````
## Background Story

It was a humid morning in Pasay City and the logistics office was already humming
with activity—vans loading packages, dispatchers talking on headsets, and the big
screen showing Metro Manila traffic maps 🚚 🚦. Odessa sat at her desk, now a
full-time developer for a fast-growing logistics tech firm. Her boss, Ate Liway,
tapped her keyboard:

“Ods, we need live traffic updates from MMDA and weather data from PAGASA.
Integrate them into our routing dashboard so our drivers avoid gridlock and
typhoons. You got this?”

Odessa took a deep breath. Fetching data from real APIs? This felt like leveling
up from simple date tricks to real-world data magic. She opened her terminal and
typed:
````

npm install node-fetch

~~~

Then in her editor, she wrote:

```js
import fetch from "node-fetch";
~~~

Her first task: call the MMDA traffic API. She sent a request:

```js
const res = await fetch("https://api.mmda.gov.ph/traffic/metro-manila");
const data = await res.json();
console.log(data.congestionLevel);
```

She saw an array of congested roads—EDSA at yellow, Roxas Boulevard at red. Next, she fetched PAGASA’s weather endpoint:

```js
const weatherRes = await fetch(
  "https://api.pagasa.gov.ph/weather/metro-manila"
);
const weather = await weatherRes.json();
console.log(weather.temperature);
```

Now she had temperature, humidity, chance of rain. Odessa combined both into a dashboard: traffic bars and weather badges in a clean table. When Quezon Avenue lit up red at 5 PM, her code popped up an alert: “Heavy traffic ahead—rerouting!” 🤖 🗺️

Her teammates were impressed. The logistics manager, Sir Liway, said, “Odessa, our vans saved two hours today. You bent time and data with code!”

She leaned back, sipping her cold brew, imagining her next feature: predictive ETAs and SMS alerts. But for now, she basked in the glow of live data—fresh from MMDA and PAGASA—bringing her dashboard to life.

Theory & Lecture Content

In this lesson, we learn how to fetch remote data, handle Promises, use async/await, and parse JSON.

1. The Fetch API
2. Working with Promises
3. Async/Await Syntax
4. JSON Parsing and Error Handling

1. The Fetch API

The Fetch API lets you make HTTP requests in JavaScript.
Basic usage:

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

Reference:
https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API/Using_Fetch

2. Working with Promises

A Promise represents a value that may be available now, later, or never.

```
const p = new Promise((resolve, reject) => {
  // async work…
  if (success) resolve(result);
  else reject(error);
});

p.then((value) => console.log(value)).catch((err) => console.error(err));
```

Chaining Promises:

```
fetch(url)
  .then((res) => res.json())
  .then((data) => process(data))
  .catch((err) => console.error(err));
```

Reference:
https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Promise

3. Async/Await Syntax

`async` functions return a Promise. Use `await` to pause until a Promise resolves.

```
async function getData() {
  try {
    const res = await fetch(url);
    if (!res.ok) throw new Error(res.statusText);
    const json = await res.json();
    return json;
  } catch (err) {
    console.error("Error:", err);
    throw err;
  }
}
```

Reference:
https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/async_function

4. JSON Parsing and Error Handling

- Always check `response.ok` before parsing.
- Use `response.json()` to convert response body into JS object.
- Wrap in `try/catch` when using `await`.

Exercises

Exercise 1: Traffic Data Fetcher

Problem Statement
Implement `fetchTrafficData(area)` in `exercise1.js` to fetch MMDA traffic data for a given area code and return the parsed JSON.

TODOs

- Import `node-fetch`.
- Use `fetch` to call `https://api.mmda.gov.ph/traffic/${area}`.
- Check `response.ok`; throw an error if not OK.
- Parse and return `response.json()`.

Starter Code (exercise1.js)

```
// exercise1.js
import fetch from "node-fetch";

export async function fetchTrafficData(area) {
  // TODO: implement fetch logic
}
```

Full Solution (exercise1.js)

```
import fetch from "node-fetch";

export async function fetchTrafficData(area) {
  const url = `https://api.mmda.gov.ph/traffic/${area}`;
  const response = await fetch(url);
  if (!response.ok) {
    throw new Error(`MMDA API error: ${response.status}`);
  }
  const data = await response.json();
  return data;
}
```

Exercise 2: Weather Temperature Extractor

Problem Statement
Write `fetchTemperature(city)` in `exercise2.js`. It should fetch PAGASA weather data (`https://api.pagasa.gov.ph/weather/${city}`) and return only the `temperature` property as a number, using Promise chaining (no async/await).

TODOs

- Import `node-fetch`.
- Use `fetch` and `.then()` to parse JSON.
- Return `data.temperature`.
- Handle HTTP errors.

Starter Code (exercise2.js)

```
// exercise2.js
import fetch from "node-fetch";

export function fetchTemperature(city) {
  // TODO: implement with Promises
}
```

Full Solution (exercise2.js)

```
import fetch from "node-fetch";

export function fetchTemperature(city) {
  const url = `https://api.pagasa.gov.ph/weather/${city}`;
  return fetch(url)
    .then((res) => {
      if (!res.ok) {
        throw new Error(`PAGASA API error: ${res.status}`);
      }
      return res.json();
    })
    .then((data) => data.temperature);
}
```

Exercise 3: Delivery Dashboard (DOM)

Problem Statement
Build a simple dashboard that fetches both traffic and weather data when a button is clicked, then displays the results in the page.

TODOs

- In `index.html`, create a button `#loadBtn` and two divs `#traffic` & `#weather`.
- In `script.js`, add an event listener on `#loadBtn`.
- Use `Promise.all` and `async/await` to fetch both data:
  - `fetchTrafficData("metro-manila")`
  - `fetchTemperature("metro-manila")`
- Display JSON in `#traffic` and temperature in `#weather`.
- Handle and display errors.

Starter Code (index.html)

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8" />
    <title>Delivery Dashboard</title>
  </head>
  <body>
    <h1>🚚 Delivery Dashboard</h1>
    <button id="loadBtn">Load Data</button>
    <div id="traffic"></div>
    <div id="weather"></div>
    <script type="module" src="script.js"></script>
  </body>
</html>
```

Starter Code (script.js)

```
// script.js
import { fetchTrafficData } from "./exercise1.js";
import { fetchTemperature } from "./exercise2.js";

const btn = document.getElementById("loadBtn");
const trafficDiv = document.getElementById("traffic");
const weatherDiv = document.getElementById("weather");

btn.addEventListener("click", async () => {
  // TODO: fetch both, update DOM
});
```

Full Solution (script.js)

```
import { fetchTrafficData } from "./exercise1.js";
import { fetchTemperature } from "./exercise2.js";

const btn = document.getElementById("loadBtn");
const trafficDiv = document.getElementById("traffic");
const weatherDiv = document.getElementById("weather");

btn.addEventListener("click", async () => {
  trafficDiv.textContent = "Loading traffic data…";
  weatherDiv.textContent = "Loading weather…";
  try {
    const [trafficData, temp] = await Promise.all([
      fetchTrafficData("metro-manila"),
      fetchTemperature("metro-manila"),
    ]);
    trafficDiv.textContent = JSON.stringify(trafficData, null, 2);
    weatherDiv.textContent = `Temperature: ${temp}°C`;
  } catch (err) {
    trafficDiv.textContent = `Error: ${err.message}`;
    weatherDiv.textContent = `Error: ${err.message}`;
  }
});
```

Test Cases

exercise1.test.js

```
import { fetchTrafficData } from "./exercise1.js";

describe("fetchTrafficData", () => {
  beforeEach(() => {
    global.fetch = jest.fn();
  });
  afterEach(() => {
    jest.resetAllMocks();
  });

  test("resolves JSON on success", async () => {
    const mockData = { congestionLevel: "Moderate" };
    global.fetch.mockResolvedValue({
      ok: true,
      json: () => Promise.resolve(mockData),
    });

    const data = await fetchTrafficData("metro-manila");
    expect(data).toEqual(mockData);
    expect(global.fetch).toHaveBeenCalledWith(
      "https://api.mmda.gov.ph/traffic/metro-manila"
    );
  });

  test("throws error on bad response", async () => {
    global.fetch.mockResolvedValue({ ok: false, status: 500 });
    await expect(fetchTrafficData("area-51")).rejects.toThrow(
      "MMDA API error: 500"
    );
  });
});
```

exercise2.test.js

```
import { fetchTemperature } from "./exercise2.js";

describe("fetchTemperature", () => {
  beforeEach(() => {
    global.fetch = jest.fn();
  });
  afterEach(() => {
    jest.resetAllMocks();
  });

  test("returns temperature on success", async () => {
    const mock = { temperature: 32 };
    global.fetch.mockResolvedValue({
      ok: true,
      json: () => Promise.resolve(mock),
    });
    const temp = await fetchTemperature("cebu");
    expect(temp).toBe(32);
    expect(global.fetch).toHaveBeenCalledWith(
      "https://api.pagasa.gov.ph/weather/cebu"
    );
  });

  test("throws on HTTP error", async () => {
    global.fetch.mockResolvedValue({ ok: false, status: 404 });
    await expect(fetchTemperature("unknown")).rejects.toThrow(
      "PAGASA API error: 404"
    );
  });
});
```

exercise3.test.js

```
/**
 * @jest-environment jsdom
 */
import fs from "fs";
import path from "path";
import { fetchTrafficData } from "./exercise1.js";
import { fetchTemperature } from "./exercise2.js";

jest.mock("./exercise1.js");
jest.mock("./exercise2.js");

describe("Delivery Dashboard DOM", () => {
  let html;
  beforeAll(() => {
    html = fs.readFileSync(path.resolve(__dirname, "index.html"), "utf8");
    document.documentElement.innerHTML = html;
    // import script after DOM setup
    require("./script.js");
  });

  beforeEach(() => {
    fetchTrafficData.mockClear();
    fetchTemperature.mockClear();
  });

  test("loads and displays data on button click", async () => {
    const mockTraffic = { roads: ["EDSA", "C5"] };
    fetchTrafficData.mockResolvedValue(mockTraffic);
    fetchTemperature.mockResolvedValue(28);

    const btn = document.getElementById("loadBtn");
    btn.click();
    // wait for microtasks
    await Promise.resolve();
    const trafficDiv = document.getElementById("traffic");
    const weatherDiv = document.getElementById("weather");

    expect(trafficDiv.textContent).toContain(
      JSON.stringify(mockTraffic, null, 2)
    );
    expect(weatherDiv.textContent).toBe("Temperature: 28°C");
  });
});
```

Closing Story

As the dashboard sprang to life on her screen—live MMDA traffic maps and PAGASA weather updates—Odessa felt the thrill of real-time data pulsing through her code. Drivers avoided bottlenecks on EDSA, and dispatchers rerouted vans before the rain poured in. With Promises and async/await, she had made distant APIs feel as close as her own database.

Ate Liway pinged her on Slack with a thumbs-up emoji 👍: “This is gold, Ods! Next up: chart libraries and real-time sockets.”

Odessa grinned. Fetching data was only the beginning—visualizing trends and building live maps would be her next frontier. Her code had given life to data, and now the future looked bright, one dashboard at a time. 🚀