HTTP Servers

CODE FELLOWS

401
Advanced
Javascript

HTTP Protocol

TCP is a protocol that allows data to flow between computers

HTTP is an application layer specification built on top of TCP

It's the data that's being sent back and forth over TCP

In order for HTTP to function, a number of components must work in concert

1. URI (the requested resource)
2. METHOD (what are you asking the server to do)
3. HEADERS (instructions and metadata that shape the request parameters)

CODE FELLOWS

401
Advanced
Javascript

The URI

scheme:[**//**[**user**[**:password**]**@**]**host**[**:port**]][**/path**][?**query**][#**fragment**]

Scheme: http, ftp, gopher, file, mailto, etc. This identifies the type/protocol

User: Optional Username
Password: Optional Password

These are mainly used in SSH or secure connection URIs

Host: The server name you are attempting to connect to
Port: A specific port on that server (80 is the default for web, and can be omitted)

Path: A path to the specific resource you are requesting

- Could be a file, an application pointer, or a RESTful address

Query: Parameters to be sent into the resource (?this=that&foo=bar)
Fragment: Generally used to identify a part on a page, but has other uses in Client Side JS

https://en.wikipedia.org/wiki/Uniform_Resource_Identifier

401
Advanced
Javascript

METHODS

GET	Retrieves a resource (read only)
HEAD	Asks for a response identical to that of a GET request, but without the response body.
POST	Sends “unlimited” data to the resource.
PUT	Requests that the enclosed entity be stored under the supplied URI
DELETE	The DELETE method deletes the specified resource.
TRACE	The TRACE method echoes the received request
OPTIONS	The OPTIONS method returns the HTTP methods that the server supports for the URI
CONNECT	The CONNECT method converts the request connection to a transparent tunnel
PATCH	The PATCH method applies partial modifications to a resource

https://en.wikipedia.org/wiki/Hypertext_Transfer_Protocol

CODE FELLOWS

401
Advanced
Javascript

HEADERS

There are about 100 of these …
Sent as key/value pairs as a part of the request to an HTTP Server

Accept: What format of response I will accept

Content Type: What format of response I am sending (text/html, application/json)

Content Length: Character count of the enclosure

Host, User-Agent, Cookies

https://en.wikipedia.org/wiki/List_of_HTTP_header_fields

401
Advanced
Javascript

The REQUEST

Once you connect to the server and port, you'll issue a series of commands, followed by a newline.

First, the METHOD, Resource, and HTTP Version:

GET /r/javascript HTTP/1.1

Then, any combination of meaningful header key/value pairs

Host: www.reddit.com
Accept: application/json

2 Newlines completes your request

CODE FELLOWS

401
Advanced
Javascript

The RESPONSE

Once you connect to the server and port, you'll issue a series of commands, followed by a newline.

First, the METHOD, Resource, and HTTP Version:

GET /r/javascript HTTP/1.1

Then, any combination of meaningful header key/value pairs

Host: www.reddit.com

Accept: application/json

2 Newlines completes your request

CODE FELLOWS

401
Advanced
Javascript

The BUCKET Protocol

- Client makes a request, formatted with headers
- Server receives, parses, retrieves the resource
- Server returns a response, formatted perfectly
- Closes the connection
- Unlike TCP which keeps that socket open, HTTP closes it

401
Advanced
Javascript

Anatomy of a (node) WebServer

So, what is a web server? It's a content delivery system. It's not just index files or some json, or images. They are specific requests for resources and then sending some content back.

That content could be static or dynamic, based on the request, params, headers, etc.

When we build our own HTTP server, we don't want to dork with all of that complexity and knowing it. We just want to deal with the raw request and responses

The server can (needs to) do logic based on any combination of the: Method, Headers, or Body

The server we build will handle some custom routing:

A "GET" on "/" and a "POST" on "/json" should work differently and independently

401
Advanced
Javascript

Cool tools and utilities ...

Some of the cool stuff we'll be using when building our server

Node "url" and "querystring" modules, which allow us to parse the URL components
Node "superagent" module which we can use to test our running web server
Node "dotenv" module which lets us work with environment variables from a file
The "jest" testing library
Promises (finally)

nodemon -- a file watcher for node servers. Huge productivity booster