ICT-MAHIDOL
DEV-CLUB

Lecture 3:

JS & React/NextJS Part 2

September 20, 2024

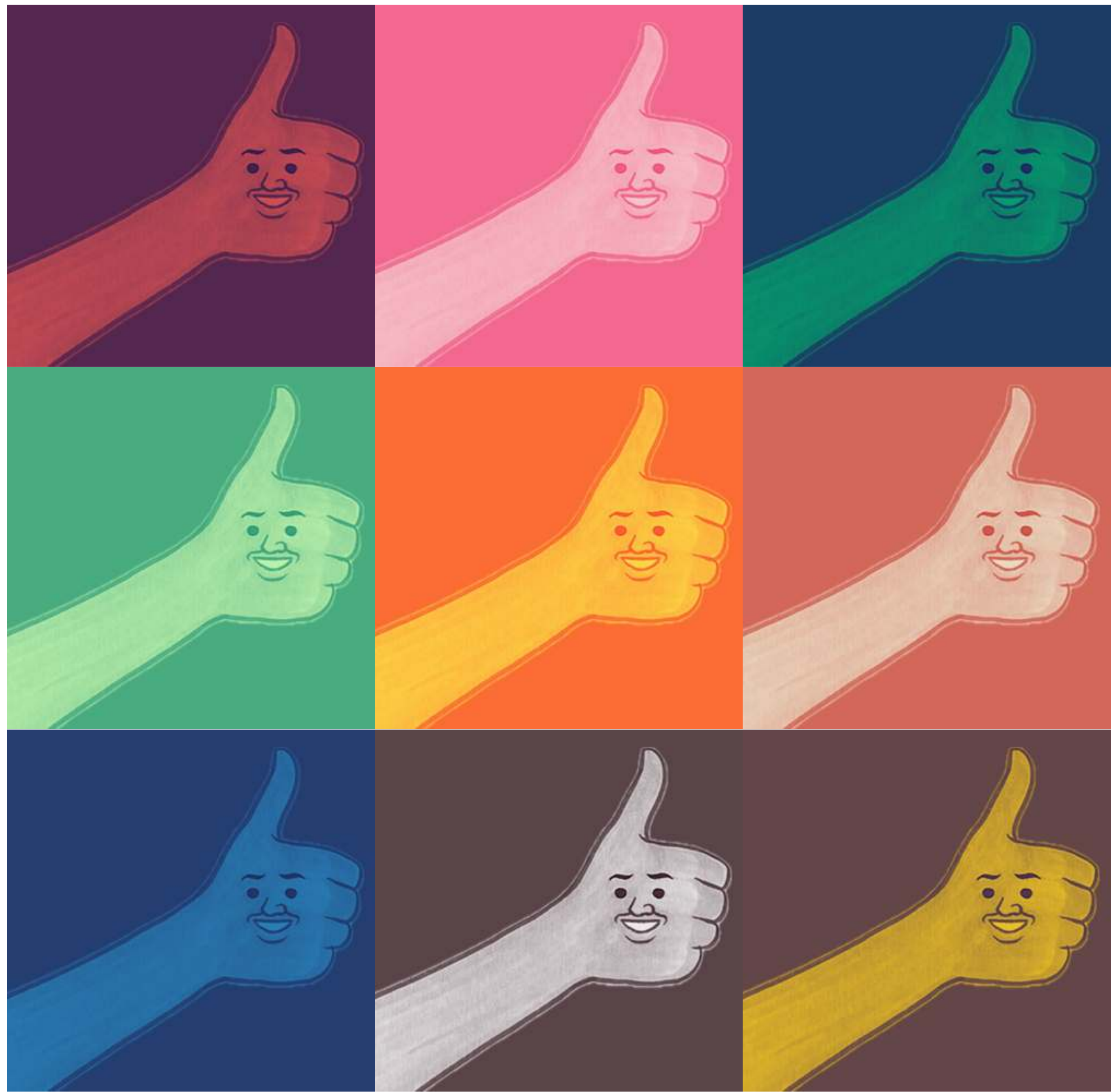
By Prachnachai Meakpaiboonwattana

Last Updated Sep 20, 2024

Contents

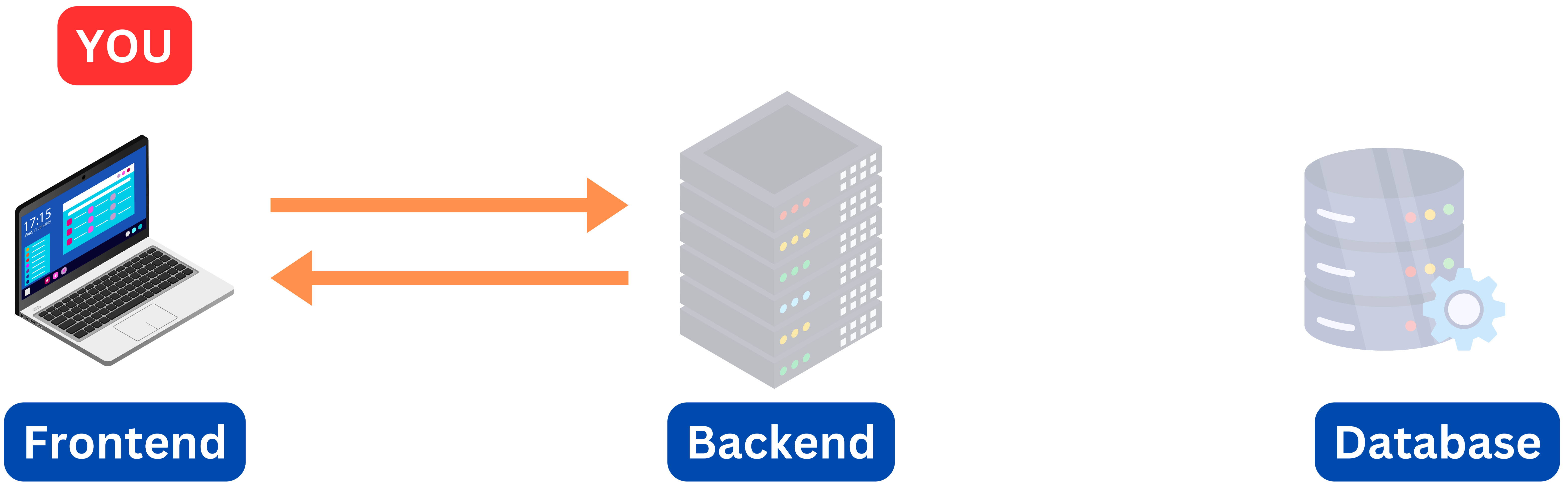
- A little more basics
  - JSON & JS Object
  - Hooks
  - Interface in typescript
- API
  - Using postman
  - Using axios

**these are considered relatively difficult (but very important still!), so you are not expected to understand them 100% in this workshop
(You'll learn them around Y2 too)



ICT-MAHIDOL DEV-CLUB

Big Picture of what we're doing today

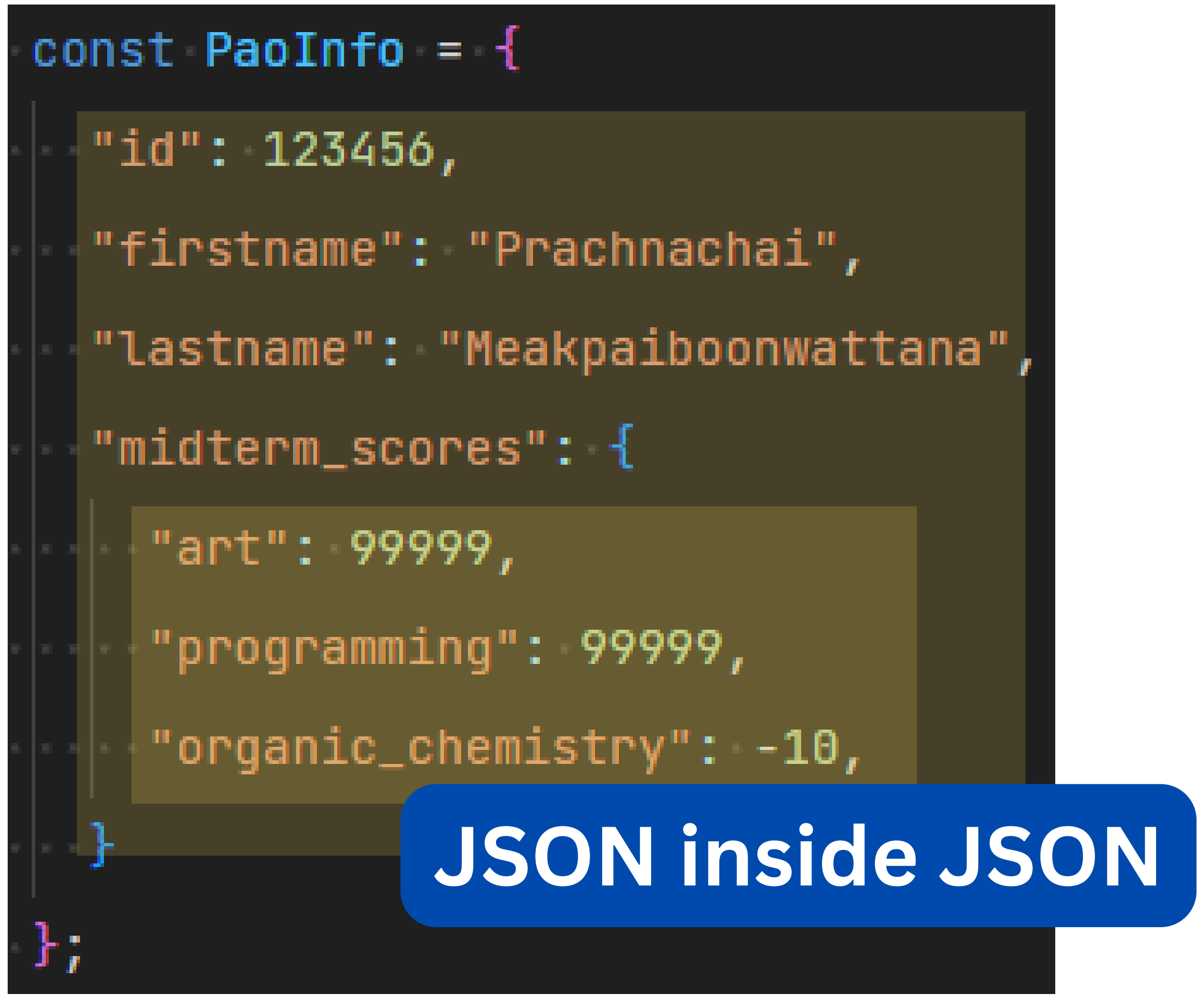


Some more basics

JSON & JS Object
State hook
Effect hook
Typescript Interface

JS Objects

- Notice: **properties are not strings**
- Common usages:
  - Configurations
  -



```
const PaoInfo = {
  "id": 123456,
  "firstname": "Prachnachai",
  "lastname": "Meakpaiboonwattana",
  "midterm_scores": {
    "art": 99999,
    "programming": 99999,
    "organic_chemistry": -10,
  }
};
```

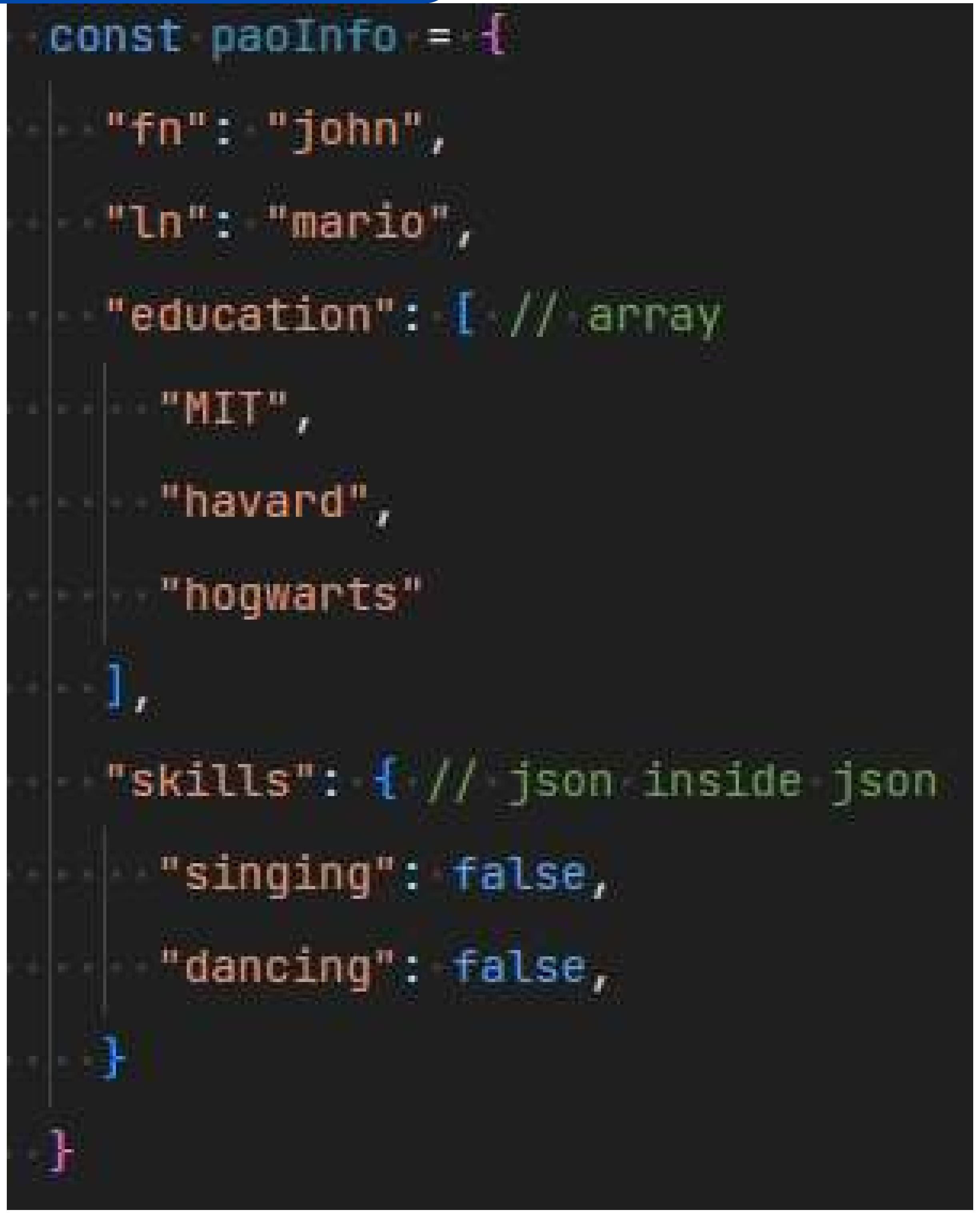
JSON inside JSON

ICT-MAHIDOL DEV-CLUB

JSON (JS Object Notations)

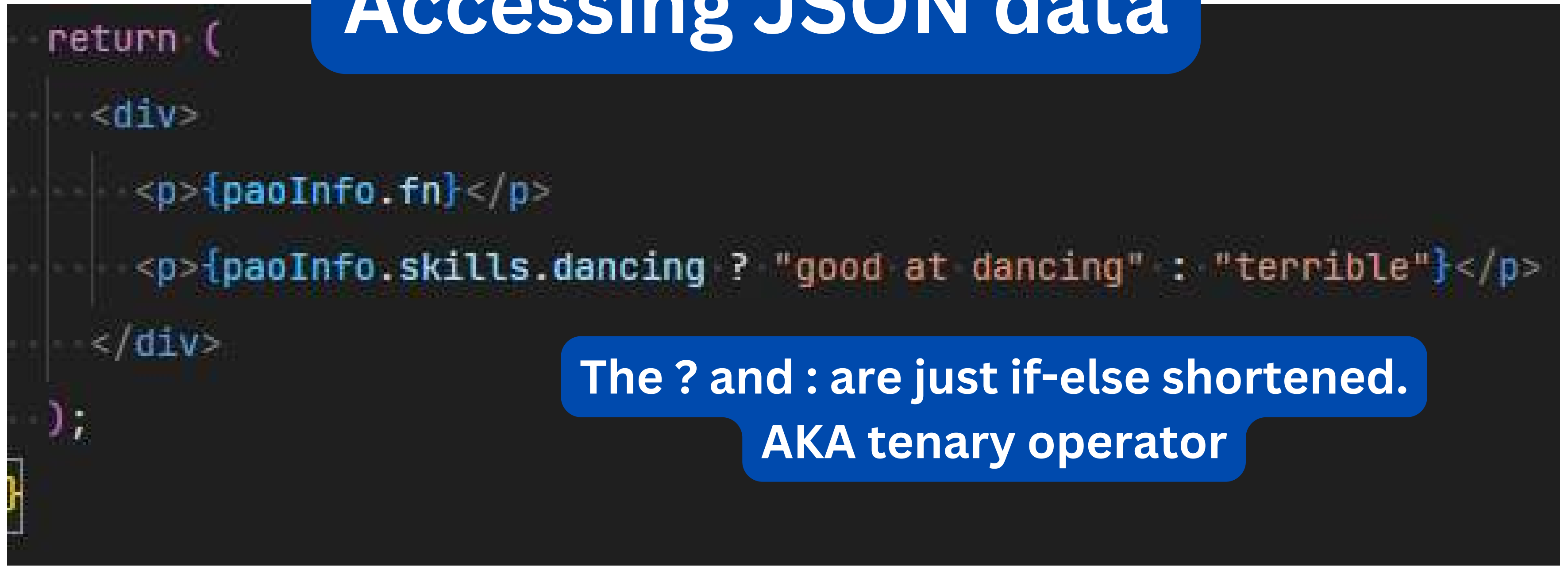
- Notice: **properties are strings**
- Common usages:
    - Transmitting data over network
    - We'll be using this a lot later today

Creating JSON



```
const paoInfo = {
    "fn": "john",
    "ln": "mario",
    "education": [ // array
        "MIT",
        "havard",
        "hogwarts"
    ],
    "skills": { // json inside json
        "singing": false,
        "dancing": false,
    }
}
```

Accessing JSON data



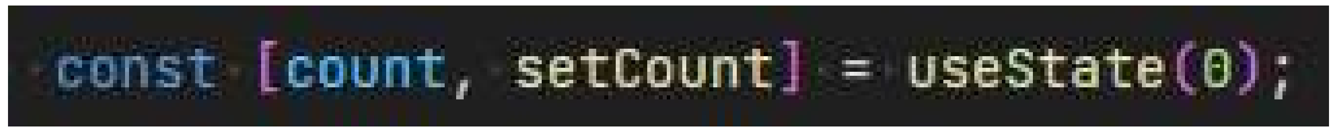
```
return (
    <div>
        <p>{paoInfo.fn}</p>
        <p>{paoInfo.skills.dancing ? "good at dancing" : "terrible"}</p>
    </div>
);
}
```

The ? and : are just if-else shortened.
AKA tenary operator

ICT-MAHIDOL DEV-CLUB

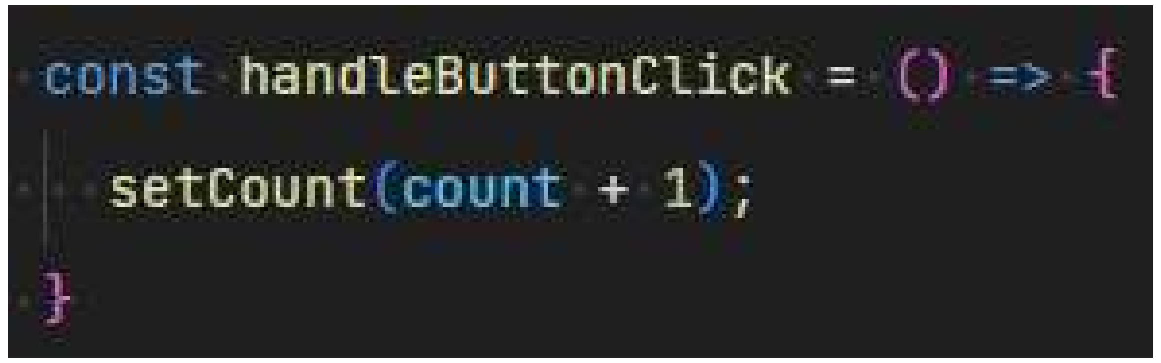
State Hooks

- Think of them like variables for UI
  - e.g. counter, display name, etc.



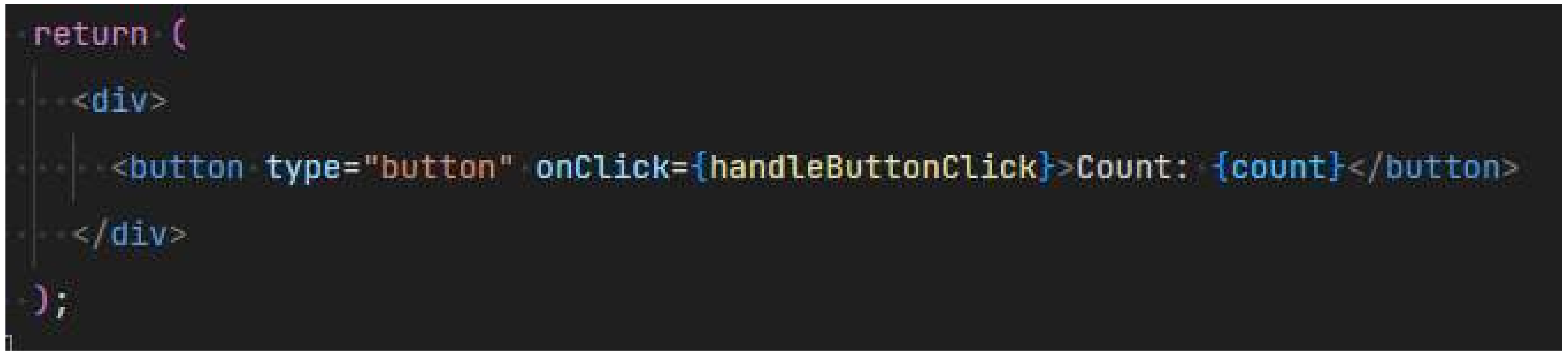
```
const [count, setCount] = useState(0);
```

Initial count: 0



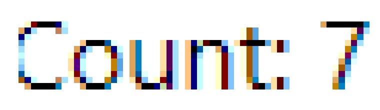
```
const handleButtonClick = () => {
    setCount(count + 1);
}
```

This is a lambda function (recall from the last part)



```
return (
    <div>
        <button type="button" onClick={handleButtonClick}>Count: {count}</button>
    </div>
);
```

+1 every time the button is clicked



Count: 7

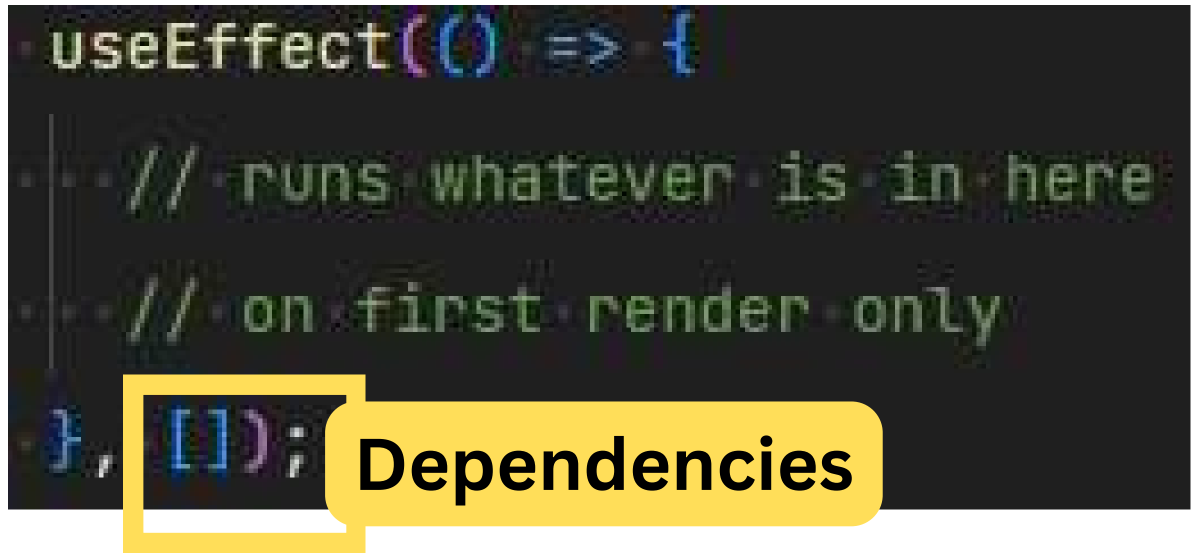
Aren’t states just fancy variables?

- You could think of them that way, but there are some differences
  - States **trigger re-rendering** of the components **whenever they change**
  - Variables don’t trigger re-rendering when changed

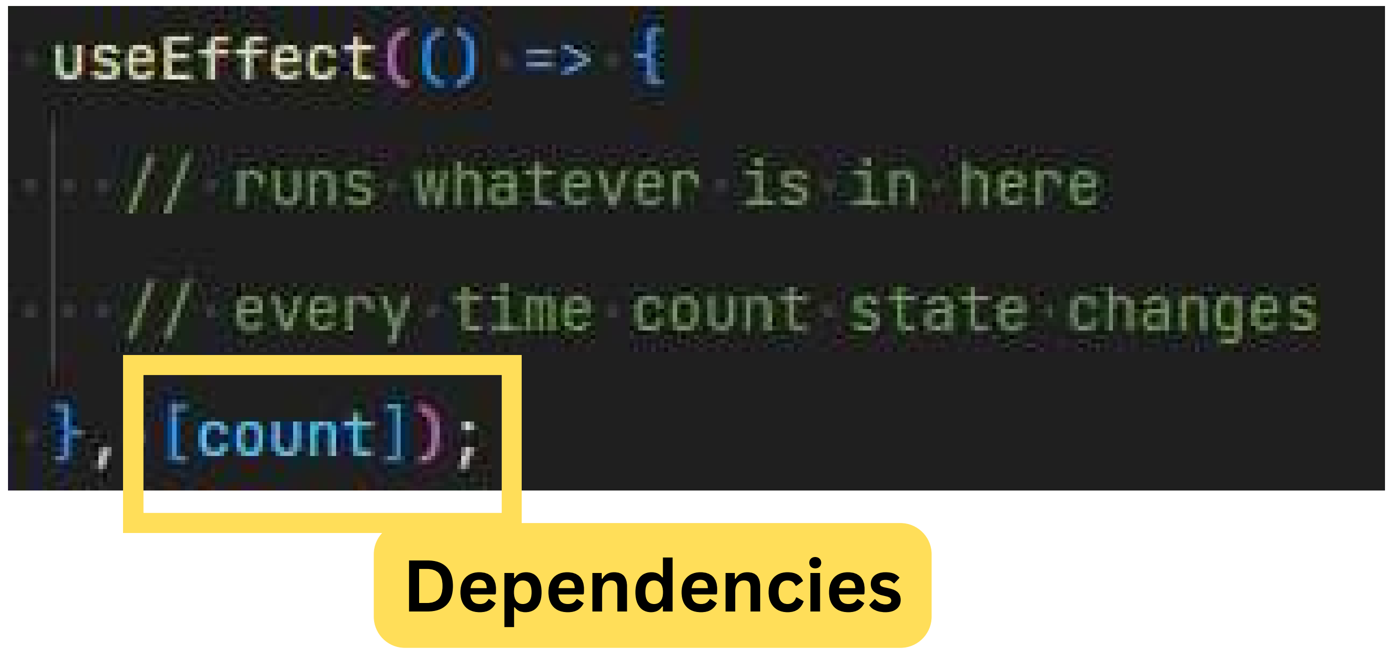
- In React/NextJS, we’d commonly use states

Effect Hook

- Introduces **side effects** to our components



```
useEffect(() => {
  // runs whatever is in here
  // on first render only
}, []);
```



```
useEffect(() => {
  // runs whatever is in here
  // every time count state changes
}, [count]);
```

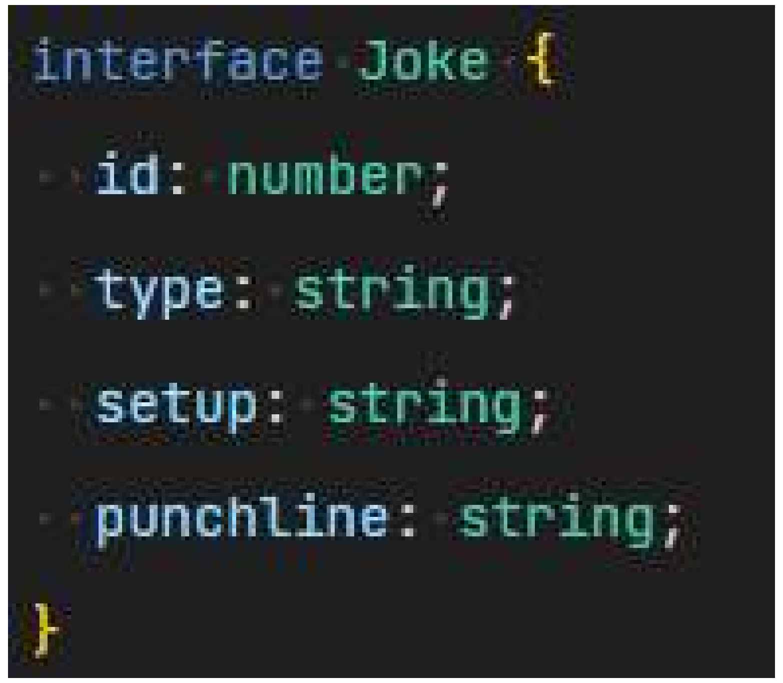
- Useful for
  - Retrieving external data via API **whenever the page is loaded**

Typescript Interfaces

- Defining our **own data type**. In this case, we're defining "Joke" datatype consisting of these properties. Similar to C's struct

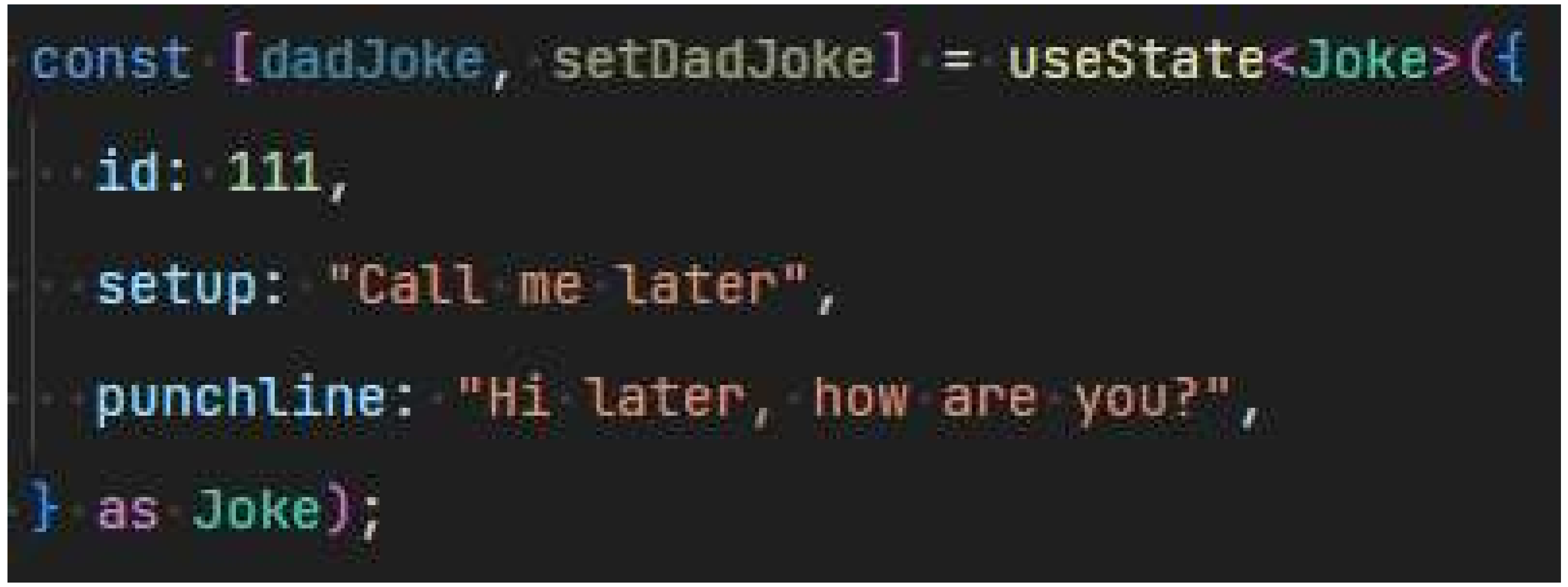
- We'll be using this later today with APIs too o_o

Setup our Joke interface



```
interface Joke {
  id: number;
  type: string;
  setup: string;
  punchline: string;
}
```

Defining a state of type Joke



```
const [dadJoke, setDadJoke] = useState<Joke>({
  id: 111,
  setup: "Call me later",
  punchline: "Hi later, how are you?",
} as Joke);
```

Recap - what we’ll be using today

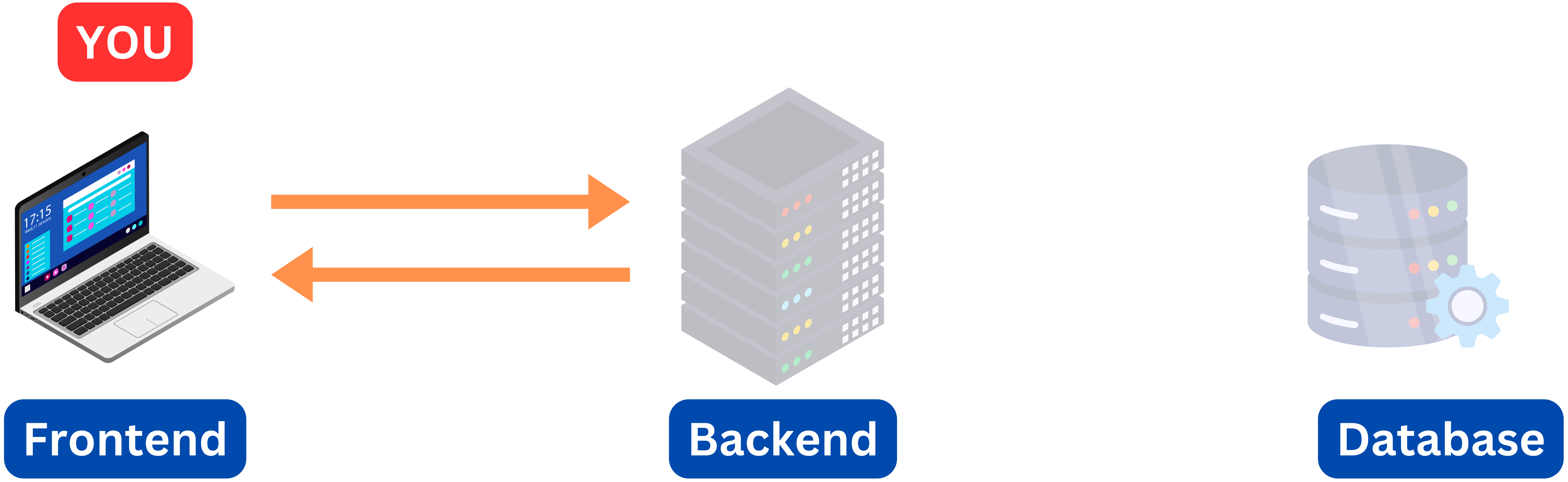
- JS Object & JSON
- State Hook
- Effect Hook
- Typescript Interface

Calling some APIs

Big Picture
Calling via Postman
Calling via Axios (code)
Async Programming Interlude

ICT-MAHIDOL DEV-CLUB

Big Picture of what we're doing today

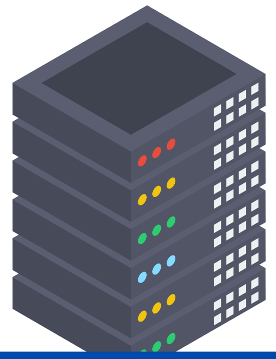


ICT-MAHIDOL DEV-CLUB

Big Picture of what we're doing today

Frontend

- Responsible for **calling** APIs and displaying the returned response



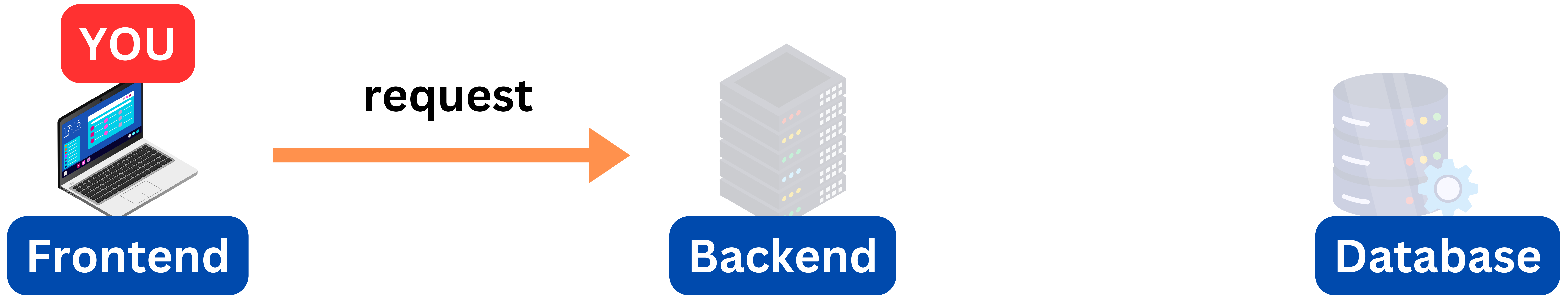
Backend

- Responsible for **creating** APIs to access data in the database
- e.g. create/update/delete

Database

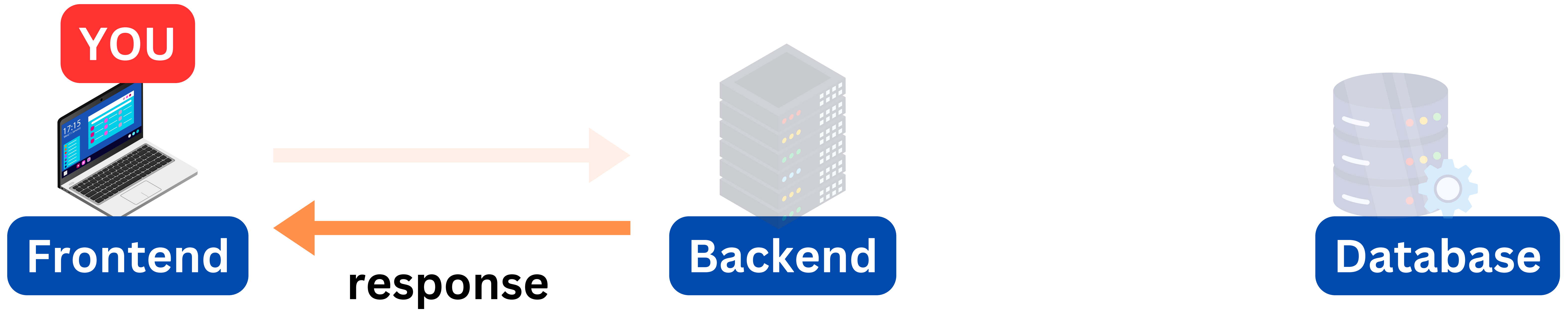
- Stores stuff lol

Big Picture of what we're doing today



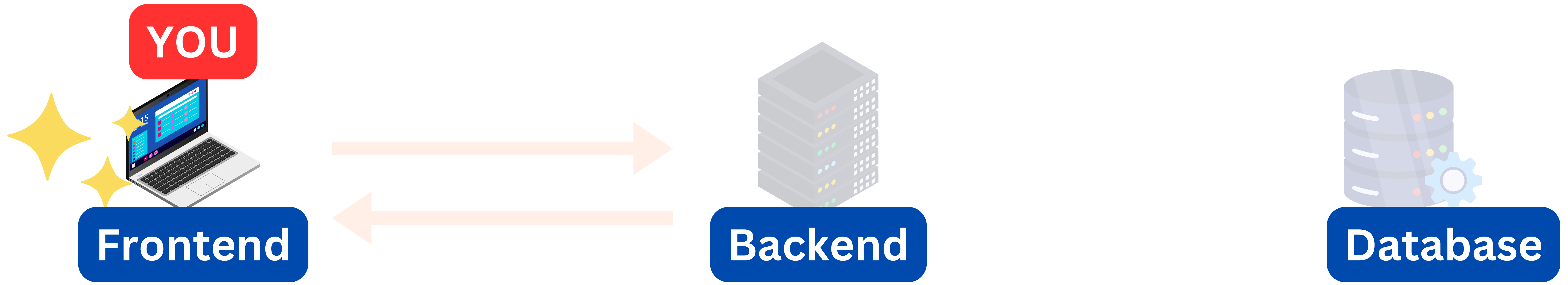
- HTTP request types:
  - GET, **reads data** from the database (We'll be mainly using this today)
  - POST, **writes new** data to the database
  - PUT, DELETE, etc.

Big Picture of what we're doing today



- Response status code common examples
  - 200 - OK
  - 404 - Not found
  - 400 - Bad request

Big Picture of what we're doing today



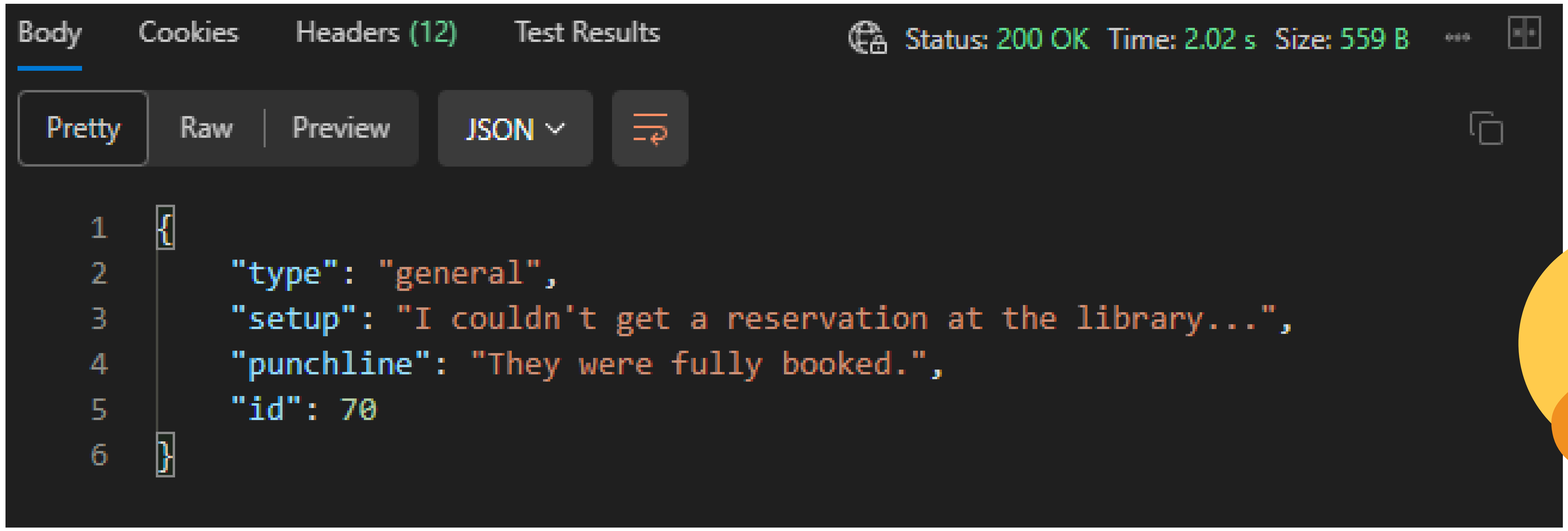
- Frontend then process the response
    - E.g. display search results to the user
    - E.g. Show errors, invalid requests, etc.

Examples using Public API

Request that we sent:

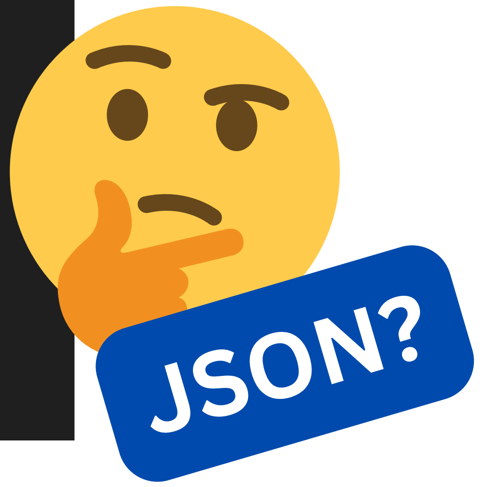
https://official-joke-api.appspot.com/random_joke

Response that we got:



Status: 200 OK Time: 2.02 s Size: 559 B

```json
{
    "type": "general",
    "setup": "I couldn't get a reservation at the library...",
    "punchline": "They were fully booked.",
    "id": 70
}
```



For simple API calls like this (**GET** mostly), you can also call them on the browser too (copy and paste into the URL bar)

ICT-MAHIDOL DEV-CLUB

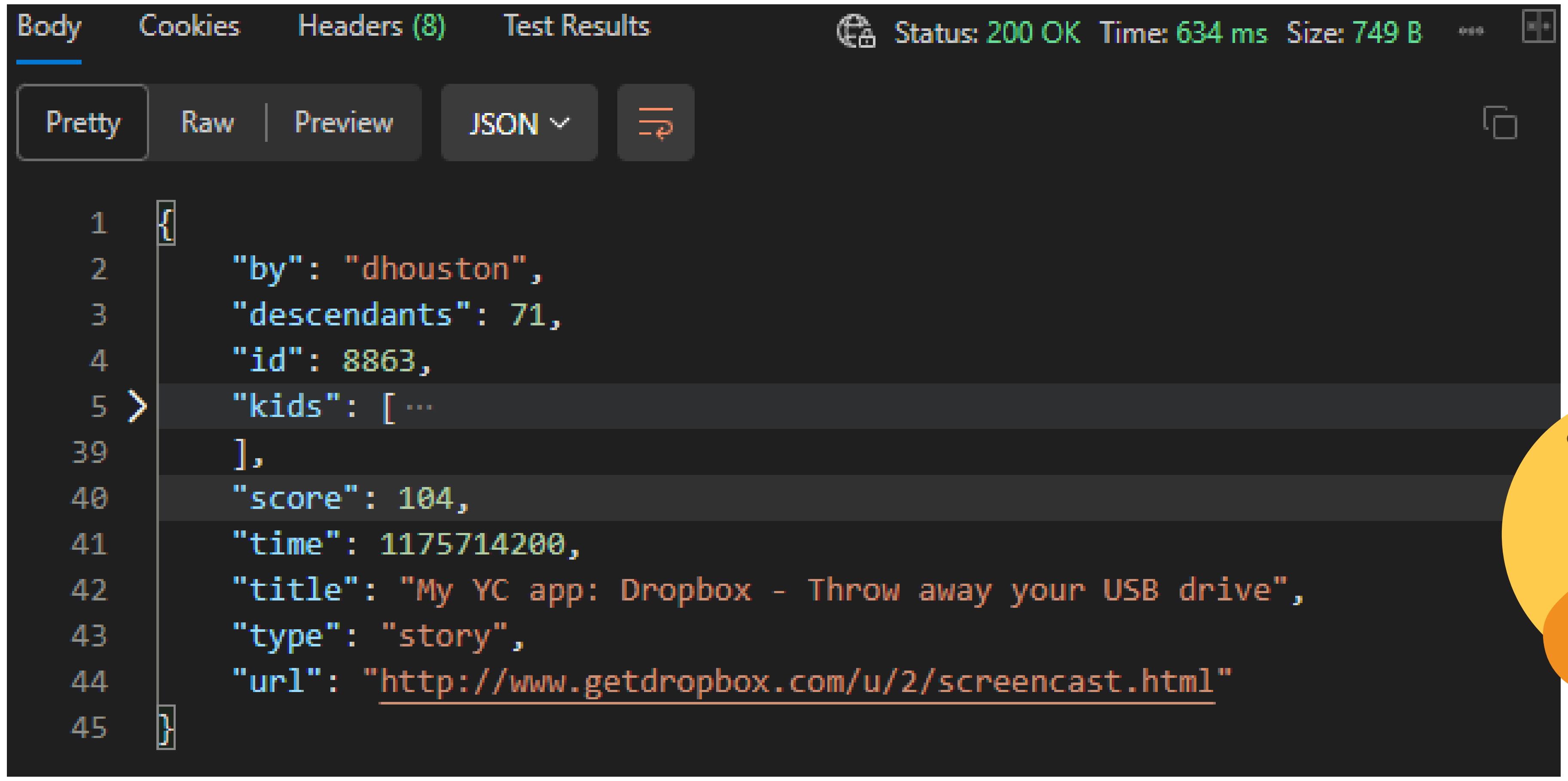
Examples using Public API

Request that we sent:

GET https://hacker-news.firebaseio.com/v0/item/**8863**.json?print=pretty

Change this ID to request for a different data

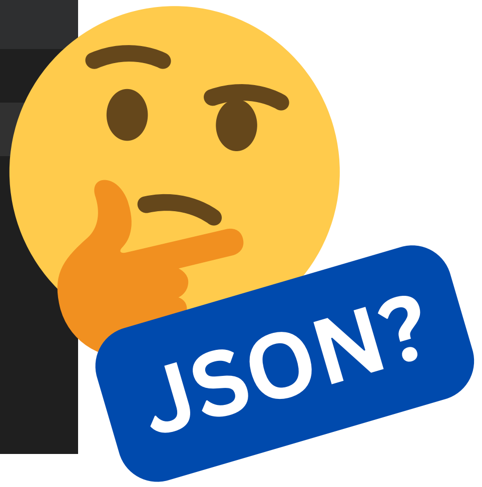
Response that we got:



Body Cookies Headers (8) Test Results Status: 200 OK Time: 634 ms Size: 749 B

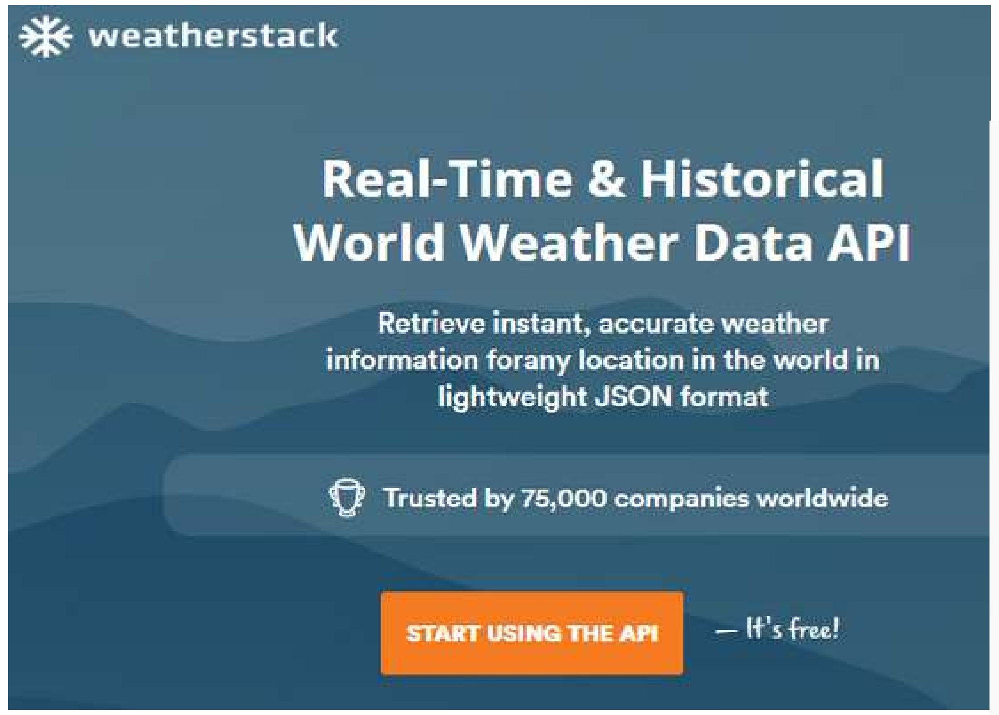
Pretty Raw Preview JSON

```
{
    "by": "dhouston",
    "descendants": 71,
    "id": 8863,
    "kids": [ ···
    ],
    "score": 104,
    "time": 1175714200,
    "title": "My YC app: Dropbox - Throw away your USB drive",
    "type": "story",
    "url": "http://www.getdropbox.com/u/2/screencast.html"
}
```



Some APIs need authentication

- This just means that you need to sign up to **get API Key**
- Often has **limits** per day/month
- You'll then need to **include your API Key** into your request
- Examples
  - https://weatherstack.com/
  - https://spoonacular.com/food-api/docs#Authentication

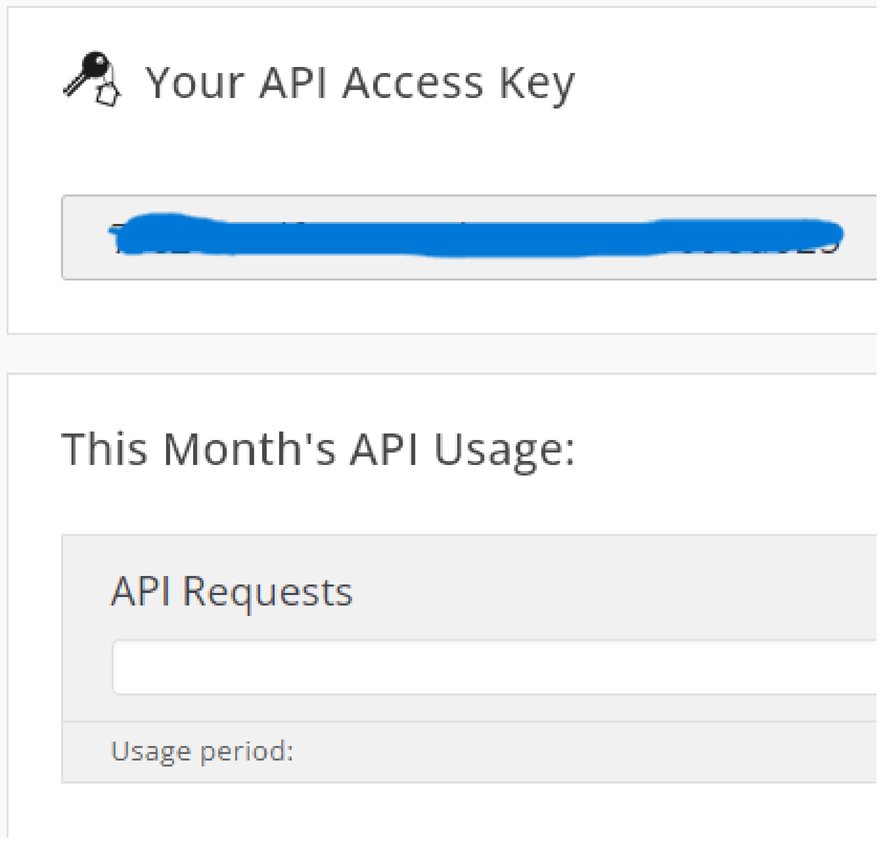


weatherstack

Real-Time & Historical World Weather Data API

Retrieve instant, accurate weather information forany location in the world in lightweight JSON format

Trusted by 75,000 companies worldwide



Your API Access Key

[redacted]

This Month's API Usage:

API Requests

Usage period:

Examples using API keys

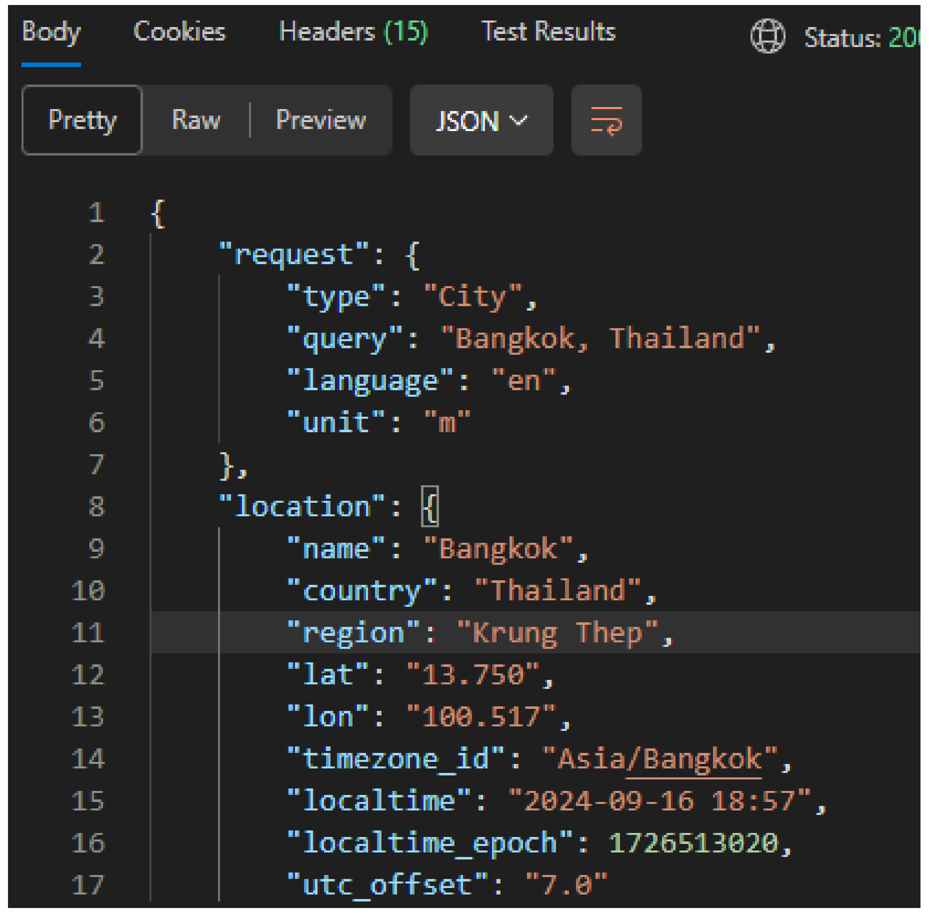
Request that we sent:

GET http://api.weatherstack.com/current?access_key=**your_api_key_here**&query=bangkok

Notice that in every requests, the 1st parameter is led with a question mark (?)

while 2nd, 3rd, 4th, etc parameters are led with an ampersand (&)

Response that we got:



```
{
    "request": {
        "type": "City",
        "query": "Bangkok, Thailand",
        "language": "en",
        "unit": "m"
    },
    "location": {
        "name": "Bangkok",
        "country": "Thailand",
        "region": "Krung Thep",
        "lat": "13.750",
        "lon": "100.517",
        "timezone_id": "Asia/Bangkok",
        "localtime": "2024-09-16 18:57",
        "localtime_epoch": 1726513020,
        "utc_offset": "7.0"
```

ICT-MAHIDOL
DEV-CLUB

Let’s convert these API calls into code

Asynchronous Programming

Synchronous VS Asynchronous
Asynchronous Functions
Promise
Async-await

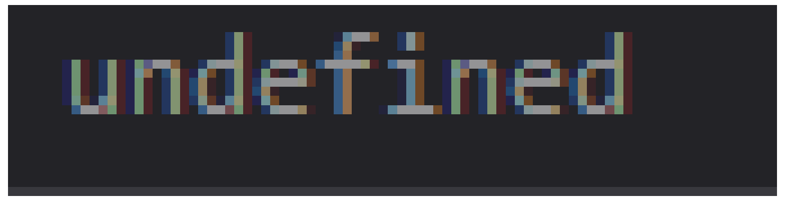
Synchronous VS Asynchronous

- Demo using event loop (simplified version)

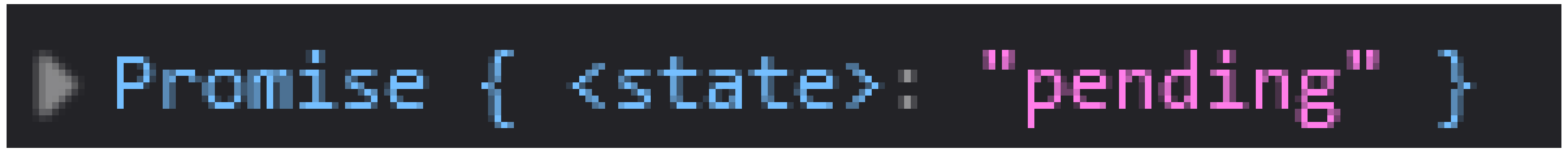
- HIGHLY recommended to watch this, a must-watch for event loop explanations
  - **https://www.youtube.com/watch?v=8aGhZQkoFbQ**

Promise

- Asynchronous functions (such as our **axios API calls**) return these when its **execution hasn't finished**
- 3 States
  - **Pending, Fulfilled, Rejected**
- Hence why we got either "pending" or "undefined"



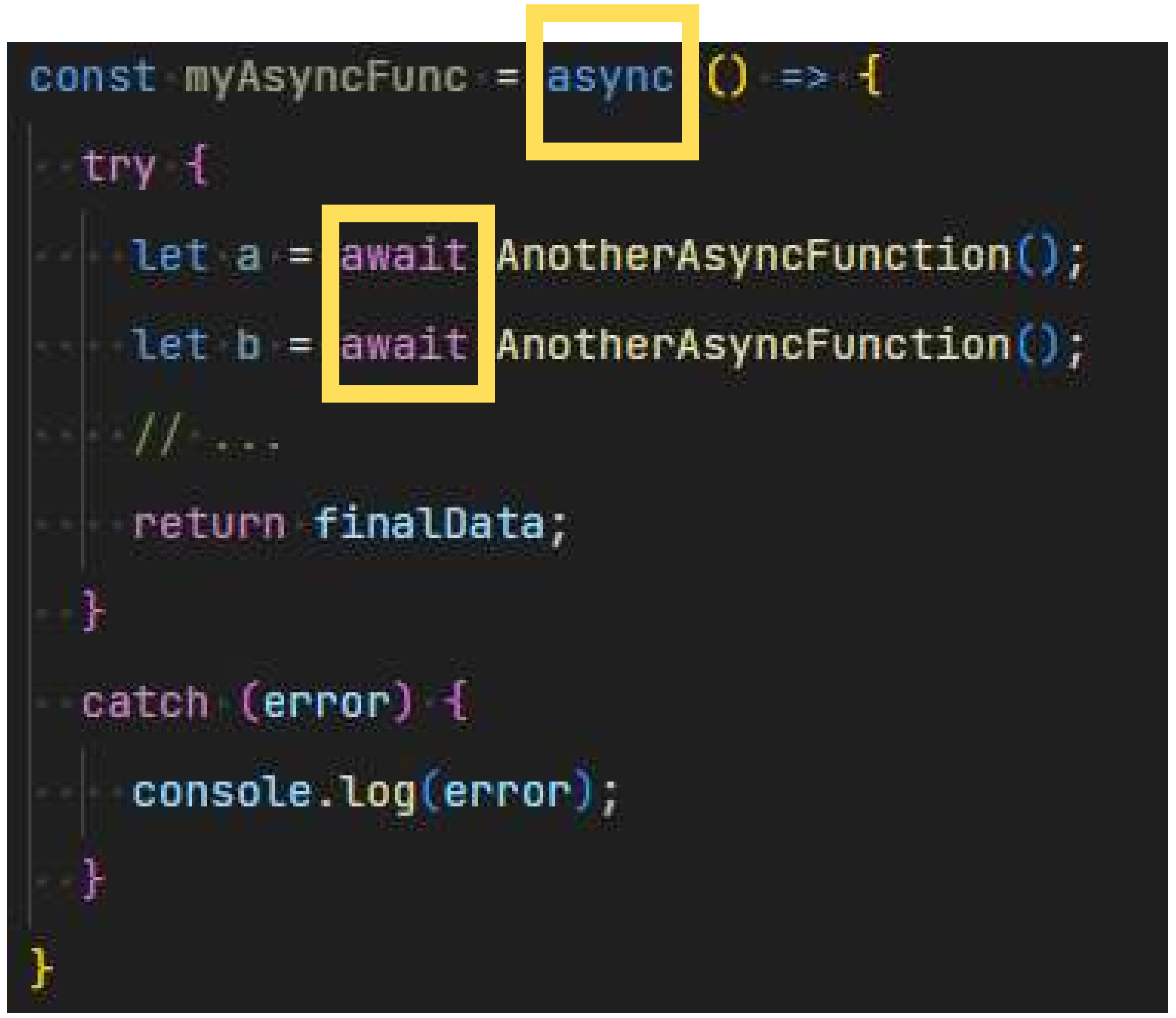
```
undefined
```



```
▶ Promise { <state>: "pending" }
```

Async-await

- Declaring our function as **asynchronous**
  - This will execute in the background when called, rather than just returning “undefined”



```
const myAsyncFunc = async () => {
  try {
    let a = await AnotherAsyncFunction();
    let b = await AnotherAsyncFunction();
    // ...
    return finalData;
  }
  catch (error) {
    console.log(error);
  }
}
```

await tells the runtime to wait until AnotherAsyncFunction finishes execution

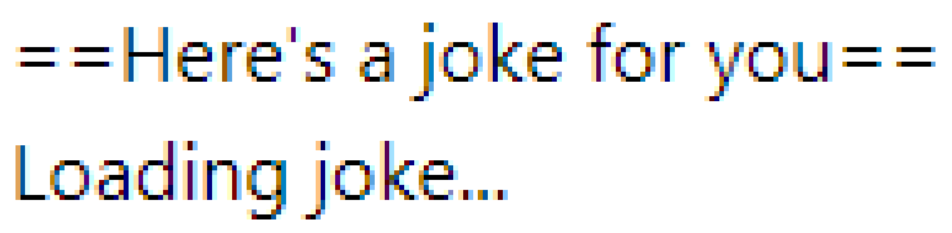
Without await, we’d get “undefined” because

- The async function hasn’t returned anything yet
- Promise is still “pending”

In-class Assignments

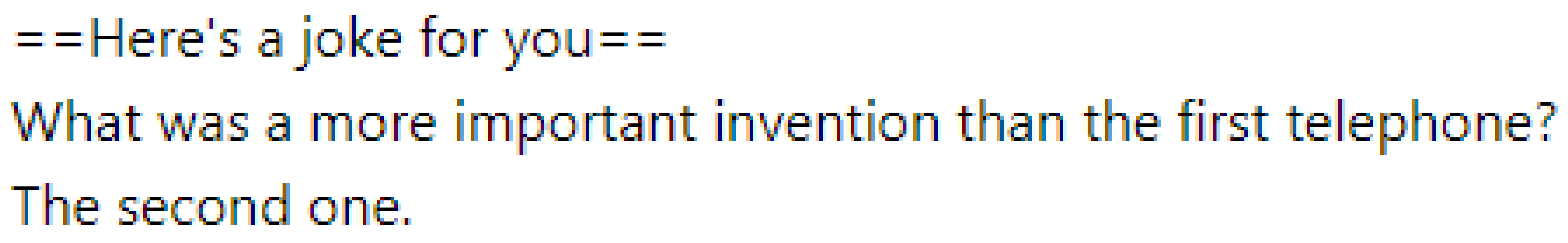
1. Call the Joke API in your NextJS app
   a. Basically follow through during the class

Pending



```
==Here's a joke for you==
Loading joke...
```

Fulfilled



```
==Here's a joke for you==
What was a more important invention than the first telephone?
The second one.
```