Sponsored by

Mobi

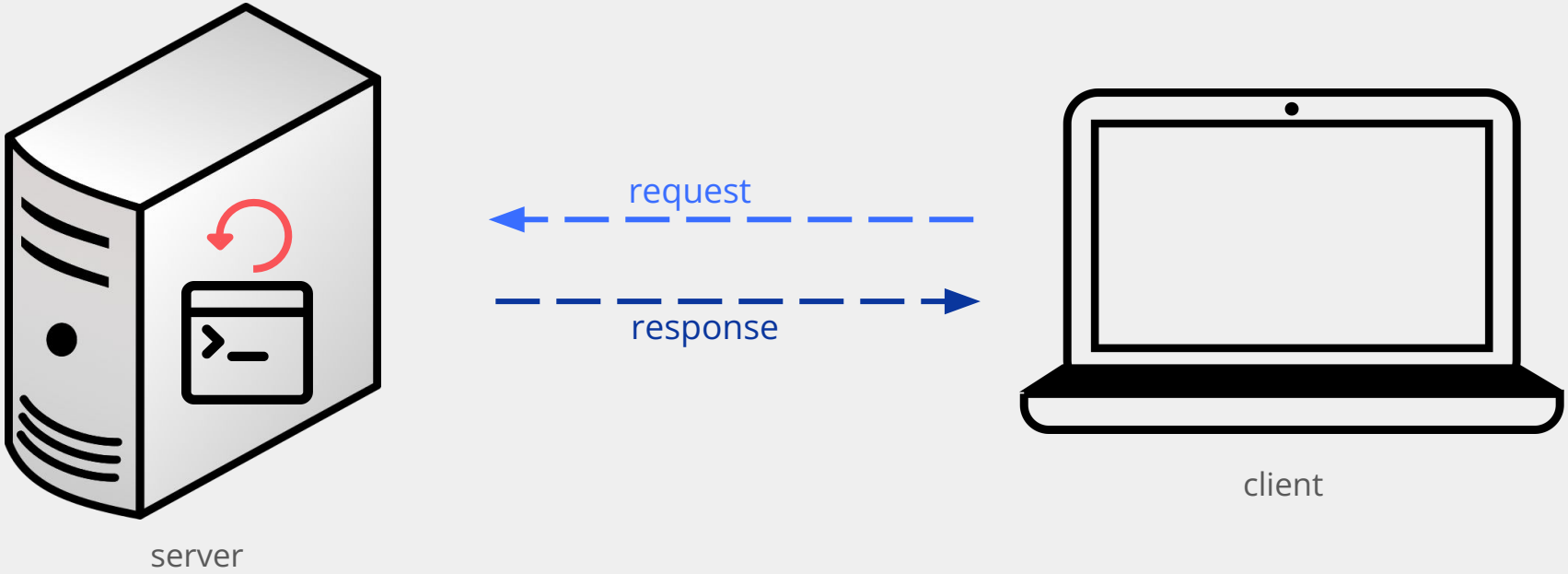
Servers and Node

Joyce Yoon and Jay Hilton

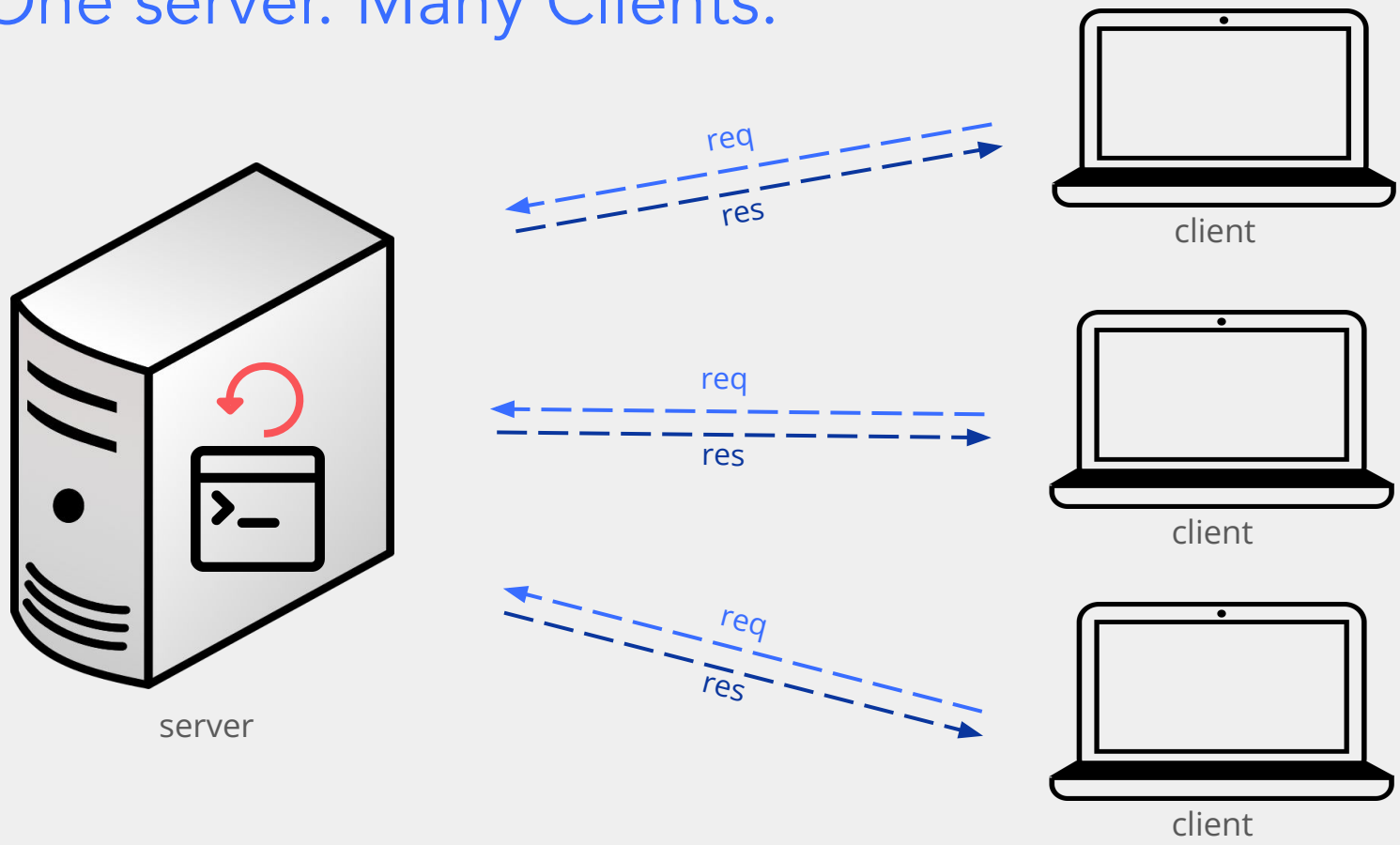
What is a server?

What is a server?

What is a server?

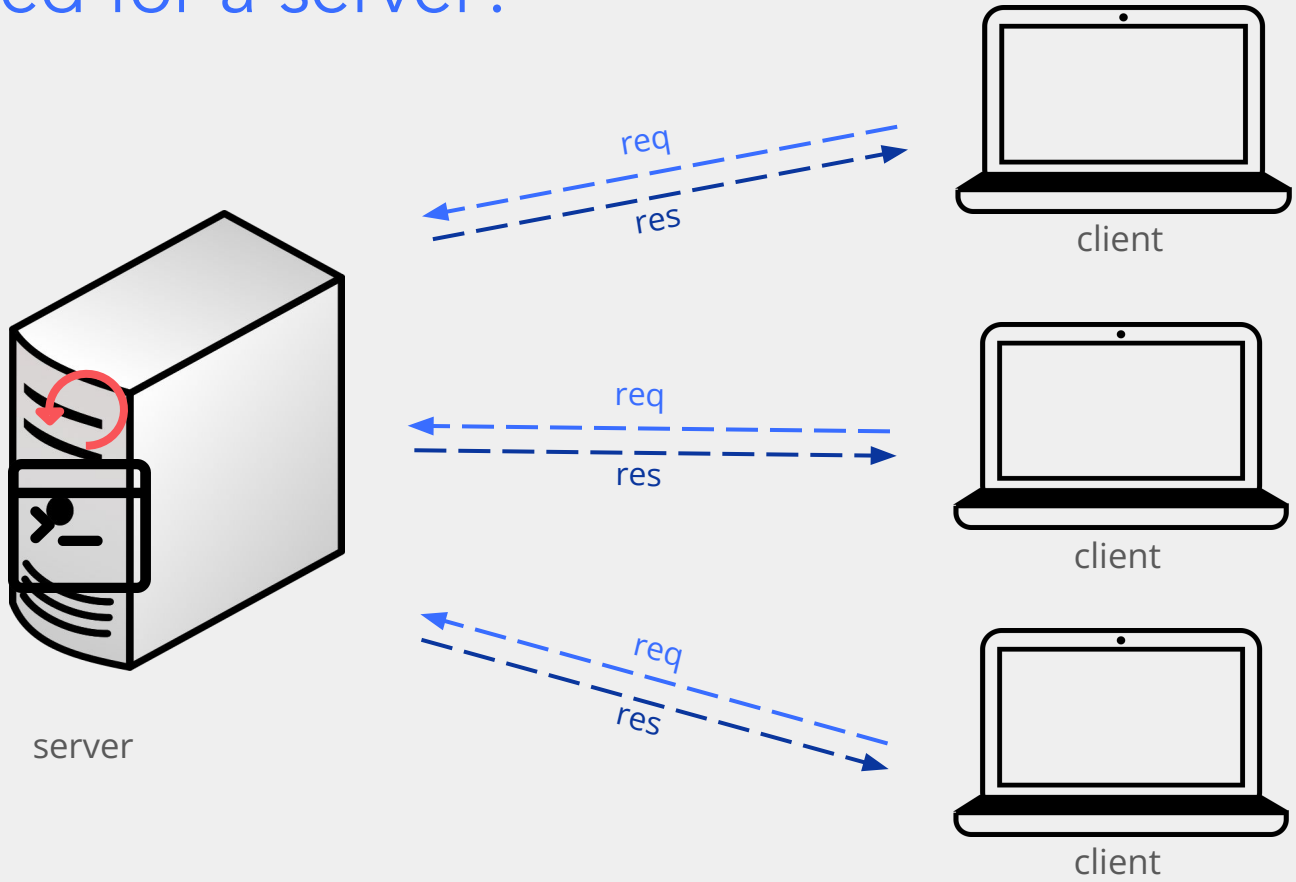


One server. Many Clients.



What is the need for a server?

- File access
- Centralization
- Security



Processes & Ports

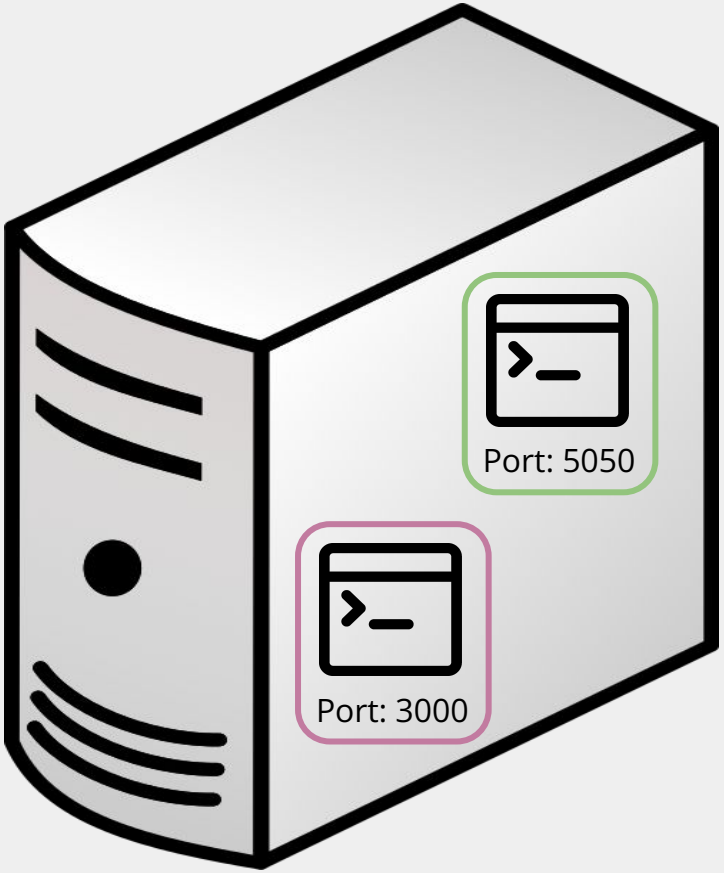
A server **binds** to a **port** on a computer.

`http://example.com:5050`

`http://example.com:3000`

`protocol://domain:port`

Why do we not need to specify ports on most websites?



Two servers on one computer

Processes & Ports

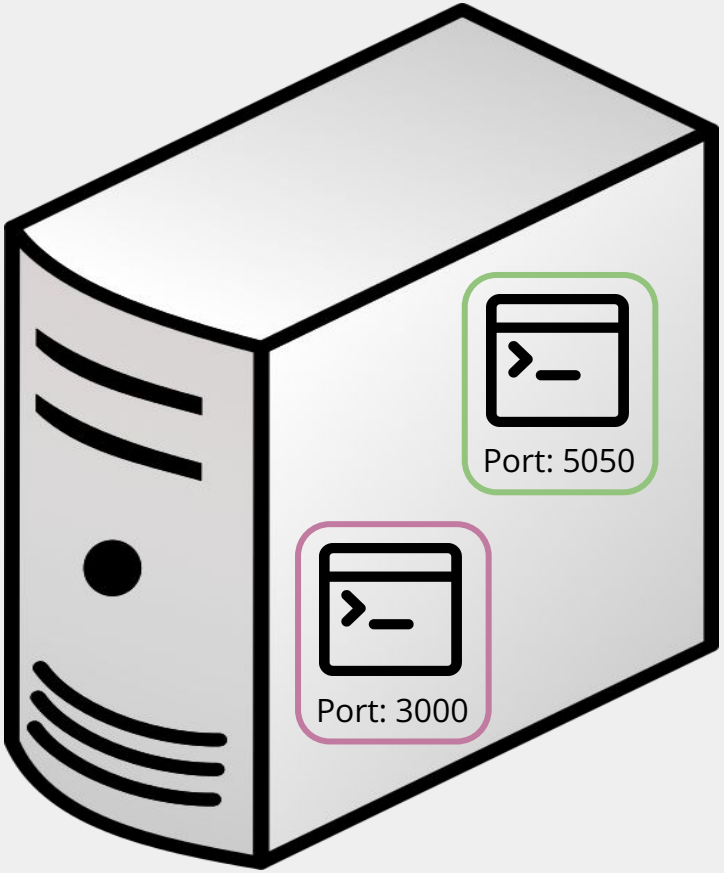
Why do we not need to specify ports on most websites?

HTTPS Websites - 443

HTTP Websites - 80

Minecraft- 25565

Apple Airplay - 5000



Two servers on one computer

Using your own machine as a server

- Every computer can run server code!

- Your own computer has a special domain: **localhost**
  - http://localhost:3000 → connects to a server on port 3000.

How do we write a server?

How do we write a server?

Many options!

python

java

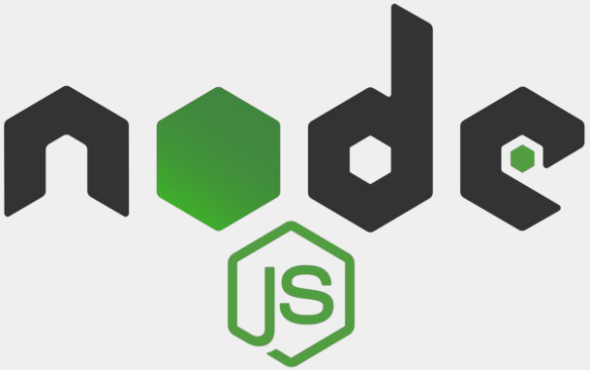
javascript

python

... and more!

What do we teach in this class?

We use **express.js**, but we also need to use a tool to **run** javascript code.



Node **runs** Javascript on your machine

Express allows you to **write your backend** in Javascript

Node Package Manager (npm)

*"**npm** install"*

*"**npm** run hotloader"*

Importing External Code

Every Javascript (Node) project has some necessary information.

Importing External Code

Every Javascript (Node) project has some necessary information.

```
{
  "name": "class-example",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "",
  "license": "ISC",
  "dependencies": {
    "express": "^4.17.1"
  }
}
```

`package.json` holds project **metadata**.

Importing External Code

Every Javascript (Node) project has some necessary information.

```
{
  "name": "class-example",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "",
  "license": "ISC",
  "dependencies": {
    "express": "^4.17.1"
  }
}
```

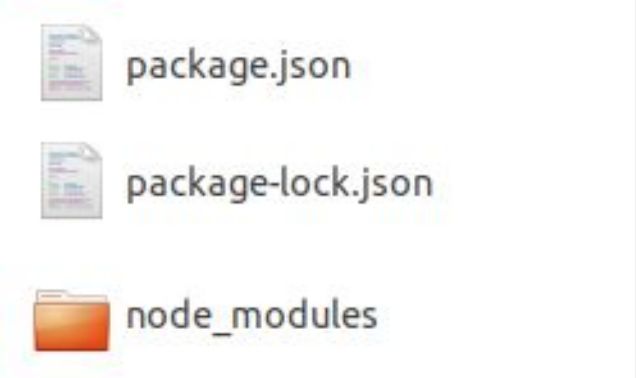
`package.json` holds project **metadata**.

Importing External Code

Every Javascript (Node) project has some necessary information.

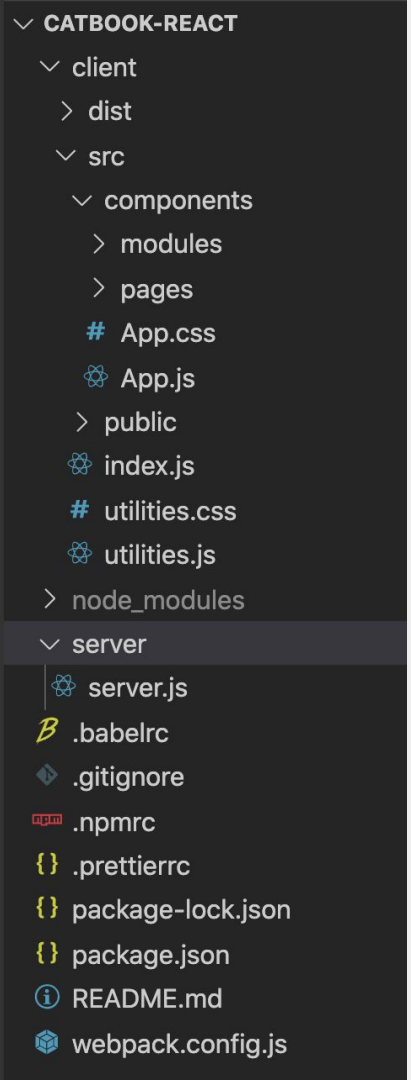
```
{
  "name": "class-example",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "",
  "license": "ISC",
  "dependencies": {
    "express": "^4.17.1"
  }
}
```

npm install →



`package.json` holds project **metadata**.

`node_modules` contains **imported code**



Understanding our codebase

/client

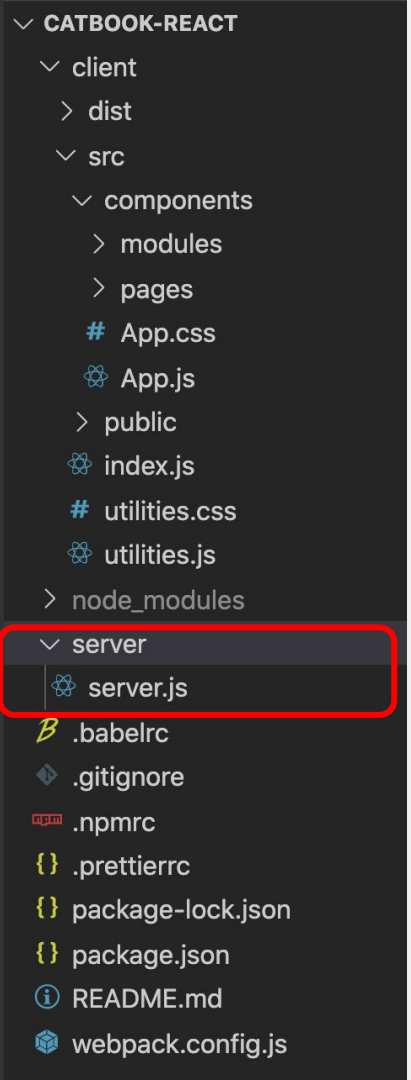
- Contains all of our React code, components, pages, utilities, etc. (Front end)

/server

- Contain all of our backend code

Other files

- Set up by staff to configure React app. Feel free to ask staff about any of these files



Understanding our codebase

/client

- Contains all of our React code, components, pages, utilities, etc. (Front end)

/server

- Contain all of our backend code

Other files

- Set up by staff to configure React app. Feel free to ask staff about any of these files

Creating our first API endpoint

// server.js

```
18  // create a new express server
19  const app = express();
20
21  app.get("/api/test", (req, res) => {
22    res.send({ message: "Wow I made my first API!" });
23  });
24
```

Creating our first API endpoint

// server.js

```
18  // create a new express server
19  const app = express();
20
21  app.get("/api/test", (req, res) => {
22    res.send({ message: "Wow I made my first API!" });
23  });
24
```

HTTP Method

Creating our first API endpoint

// server.js

```
18  // create a new express server
19  const app = express();
20
21  app.get("/api/test", (req, res) => {
22    res.send({ message: "Wow I made my first API!" });
23  });
24
```

Express Route

Creating our first API endpoint

// server.js

```
18  // create a new express server
19  const app = express();
20
21  app.get("/api/test", (req, res) => {
22    res.send({ message: "Wow I made my first API!" });
23  });
24
```

Function handler

Creating our first API endpoint

// server.js

```
18  // create a new express server
19  const app = express();
20
21  app.get("/api/test", (req, res) => {
22    res.send({ message: "Wow I made my first API!" });
23  });
24
```

Function handler

`req` is the incoming request

`res` is your server's response

Creating our first API endpoint

// server.js

```
18  // create a new express server
19  const app = express();
20
21  app.get("/api/test", (req, res) => {
22    res.send({ message: "Wow I made my first API!" });
23  });
24
```

Server Response

Similar to a return statement

Creating our first API endpoint

// server.js

```
18  // create a new express server
19  const app = express();
20
21  app.get("/api/test", (req, res) => {
22    res.send({ message: "Wow I made my first API!" });
23  });
24
```

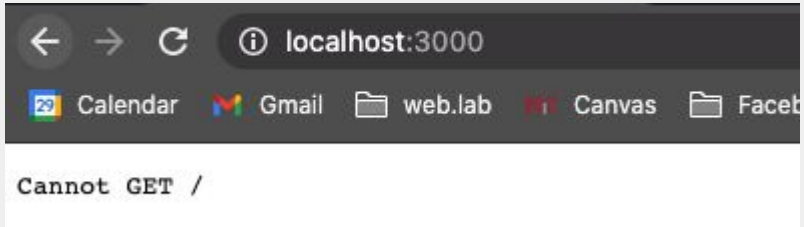
Server Response

Similar to a return statement

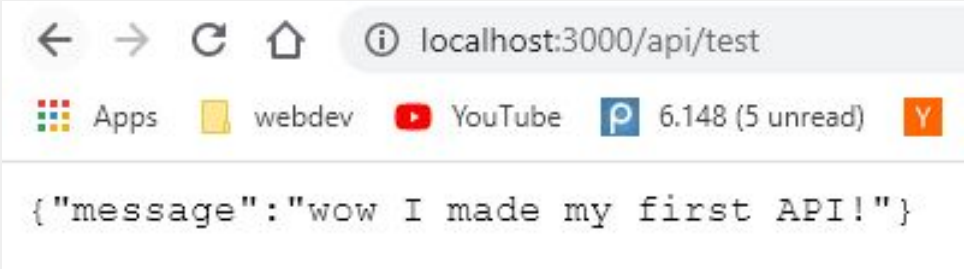
HTTP Method | Express Route | Function handler

Serving React Files

Navigate to `http://localhost:3000` in your browser.



Navigate to `http://localhost:3000/api/test` in your browser.



Serving React Files

Order matters!

You must define your endpoints from **most specific** to **least specific.**

1.

```
app.get("/api/test", (req, res) => {
  res.send("Endpoint A");
});

app.get("/api/*", (req, res) => {
  res.send("Endpoint B");
});
```

2.

```
app.get("/api/*", (req, res) => {
  res.send("Endpoint B");
});

app.get("/api/test", (req, res) => {
  res.send("Endpoint A");
});
```

Serving React Files

Order matters!

You must define your endpoints from **most specific** to **least specific.**

```
app.get("/api/test", (req, res) => {
  res.send("Endpoint A");
});

app.get("/api/*", (req, res) => {
  res.send("Endpoint B");
});
```

good!

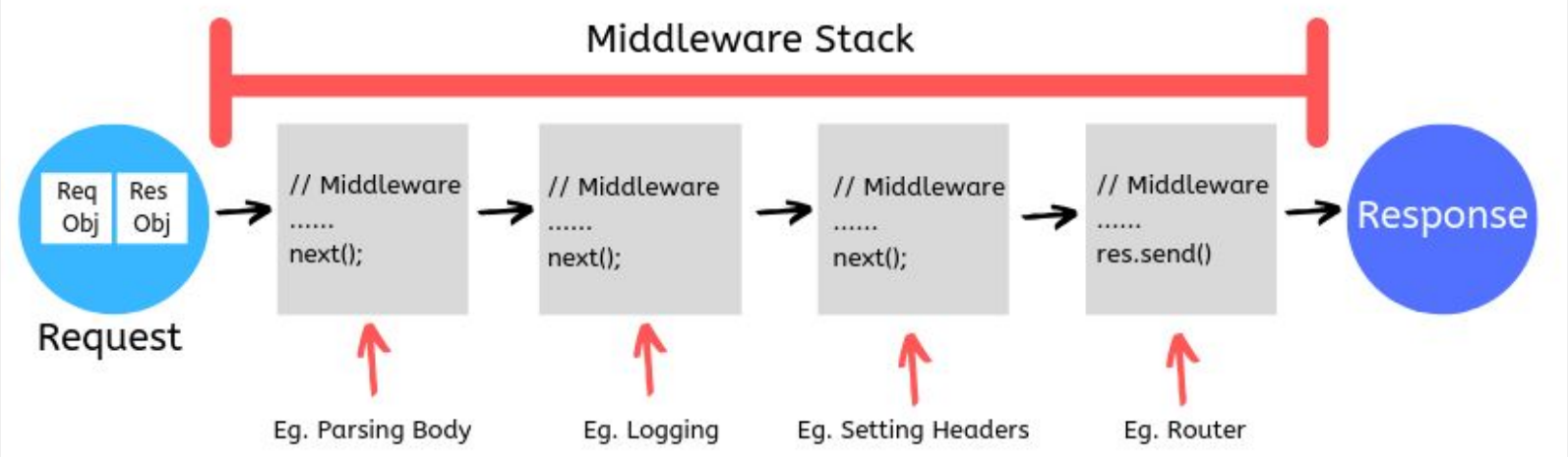
```
app.get("/api/*", (req, res) => {
  res.send("Endpoint B");
});

app.get("/api/test", (req, res) => {
  res.send("Endpoint A");
});
```

bad :(

Middleware

- Run code in between receiving a request and running endpoint code
- Middlewares are like workers in an assembly line
  - They can pass the request and modify/return a response
- Middlewares are called in order of definition



Adding Middleware

Express allows you to add **middleware.**

Register middleware by calling **`app.use(...)`**

Express JSON Parser Middleware will be needed for the next workshop!

```
19  // create a new express server
20  const app = express();
21
22  // allow us to make post requests
23  app.use(express.json());
24
```

Middlewares

- Middlewares are functions that run when a request is received
  - Can run before route handler
  - Route handlers are middleware too!
- Middleware can:
  - Modify requests (adding required properties)
  - Preprocess requests
    - Parsing requests as JSON
  - Serve static content
  - Handle errors
  - And more!
- We can definite middleware using Express's app.use(middlewareFunction)
  - We will see an example in a few slides!

Middleware Examples

- app.use takes an optional path and a “middleware object”
  - “middleware object” is often a callback function

```
// A middleware function for paths prefixed with "/time"
app.use("/time", (req, res, next) => {
  console.log('Time:', Date.now())
  next()
});
```

Optional path

Callback function

Middleware Examples

- app.use takes an optional path and a “middleware object”
  - “middleware object” is often a callback function

```
// A middleware function with no mount path,
// This code is executed for every request to the router
app.use((req, res, next) => {
  console.log('Time:', Date.now())
  next()
});
```

middleware parameters

- Most middleware callback functions take three arguments:
  - Request and response should look familiar
  - next (optional): the only way to call another middleware to be immediately after
    - If none is passed in, it will proceed to the next middleware defined

Error Middlewares

- Error middleware takes four arguments: error, request, response, next
  - error: an object with some context for displaying errors nicely
    - error.status: status code
  - Other arguments are the same
- Error middleware is defined last
  - We will go through an example error middleware!

```
app.use((error, req, res, next) => {
  // ... error handling code ...
});
```

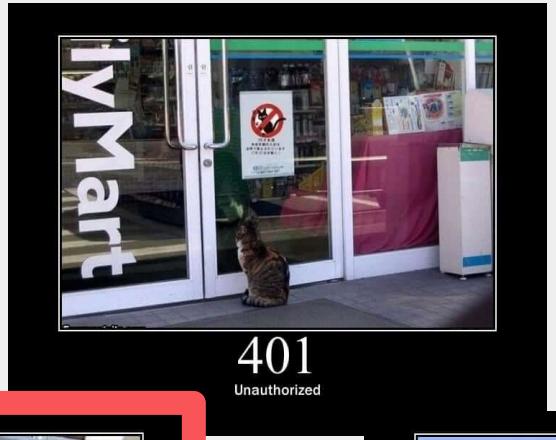
middleware parameters

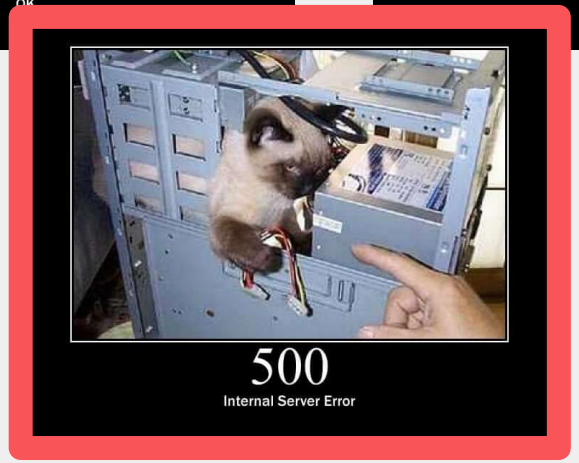
HTTP Status Codes

- 1xx- informational
- 2xx- you succeeded
- 3xx- redirect
- 4xx- you did something wrong
- 5xx- server did something wrong
- https://www.restapitutorial.com/httpstatuscodes.html

Comm(eow)n HTTP Status Codes

res.status(status_code)





Source:
https://http.cat/

Adding Error Handling

```
25  // any server errors cause this function to run
26  app.use((err, req, res, next) => {
27
28
29
30
31
32
33
34
35
36
37
38
39  });
40
```

server.js

Adding Error Handling

```
25  // any server errors cause this function to run
26  app.use((err, req, res, next) => {
27    const status = err.status || 500;
28
29
30
31
32
33
34
35
36
37
38
39  });
40
```

server.js

Adding Error Handling

```
25  // any server errors cause this function to run
26  app.use((err, req, res, next) => {
27    const status = err.status || 500;
28    if( status === 500 ){
29      // 500 means Internal Server Error
30      console.log("The server errored when processing a request");
31      console.log(err);
32    }
33
34
35
36
37
38
39  });
40
```

server.js

Adding Error Handling

```
25  // any server errors cause this function to run
26  app.use((err, req, res, next) => {
27    const status = err.status || 500;
28    if( status === 500 ){
29      // 500 means Internal Server Error
30      console.log("The server errored when processing a request");
31      console.log(err);
32    }
33
34    res.status(status);
35
36
37
38
39  });
40
```

server.js

Adding Error Handling

```
25  // any server errors cause this function to run
26  app.use((err, req, res, next) => {
27    const status = err.status || 500;
28    if( status === 500 ){
29      // 500 means Internal Server Error
30      console.log("The server errored when processing a request");
31      console.log(err);
32    }
33
34    res.status(status);
35    res.send({
36      status: status,
37      message: err.message,
38    });
39  });
40
```

server.js

Introducing a "Catch All" Endpoint

All endpoints which are not concretely defined will hit the ***** endpoint.

We want to return the HTML and bundle files of our React app.

Introducing a "Catch All" Endpoint

```
15  // import libraries needed for the webserver to work!
16  const express = require("express"); // backend framework for our node server.
17  const path = require("path"); // provide utilities for working with file and directory paths
18
19  // create a new express server
20  const app = express();
21
22  app.get("/api/test", (req, res) => {
23    res.send({ message: "Wow I made my first API!" });
24  });
25
26  // Load the compiled react files, which will serve /index.html and /bundle.js
27  const reactPath = path.resolve(__dirname, "..", "client", "dist");
28  app.use(express.static(reactPath));
29
30  // for all other routes, render index.html and let the react router handle it
31  app.get("*", (req, res) => {
32    res.sendFile(path.join(reactPath, "index.html"));
33  });
```

Import libraries

Introducing a "Catch All" Endpoint

```
15  // import libraries needed for the webserver to work!
16  const express = require("express"); // backend framework for our node server.
17  const path = require("path"); // provide utilities for working with file and directory paths
18
19  // create a new express server
20  const app = express();
21
22  app.get("/api/test", (req, res) => {
23    res.send({ message: "Wow I made my first API!" });
24  });
25
26  // Load the compiled react files, which will serve /index.html and /bundle.js
27  const reactPath = path.resolve(__dirname, "..", "client", "dist");
28  app.use(express.static(reactPath));
29
30  // for all other routes, render index.html and let the react router handle it
31  app.get("*", (req, res) => {
32    res.sendFile(path.join(reactPath, "index.html"));
33  });
```

Define an API with 1 route, /api/test

Introducing a "Catch All" Endpoint

```
15  // import libraries needed for the webserver to work!
16  const express = require("express"); // backend framework for our node server.
17  const path = require("path"); // provide utilities for working with file and directory paths
18
19  // create a new express server
20  const app = express();
21
22  app.get("/api/test", (req, res) => {
23    res.send({ message: "Wow I made my first API!" });
24  });
25
26  // Load the compiled react files, which will serve /index.html and /bundle.js
27  const reactPath = path.resolve(__dirname, "..", "client", "dist");
28  app.use(express.static(reactPath));
29
30  // for all other routes, render index.html and let the react router handle it
31  app.get("*", (req, res) => {
32    res.sendFile(path.join(reactPath, "index.html"));
33  });
```

Get path to react files, serve React files as static content

Introducing a "Catch All" Endpoint

```
15  // import libraries needed for the webserver to work!
16  const express = require("express"); // backend framework for our node server.
17  const path = require("path"); // provide utilities for working with file and directory paths
18
19  // create a new express server
20  const app = express();
21
22  app.get("/api/test", (req, res) => {
23    res.send({ message: "Wow I made my first API!" });
24  });
25
26  // Load the compiled react files, which will serve /index.html and /bundle.js
27  const reactPath = path.resolve(__dirname, "..", "client", "dist");
28  app.use(express.static(reactPath));
29
30  // for all other routes, render index.html and let the react router handle it
31  app.get("*", (req, res) => {
32    res.sendFile(path.join(reactPath, "index.html"));
33  });
```

For all other routes, send back the root React page

Introducing a "Catch All" Endpoint

```
15  // import libraries needed for the webserver to work!
16  const express = require("express"); // backend framework for our node server.
17  const path = require("path"); // provide utilities for working with file and directory paths
18
19  // create a new express server
20  const app = express();
21
22  app.get("/api/test", (req, res) => {
23    res.send({ message: "Wow I made my first API!" });
24  });
25
26  // Load the compiled react files, which will serve /index.html and /bundle.js
27  const reactPath = path.resolve(__dirname, "..", "client", "dist");
28  app.use(express.static(reactPath));
29
30  // for all other routes, render index.html and let the react router handle it
31  app.get("*", (req, res) => {
32    res.sendFile(path.join(reactPath, "index.html"));
33  });
```

localhost:3000 or localhost:5050?

- Both are servers running on our machine!
  - localhost:5050 → **hotloader, processes react code.**
  - localhost:3000 → **node server**, **our backend server.**

- To view your website → use localhost:5050

- To test your backend → use localhost:3000