**What is "Full Stack"?**

Diagram

Description automatically generated

- "Full Stack" is a term used to describe something that encompasses BOTH the "frontend" and the "backend"

- It encompasses the company's entire "tech stack"

- A "tech stack" is simply a group of technologies that a given application uses

- Project 3 full stack tech stack will include

HTML, CSS, JavaScript, Angular, (Bootstrap), Java, Spring Boot, MySQL

- A full stack engineer is someone who has a good handle on how things are working from both sides (frontend/backend)

- Have a bird's eye view on a project as a whole

Diagram

Description automatically generated

**Frontend:**

- Frontend gets its name because it is "front facing" AKA "client facing"

- As such, the core values of a frontend application are UI/UX

- UI is user interface

- UX is user experience

- At its core are technologies like HTML, CSS, and JavaScript

- Frontend is the place the client will visit (typically in a web browser, but could be a mobile app)

- In order for things to work, the client must the client must download all static assets when visiting the webpage (HTML, CSS, JavaScript)

- Note: For Angular the client the client receives a pretty much empty HTML file and a big JS file with how to put together the HTML file. This is called client side rendered which means that a user visiting the webpage is given an empty HTML file and a big JS file to build the webpage with

-- This is called client side rendered

-- This is as opposed to Server-Side Rendering where the server puts together all the parts of the HTML page before sending it over the web, this takes a fraction of a second but with lots of requests to the server it ends up being slower than offloading that work to the client

- All code written on the frontend is PUBLIC

- Anything you write you can assume will be seen by a prying eye

- As such, NEVER store anything private or valuable on the frontend because You can see everything since visiting the site downloads all the HTML, CSS, and JS

- This is the reason backends exist, since they are since they are private and not publicly accessible

- Additionally, we should never trust data coming from the user

- We should assume the worst of it, and work to prove otherwise with input validation/sanitization

- This is done on the backends because , since they are private and not publicly accessible

**Backend:**

- The backend is the "back side" of the application where we handle sensitive data and/or company algorithms

- Useful when we want to handle logic/data that isn't meant to be seen by an end user

- Often times the backend is referred to as a server

- Can be done with any programming language that is able to listen to a port on a computer for incoming requests

- A port is an open window on your computer where communication is allowed

- The IP address is like a business’s telephone extension number, you dial a businesses number and select who you want to talk to, so instead of just a computer IP address you talk to a specific service on the computer by selecting a port number

<https://www.techtarget.com/searchnetworking/definition/port#:~:text=A%20port%20in%20computer%20networking,is%20a%2016%2Dbit%20number>

- Port 80 is HTTP

- Port 443 is HTTPS

- Port 22 is SSH

- Port 23 is TelNet

- I can have a server listen on any port of my computer for requests

- Since 80 is the default http port I don't need to specify it in the web browser

- These requests come in the form of HTTP messages of HTTP messages (must be properly formatted like mailing a letter it needs to be properly addressed and put in an envelope (can’t just send a stapled together stack of papers)

**HTTP:**

- HTTP stands for "Hypertext Transfer Protocol"

- In order to have a backend and frontend talk to one another, we needed a protocol to follow

- HTTP was the answer

- HTTP comes with requests and responses

- HTTP is the language of the internet and is what it uses to communicate

**HTTP Request:**

- The request is a request to the web server to perform some task

- To specify what task they need to perform we need to pass a couple different things

- Request Line

- ex. GET doc/test/index.html HTTP/1.1

- GET refers to HTTP method

- doc/test/index.html is the URL/endpoint

- HTTP/1.1 is the version of the HTTP protocol we are using

- *Request Headers*

- Used to specify any information about the request

- Common headers include:

- Accept which specifies content type to accept

Ex: text/HTML

images/png

images/\* = any file type

\*/\* = any content type of any file type

- Authorization which passes a badge of sorts indicating your level of authorization

-*Request Body*

- The section which contains the data we want to save or update

- Most endpoints ignore the request body for a GET request

- Some HTTP methods like a HEAD request does not contain a body

- POST requests use the response body to hold the data to be saved

- Ex: Create an account by \_\_\_\_\_\_\_\_\_\_\_\_\_\_ in the request body

- {username: '', password: '', firstName: '', lastName: '', birthday: '', email: ''}

**HTTP Methods:**

- HTTP methods allow us to add a layer of semantic meaning to our HTTP request

- The big 4 HTTP methods:

- GET

- Is the most common one

- Is used retrieve data

- It is used to indicate that the user would like to fetch a resource

- It is the READ of the CRUD acronym

- POST

- It is used to indicate the user wants to store data

- It is the create of the CRUD acronym

- Think of it like if you "post" a blog, you're creating a blog post. POST indicates the same

- Often times has a request body

- Will cause side effects due to not being indempotent

- PUT

- Is is used to update an existing resource

- Take out the old value there and put in the new one

- The UPDATE of the CRUD acronym

- would need a request body bc passes all the same information as when you did the create

- PATCH

- Is like PUT but instead of replacing an existing database entry with an entirely new one

it is just updating a portion of an existing entry

- would need a request body but not as much redundant information, don’t need to pass the parts that aren’t changing

- DELETE

- Deltes a resource/piece of data

- Other HTTP methods:

- PATCH

- Works very similary to PUT, but instead of \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_, we update a small piece of the old value

- PATCH is used for smaller updates (ex. changing user preferences)

- Think of patching a tire with a hole in it as opposed to replacing the tire entirely

- HEAD

- Functionally the same as a GET request in all ways, but name and the fact that it doesn't have a request body

- It's commonly used as something of a heartbeat check

- Used as a ping of sorts

HTTP Method Categories:

Safe = does not alter the state of the database

Includes: HEAD, GET

Indempotent = you can perform the action multiple times without having any side-effects (causing any changes to the state of the database) after the first time

Includes: GET, PUT, PATCH, DELETE

Does NOT include: POST

**HTTP Responses:**

- Is the response to an HTTP request and contains an status code to indicate how the request went

- The response often contains a response body section with the requested resource

- A GET request to https://www.mywebpage.com/users/7 assuming successful might a response like:

- 200 OK status with a body of the user data in JSON format like:

{id: 7, firstName: 'John', lastName: 'Doe', birthday: ''}

- An HTTP endpoint is the url you’ve set up your api to listen to requests on

**HTTP Status Codes:**

[**https://developer.mozilla.org/en-US/docs/Web/HTTP/Status**](https://developer.mozilla.org/en-US/docs/Web/HTTP/Status)

- HTTP Status Codes are used to indicate success or failure of the request sent

- Different types of status codes:

- 100s

- Statuses in the 100s represent information responses

- Example: 100 Continue

- 200s

- Statuses in the 200s represent success

- Example: 200 OK

201 CREATED

204 NO CONTENT

- 300s

- Statuses in the 300s represent redirects

- Examples: 307 Temporary Redirect

308 Permanent Redirect

- 400s

- Statuses in the 400s represent error

- Example: 404 Page Not Found

405 Method Not Allowed

- 500s

- Statuses in the 500s represent server error

- Example: 501 Not Implemented - The request method is not supported by the server and cannot be handled

500 Internal Server Error

TODO: Download Postman – practice sending and looking at requests

**Naming Conventions for API Endpoints:**

1. Uniform Interface
   * Due to the Uniform Interface constraint I shouldn't have API endpoints that return the same thing
   * Example: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_
2. Avoiding Verbs
   * You should avoid using verbs in url as it is unnecessary
   * Instead, \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_
   * Avoid routes like /getUsers or /updateUsers
   * Instead do something like \_\_\_\_\_\_\_\_\_\_\_\_\_
3. URL Hierarchy
   * URLs should have a hierarchy going from \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_
   * Example: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_
4. Filtering
   * We can add filters in the form of \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ to indicate constraints or filter criteria
   * If I wanted to retrieve the first 10 orders of a user's all time orders I could do this:
     + ex. [\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_](https://www.mysite.com/users/28/orders) (This implies retrieve ALL orders)
     + ex. [\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_](https://www.mysite.com/users/28/orders?limit=10) (Implies to retrieve the first 10 orders)
     + ex. [\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_](https://www.mysite.com/users/28/orders?type=sameDay) (Only retrieves orders with same day delivery)
     + ex. [\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_](https://www.mysite.com/users/28/orders?limit=10&type=sameDay) (Does both)
5. Versioning
   * Versioning endpoints allows \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_
   * Imagine I have endpoint /movies which retrieves a movies if they do a GET request
     + At first maybe the data back as just the movie title
       - "Star Wars"
     + Now that my app is bigger, only sending the title is not enough anymore and I want to add new data
     + So I change it to now send back a JSON object containing movie data
       - {"title": "Star Wars", "year": "1977"}
   * Instead, we want to version our URLs such that the old URL still works, but we promote the new one
     + Old url: [\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_](https://www.mysite.com/v1/movies) returns ["Star Wars", "Harry Potter"]
     + New url: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ returns [{"title": "Star Wars", "year": "1977"}, {"title": "Harry Potter", "year": "2001"}]