Server-Side!

The Coding Bootcamp

A Moment of Caution

Focus!

Node + Express Servers and Routing are two of the

MOST IMPORTANT CONCEPTS

in the **<u>ENTIRE</u>** program.

Seriously… Try to learn this now.

Forewarning: This is the *HARD* Stuff

- These next three weeks are some of the hardest to grasp.
- **But are also some of the most important.**
- This is where you go from humble HTML, CSS hackers to full-stack, employable engineers.
- Bring your A-Game!!!

Don’t let this be you!

How to Succeed Through the Full-Stack Apocalypse

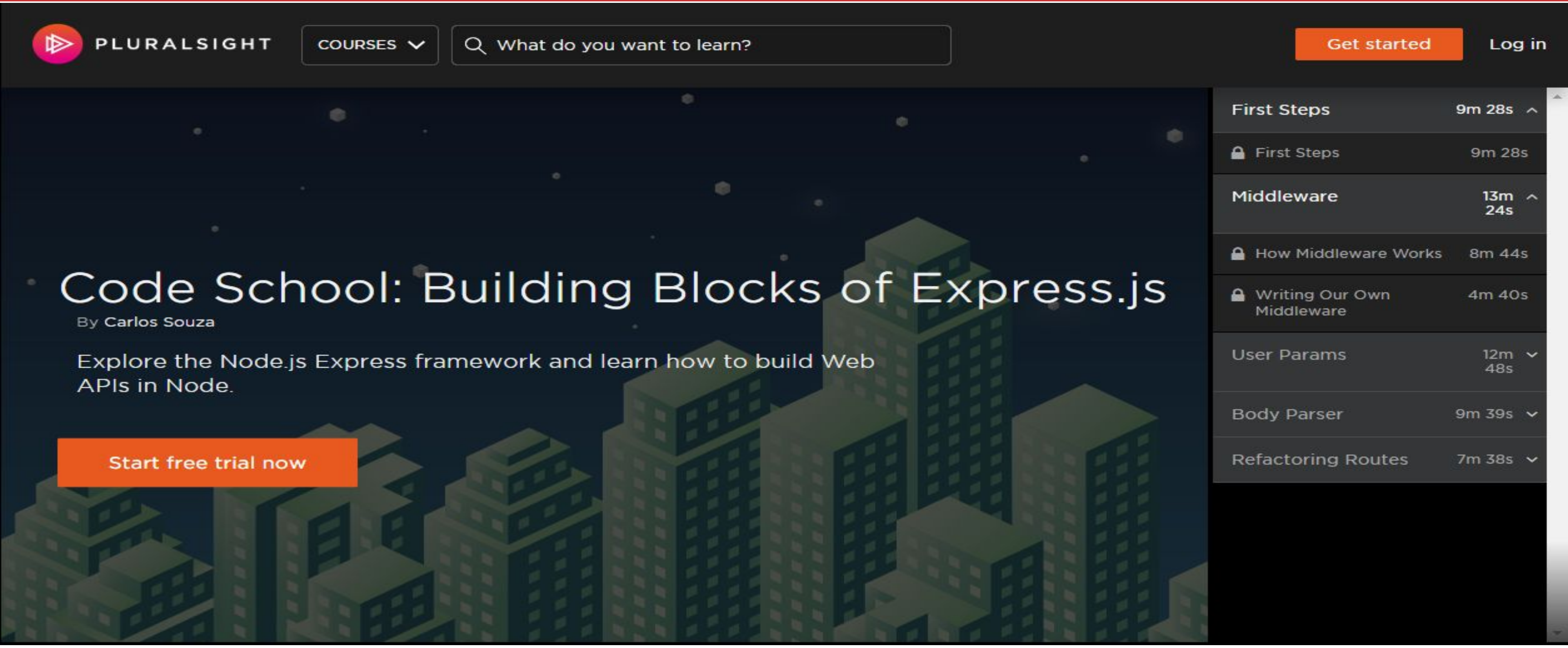
1. **Form a Study Group Now**
   Get in the habit of explaining in-class exercises to one another. Work together on homework assignments.

2. **Take notes!**
   Jot down concepts or key ideas that come out of activities. This class isn't a lecture, but it may help you keep things straight!

3. **Ask "Conceptual" Questions**
   Don't let big picture ideas gloss over you. Be courageous and ask questions in class. (Don't worry -- if your question isn't relevant, we'll let you know. But if it is, you owe it to yourself to understand!)

4. **Come to Office Hours**
   As things get trickier, we'll be available to review any concepts during office hours. Come to these. Ask questions! We want to help.

How to Succeed Through the Full-Stack Apocalypse



Code School: Building Blocks of Express.js

By Carlos Souza

Explore the Node.js Express framework and learn how to build Web APIs in Node.

First Steps	9m 28s
First Steps	9m 28s
Middleware	13m 24s
How Middleware Works	8m 44s
Writing Our Own Middleware	4m 40s
User Params	12m 48s
Body Parser	9m 39s
Refactoring Routes	7m 38s

5. Get a PluralSight Trial Account

Then immediately start completing the Building Blocks of Express.js course. It's *very* good.

https://www.pluralsight.com/courses/code-school-building-blocks-of-express-js

6. Lastly, be confident !!!

There is zero reason to get despondent at this point. 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…

Remember Our Mantra…

I know ***<u>nothing.</u>***

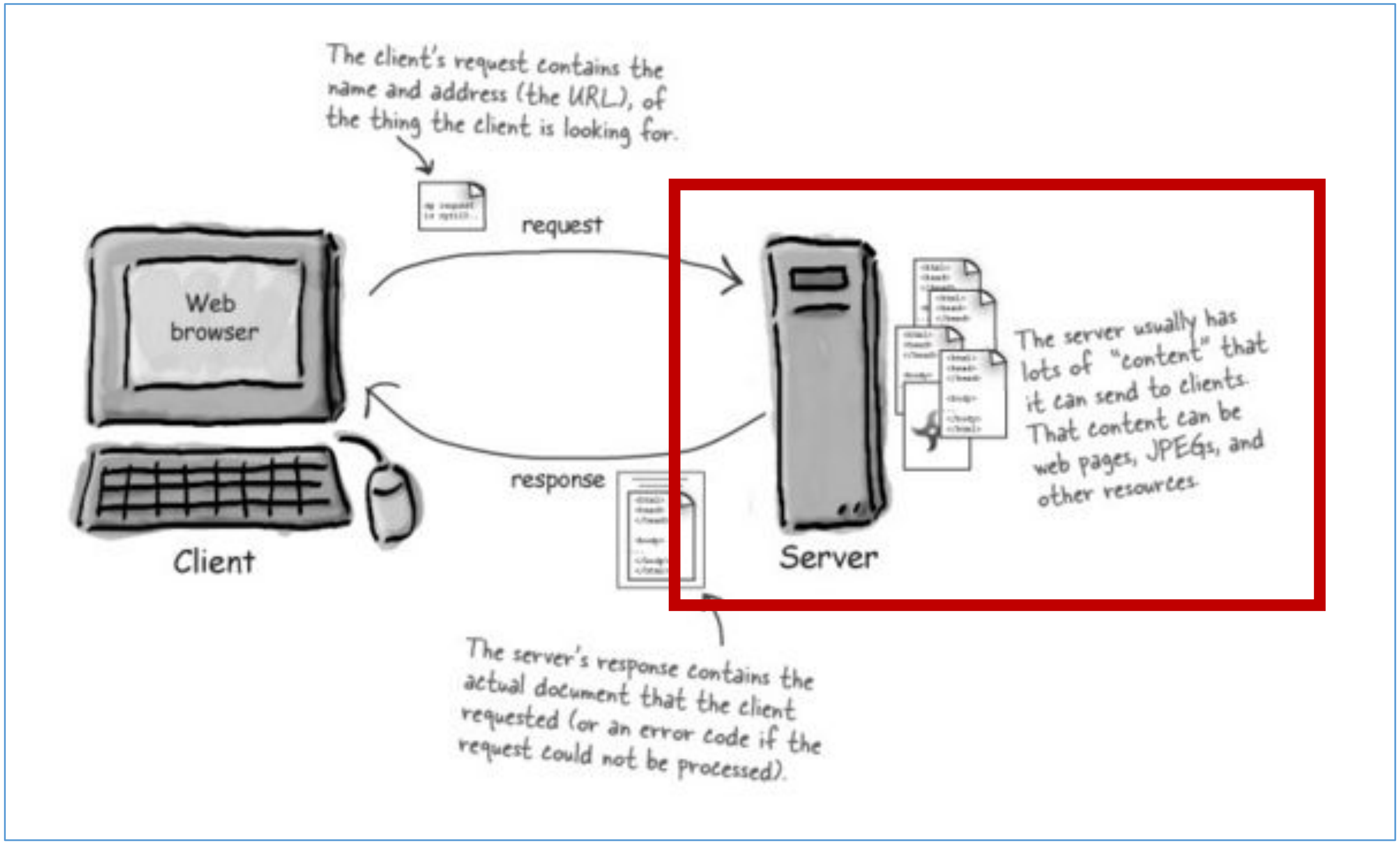
You.

So Let’s Begin…

Remind me again…

What is a **server**?

Server Definition


The client's request contains the name and address (the URL), of the thing the client is looking for.
request
Web browser
Client
response
Server
The server usually has lots of "content" that it can send to clients. That content can be web pages, JPEGs, and other resources
The server's response contains the actual document that the client requested (or an error code if the request could not be processed).

Server:

The Machine and Code that handles requests and respond to them.

Remind me again…

What are examples of **server-side** functions?

Server-Side Code in Action!

- Visiting a URL and then being given an HTML page.
- Visiting an API end-point that parse URL parameters to provide selective JSONs.
- Clicking an invoice that provides a PDF report.
- Image processing software that takes an image applies a filter, then saves the new version.
- Google providing “results” relevant to your searches on other sites.

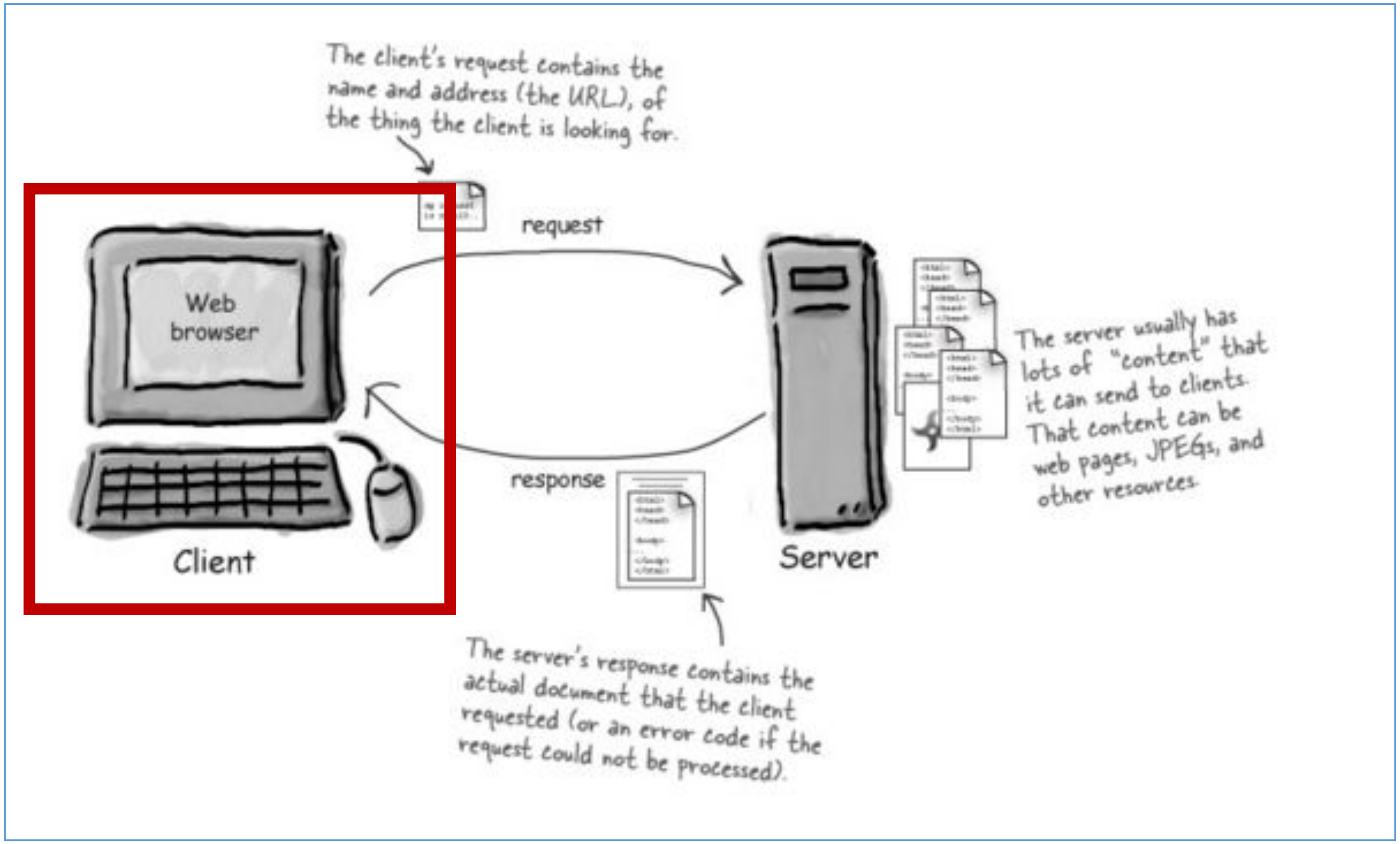
Server-Side Code in Action!

- Visiting a URL and then being given an HTML page.
- Visiting an API end-point that parse URL parameters to provide selective JSONs.
- Clicking an invoice that provides a PDF report.
- Image processing software that takes an image applies a filter, then saves the new version.
- Google providing “results” relevant to your searches on other sites.

Remind me again…

What is a **client**?

Client Definition


The client's request contains the name and address (the URL), of the thing the client is looking for.
request
Web browser
Client
response
Server
The server usually has lots of "content" that it can send to clients. That content can be web pages, JPEGs, and other resources.
The server's response contains the actual document that the client requested (or an error code if the request could not be processed).

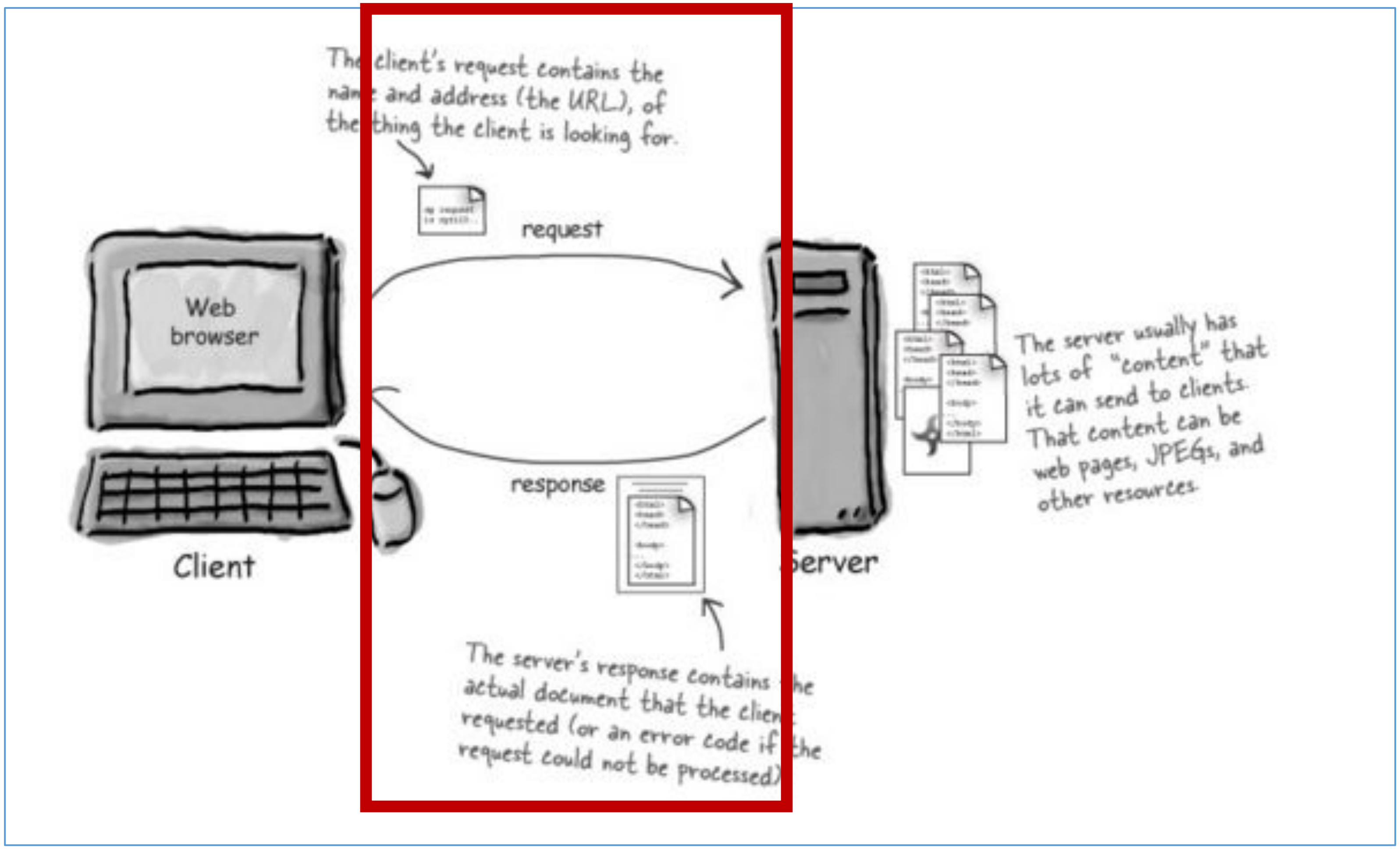
Client:

The users' personal machines that make "requests" of the server.

Bonus Question:

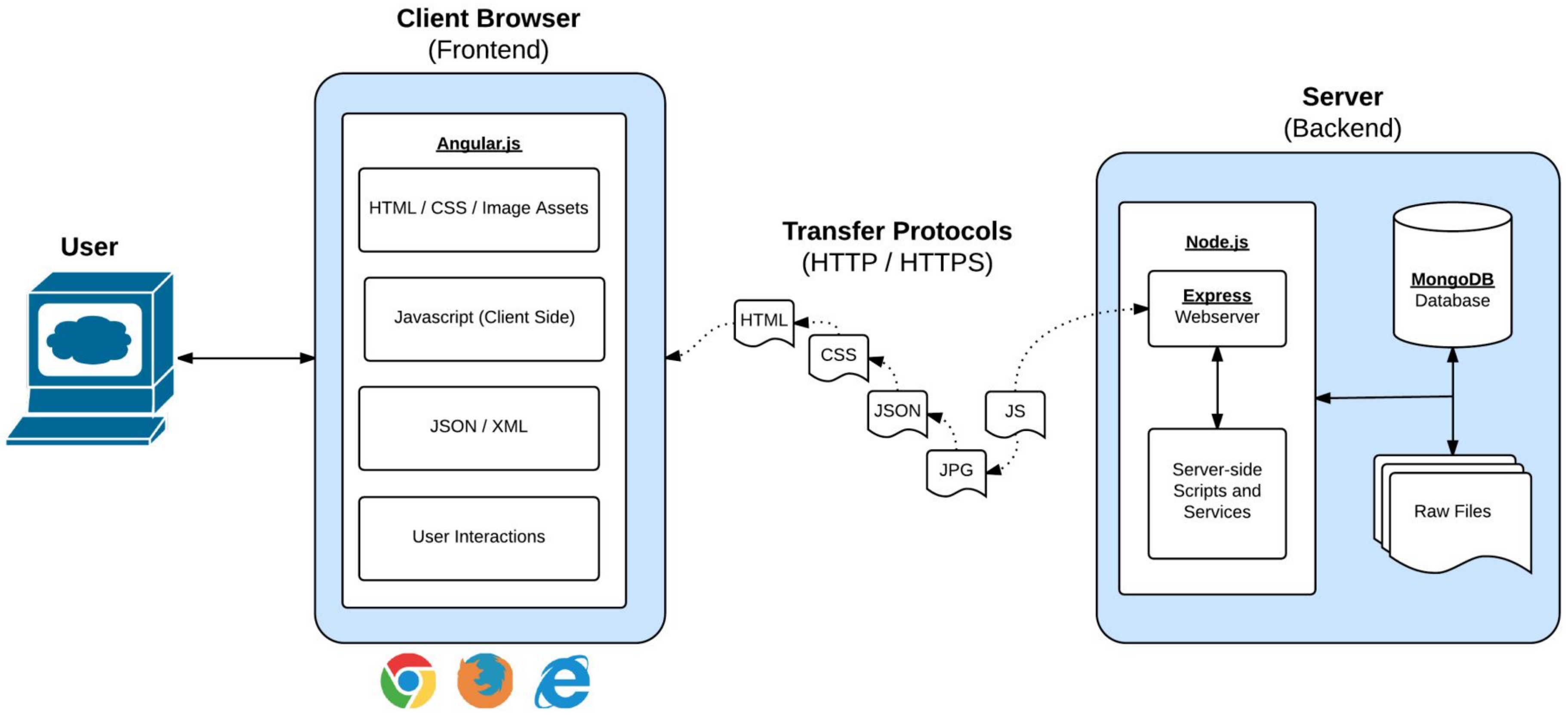
How do the client and server **communicate** with one another?

HTTP Definition


The client's request contains the name and address (the URL), of the thing the client is looking for.
request
Web browser
Client
response
Server
The server usually has lots of "content" that it can send to clients. That content can be web pages, JPEGs, and other resources
The server's response contains the actual document that the client requested (or an error code if the request could not be processed)

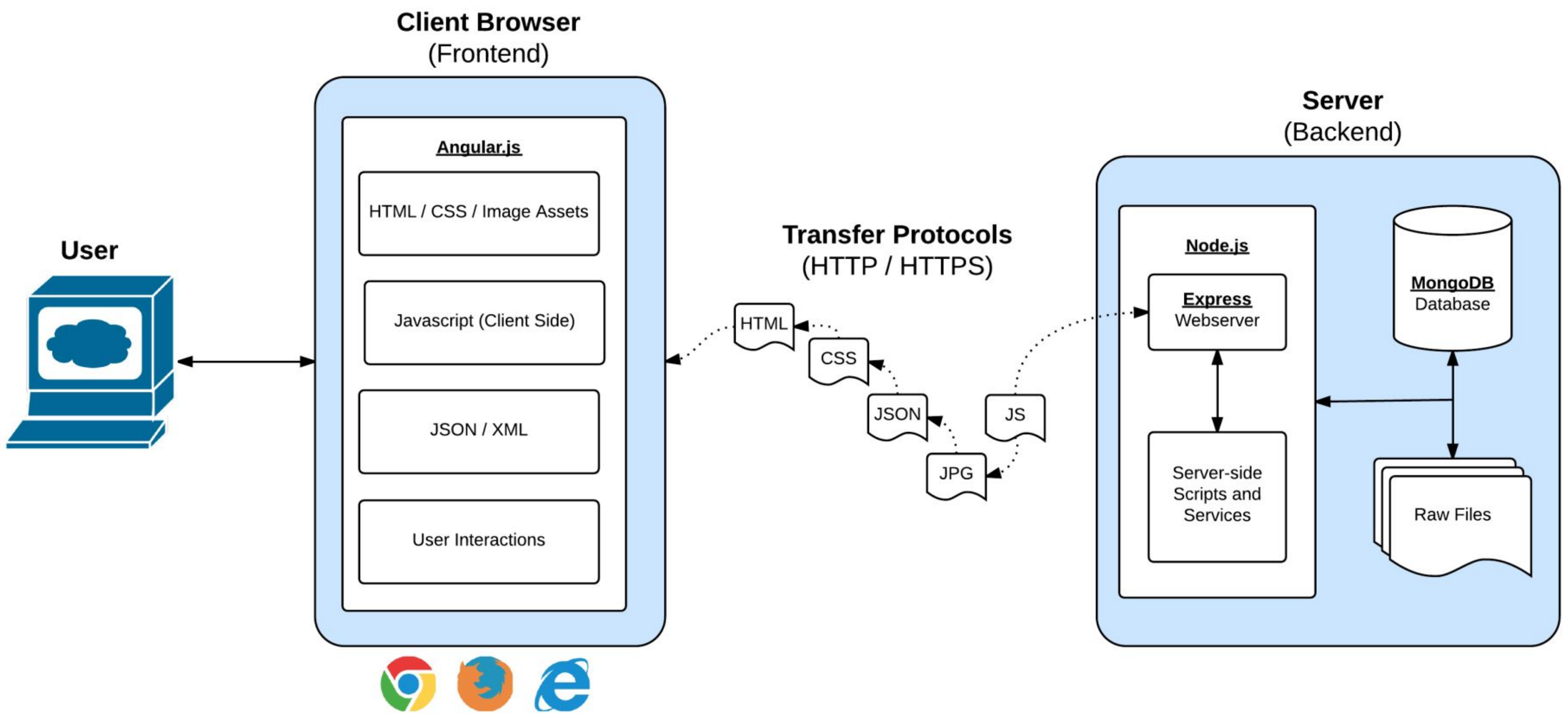
Clients and Servers communicate back and forth using a series of understood communications defined by **HTTP / HTTPs**.

Full-Stack Development



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

Full-Stack Development



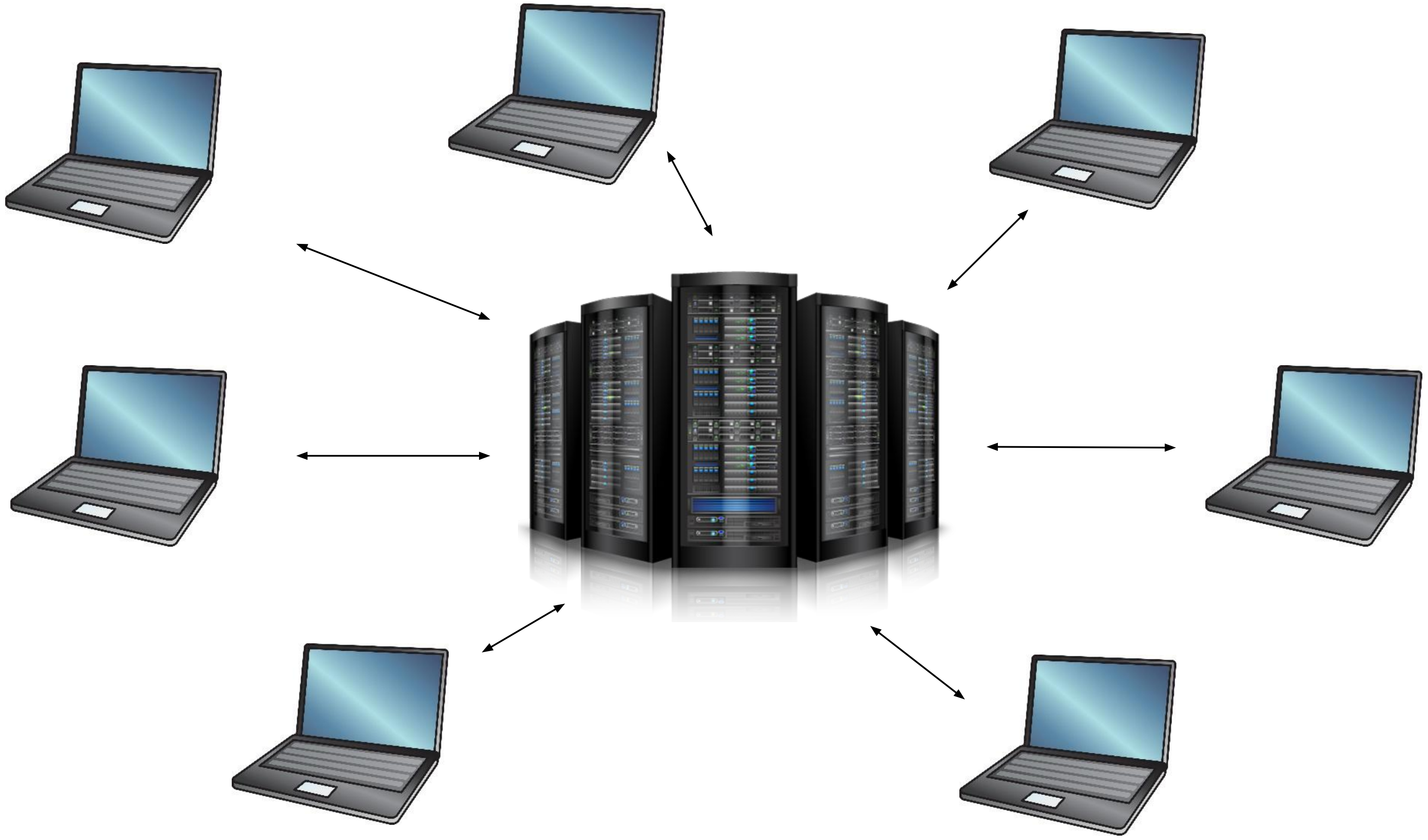
- In a way think of this as being **two distinct machines.**
- A server is one machine and the client is a second machine.

Digging Deep into Server

Visualizing Servers

Server Hardware look like large, ruggedized versions of desktop computers.

Visualizing Servers



These machines (and their respective code) handle all of the requests coming in from browsers accessing a website.

Visualizing Servers

Where does the server **live**?

Where Do Servers Live?

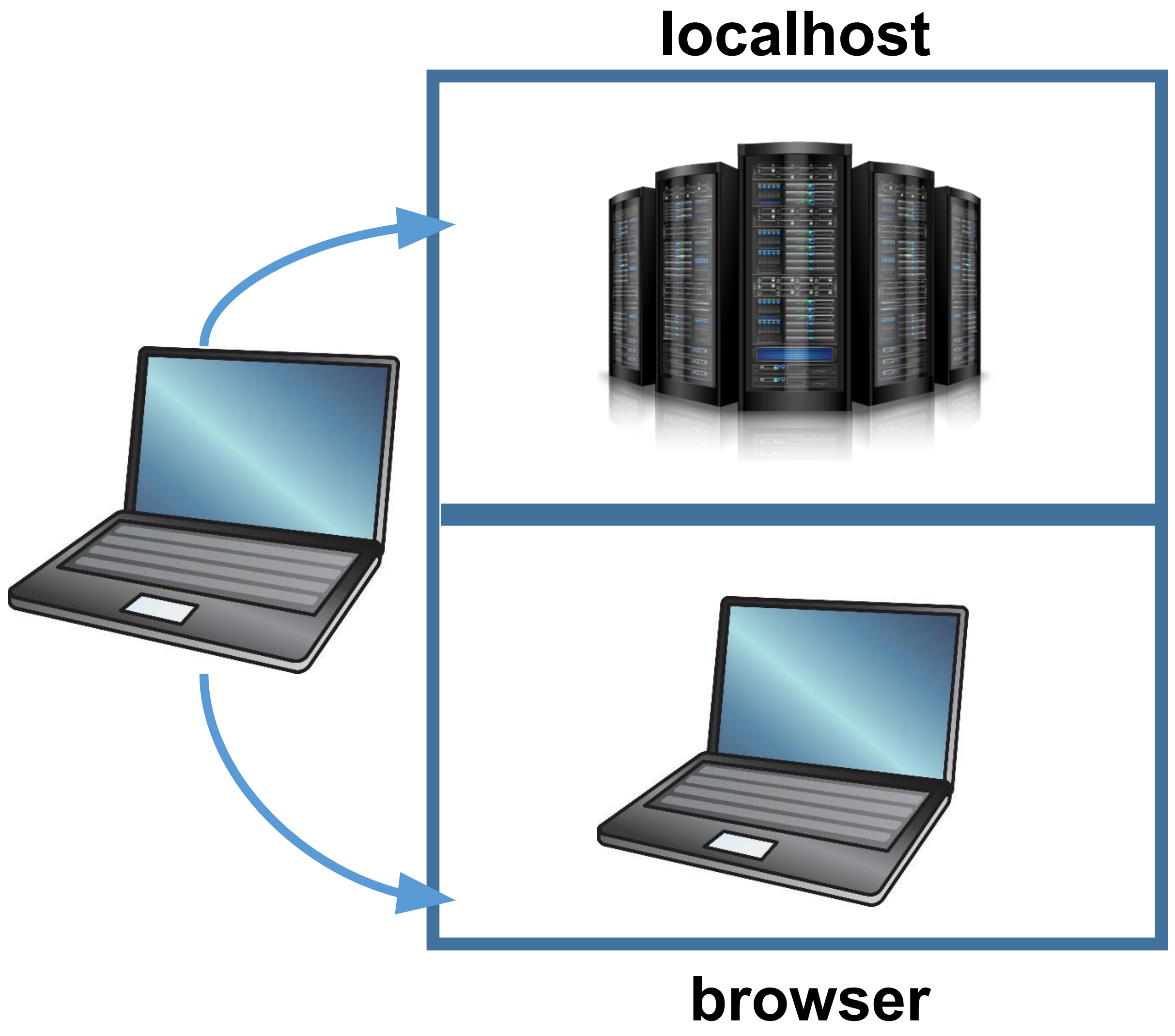
- Servers live in dedicated hardware intended to handle ALL the requests and responses of many clients.

- Servers can live most often live on cloud platforms like AWS, Heroku, Google Cloud, etc.

Servers During Development

Important Note:

- During development our personal computers will be able to simulate both.
- We will create a “local server”
- And then use our browser to interact with it.



Building a “Server”

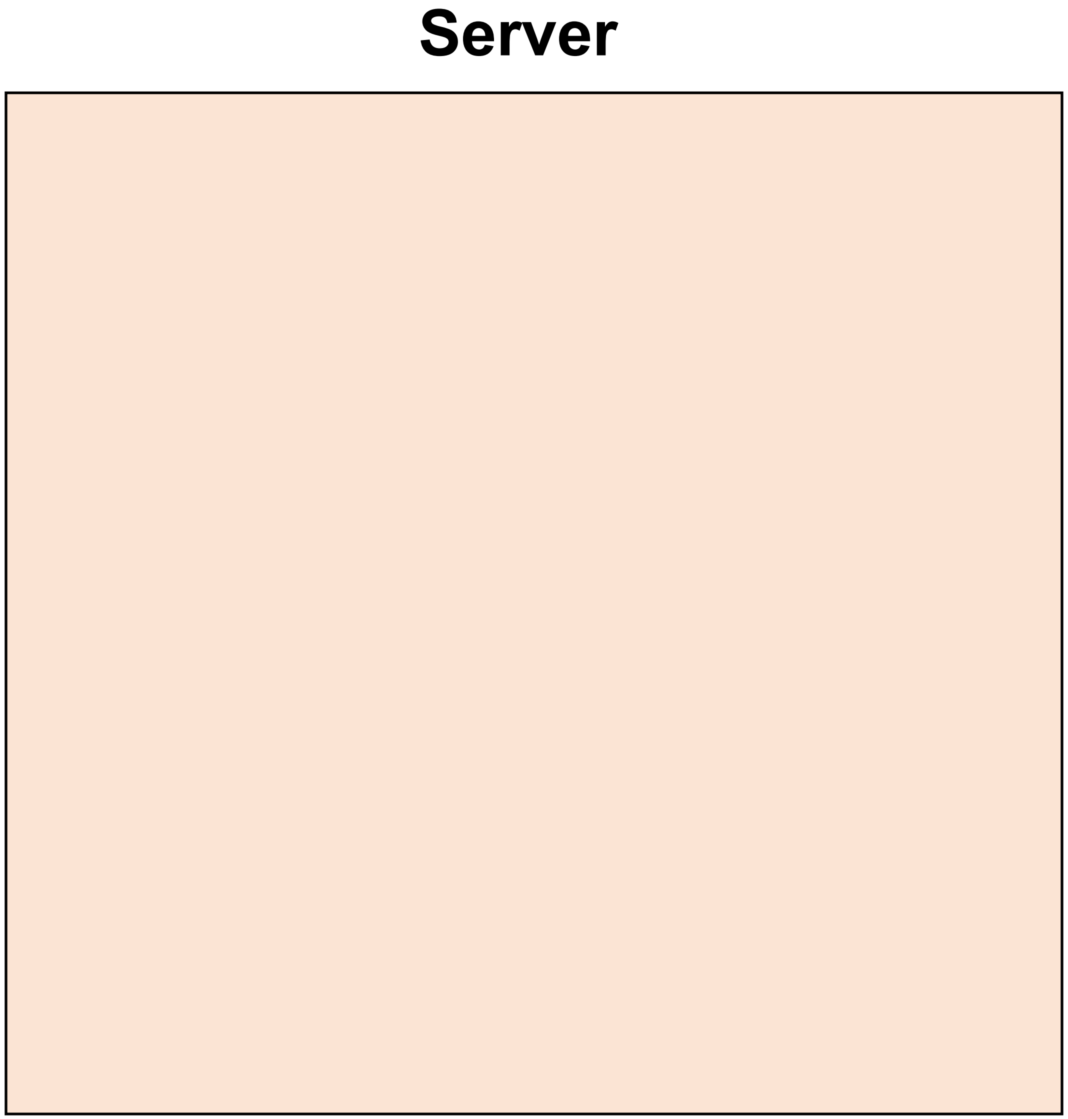
Creating a Server

- For our purposes, “creating a server” equates to writing the 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 create the code that goes inside.

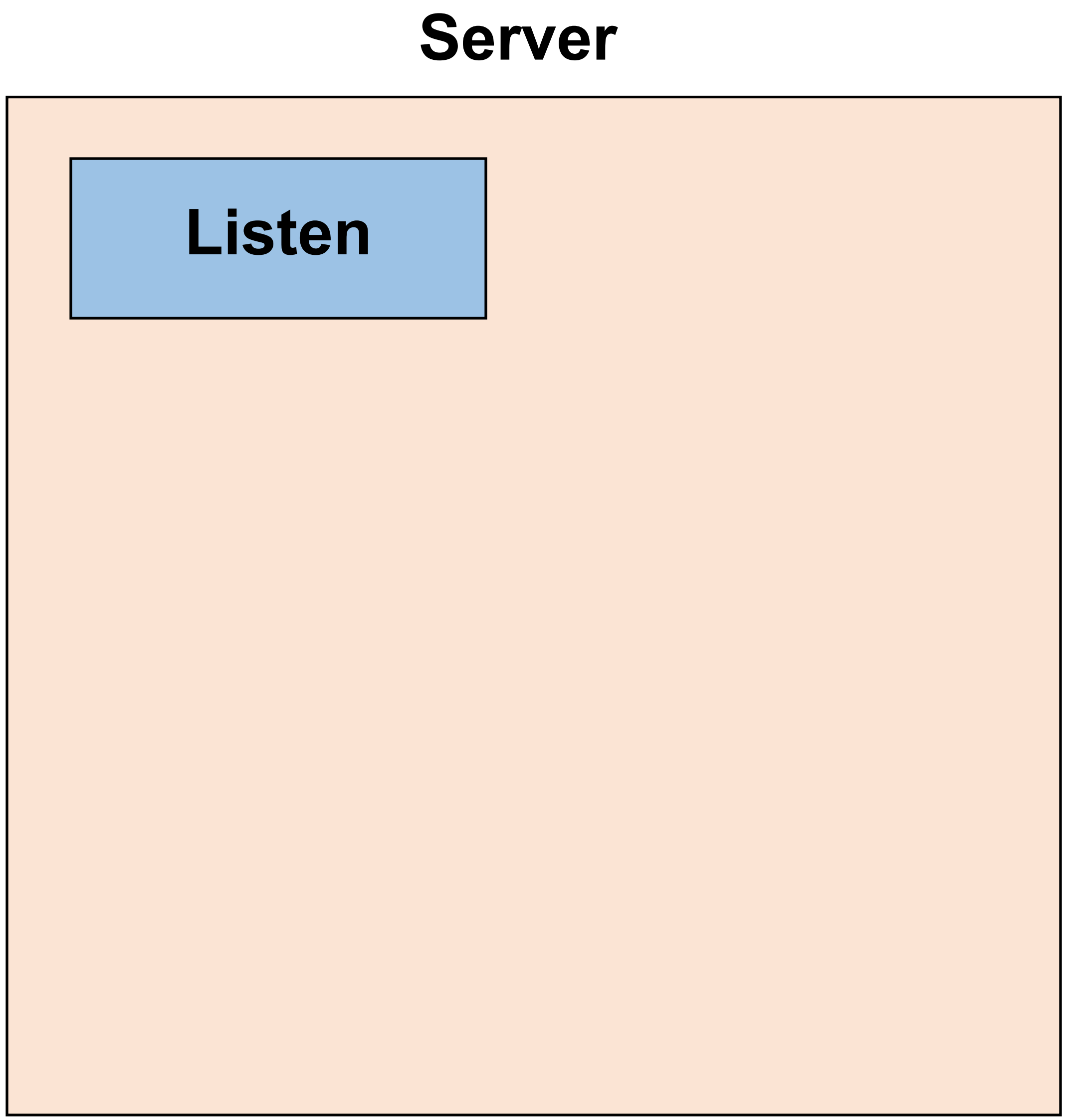
- **This code you create handles things like:**
  - Connections to the database
  - Handling client-side URL requests
  - Performing server-side processes
  - Authenticating user requests
  - Logging client requests

A Big Box



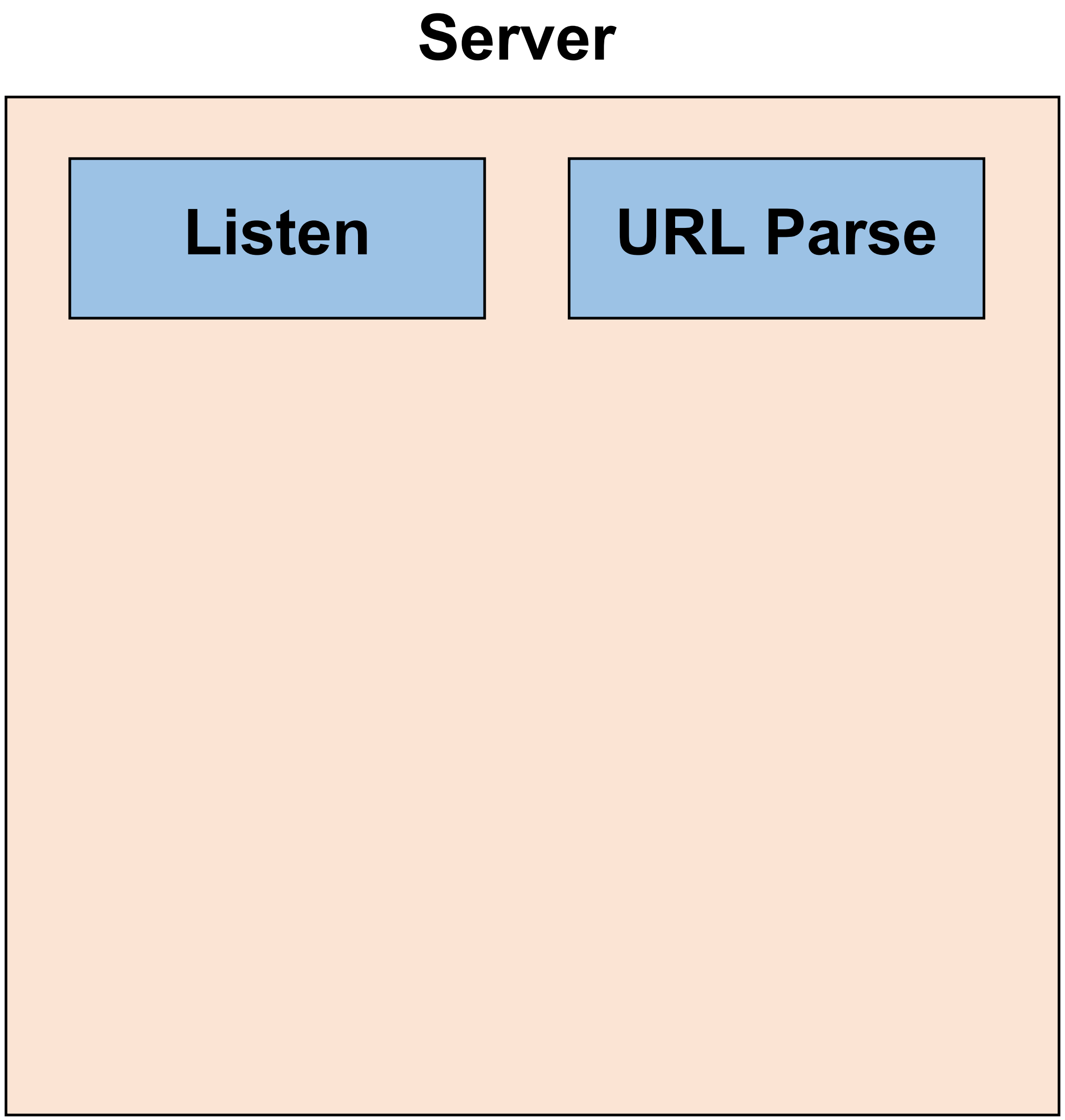
- Throughout this week… imagine your server to be a big, empty box.
- We will be adding code snippets and modules to give our big, empty box the powers to ***do*** stuff in response to all the requests that come in.

Inside the Box: Connections



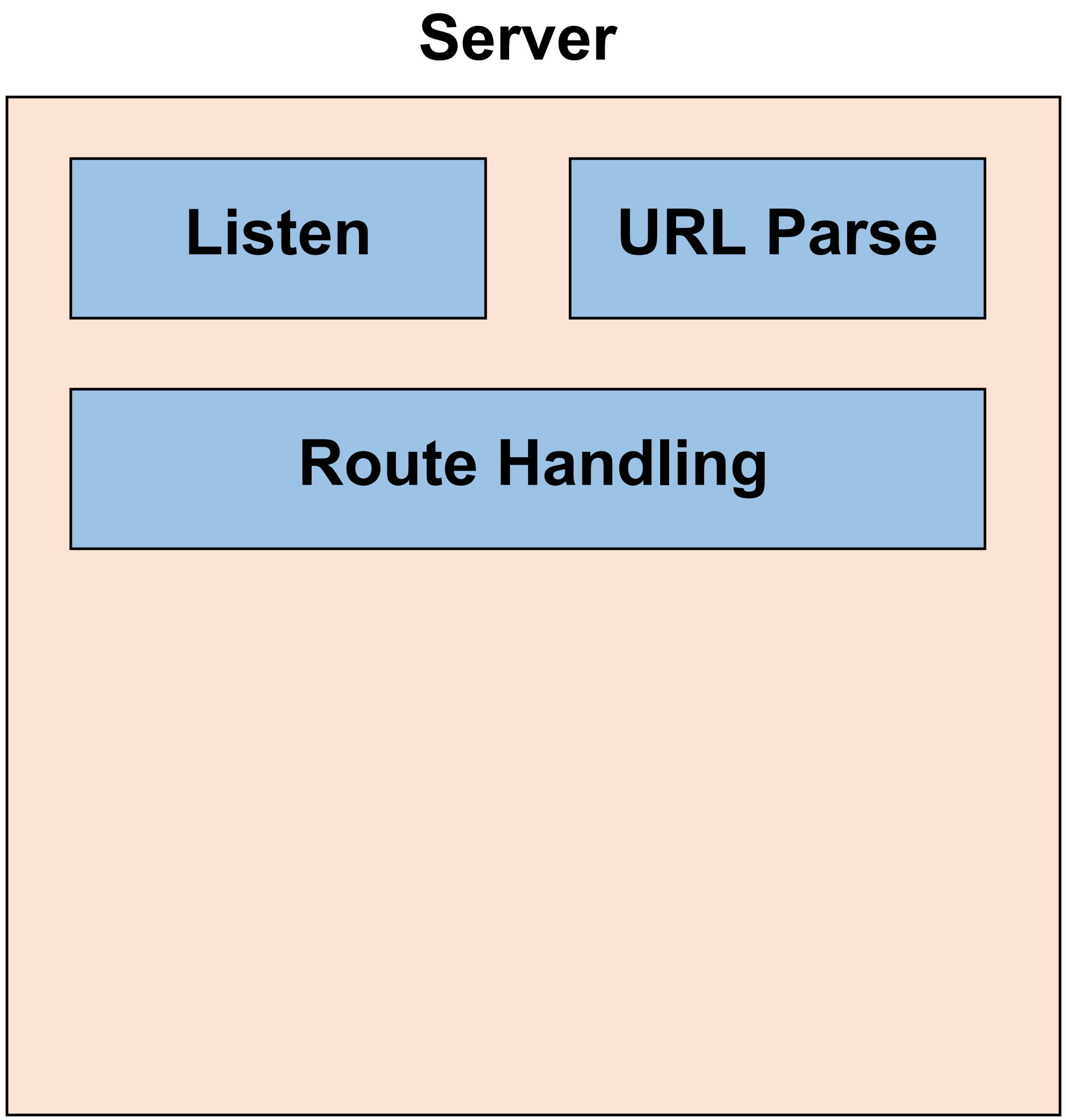
- We'll add listener such that the server can "begin" listening for requests.

Inside the Box: Parsing



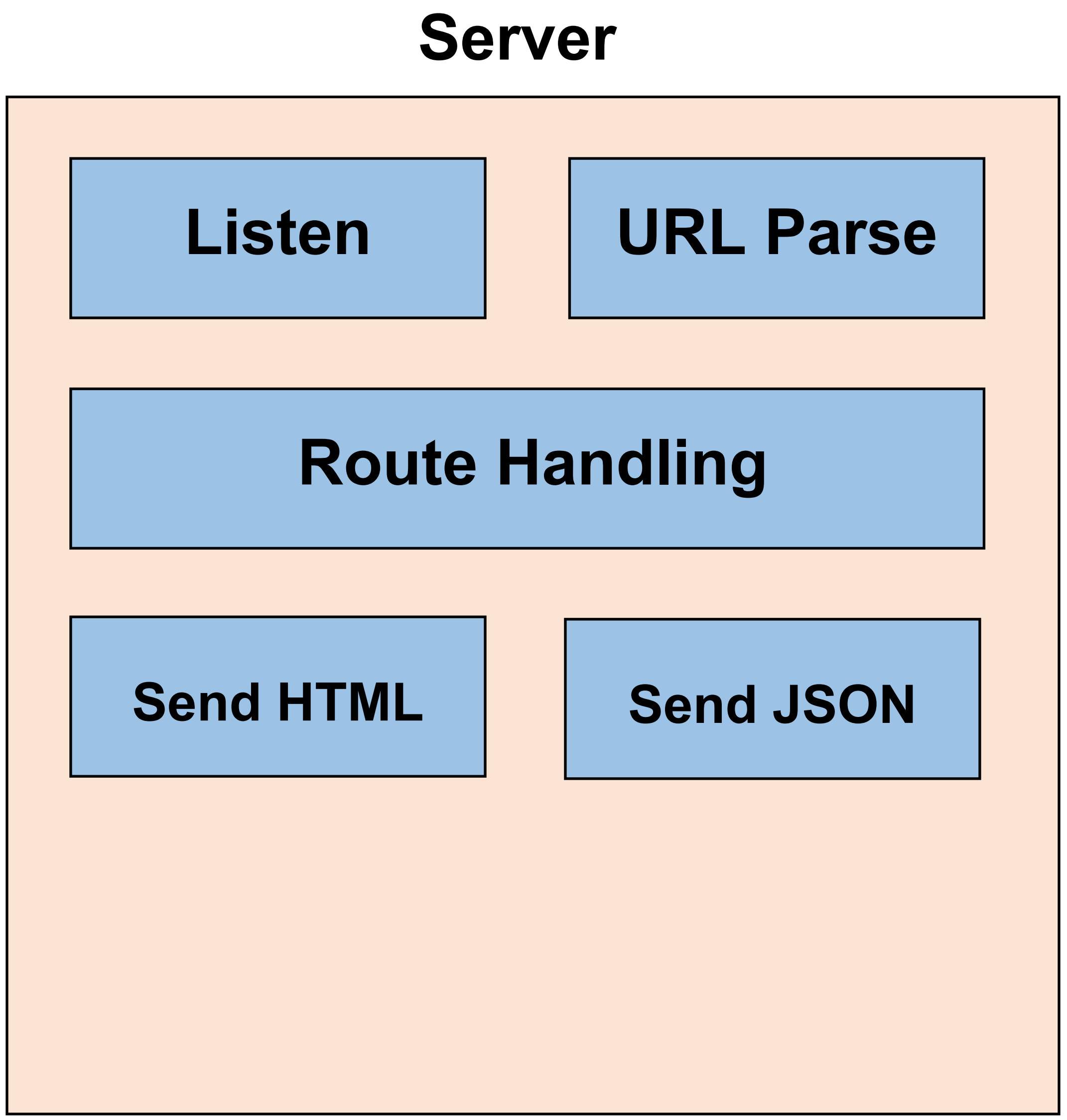
- We’ll give our server the ability to “parse” URLs that the user requests.

Inside the Box: Routing



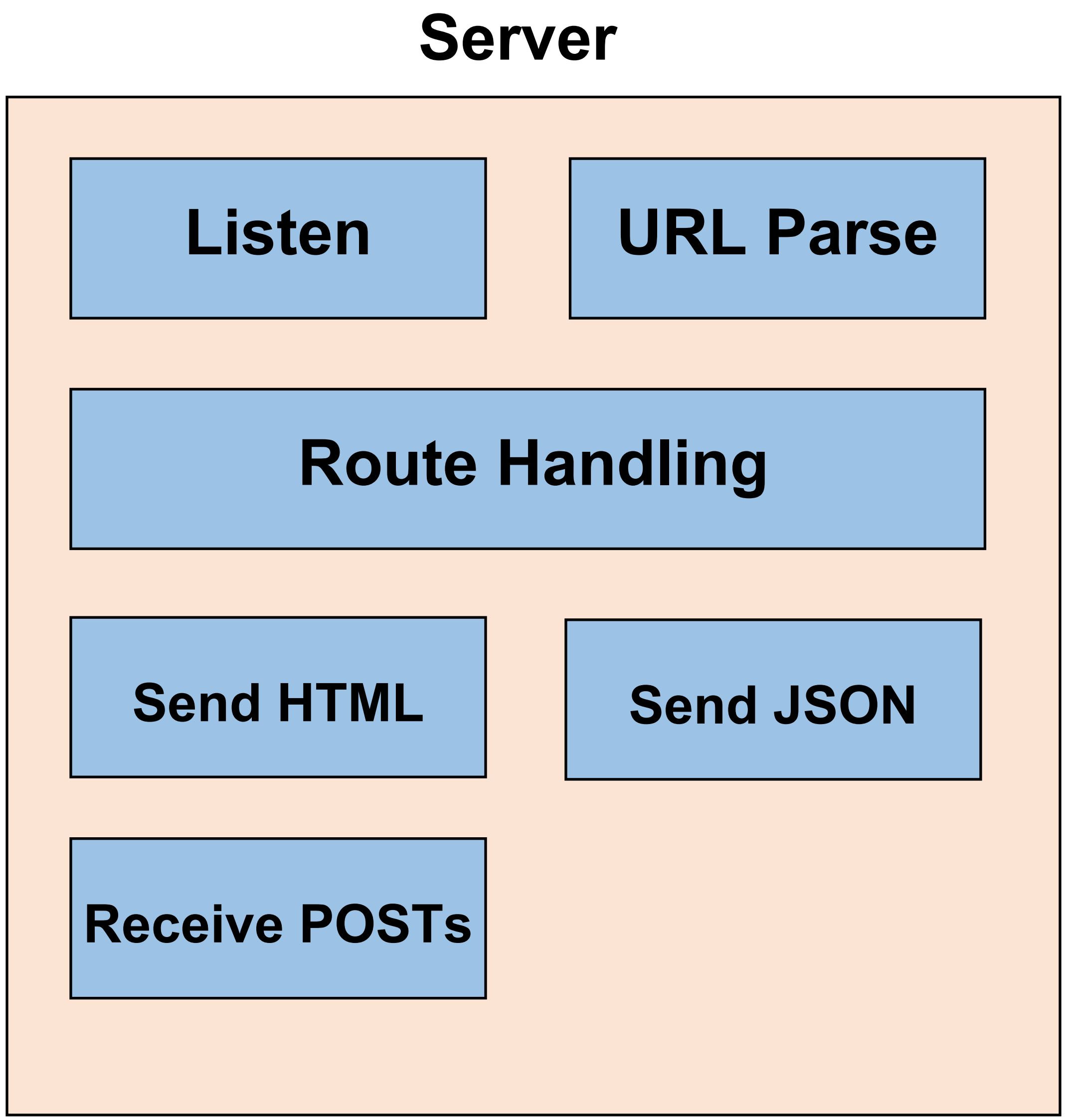
- Then based on the URL’s keywords, our server will be able to **route** (or direct the flow of logic to initiate other processes)

Inside the Box: Sending Files



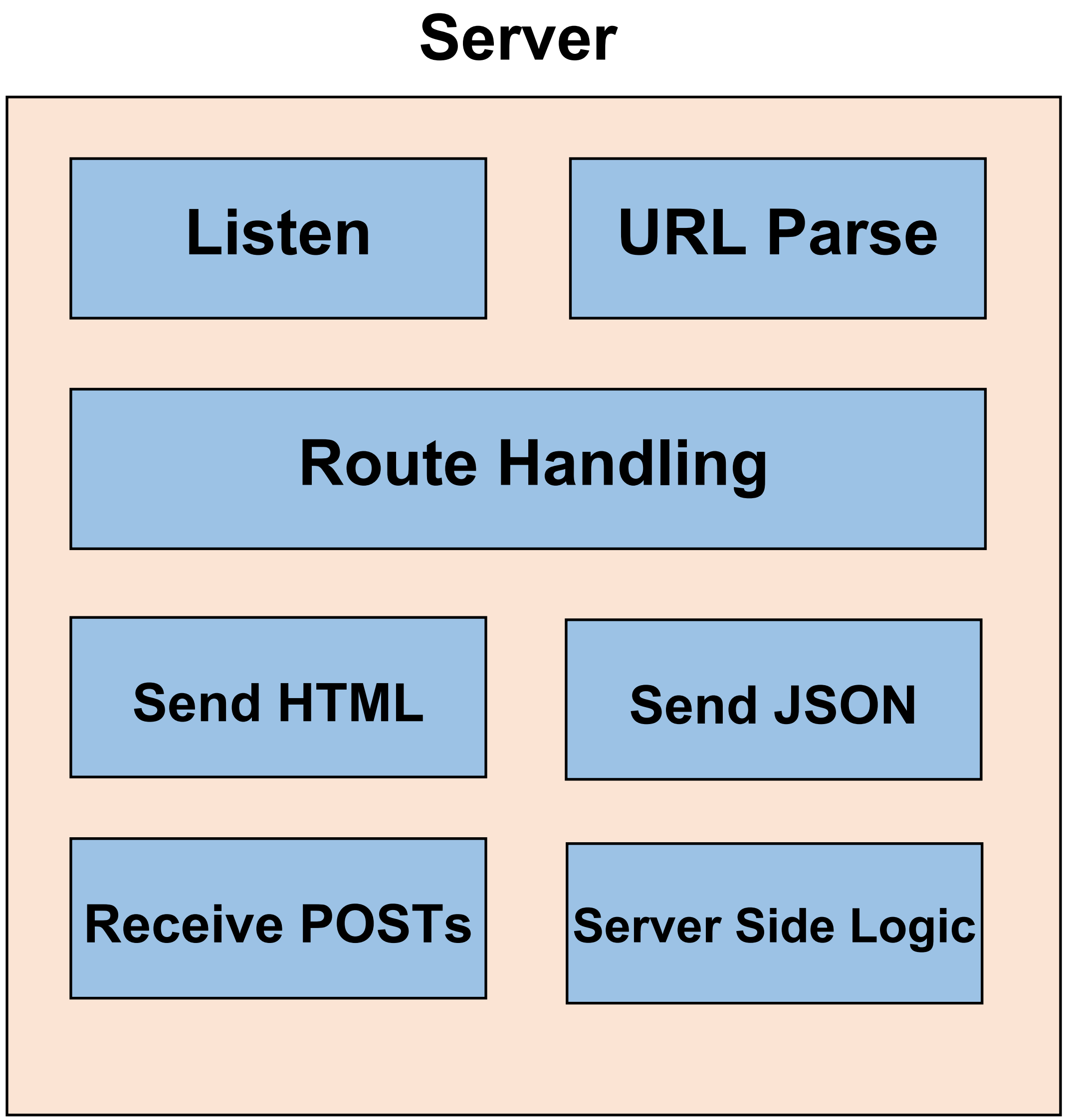
- This subsequent process may be to send an HTML file to be rendered or to send a JSON file to be rendered…

Inside the Box: Receiving Posts



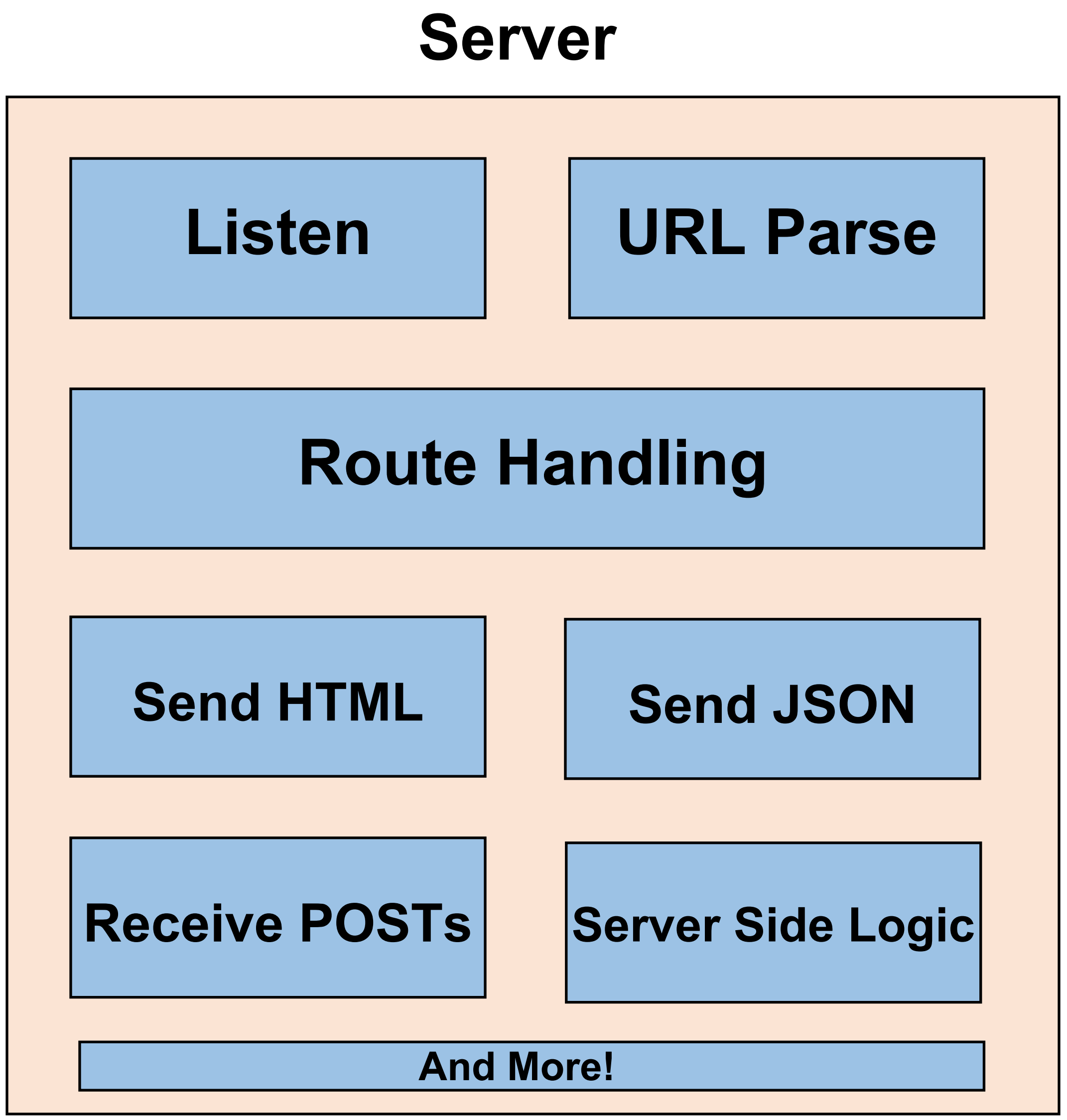
- We may also have a new module to handle receiving user’s POST requests (i.e. the data they send the server)

Inside the Box: Performing Logic



- We may also have complex server-side logic that we want to initiate in response to user's visiting a route endpoint or sending us data.

Inside the Box: And More!



- But it doesn’t stop there!
- We may add in functionality for authentication, logging requests, connecting to databases, and so much more.
- But always remember… we’re **<u>coding out these functionalities into our box!</u>**

Questions?

Let’s Get Coding!