Phase 1 blog post:

There and Back Again: The Prologue

I started a software engineering bootcamp through Flatiron School a week ago. So far, learning how to be a software engineer has been like trying to drink from a fire hose. The more I learn, the more I discover just how *much* there is to learn. I feel like Bilbo Baggins leaving the Shire: my blissful existence as a user is over, and my journey to the Lonely Mountain has begun. But before I walk through the front gate, let’s do some light recon:

At a high-level, software engineering is divided into two areas of concern: frontend and backend. The frontend is typically comprised of a user-interface, and the backend stores data and data-processing logic. Because of my background in research, I have some experience building pipelines for processing research data like signals, which ran in a backend-type framework. However, I don’t have any experience in frontend development. (Un)lucky for me, that’s exactly where our journey starts!

Most people interact with front-end applications all the time; you’re reading this text on a front-end application right now! Most of the time, the frontend also needs information that lives in the backend in order to function properly. To access this information, the frontend communicates with the backend through an API (application programming interface). In the case of web-based applications, like Instagram, the frontend, which displays images and text for the user and supports interactivity through likes, comments, and posts, communicates with the backend, where the data and data-processing code actually live, via an API that uses HTTP (HyperText Transfer Protocol) requests/responses.

Ok, that was a lot of technical jargon for this early in the prologue, but stay with me! Let’s stick with our Instagram example: When we open Instagram, our feed loads with user stories at the top, and images, videos, and text in the form of posts, captions, and comments below. As a user, we usually take this part for granted: *of course the page loads when I open Instagram*. But how did that actually happen?

There are a few different layers within the frontend that work together to give us our frontend experience of Instagram. The webpage structure is specified using HTML (HyperText Markup Language) on an HTML document. An example of an HTML header might look something like this:

<h1>This is the Instagram header!</h1>

Some of the architecture for Instagram is static: its characteristics don’t change. The background is always white (or black, if you have night-mode on), stories are always at the top, with the feed below that, and the ‘explore,’ ‘search,’ ‘post,’ ‘reels,’ and ‘my profile’ buttons are always at the bottom of the screen. In fact, if you’re looking at Instagram from a web browser, you can see the underlying HTML for yourself by right-clicking anywhere on the page and selecting “inspect” to open the developer window (neato!).

But some of the webpage contents are dynamic. Suppose we’re scrolling down in our feed, and we need the webpage to load new information, or maybe we want to make new content of our own by writing and posting a comment. How does that happen? For that, we need to be able to update the webpage dynamically, only without having to go in and hardcode the underlying HTML every time.

The HTML document is rendered on the webpage for us via the DOM (document object model). The DOM is just what the acronym states: a model of our underlying HTML document, and the DOM acts as a programming interface. It represents the page so programming languages can change the web page (document) dynamically. The DOM allows us to have an *easier*, *faster, and* *interactive* experience with our webpage.

Let’s say we want to ‘like’ a post. How does the webpage “know” to update itself and display a red heart and increase the ‘like count’? The DOM! Instagram has frontend code “listening” for events happening on the webpage, such as hitting the ‘like’ button, and that code then interacts with the DOM to update the HTML document. That code might look something like this:

// select the like button so we can do things with it

>> const likeButton = document.querySelector(“.like-button”);

// add an event listener so we can do something when the like button is clicked

>> likeButton.addEventListener(“click”, () => {

// increase the like count

// update the DOM to display the new like count

});

Great, that means our Instagram page will update to reflect our ‘like’!

But how can we see the most recent like count in the first place? Along with a list of all the other users who previously liked the post? Behind the scenes, the frontend Instagram application is actually interacting with a backend server storing the data through an HTTP request/response communication pattern (ah, we’ve come full-circle!).

When we first open Instagram, the frontend sends a ‘GET’ request to the backend asking for the latest version of information to display in our stories and on our feeds. The backend sends a response, and if everything goes well, that response includes the information our frontend needs to update the webpage via the DOM. That code might look something like this:

// send GET request to server

>> const fetchFeedInformation = async () => {

>> const response = await fetch(“Instagram server address”);

>> const feedInformation = await response.json();

// do things with the feedInformation like load content

};

>> fetchFeedInformation();

When we hit the ‘like’ button, along with updating the DOM to reflect our new like count, the frontend also sends a ‘POST’ request to the backend with information necessary to update the like count and the user list in the server, and the backend sends a response with a status indicating how things went. Turns out a lot is happening when we open up and interact with a webpage, even with an interaction as “simple” as liking a post!

Now that we’ve done a little recon we have a solid foundation for adding more complexity and creating more sophisticated web applications. We’re ready to leave the Shire!