1. How is the Angular project structure different from that of the Express HTML customer-facing page?

During some aspects of this course, I worked on the architecture, functionality, and testing of the SPA for the web server's client-side administration. The Angular project structure contrasts starkly with the Express HTML customer-facing page regarding its organization and approach. The Angular is organized into modules, components, services, and directives within the src directory. In contrast, an Express HTML customer-facing page typically has a simpler file structure with HTML, CSS, and JS files organized in folders as needed. The Angular project structure is distinct from the Express HTML customer-facing page due to its utilization of a component-based design. This means that each component is accountable for a certain feature or functionality. Because of this, it is simpler to manage the code and modify the app without affecting other system components. Regarding components, Angular is built on a component-based architecture with reusable and encapsulated components. In contrast, Express HTML's customer-facing page uses templating engines but lacks a strict component-based structure.

2. What are some advantages and disadvantages of the SPA functionality? What additional functionality is provided by a SPA compared to a simple web application interaction?

One of the advantages of SPA in terms of functionality is that SPAs offer faster navigation and responsiveness as they load only once, fetching data dynamically and enhancing user experience. However, one of the drawbacks is that the fact that they rely on JavaScript for rendering could hurt them in terms of initial load speed and search engine optimization.
When it comes to functionality, SPAs are more feature-rich than standard web apps. They are unique because they can dynamically change material without requiring a page refresh. Obtaining data asynchronously from the server provides more engaging and fluid user interfaces that provide a seamless user experience.

3. What is the process of testing to make sure the SPA is working with the API to GET and PUT data in the database? What are some errors you ran into or what are some errors you could expect to run into?

It took a lot of work to test the SPA's ability to retrieve and update data via the API. Before accessing and editing the database, I had to ensure the SPA was correctly interacting with the server using GET and PUT requests. Incorrect API endpoint setups or problems with CORS were the most frequent causes of errors during this phase. I also found that I was missing some of the methods and functions, which caused some of the errors I encountered. For instance, I created a "TripDataService" method, but it was empty, so I had an error when I compiled the code. Other possible issues include network, request, and page loading errors.
In order to solve such forms of errors, proper testing needs to be done. We should also check to be certain that we are not missing anything. Verifying the login credentials stored in the database is one way to test if the SPA is operational. Ensure the application can comprehend the login API responses and that the API URL is accurate. Also, ensure the dashboard, login form, and loading spinner components are up and running and working as expected.

4. What questions do you still have that will help you with SPA in future builds?

SPA offers faster load time once the initial assets are loaded. One issue with the SPA is the initial load time. I would like to ask how I can optimize initial load times for an SPA. In the future, I would also like to know how applications like this, or even different applications would handle routing and navigation and how to best utilize the SPA's performance across various kinds of screens and platforms.