**What is "Full Stack"?**

- "Full Stack" is a term used to describe \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

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

- A "tech stack" is simply \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

- Project 3 full stack tech stack will include

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

- A full stack engineer is someone who \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Frontend:**

- Frontend gets its name because it is \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

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

- UI is \_\_\_\_\_\_\_\_\_\_\_\_\_\_

- UX is \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

- At its core are technologies like \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

- Frontend is \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

- In order for things to work, the client must \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

- Note: For Angular the client \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

-- This is called \_\_\_\_\_\_\_\_\_\_\_\_\_

-- This is as opposed to \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ where \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

- All code written on the frontend is \_\_\_\_\_\_\_\_\_

- Anything you write you can assume will be \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

- As such, \_\_\_\_\_\_\_\_ store anything private or valuable on the frontend because \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

- This is the reason \_\_\_\_\_\_\_\_\_ exist, since they are \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

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

- We should assume the worst of it, and work to prove otherwise with \_\_\_\_\_\_\_\_\_\_\_\_\_\_

- This is done on the \_\_\_\_\_\_\_\_ because \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Backend:**

- The backend is the \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

- It's where we handle \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

- Useful when we want to handle \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

- Often times the backend is referred to as a \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

- Can be done with \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ that is able to \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

- A port is \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

- Port 80 is \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

- Port 443 is \_\_\_\_\_\_\_\_\_\_\_\_\_\_

- Port 22 is \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

- Port 23 is \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

- I can have a server \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

- Since 80 is \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_, I don't need to specify it in the web browser

- These requests come in the form of \_\_\_\_\_\_\_\_\_\_

**HTTP:**

- HTTP stands for \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

- In order to have a backend and frontend talk to one another, we needed \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

- HTTP was the answer

- HTTP comes with \_\_\_\_\_\_\_\_ and \_\_\_\_\_\_\_\_\_

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

**HTTP Request:**

- The request is \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

- 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 \_\_\_\_\_\_\_\_\_\_\_\_\_

- doc/test/index.html is the \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

- HTTP/1.1 is the \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

- Request Headers

- Used to specify \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

- Common headers include:

- Accept which specifies \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

- Authorization which passes a \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

- Request Body

- The section which contains \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

- Most endpoints ignore the request body for a \_\_\_\_\_ request

- Some HTTP methods like a \_\_\_\_\_\_\_ request does not contain a body

- POST requests \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

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

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

**HTTP Methods:**

- HTTP methods allow us to \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

- The big 4 HTTP methods:

- GET

- Is \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

- Is used \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

- It is used to indicate that \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

- It is the "\_\_\_\_\_" of the CRUD acronym

- POST

- It is used to indicate the \_\_\_\_\_\_\_\_\_\_\_\_

- It is the "\_\_\_\_" 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 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

- Will cause side effects due to not being \_\_\_\_\_\_\_\_\_\_\_

- PUT

- Is is used to update an existing resource

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

- DELETE

- Deltes a resource/piece of data

- Other HTTP methods:

- PATCH

- Works very similary to PUT, but instead of putting in an entirely new value, 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 =

Includes: \_\_\_\_\_\_\_\_\_\_\_\_\_\_

Indempotent =

Includes: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**HTTP Responses:**

- Is the response to an HTTP request and contains an \_\_\_\_\_\_\_\_\_ to indicate how the request went

- The response often contains a response body section with the \_\_\_\_\_\_\_\_

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

- An HTTP endpoint is \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**HTTP Status Codes:**

- HTTP Status Codes are used to \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

- Different types of status codes:

- 100s

- Statuses in the 100s represent \_\_\_\_\_\_\_\_\_\_\_\_\_

- Example: \_\_\_\_\_\_\_\_\_

- 200s

- Statuses in the 200s represent \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

- Example: \_\_\_\_\_\_\_\_\_\_

- 300s

- Statuses in the 300s represent \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

- Examples: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

- 400s

- Statuses in the 400s represent \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

- Example: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

- 500s

- Statuses in the 500s represent \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

- Example: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

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, let the HTTP verb do the talking
   * ex. Avoid routes like /getUsers or /updateUsers
   * Instead do something like this GET /users or PUT /users
3. URL Hierarchy
   * URLs should have a hierarchy going from most specific to least specific
   * ex. <https://www.mysite.com/users/28/orders>
   * ex. <https://www.mysite.com/users/28/orders/51>
4. Filtering
   * We can add filters in the form of "request parameters" 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 new endpoints to be added without breaking the old ones
   * 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
     + <https://www.mysite.com/v1/movies> returns ["Star Wars", "Harry Potter"]
     + <https://www.mysite.com/v2/movies> returns [{"title": "Star Wars", "year": "1977"}, {"title": "Harry Potter", "year": "2001"}]