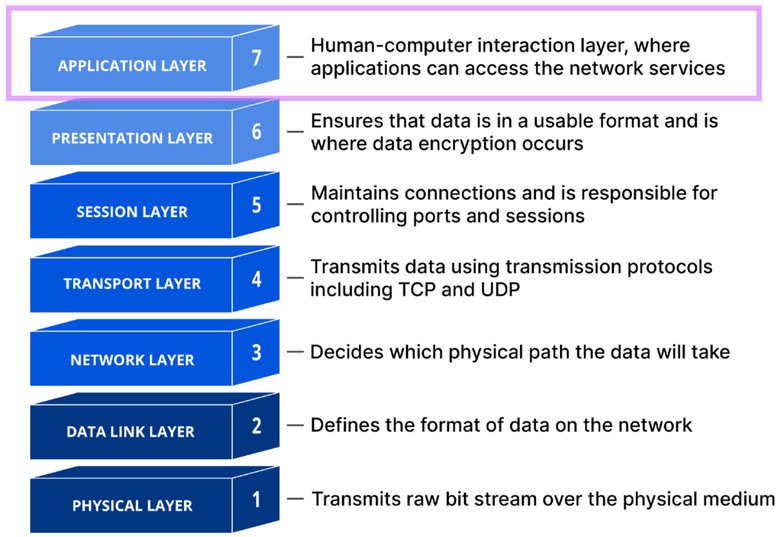
Express.js 

REST APIs

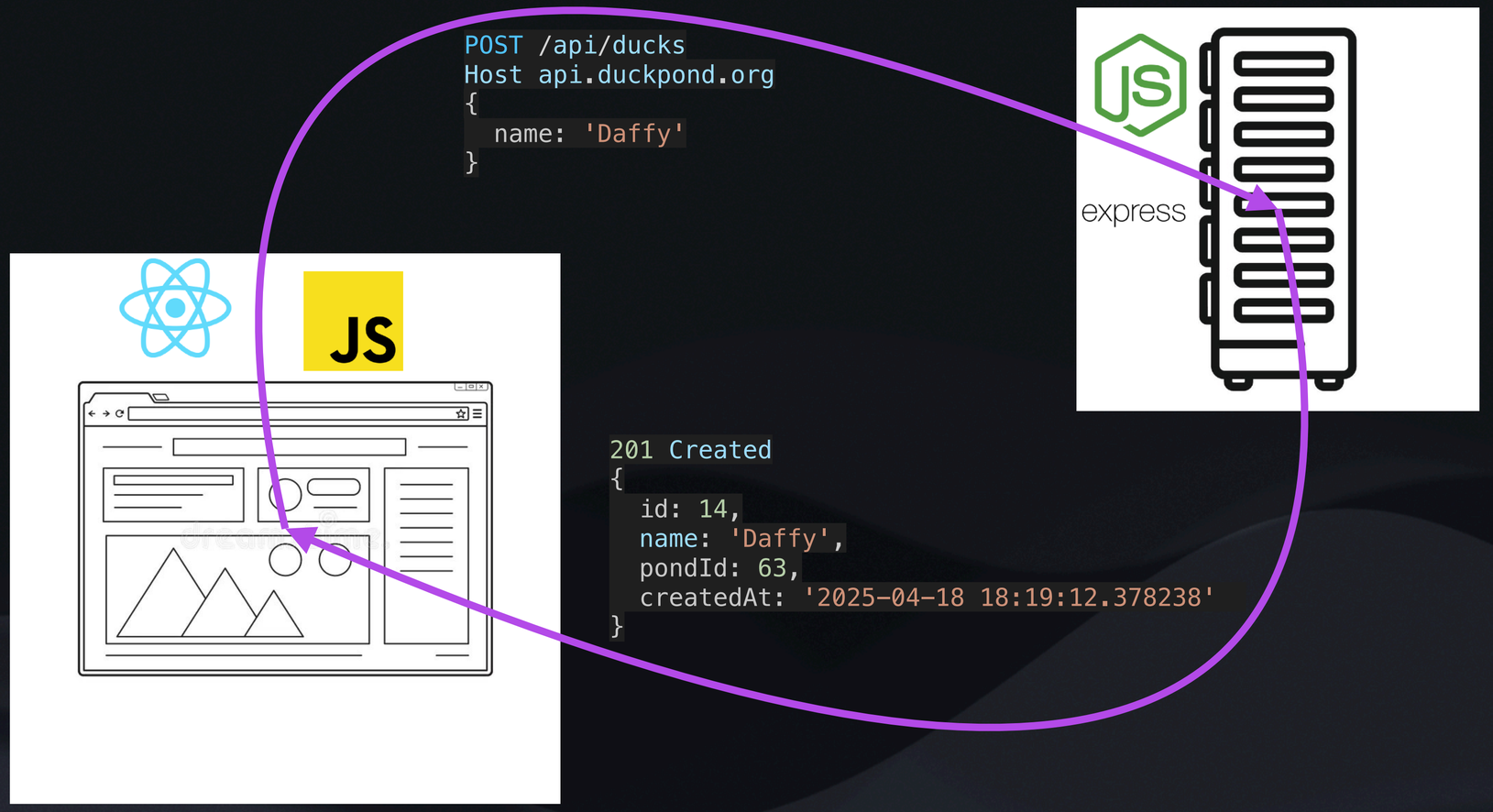
What is the internet?

What is the internet?



- The internet is a series of connected computers.
- The open systems interconnection (OSI) model describes different *layers of abstraction* that make the internet possible.
- We're mostly working with the very top layer, the Application layer, of which HTTP is a part.

Request and Response



Request Method Refresher

Method	Meaning
GET /puppies	Give me all the puppies
GET /puppies/2	Give me the puppy with id 2
POST /puppies	Here’s a new puppy
DELETE /puppies/4	Remove puppy with id 4
PATCH /puppies/7	Update puppy with id 7
PUT /puppies/9	Replace all data about puppy 9 with this data

Response Status Codes

Status Code	What the server says back to the client
200	OK! Your request was successful
201	You asked me to create something and I did
304	You asked for this recently and it hasn’t changed
400	You good, bro? That request didn’t make sense to me
404	That thing? It doesn’t exist. Couldn’t find it
500	I’m having a bad day. Come back later when I’m not on fire

NOTE: These are just a few of the MANY status codes. MDN is your friend

Response Status Codes

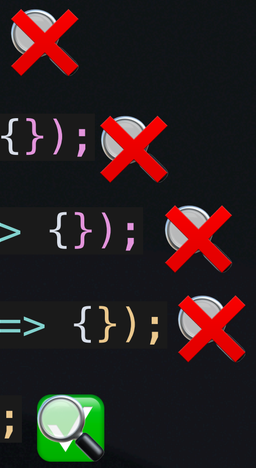
418 I'm a teapot

The HTTP `418 I'm a teapot` status response code indicates that the server refuses to brew coffee because it is, permanently, a teapot. A combined coffee/tea pot that is temporarily out of coffee should instead return `503`. This error is a reference to Hyper Text Coffee Pot Control Protocol defined in April Fools' jokes in 1998 and 2014.

Some websites use this response for requests they do not wish to handle, such as automated queries.

Express.js

```
app.get(“/api/ducks", (req, res) => {});
app.get(“/api/ducks/:id", (req, res) => {});
app.patch(“/api/ducks/:id", (req, res) => {});
app.delete(“/api/ducks/:id", (req, res) => {});
app.post("/api/ducks", (req, res) => {});
```



```
POST /api/ducks
Host api.duckpond.org
{
  name: 'Daffy'
}
```

- Express allows us to define a series of routes
- Each route has a method and a path
- Each route has a callback function that runs when it finds a match

Express.js: Hello World

```
const express = require("express");

const app = express();

app.get("/", (req, res) => {
  res.send("Hello World");
});

app.listen(8080, () => {
  console.log("The server is listening on port 8080!");
});
```

- Express is an NPM library that must be installed and imported
  - Note that we're using require()
- Initialize the Express application
- Define a route
- Send a response
- Start the server on a port

Express.js: Response

```
app.get("/", (req, res) => {
  res.send("Hello World");
});
```

✅ `res.send` sends a response

```
app.post("/", (req, res) => {
  res.sendStatus(201);
});
```

✅ `res.sendStatus` sends only a status code

```
app.get("/ducks", (req, res) => {
  res.status(400).send("You good, bro?");
});
```

✅ `res.status().send()` can be “chained" to customize status

```
app.get("/ducks", (req, res) => {
  res.send(["Donald", "Daffy", "Scrooge"]);
});
```

✅ We can send any kind of data – objects and arrays will be encoded to JSON

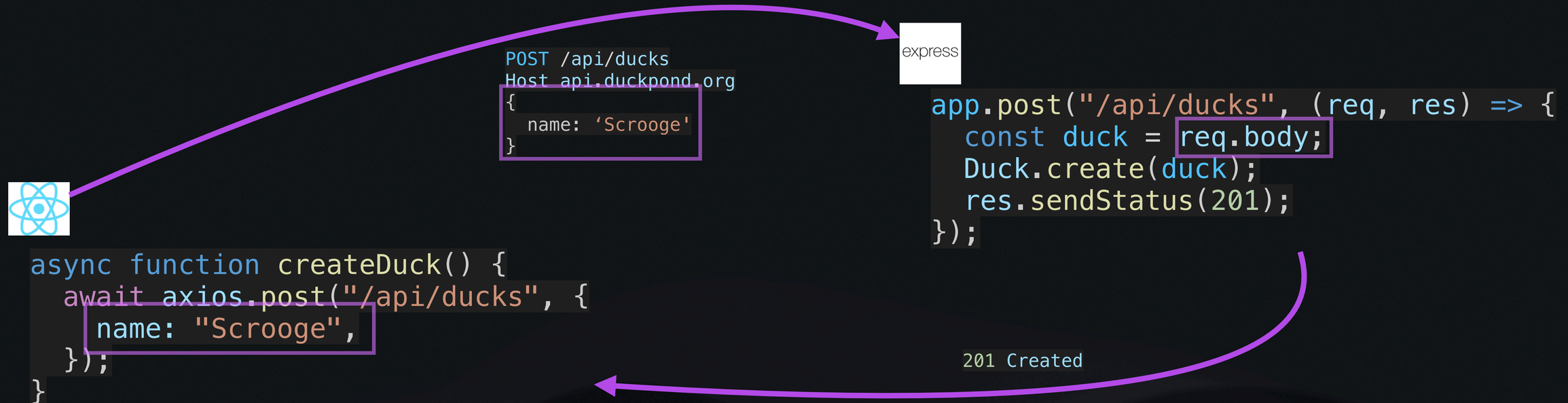
```
app.get("/ponds", (req, res) => {
  return ["Donald", "Daffy", "Scrooge"];
});
```

❌ Each route must send a response – return doesn’t count

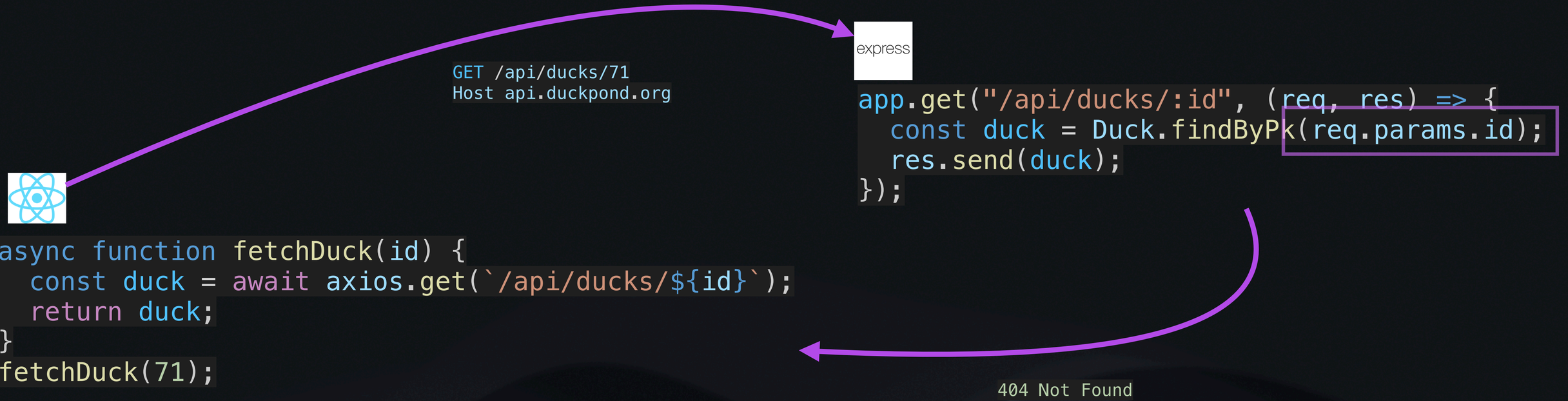
```
app.get("/ponds", (req, res) => {
  res.sendStatus(202);
  res.send(["Donald", "Daffy", "Scrooge"]);
});
```

❌ Each route must only send ONE response – sending two will confused the client

Express.js: Request Body



Express.js: Request Params

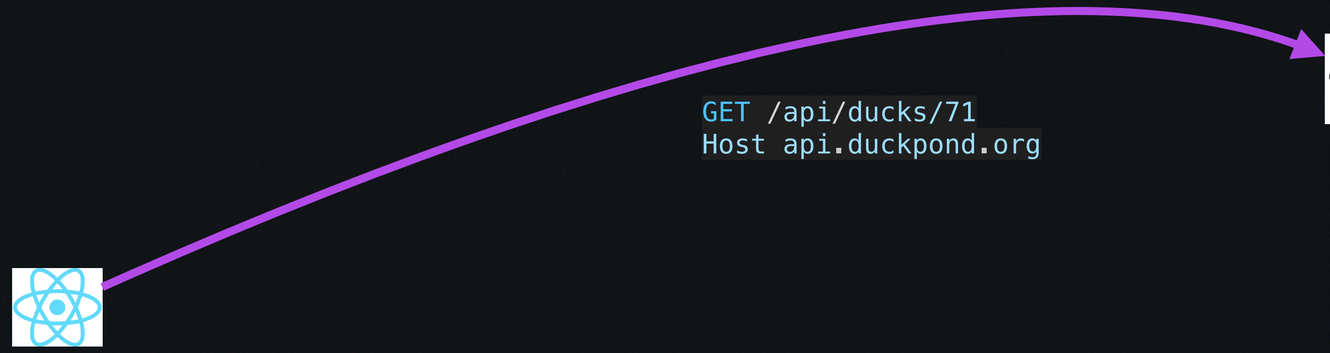


🐛 Common bug! Check the type of req.params.id

By default, it will be a string 🧵

```
req.params.id === “71”
```

Express.js: Request Params



```
GET /api/ducks/71
Host api.duckpond.org
```

```
async function fetchDuck(id) {
  const duck = await axios.get(`/api/ducks/${id}`);
  return duck;
}
fetchDuck(71);
```

```
app.get("/api/ducks/:id", (req, res) => {
  const id = Number(req.params.id);
  const duck = Duck.findByPk(id);
  res.send(duck);
});
```

```
200 OK
{
  id: 71,
  name: 'Darkwing',
  pondId: 91,
  createdAt: '2025-05-29 12:21:12.341238'
}
```

req.body = the request body

req.params = the URL params

Express.js: Middleware

```
(req, res, next) => {}
```

next() = “move on to the next matching route”

app.use() = “use this middleware”

Using middleware

Express is a routing and middleware web framework that has minimal functionality of its own: An Express application is essentially a series of middleware function calls.

Middleware functions are functions that have access to the request object (`req`), the response object (`res`), and the next middleware function in the application’s request-response cycle. The next middleware function is commonly denoted by a variable named `next` .

Middleware functions can perform the following tasks:

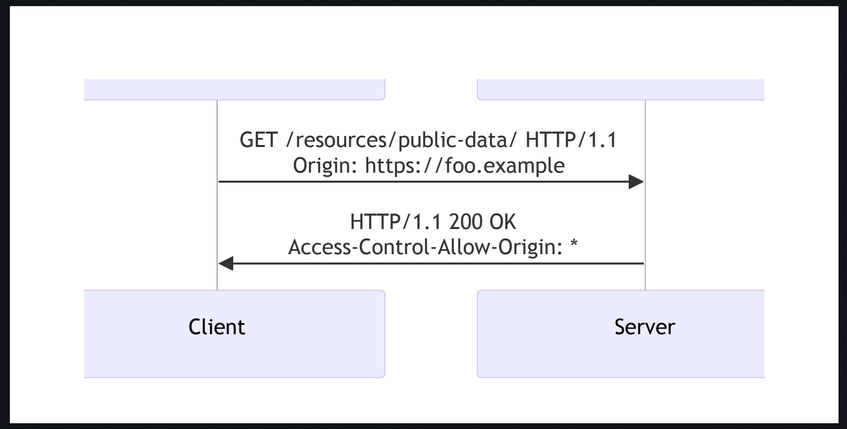
- Execute any code.
- Make changes to the request and the response objects.
- End the request-response cycle.
- Call the next middleware function in the stack.

If the current middleware function does not end the request-response cycle, it must call `next()` to pass control to the next middleware function. Otherwise, the request will be left hanging.

https://expressjs.com/en/guide/using-middleware.html

```
app.use(
  // CORS: Allow requests only from localhost:3000
  cors({
    origin: "http://localhost:3000",
    credentials: true,
  })
);
app.use(express.json()); // body parser middleware
app.use(morgan("dev")); // logging middleware
app.use(express.static(path.join(__dirname, "public"))); // serve static files from public folder
app.use("/api", apiRouter); // mount api router
```

Express.js: CORS




```
const cors = require("cors");

app.use(cors({}));
```

- Cross-Origin Resource Sharing (CORS) protects servers from requests that come from untrusted sources
- Access-Control-Allow-Origin is a response header that tells the client who is allowed to make requests
- “Access-Control-Allow-Origin: *” means everyone is invited

Express.js: Structure

`app.js` (root of our project)

```
const apiRouter = require("./api");

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

app.listen(PORT, () => {
  console.log(`🚀 Server is running on port ${PORT}`);
});
```

`api/index.js`

```
const router = express.Router();
const ducksRouter = require("./ducks");
const pondsRouter = require("./ponds");

router.use("/ducks", ducksRouter);
router.use("/ponds", pondsRouter);

module.exports = router;
```

- We don’t want to write all of our API routes in one file
- Let’s split it up by resources!

`api/ducks.js`

```
const router = express.Router();

// GET /api/ducks
router.get("/", (req, res) => {
  const ducks = Duck.findAll();
  res.json(ducks);
});

module.exports = router;
```

Express.js: Tools

www.postman.com

ngrok

ngrok.com

- **Postman** lets you make customized requests to any server
  - SUPER useful when you want to test out a route before building UI
- **Ngrok** opens up a publicly accessible URL that points to your local machine
  - Useful when you want to show off a work-in-progress API

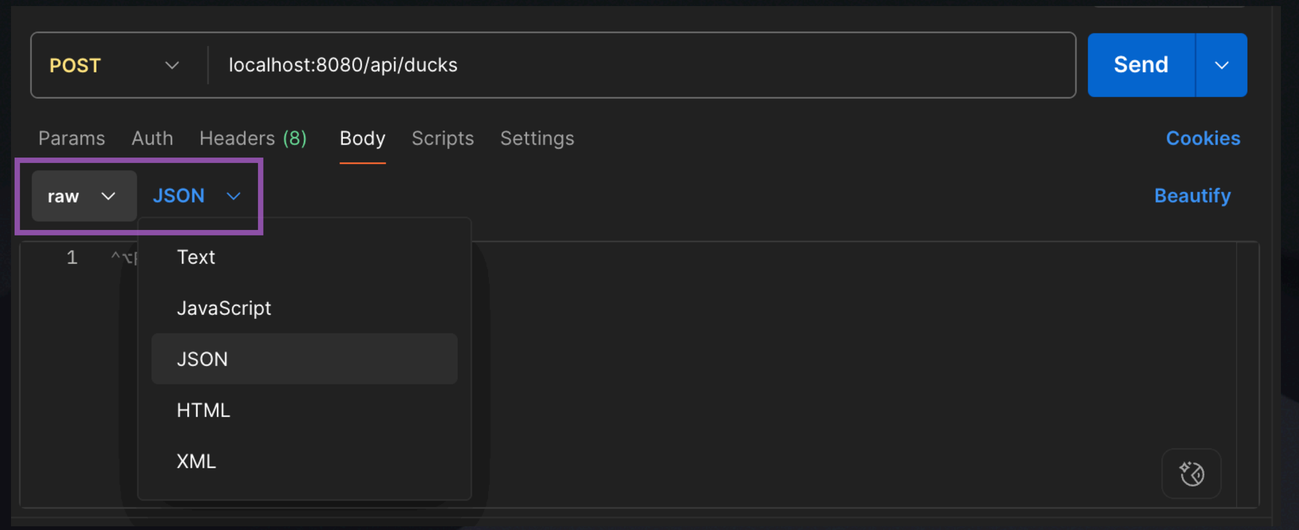
Express.js: Tools

POSTMAN

www.postman.com

Pay attention to the format of the request body

Usually, you want raw/JSON



Express.js Practice