• High level: what happens when you visit a URL in browser • Explain what IP and DNS are • Describe the different parts of a URL • Describe the request / response cycle An Introduction • Compare **GET** vs **POST** requests What Happens When... How the Web Works Networks **An Introduction** Networks Hostnames **What Happens When...** IP Addresses 127.0.0.1 When I type http://site.com/some/page.html into a browser, what really happens? Protocols This is a common interview question for software engineers. Hostname **How the Web Works** Resource The internet is complicated. **Query String** Really, really complicated. Fortunately, to be a software developer, you only need to know a bit. For people who want to work in "development operations," or as a system administrator, it's typical to have to **Browsers and Servers** learn more about the details here. Request and Response What's in a Request? What's in a Response **Networks** Watch a Request/Response Response Codes A network is a set of computers that can intercommunicate. Serving Over HTTP The internet is just a really, really big network. Serving Over HTTP The internet is made up of smaller, "local" networks. Multiple Requests Sample HTML **Hostnames** Image We often talk to servers by "hostname" - *site.com* or *computer-a.site.com*. Javascript Hey, That's a Lot of Work! That's just a nickname for the server, though - and the same server can have many hostnames. Trying on Command Line **IP Addresses** Curl (OSX) On networks, computers have an "IP Address" — a unique address to find that computer on the network. Methods: GET and POST IP addresses look like 123.77.32.121, four numbers (0-255) connected by dots. **GET vs POST** There are a lot of advanced edges here that make this more complicated, but most of these details aren't Sample GET Requests important for software engineers: Sample POST Request HTTP Methods • there another whole way to specify networks, "IPv6", that use a different numbering scheme. • some computers can have multiple IP addresses they can be reached by Further Study Further Study • under some circumstances, multiple computers can share an IP address and have this be handled by a special kind of router. If you're interested in system administration details, you can learn about this by reading about "Network Address Translation". 127.0.0.1 127.0.0.1 is special — it's "this computer that you're on". In addition to their IP address on the network, all computers can reach themselves at this address. The name *localhost* always maps to 127.0.0.1. **URLs** http://site.com/some/page.html?x=1 turns into: Resource **Port** Query **Protocol** Hostname http 80 /some/page.html ?x=1 site.com **Protocols Port Protocol** Hostname Resource Query /some/page.html ?x=1 80 http site.com "Protocols" are the conventions and ways of one thing talking to another. http Hypertext Transfer Protocol (standard web) (How browsers and servers communicate) https HTTP Secure (How browsers and servers communicate with encryption) ftp File transfer protocol (An older protocol for sending files over internet) There are many others, but these are the common ones. In this lecture, we'll be focusing only on HTTP. Hostname **Protocol** Hostname **Port** Resource Query /some/page.html ?x=1 http site.com 80 DNS (domain name service) turns this into an IP address So **site.com** might resolve to 123.45.67.89 **Port Port** Hostname Resource Query **Protocol** 80 /some/page.html ?x=1 http site.com • Every server has 65,535 unique "ports" you can talk to Services tend to have a default port • For HTTP, is port 80 • For HTTPS, is port 443 • You don't have to specify in URL unless you want a different port • To do: http://site.com:12345/some/page.html **Resource Protocol** Hostname Port Resource Query http 80 /some/page.html ?x=1 site.com • This always talks to some "web server" program on the server • For some servers, may just have them read an actual file on disk: /some/page.html • For many servers, "dynamically generates" a page **Query String Protocol** Hostname Port Resource Query http 80 /some/page.html ?x=1 site.com • This provides "extra information" — search terms, info from forms, etc • The server is provided this info; might use to change page • Sometimes, JavaScript will use this information in addition/instead Multiple arguments are separated by &: ?x=1&y=2 • Argument can be given several times: ?x=1&x=2 So http://site.com/some/page.html?x=1 means • Turn "site.com" into 123.45.67.89 • Connect to 123.45.67.89 • On port 80 (the default) • Using the HTTP protocol • Ask for /some/page.html • Pass along query string: x = 1 **DNS** "I want to talk to site.com" Err: Not Found where is site.com? Computer DNS Your ISP Router Browser Servers Cache site.com is at 123.45.67.89 Unix (and OSX and Linux) systems ship with a utility, *dig*, which will translate a hostname into an IP address for you, and provide debugging information about the process by which it answered this. \$ dig site.com ; <<>> DiG 9.8.3-P1 <<>> site.com ;; global options: +cmd ;; Got answer: ;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 959 ;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDTL: 0 ;; QUESTION SECTION: ;site.com. IN A ;; ANSWER SECTION: 631 IN A 123.45.67.89 # Answer site.com. ;; Query time: 28 msec ;; SERVER: 10.0.1.1#53(10.0.1.1) ;; WHEN: Mon Apr 20 00:48:16 2018 ;; MSG SIZE rcvd: 46 **Browsers and Servers Request and Response** When you point your browser to a webpage on a server, your browser makes a *request* to that server. This is almost always a *GET* request and it contains the exact URL you want. Request GET / HTTP/1.1 Headers Server Web Browser The server then responds with the exact HTML text for that page: Response Headers <html>...<html> Server Web Browser It's often the case, though, that the web server itself will have to do some work to get the page you want, often interacting with other things, such as database servers. Request GET / HTTP/1.1 Headers Server Web Browser **Internal Work** DB Server Web Browser Server And then it can give back the response you want: Response Headers <html>...<html> Server Web Browser What's in a Request? • Method (ex: **GET**) • HTTP protocol version (almost always 1.1) Resource URL you want Headers • Hostname you're asking about • Date your browser thinks it is • Language your browser wants information in Any cookies that server has sent • And more! What's in a Response • HTTP protocol version (almost always 1.1) • Response Status Code (200, 404, etc) Headers Content Type (typically text/html for web pages) • Date/time the server thinks it is • Any cookies server wants to set • Any caching information • And more! Watch a Request/Response \$ telnet 123.45.67.89 80 Trying 123.45.67.89... Connected to site.com. Escape character is '^]'. **GET / HTTP/1.1** Host: site.com HTTP/1.1 200 OK Date: Mon, 20 Apr 2018 07:09:16 GMT # Date Header Server: Apache Content-Type: text/html # This is HTML content <!doctype html> <html> <head> <title>The Site</title> </head> <body> </body> </html> **Response Codes** 200 OK 301 What you requested is elsewhere 404 Not Found **500** Server had an internal problem There are more, but these are the most important ones. **Serving Over HTTP** Just opening an HTML file in browser uses file protocol, not http Some things don't work same (esp security-related stuff) It's often useful to start a simple HTTP server for testing You can start a simple, local HTTP server with Python: \$ python3 -m http.server Serve files in current directory (& below): \$ python3 -m http.server Serving HTTP on 0.0.0.0 port 8000 (http://0.0.0.0:8000/) ... **Multiple Requests Sample HTML** demo/demo.html <!doctype html> <html> <head> k rel="stylesheet" href="demo.css"> </head> <body> <h1>Hi There!</h1> <script src="demo.js"></script> </body> </html> CSS demo/demo.html <!doctype html> <html> <head> k rel="stylesheet" href="demo.css"> </head> <body> <h1>Hi There!</h1> <script src="demo.js"></script> </body> </html> Connects to **site.com** on port 80 and requests: GET /demo.css HTTP/1.1 Host: site.com Date: [date browser thinks it is] Cookie: [any cookies browser is storing for that host] Response: HTTP/1.1 200 OK Date: [date server thinks it **is**] Content-Type: text/css Cookie: [any cookies server wants you to set] body { background-color: lightblue; **Image** demo/demo.html <!doctype html> <html> <head> k rel="stylesheet" href="demo.css"> </head> <body> <h1>Hi There!</h1> <script src="demo.js"></script> </body> </**html>** Connects to *tinyurl.com* on port 80 and requests: GET /rithm-logo HTTP/1.1 Host: tinyurl.com Date: [date browser thinks it **is**] Cookie: [any cookies browser is storing for tinyurl.com] **Javascript** demo/demo.html <!doctype html> <html> <head> k rel="stylesheet" href="demo.css"> </head> <body> <h1>Hi There!</h1> <script src="demo.js"></script> </body> </html> Connects to **site.com** on port 80 and requests: GET /demo.js HTTP/1.1 Host: site.com Date: [date browser thinks it **is**] Cookie: [any cookies browser is storing for that host] Hey, That's a Lot of Work! • Yes, it is • Requesting 1 webpage often involves many separate requests! • Browsers issue these requests asynchronously • They'll assemble the final result as requests come back You can view this in the browser console → Network **Trying on Command Line Curl (OSX)** OSX systems come with a utility, curl, which will make an HTTP request on the command line. \$ curl -v site.com * Rebuilt URL to: http://site.com/ * Trying 123.45.67.89... * Connected to site.com (123.45.67.89) port 80 (#0) > GET / HTTP/1.1 > User-Agent: curl/7.41.0 > Host: site.com < HTTP/1.1 200 OK < Date: Mon, 20 Apr 2018 08:28:50 GMT</pre> < Server: Apache/2.4.7 (Ubuntu) < Vary: Accept-Encoding < Transfer-Encoding: chunked < Content-Type: text/html; charset=UTF-8 <!doctype html> <html> <head> <link rel="stylesheet" href="demo.css"> </head> Hey... **Everything is a string! Methods: GET and POST GET vs POST** • **GET**: requests without side effects (ie, don't change server data) • Typically, arguments are passed along in query string • If you know the arguments, you can change the URL • Entering-URL-in-browser, clicking links, and *some* form submissions • **POST**: requests with side effects (ie, change data on server) Typically, arguments sent as body of the request (not in query string) • Some form submissions (but never entering-URL-in-browser or links) • Always do this if there's a side-effect: sending mail, charge credit card, etc "Are you sure you want to resubmit?" **Sample GET Requests** About Us Search For Lemurs! <!-- will submit to URL like /search?q=value-in-input --> <form action="/search" method="GET"> Search for <input name="q"> <button type="submit">Search!</button> </form> **Sample POST Request** POST requests are always form submissions: <!-- will submit to URL add-comment, with value in body --> <form action="add-comment" method="POST"> <input name="comment"> <button type="submit">Add</button> </form> **HTTP Methods GET** and **POST** are "HTTP methods" (also called "HTTP verbs") They're the most common, by far, but there are others **Further Study** • How the Internet Works: A Code.org Video Series How Does the Internet Work? • A cute webcomic about how DNS works

Colt Courses

How the Web Works

« Back to Homepage

Goals

URLs

URLs

Port

So

DNS

DNS

CSS

Hey...

Goals

How the Web Works

Goals