Back End 1

Overview

- Back end basics
  - Server
  - Ports
  - Endpoints
  - HTTP Requests
  - HTTP Responses

Review

What is HTTP?

HTTP stands for Hyper Text Transfer Protocol and it is the protocol we follow when making requests and receiving responses from a client to a server.

Note: Client vs Server

Client This is usually a personal computer that talks to a server through the internet.

The client is also known as the front-end part of our application.

We will be using React for our front-end.

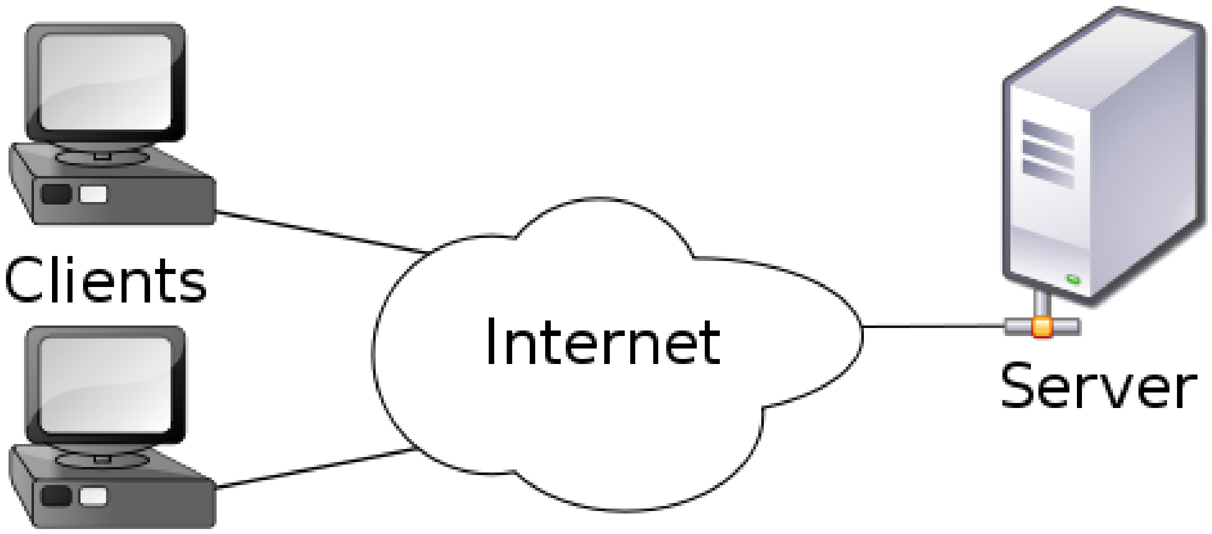
Server This is usually a computer that has clients that will talk to to request information from.

The server is also known as the back-end of the application.

We will be building our servers with Node.

Servers with Node

Client-Server Model



What is Node.js?

- Node is a JavaScript runtime environment.
- Traditionally, JavaScript could only be executed in browsers.
- Node is what allows JavaScript to be able to run outside of the browser.
- Node is built on Chrome’s V8 JavaScript engine.

Note: JavaScript

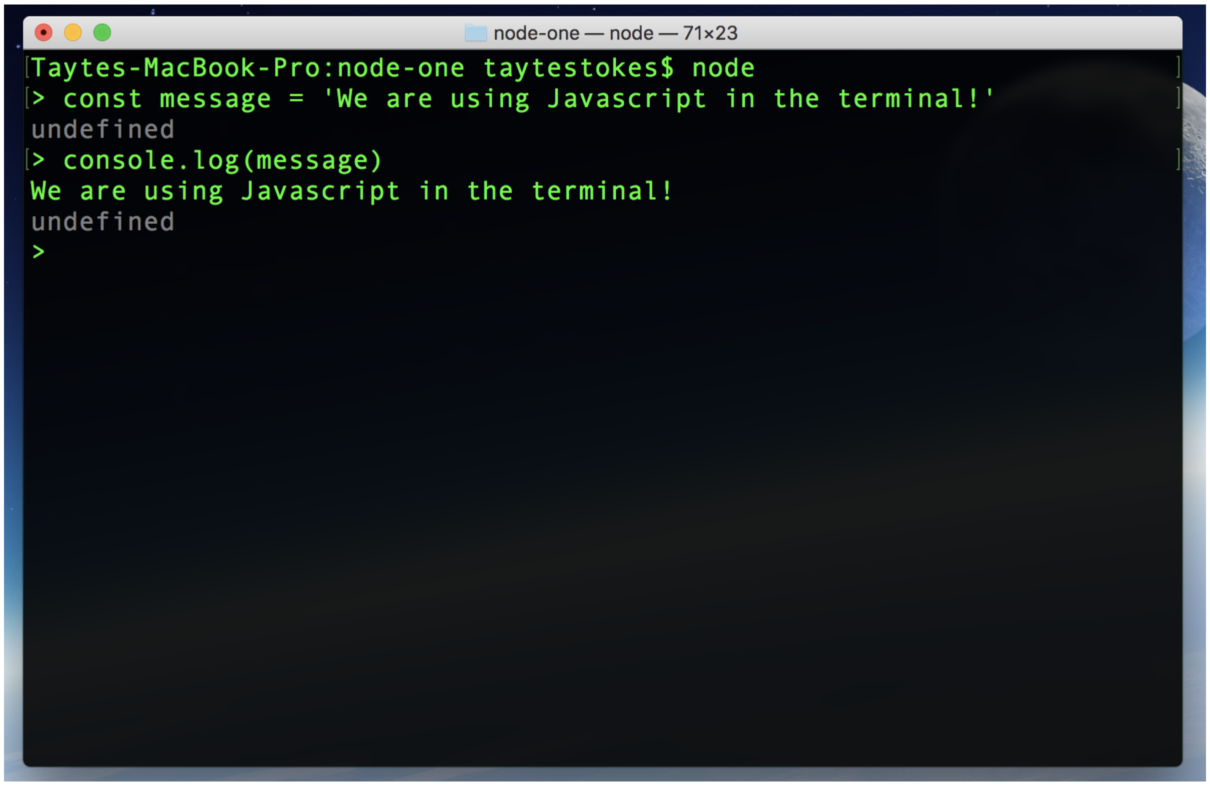
JavaScript has been around since 1995 and for a while it was only used inside of a browser environment, meaning that was the only place we could use it.

This meant that developers were having to code in multiple languages to create a front-end and back-end to their applications.

When Node came out, it allowed developers to write JavaScript code that runs directly on the computer process itself, rather than being confined to a browser environment. This means that Node can be used to write server-side applications that have full access to the operating system, file system, etc.

Node was written in C, C++, and JavaScript. It was built on top of the V8 JavaScript engine. This is the same engine that browsers like Google Chrome use.

Using Node in the Terminal



```
Taytes-MacBook-Pro:node-one taytestokes$ node
> const message = 'We are using Javascript in the terminal!'
undefined
> console.log(message)
We are using Javascript in the terminal!
undefined
>
```

Using Node with JS Files

Node can also be used to run JavaScript files.

To do this, use the command ‘node’ followed by the name of the JS file you want to run.

```
node index.js
```

The command above would run JavaScript file ‘index.js’.

Nodemon

- Note that Node only runs the file once.
- If any changes are made to your code, Node needs to be executed again.
- Fortunately, there is a package called Nodemon that can ‘watch’ our JavaScript files, and restart Node for us.

To install Nodemon, run the following command:

```
npm install -g nodemon
```

To run Nodemon with a file, just type:

```
nodemon index.js
```

NPM

What is NPM?

- We've used NPM a lot, but what is it?
- NPM stands for Node Package Manager.
- NPM is what gives us access to many great packages and libraries to use, such as Nodemon, axios, and more.
- https://docs.npmjs.com/about-npm/

Using NPM to Set Up a Project

When creating a Node.js project, we may want to add packages.

To accomplish that we need a `package.json` file and a `node_modules` folder.

We can create these by running the following NPM command:

```
npm init -y
```

Note: Y Flag

Without the '-y' flag, npm will prompt for some configuration settings. Adding the -y says yes to all the default settings.

Note: Node Modules Must Be in .gitignore

Make sure to put your node_modules/ folder in your .gitignore file to avoid pushing unnecessary files to github.

Express

Express is a minimal, flexible Node.js framework that is great for building servers.

To install Express, run the following in your terminal:

```
npm install express
```

Note: Node for Servers

One of the primary uses of Node is to act as a server.

A server will receive a request then perform some logic and return a response.

Express is the most popular framework we can use to build a server with node.

It will make it easier to matchup HTTP methods to by using endpoints to control what data a front end application can send and ask for.

Server Setup

To set up a server, Express needs to be required (imported) to the top of the file.

Then, you need to include the following lines like so:

```
const express = require('express')

const app = express()

app.use(express.json())  // When we want to be able to accept JSON.

app.listen(4040, () => console.log('Server running on 4040'))
```

Now we can run Nodemon with your server file to see it live!

Endpoints

Creating Endpoints

With a server, we can create endpoints.

Endpoints are where we define how our server, or API, can be interacted with.

To build an endpoint, we specify the type of request the server can receive, the endpoint URL for the request, and any handler functions we would like to fire when receiving that request.

```
app.get('/api/users', handlerFunction)
```

Endpoint Handler Functions

Handler functions determine what functionality should happen when a specific request is made to a given endpoint.

Handler functions will receive request (req) and response (res) object parameters.

```
app.get('/api/users', (req, res) => {
  // functionality goes here
})
```

Using Params

To create an endpoint that accepts params, a placeholder needs to be created on the server's endpoint. We denote that param with a colon. This works like a parameter in a function. When we "hit" the endpoint, we "send in" the value we want the param to have.

```
// server code
app.get('/api/users/:id')

// front end code
axios.get('/api/users/3')
  .then(...)
```

Using Queries

A quick look at how queries are created in a url:

https://www.mywebsite.com?user=john

Queries are signified by a `?`. After the question mark we have a key and a value, separated by an equal sign, in this syntax: key=value.

We can have as many queries as we want in a url, but we will still only use the single `?` and each additional query should just be separated by an `&`.

https://www.mywebsite.com?user=john&color=blue

When creating an endpoint that takes queries, NOTHING is needed to specify the query object in the endpoint, so it looks the same as any other endpoint.

The query object can be handled in the handler function off of the request object if the user created queries on their request.

```
app.get('/api/users')
```

The Request Object

- The request object is passed in by express as an argument to handler functions.
- It contains many things, but most importantly contains the params, query, and body objects that can be sent in a request.
- The params, query, and body can be accessed through the request object like so:

  ```
  app.put('/api/users/:id', (req, res) => {
    console.log(req.params)
    console.log(req.query)
    console.log(req.body)
  })
  ```

The Response Object

- The response object is the information that the server will send back to the client making the request.
- When sending a response object, it is important to include status codes.
- Status codes are a standardized short-hand syntax for relaying information about requests.

```
app.get('/api/users', (req, res) => {
  res.status(200).send(users)
})
```

- Available status codes here: https://http.cat/

The End