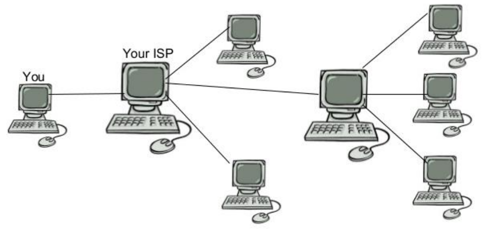
The Internet & Web Development

- **As we discussed in Week 1 of this course, a computer network is a set of computers that provide a standard communication path between computers**
  - A simplified view of a network might be:

  - But the reality is there is even more interconnection:
    - The Internet is a network of networks!

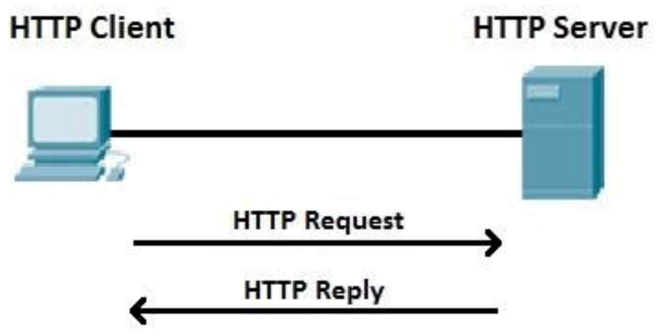


  - Everything you access on the Web, must be *downloaded* to your computer in some form in order to be *viewed* on your computer
    - Every time you tell your browser to navigate to a website, the browser must download the files (HTML, CSS, JavaScript, images, and fonts) needed to display that page correctly in your browser
    - When you use a live server (like the one in your code editor), you are simulating a server that would normally be somewhere else online, and the browser is *still* downloading your website from that local server. Largely, your browser continues to treat websites from local servers as though they were online.

Communication Protocols

- **When two computers try to communicate with each other, they have to use some mutually understood language: a protocol**
  - A "protocol" is *like* a programming language, but is far, far simpler and far, far smaller; it really is just a handful of rules about how communication should work
- **One protocol used on the Web is HTTP (Hypertext Transfer Protocol)**
  - There is also a more secure version, called HTTPS, which is now more common; it is basically just HTTP, but wrapped up by a layer of encryption and other security features to make a delicious Web burrito which isn't as likely to leak the important stuff all over the place
- **Other Internet protocols you may have heard of include SMTP (email) and FTP (file transfer)**
  - As well as binary data protocols, including TCP and UDP – about which many memes have been made
  - But to be a web programmer during this developer academy… the only thing you need to know is HTTP
- **And you don't really even need to understand the low-level details of HTTP**
  - You just need to understand the basics of the HTTP request/response cycle

Request-Response Cycle



- **When a user enters a URL (address) in a browser's address bar and presses GO, the browser sends an HTTP request to the specified server**
  - How does it find the server? More on this soon!

- **The server listens for requests and figures out how to answer them when they come in**
  - The word 'server' refers to both the ***software*** that's running and the ***actual computer*** (***hardware***) that's running the software

- **The server generates a response and returns it to the browser**
  - HTTP responses are typically a mixture of HTML, CSS and JavaScript

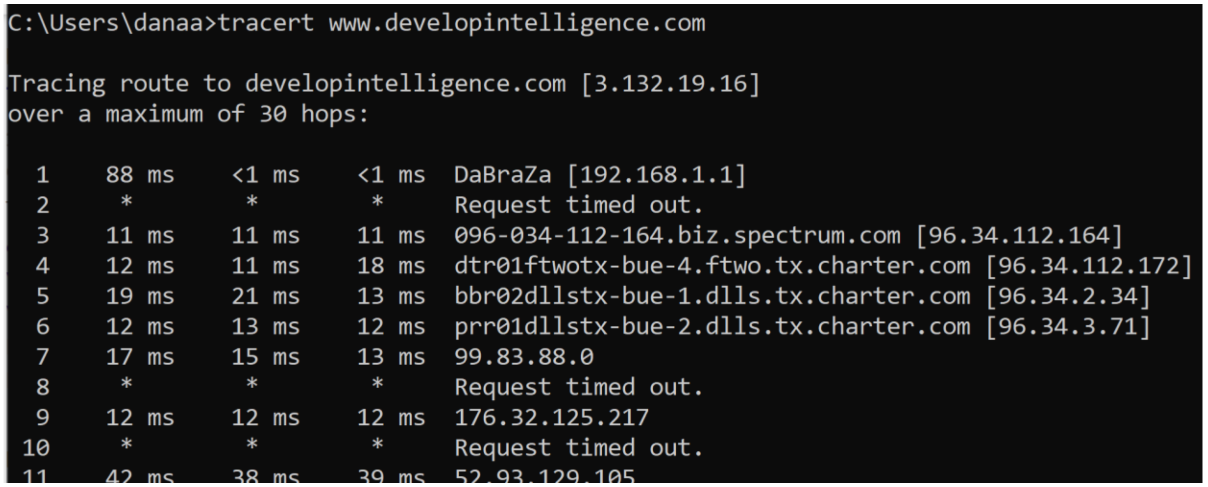
- **The browser retrieves the response, parses it, executes some JavaScript and then 'renders' it visually**

Domain Name and IP Address

- **When an individual or organization wants to create their own web site, they reserve a domain name like `www.xyz.com` from a domain name registrar**
- **A *domain name registrar* is a business that manages the reservation of domain names and the IP addresses associated with those domain names**
  - You've probably seen a commercial for the largest domain name registrar out there: `godaddy.com`
- **All domain name registrars work with the Internet Corporation for Assigned Names and Numbers (ICANN) to ensure domain names are unique**
  - ICANN is the non-profit organization that oversees the registry of domain names and IP addresses
- **An IP address (IPv4) is a sequence of four sets of numbers separated by dots such as 3.132.19.16 and is what identifies a computer on a network**
  - IP addresses of web servers on the Internet must be unique
- **Domain names make it easier to access websites without having to memorize and enter numeric IP addresses**
  - For example, Google's domain name is `google.com` and their IP address is 8.8.8.8

How Does the Communication from Client to Server Work?

- **When you enter a URL into your browser's address bar and hit go, the browser checks its local cache to see if the website has been visited recently**
  - If so, it will know the IP address of the website
- **Otherwise, the request goes to the Recursive DNS Servers established by your ISP**
  - A complicated search process ensues, but eventually the DNS server will associate the domain name with an IP address and forward the request on to the web server at that IP address
- **The request's path between the client and server isn't usually direct**
  - It will pass between a bunch of different routers on its way to the server and back again
  - Each request (and even individual packets in the same request) may follow different routes
- **Each packet has a 'source' and 'destination' IP address the same way that a physical envelope has a 'to' and 'from' address**
  - You can even see a route from your machine to any other machine using the `tracert` (Windows) or `traceroute` (Linux/macOS) program from the terminal



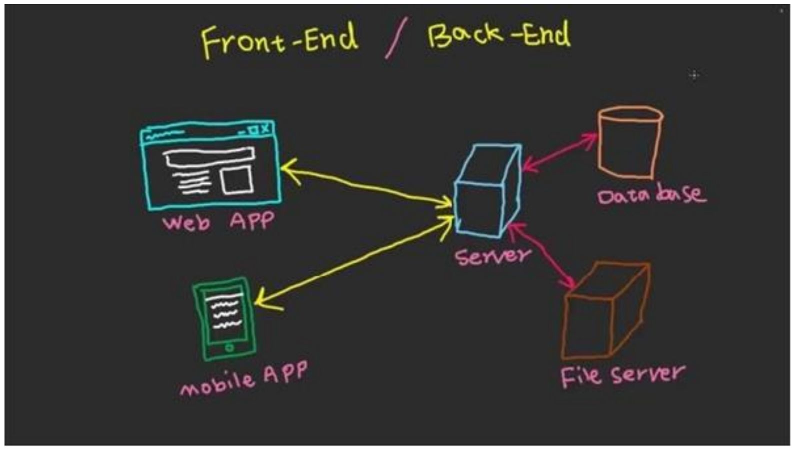
```
C:\Users\danaa>tracert www.developintelligence.com

Tracing route to developintelligence.com [3.132.19.16]
over a maximum of 30 hops:

  1    88 ms    <1 ms    <1 ms  DaBraZa [192.168.1.1]
  2     *        *        *     Request timed out.
  3    11 ms    11 ms    11 ms  096-034-112-164.biz.spectrum.com [96.34.112.164]
  4    12 ms    11 ms    18 ms  dtr01ftwotx-bue-4.ftwo.tx.charter.com [96.34.112.172]
  5    19 ms    21 ms    13 ms  bbr02dllstx-bue-1.dlls.tx.charter.com [96.34.2.34]
  6    12 ms    13 ms    12 ms  prr01dllstx-bue-2.dlls.tx.charter.com [96.34.3.71]
  7    17 ms    15 ms    13 ms  99.83.88.0
  8     *        *        *     Request timed out.
  9    12 ms    12 ms    12 ms  176.32.125.217
 10     *        *        *     Request timed out.
 11    42 ms    38 ms    39 ms  52.93.129.105
```

- **Unless you work as a network administrator, this low-level route information will probably not be of interest to you**

Front-end Development



- **Front-end development is the name people often use for that part of development that creates and programs the user interface elements**
  – In this portion of the code academy, we are specifically talking about being a front-end web developer

- **’Front-end’ also means ’client-side’ (or the user’s computer)**
  – And because our web applications run in the browser, we need to understand a bit about how browsers and the internet work

- **And we will spend a lot of time studying HTML, CSS, and JavaScript because they are the core technologies of web front ends**

Browsers, a.k.a. our gracious hosts

- **Desktop front-end developers create user interfaces that run in a self-contained program**
  - Think about your computer's calculator or basic text editor

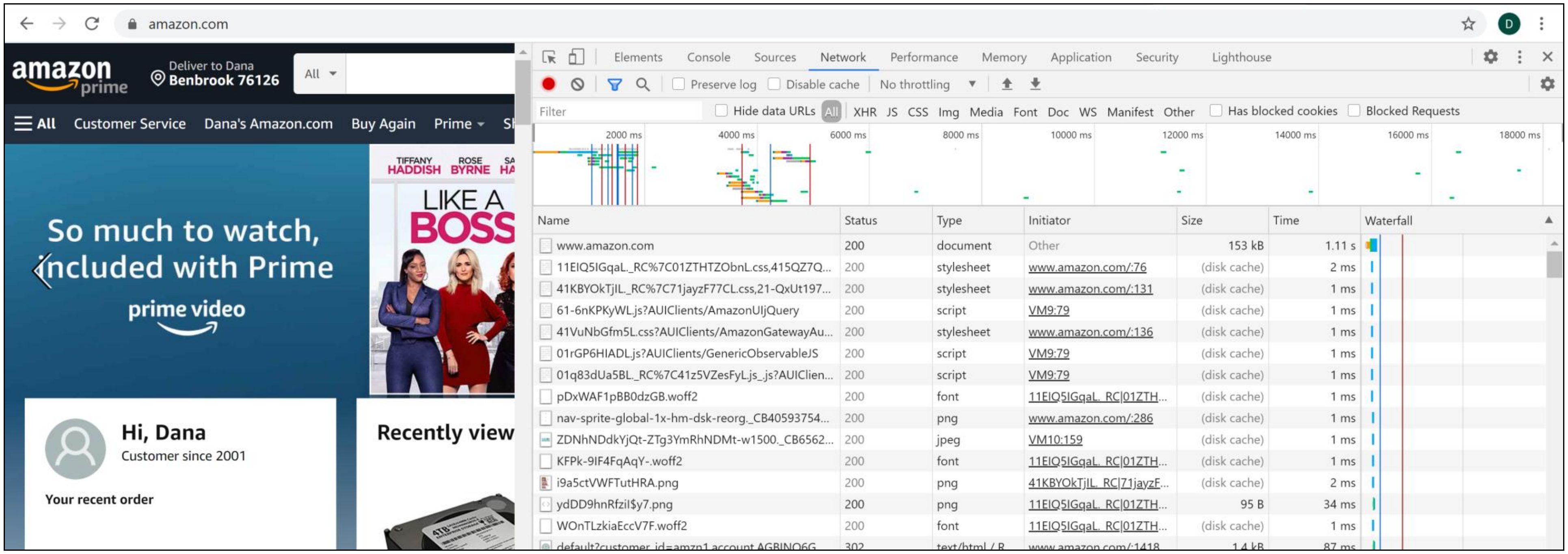
- **Mobile application developers create user interfaces that run in apps installed on the device from stores like Google Play or the Apple Store**

- **Web front-end developers create applications that run in browsers**
  - Browsers designed to retrieve and display web pages

- **Even though the internet and browsers have changed some over the years, the general flow for *what a user does with a web browser* has actually changed very little**
  - When you type in a URL, you're requesting that the client make an HTTP request to the specified server, retrieve the response, and then 'render' it visually

Making an HTTP Request

- **When the user enters a URL and clicks Go, or clicks a link on a page, the browser generates an HTTP request that goes to the specified server**
  – You can actually view requests in most modern web browsers by taking a look at the ’network’ tab in developer tools



Name	Status	Type	Initiator	Size	Time	Waterfall
www.amazon.com	200	document	Other	153 kB	1.11 s	
11EIQ5IGqaL._RC%7C01ZTHTZObnL.css,415QZ7Q...	200	stylesheet	www.amazon.com/:76	(disk cache)	2 ms	
41KBYOkTjIL._RC%7C71jayzF77CL.css,21-QxUt197...	200	stylesheet	www.amazon.com/:131	(disk cache)	1 ms	
61-6nKPKyWL.js?AUIClients/AmazonUIjQuery	200	script	VM9:79	(disk cache)	1 ms	
41VuNbGfm5L.css?AUIClients/AmazonGatewayAu...	200	stylesheet	www.amazon.com/:136	(disk cache)	1 ms	
01rGP6HIADL.js?AUIClients/GenericObservableJS	200	script	VM9:79	(disk cache)	1 ms	
01q83dUa5BL._RC%7C41z5VZesFyL.js_.js?AUIClien...	200	script	VM9:79	(disk cache)	1 ms	
pDxWAF1pBB0dzGB.woff2	200	font	11EIQ5IGqaL._RC|01ZTH...	(disk cache)	1 ms	
nav-sprite-global-1x-hm-dsk-reorg._CB40593754...	200	png	www.amazon.com/:286	(disk cache)	1 ms	
ZDNhNDdkYjQt-ZTg3YmRhNDMt-w1500._CB6562...	200	jpeg	VM10:159	(disk cache)	1 ms	
KFPk-9IF4FqAqY-.woff2	200	font	11EIQ5IGqaL._RC|01ZTH...	(disk cache)	1 ms	
i9a5ctVWFTutHRA.png	200	png	41KBYOkTjIL._RC|71jayzF...	(disk cache)	2 ms	
ydDD9hnRfziI$y7.png	200	png	11EIQ5IGqaL._RC|01ZTH...	95 B	34 ms	
WOnTLzkiaEccV7F.woff2	200	font	11EIQ5IGqaL._RC|01ZTH...	(disk cache)	1 ms	
default?customer_id=amzn1.account.AGBINQ6G	302	text/html / R	www.amazon.com/:1418	1.4 kB	87 ms	

HTTP Request - Under the Hood

- **Let's take a look at the key fields of a pretty standard web request to Wikipedia**

```
host: en.wikipedia.org

method: GET

path: /wiki/Hypertext_Transfer_Protocol

scheme: https

version: HTTP/1.1
```

- **The server will receive the request (as well as some other details) and formulate a response**

- **You just saw the request consisted of:**
  - **Host and path:**
    - Your computer will look up the IP address of the URL `en.wikipedia.org` (the host) and send the request to that address
    - Because some IP addresses host multiple servers, the request includes the host name
    - It also includes the URL so that the server can know what you're looking for (the path)
    - For example, the `/wiki/Hypertext_Transfer_Protocol` resource
  - **Method**:

    * There are nine different methods specified in the HTTP protocol that describes to the server what you expect it to do with your request
    * The most common are `GET`, `POST`, `PUT`, and `DELETE`
  - **Scheme and version:**
    * The scheme tells the server what protocol the client wants to use to communicate (HTTP or HTTPS)
    * The version specifies what version of that protocol to use

Processing the Response: Parsing HTML and CSS

- **When the response returns, it will contain:**
  - a status code
  - response headers (perhaps identifying the type of data returned)
  - a response body
- **When the request is sent to a website, the response body typically contains HTML, CSS, and JavaScript**
  - HTML is used to describe the page content
  - CSS is used to describe how the content appears
- **The actual document that appears on the screen is the result of the browser creating and rendering the Document Object Model (DOM)**
- **The process of converting the HTML document into the DOM is called *parsing***

```
<div class="main">

  <h2>Books to read</h2>

  <ul>

    <li>To Kill a Mockingbird</li>

    <li>Hitchhiker's Guide to the Galaxy</li>

    <li>Eloquent Javascript</li>

  </ul>

</div>
```

- **What does it have to do when it parses the HTML to create the DOM?**
  - Create an `HTMLDivElement` object with one item in the list of classes (`main`)
  - Create an `HTMLHeadingElement`
    * Then create an `HTMLTextElement` that contains the text "Books to read"
    * Then add the text element as a child of the heading element
    * Then append the heading element into the div element
  - Create an `HTMLUListElement`
    * Then create an `HTMLLIElement` element
    * And create a child `HTMLTextElement` element with text being the names of the books
    * Then add the text element as a child of the `HTMLLIElement` element

    * Then append the li element to the ul list

    * And finally, insert the unordered list element into the div element.

- **Once the browser is done parsing the HTML file, it has an in-memory representation of the file (now called the DOM)**
  - There's a similar process for parsing CSS files into what's called the CSSOM (CSS Object Model).

Special Tags

- **There are a few tags that kick off special behaviors, including:**
  - the `<link>` tag
  - the `<script src="">` tag

- **When you link in a remote file, such as a stylesheet, the browser creates a new HTTP request to fetch that file**
  - It does this without pausing its parsing
  - This means the request executes *in parallel* with the parsing

- **If you include a remote script, the browser will kick off a new HTTP request for that file and pause its parsing at the end of the `<head>`**

- **Why the different behaviors?**
  - CSS files ***cannot*** manipulate the DOM, so there's no potential for conflict with what the HTML document says.
  - JS ***can*** manipulate the DOM so the browser needs to execute the code to ensure that it builds the proper representation of the DOM
    - Note: there's an `async` property that lets you instruct the browser to fetch the JS file asynchronously

Processing the Response: Rendering

- **'Rendering' involves a few different steps:**
  - Taking the DOM and computing boxes for every element
    - Computing the (x, y) coordinates each box should appear at given the bounding boxes
    - Computing style properties (by exploring the CSSOM)
  - Actually painting the content of the boxes at the computed coordinates with the computed style properties

- **Different browsers use different rendering engines, which you may or may not have heard of:**
  - Chrome uses Blink (forked from Webkit)
  - IE uses Trident
  - Safari uses Webkit
  - Firefox uses Gecko

- **When we say a browser 'supports' a given HTML or CSS feature, it really means the rendering engine knows how to do its job with it**

Executing JS Code

- **Browsers use different JS execution engines:**
  - Chrome uses V8
  - IE uses Chakra
  - Safari uses Nitro
  - Firefox uses SpiderMonkey
- **This is what accounts for small differences in the way the same script will sometimes execute in different browsers**
- **JavaScript code can manipulate the browser's DOM for the current page**
- **JavaScript code might run at different times, including:**
  - When the browser loads the page, it interprets some of the JavaScript code it finds on the page and runs it
  - Other JavaScript doesn't run until the user interacts with the page (clicks a button, tabs out of a form field, hovers a mouse over an image, etc.)
- **JavaScript is what gives the page dynamic behavior**

The Tale of the Slow Kid

- **One common urban legend that you'll likely encounter is that JavaScript, or maybe the DOM API, is slow**

- **JavaScript used to be slow**
  - Browser manufacturers have focused on improving performance within the browser
    - Google has been a big driver of progress with its V8 engine
  - Having said that, there are some things that JS isn't particularly great at; for example, intense array-based math operations like matrix multiplication

- **The DOM API isn't slow, but you have to know how to use it**
  - It's possible that you might write DOM API code that is quite inefficient
  - For example, if you insert a bunch of elements one at a time, the browser needs to recompute the position of all surrounding elements (as well as all of their bounding boxes, etc.) each time! This can be slow!