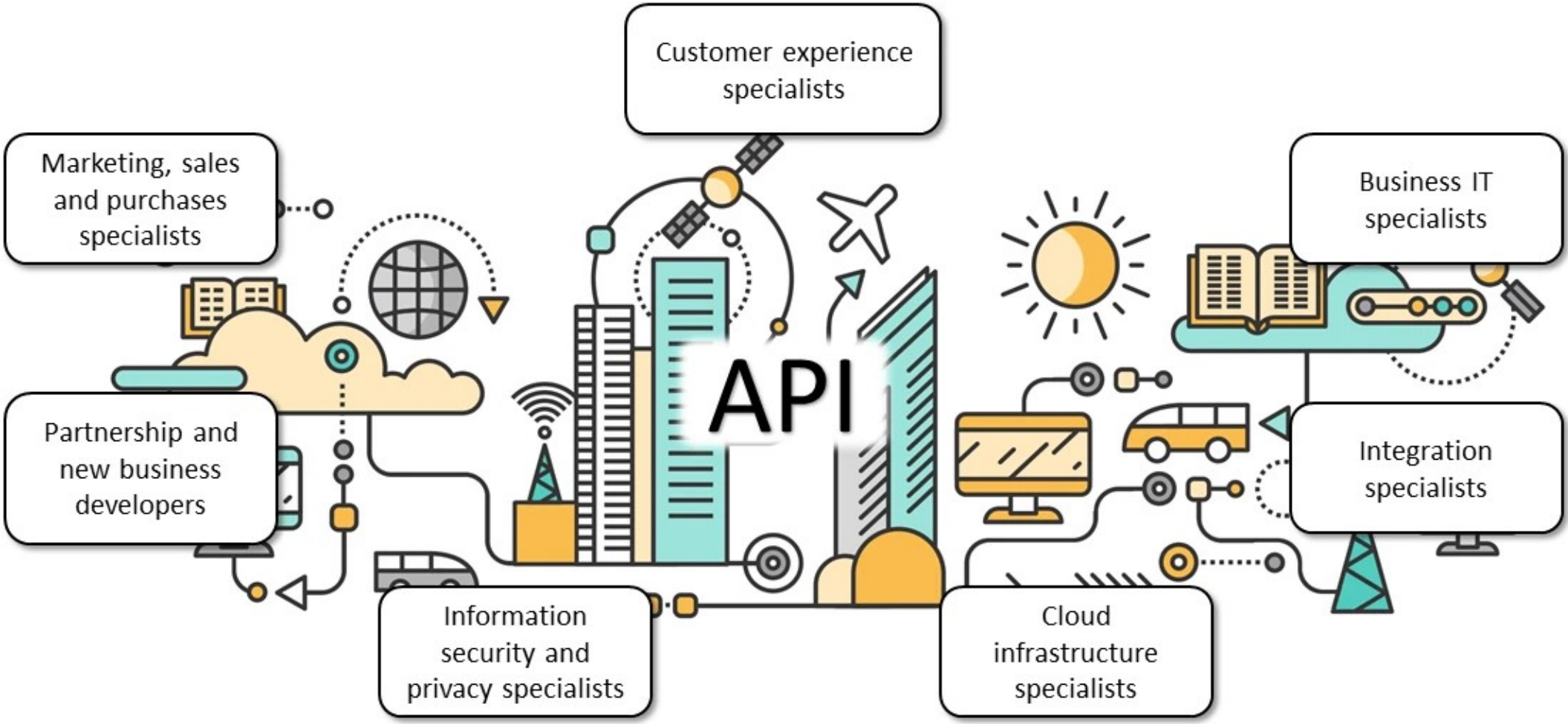
CS385 Lecture 9 APIs

APIs (Application Programming Interfaces) are used EVERYWHERE



So far, in CS385, we've imported data into our applications using static Javascript arrays, declared in a JS file.

```
export const wasteList =
[

{rID:1133, name: "Paper: office paper", ryc:1},
{rID:893, name: "Paper: magazines", ryc:1},
{rID:3, name: "Paper: newspapers", ryc:1},
{rID:1, name: "Paper: junk mail", ryc:1},
{rID:71309, name: "Cardboard", ryc:1},
{rID:71390, name: "Glass: Green, clear and brown glas
{rID:7139, name: "Glass: Green, clear and brown glass
{rID:7130, name: "Juice and milk cartons", ryc:1},
{rID:2137, name: "Plastic bottles", ryc:1},
{rID:3137, name: "Plastic containers", ryc:1},
{rID:4137, name: "Tins and cans (steel)", ryc:1},
{rID:5137, name: "Tins and cans (aluminium)", ryc:1},
{rID:6137, name: "Empty aerosols", ryc:1},
{rID:1993, name: "Plastic bags", ryc:0},
{rID:12213, name: "Plastic wrapping/film", ryc:0},
```

- As we have seen in many of our previous examples, we have used Javascript arrays where the contents are JSON objects
- This has worked very well for us. We have been able to import or include real-world data in our application (For example, **`wasteList`** above).

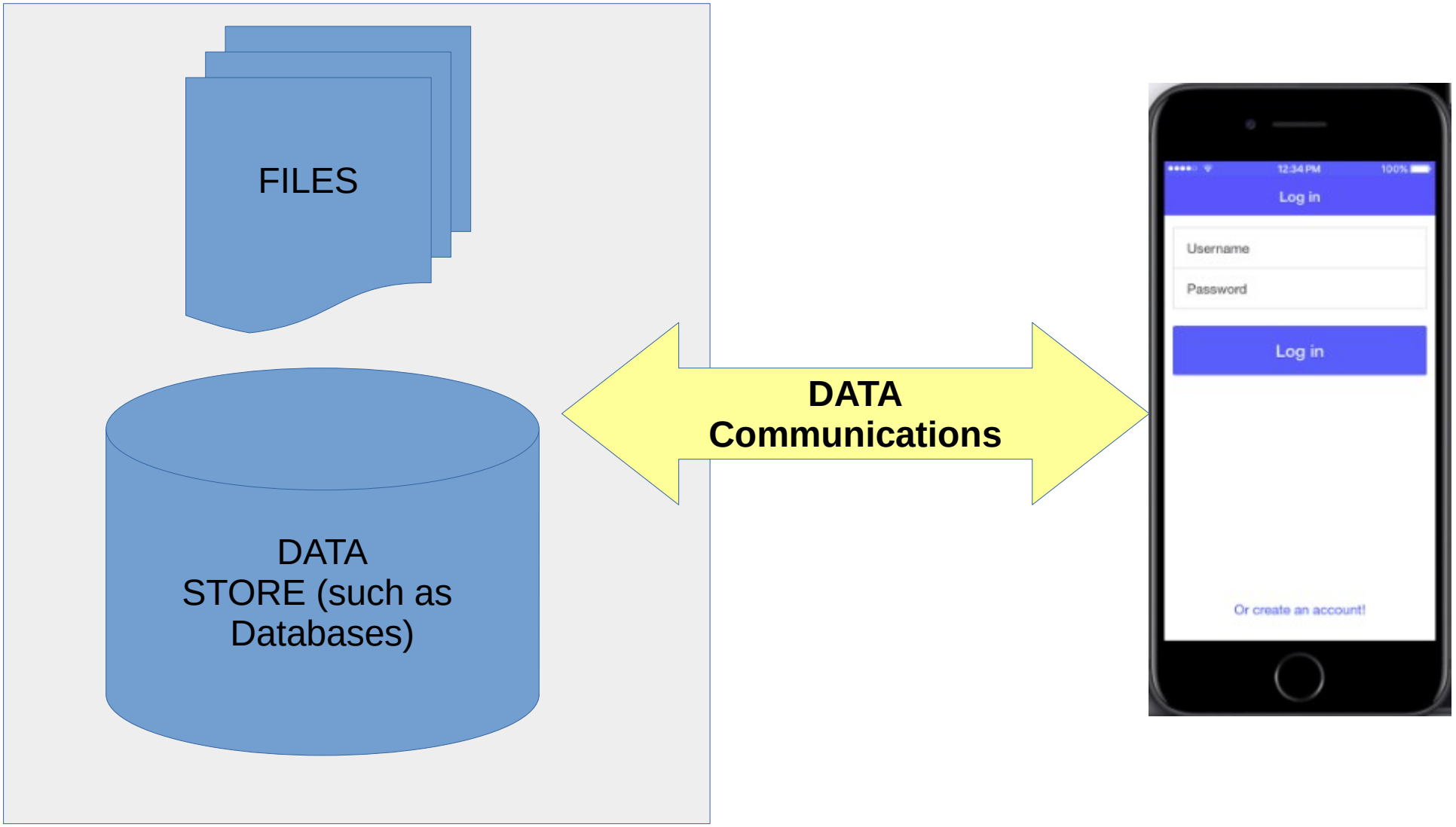
The problem with this approach is our application becomes tightly bound to the data in the JS file (and the array)

- What happens if this was a real mobile application, used by hundreds of users?
- What if we needed to make changes to the wasteList array? For example, add new items to the array or make changes to existing objects in the array.
- This would lead to the unwanted situation of needing to update everyone's application so that the Javascript array is updated.
- **This is unsatisfactory and should be avoided.**

Using our Javascript array in JS file approach would create a nightmare for us when the application is deployed .

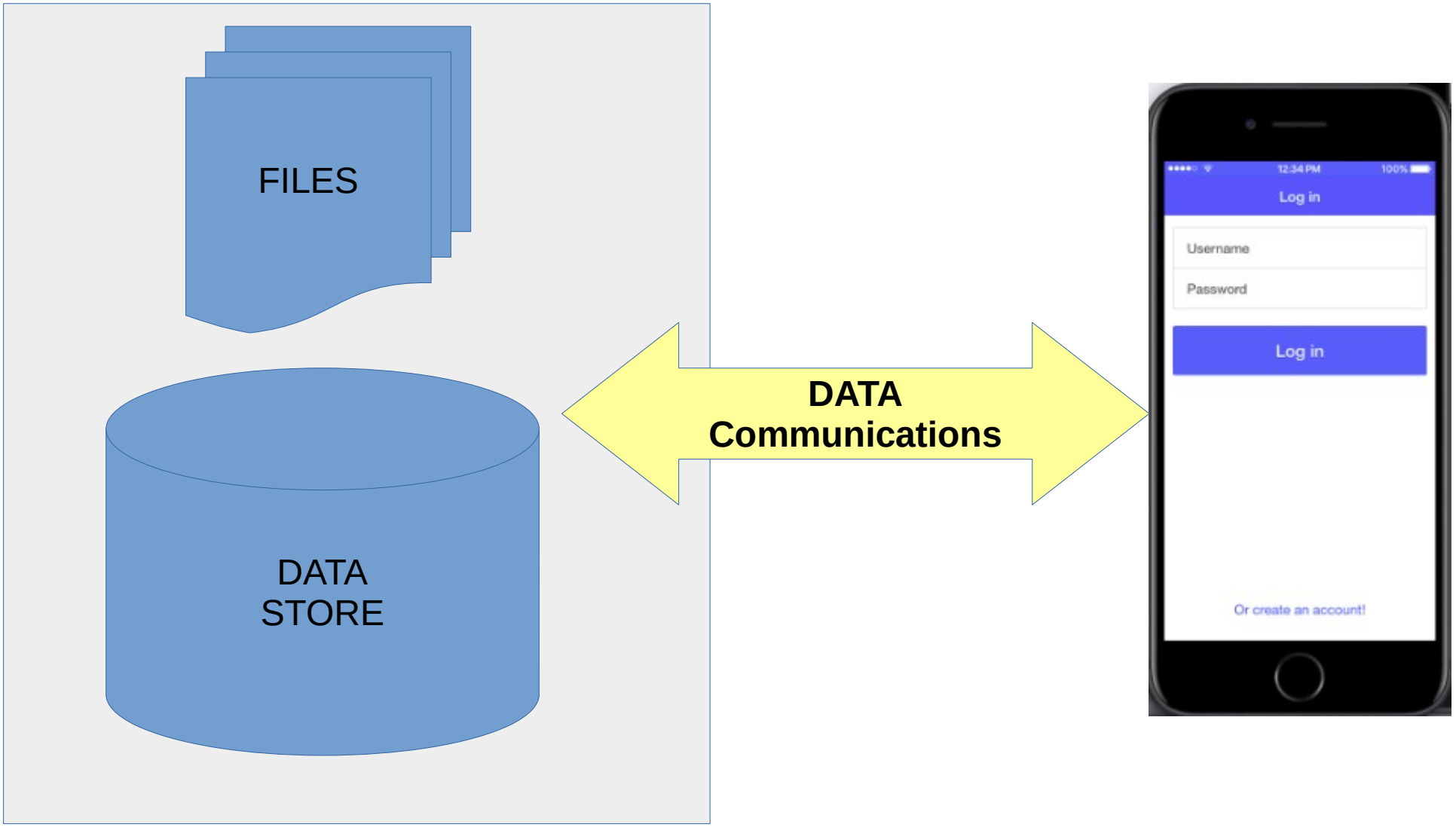
- **We need to decouple our application from the sources of data and the rendering of that data.**
- Users want to be able to easily interact with the digital world around them without having to leave the application space. **They expect the data that drives their experiences to be synchronized, regardless of the platform they choose or the application software they install and run.**

In a very simplistic way Internet-based data stores allow mobile applications to always connect to the most up-to-date data



DATA STORAGE on SERVER(s)

Our Data Storage can be on our own servers (in our organisation) or anywhere on the Internet

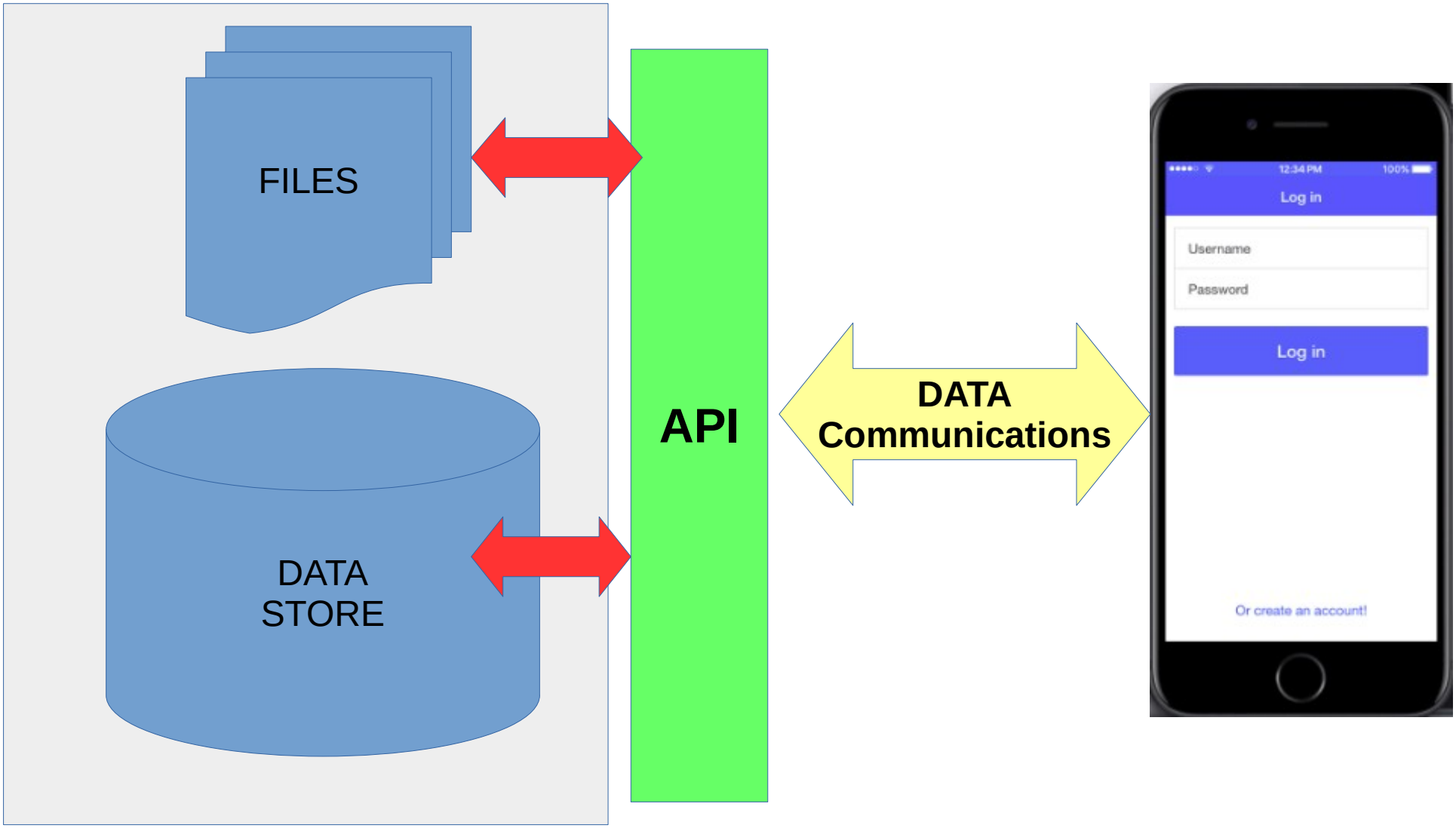


DATA STORAGE on SERVER(s)

Data Stores take care of many resource intensive activities

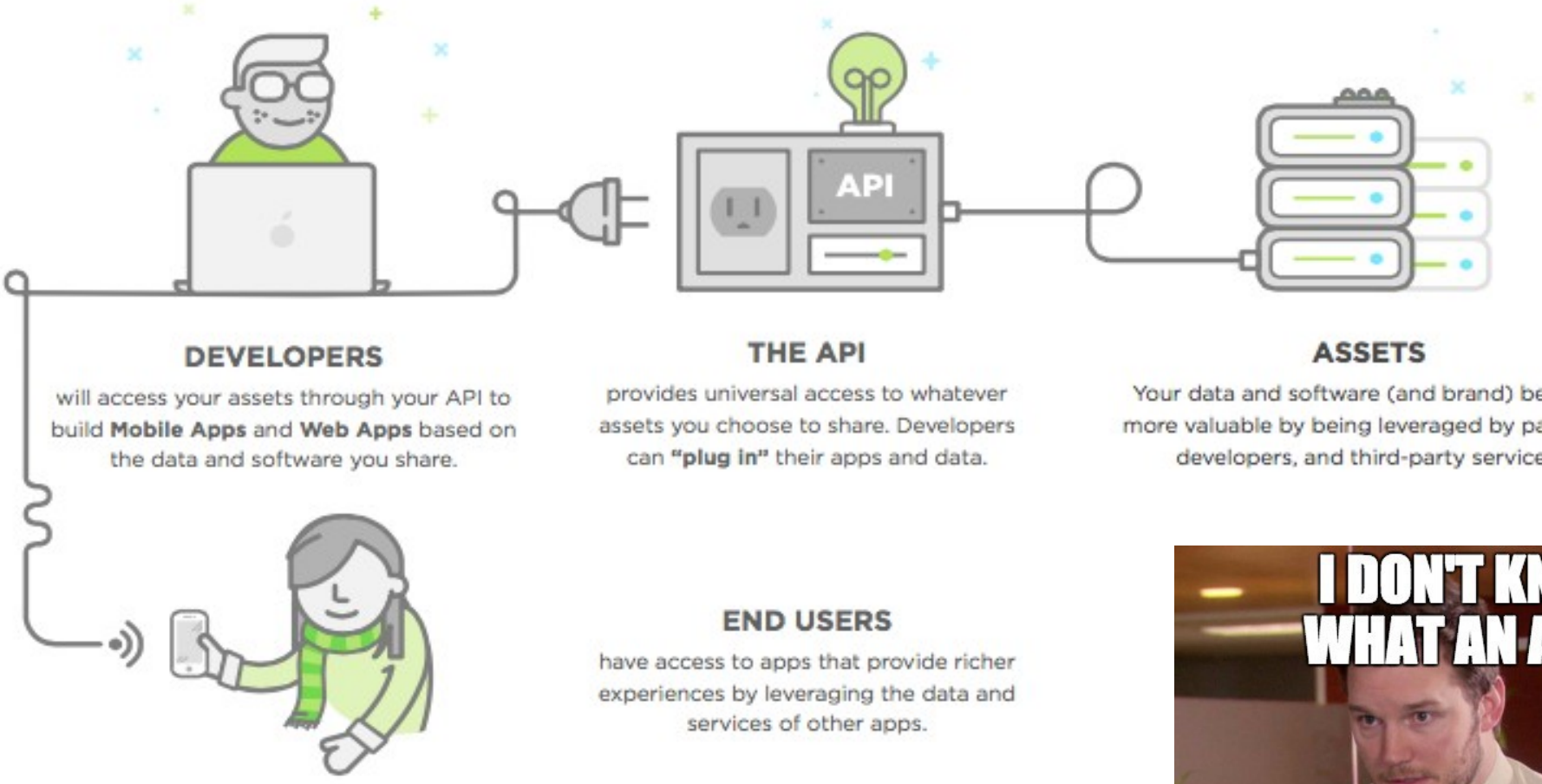
- **The need for backend connectivity stems from the constraints of the mobile device itself.**
- While portability is a great convenience, it has given rise to limits on power, CPU processing and storage within devices.
- Of course, portable technology continues to improve, **but offloading intensive processing, large data storage and battery intensive activities to backend systems is often the most cost-effective and resource-efficient plan**.

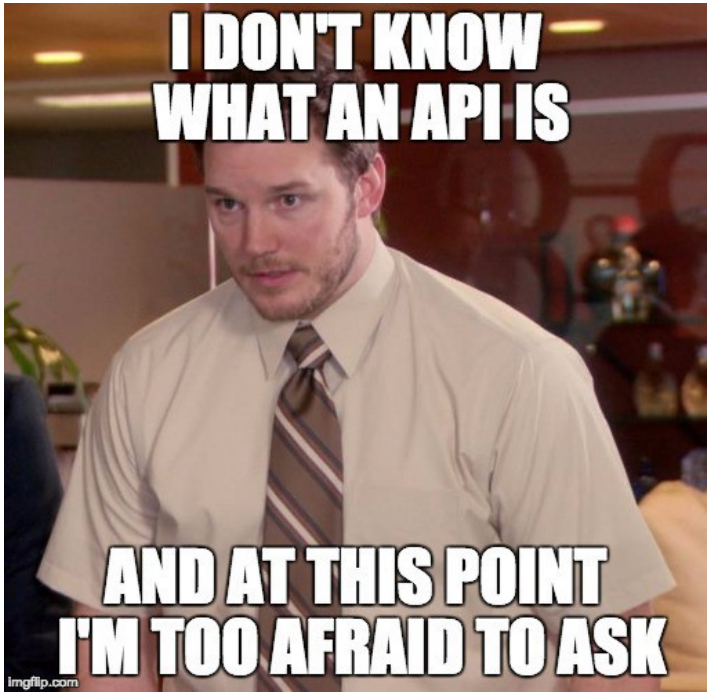
The key to accessing these data stores are Application Programming Interfaces (API)



DATA STORAGE on SERVER(s)

How APIs work

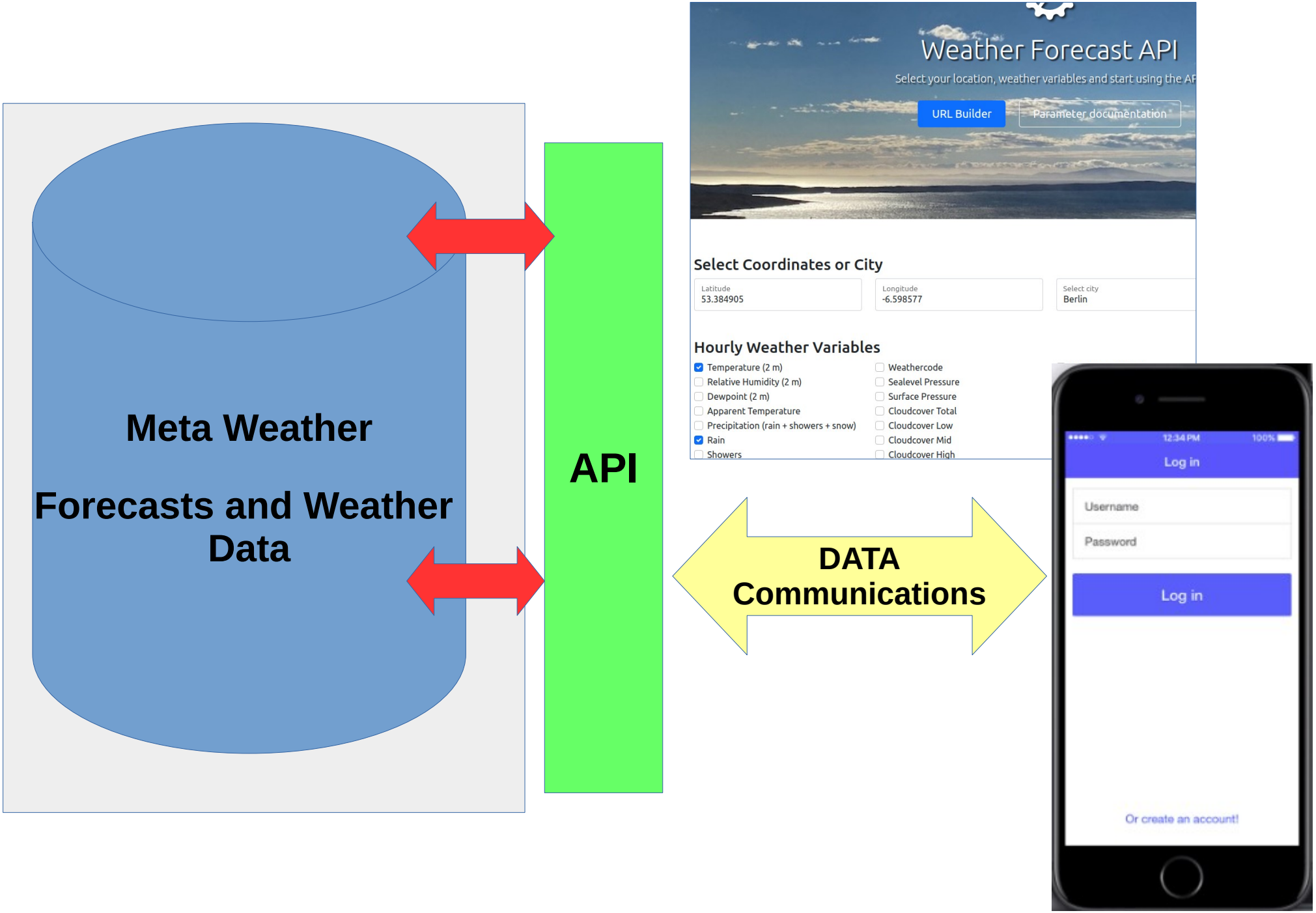




API. Let's have a look at a very simple example

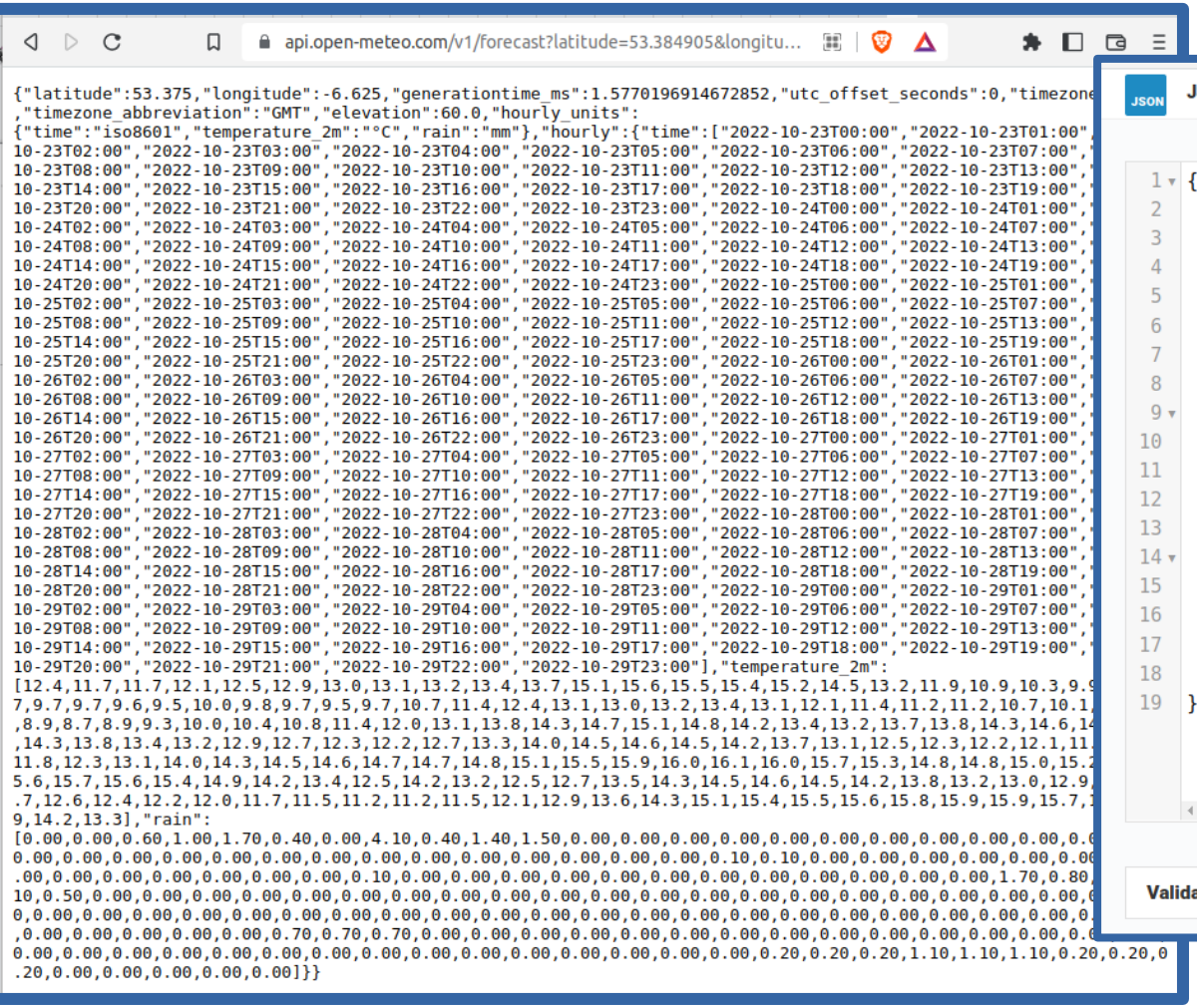
- **Suppose in our mobile application we want users to be able to access Weather Data for their location.** Suppose we allow users to pick their location from a list.
- We would then like to display the current weather for that location on our mobile application.
- But how can we possibly store the current weather for EVERY location we wish to provide?
- **SOLUTION – Obtain access to an API which allows us to connect to a service (data source) of weather information**

Weather: https://open-meteo.com/en/docs



Rain and temp forecast for Maynooth

https://api.open-meteo.com/v1/forecast?latitude=53.384905&longitude=-6.598577&hourly=temperature_2m,rain



API Response (in JSON)

API Response (in JSON)
'Prettified' – for human reading

An App: This is a simplistic way the API call to Open Meteo works

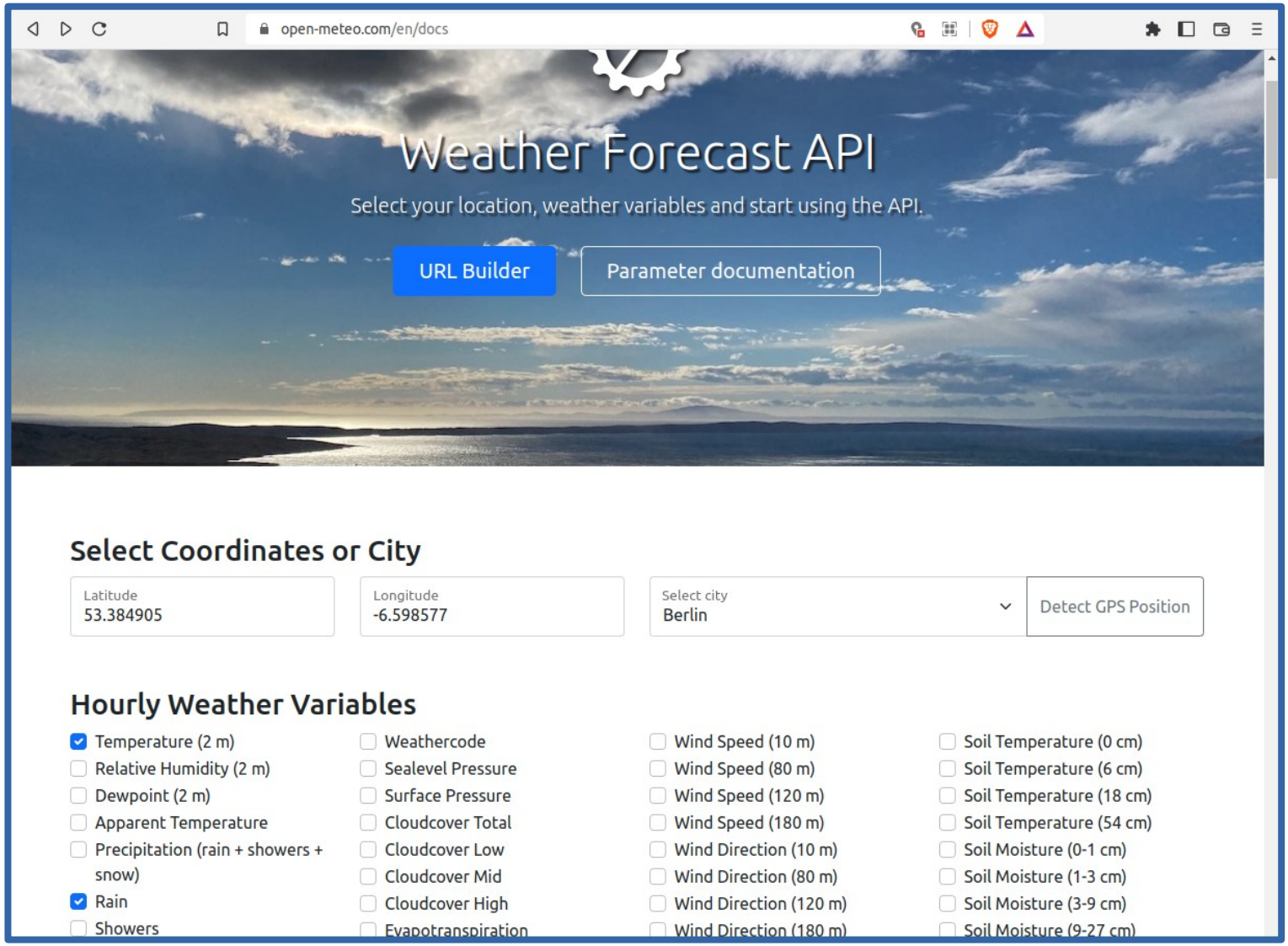
1) Our **user specifies their location** (maybe from a list)
2) Our **React App takes this location and creates a call to the OpenMeteo API** (by using the URL)
3) If successful t**he OpenMeteo API returns a RESPONSE** to our React Component.
4) We then **display the (response) weather for the users** area on our application (maybe using a map function)
5) We can **repeat this again and again for ANY location.**
6) **We don't need to maintain any arrays! Our data is real-time**

What is the RESPONSE from OpenMeteo?

- If we try out the URL
- https://api.open-meteo.com/v1/forecast?latitude=53.384905&longitude=-6.598577&hourly=temperature_2m,rain

- The URL has an **API ENDPOINT** called forecast. forecast takes parameters: latitude, longitude and the meteo parameters.
- Then the RESPONSE is generated in a special format called JSON. We've seen JSON objects already many times in CS385.

Good APIs will have documentation to help build URLS



This RESPONSE in JSON format can be CONSUMED by software

- **The reason why the initial JSON view is very hard to read is because it is not designed for human readibility.**
- **JSON responses (as we'll see) are designed for machine-to-machine communication or application-to-application communication.**
- Subsequently, JSON (JavaScript Object Notation) is a lightweight data-interchange format which plays a critical role in mobile application communications.

Sometimes APIs require you to register to use them (using an Access Token)

Instagram | Sandbox Invites | Manage Clients | Log in

- Overview
- Authentication
- Login Permissions
- Permissions Review
- Sandbox Mode
- Secure Requests
- Endpoints
  - Users
  - Media
  - Comments
  - Tags
  - Locations
- Embedding
- Mobile Sharing

Media Endpoints

GET /media/search ··· Search for recent media in a given area.

GET /media/search

https://api.instagram.com/v1/media/search?lat=48.858844&lng=2.294351&access_token=ACCESS-TO RESPONSE

Search for recent media in a given area.

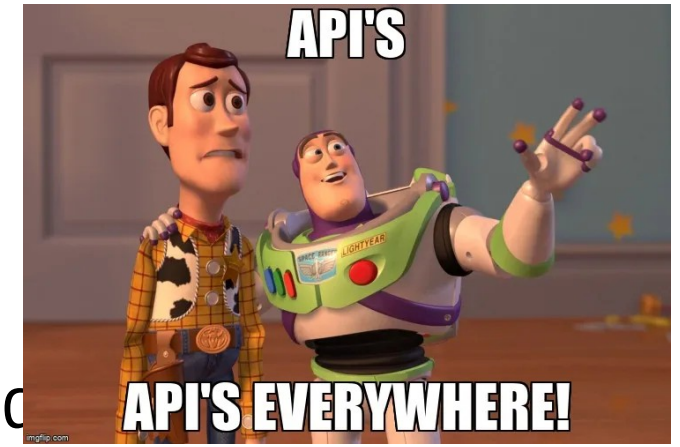
REQUIREMENTS

Scope: public_content

PARAMETERS

ACCESS_TOKEN	A valid access token.
LAT	Latitude of the center search coordinate. If used, lng is required.
LNG	Longitude of the center search coordinate. If used, lat is required.
DISTANCE	Default is 1km (distance=1000), max distance is 5km.

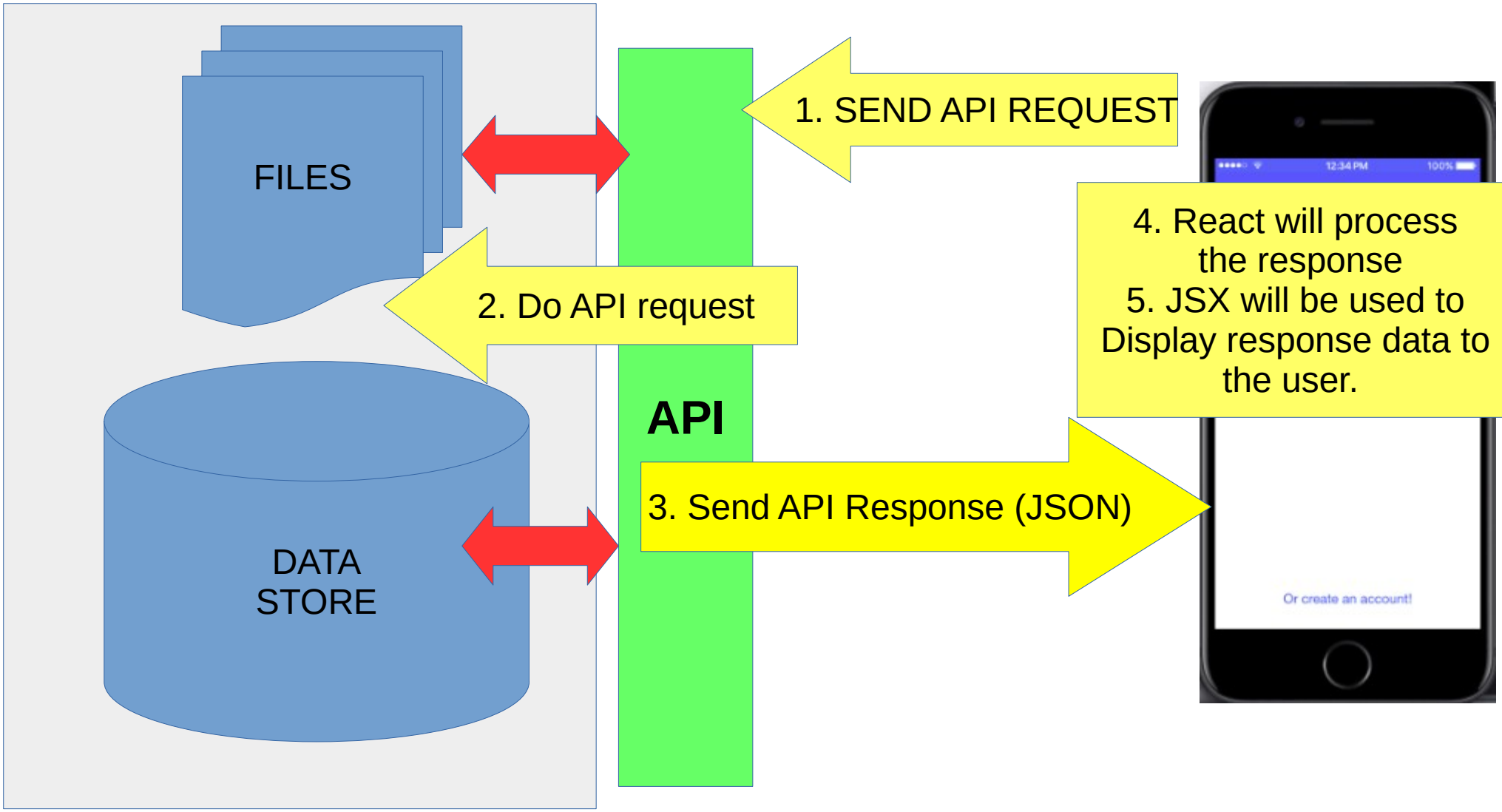
API Summary (for the moment)



- **APIs are EVERYWHERE on the Internet – providing us with access to data and services**
- Once we know the URL, the ENDPOINT and any parameters required then we can access the API service.
- In most cases the RESPONSE will be in JSON which then can be easily consumed by our application.
- We'll try to stick with using APIs which are free to use (with perhaps the need to register for free use)

So how do we connect an API to our React code?

A schematic of an API interaction with React



Let's consider a very simple example of how we can access an API with React.

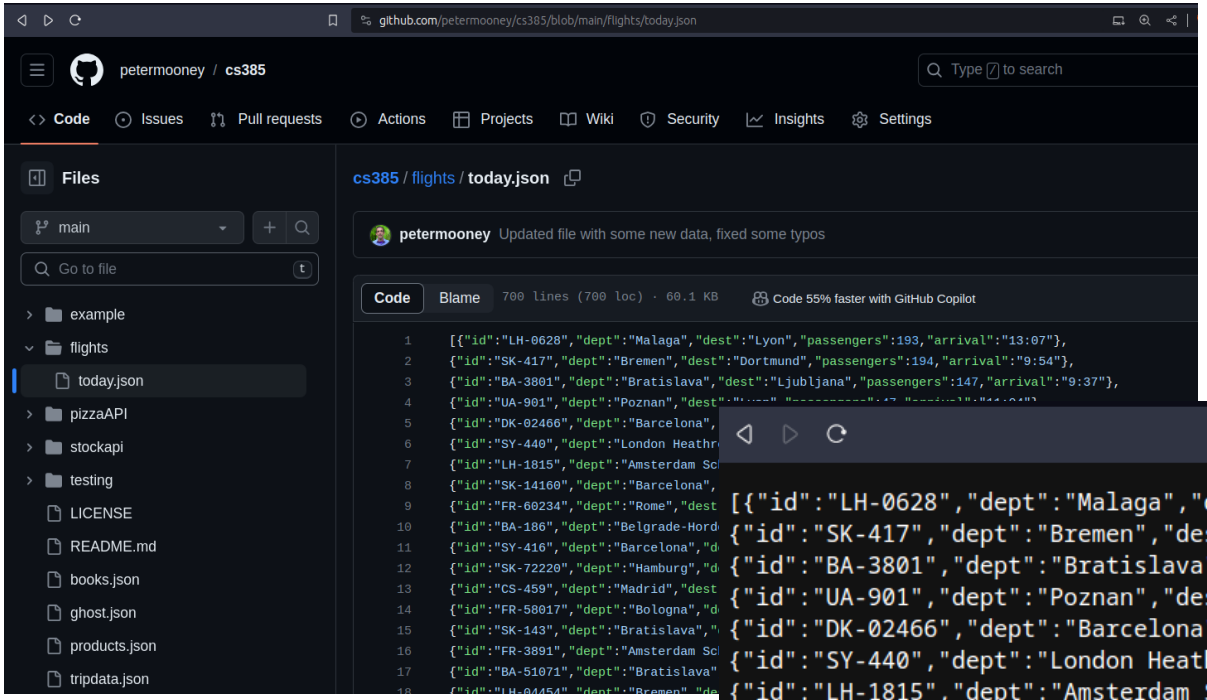
(source code [`Lecture9-FlightData-API-Example.js`] available on Moodle)

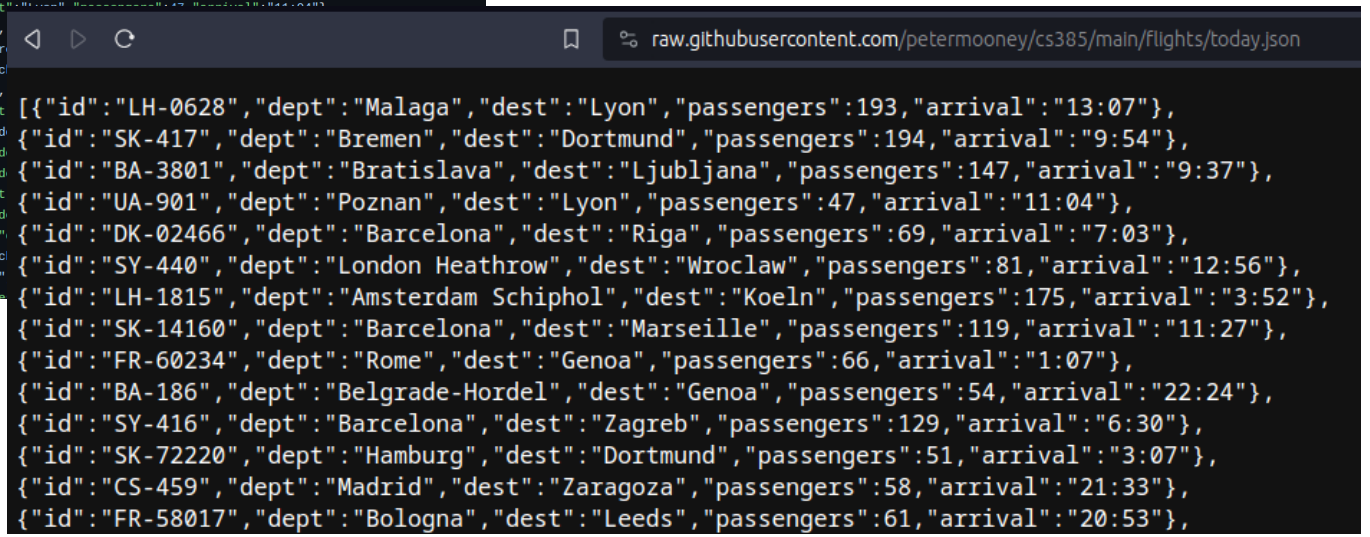
Notice – to students who have already programmed API calls. **There are LOTS of ways to do this same task in Javascript**. Lots of ways! The following slides describes just one of those approaches. It works, it's easy to understand, and it's reusable.

Feel free to use whatever approach works for you!

Where is our API?

- I've created a static JSON file on GitHub which allows us to simulate an API call. The URL will return a valid JSON response





Step 1: Let's declare some state variables using useState

- Notice that we've also imported **useEffect** (more later)
- Our three state variables are set to their initial or default states.

```
import React, { useEffect, useState } from "react";

function App() {
  // the data response from the API - initially empty array
  const [data, setData] = useState([]);
  // a flag to indicate the data is loading - initially false
  const [loading, setLoading] = useState(false);
  // a flag to indicate an error, if any - initially null.
  const [error, setError] = useState(null);
```

Step 2 – our first use of `useEffect()`

- In React, **`useEffect`** is a HOOK to allow interaction with external 'systems'

```
13  useEffect(() => {
14    // This is just a static JSON file - so not a 'real' API
15    const URL =
16      "https://raw.githubusercontent.com/petermooney/cs385/main/flights/today.json";
17
18    async function fetchData() {
19      try {
20        const response = await fetch(URL);
21        const json = await response.json(); // wait for the JSON response
22        setLoading(true);
23        // IMPORTANT - look at the JSON response - look at the structure
24        setData(json); // IMPORTANT - the JSON response is
25        // assigned to 'data' as an array of JSON objects
26      } catch (error) {
27        setError(error); // take the error message from the system
28        setLoading(false);
29      } // end try-catch block
30    } // end of fetchData
31
32    fetchData(); // invoke fetchData in useEffect
33  }, []); // end of useEffect
```

Step 2 – explaining `useEffect()`

- `useEffect` is a React Hook that lets you synchronize a component with an external system.
- **Effects let you run some code after rendering so that you can synchronize your component with some system outside of React.**
- **Effects let you specify side effects that are caused by rendering itself, rather than by a particular event.**
- Don't rush to add Effects to your components. Keep in mind that **Effects are typically used to "step out" of your React code and synchronize with some external system.** This includes browser APIs, third-party widgets, network, and so on.

Step 2 – here `useEffect()` is used to allow us to fetch data from an API

- We'll look at **`fetchData`** next. Notice line 33 – it's very important to notice the **empty [] - this means no state variables or props are involved in this effect.**

```
11  //useEffect is a React Hook
12   // lets you synchronize a component with an external system.
13  useEffect(() => {
14    // This is just a static JSON file - so not a 'real' API
15    const URL =
16      "https://raw.githubusercontent.com/petermooney/cs385/main/flights/today.json";
17
18    async function fetchData() {
19      try {
20        const response = await fetch(URL);
21        const json = await response.json(); // wait for the JSON response
22        setLoading(true);
23        // IMPORTANT - look at the JSON response - look at the structure
24        setData(json); // IMPORTANT - the JSON response is
25        // assigned to 'data' as an array of JSON objects
26      } catch (error) {
27        setError(error); // take the error message from the system
28        setLoading(false);
29      } // end try-catch block
30    } // end of fetchData
31
32    fetchData(); // invoke fetchData in useEffect
33  }, []); // end of useEffect
```

What does async mean?

- **Async** – means **asynchronous** which is the opposite in effect to synchronous
- In the example below, see how task A is running but task B and C can also be happening at the same time, until eventually all tasks are comp

Synchronous (one thread):

```
1 thread ->   |<---A---->||<----B---------->||<------C----->|
```

Synchronous (multi-threaded):

```
thread A -> |<---A---->|
                      \
thread B ------------>   ->|<----B---------->|
                                              \
thread C ---------------------------------->   ->|<------C----->|
```

Asynchronous (one thread):

```
         A-Start ------------------------------------------ A-End
           | B-Start ----------------------------------------|--- B-End
           |    |      C-Start ------------------ C-End      |      |
           |    |       |                          |         |      |
           V    V       V                          V         V      V
1 thread->|<-A-|<--B---|<-C-|-A-|-C-|--A--|-B-|--C-->|---A---->|--B-->|
```

Asynchronous (multi-Threaded):

```
thread A ->     |<---A---->|
thread B ----->     |<----B---------->|
thread C --------->     |<------C--------->|
```

- Start and end points of tasks A, B, C represented by `<`, `>` characters.

What is **async** again?

- Using the **async** keyword means that we make the function or method an asynchronous function.
- You'll really see what this means when we look at the **render** function.
- As the function `fetchData` is trying to access the API and gather there response React can be busy as a front-end trying to render something on screen.
- **So there are several tasks as happening at once! (Asynchronously)**

Step 3 – fetching our data from the API or external source

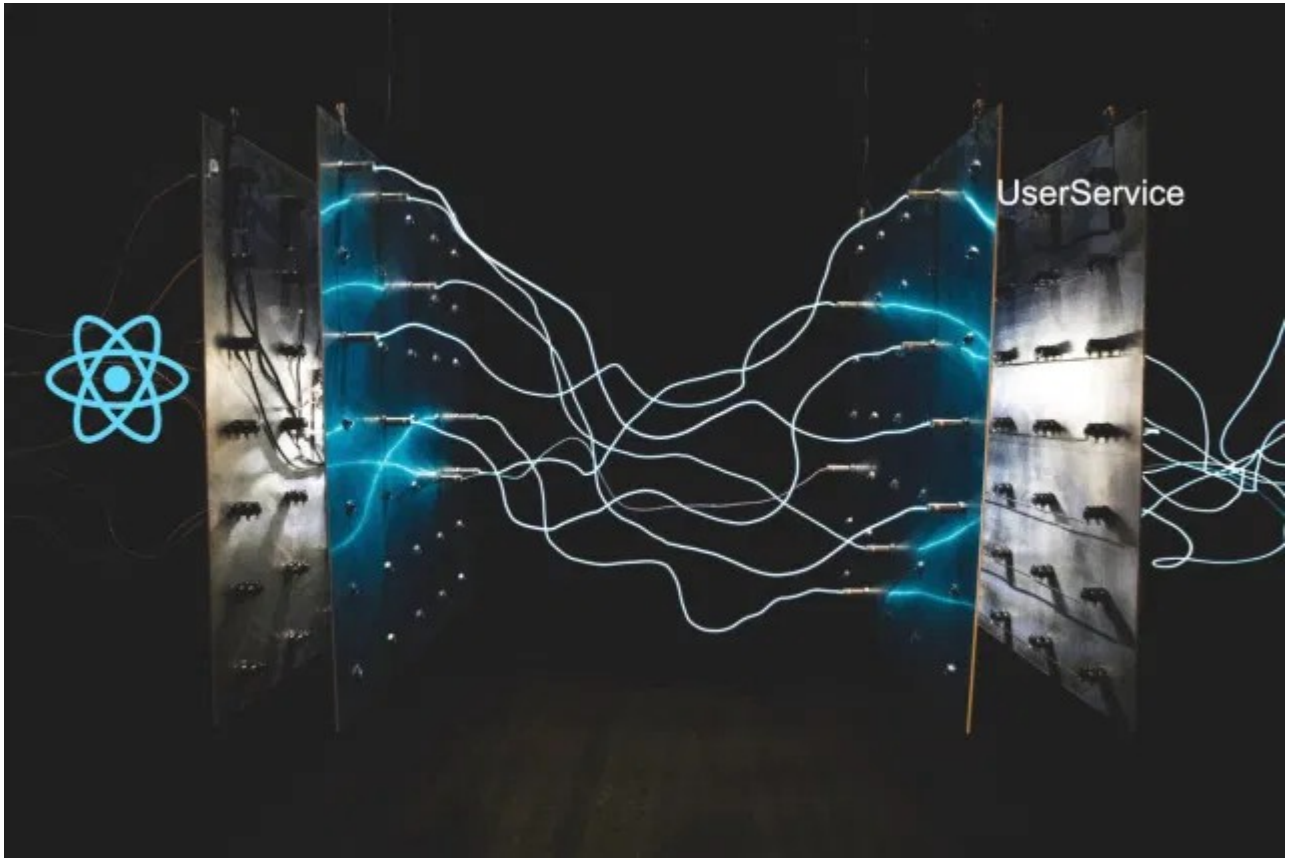
- We use a **`try-catch`** block (1st time). We use our set methods from the **`useState`** hook.
- **Line 21 is the crucial line** – waiting for the API to respond with our JSON data

```
18    async function fetchData() {
19      try {
20        const response = await fetch(URL);
21        const json = await response.json(); // wait for the JSON response
22        setLoading(true);
23        // IMPORTANT - look at the JSON response - look at the structure
24        setData(json); // IMPORTANT - the JSON response is
25        // assigned to 'data' as an array of JSON objects
26      } catch (error) {
27        setError(error); // take the error message from the system
28        setLoading(false);
29      } // end try-catch block
30    } // end of fetchData
```

And whs is this `fetch()` call?

- The `fetch()` function is provided by React – and in CS385 we don't really need to worry about HOW the `fetch()` function retrieves all this JSON from APIs for us.
- In many ways `fetch()` is doing a very similar job to your web browser if you asked your browser to display the URL we've seen.
- Behind the scenes of `fetch()` is some pretty complicated network programming (as there is a lot to do in order to fetch the API response)
- This complexity, lucky for us, is encapsulated (or hidden) by the `fetch()` function

How React `fetch()` really works ;-)



Step 4 – Rendering the response from our API call.

- Here the **App()** component can render in an **async** way – as the state variables (**`error, loading, data`**) change – the whole component is RE-RENDERED – therefore line 39 – 50 is executed each time.

```
39    if (error) {
40      return <h1>Opps! An error has occurred: {error.toString()}</h1>;
41    } else if (loading === false) {
42      return <h1>Loading Data...... please wait!</h1>;
43    } else {
44      return (
45        // send the data to the ResultsComponent for render
46        <>
47          <ResultsComponent APIData={data} />
48        </>
49      );
50    } // end else
51  } // end App() function or component
```

Step 5 – our response – rendered

App.js

```
function App() {
  // the data response from the API - initially empty array
  const [data, setData] = useState([]);
  // a flag to indicate the data is loading - initially false
  const [loading, setLoading] = useState(false);
  // a flag to indicate an error, if any - initially null.
  const [error, setError] = useState(null);

  //useEffect is a React Hook
  // lets you synchronize a component with an external system.
  useEffect(() => {
    // This is just a static JSON file - so not a 'real' API
    const URL =
      "https://raw.githubusercontent.com/petermooney/cs385/main/flights/today.js

    async function fetchData() {
      try {
        const response = await fetch(URL);
        const json = await response.json(); // wait for the JSON response
        setLoading(true);
        // IMPORTANT - look at the JSON response - look at the structure
        setData(json); // IMPORTANT - the JSON response is
        // assigned to 'data' as an array of JSON objects
      } catch (error) {
        setError(error); // take the error message from the system
        setLoading(false);
      } // end try-catch block
    } // end of fetchData

    fetchData(); // invoke fetchData in useEffect
  }, []); // end of useEffect
```

Browser Tests

https://7gr67n.csb.app/

Number of flights returned: 18

Flight No.	Dept.	Dest	Arrival
LH-0628	Malaga	Lyon	13:07
SK-417	Bremen	Dortmund	9:54
BA-3801	Bratislava	Ljubljana	9:37
UA-901	Poznan	Lyon	11:04
DK-02466	Barcelona	Riga	7:03
SY-440	London Heathrow	Wroclaw	12:56
LH-1815	Amsterdam Schiphol	Koeln	3:52
SK-14160	Barcelona	Marseille	11:27
FR-60234	Rome	Genoa	1:07
BA-186	Belgrade-Hordel	Genoa	22:24
SY-416	Barcelona	Zagreb	6:30
SK-72220	Hamburg	Dortmund	3:07
CS-459	Madrid	Zaragoza	21:33
FR-58017	Bologna	Leeds	20:53
SK-143	Bratislava	Naples	9:19
FR-3891	Amsterdam Schiphol	Sarajevo	6:49
BA-51071	Bratislava	Turin	3:35
LH-04454	Bremen	Valencia	6:48

Console 0 Problems 1 React DevTools 0

Console was cleared

Step 5 – The `ResultsComponent` (child) takes care of rendering the response from the API

```
57  function ResultsComponent(props) {
58    // To reduce the array size – let's just take
59    // a smaller number of elements.
60    let dataLength = props.APIData.length;
61    let n = Math.floor(Math.random() * dataLength);
62    // now use slice() to take the first n objects in the array
63    let localAPIData = props.APIData.slice(0, n);
64    return (
65      <>
66        <h1>Number of flights returned: {localAPIData.length}</h1>
67        <table border="1">
68          <thead>
69            <tr>
70              <th>Flight No.</th>
71              <th>Dept.</th>
72              <th>Dest</th>
73              <th>Arrival</th>
74            </tr>
75          </thead>
76          <tbody>
77            {localAPIData.map((p, index) => (
78              <tr key={index}>
79                <td>
80                  <b>{p.id}</b>
81                </td>
82                <td>
83                  <b>{p.dept}</b>
84                </td>
85                <td>
86                  <b>{p.dest}</b>
87                </td>
```

Number of flights returned: 18

Flight No.	Dept.	Dest	Arrival
LH-0628	Malaga	Lyon	13:07
SK-417	Bremen	Dortmund	9:54
BA-3801	Bratislava	Ljubljana	9:37
UA-901	Poznan	Lyon	11:04
DK-02466	Barcelona	Riga	7:03
SY-440	London Heathrow	Wroclaw	12:56
LH-1815	Amsterdam Schiphol	Koeln	3:52
SK-14160	Barcelona	Marseille	11:27
FR-60234	Rome	Genoa	1:07
BA-186	Belgrade-Hordel	Genoa	22:24
SY-416	Barcelona	Zagreb	6:30
SK-72220	Hamburg	Dortmund	3:07
CS-459	Madrid	Zaragoza	21:33
FR-58017	Bologna	Leeds	20:53
SK-143	Bratislava	Naples	9:19
FR-3891	Amsterdam Schiphol	Sarajevo	6:49
BA-51071	Bratislava	Turin	3:35
LH-04454	Bremen	Valencia	6:48

Console 0 Problems 1 React DevTools 0

Console was cleared

Step 5 – there are 100's of objects being returned

- **JUST FOR DISPLAY PURPOSES ON THESE SLIDES** – I have added a few lines of JavaScript to randomly select the first 'n' objects

```
57  function ResultsComponent(props) {
58    // To reduce the array size - let's just take
59    // a smaller number of elements.
60    let dataLength = props.APIData.length;
61    let n = Math.floor(Math.random() * dataLength);
62    // now use slice() to take the first n objects in the ar
63    let localAPIData = props.APIData.slice(0, n);
64    return (
65      <>
66        <h1>Number of flights returned: {localAPIData.length}</h1>
67        <table border="1">
68          <thead>
```

Browser Tests

https://7gr67n.csb.app/

Number of flights returned: 457

Flight No.	Dept.	Dest	Arrival
LH-0628	Malaga	Lyon	13:07
SK-417	Bremen	Dortmund	9:54
BA-3801	Bratislava	Ljubljana	9:37
UA-901	Poznan	Lyon	11:04
DK-02466	Barcelona	Riga	7:03

Browser Tests

https://7gr67n.csb.app/

Number of flights returned: 231

Flight No.	Dept.	Dest	Arrival
LH-0628	Malaga	Lyon	13:07
SK-417	Bremen	Dortmund	9:54
BA-3801	Bratislava	Ljubljana	9:37
UA-901	Poznan	Lyon	11:04
DK-02466	Barcelona	Riga	7:03
SY-440	London Heathrow	Wroclaw	12:56

WARNING – Always check the JSON response – there are some situations to be aware of

Example 1: Watch for array names within the JSON response

- https://raw.githubusercontent.com/petermooney/cs385/main/flights/live.json


```
{"flightData" :[{"id":"DK-90025","dept":"Bydgoszcz","dest":"Frankfurt am Main","p
{"id":"FH-5674","dept":"Las Palmas de Gran Canaria","dest":"Antwerpen","passenger
{"id":"IE-464","dept":"Skopje","dest":"Dortmund","passengers":87,"arrival":"0:03"
{"id":"IE-31602","dept":"Tirana","dest":"Koeln","passengers":176,"arrival":"4:48"
{"id":"BA-4137","dept":"Munich","dest":"Glasgow","passengers":188,"arrival":"17:00"},
{"id":"SK-74758","dept":"
{"id":"FH-709","dept":"Sk
{"id":"BA-2406","dept":"S
{"id":"SK-849","dept":"Ma
```


```
useEffect(() => {
  // This is just a static JSON file - so not a 'real' API
  const URL =
    "https://raw.githubusercontent.com/petermooney/cs385/main/flights/live.json";

  async function fetchData() {
    try {
      const response = await fetch(URL);
      const json = await response.json(); // wait for the JSON response
      setLoading(true);
      // IMPORTANT - look at the JSON response - look at the structure
      setData(json.flightData); // IMPORTANT - the JSON response is
      // assigned to 'data' as an array of JSON objects
    } catch (error) {
      setError(error); // take the error message from the system
      setLoading(false);
    } // end try-catch block
  } // end of fetchData
```

Example 2 – Watch for NESTED ARRAYS within the JSON response

- https://raw.githubusercontent.com/petermooney/cs385/main/flights/livefull.json

```json
{
    "flightTracker": [
        {
            "id": "FR-169",
            "dept": "London Heathrow",
            "dest": "Leeds",
            "passengers": 197,
            "arrival": "21:44",
            "aircraft": {
                "type": "737-800",
                "crew": 7,
                "captain": "Correy Tabourin"
            }
        },
        {
            "id": "FH-864",
            "dept": "Tirana",
            "dest": "Stuttgart",
            "passengers": 144,
            "arrival": "0:37",
            "aircraft": {
                "type": "737-800",
                "crew": 7,
                "captain": "Lexi Wenzel"
            }
```

```
const URL =
  "https://raw.githubusercontent.com/petermooney/cs385/main/flights/livefull.json";

async function fetchData() {
  try {
    const response = await fetch(URL);
    const json = await response.json(); // wait for the JSON response
    setLoading(true);
    // IMPORTANT - look at the JSON response - look at the structure
    setData(json.flightTracker); // IMPORTANT - the JSON response is
    // assigned to 'data' as an array of JSON objects
```

```
    </tr>
  </thead>
  <tbody>
    {localAPIData.map((p, index) => (
      <tr key={index}>
        <td>
          <i>{p.aircraft.captain}</i>
        </td>
        <td>
          <b>{p.id}</b>
        </td>
        <td>
          <b>{p.dept}</b>
        </td>
```

Number of flights return

Pilot	Flight No.	De
Correy Tabourin	**FR-169**	**London Heath**
Lexi Wenzel	**FH-864**	**Tirana**
Ellsworth Doblin	**PM-165**	**Skopje**
Gill Flemming	**SY-033**	**Helsinki**
Turner Aguirrezabal	**BA-825**	**Barcelona**
Miguelita Meade	**SY-41087**	**Berlin**
Ula Rea	**FH-22198**	**Tallinn**
Kimberlyn Dedman	**FR-630**	**Bremen**
Britney Pinckard	**UA-59368**	**Bochum**

RECAP of APIs

We have decoupled our application from using arrays in JS files

- **Now, as the data at the source of the API changes, our application is always up-to-date.**
- We don't have to worry if new data is made available.
- **As developers we just have to ensure that we keep up-to-date with the API itself** – in case some of the JSON structure changes or new properties are added.
- Then our map function (and hence the application code) would need to be updated.
- But this is a reasonably rare event for most APIs.

Can I use a different API?

- Yes, if you have an API then you can basically reuse this code example here (if you only want a basic app)
- Obviously, you'll need to change how the map function works in the render() method.
- **Your map function will have to "map" directly onto the structure of the JSON returned from your specific API. <u>So you'll need to study the DATA MODEL of the JSON response</u>**

Word of wisdom – APIs change.

- Think of an API as a service.
- Services change sometimes.
- An API can change its URL , its parameters or the structure or data model of the JSON returned.
- In this way, they add an administrative overhead to any application.

Before you use an API...

1) **Read the documentation** about the API (what does it do, what does it not do, etc)

2) Try it out in the web-browser.

3) **Understand the JSON returned** (structure, nesting, etc)

4) **Copy JSON into JSONLint (or other) to see the output in human-readable format.**

5) **When you understand the structure THEN you can start building your React application**

End of Lecture 9