How internet?

Back-End Introduction

1. Client-Server Relationships
2. How do the applications we build interact with each other and with other services?
3. Full stack development
   1. Work on both front-end (client side) and back-end (server side), delivering end-to-end solutions.
4. Clients and servers
   1. On the front-end
   2. Facebook, twitter, amazon can be thought of in terms of third-party APIS.
   3. Client, database, and third-party APIs all connect to a server.
   4. A server -
      1. Anything that can take a request and send a response.
   5. A database –
      1. Where we persist information
   6. A client –
      1. Anything that can send a request (whether that’s a user or an object or a function requesting data – could often refer to your browser)
   7. A client sends a request to a server. “Can you get me the posts on my Reddit front page?”
      1. It interacts with a 3rd-party API and its databases, then sends a response back to a client.
   8. (Remember – there are many middlemen between the client and the server, for example, ISPs).
   9. Hackers are often going to target databases, because APIs are generally quite secure; 3rd parties (fb, amazon, etc) will try to protect their information.
5. Front-End
   1. Whatever deals with the user interface of an app or website
   2. Structured with HTML and CSS.
   3. Web applications and the logic of the front end are controlled using JS.
6. A server
   1. Is any *software* that can accept and respond to an HTTP request.
      1. No longer can a computer only manage one server.
   2. Servers :
      1. deliver content
      2. Connect to databases
      3. Connect to web APIs
      4. And provide central business logic.
         1. Business logic is anything that the software needs to do on the back end.
         2. For example, how it takes your information or your preferences and sets up your profile.
7. Databases
   1. They store the state of our application
   2. And is often stored on another remote computer
   3. And persists information outside of that server.
8. Third-party APIs
   1. It can connect to other apps on the web
   2. Allows software to speak to other software
      1. For example, the console log is an API that connects to the browser and displays information.
   3. Document is also an API
      1. It’s kind of like a library or a collection of functionalities that allow us to interact with the software providing it.
9. We will be building our **server and database on our laptop.**
   1. Eventually, those servers and databases will be external computers
   2. But it is absolutely possible to have all elements of “the internet” on a single computer.
10. HTTP Requests & responses
    1. This is an app=-layer protocol for transmitting docs, such as HTML
    2. It’s designed for communications between web browsers and web servers.
    3. A client initiates the conversation (a request) and the server replies (response).
       1. The response is always given, even if the response is “Idk bro”
11. Parts of an HTTP request
    1. These are sent to the server so the server knows how to handle the request (do we get stuff from the server? Do we get stuff from an external API? Do we update info in the database?)
    2. Method: “GET”, “POST”, “PUT”, “PATCH” “DELETE”
       1. Get stuff, post stuff, updating an entire stuff, updating a part of a stuff or delete stuff.
    3. Is passed to the URL
    4. Header: CORS, User-agent, cookies, etc that gives context about the request.
    5. A POST request
       1. To ebay.com
       2. Header: accept: text/html, accept-language: en-us
       3. Body <- a post needs more information that we’ll actually send.
          1. Id: 001
          2. Quantity: 2
    6. The HTTP Request method is used to choose how to route the request.
       1. Get, Post, Put, Patch, and Delete
    7. We use “GET” when:
       1. Retrieving or reading information.
    8. We use “POST” when
       1. We need to add data to a database or update information.
12. HTTP **Responses**!
    1. Headers again
    2. 200-299 is OK
    3. 300-399 is additional action
    4. 400-499 is client error
    5. 500-599 is server error.
13. A CRUD Application
    1. Create, Read, Update, Delete
    2. Refers to an app that has all these functionalities.
    3. Create – POST
    4. Read – GET
    5. Update – PUT/PATCH
    6. Delete – DELETE
    7. All our projects will be CRUD applications.
14. The Internet is just a web of requests.
    1. Hundreds of millions of requests are being made by different servers, different applications, etc.
15. When creating servers:
    1. We need organization.
       1. It should be semantic and intuitive.
       2. It should be amenable to optimization
       3. It should be scalable.
16. RESTful APIs
    1. A set of *guidelines* for structuring your app and API.
    2. REST – Representational State Transfer
       1. Intuitive routing (routes that make sense to the client)
       2. Stateless (client state is not stored on the server)
          1. What this means: Whenever a RESTful server handles a request,
          2. It acts as though it will be running this for the last time.
          3. Nothing from past calls is going to modify the current call.
       3. Allows for performance tuning.
    3. Intuitive RESTful routes – readable and obvious.
       1. GET /dogs -> GET A LIST OF DOGS
       2. Post /DOGS -> create a new dog
       3. GET /dogs/17 -> get a dog with id 17 (or)
       4. GET /dogs/fido -> get a dog with username “fido”
       5. PUT /dogs/17 -> editing information for dog 17.
       6. DELETE /dogs/17 -> delete dog #17
    4. REST is stateless.
       1. Visible – a request contains the context that is necessary to understand it.
       2. Reliable – a failure of one request does not influence others (each request should be handled uniquely and uninfluenced by previous requests)
       3. Scalable – The server doesn’t have to remember the application state, enabling it to server more requests in a shorter amount of time.
17. Specific concepts
    1. Routers
    2. The path that HTTP requests take is not linear. It’s routed through many directions.
       1. A request is sent to a router, then to a modem, then to an ISP, etc etc etc.
    3. DNS
       1. A Domain Name System
       2. Is used to convert IP addresses into humanly readable domain names.
       3. For example, we can access the google website via google.com or by inputting its IP address directly.
    4. We type “ebay.com”
       1. The browser check the cache for a DNS record
       2. If the URL is not in the cache, ISP’s DNS server initiates a DNS query to find the IP address
       3. And then….
       4. ………
       5. A GET request is sent (header: {accept: text/html})
       6. Server sends back a response. 200 {content-type: text/css} .container{margin:0 auto;}
       7. As the computer reads through our code
          1. Once it reads <script src = “jQueryURL.jQuery.com”>
          2. It requests the js file from the server with another get request for the jQuery file stuff.
18. Security attacks
    1. Phishing attacks – an attacker impersonates a qebsite to gain access to its users’ info
    2. HTML injection or content spoofing
       1. Injects HTML content into a website, for example, by changing a guestbook to look like a website
    3. Cross-site scripting XSS
       1. Malicious JS is embedded into a website and runs in a client’s browser
    4. Reflected XSS
       1. Injectable code sent to the server from a client’s browser (often after a malicious link)
    5. Persistent XSS
       1. Injectable code is permanently saved on the server, from which it is used to attack clients
    6. NEVER TRUST CLIENT-SIDE INPUT
       1. Always assume that someone will try to break your system, your client input, etc etc etc.
       2. Whenever information is passed in, all data should be sanitized (from a request body, URL parameters, etc) prior to using it
       3. And obscure everything. This can be accomplished by encoding data.
19. Note:
    1. A server can also be a client (when it sends a request to a database)
    2. Remember that the client-server relationship is relative and situational.
20. CDN
    1. Servers with Caches
    2. Caching information oftentimes for connectivity
    3. When you make a change to a website, a CDN must be configured to know how it will cache/handle that information.
21. HTTP:
    1. Has a transport layer protocol (like tcp, udp)
       1. The connection to a server is established here
       2. And says we can handle encrypted requests
    2. So we can send encrypted requests to a server and the server will still know where the request is coming from.
    3. OSI model -> 7 different models of protocols that all build upon one another.