**1) What is app.use()?**

Registers **middleware** on your Express app.

* Runs **in the order you define**.
* Can be **global** or **path-scoped**.

js

CopyEdit

app.use(express.json()); // global

app.use('/api', authMiddleware); // only for paths starting with /api

# 2) What if you don’t send a response in a route handler?

If no handler sends a response and no error is thrown, Express eventually returns a **default 404**. If you forget to send a response and you don’t call next(), the request may **hang**.

# 3) What happens when you call res.send()?

**res.send()**

* Sends the response body **and ends the response**.
* Automatically sets Content-Type (JSON, text, etc.).
* **After calling it, you can’t send more data or change headers**.
* Calling it **twice** → Error: Can't set headers after they are sent.

import express from "express";

const app = express();

// Example 1: Normal usage

app.get("/ok", (req, res) => {

res.send({ message: "Hello World" }); // ✅ Sends JSON + ends response

});

// Example 2: Calling res.send() twice

app.get("/error", (req, res) => {

res.send("First response sent"); // ✅ sends and ends

// ❌ This line will cause an error:

res.send("Second response"); // Error: Can't set headers after they are sent.

});

// Example 3: Setting status before sending

app.get("/status", (req, res) => {

res.status(201).send({ ok: true }); // Sets status 201 + sends JSON

});

// Example 4: Using res.json() shorthand

app.get("/json", (req, res) => {

res.json({ name: "Likan", skill: "Node.js" }); // JSON + Content-Type

});

// Example 5: sendStatus (status only, no body)

app.get("/no-content", (req, res) => {

res.sendStatus(204); // Sends status 204, no body allowed

});

app.listen(3000, () => console.log("Server running on port 3000"));

### **What Happens Internally**

1. **res.send()** → Writes headers + body → ends the HTTP response.
2. After it ends:
   * Any further res.send() / res.write() / res.set() **will throw an error**.
   * Reason: HTTP protocol only allows headers once, before the body is sent.
3. **res.json()** → Just calls res.send() internally with Content-Type: application/json.
4. **res.sendStatus()** → Sets status code + ends the response (no body if status is like 204).

# 4) How many middleware functions can one route have?

**As many as you want.** They run **left-to-right**.

js

CopyEdit

app.get('/users', mw1, mw2, mw3, handler);

# 5) If you send the response in the first route handler, will the second run?

**No.** Once a handler sends the response **and doesn’t call next()**, Express stops for that request.  
If the first handler **calls next() before sending**, the next one runs and can send.

6) Can we call next() before/after res.send()?

## **Case A —** next() **before** res.send() **✅**

* next() passes control to the next middleware/route handler.
* A later handler can safely call res.send() **once**.
* After res.send(), the response is **finished**.

**Code Example:**

js

CopyEdit

import express from "express";

const app = express();

// Middleware 1: runs first

app.use((req, res, next) => {

console.log("Middleware 1 executed");

next(); // ✅ Passing control before sending a response

});

// Middleware 2: final responder

app.get("/caseA", (req, res) => {

res.send("Hello from Case A — sent in second handler");

});

app.listen(3000, () => console.log("Server running on port 3000"));

**Flow:**  
Middleware 1 → next() → Middleware 2 → res.send() (ends response).

## **Case B —** next() **after** res.send() **❌**

* res.send() sends **headers + body** and **ends the response**.
* Calling next() afterward keeps the chain going; if any later middleware tries to write headers/body, you’ll get:  
  **Error: Can't set headers after they are sent.**
* Even if later middleware doesn’t write, it’s **confusing** and can trigger unwanted side effects.

**Code Example:**

js

CopyEdit

import express from "express";

const app = express();

// Route handler that sends response and then calls next (BAD)

app.get("/caseB", (req, res, next) => {

res.send("Response sent in Case B"); // Ends response

next(); // ❌ Unsafe — response already finished

});

// This middleware still runs after next(), but the response is closed

app.use((req, res) => {

console.log("This still runs, but response is already sent");

// Any res.send() or res.set() here will cause an error

// res.send("Another response"); // ❌ Would throw

});

app.listen(3001, () => console.log("Server running on port 3001"));

**Flow:**  
Route handler → res.send() (ends) → next() → Another middleware (but can’t send anything now).

7) What happens if no one sends a response and everyone just calls next()?

## **What Happens**

* In Express, each incoming request goes through the **middleware stack** in the order you defined it.
* If **every middleware/route handler** calls next() **without** sending a response (res.send(), res.json(), etc.), the request will keep moving forward in the stack.
* When Express reaches the **end of the stack** and finds **no more matching middleware**, it triggers its **built-in final handler**.
* The built-in final handler automatically sends a **404 Not Found** response.
* If somewhere in the chain a handler calls next(err), Express **skips all normal middleware** and jumps directly to **error-handling middleware** (middleware with **4 parameters**: err, req, res, next).

## **Normal Flow (No Response Sent) → 404**

js

CopyEdit

import express from "express";

const app = express();

app.use((req, res, next) => {

console.log("Middleware 1");

next(); // Passing along without sending

});

app.use((req, res, next) => {

console.log("Middleware 2");

next(); // Still no response sent

});

// No route handler here → Express built-in final handler kicks in

// Sends 404 Not Found

app.listen(3000, () => console.log("Server running on port 3000"));

**Result:**  
If you hit /anything, you’ll see a **404 Not Found** because no middleware sent a response.

## **Error Flow (**next(err)**) → 500**

js

CopyEdit

app.use((req, res, next) => {

next(new Error("Something broke")); // Jump to error middleware

});

// Error-handling middleware (MUST have 4 args)

app.use((err, req, res, next) => {

console.error("Error:", err.message);

res.status(500).send("Something broke");

});

**Result:**  
If any middleware calls next(err), Express **skips** the rest of the normal middleware and **only runs** error-handling middleware.

## **Key Points to Remember**

* **If no one sends a response** → Express sends 404 Not Found automatically.
* **If someone calls next(err)** → Express jumps directly to error-handlers.
* Always **end a request** by either:
  + Sending a response (res.send(), res.json(), res.end()), **or**
  + Passing to error middleware with next(err).

# 8) What is an HTTP response status code & how to set it manually?

A **3-digit number** telling the client the result (200 OK, 404 Not Found, 500 Server Error, etc.).  
Set it with:

js

CopyEdit

res.status(201).send('Created');

res.sendStatus(400); // sets 400 and ends

9) app.use() vs app.all()

## **Difference #1 — Path Matching**

app.use() → **prefix match** (runs for anything under that path)  
app.all() → **exact path match** (only that route, unless you use wildcards)

import express from "express";

const app = express();

// app.use — prefix match

app.use("/api", (req, res, next) => {

console.log("app.use('/api') matched:", req.method, req.url);

next();

});

// app.all — exact path match

app.all("/api", (req, res) => {

res.send("app.all('/api') matched exactly");

});

app.get("/api/users", (req, res) => {

res.send("GET /api/users — runs after app.use but NOT app.all('/api')");

});

app.listen(3000, () => console.log("Running on 3000"));

**Try:**

* GET /api → runs app.use('/api') **and** app.all('/api')
* GET /api/users → runs app.use('/api') but **not** app.all('/api')

## **Difference #2 — Purpose**

app.use() → middleware for **all methods** (often no response, just passes control)  
app.all() → final handler for **all methods on that route**

js

CopyEdit

// app.use for logging

app.use((req, res, next) => {

console.log(`[LOG] ${req.method} ${req.url}`);

next(); // doesn't send response

});

 This middleware runs **for every request** (any path, any method).

 Purpose: Just logs request info.

 It **does not** send a response — it calls next() so the request moves on to the next middleware/route handler.

// app.all to block a path regardless of method

app.all("/maintenance", (req, res) => {

res.status(503).send("Service down");

});

 This is a **route handler** for /maintenance.

 It runs **for all HTTP methods** (GET, POST, PUT, etc.) but **only** on /maintenance (exact match).

 Purpose: Send the same “Service down” response for any method hitting that path.

 Once it sends the response, the request ends — it doesn’t call next().

## **Difference #3 — Flow Control**

app.use() can run **before** other routes (acts globally or on prefixes)  
app.all() **is a route** — if matched and sends a response, request ends there.

js

CopyEdit

// Prefix middleware

app.use("/secure", (req, res, next) => {

if (!req.query.token) return res.status(401).send("No token");

next();

});

// Route for all methods

app.all("/secure", (req, res) => {

res.send("Secure root accessed");

});

app.get("/secure/data", (req, res) => {

res.send("Secure data route");

});

**Try:**

* /secure?token=abc → matches middleware then exact /secure route.
* /secure/data?token=abc → matches middleware then /secure/data route.

Summary

 **app.use(path, middleware)** → Runs for **all HTTP methods** on any route that starts with path (prefix match), mainly for middleware logic.

 **app.all(path, handler)** → Runs for **all HTTP methods** on the **exact path** (no prefix), mainly for handling that route directly.

* **Prefix match** → The path you give is just the **starting part** of the URL.  
  If it matches, Express runs it **for everything under it**.  
  Example:

js

CopyEdit

app.use("/api", mw);

Runs for:

* + /api
  + /api/
  + /api/users
  + /api/v1/products ✅  
    (Because they all **start with** /api)
* **Exact match** → The path must match **exactly** (except a trailing slash) for it to run.  
  Example:

js

CopyEdit

app.all("/api", handler);

Runs for:

* + /api
  + /api/ ✅  
    ❌ Not /api/users  
    ❌ Not /api/v1