Intro to Node Servers

A Moment of Caution

WARNING

Node and Express servers and **routing** are two of the *most important concepts* in the *entire* program.

Forewarning: This Is the Challenging Stuff

The topics we'll cover over the next three weeks will be challenging at times, but it's also some of the most important material. This is when you'll go from humble HTML/CSS hackers to employable full-stack engineers. Bring your A game!

Tips for Success

01 **Form a study group.** Get in the habit of explaining in-class exercises to one another, and work together on homework assignments.

02 **Take notes!** Jot down key concepts and ideas that come out of the in-class activities to help you keep things straight!

03 **Ask "conceptual" questions.** It's important to understand the big picture, so if a key concept is confusing, be sure to ask about it!

04 **Come to office hours.** Seeking help during office hours will be increasingly valuable as the material becomes more complex. Come to these sessions and ask questions! We're here to help you.

Be Confident!

Remember: If you've made it this far, you've proven that you have what it takes to succeed. Keep going!

Remember Our Mantra:
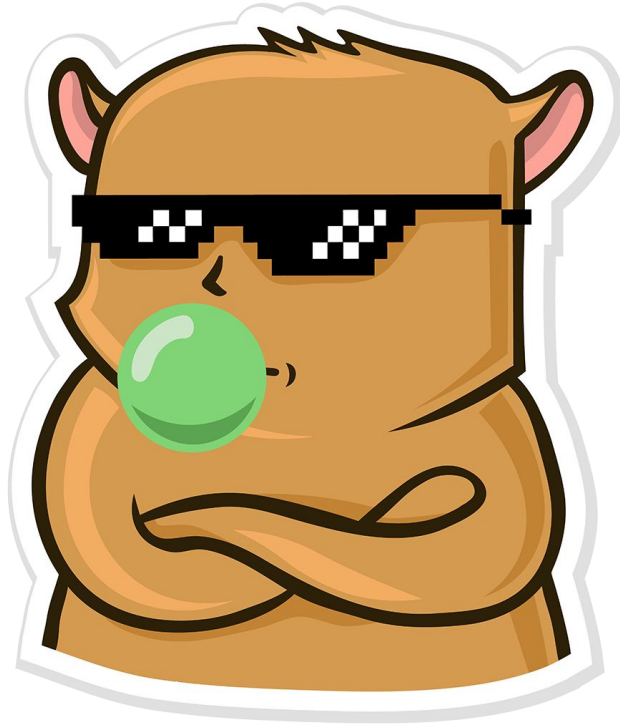When it comes to web development…

I know nothing.

You

So Let's Begin

Remind Me Again
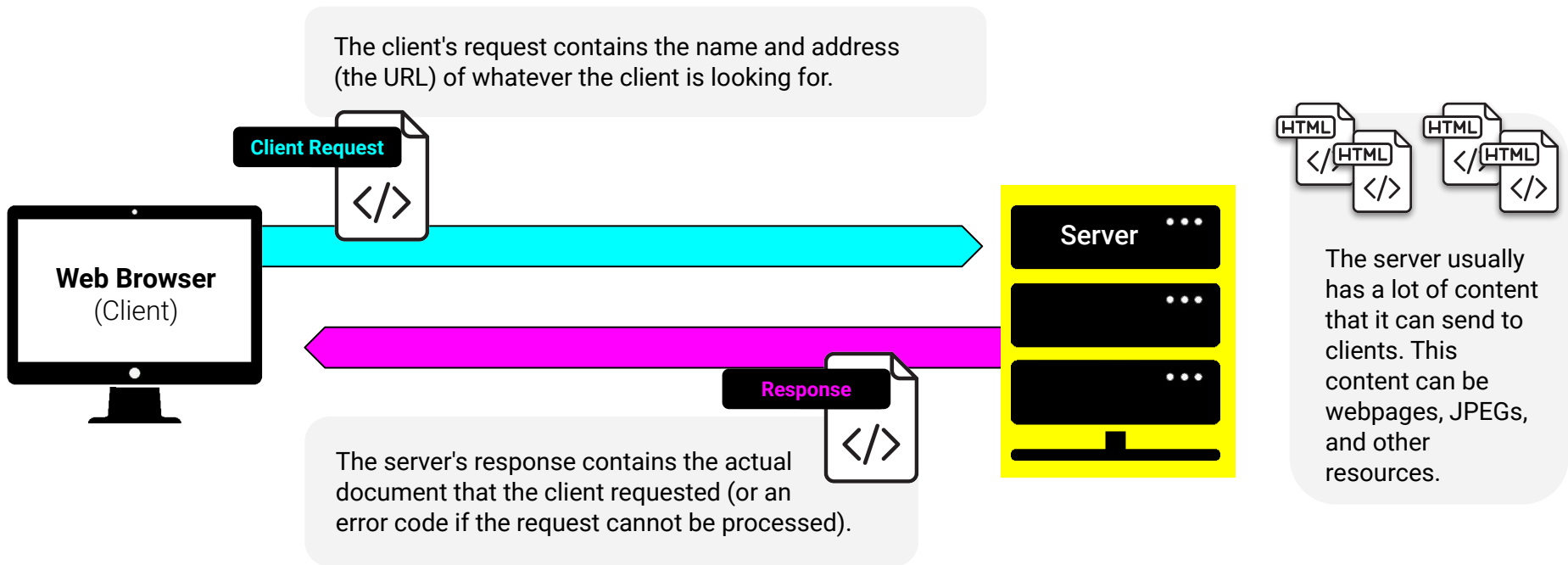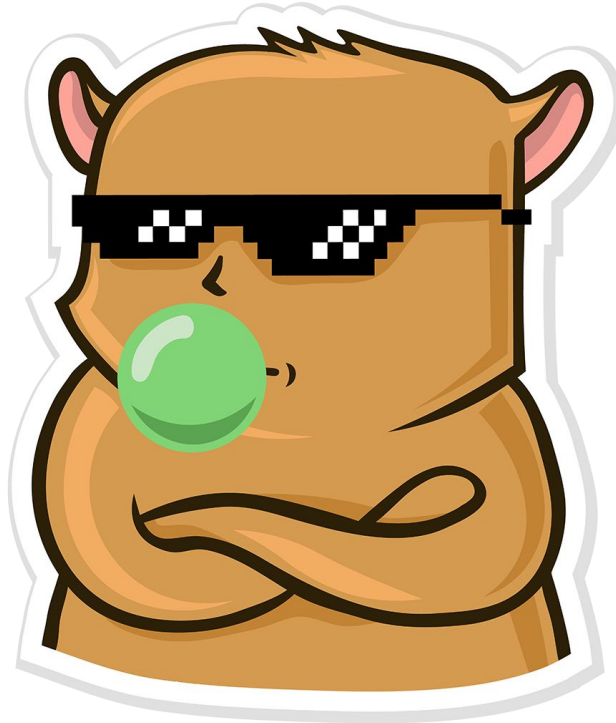
What is a **server**?

What Is a Server?

A server is the **machine** and **code** that handles client requests and responds to them.

The client's request contains the name and address (the URL) of whatever the client is looking for.

Client Request

Web Browser
(Client)

Server

The server usually has a lot of content that it can send to clients. This content can be webpages, JPEGs, and other resources.

Response

The server's response contains the actual document that the client requested (or an error code if the request cannot be processed).

Remind Me Again

What are some examples of <mark>**server-side**</mark> functions?

Server-Side Code in Action!

Visiting a URL and then being presented with an HTML page

Visiting an API endpoint that parses URL parameters to provide selective JSONs

Clicking an invoice that provides a PDF report

Image-processing software that applies a filter to an image and saves the new version
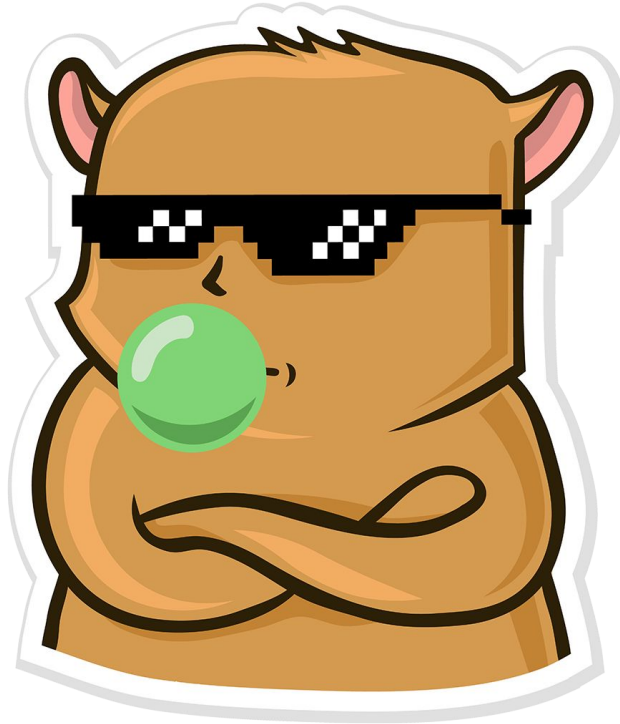
Google providing results relevant to searches on other sites
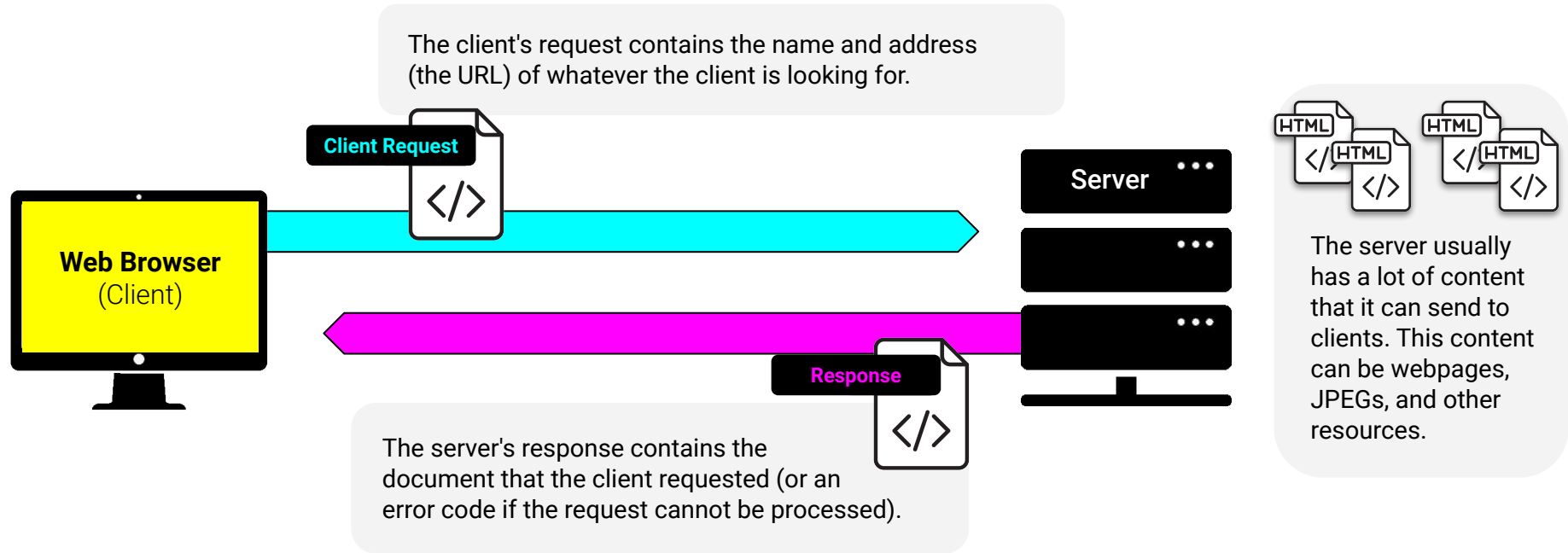
Remind Me Again

What is a **client**?

What Is a Client?

A **client** is a user's personal machine which makes "requests" of the server.

The client's request contains the name and address (the URL) of whatever the client is looking for.

Client Request

Web Browser
(Client)

Server

Response

The server's response contains the document that the client requested (or an error code if the request cannot be processed).

The server usually has a lot of content that it can send to clients. This content can be webpages, JPEGs, and other resources.
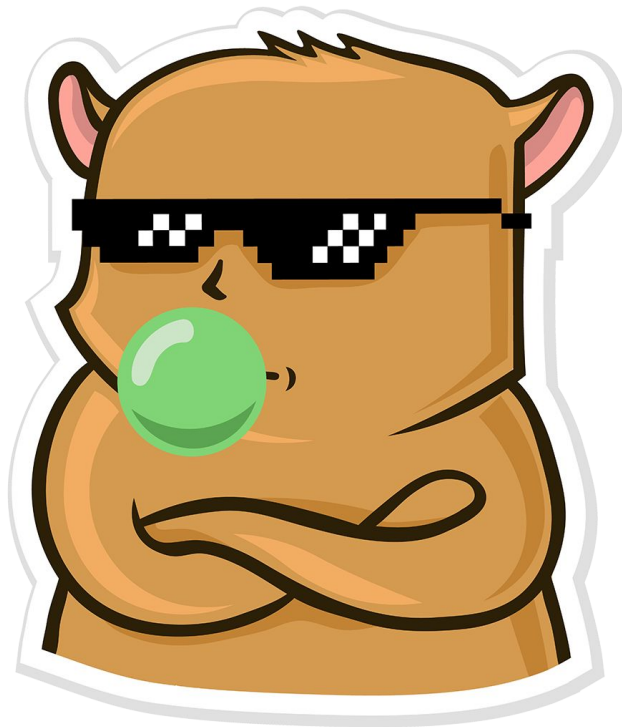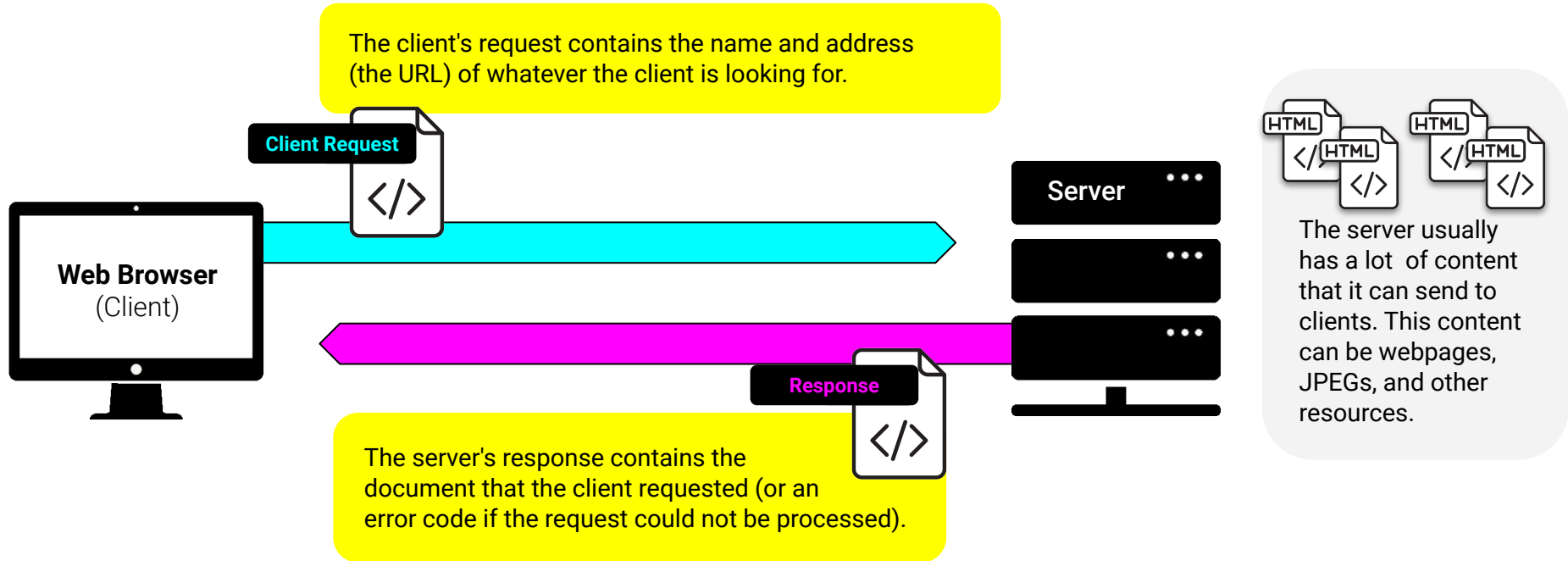
Remind Me Again

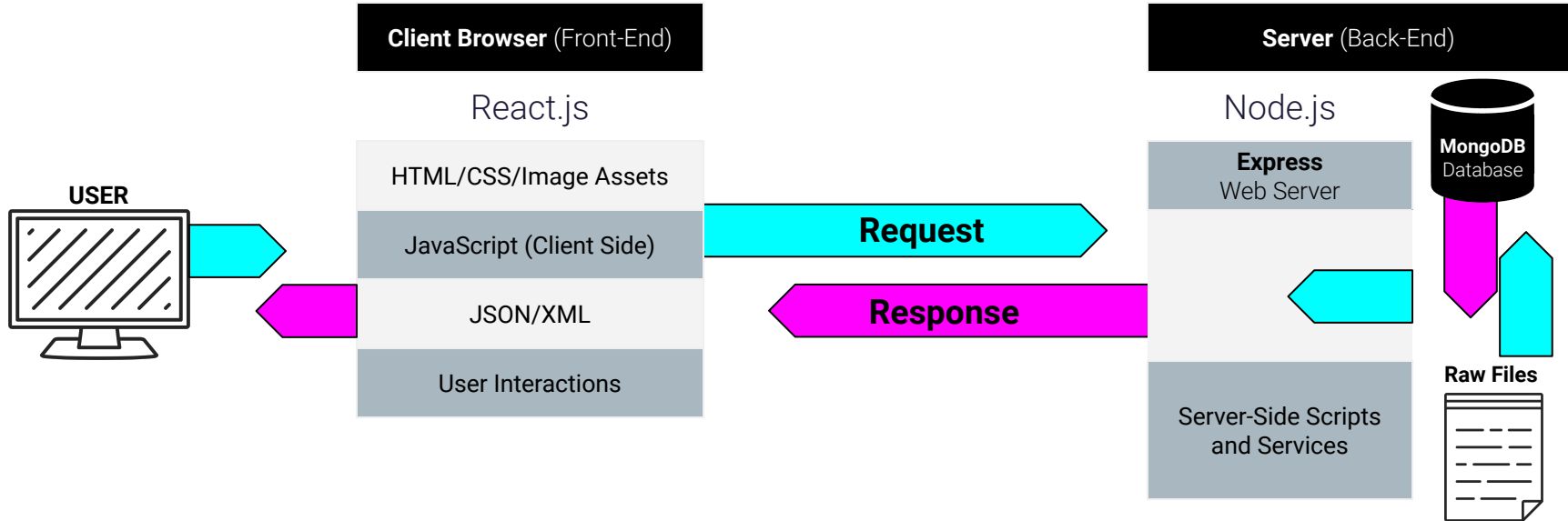How do the client and server **communicate** with each other?

HTTP/HTTPS

Clients and servers communicate with each other using a series of requests and responses defined by **HTTP / HTTPS (Hypertext Transfer Protocol Secure).**

The client's request contains the name and address (the URL) of whatever the client is looking for.

Client Request

Web Browser
(Client)

Server

The server usually has a lot of content that it can send to clients. This content can be webpages, JPEGs, and other resources.

Response

The server's response contains the document that the client requested (or an error code if the request could not be processed).
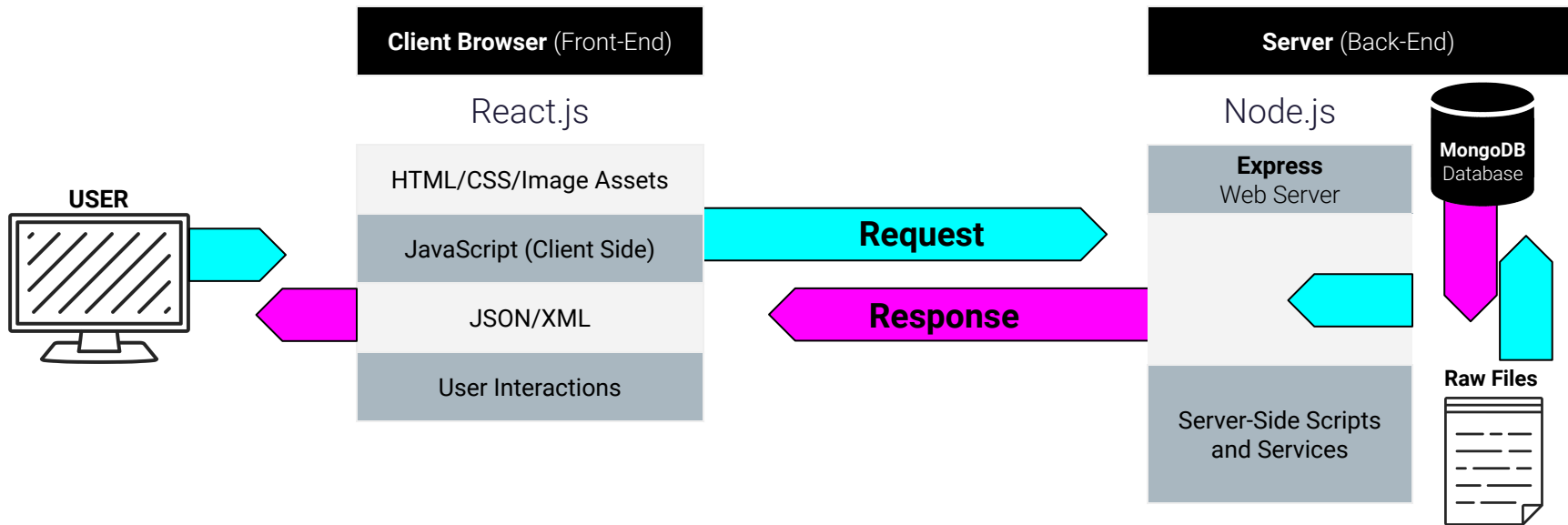
Full-Stack Development

In modern web applications, there is constant back-and-forth communication between the visuals displayed on the user's browser **(front-end)** and the data and logic stored on the server **(back-end)**.

Full-Stack Development

You can think of the server and the client as two distinct "machines."

Client Browser (Front-End)

React.js

HTML/CSS/Image Assets

JavaScript (Client Side)

JSON/XML

User Interactions

USER

Request

Response

Server (Back-End)

Node.js

Express
Web Server

Server-Side Scripts
and Services

MongoDB
Database

Raw Files

Deep Dive into Servers

Visualizing Servers

Server hardware looks like large, ruggedized versions of desktop computers.
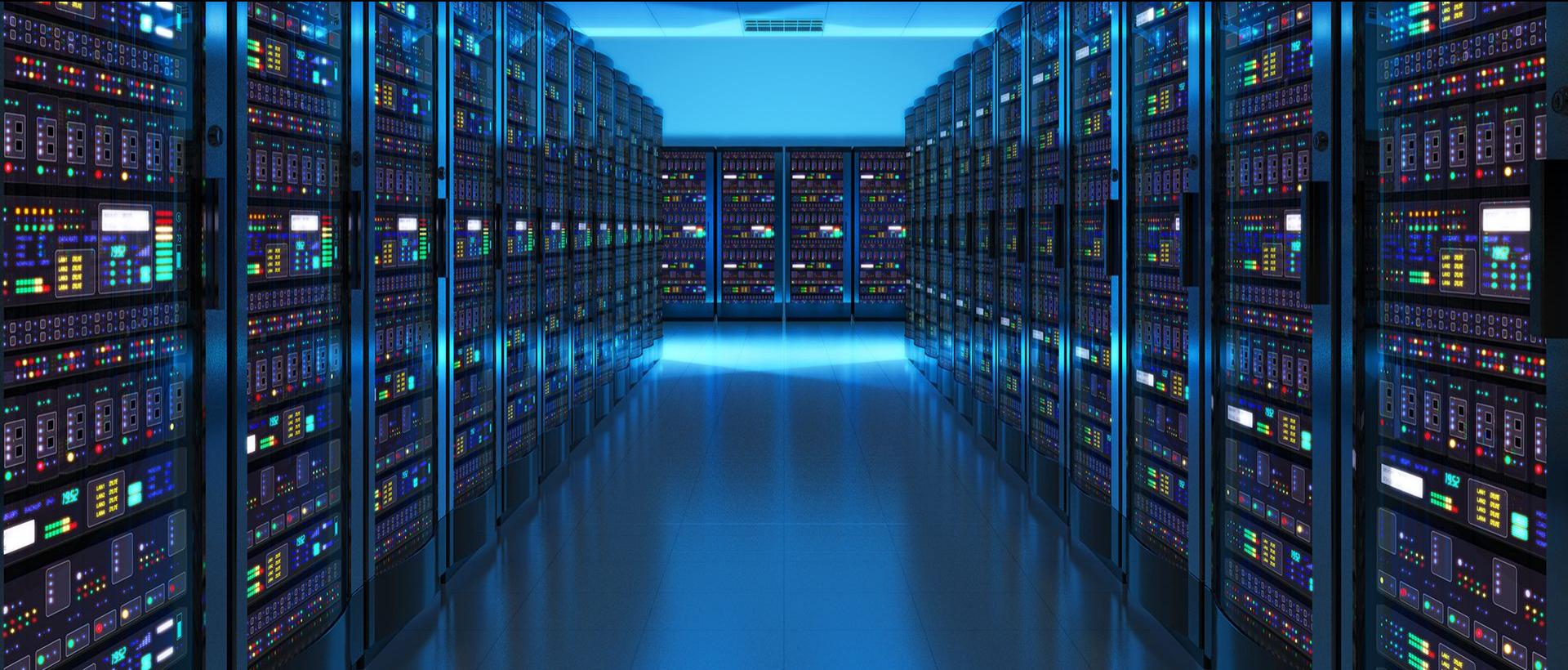
Visualizing Servers

These machines and their code handle all requests coming in from the browsers accessing a website.
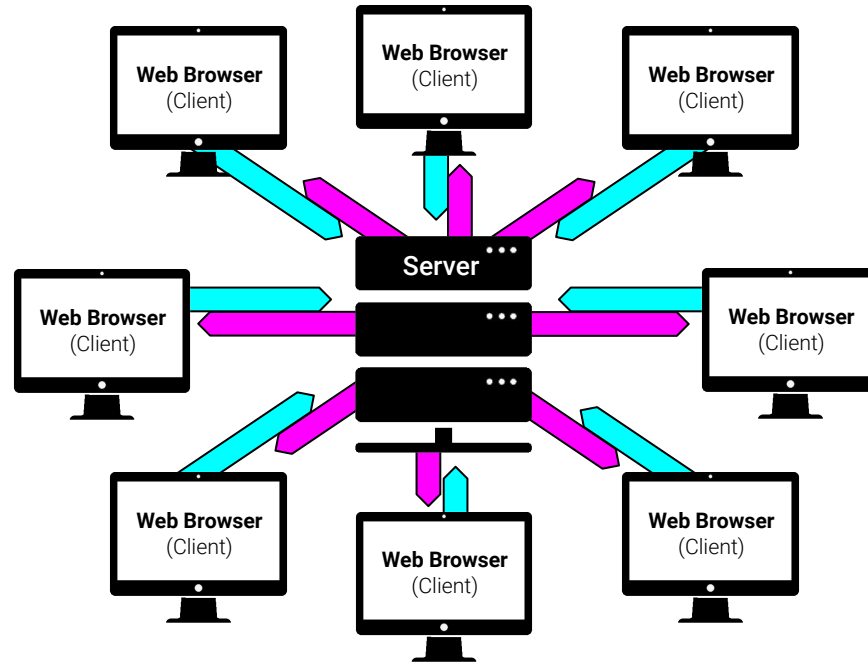
Visualizing Servers

Where does the server live?

Where Do Servers Live?

Servers live in dedicated hardware intended to handle the requests and responses from many clients.

Servers most often live on cloud platforms like AWS, Heroku, and Google Cloud.
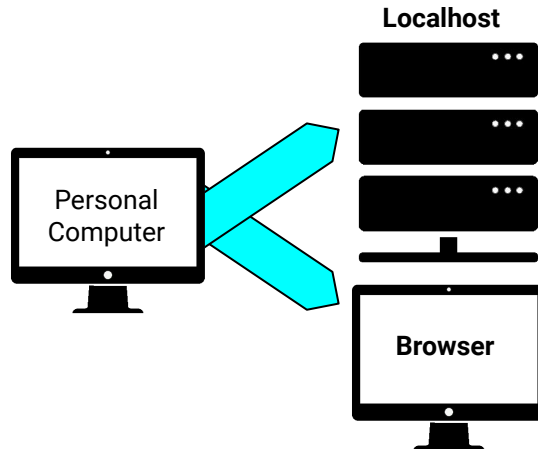
Servers During Development

During development, our personal computers will simulate both the client and the server.

We will create a local server (Localhost) on our computer and then use our browser to interact with it.

Localhost

Personal
Computer

Browser

Building a Server

Building a Server

For our purposes, "building a server" means writing code that handles what the server will do.

It's important to note that even though you pay for server-side hardware, you still need to write the code that goes inside.

This code enables functions such as:

Establishing connections to the database

Authenticating user requests

Handling client-side URL requests

Logging client requests

Performing server-side processes

Your Server: A Big Box

Throughout this week, imagine your server as a **big, empty box**.

We will add **code** snippets and **modules** to give our big, empty box the power to do stuff in response to all the requests that come in.

Server

Inside the Box: Connections
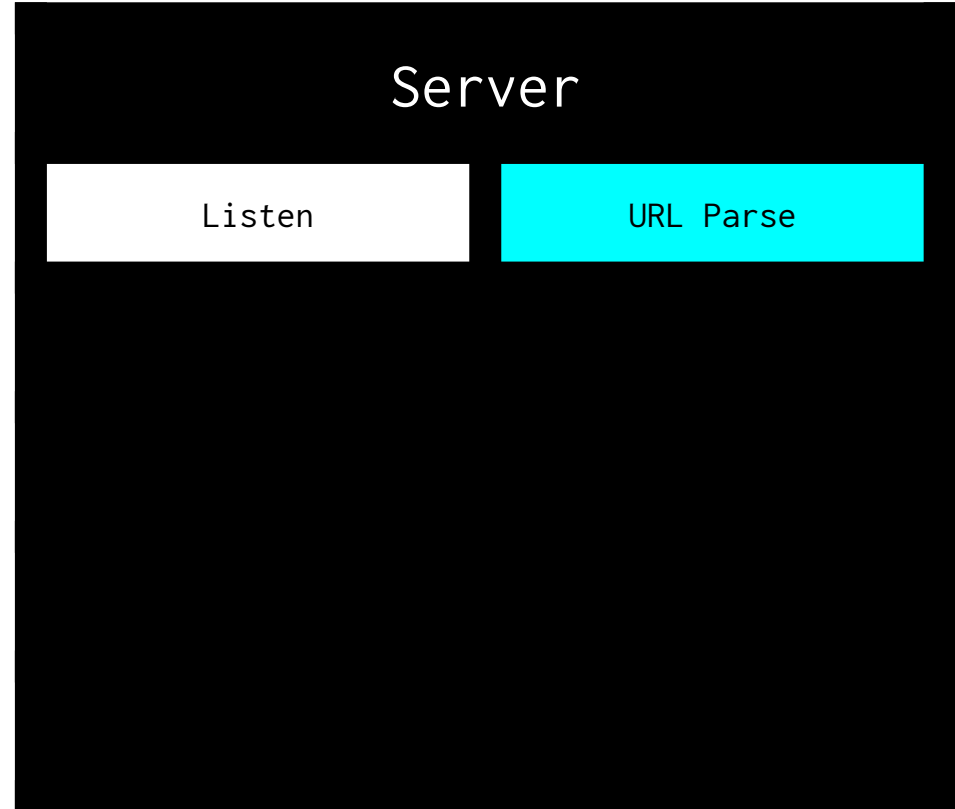
How do we allow the client to connect to the server?

We'll add a listener to the server that allows it to **listen** for client requests.
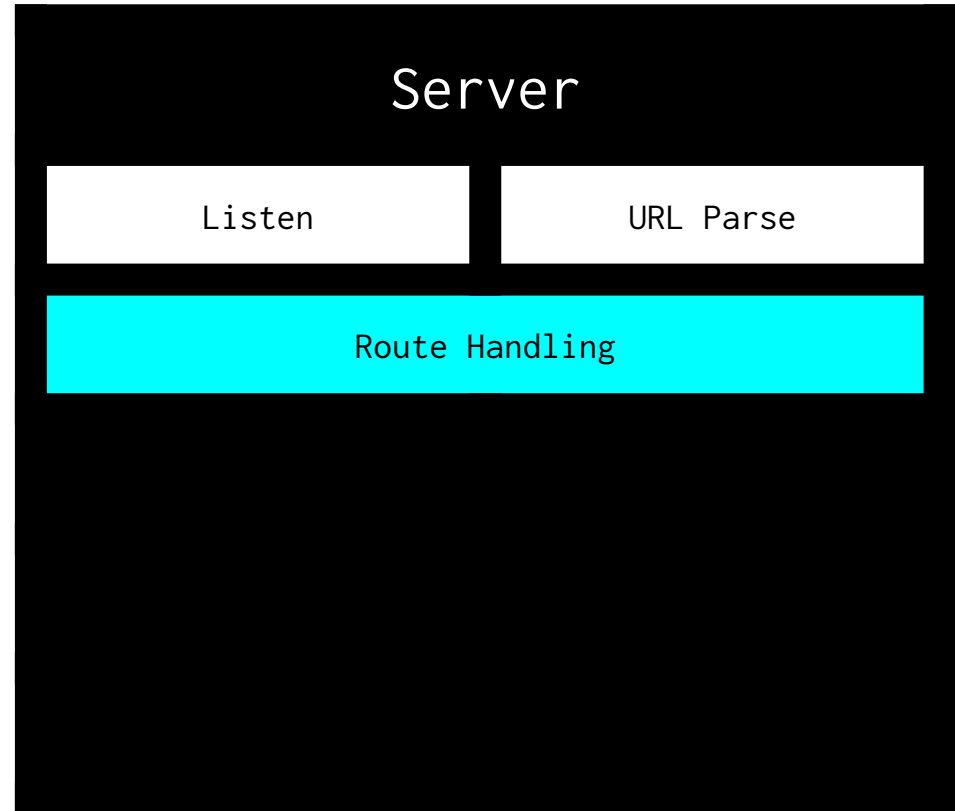

Server

Listen

Inside the Box: Parsing

We give the server the ability to **parse**, or interpret, URLs that the user requests.

```
Server

Listen          URL Parse
```

Inside the Box: Routing

Then, based on a URL's keywords, the server will be able to **route**, or direct, the flow of logic to initiate other processes.

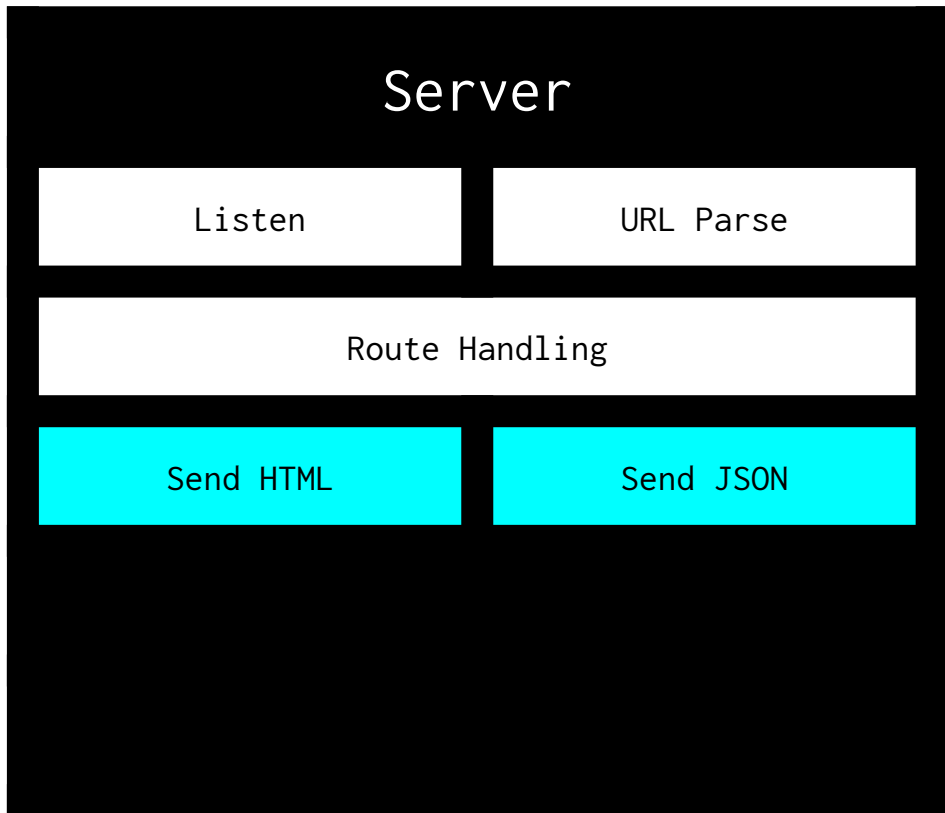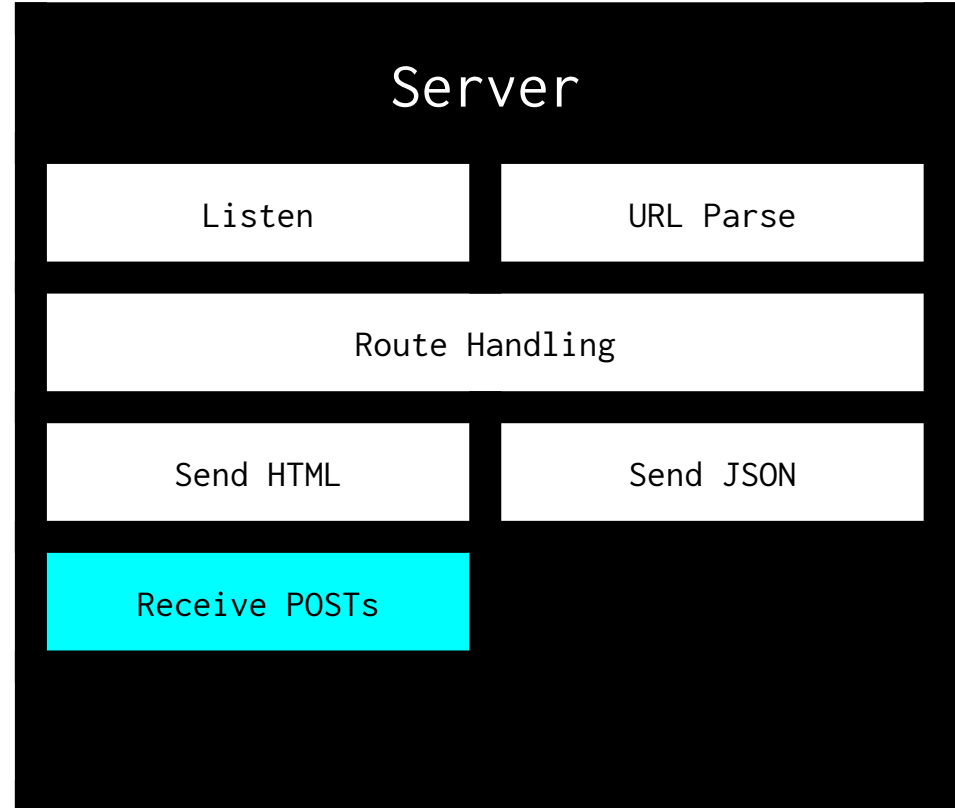# Inside the Box: Sending Files

These "other processes" might be sending an HTML file to be rendered or sending a JSON file to be rendered.

Inside the Box: Receiving Posts

We can also include a module to handle receiving a user's POST requests—the data they send the server.

Server

Listen

URL Parse

Route Handling

Send HTML

Send JSON

Receive POSTs

Inside the Box: Performing Logic

We can also have complex server-side logic that will initiate in response to users visiting a route endpoint or sending us data.

Server

Listen

URL Parse

Route Handling

Send HTML

Send JSON

Receive POSTs

Server-Side Logic
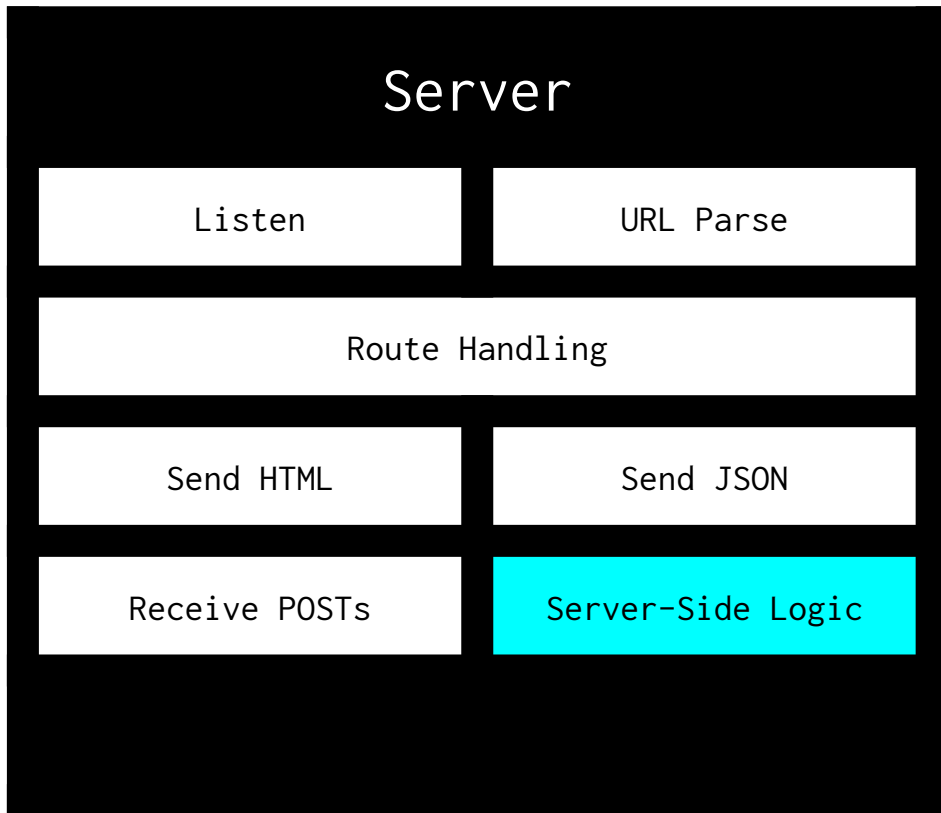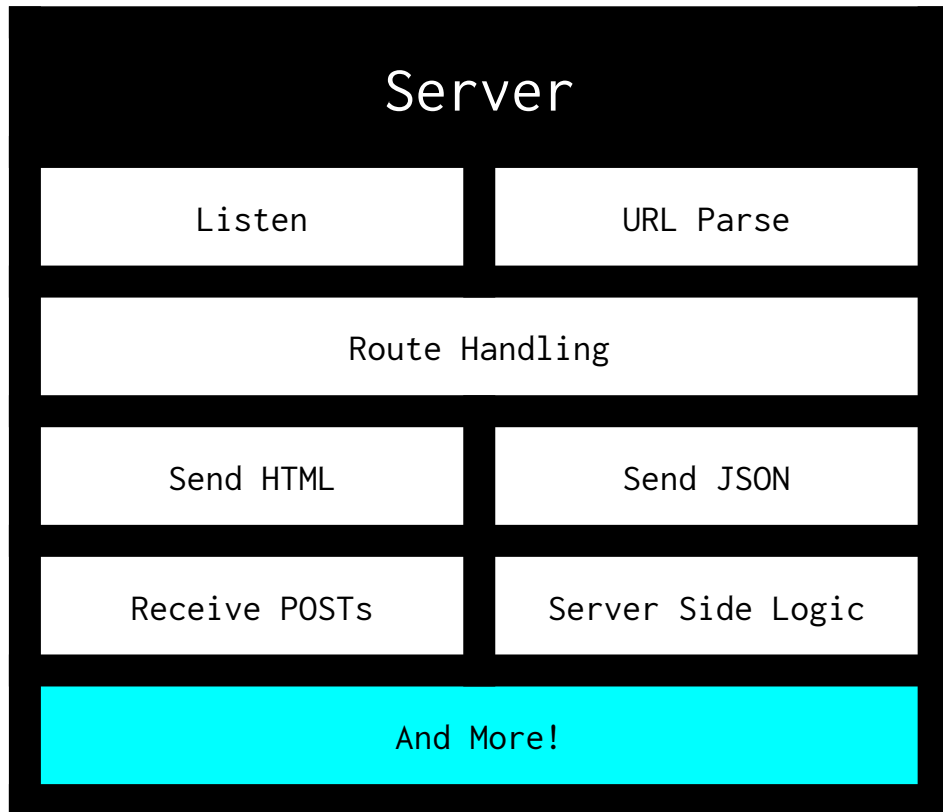
Inside the Box: And More!

But it doesn't stop there!

We can add functionality for authentication, logging requests, connecting to databases, and so much more.

Always remember that we're coding out these functionalities into our box!

Server

| Listen | URL Parse |

| Route Handling |

| Send HTML | Send JSON |

| Receive POSTs | Server Side Logic |

And More!

Questions?

<Time to Code>

Express Yourself

Early Concerns

> This is all so confusing.

Comment #1: "This is all so confusing."

Response: And it will be for your employer as well.

"I feel like I'm just copying and pasting."

Ctrl + C   Ctrl + V

Comment #2: "I feel like I'm just copying and pasting."

Response:

Back-end code libraries mean you have to code less.

Often you are just copying and pasting "best practices" over and over again.
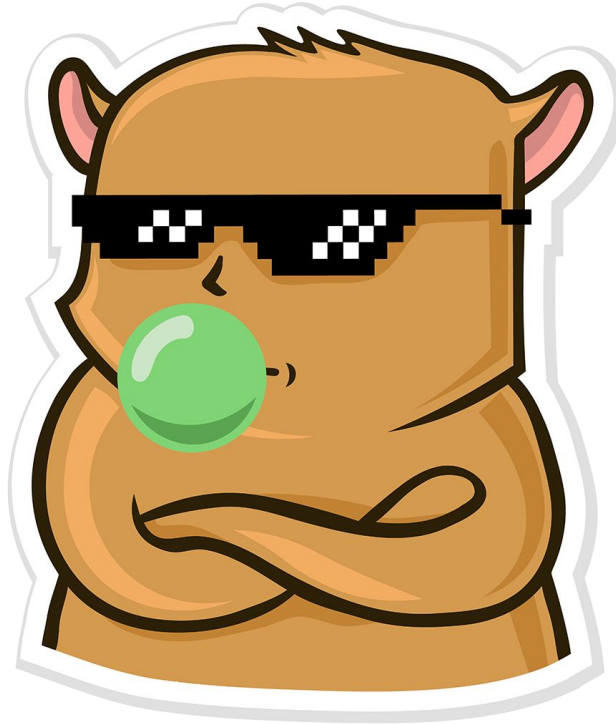
Ctrl + C   Ctrl + V

Express

Remind Me Again

What is **Express**?
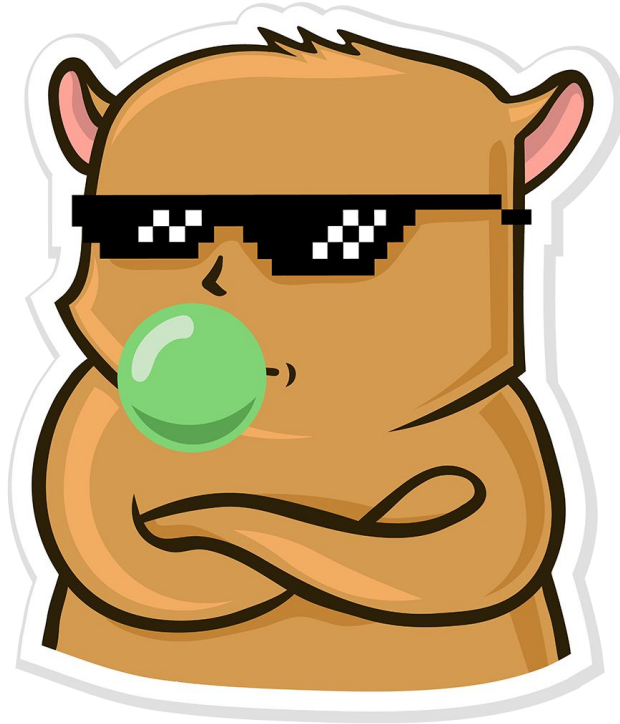
Express.JS
Express is a web framework for Node that makes creating server-side code much simpler.

Remind Me Again

What is a **route**?

Routes = Maps

Routes = Maps

User visits a link...

/

/saved

/api/headlines

Server

User is shown the "Home" Page

User is shown the "Saved" Page

User is given JSON containing all of the headline data.
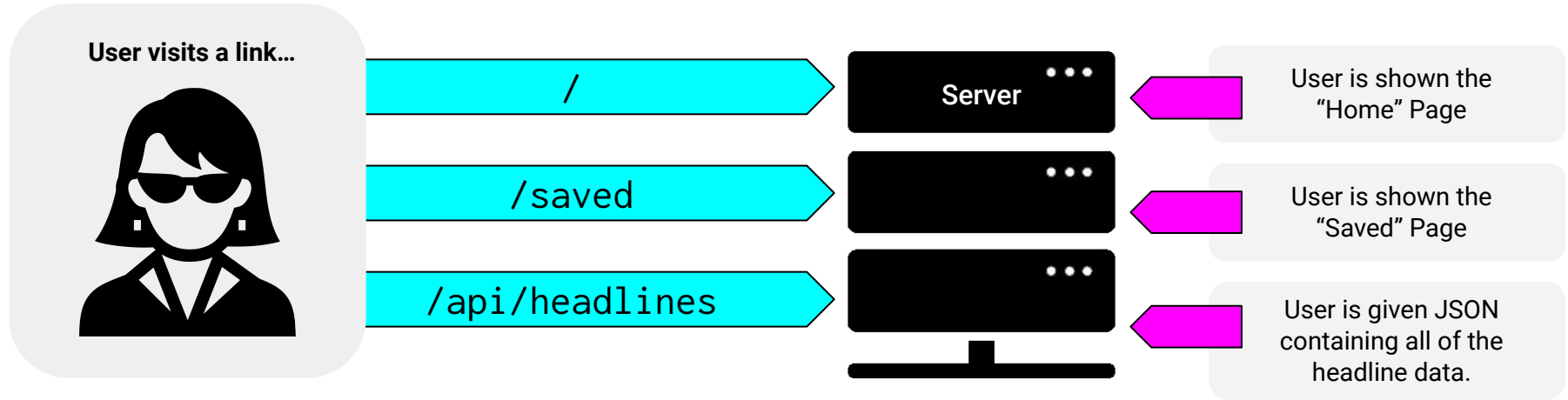
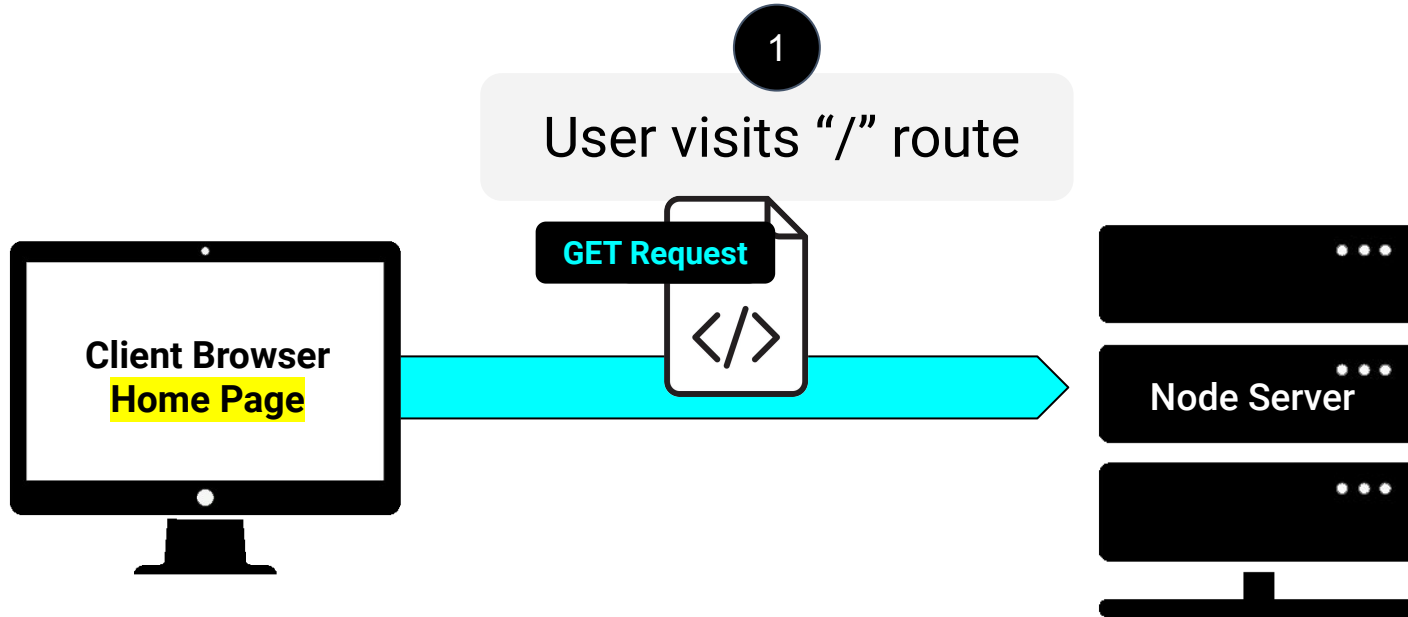**Quick Example**

nyt-mongo-scraper.herokuapp.com

Client-Server Communication (GET)

1

User visits "/" route

GET Request

Client Browser
Home Page

Node Server

Client-Server Communication (GET)

1

User visits "/saved" route

GET Request

</>

Client Browser
Saved Articles

Node Server

2

Request triggers the code in the server route. The server then finds the relevant HTML content and data.

HTML
</>

3

Server responds by providing HTML with saved articles.

Client-Server Communication (POST)

1

User saves an article using "api/headlines".

POST Request

Client Browser
Clicks on
Save Article
button

Node Server

Client-Server Communication (POST)

1

User saves an article using "api/headlines".

POST Request

Client Browser
Clicks on
Save Article
button

Node Server

2

Request triggers the code in the server route. The server then sets the article as "saved" in the database.

HTML

3

"Saved" articles are loaded on the saved page.

<Time to Code>