Name: Shekhar Chaudhary

Date: 09/12/2025

### Reflection on the Angular Single Page Application (SPA)

#### **Architecture**

The Angular project architecture significantly differs from the Express HTML customer-facing page in terms of structure, logic separation, and interaction model. While the Express project primarily relies on server-side rendering, where each user action triggers a new HTTP request and response cycle, Angular operates on the client side using a component-based architecture. This means that Angular applications are divided into reusable UI components, each managing its own data and logic, leading to more modular and maintainable code.

Angular leverages TypeScript and the Model-View-Controller (MVC) pattern, allowing clear separation between presentation and logic. The Express app, by contrast, uses server-rendered HTML templates (like EJS or Handlebars) and has limited client interactivity. With Angular, the SPA loads once and dynamically updates content using data binding and the Angular Router, which simulates navigation without reloading the entire page. This architecture greatly improves performance and user experience, especially in modern web environments.

#### **Functionality**

The functionality provided by a SPA far exceeds that of a traditional HTML-based application. Angular’s two-way data binding and dependency injection make it easier to create dynamic, responsive, and interactive interfaces. The SPA model allows data updates in real time without requiring full page reloads. For instance, admin users can manage or edit content instantly, and changes reflect on the client side while asynchronously updating the backend database.

Another advantage of the Angular SPA is its ability to integrate seamlessly with RESTful APIs. The application communicates with the backend using HTTP methods such as GET, POST, PUT, and DELETE, enabling smooth CRUD operations. Additionally, Angular’s reusable components and services enhance maintainability and scalability. However, a key disadvantage is the increased complexity of setup and initial load time due to larger JavaScript bundles.

#### **Testing**

Testing the SPA to ensure API communication is functioning correctly involves verifying both the frontend and backend interaction. For instance, to confirm that the Angular application can successfully GET and PUT data to the database, I used testing tools like Postman and Angular’s built-in HttpClientTestingModule. These tools help simulate API calls and validate that the responses and data updates are correct.

During testing, I encountered potential issues such as CORS (Cross-Origin Resource Sharing) errors and mismatched data schemas between the frontend and backend. These were mitigated by updating the API headers and validating the request payloads. End-to-end testing (E2E) was also essential to confirm that user interactions correctly triggered backend operations and UI updates.

#### **Future Considerations**

While I have gained substantial understanding of SPA architecture, there are still areas to explore further. I plan to deepen my knowledge of Angular’s performance optimization techniques, such as lazy loading, change detection strategies, and state management using NgRx. I also want to implement more advanced testing frameworks like Cypress for real-world simulation testing.

In future builds, I aim to expand the SPA by incorporating user authentication, route guards, and role-based access control for secure and personalized user experiences. These improvements would bring the application closer to production-grade readiness.

### ****References :****

* Google Developers. (2023). Angular documentation. https://angular.dev
* Express.js. (2023). Express application structure. https://expressjs.com/en/starter
* Mozilla Developer Network (MDN). (2024). Single Page Applications (SPAs). https://developer.mozilla.org/en-US/docs/Glossary/SPA
* Postman Learning Center. (2024). Testing your API. https://learning.postman.com/docs