JavaScript

Synchronous and Asynchronous

what is mean??????????

Control Flow

is the order in which statements are executed in a program.

By default, JavaScript runs code from top to bottom and left to right.

Control flow statements let you change that order, based on conditions, loops or keywords.

Default Flow

Default flow executes code sequentially (from top to bottom / from left to right).

Example :

```
console.log("1");
console.log("2");
console.log("3");
```

Control Flow

Conditional Control Flow

- Conditions let you make decisions using:
  - if
  - if...else
  - switch

Example :

```
let text = "Unknown";

if (age >= 18) {
  text = "Adult";
} else {
  text = "Minor";
}
```

Control Flow

Loops (Repetition Control Flow)

Loops let you run code multiple times using:

- for
- while
- do...while

Function Flow

Functions are callable and reusable code blocks

Control Flow

Jump Statements

Jump statements let you change the flow abruptly using:

- break - exits a loop or switch
- continue - skips the current loop iteration
- return - exits from a function
- throw - jumps to error handling

JavaScript Is Single-Threaded

- **JavaScript runs on a single thread.**
- **It can only do one thing at a time.**
- **Every task has to wait for the previous one to finish.**
- **This can freeze an application during slow operations (like file requests).**

Sync Programming

Synchronous = Line by line execution

- JavaScript executes one line at a time
- A new line cannot start until the previous one finishes
- This is also called Blocking

Example :

```
console.log("1");
console.log("2");
console.log("3");
```

Sync Programming

Why is Sync a Problem??

```
console.log("Start");

for (let i = 0; i < 10000000; i++) {
    console.log("Loop");
}

console.log("End");
```

- 👉 The browser freezes
- 👉 UI stops responding
- 👉 Bad user experience

Asynchronus Flow

JavaScript Asynchronous Flow refers to how JavaScript handles tasks that take time to complete, like reading files, or waiting for user input, without blocking the execution of other code.

To prevent blocking, JavaScript can use asynchronous programming.

This allows certain operations to run in the background, and their results are handled later, when they are ready.

Real-Life Example (Simple)

Synchronous life example:

- You stand in a queue…
- You cannot do anything until it’s your turn.

Asynchronous life example:

- You order food →
- They call you when it’s ready →
- Meanwhile you sit, talk, work…

Asynchronous

What is “Asynchronous”?

Asynchronous = Code that can run later, without blocking

- JS sends the task to the browser (Web APIs)
- JS continues executing the rest of the code
- When the task finishes → JS gets the result

This is called Non-Blocking.

Example :

```
console.log("1");

setTimeout(() => {
  console.log("2");
}, 2000);

console.log("3");
```

Why Async is Important

- UI doesn’t freeze
- Performance is better
- We can make API requests
- User interacts normally
- Modern web apps depend on it

Asynchronus Patterns

- Promises
- Async/Await
- Events Loops

Promises

A Promise is an object representing the eventual completion (or failure) of an async operation. It has three states: pending, fulfilled, rejected.

- resolved
- rejected

You use .then() and .catch().

```
function getData() {
  return new Promise((resolve, reject) => {
    setTimeout(() => resolve("Data Loaded"), 1000);
  });
}

getData().then(result => console.log(result));
```

Promises

Example:

```
function getUser(id) {
  return new Promise((resolve, reject) => {
    setTimeout(() => {
      resolve({ id: id, name: "Ahmed" });
      // reject(new Error("User not found"));
    }, 2000);
  });
}

getUser(1)
  .then(user => console.log("User:", user))
  .catch(err => console.error(err));
```

Async / Await

async/await is built on top of Promises.

It makes asynchronous code look synchronous.

```
async function displayUser() {
  try {
    const user = await getUser(1);
    console.log("User:", user);
  } catch (err) {
    console.error(err);
  }
}

displayUser();
```

```
function getData() {
  return new Promise(resolve => {
    setTimeout(() => resolve("Hello"), 1000);
  });
}

async function show() {
  let result = await getData();
  console.log(result);
}
show();
```

Modern Recommendation

Use Async/Await + Promise.all for 99% of cases.

```
async function getFullUserData(userId) {
  try {
    const [user, posts, friends, notifications] = await Promise.all([
      api.getUser(userId),
      api.getPosts(userId),
      api.getFriends(userId),
      api.getNotifications(userId)
    ]);

    const enrichedPosts = await enrichPostsWithLikes(posts);

    return { user, posts: enrichedPosts, friends, notifications };
  } catch (error) {
    logError(error);
    throw new AppError("Failed to load user data", 500);
  }
}
```

Task

Fake Weather App – Get weather for city "Paris"

What it does:

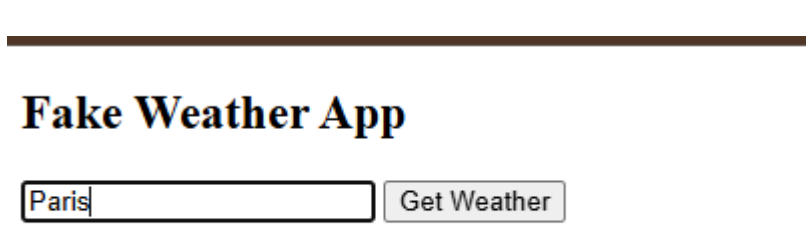
Click button → shows “Loading…” for 2 seconds

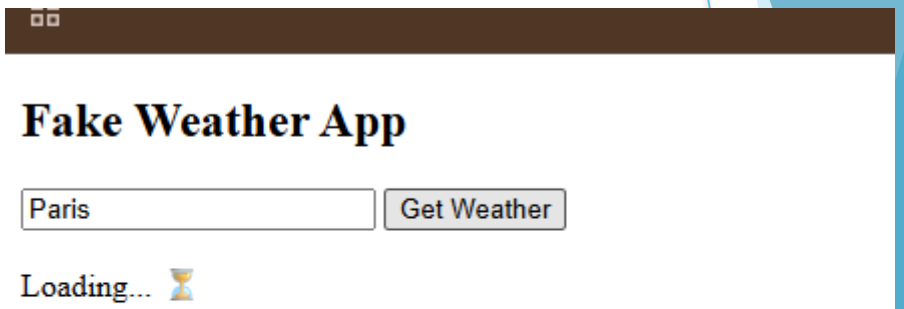
If city is exactly "Paris" → success

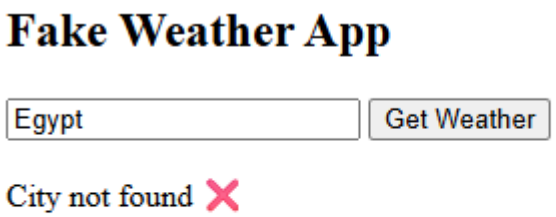
Any other city → error

Uses Promisea + async/await + try/catch

Task


Fake Weather App
Paris
Get Weather


Fake Weather App
Paris
Get Weather
Loading...


Fake Weather App
Egypt
Get Weather
City not found

Event Loop

The Event Loop decides when async code runs

JS has:

- Call Stack
- Web APIs
- Callback Queue
- Microtask Queue (Promises)

```
console.log("1");

setTimeout(() => console.log("2"), 0);

Promise.resolve().then(() => console.log("3"));

console.log("4");
```

How JavaScript Works Internally

JavaScript has 3 main parts:

1 Call Stack

Where synchronous code runs.

- Executes one thing at a time.
- If a function is running → stack is busy.

2 Web APIs

The browser provides async features like:

- setTimeout
- fetch
- DOM events (click, keyup)
- AJAX calls

They run outside the JS engine.

- Microtask Queue → promises, async/await

How JavaScript Works Internally

3️⃣ Task Queues

Where asynchronous callbacks wait:

Two types:

- Callback Queue / Macrotask Queue → setTimeout, events
- Microtask Queue → promises, async/await

How the Event Loop Works

- Executes all synchronous code in the call stack
- When stack is empty:
- Check microtask queue (first priority)
- Then check callback queue
- Take the first task → push to stack → run it
- Repeat forever

Important Rules

Microtasks run before Macrotasks

Example microtasks:

- .then()
- async/await

Example macrotasks:

- setTimeout
- DOM events

Event Loop

Example

```
console.log("Start");

setTimeout(() => {
  console.log("Timeout");
}, 0);

Promise.resolve().then(() => {
  console.log("Promise");
});

console.log("End");
```

=> output

Event Loop

Example :

```
console.log("Start");

setTimeout(() => {
  console.log("Timeout");
}, 0);

Promise.resolve().then(() => {
  console.log("Promise");
});

console.log("End");
```

=> output

```
Start
End
Promise
Timeout
```

Why?

- console.log("Start") → runs
- setTimeout goes to Web APIs → macrotask queue
- Promise → microtask queue
- console.log("End")
- Event loop checks queues:
- Microtask first → Promise
- Then macrotask → Timeout

Event Loop

Example :

```
console.log("A");

setTimeout(() => console.log("B"), 0);

Promise.resolve().then(() => console.log("C"));

Promise.resolve().then(() => {
    console.log("D");
    setTimeout(() => console.log("E"), 0);
});

console.log("F");
```

=> output

Event Loop

Example :

```
console.log(1);

setTimeout(() => console.log(2), 0);

Promise.resolve().then(() => console.log(3));

for (let i = 0; i < 10000000; i++) {}

console.log(4);
```

=> output

Task

Build a small “Fake Login System” using async concepts:

✓ Show “Loading…” for 2s

✓ If username === "admin", resolve promise

✓ Else reject

✓ Use async/await to call the function

✓ Use try/catch to handle error

✓ Show messages in the DOM using events

what is an Api?

API – (Application Programming Interface) A developer extensively uses APIs in his software to implement various features by using an API call without writing complex codes for the same. We can create an API for an operating system, database system, hardware system, JavaScript file, or similar object-oriented files. Also, an API is similar to a GUI(Graphical User Interface) with one major difference. Unlike GUIs, an API helps software developers to access web tools while a GUI helps to make a program easier to understand by users.

How do APIs Work?

The client initiates the requests via the APIs URI (Uniform Resource Identifier)

The API makes a call to the server after receiving the request

Then the server sends the response back to the API with the information

Finally, the API transfers the data to the client

Type of APIs

- WEB APIs

A Web API also called Web Services is an extensively used API over the web and can be easily accessed using the HTTP protocols(Rest api). A Web API is an open-source interface and can be used by a large number of clients through their phones, tablets, or PCs.

Types of Web APIs

SOAP (SIMPLE OBJECT ACCESS PROTOCOL):

- It defines messages in XML format used by web applications to communicate with each other.

REST (Representational State Transfer):

- It makes use of HTTP to GET, POST, PUT, or DELETE data. It is basically used to take advantage of the existing data.

REST Api

HTTP Methods and Operations:

- GET: Retrieve a representation of a resource.
- POST: Create a new resource or perform a custom action.
- PUT: Update a resource or create it if it doesn't exist.
- DELETE: Remove a resource.
- These methods map to CRUD (Create, Read, Update, Delete) operations on resources.

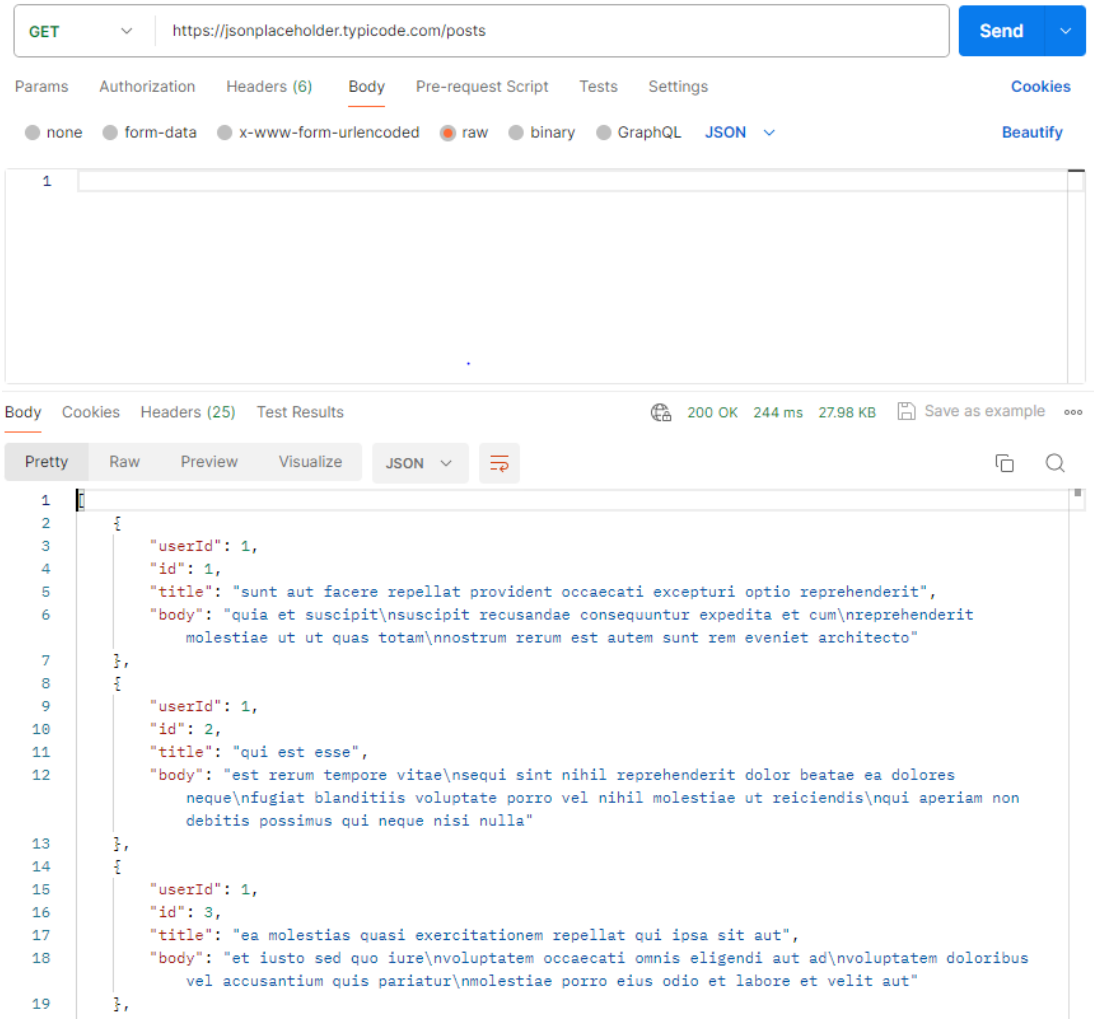
Components of an API

- Endpoints: Specific URLs or URIs where API requests can be made to access certain functionalities.
- Request and Response: APIs involve sending requests (commands or queries) from a client to an API, and receiving responses that contain the requested data or confirmations.

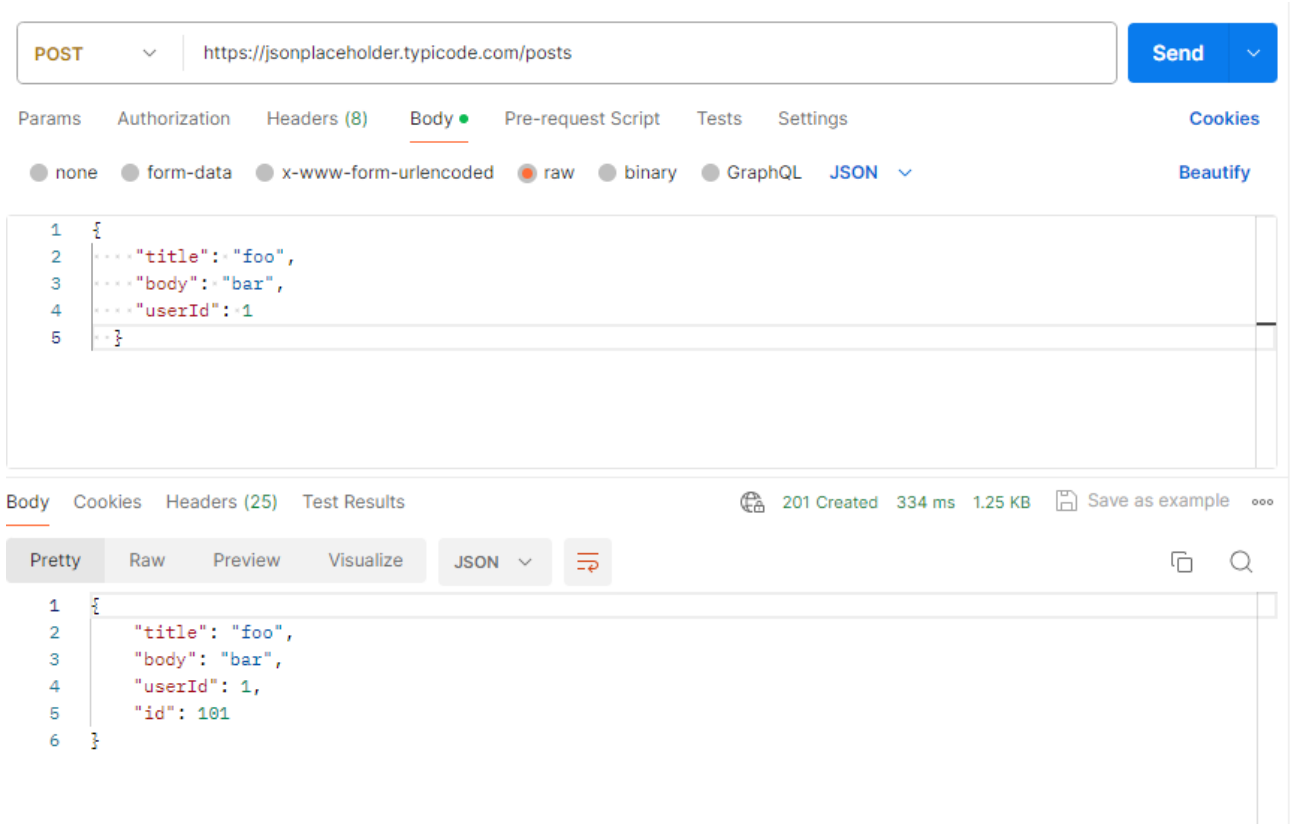
API Testing Tools

- Postman
- Apigee
- JMeter
- Ping API
- Soap UI
- vREST
- we can use postman to test apis now: https://web.postman.co/https://www.postman.com/downloads/
- and we can use this data: https://jsonplaceholder.typicode.com/guide/

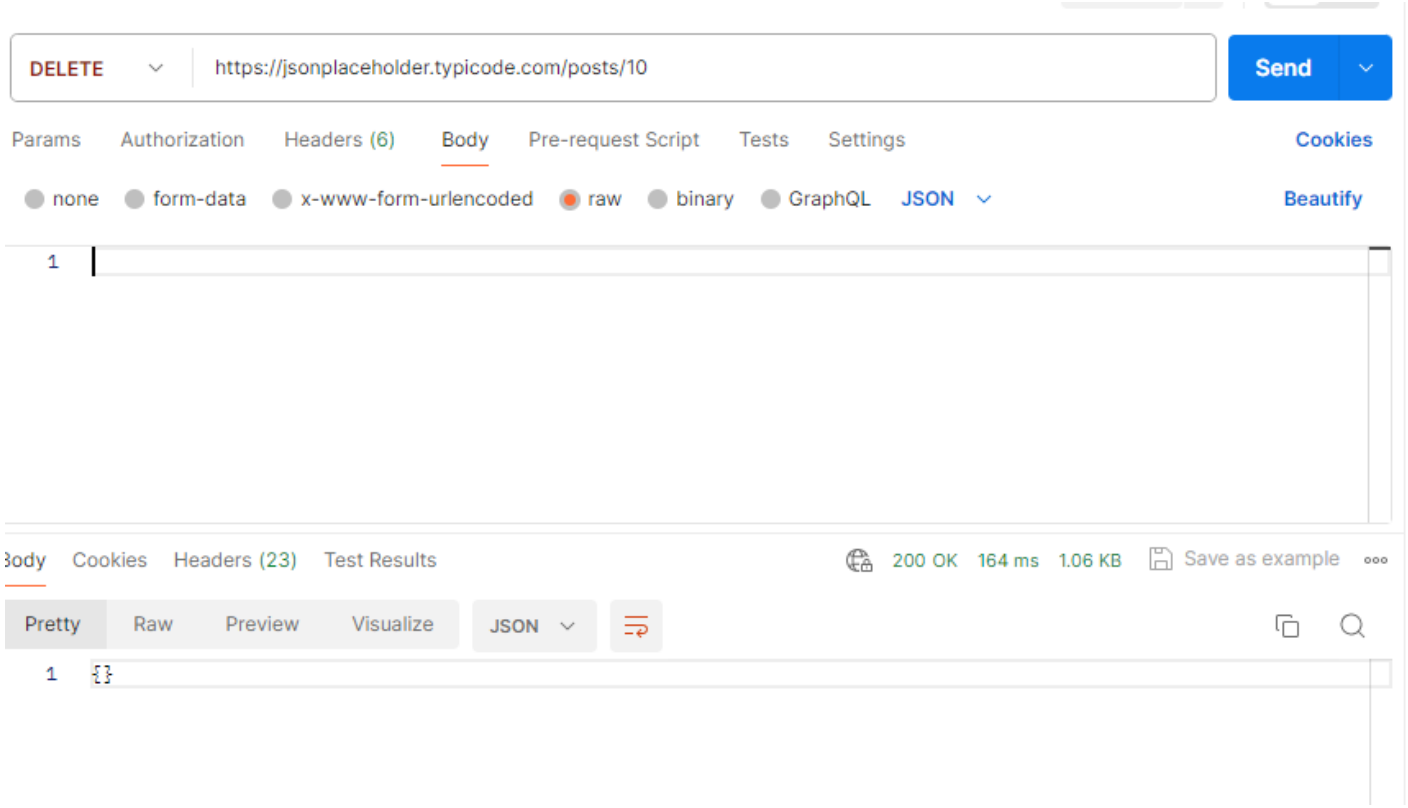
Get Method


GET
https://jsonplaceholder.typicode.com/posts
Send
Params
Authorization
Headers (6)
Body
Pre-request Script
Tests
Settings
Cookies
none
form-data
x-www-form-urlencoded
raw
binary
GraphQL
JSON
Beautify
Body
Cookies
Headers (25)
Test Results
200 OK
244 ms
27.98 KB
Save as example
Pretty
Raw
Preview
Visualize
JSON
[
{
"userId": 1,
"id": 1,
"title": "sunt aut facere repellat provident occaecati excepturi optio reprehenderit",
"body": "quia et suscipit\nsuscipit recusandae consequuntur expedita et cum\nreprehenderit molestiae ut ut quas totam\nnostrum rerum est autem sunt rem eveniet architecto"
},
{
"userId": 1,
"id": 2,
"title": "qui est esse",
"body": "est rerum tempore vitae\nsequi sint nihil reprehenderit dolor beatae ea dolores neque\nfugiat blanditiis voluptate porro vel nihil molestiae ut reiciendis\nqui aperiam non debitis possimus qui neque nisi nulla"
},
{
"userId": 1,
"id": 3,
"title": "ea molestias quasi exercitationem repellat qui ipsa sit aut",
"body": "et iusto sed quo iure\nvoluptatem occaecati omnis eligendi aut ad\nvoluptatem doloribus vel accusantium quis pariatur\nmolestiae porro eius odio et labore et velit aut"
},

Post Method


POST
https://jsonplaceholder.typicode.com/posts
Send
Params
Authorization
Headers (8)
Body
Pre-request Script
Tests
Settings
Cookies
none
form-data
x-www-form-urlencoded
raw
binary
GraphQL
JSON
Beautify
{
"title": "foo",
"body": "bar",
"userId": 1
}
Body
Cookies
Headers (25)
Test Results
201 Created
334 ms
1.25 KB
Save as example
Pretty
Raw
Preview
Visualize
JSON
{
"title": "foo",
"body": "bar",
"userId": 1,
"id": 101
}

Delete Method



Fetch

A built-in browser function for calling APIs using promises.

```
fetch(url)
  .then(response => response.json())
  .then(data => console.log(data));
```

Example

```
fetch("https://jsonplaceholder.typicode.com/posts/1")
  .then(res => res.json())
  .then(data => console.log("Post:", data));
```

Fetch

Task: Fetch User → Then Fetch User's Todos

```
async function getUserTodosAsync() {
  try {
    const userRes = await fetch("https://jsonplaceholder.typicode.com/users/3");
    const user = await userRes.json();

    const todoRes = await fetch(`https://jsonplaceholder.typicode.com/todos?userId=${user.id}`);
    const todos = await todoRes.json();

    console.log("User Todos (Async):", todos);
  } catch (err) {
    console.error("Error:", err.message);
  }
}

getUserTodosAsync();
```

Handling JSON Data

```
fetch("https://jsonplaceholder.typicode.com/todos")
  .then(res => res.json())
  .then(todos => todos.forEach(t => console.log(t.title)));
```

Task

Students must:

- Count how many todos are completed.
- Display only titles longer than 20 characters.
- Map todos into a new array with {id, titleLength}.

Task

```
fetch("https://jsonplaceholder.typicode.com/todos")
  .then(res => res.json())
  .then(todos => {
    // 1. Count completed todos
    const completedCount = todos.filter(t => t.completed).length;

    // 2. Titles longer than 20 characters
    const longTitles = todos
      .filter(t => t.title.length > 20)
      .map(t => t.title);

    // 3. New array with id + titleLength
    const mappedData = todos.map(t => ({
      id: t.id,
      titleLength: t.title.length
    }));

    console.log("Completed Todos:", completedCount);
    console.log("Titles > 20 chars:", longTitles);
    console.log("Mapped Data:", mappedData);
  })
  .catch(err => console.error("Error:", err));
```

API Error Handling

Types of Errors

- Network error → no internet
- Server error → status 500
- Not found → status 404
- Unauthorized → status 401
- Bad request → status 400

API Error Handling

How to handle errors in Fetch?

API Error Handling

How to handle errors in Fetch?

✓ Check response status

```
fetch(url)
  .then(res => {
    if (!res.ok) {
      throw new Error("HTTP Error " + res.status);
    }
    return res.json();
  })
  .then(data => console.log(data))
  .catch(err => console.error("Error:", err.message));
```

API Error Handling

How to handle errors in Fetch?

✓ Try/Catch with Async/Await

```
async function getUser() {
  try {
    let res = await fetch("https://jsonplaceholder.typicode.com/users/1");

    if (!res.ok) {
      throw new Error("Server responded with: " + res.status);
    }

    let data = await res.json();
    console.log(data);
  } catch (err) {
    console.log("Error:", err.message);
  }
}

getUser();
```

Fetch — POST, PUT, DELETE

POST Method

```
fetch("https://jsonplaceholder.typicode.com/posts", {
  method: "POST",
  headers: {
    "Content-Type": "application/json"
  },
  body: JSON.stringify({
    title: "My Post",
    body: "Hello world",
    userId: 1
  })
})
  .then(res => res.json())
  .then(data => console.log("Created:", data));
```

Fetch — POST, PUT, DELETE

PUT Method

```
fetch("https://jsonplaceholder.typicode.com/posts/1", {
  method: "PUT",
  headers: { "Content-Type": "application/json" },
  body: JSON.stringify({ title: "Updated Title" })
})
  .then(res => res.json())
  .then(data => console.log("Updated:", data));
```

Fetch — POST, PUT, DELETE

DELETE Method

```
fetch("https://jsonplaceholder.typicode.com/posts/1", {
  method: "DELETE"
})
  .then(res => console.log("Deleted:", res.status));
```

Task

Create dashboard for users to get all and detailed, add, edit, delete on users table with a good design

Thank you