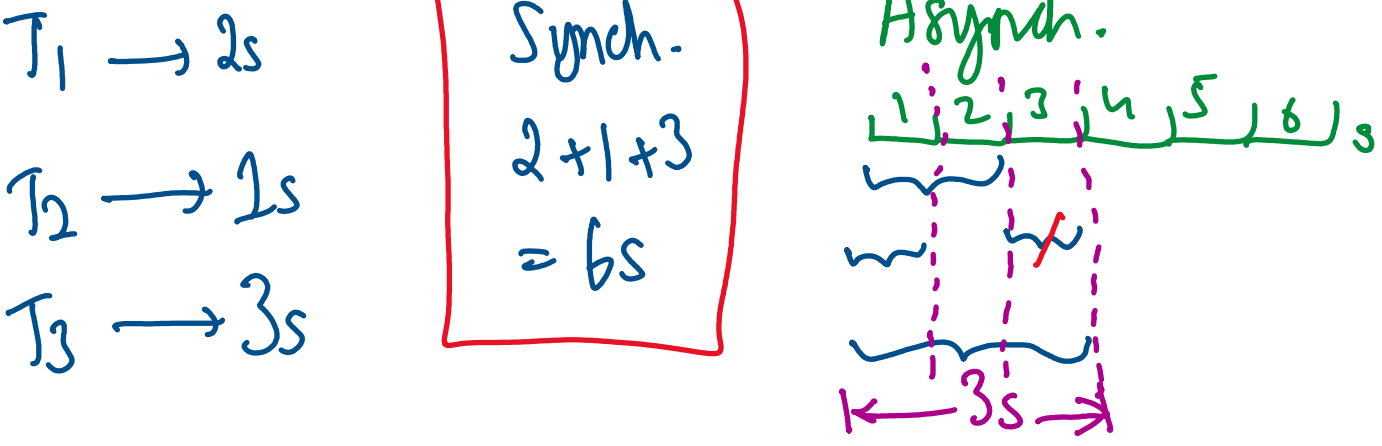
- HTTP calls (**httpx**, **aiohttp**)
- DB access with async drivers (e.g., **asyncpg**)
- I/O operations that support **async**

Use def if:

- Your function is CPU-bound (e.g., ML inference, image processing)
- You're calling a blocking library (e.g., **requests**, **psycopg2**)








3's