What are the advantages of this project?

This project is developed using Salesforce's Lightning Web Component (LWC) framework. LWC, or Lightning Web Components, is a powerful tool for building web components within the Salesforce Lightning Platform. It empowers developers to create dynamic, reusable, and efficient user interface elements for Salesforce applications. LWC is a modern development model that utilizes standard web technologies such as JavaScript and HTML. It offers a lightweight, secure, and highly customizable way to extend and enhance the Salesforce user experience. LWC is particularly valuable for building interactive and responsive components that seamlessly integrate into the Salesforce environment.

The primary objective of this project is to present a 2D matrix table on the Salesforce UI, providing users with clear and accessible information about the data.

In this project, we've concentrated on three key features:

- **Configurability**: We've ensured that users can configure the columns and rows within the table without the need to modify any code.
- **Dynamism**: We've made sure that the data in the table is dynamic. If a user adds or deletes records in Salesforce, the latest data will be visible in the table.
- **Reusability**: We've crafted the project's code to be reusable. This ensures that if there's a requirement to add more tables to the Salesforce UI in the future, it can be accomplished with minimal code changes.

How is the code structure?

We're utilizing the LWC framework to build the UI component, which will be integrated into the Salesforce org. The UI component is constructed using HTML and JavaScript. Additionally, we're using Apex code to fetch data from Salesforce.

Currently, the project uses the Imperative method to call the Apex class. The data is displayed after the user clicks a button. We're using a wrapper class to collect user-inputted data and then passing that wrapper class as a parameter to the Apex controller. The Apex class performs an SOQL query on Salesforce records using the inputs from the wrapper class.

To enhance table configurability, the columns' data is stored in custom metadata. Custom metadata in Salesforce acts like a flexible database within your Salesforce organization, allowing you to manage custom configuration settings and data required for your applications and processes.
Upon receiving the imperative call from the LWC component to the Apex controller, we first query the records in Salesforce and then map the records to the correct columns in the table using a MAP collection variable.

To make the table more configurable I have stored the columns data of the table in custom metadata. Custom metadata in Salesforce is like a flexible database within your Salesforce organization that allows you to store and manage custom configuration settings and data that you need for your applications and processes.

Once we receive the imperative call from the lwc component to the apex controller we first query the records in salesforce and then map the records with the correct column in the table using the MAP collection variable.

What is not good about your project?

It would be beneficial to consider using the "wire" method to call the Apex controller for data retrieval. Currently, in LWC, there are two methods for fetching data from Salesforce: using a "wire" decorator and using imperative calls. In simple terms, "wire" works automatically behind the scenes, while imperative calls are initiated by the developer when data retrieval is necessary.

One significant difference between "wire" and imperative calls is how data is retrieved and updated. "Wire" calls provide real-time updates to data in the LWC, while imperative calls require the developer to explicitly fetch and update the data.

For more information, please refer to the following article: Understanding Wire vs. Imperative Calls in LWC.

What would you do better next time?

In future projects, I will opt to use the "wire" decorator to retrieve data in LWC and store it in a reactive variable. This approach allows for real-time data updates, ensuring that any changes in the data are promptly reflected. It offers the advantage of presenting real-time data to Salesforce users.

After retrieving data using the "wire" method, I will store it in an array list. Based on user input, I will then apply data filtering logic in JavaScript. This eliminates the need to make calls to Salesforce for every piece of logic.

This approach will lead to faster component loading, as the logic is executed in JavaScript, and there's no need to wait for data from Salesforce for each operation.