Major Project Report

On

" Dev Automation of CPQ using Logik IO "

Submitted by:

Madhurya Hait (202100516)

In partial fulfilment of requirements for the award of degree in

Bachelor of Technology in Artificial Intelligence and Data science

(2025)

Internal Reviewer

Mr. Sital Sharma
Department of AI&DS,
SMIT, SMU.

External Guide

Mr. Bhavesh Gandhi (Manager)
Keysight Technology
Gurugram

Department of Artificial Intelligence and Data Science
Sikkim Manipal Institute of Technology, Sikkim, India

PROJECT COMPLETION CERTIFICATE

This is to certify that the below mentioned students of Sikkim Manipal Institute of Technology have worked under my supervision and guidance from **22nd July 2024 to 27th June 2025** and successfully completed the Major project entitled **"Dev Automation of CPQ using Logik IO"** in partial fulfilment of the requirements for the award of Bachelor of Technology in Artificial Intelligence and Data Science.

University Reg. No	Name of Student	Course
202100516	**Madhurya Hait**	**B.Tech AI&DS**

Bhavesh Gandhi
External Guide
IT Manager
Keysight Technologies
Sector -39, Gurugram - 122001

Keysight Technologies International India Private Limited
Tower D, Ground Floor,1st to 5th Floor,
Unitech Cyber Park,
Sector 39, Gurugram,
Haryana -122001,India

Tel:+91-124-486-2999
+91-114-486-2327
www.keysight.com

CONFIDENTIAL

Jun 30, 2025

RE: Internship Verification Letter
Intern: Madhurya Hait
Intern ID: 01033600

To Whomsoever It May Concern

Please accept this letter as verification of the Internship with Keysight Technologies International India Private Limited for the above-mentioned Intern.

Employee Status	: Intern
Date of Joining	: July 22, 2024
Cumulative Length of Internship(years)	: 0.94
Job Title	: Intern Tech I
Stipend/Training Allowance	: INR 50,000/month

Warm Regards,

Sonam Yadav
Human Resources
Keysight Technologies International India Private Limited

Regd. Office: Omaxe Square, Plot No. 14, 2nd Floor, Non-Hierarchical, Jasola District Centre, Delhi-110025, India.
CIN No-U72300DL2014FTC264723

PROJECT REVIEW CERTIFICATE

This is to certify that the work recorded in this project report entitled **"Dev Automation of CPQ using Logik IO"** has been jointly carried out by **Madhurya Hait (Reg no: 202100516)** of Artificial Intelligence and Data Science Department of Sikkim Manipal Institute of Technology in partial fulfilment of the requirements for the award of Bachelor of Technology. This report has been duly reviewed by the undersigned and recommended for final submission for Major Project Viva Examination.

Sital Sharma
Department of Artificial Intelligence and Data Science
Sikkim Manipal institute of Technology
Majhitar, Sikkim – 737136

CERTIFICATE OF ACCEPTANCE

This is to certify that the below mentioned student of Artificial Intelligence and Data Science Department of Sikkim Manipal Institute of Technology (SMIT) have worked under the supervision of **Mr. Bhavesh Gandhi,** IT Manager, **Keysight Technologies**, from 22nd July 2024 to 27th June 2025 on the project entitled **"Dev Automation of CPQ using Logik IO".**

The project is hereby accepted by the Department of Artificial Intelligence and Data Science, SMIT in partial fulfilment of the requirements for the award of Bachelor of Technology in Artificial Intelligence and Data Science.

Registration No	Name of Student	Project Venue
202100516	**Madhurya Hait**	**Keysight Technologies, Sector -39, Gurugram - 122001**

Dr. Om Prakash Singh
Department of Artificial Intelligence and Data Science
Sikkim Manipal Institute of Technology
Majhitar, Sikkim – 737136

DECLARATION

I, the undersigned, hereby declare that the work recorded in this project report entitled "**Dev Automation of CPQ using Logik IO**" in partial fulfilment for the requirements of award of B.Tech (AI&DS) from Sikkim Manipal Institute of Technology (A constituent college of Sikkim Manipal University) is a faithful and bonafide project work carried out at "Keysight Technology, India" under the supervision and guidance of **Bhavesh Gandhi** Senior Manager of Keysight Technologies.

The results of this investigation reported in this project have so far not been reported for any other Degree or any other Technical forum.

The assistance and help received during the course of the investigation have been duly acknowledged.

Madhurya Hait

(Reg. No. 202100516)

ACKNOWLEDGEMENT

This project, has been a transformative journey, made possible through the unwavering support, guidance, and encouragement of numerous individuals. I extend my heartfelt gratitude to all those who contributed to its successful completion.

First and foremost, I express my sincere thanks to **Mr. Bhavesh Gandhi**, Senior Manager, **Keysight Technologies**, for his exceptional guidance, encouragement, and insightful mentorship throughout the project. His expertise and dedication were instrumental in shaping the outcomes of this work and provided me with invaluable knowledge and skills.

I am also deeply grateful to the esteemed faculty members of the Artificial Intelligence and Data Science Department at **Sikkim Manipal Institute of Technology**. In particular, I wish to thank **Dr. Om Prakash Singh**, Head of the Department, for his continuous support and motivation, which inspired me to undertake and successfully complete this endeavour.

A special note of appreciation goes to my mentors and guides, including Dr. Swarup Sarkar and Mr. Sital Sharma, for their constructive feedback, constant encouragement, and keen interest in my progress. Their invaluable advice has been a guiding light throughout this journey.

I am equally thankful to the management and team at Keysight Technologies for providing an opportunity to work on this advanced project and for fostering an environment conducive to learning and innovation. Their support, tools, and resources were pivotal in realizing this project's goals.

Madhurya Hait

(Reg. No. : 202100516)

DOCUMENT CONTROL SHEET

1	Report No	AI&DS/Major Project/External/B.Tech AI&DS/ 2025
2	Title of the Report	Dev Automation of CPQ using Logik IO
3	Type of Report	Technical
4	Author	Madhurya Hait (202100516)
5	Organizing Unit	Sikkim Manipal Institute of Technology
6	Language of the Document	English
7	Abstract	Developing a CPQ platform with Logik IO to modernise the way Quotation process was implemented traditionally using Oracle CPQ and automating various repetitive developement processes.
8	Security Classification	General
9	Distribution Statement	General

ABSTRACT

This presentation highlights the key learnings and contributions made during an ongoing internship at Keysight Technologies, within the IT department, with a specialized focus on Configure, Price, Quote (CPQ) solutions. The internship began with comprehensive training in Salesforce Administration, establishing a strong foundation in CRM fundamentals such as customer data management, workflow automation, and interaction optimization. Following this, hands-on experience was gained with Oracle CPQ and Logik IO two leading platforms for managing intricate product configurations and pricing mechanisms. The work involved analyzing business logic, configuring systems, and implementing optimizations.

A significant portion of the project was dedicated to supporting Keysight's strategic migration from Oracle CPQ to the next-generation Logik IO platform. This included evaluating legacy configurations, identifying core functionalities, and ensuring their effective replication or enhancement in the new system. Automation scripts were developed to streamline the migration of blueprints, data tables, and product information, while validation tools were implemented to ensure consistency and catch discrepancies across environments. Additional responsibilities included contributing to debugging and testing activities to uphold the reliability and efficiency of the migration process.

Some Key Highlights:

i. Salesforce CRM and CPQ platform expertise

ii. Oracle CPQ to Logik IO migration experience

iii. Automation and validation script development

iv. System debugging and multi-environment configuration testing

v. Exposure to real-world enterprise IT operations.

Keywords : Logik IO , CRM , Product Configuration, Quote, Automation Scripts, Migration Strategy

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF PSEUDO CODES

Chapter 1: INTRODUCTION

1.1. CRM (Customer Relationship Management) platforms play a vital role in today's business environment by helping organizations manage interactions with existing and prospective customers. These systems centralize customer information, streamline workflows through automation, and enhance overall engagement—leading to increased customer satisfaction and long-term loyalty.

Key aspects of CRM include:

1.1.1. Customer Data Management:

CRM platforms such as Salesforce serve as a unified hub for storing and managing customer data, including contact information, purchase records, communication history, and individual preferences. This comprehensive view helps businesses better understand customer behavior, allowing them to deliver tailored experiences and make informed, data-backed decisions.

1.1.2. Process Automation

CRM systems boost efficiency by automating routine activities like lead tracking, follow-ups, email campaigns, and report generation. This automation not only minimizes human error but also allows teams to allocate more time to high-value, strategic tasks while maintaining consistency in customer interactions.

1.1.3. Enhanced Customer Engagement

With built-in tools for targeted outreach, responsive support, and multichannel communication, CRMs help businesses connect more effectively with their customers. These capabilities support stronger relationships, higher customer retention, and increased brand loyalty over time.

1.2. Salesforce stands out as a top-tier CRM solution, providing a wide range of powerful tools and integrations suited for businesses across various scales. Its user-friendly design, advanced analytics features, and high level of customization make it a preferred platform for companies looking to strengthen their customer relationship strategies.

Fig 1.2: Salesforce Logo [8]

1.3. CPQ (Configure, Price, Quote) solutions are a specialized category of software designed to optimize the sales process, especially for businesses dealing with complex or highly customizable products and services.

Key functionalities of CPQ include:

- **1.3.1. Configuration:**

  CPQ systems assist users in choosing the appropriate product features and options by enforcing valid configuration rules. This helps prevent the selection of incompatible combinations, minimizing the chances of errors and the need for corrections.

- **1.3.2. Dynamic Pricing:**

  CPQ tools incorporate pricing rules, discount policies, and promotional offers to automatically calculate accurate prices. This ensures consistency in pricing across all sales channels while accommodating variables like volume discounts, regional pricing, or custom quotes.

- **1.3.3. Automated Quote Generation:**

  CPQ solutions streamline the creation of professional, error-free quotes. These quotes can be generated quickly, complete with detailed product specifications, terms, and conditions, significantly reducing turnaround time and improving the customer experience.

By automating these critical sales tasks, CPQ systems not only improve accuracy but also empower sales teams to focus on building relationships and closing deals. This leads to faster sales cycles, increased revenue, and enhanced customer satisfaction.

1.4. Logik IO represents the next evolution in CPQ solutions, offering advanced features and capabilities that address the limitations of traditional CPQ systems like Oracle CPQ.

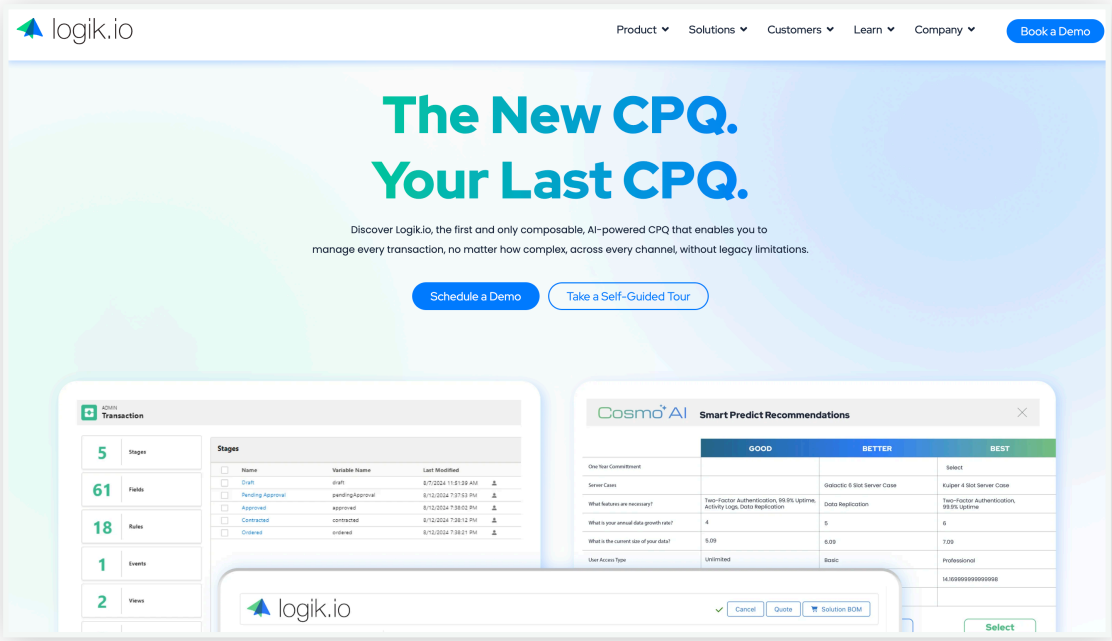

Fig 1.4: Logik IO website [6].

Why Logik IO stands out:

- **Faster Processing and Quote Generation:**

  Leveraging modern technology and optimized algorithms, Logik IO significantly reduces processing times for complex configurations and quotes. This enables businesses to respond to customer inquiries quickly, a crucial factor in today's competitive markets.

- **Enhanced Configuration Capabilities:**

  Logik IO offers an intuitive, user-friendly interface that simplifies the process of configuring even the most intricate products. Its powerful configurator supports advanced rules and logic, enabling sales teams to create tailored solutions effortlessly while ensuring accuracy and compliance.

- **Scalability and Flexibility:**

  Unlike legacy systems, Logik IO is built to adapt to the evolving needs of businesses. It can scale to accommodate growing product catalogs, complex pricing models, and diverse customer requirements. Its cloud-native architecture ensures seamless updates and integrations, keeping the system future-proof.

- **Improved User Experience:**

  Logik IO focuses on delivering a superior user experience for sales representatives and end customers alike. Its clean interface, faster navigation, and real-time feedback make it a joy to use, improving productivity and customer satisfaction.

- **Seamless Integration:**

  Logik IO integrates smoothly with existing CRM systems like Salesforce, ensuring a unified platform for managing customer interactions and sales processes.

By combining cutting-edge technology with a focus on user needs, Logik IO redefines the CPQ experience, enabling businesses to streamline their sales processes, enhance efficiency, and maintain a competitive edge in the market.

In summary, CRM platforms, CPQ solutions, and next-generation tools like Logik IO work in harmony to transform how businesses manage customer relationships and sales processes. Together, they empower organizations to deliver exceptional customer experiences, boost operational efficiency, and drive sustained growth.

Chapter 2: LITERATURE SURVEY

Table 1 : Literature Survey

Sl. No.	Author, Publisher, vol. year	Title of Paper	Inference	Research Gap	Relevance
1	**Näsi, Aleksi.** (2020). Journal of Business and Technology, 5(2), 45–52.	Enhancing sales & product management with CPQ systems. [10]	• CPQ systems improve sales accuracy and speed. • They streamline product configuration and management. • Integration with CRM/ ERP boosts operational efficiency. • The paper shows strong potential for business growth via CPQ. • Further research is needed for broader applicability and tech advancements.	The study lacks detailed empirical evidence and does not deeply explore the challenges faced after CPQ implementation. It overlooks the system's impact on customer behavior and gives limited attention to its adoption by SMEs. Emerging trends like AI integration and personalization in CPQ are also not addressed.	• Boosts sales efficiency through automation of quoting. • Simplifies product management for complex configurations. • Integrates with CRM and ERP systems for smoother operations. • Improves customer experience with faster, accurate quotes. • Supports data-driven decisions in pricing and sales. • Reduces training needs for sales teams. • Enables scalability in sales processes. • Drives digital transformation in sales.

Table 2 : Literature Survey

2	Jordan, Michelle & Auth, Gunnar & Jokisch, Oliver & Kühl, Jens-Uwe. (2020). Online Journal of Applied Knowledge Management. 8. 17-30. 10.36965/OJAKM.2020.8(2)17-30.	Knowledge based systems for the Configure Price Quote (CPQ) process - A case study in the IT solution business, 2020 [2]	• Knowledge-based systems improve CPQ accuracy and efficiency. • Expert knowledge formalization aids automation. • The case study validates practical benefits in IT solutions. • Highlights potential for broader application with further research.	While the study offers valuable insights into using knowledge-based systems in the CPQ process, it focuses mainly on a single case in the IT sector, limiting generalizability. It also lacks exploration of scalability, cross-industry applicability, and integration with emerging technologies like AI or machine learning.	• Highlights how knowledge-based systems enhance the CPQ process. • Demonstrates improved accuracy and consistency in complex IT solution sales. • Shows how expert knowledge can be formalized for automation. • Emphasizes better decision-making and reduced errors in configuration. • Provides a real-world case study, making findings practically applicable.

Table 3 : Literature Survey

3	(2024). Journal of Artificial Intelligence, Machine Learning and Neural Network. 4. 27-38. 10.55529/jaimlnn.46.27.38.	Machine Learning Applications in SalesForce CPQ Transforming Sales, 2024 [3]	• Machine learning enhances sales forecasting and pricing accuracy. • Automates complex configuration and quoting processes. • Improves personalization and customer targeting. • Drives efficiency and competitive advantage in sales. • Shows strong potential for future AI-driven CPQ innovations.	The study focuses on technological benefits but lacks detailed analysis of implementation challenges and user adoption. It also overlooks the impact on sales team dynamics and the integration of ethical considerations in AI-driven pricing decisions.	• Shows how machine learning improves pricing and sales accuracy. • Highlights automation of complex CPQ tasks. • Demonstrates enhanced customer personalization. • Emphasizes increased sales efficiency and competitiveness. • Supports adoption of AI in modern sales processes. • Offers early insights into digital transformation of sales processes. • Useful for understanding the evolution of configuration-based quoting systems.

Table 4 : Literature Survey

4	Hvam, L., Pape, S., & Nielsen, M. K. (2006). *Computers in Industry*, 57(7), 607-621. Elsevier.	Improving the quotation process with product configuration, 2006 [9]	• Product configuration significantly streamlines the quotation process. • Reduces errors and speeds up quote generation. • Enhances coordination between sales and engineering teams. • Improves overall efficiency and customer satisfaction. • Highlights the need for integrated IT solutions in sales.	The study focuses primarily on technical improvements in product configuration but provides limited insight into the organizational and human factors affecting quotation processes. It also lacks exploration of integration challenges with newer digital tools and scalability for diverse industries.	• Demonstrates how product configuration optimizes the quotation process. • Highlights benefits like reduced lead times and fewer errors. • Relevant for companies offering complex, customizable products. • Supports integration of sales and engineering workflows. • Provides a foundation for modern CPQ system development.

2.1 SUMMARY OF LITERATURE SURVEY

The following literature explores the evolution, impact, and technological advancement of **Configure, Price, Quote (CPQ)** systems across different industries, offering insights into how CPQ enhances sales processes, product configuration, and pricing efficiency.

I. **Title**: *Enhancing Sales & Product Management with CPQ Systems [10]*

This paper outlines the strategic role CPQ systems play in improving sales operations and product lifecycle management. It emphasizes how CPQ enables businesses to streamline quote generation, reduce manual errors, and deliver personalized product offerings, thus increasing customer satisfaction and sales team efficiency.

II. **Title**: *Knowledge-Based Systems for the Configure Price Quote (CPQ) Process – A Case Study in the IT Solution Business [2]*

This case study explores the implementation of knowledge-based systems in CPQ within the IT solutions sector. It highlights the integration of expert rules and domain knowledge to automate complex configurations, improving the accuracy and speed of quote generation in a dynamic product environment.

III. **Title**: *Machine Learning Applications in Salesforce CPQ: Transforming Sales [3]*

This modern study focuses on the integration of machine learning into Salesforce CPQ platforms. It presents how AI-driven insights enable predictive pricing, personalized quote recommendations, and automation in quote approval workflows making CPQ systems smarter, faster, and more adaptive to customer needs.

IV. **Title**: *Improving the Quotation Process with Product Configuration [9]*

This foundational research discusses how product configuration systems support the quotation process in complex manufacturing environments. The paper details methods to reduce configuration errors, improve product customization, and enhance quote accuracy laying the groundwork for modern CPQ system architectures.

The surveyed literature demonstrates that CPQ systems are evolving from rule-based engines to intelligent, AI-powered platforms. These systems not only improve the efficiency and accuracy of the quoting process but also contribute strategically to sales enablement and customer satisfaction across industries.

Chapter 3: PROBLEM DEFINITION

Keysight's current sales processes are significantly challenged by the limitations of its existing Configure, Price, Quote (CPQ) system, Oracle CPQ. As the company deals with a growing portfolio of highly customizable products and increasingly complex business rules, the ability to manage and generate accurate quotations has become increasingly difficult.

One of the most critical issues is the extended time required to generate quotes. Oracle CPQ performs poorly when handling large datasets and complex rule structures, leading to slow processing speeds and frequent system lags. These delays negatively impact sales performance, customer responsiveness, and overall user satisfaction. Moreover, the system's outdated interface and rigid structure make it difficult for users to customize workflows or tailor the platform to their specific needs, resulting in a steep learning curve and reduced productivity for the sales and operations teams. To overcome these limitations, the organization has initiated a transition to a modern CPQ platform, Logik IO. However, this migration process presents its own set of complex challenges. The move involves transferring thousands of interconnected products, rules, fields, and blueprint components from Oracle CPQ to Logik IO. This process is largely manual, making it time-consuming and prone to human error. Inconsistencies between different instances often lead to missing data, misconfigured rules, and broken associations between blueprint elements all of which compromise the integrity of the new configuration environment.

Compounding the issue is the lack of an automated system to track, compare, or auto-associate blueprint components across environments. Manual validation and mapping are not scalable and result in significant delays, increasing the risk of data loss and operational disruptions. Additionally, when technical issues arise such as urgent bugs developers must resort to time-intensive debugging methods.

In summary, the organization faces a dual-layered problem: first, an outdated CPQ system that limits sales efficiency and agility. second, a highly complex and under-automated migration process to a more advanced platform.

Chapter 4: OBJECTIVES

To address the growing inefficiencies and technical limitations of its current Oracle CPQ system, Keysight has initiated a strategic migration to the more modern and flexible Logik IO platform. This transition aims not only to enhance sales performance and user experience but also to streamline the complex migration process and ensure long-term scalability. The following objectives outline the key goals driving this initiative:

I. Improve Quote Generation Efficiency

- Minimize the time taken to configure and generate sales quotes by replacing Oracle CPQ with Logik IO, which can better handle complex configurations and datasets.

II. Enhance System Usability and Flexibility

- Provide a modern, intuitive interface and customizable workflows to reduce the learning curve and boost productivity for sales and operations teams.

III. Automate and Streamline the Migration Process

- Develop or integrate tools that automate the transfer of product data, business rules, and blueprint elements from Oracle CPQ to Logik IO to reduce manual effort and human error.

IV. Ensure Data Integrity and Configuration Accuracy

- Implement validation mechanisms to track, compare, and auto-associate blueprint components across different environments, ensuring consistency and accuracy post-migration.

V. Reduce Operational Risks During Migration

- Mitigate potential data loss, misconfigurations, and disruptions by introducing scalable and systematic migration practices.

Chapter 5: SOLUTION STRATIGY

To address the limitations of the legacy Oracle CPQ system and ensure a seamless transition to Logik IO, a comprehensive solution strategy has been developed. This approach not only resolves existing inefficiencies but also establishes a scalable, high-performance CPQ environment tailored to the organization's evolving needs. The strategy is organized across five key areas, each designed to support a smooth migration, optimize operations, and promote long-term success. The strategy is structured into five key components:

I. Strategic Migration Planning

The first phase of the solution strategy involves an in-depth analysis of the current Oracle CPQ setup. This includes reviewing all product configurations, pricing rules, determination logic, and system dependencies. By thoroughly understanding the existing blueprint, the team is able to identify bottlenecks, eliminate redundant logic, and restructure complex dependencies in preparation for a more optimized architecture in Logik IO. This analysis also helps to establish a cleaner foundation for migration and ensures that only necessary, well-structured components are carried forward. The goal at this stage is to simplify the legacy system’s complexity and prepare a strategic roadmap that guides a phased and controlled migration process.

II. Automated Data Handling and Blueprint Management

Given the scale of the migration which involves thousands of products, fields, rules, and configuration sets manual handling is both impractical and error-prone. To overcome this, custom automation scripts were developed to support blueprint management. These scripts are capable of comparing instances, identifying discrepancies, highlighting missing fields or validations, and auto-associating related configuration elements. This automation dramatically reduces manual effort, accelerates the migration process, and ensures consistency between

environments. Additionally, when bugs are reported, the automation logic helps identify root causes with precision, often tracing issues back to small but critical configuration errors. This capability significantly improves debugging speed and system reliability. Blueprint migrations are triggered automatically when specific gaps or misalignments are detected, ensuring seamless updates and system integrity across all environments.

III. Integration with various platforms like Salesforce and System Optimization

A critical component of the migration strategy is the integration of Logik IO with Salesforce CRM. This ensures that customer data, sales workflows, and configuration logic operate cohesively within a single ecosystem. Logik IO's architecture complements Salesforce by enabling dynamic data synchronization and automated task handling, resulting in improved operational efficiency. The platform's logic-driven approach supports guided selling, which helps sales teams navigate complex product configurations, apply pricing, discounts, purchase agreements with confidence and minimal errors. Additionally, Logik IO's advanced pricing engine supports complex discounting structures and dynamic pricing models, delivering accurate, real-time quotes. These enhancements not only improve internal efficiency but also elevate the customer experience by enabling faster response times and highly tailored sales interactions.

IV. Enhanced User Experience and System Adoption

Improving user experience is a central focus of the new CPQ system. Logik IO offers a modern, intuitive interface that significantly reduces the learning curve for sales representatives and system administrators. This ease of use facilitates faster adoption and minimizes disruption during the transition. Furthermore, the platform's high degree of customization empowers teams to adapt workflows and configurations to meet specific business requirements without relying heavily on technical resources. This flexibility supports increased productivity, as users can quickly tailor the system to suit unique client needs, sales strategies, or market

demands. By providing a streamlined and accessible interface, Logik IO creates a user-centric environment that enhances engagement and performance across the organization.

V. Rigorous Testing and Continuous Improvement

Quality assurance is embedded at every stage of the solution strategy to ensure that the migration is robust, reliable, and aligned with business objectives. Post-migration, each configuration, pricing rule, and workflow undergoes detailed validation to verify functional accuracy and data integrity. Stress testing is conducted to assess the system's performance under realistic workloads, helping to uncover and resolve any potential bottlenecks before full deployment. In addition to technical testing, continuous improvement is driven by user feedback and performance analytics. Support issues are managed efficiently through structured tools like Jira, which streamline issue tracking and resolution. Logik IO's analytics features also provide valuable insights into usage trends, system behavior, and user interactions, enabling ongoing enhancements. This iterative improvement process ensures that the platform evolves in alignment with organizational goals and changing market conditions.

Chapter 6: FEATURES OF CPQ SYSTEMS

6.1 Introduction to CPQ Solutions

Configure, Price, Quote (CPQ) solutions are a specialized category of software designed to streamline and optimize the sales process, particularly for businesses offering complex or highly customizable products and services. CPQ tools reduce manual workload, minimize errors, and enhance the overall customer experience by automating key aspects of the quoting process.

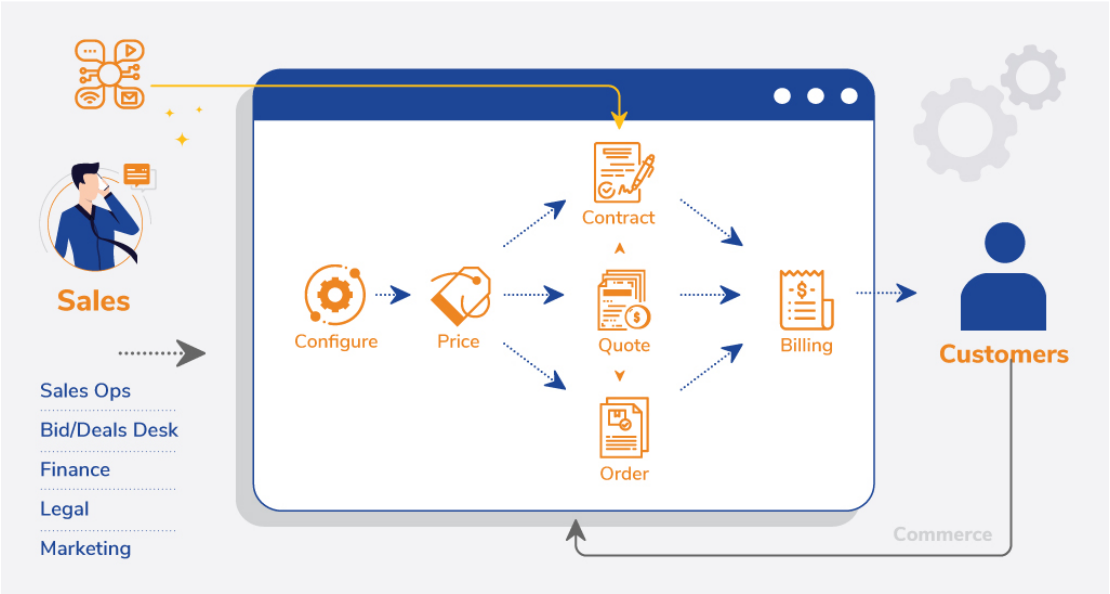

Fig 6.1 : diagrammatic representation of a Quotation process [8]

6.2 Core Functionalities of CPQ

6.2.1 Configuration

CPQ systems guide sales representatives through product selection, ensuring that only compatible and valid configurations are presented. This prevents errors related to incompatible options and minimizes rework.

6.2.2 Dynamic Pricing

CPQ tools apply pricing rules, discount structures, and promotions to automatically generate accurate pricing. This helps ensure consistency across different sales channels while accommodating various pricing models, such as regional pricing or volume-based discounts.

6.2.3 Automated Quote Generation

By automating the creation of professional and accurate quotes including product details, pricing, terms, and conditions. CPQ systems significantly reduce quote turnaround time and enhance the customer experience.

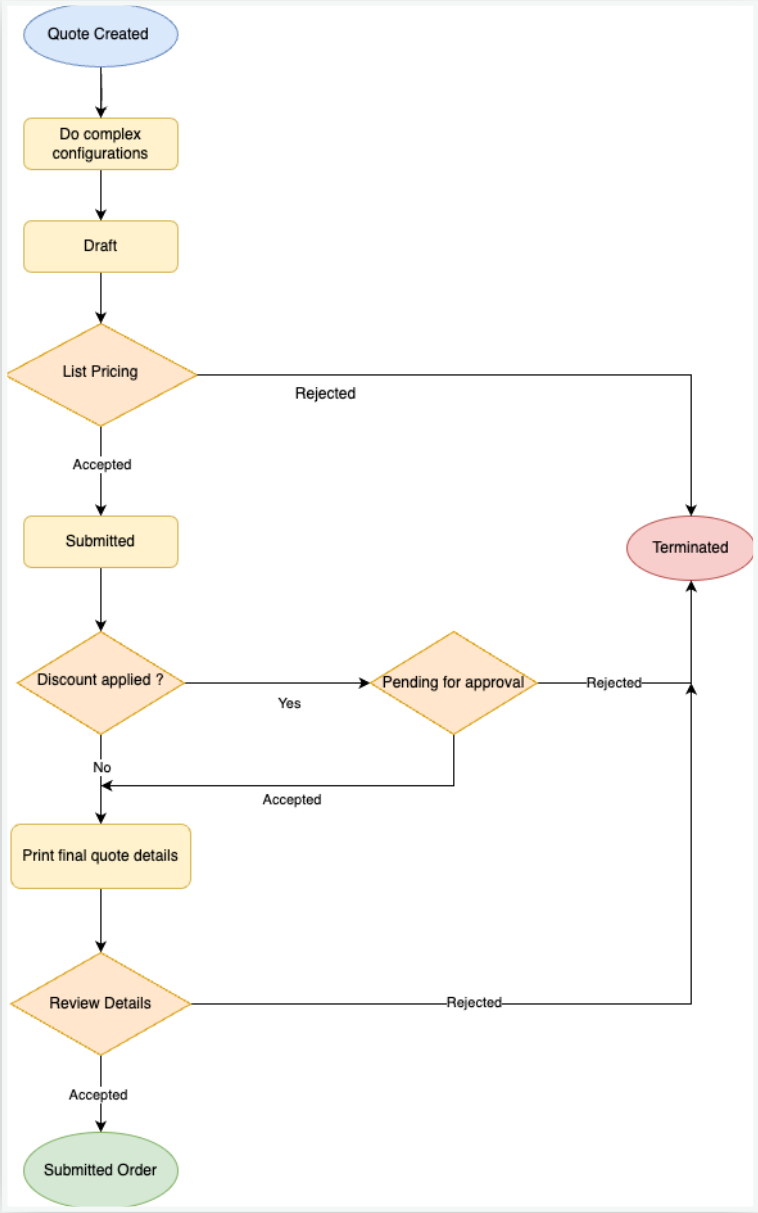

Fig 6.2 : Flow diagram of a Quotation process

6.3 Business Impact of CPQ Solutions

CPQ systems have a direct and measurable effect on key business outcomes, including:

- Shortened sales cycles
- Increased company-wide productivity
- Higher deal values
- Improved brand presence and customer engagement
- Simplified business processes
- Streamlined pricing and discount controls
- Structured approval flows
- Enhanced self-service for customers and partners

6.4 Department-Specific Benefits of CPQ

6.4.1 CPQ for Sales

Sales teams benefit from faster and more accurate quoting, centralized discount logic, and guided selling interfaces. CPQ improves:

- Sales efficiency and productivity
- Configuration and pricing accuracy
- Discount approval workflows
- Omnichannel access to pricing and quotes

6.4.2 CPQ for Sales Operations

CPQ simplifies administrative tasks and enhances operational efficiency, offering:

- Better visibility into sales funnel
- Streamlined approval processes
- Improved forecasting accuracy
- Faster quote cycle completion

6.4.3 CPQ for Finance

Finance teams use CPQ to manage pricing governance, analyze margins, and enforce terms. Key improvements include:

- Enforcement of financial terms and policies
- Transparent discount visibility
- Enhanced margin analysis
- Revenue recognition and assurance

6.4.4 CPQ for Marketing

With seamless CRM and ERP integration, CPQ supports marketing in delivering targeted and profitable campaigns. Benefits include:

- Improved campaign performance
- Higher promotion return on investment
- Streamlined B2B eCommerce

6.4.5 CPQ for Legal

CPQ tools ensure legal accuracy and consistency across all quote documents. Legal teams benefit from:

- Easier proposal review and redlining
- Accurate inclusion of pricing and product details
- Predefined terms and conditions compliance

6.4.6 CPQ for Order Operations

By connecting CPQ with CRM, operations teams improve order processing through:

- Higher conversion of quotes to orders
- Reduced manual entry & faster order cycle times

Chapter 7: QUOTATION PROCESS

Quotation has various stages as you can see when a customer quote is created it starts from a draft state then proceeds towards converting into a potential customer.

In **Logik IO**, the quote-to-order process is a highly efficient and streamlined sequence of activities designed to enhance sales operations while prioritizing accuracy and customer satisfaction. This process leverages Logik IO’s advanced CPQ capabilities to ensure seamless transitions between each stage, creating a cohesive workflow that minimizes errors and optimizes efficiency.

The process begins with the **Quote Creation** stage, where sales representatives or customers initiate a quote request. At this stage, the user outlines specific needs and requirements, providing the necessary details to customize the product or service offering. Logik IO's intuitive interface and guided selling tools make it simple for users to navigate through this step, ensuring clarity and precision in capturing the customer's demands.

Following this, the **Configure Your Product** step allows users to tailor the product or service to meet their unique needs. Logik IO’s robust configuration engine enables users to select from a variety of options, features, and add-ons, all while ensuring compliance with pre-defined business rules and dependencies. The system's flexibility ensures that even the most complex configurations can be managed effortlessly, eliminating the risk of incompatible or erroneous selections.

Once the product configuration is complete, the system dynamically calculates **List Pricing**, factoring in multiple variables such as applicable discounts, promotions, tiered pricing structures, and regional adjustments. This dynamic pricing capability ensures that quotes are not only accurate but also reflective of the customer’s specific circumstances. By automating this step, Logik IO eliminates manual errors and accelerates the pricing process, providing customers with real-time cost insights. When the order is received, the quote is then updated with the order information and converted to an order.

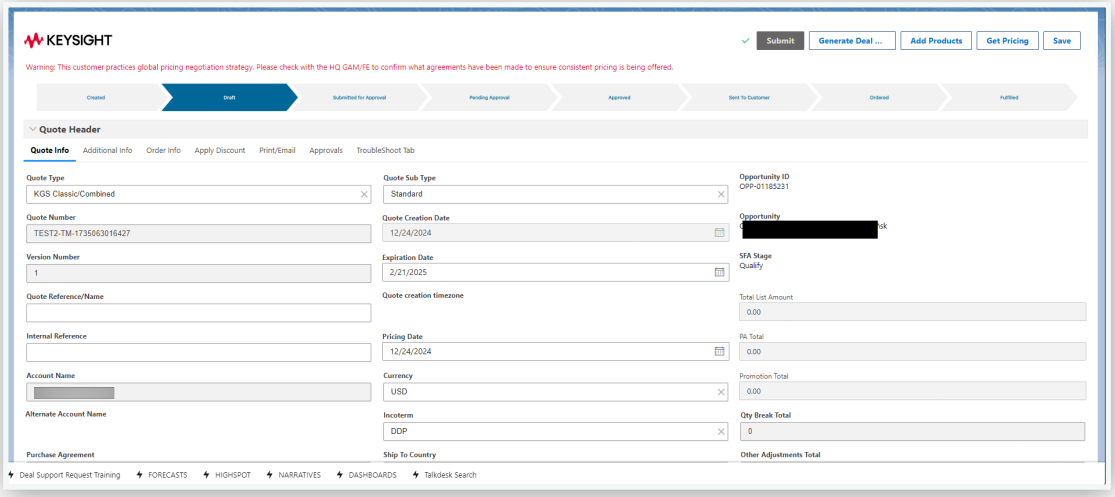

Fig 7.1 : Logik IO Quote creation UI. [4]

The next phase is **Submission for Verification**, where the generated quote is sent to a designated approver for review. During this step, the approver ensures that the quote aligns with internal policies, pricing guidelines, and contractual obligations. This layer of verification is crucial in maintaining compliance and upholding the company's standards, ensuring that every quote is both accurate and appropriate.

Upon approval, the quote moves to the **Final Quote Generation** stage. Here, Logik IO produces a professional and polished quote document, complete with all relevant details such as product specifications, pricing breakdowns, and terms and conditions. This document is then sent to the customer for their review. The system supports version control and tracking, enabling easy collaboration and updates if adjustments are required.

Once the customer verifies the quote and provides their confirmation, the process transitions to the **Order Creation** stage. At this point, the quote is converted into a formal order, which is then handed over to other departments such as fulfillment, procurement, or logistics for further processing. The seamless integration with downstream systems ensures that the order is executed efficiently, maintaining alignment across teams.

Throughout this entire process, **Logik IO** ensures that all stakeholders remain aligned, from sales representatives and approvers to customers and operational teams. Its advanced CPQ functionalities streamline the workflow, reduce bottlenecks, and

provide real-time visibility into each stage of the quote-to-order lifecycle. By automating repetitive tasks, simplifying complex configurations, and ensuring data accuracy, Logik IO delivers a customer-centric experience that enhances trust, improves operational efficiency, and drives overall business success.

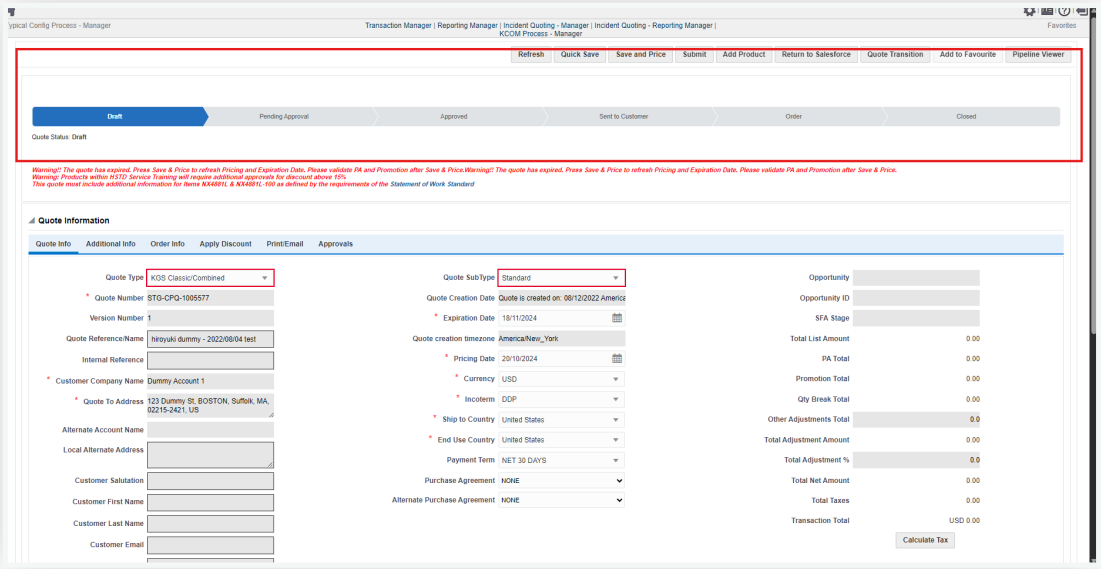

Fig 7.2 : Oracle CPQ Quote creation UI [3].

Chapter 8: TECH STACK

The technologies used in the project encompass powerful tools and languages that facilitate efficient data management, configuration, and customization within the CPQ domain. Each component of the tech stack serves a distinct purpose in streamlining operations and enhancing the platform's capabilities.

Here's an expanded explanation of each:

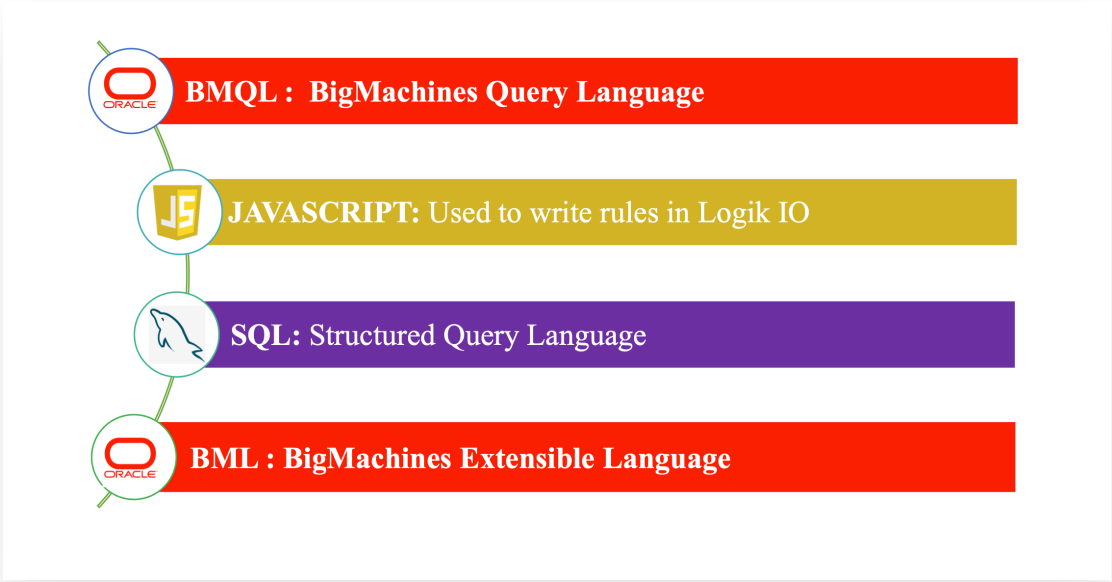

Fig 8 : Tech Stack used

8.1. BMQL (BigMachines Query Language):

BMQL is a specialized query language used in Oracle CPQ (formerly BigMachines) to interact with the system's data model. It allows users to retrieve, filter, and manipulate data stored within Oracle CPQ. BMQL is highly optimized for CPQ-specific use cases, such as fetching configuration rules, pricing logic, and user-specific data.

BMQL is central to managing dynamic and conditional logic in Oracle CPQ. It enables the retrieval of relevant data for pricing, configurations, and rules that adapt to customer requirements.

8.2. SQL (Structured Query Language):

SQL (Structured Query Language) is a fundamental tool used for interacting with relational databases. It enables efficient handling of structured data, including storing, retrieving, and updating records. In the context of CPQ systems and Logik IO, SQL is frequently utilized in backend processes to maintain data consistency and facilitate integration with CRM platforms and other databases. It supports essential tasks such as creating and managing tables, organizing product catalogs, and processing large volumes of data. Additionally, SQL plays a crucial role in data migration activities, helping ensure accuracy and consistency during system transitions.

8.3. JavaScript:

JavaScript is a flexible and widely adopted programming language used in both front-end and back-end development. In CPQ platforms, it plays a key role in making user interfaces interactive and responsive, enabling smooth navigation and dynamic content rendering. It is also used to implement custom business logic, allowing developers to tailor workflows, validations, and trigger-based actions to meet specific operational needs. This enhances both functionality and user experience within the CPQ environment.

8.4. BML (BigMachines Language):

BML is a proprietary scripting language used in Oracle CPQ for creating complex business logic. It is highly tailored to the needs of CPQ systems, allowing developers to write custom functions and formulas to implement advanced configurations, pricing rules, and validations.

BML is used to handle intricate business rules and logic that cannot be achieved through standard configurations. It supports conditional logic, looping constructs, and calculations to address complex scenarios.

Chapter 9: METHODOLOGY

This chapter outlines the structured approach followed during the integration of **Logik IO** with **Salesforce**, aiming to enhance sales operations through intelligent automation, dynamic configuration capabilities, and seamless platform connectivity. The methodology was centered around streamlining complex product configurations, ensuring real-time data consistency, and building scalable, user-friendly workflows.

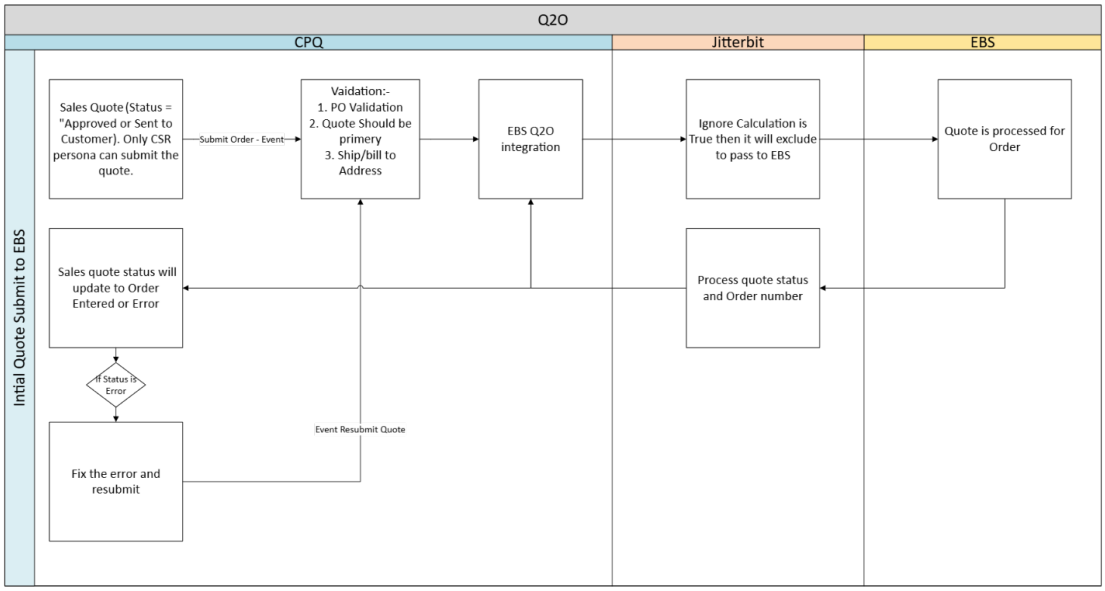

Fig 9.1 : Logik IO Quote to Order architecture

I. Integration Strategy with Logik IO

The methodology began by embedding Logik IO's advanced configuration engine within Salesforce to form a unified and efficient quoting environment. The configuration engine was designed to handle layered product dependencies, customizable options, and region-specific business logic with minimal manual intervention.

Key activities included:

- Defining integration points between Salesforce and Logik IO.
- Setting up product structures and configuration rules.
- Assigning user roles and access controls to maintain process integrity.

II. Data Synchronization and Configuration Setup

Real-time data synchronization between Salesforce and Logik IO was enabled to ensure consistency across platforms. Any updates to pricing logic, customer information, or product catalogs were instantly reflected in both systems. This eliminated version mismatches and ensured faster response times during quote generation.

Pricing strategies were implemented using conditional logic, enabling:

- Automatic calculation of discounts and promotional pricing.
- Dynamic adjustment based on customer tiers or location.
- Accurate pricing throughout the sales process without manual edits.

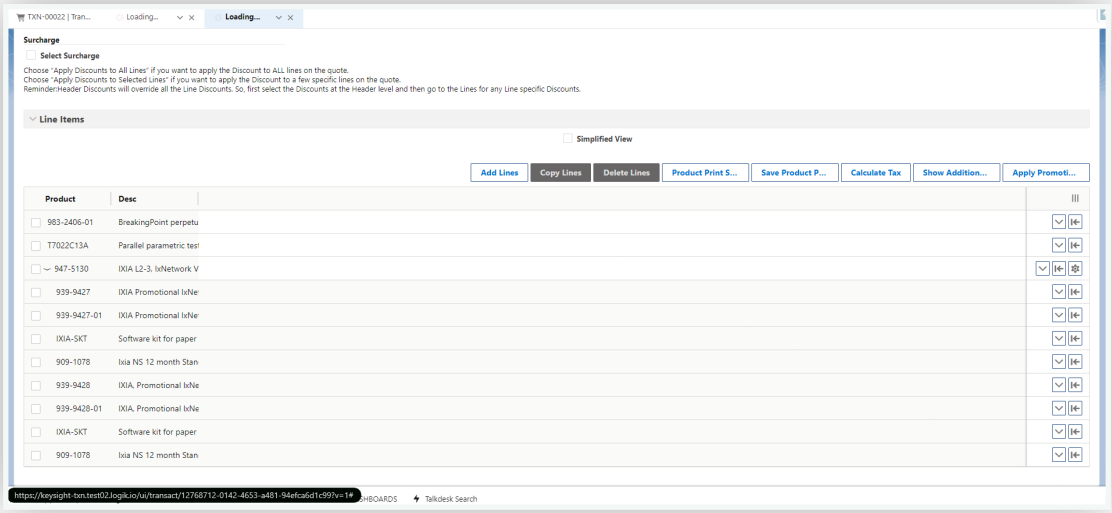

Fig 9.2 : Logik IO Product configuration UI. [4]

III. Guided Configuration and Workflow Control

To simplify complex quoting scenarios, guided configuration logic was introduced. This ensured that sales representatives were directed through appropriate options based on predefined compatibility rules. As a result, incorrect product combinations or pricing errors were significantly reduced.

Additionally, validation mechanisms were embedded at each step to confirm that all required inputs were complete before the process advanced, improving accuracy and reducing rework.

IV. Modular Design and Scalability

The system was built using a modular architecture, allowing flexibility for future enhancements. New product types, regional business logic, or evolving pricing structures could be added without affecting the existing configuration setup. This adaptability ensured that the system could evolve with business growth.

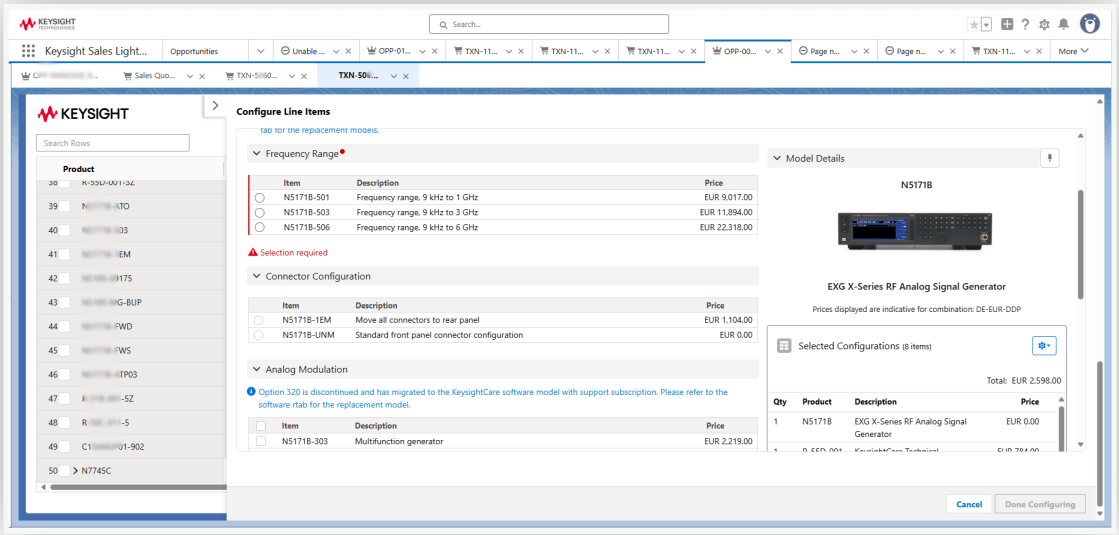

Fig 9.3 : Logik IO Product configuration UI. [4]

Chapter 10: LOGIK IO (Development)

Logik IO is primarily built using JavaScript for its core development and relies heavily on SQL for data retrieval and manipulation. The administrative interface is cloud-based, allowing users to configure and customize the platform directly through the web. It offers a range of features and settings, which we will explore in the following sections.There are lots of special Stages play a crucial role in managing the workflow within a CPQ (Configure, Price, Quote) process. By segmenting the transaction into clearly defined steps, businesses can ensure that each phase of the sales cycle is completed accurately. This structured approach allows sales teams and approvers to monitor progress, make timely adjustments, and verify critical details at every stage. It promotes clarity, consistency, and coordination across departments, ultimately enhancing efficiency and minimizing errors. Furthermore, incorporating validation at each stage ensures that essential data is collected and confirmed before proceeding, leading to more precise and reliable final orders.

The schema of CPQ is :

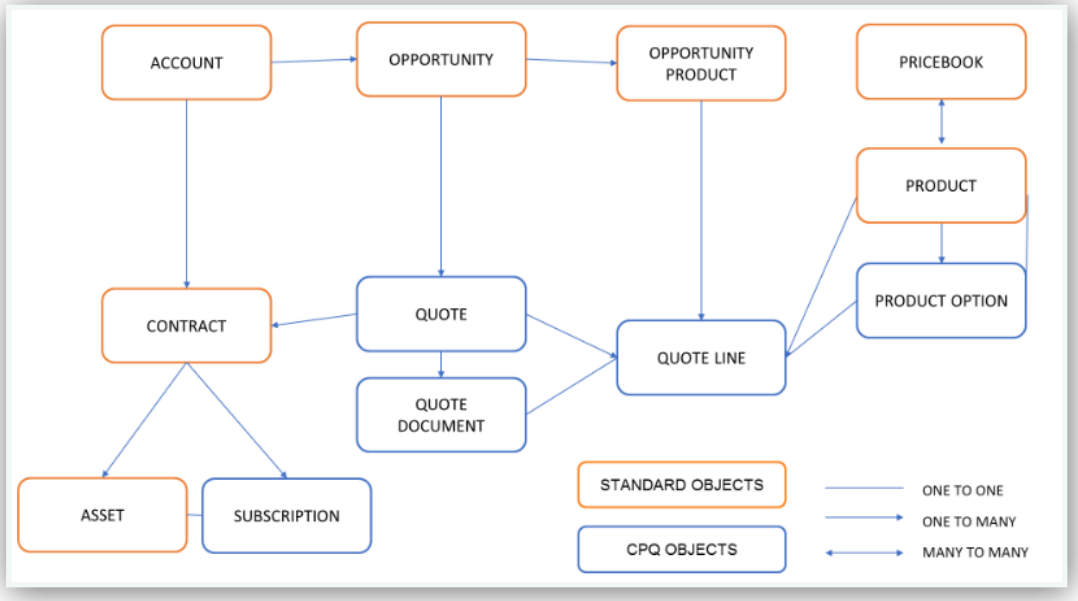

Fig 10 : CPQ Data Model [8].

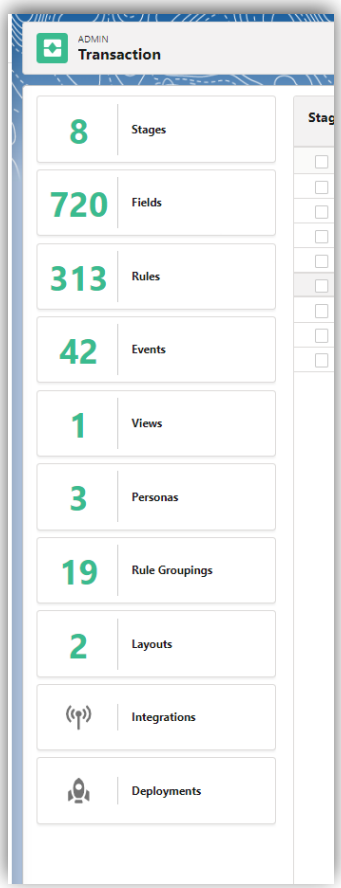

Fig 9.1 : Logik IO Developement UI [5].

10.1. Stages

Stages represent the distinct phases that a transaction goes through in a CPQ (Configure, Price, Quote) process. These stages encapsulate the key steps that a quote or order undergoes from the beginning of the sales cycle to its completion. Typical stages in a CPQ process include **Configuration**, **Pricing**, **Approval**, and **Deployment**. Each of these stages plays a critical role in shaping the transaction, ensuring that the process is handled systematically and efficiently.

10.1.1. Created

This is the initial stage where a new quote or configuration is generated. At this point, the basic quote structure exists but has not yet been developed or validated. It's often initiated when a sales rep begins working on a customer request.

10.1.2. Draft

In this stage, the quote is actively being built or modified. Product configurations are selected, pricing logic is applied, and discounts (if any) are considered. The draft is not yet ready for review or submission, allowing users to make iterative changes.

10.1.3. Submitted for Approval

Once the quote is finalized by the sales team, it is formally submitted into the approval workflow. This triggers a predefined approval process based on rules such as discount thresholds, deal size, or non-standard terms.

10.1.4. Pending Approval

At this stage, the quote is waiting for review by the appropriate approvers such as sales managers, finance, or legal. No further edits are allowed unless it's rejected or returned for changes.

10.1.5. Approved

The quote has successfully passed all required approval checks and is now considered finalized internally. It is ready to be shared with the customer for review or signature.

10.1.6. Sent to Customer

The approved quote is formally presented to the customer. This may include sharing a digital or physical document outlining the agreed configuration, pricing, terms, and conditions.

10.1.7. Ordered

Once the customer accepts the quote, it is converted into an order. This marks the transition from a sales opportunity to a committed transaction and typically triggers backend processes such as procurement and invoicing.

10.1.8. Fulfilled

The final stage where the product or service is delivered to the customer. All commitments outlined in the quote and order are completed. Fulfillment also includes updates to inventory, shipping status, and closing financials.

10.2. Fields

Fields in a CPQ system are custom data points or attributes that are used to capture and store the transactional details associated with a quote or order. These fields represent different types of information that are necessary for completing a

transaction, including **customer details**, **product specifications**, and **pricing inputs**. Fields can be categorized into different types based on the nature of the data they capture.

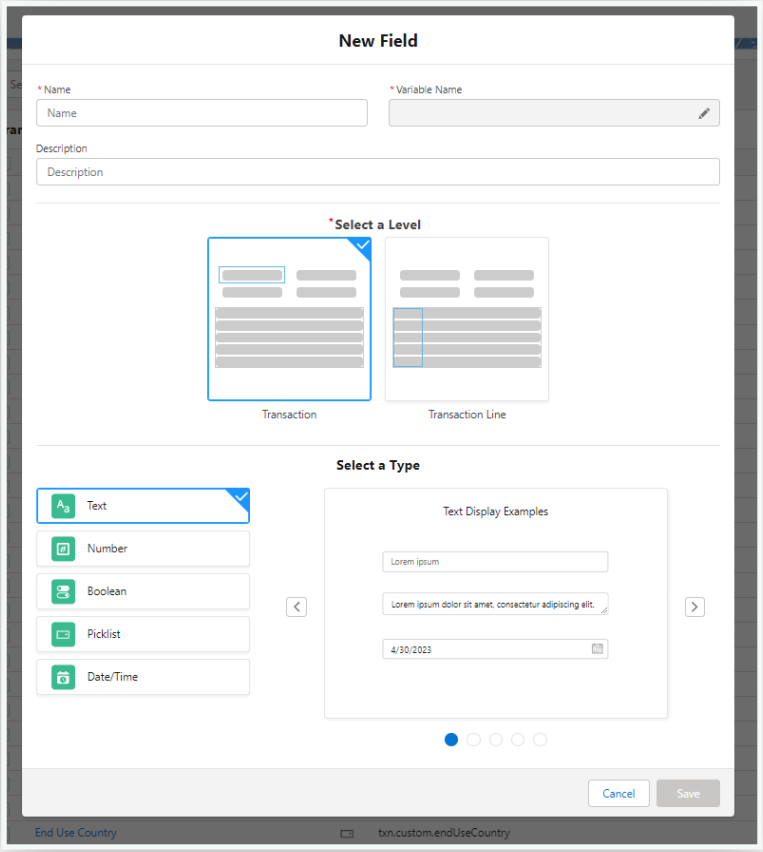

Fig 10.2 : Logik IO Field Creation [5].

Common field types include:

- **Text Fields:** These are used for capturing alphanumeric data such as customer names, product descriptions, or notes.
- **Number Fields:** These capture numerical data, such as quantities, prices, or discount percentages.
- **Boolean Fields:** These represent true/false values and are used for binary choices, such as whether a particular option is selected or not.
- **Picklist Fields:** These provide a list of predefined options from which the user can select, such as product categories, regions, or payment terms.

- **Date/Time Fields:** These capture dates and times, useful for tracking deadlines, delivery schedules, or contract durations.

10.3. Rules

Rules in a CPQ system refer to the logical constraints or conditions that are applied to ensure valid, accurate, and optimal configurations during the sales process. These rules automate decision-making, reduce manual intervention, and enforce consistency

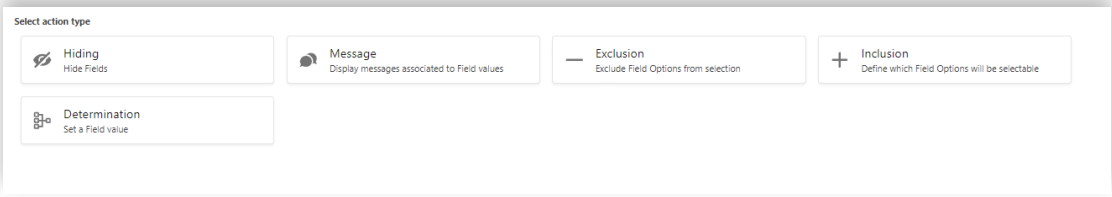

Fig 10.3.1 : Logik IO Various rule options [5].

across transactions by defining specific behaviors that must be followed at each stage. The types of rules in a CPQ system include:

- **Hiding Rules:** These rules allow the system to hide certain fields or options based on specific conditions. For example, if a customer selects a particular product option, other irrelevant options might be hidden from view, simplifying the configuration process.
- **Message Rules:** These rules display custom messages or error alerts when certain conditions are met. For instance, if an invalid combination of product options is selected, the system could show an error message instructing the user to adjust their selection.
- **Exclusion Rules:** Exclusion rules prevent certain options or features from being selected when specific conditions are met. For example, if a customer selects a specific product feature, an incompatible option might be automatically excluded from selection.
- **Inclusion Rules:** Inclusion rules ensure that specific options or features are available for selection when predefined conditions are met. For example, if a

customer selects a particular package, additional related products might be automatically included for selection.

- **Determination Rules:** These rules set or populate specific fields based on the values in other fields. Determination rules can be relatively simple (e.g., setting a field value based on a picklist selection) or more complex (e.g., using a set of rules to calculate pricing based on multiple criteria).

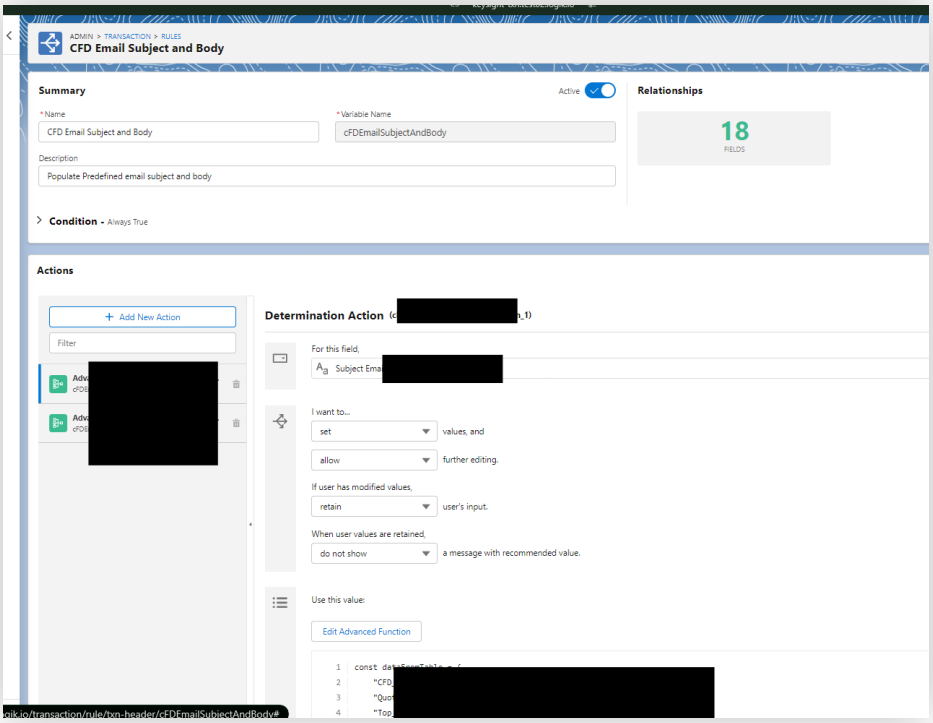

Fig 10.3.2 : Logik IO Rule Creation. [5]

10.4. Events

Events are triggers within the CPQ system that execute specific actions based on predefined conditions or criteria during a transaction. These events are set to activate when certain actions occur or conditions are met, such as when a user selects a product option, updates data, or moves between stages in the sales process. Events can initiate a variety of actions, including data updates, notifications, recalculations, and triggering additional business rules.

Examples of events include:

- **Data Updates:** Automatically updating fields or records when a selection or input is made.

- **Notifications:** Sending alerts or emails to stakeholders when specific conditions are met, such as when a quote is ready for approval.
- **Recalculations:** Triggering price recalculations or discount adjustments when certain conditions change, like applying a discount based on the quantity selected.

10.5. View

Views in a CPQ system refer to the customizable user interfaces that are designed to display transactional data in an organized and user-friendly manner. Views allow users to interact with the system's data according to their specific needs, presenting information in a way that is relevant to their role within the process. Different users may have different views depending on their access and responsibilities, ensuring that they are presented with the right data and tools for their tasks.

For example:

- **Sales Representative View:** Displays product options, pricing details, and customer-specific configurations to facilitate the quote creation process.
- **Manager View:** Provides a high-level overview of quotes, approvals, and the ability to track sales performance metrics.
- **Admin View:** Displays system settings, user configurations, and management tools for system maintenance and configuration.

10.6. Personas

Personas refer to the distinct user roles or profiles within the CPQ system that define the access level, functionality, and user experience for each type of user. These roles are tailored to specific tasks and responsibilities within the sales and configuration process, ensuring that each user interacts with the system in a way that aligns with their needs. Common personas in CPQ systems include Sales Representatives, Managers, and Administrators, each with different levels of access to data and system functionalities.

Examples of personas include:

- **Sales Rep Persona:** Access to product catalogs, pricing tools, and customer-specific configurations for generating quotes.
- **Manager Persona:** Access to dashboards, sales performance data, and approval processes for overseeing quote-to-order workflows.
- **Administrator Persona:** Access to system configurations, user permissions, and settings for managing the CPQ environment.

10.7. Rule Grouping

Rule grouping refers to the logical bundling of related rules within the CPQ system to simplify management, deployment, and execution. Rather than managing each individual rule separately, related rules can be grouped together based on their function or purpose, such as pricing rules, configuration rules, or approval rules. These groups allow for better organization and ensure that related rules are applied consistently across the system.

For example:

- **Pricing Rule Group:** Contains all rules related to pricing, including discounts, price adjustments, and currency conversions.
- **Configuration Rule Group:** Includes rules that define valid product configurations, options, and exclusions based on customer requirements.
- **Approval Rule Group:** Contains rules that define the approval workflows and conditions under which a quote needs managerial or executive approval.

10.8. Layout

Layout refers to the arrangement and design of the user interface (UI) elements within a CPQ system. It determines how data fields, product options, configuration steps, and other components are presented to users. The layout is a crucial aspect of the user experience (UX), as it defines how easily users can navigate the system and perform tasks such as creating quotes, selecting product configurations, or reviewing pricing

details. A well-designed layout ensures that the flow of information is clear, logical, and efficient.

For example:

- **Product Configuration Layout:** Organizes product options and configuration steps in a logical order, allowing users to easily navigate between different configuration options.
- **Quote Creation Layout:** Displays fields for customer information, pricing, and additional terms in a clear sequence to facilitate accurate quote generation.
- **Approval Layout:** Groups approval options and review fields in a way that allows managers or approvers to efficiently evaluate and approve quotes.

10.9. Integrations

Integrations refer to the connections between the CPQ system and external systems, such as Salesforce CRM, ERP platforms, or payment gateways. These integrations enable seamless data exchange and communication between different systems, ensuring that information flows smoothly across platforms. For instance, a CPQ system may integrate with Salesforce CRM to pull customer data or with an ERP system to check inventory and fulfill orders. Payment gateway integrations allow for real-time payment processing during the quote-to-order process.

Examples of integrations:

- **Salesforce CRM Integration:** Automatically sync customer data, opportunities, and contacts from Salesforce into the CPQ system for use in quotes.
- **ERP Integration:** Retrieve real-time inventory data, product availability, and shipping information to ensure that quotes are accurate and align with product stock levels.
- **Payment Gateway Integration:** Enable secure and real-time processing of customer payments directly from the CPQ platform.

10.10. Deployments

Deployments refer to the process of rolling out configuration changes, rules, or updates to the live production environment within a CPQ system. This involves transferring new or modified configurations, pricing models, business rules, or user interface updates from the development or testing environment into the live environment where they will be used by end-users. Deployments are typically carried out following rigorous testing to ensure that the new features or changes work as expected without disrupting ongoing transactions or processes.

Examples of deployments include:

- **Configuration Updates:** Rolling out changes to product configurations or pricing models based on new business rules or customer requirements.
- **Business Rule Modifications:** Deploying updates to pricing or exclusion rules to reflect changes in product offerings or business strategies.

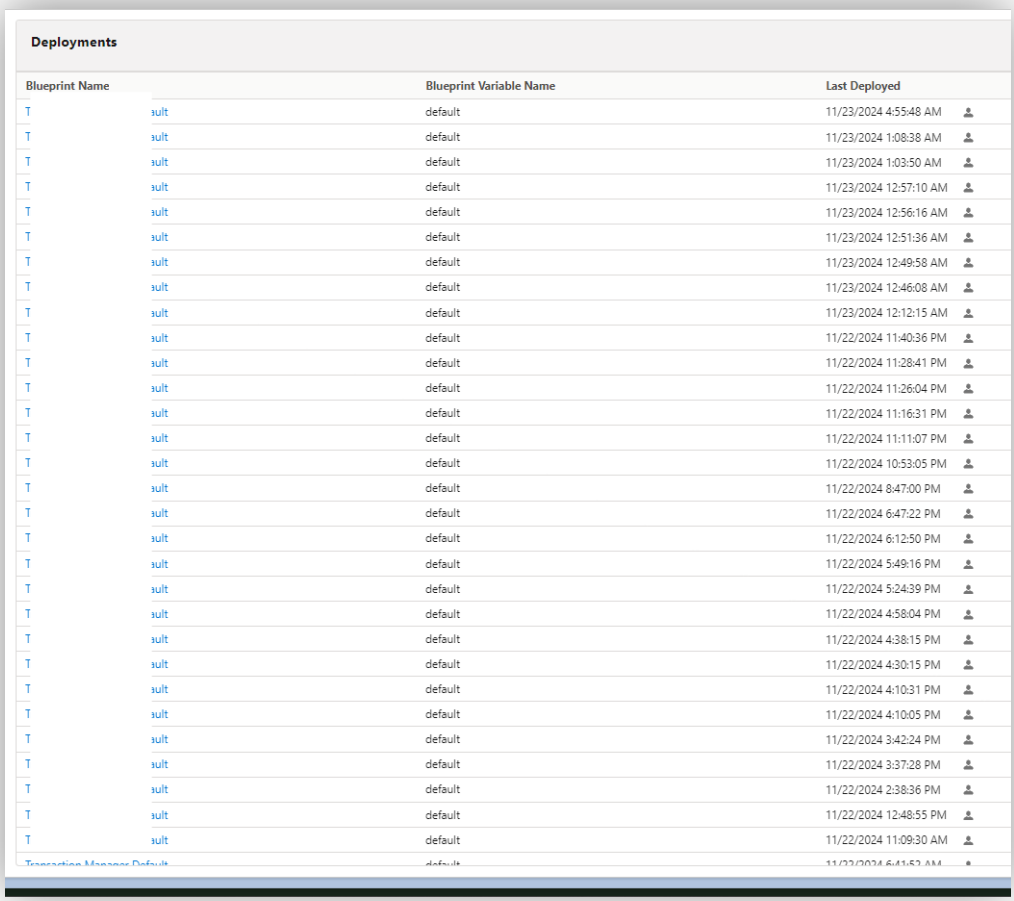

Fig 10.10 : Logik IO Deployments UI [5].

Chapter 11: DETERMINATION RULE (Sample Code)

The address formatting determination rule is designed to automate the generation of structured, country-specific addresses within the system. It functions by interpreting various input parameters such as country, address type (e.g., billing, shipping, legal), and customer classification. These parameters guide the system in identifying how an address should be formatted to comply with local standards.

Once the inputs are received, the rule initiates queries across multiple internal and external databases to gather relevant address components. A key output of this step is the retrieval of a sequence code, which defines the correct order in which the address elements (such as street, city, postal code, and country) should appear. This code varies based on the country's address structure and ensures regional compliance.

The sequence code is then mapped with the data fetched from other databases, and the rule proceeds to construct the full address. This includes arranging components in the right order and applying correct formatting elements like line breaks or punctuation.

The final, formatted address is stored in the system and can be used across various business functions, including invoicing, shipping, and legal documentation. This rule ensures address accuracy, reduces manual intervention, and provides consistent formatting across systems, making it essential for scalable and error-free global operations.

Additionally, the rule improves overall data hygiene, simplifies compliance with regional postal regulations, and enhances customer communication by ensuring addresses are always complete and properly structured. Its adaptability to different address formats makes it a vital part of any enterprise system handling global customer data, significantly contributing to operational efficiency and reliability.

```
var seqNum = "";
let cityName = "";
let stateName = "";
let postalCode = "";
let clientName = "";
let countyName = "";
let fullAddress = "";
let countryName = "";
let provinceName = "";
let addressLine1 = "";
let addressLine2 = "";
let addressLine3 = "";
let addressLine4 = "";
let seqIdentifier = "";
let addressLanguage = "";
let countryDescription = "";

let params1 = {
    "unit": txn.custom.unit //taking field variable value
};

addressSet = lookup("SELECT ADDRESS_ID, LINE1, LINE2, LINE3, LINE4, CITY, STATE, ZIP, COUNTRY, COUNTY
from ADDRESS_DETAILS where UNIT = :unit", params1);
for (var addr of addressSet) {
    cityName = addr.get("CITY");
    stateName = addr.get("STATE");
    postalCode = addr.get("ZIP");
    countryName = addr.get("COUNTRY");
    countyName = addr.get("COUNTY");
    addressLine1 = addr.get("LINE1");
    addressLine2 = addr.get("LINE2");
    addressLine3 = addr.get("LINE3");
    addressLine4 = addr.get("LINE4");
}
```

```
let params2 = {
    "country": countryName
};

countrySet = lookup("SELECT COUNTRY_DESC from COUNTRY_MAPPING where COUNTRY_CODE = :country", params2);
for (var countryRec of countrySet) {
    countryDescription = countryRec.get("COUNTRY_DESC");
}

if (countryName && countryName.trim() !== "") {
    let params = {
        "countryCode": countryName
    };

    let countryInfoSet = lookup("SELECT NAME, SEQ FROM COUNTRY_INFO WHERE CODE = :countryCode",
params);

    for (var infoItem of countryInfoSet) {
        seqNum = infoItem.get("SEQ");
        countryFullName = infoItem.get("NAME");
    }
}

const addressMapping = {
    "CL": clientName,
    "L1": addressLine1,
    "L2": addressLine2,
    "L3": addressLine3,
    "L4": addressLine4,
    "CT": cityName,
    "CY": countyName,
    "ST": stateName,
    "PR": provinceName,
    "CD": countryDescription,
    "PC": postalCode
};
```

Fig 11.1 : Dummy Determination Rule Code part -1

```
let seqParts = seqNum.split(":");
let outputArray = [];
if (seqParts.length > 1) {
    seqParts.forEach(part => {
        if (part.includes('-')) {
            const subParts = part.split('-');
            let subOutput = [];
            subParts.forEach(subItem => {
                if (addressMapping[subItem] && addressMapping[subItem].trim() !== "")
{                   subOutput.push(addressMapping[subItem]);
                }
            });
            if (subOutput.length > 0) {
                outputArray.push(subOutput.join(" "));
            }
        } else if (addressMapping[part] && addressMapping[part].trim() !== "") {
            outputArray.push(addressMapping[part]); // Each colon-separated part
        }
    });
}

let finalOutput = [];
for (var i = 0; i < outputArray.length; i++) {
    if (outputArray[i] && outputArray[i].trim() !== "") {
        finalOutput.push(outputArray[i]);
    }
}
// Final formatted result
let formattedOutput = finalOutput.join("\n");
return formattedOutput;
```

Fig 11.2 : Dummy Determination Rule Code part-2.

Chapter 12: BLUEPRINTS

The **Blueprint** serves as the foundational framework that defines the configuration experience within a CPQ system like Logik IO. It acts as a centralized container that brings together all the essential elements fields, rules, layouts, and configurable products necessary to deliver a coherent and efficient user interface for product configuration. Essentially, it is the architectural layer that connects user interaction with the underlying business logic and data models.

At its core, the Blueprint allows administrators to associate a selective subset of **fields and rules** that are specifically relevant to the configuration journey it represents. This ensures that only pertinent components are exposed to the end user, thereby enhancing clarity and reducing unnecessary complexity. For example, a Blueprint configured for a specific product line might include unique pricing rules, product attributes, and selection constraints tailored exclusively for that offering.

The Blueprint also governs the **layout** of the configuration interface, determining how and where fields are displayed, in what sequence, and how navigation flows across the different stages of the configuration process. This results in a more intuitive and seamless user experience, especially in scenarios involving complex or highly customizable product offerings. Additionally, by defining **configurable products** within the Blueprint, the system ensures that each configuration session is both dynamic and contextual, responding in real time to the inputs and selections made by the user.

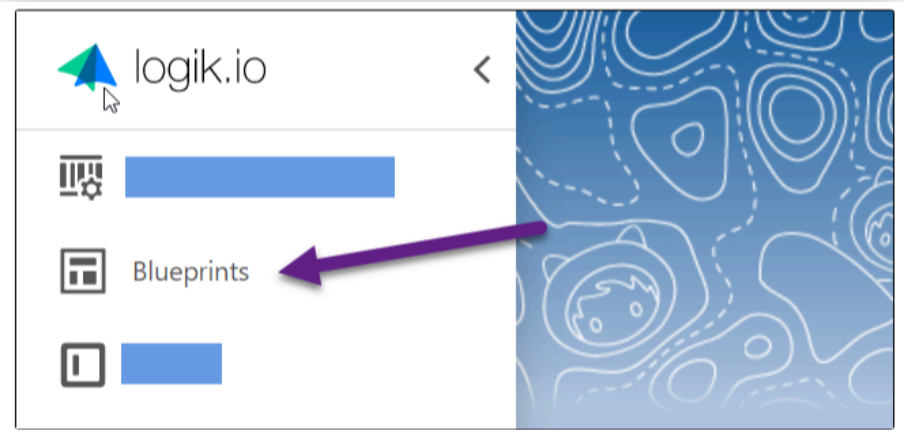

Fig 12 : Logik IO Developement UI [6].

Blueprint Components and Their Roles :

A Blueprint in Logik.io is the foundational framework that drives the configuration experience. It defines how configurable products behave, how options are presented to users, and how rules and scripts shape the configuration process. Each Blueprint acts as a self-contained unit that brings together fields, rules, layouts, and products into a coherent, interactive interface that adapts dynamically to user inputs. Below is a breakdown of the key components within a Blueprint and their significance:

I. **Associated Fields**

Associated fields are the input elements within a configuration, tied directly to product features or customer preferences. These fields are how users express their configuration choices such as selecting a size, choosing a color, or entering a quantity. For example, in the configuration of a customizable T-shirt, the “Color” field might present a picklist of options like Red, Blue, or Black. These fields are tightly integrated with rules and scripts that interpret the selections and influence the configuration's outcome, ensuring relevance and logic throughout the experience.

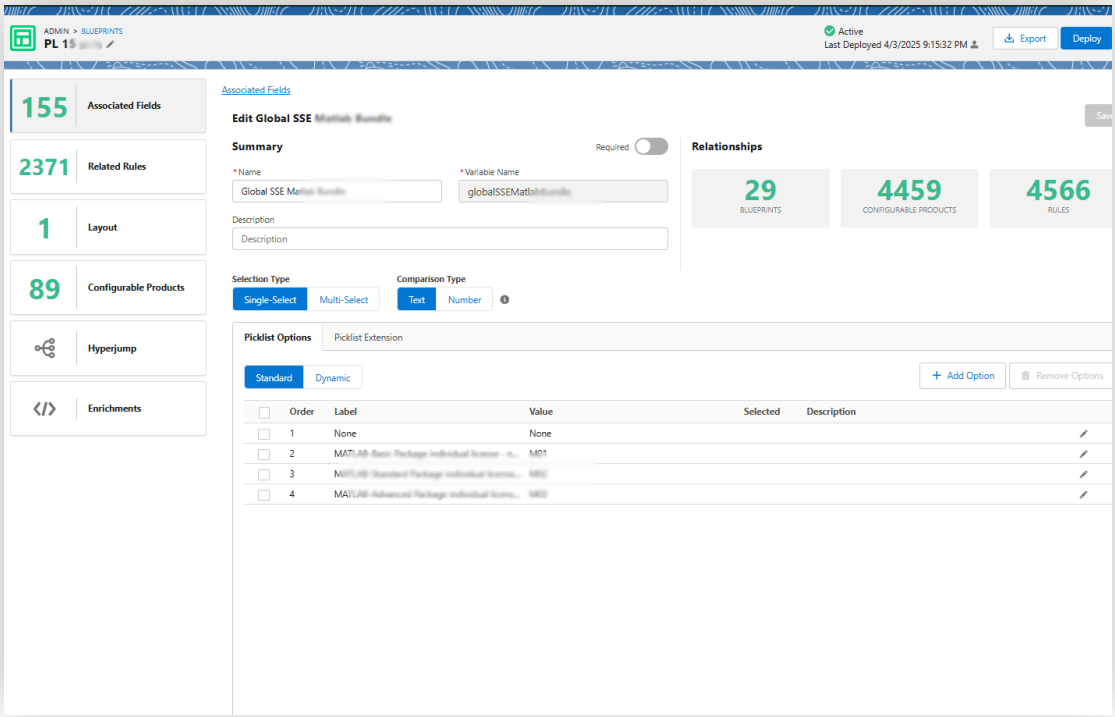

Fig 12.1 : Logik IO developement UI showing an associated field of a Blueprint. [5]

II. **Related Rules**

Rules serve as the logic layer of the Blueprint, governing how the system reacts to user input. There are various types of rules, including dependency rules that control field visibility based on conditions (e.g., "if 'Premium Material' is selected, show 'Color Finish'") and validation rules that enforce correctness by disallowing invalid configurations. These rules ensure that the configuration not only meets business constraints but also helps prevent user errors by guiding them through only the valid paths.

III. **Layout**

The layout determines the visual structure and flow of the configuration UI. It organizes fields into intuitive sections or tabs like "Basic Info," "Features," "Add-ons," or "Pricing." A well-designed layout enhances usability by grouping related options, reducing clutter, and making the process smoother for the end-user. Layouts are essential in large configurations where too many options might otherwise overwhelm the user.

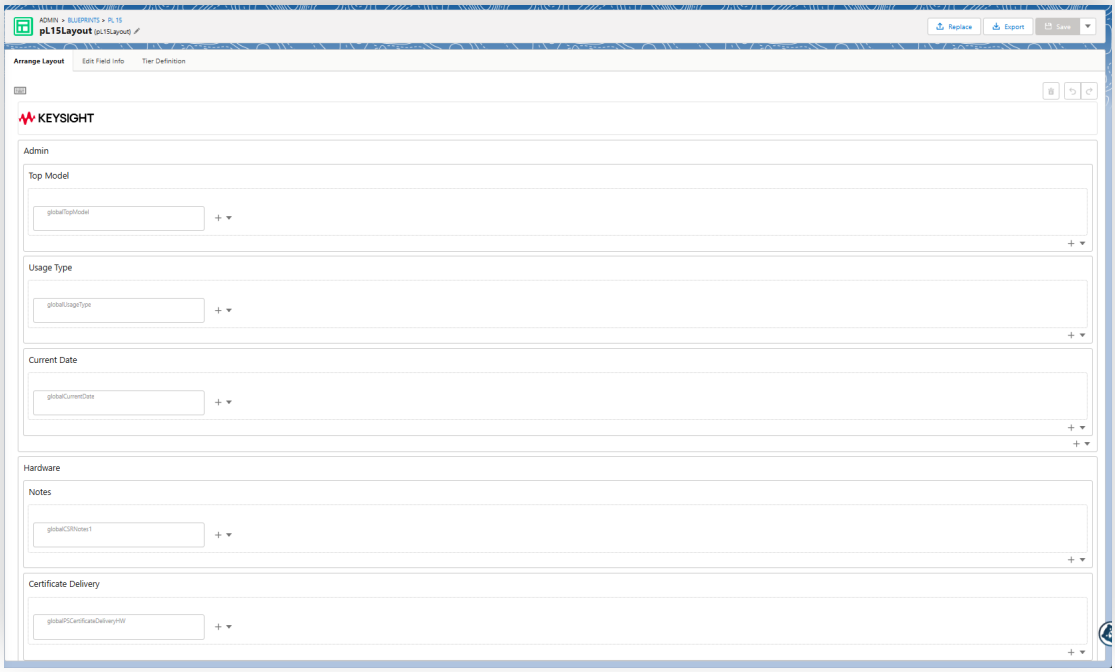

Fig 12.3 : Logik IO developement UI showing layout of a Blueprint [5].

IV. Configurable Products

This tab within the Blueprint lists all the configurable products associated with it. These products, when selected by a user either through the Logik.io platform or via external systems like Salesforce trigger the launch of the Blueprint. Each product inherits the defined logic, fields, rules, and layout from its Blueprint, ensuring consistent behavior across systems. This also promotes reusability, as the same logic can power multiple products.

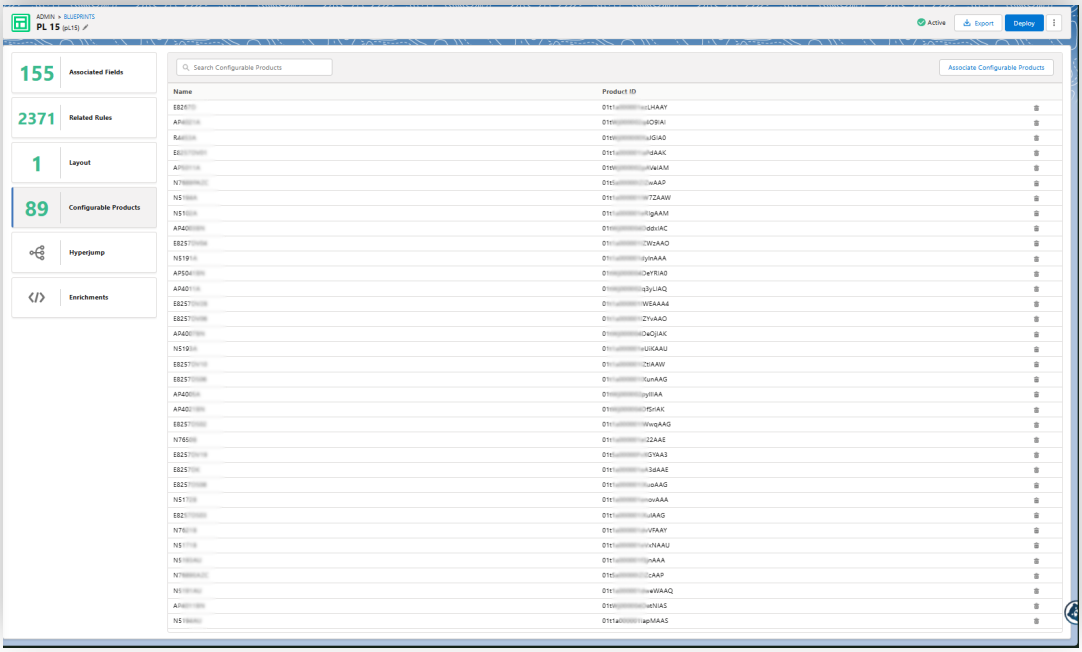

Fig 12.4 : Logik IO developement UI showing a blueprint with all the associated products [5]

V. Hyperjump

Hyperjump provides a graphical, node-based view of the Blueprint's logic. It visually maps the relationships between fields, rules, picklist options, and associated conditions. This tool is especially helpful for administrators and developers who want to understand or debug the configuration logic. Through Hyperjump, one can explore how field selections influence each other, identify rule conflicts, and ensure the integrity of the configuration flow.

VI. Enrichment Scripts

Enrichment scripts introduce advanced logic, automation, and customization into the configuration experience. These scripts execute at specific stages in the

product configuration lifecycle to add dynamic behavior. There are several types of enrichment scripts:

- **On Config/Reconfig Script**: This script runs when the configuration session starts or is reopened. It can pre-populate fields, assign default values, or apply rules based on context (e.g., selecting a warranty option automatically when a product category is chosen).
- **On BOM (Bill of Materials) Response Script**: Triggered after a configuration is finalized, this script modifies or adds to the final Bill of Materials. For example, selecting a large appliance might automatically add an installation kit.
- **Picklist Extension Pricing Script**: Dynamically updates picklist options and adjusts pricing based on selections.
- **Validation Script**: Runs before the user completes the configuration. It checks for rule compliance and ensures all required selections are made. If conditions are not met, the "Done Configure" option is disabled, ensuring completeness.

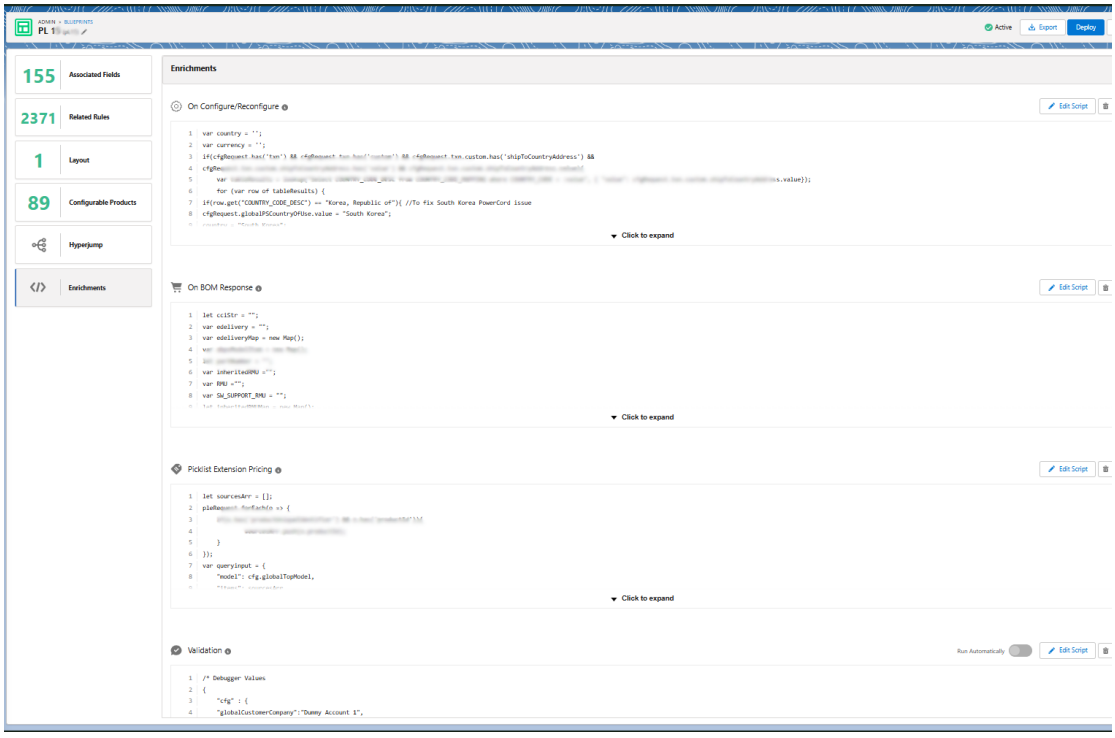

Fig 12.6 : Logik IO developement UI showing Enrichment of a blueprint.[5]

Chapter 13: BLUEPRINT MANAGEMENT AUTOMATION

A suite of powerful automation scripts designed to streamline and standardize blueprint management across multiple product configuration instances was developed by me. These scripts addressed critical pain points in product consistency, rule validation, data integrity, and migration, resulting in significant time savings, improved accuracy, and enhanced scalability. Key functionalities included:

```python
BLUEPRINT_IDS = [
    "Blueprint1",
    "Blueprint2",
    "Blueprint3",
    "Blueprint4",
    "Blueprint5"
]

API_URL_1 = "https://dummy-api1.example.com/v1/blueprints"
API_URL_2 = "https://dummy-api2.example.com/v1/blueprints"


headers_1 = {
    "Authorization": "Bearer dummy_token_1",
    "Accept": "application/json",
    "Content-Type": "application/json"
}

headers_2 = {
    "Authorization": "Bearer dummy_token_2",
    "Accept": "application/json",
    "Content-Type": "application/json"
}


faulty_blueprints = []


for blueprint_id in BLUEPRINT_IDS:
    # Fetch data from the first API
    response_1 = requests.get(f"{API_URL_1}/{blueprint_id}", headers=headers_1, verify=False)
    if response_1.status_code == 200:
        data_1 = response_1.json()
        fields_1 = set(data_1.get("fields", []))
        rules_1 = set(data_1.get("rules", []))
        sets_1 = set(data_1.get("sets", []))
    else:
        print(f"Failed to retrieve data for {blueprint_id} from API 1")
        continue


    response_2 = requests.get(f"{API_URL_2}/{blueprint_id}", headers=headers_2, verify=False)
    if response_2.status_code == 200:
        data_2 = response_2.json()
        fields_2 = set(data_2.get("fields", []))
        rules_2 = set(data_2.get("rules", []))
        sets_2 = set(data_2.get("sets", []))
    else:
        print(f"Failed to retrieve data for {blueprint_id} from API 2")
        continue

    print(f"Comparing {blueprint_id}:")
    print(f"API 1 Rules Count: {len(rules_1)}")
    print(f"API 2 Rules Count: {len(rules_2)}")

    fields_missing_in_api1 = fields_2 - fields_1
    rules_missing_in_api1 = rules_2 - rules_1
    sets_missing_in_api1 = sets_2 - sets_1

    print("Fields missing in API 1:", fields_missing_in_api1)
    print("Rules missing in API 1:", rules_missing_in_api1)
    print("Sets missing in API 1:", sets_missing_in_api1)


    if (fields_missing_in_api1 or rules_missing_in_api1 or sets_missing_in_api1):
        print("Discrepancies found for:", blueprint_id)
        faulty_blueprints.append(blueprint_id)
```

Fig 13 : Sample checking script.

I. Product Comparison Across Instances

- Enabled comparison of thousands of products across multiple instances.
- Automatically detected missing or unmatched products, ensuring all blueprint configurations remained consistent and complete.

II. Rule, Field & Field Set Validation

- Parsed and analyzed all rules, fields, and field sets within each blueprint.
- Automatically associated valid fields to their corresponding rules and sets where applicable, reducing manual rework and preventing errors.

III. Table Management Automation

- Standardized table structures and cleaned up data inconsistencies across.
- Automated formatting, schema alignment, and data population to eliminate manual intervention.
- Significantly reduced errors in table configurations and accelerated blueprint readiness.

IV. Rule Grouping & Execution Control

- Introduced logic for intelligent rule grouping, organizing thousands of rules into logical and manageable clusters.
- Enabled precise control over rule execution stages within the configuration process.

V. Seamless Blueprint Migration Between Instances

- Designed and implemented scripts to handle complete blueprint migrations products, fields, rules, layouts, and metadata from one instance to another.
- Ensured high-fidelity transfers without data loss or structural mismatches.
- Greatly reduced deployment time and risks associated with manual migration.

Chapter 14: SAMPLE SCRIPT

A script that automates the complete download of all tables from a given instance, a process that would otherwise take hours if done manually through the UI was developed. By accepting a session ID and instance link, the script authenticates access, dynamically fetches the list of all available tables, and simulates the manual export process by triggering export requests and downloading each table sequentially. This eliminates the need to individually click the export and download buttons for each table. A counterpart script handles the uploading of these downloaded tables to another instance, enabling seamless and efficient table migration between environments. This automation not only saves significant time but also ensures consistency and accuracy during large-scale data transfers.

```python
import requests
import os
import time
import urllib3

urllib3.disable_warnings(urllib3.exceptions.InsecureRequestWarning)

session = "dummy-session-id"
Tcount = 0
passed = []
Inst = "dummy-instance.test"
headers = {
    "Cookie": f"SESSION={session}",
    "Accept": "application/json",
    "Content-Type": "application/json"
}

headers2 = {
    "Cookie": f"SESSION={session}",
    'Accept': 'application/octet-stream',
    "Content-Type": "application/json"
}

downloads_folder = os.path.join(os.path.expanduser("~"), "Downloads")
backup = os.path.join(downloads_folder, "BackupDummy")
tables_folder = os.path.join(backup, "tables")

os.makedirs(tables_folder, exist_ok=True)
os.makedirs(backup, exist_ok=True)

response = requests.get(f'https://{Inst}/api/managed_tables/v1/tables?size=500&sort=dateModified,DESC', headers=headers, verify=False)
if response.status_code == 200:
    current_table = response.json()
    Data = current_table["content"]
    table_names = [table["name"] for table in Data]
else:
    print(f"Failed to fetch tables. Status code: {response.status_code}")
```

Fig 14.1 : Sample table download script part -1

```python

for Tname in table_names:
    r = requests.post(f'https://{Inst}/api/managed_tables/v3/tables/{Tname}/export', headers=headers, verify=False)
    file_name = f"{Tname}.zip"
    file_path = os.path.join(tables_folder, file_name)

    if r.status_code == 200:
        response_json = r.json()
        ID = response_json.get('id')

        while True:
            file = requests.get(f'https://{Inst}/api/managed_tables/v2/job/{ID}', headers=headers, verify=False)
            if file.status_code == 200:
                file_json = file.json()
                if file_json.get("status") == "COMPLETED":
                    download = requests.get(f"https://{Inst}/api/managed_tables/v2/job/{ID}/file", headers=headers2, verify=False)
                    if download.status_code == 200:
                        with open(file_path, "wb") as f:
                            f.write(download.content)
                        Tcount += 1
                        passed.append(Tname)
                    break
                elif file_json.get("status") == "FAILED":
                    break
            time.sleep(5)

print("TOTAL TABLES EXPORTED - ", Tcount)
print("passed=", passed)
not_in_list = [item for item in table_names if item not in passed]
print("TOTAL FAILED = ", not_in_list)
```

Fig 14.2 : Sample table download script part -2

A cross-platform automation script that streamlines the complete download of all tables from a given instance a task that would otherwise require hours of manual effort was developed. The script accepts a session ID and instance URL, authenticates access, retrieves the list of all available tables in the environment, and programmatically replicates the UI process of exporting and downloading each table. This includes simulating export initiation, polling for export readiness, and downloading the data all without any manual interaction. To ensure ease of use across different operating systems,The scripts are packed as standalone executables using **PyInstaller** for Windows and **py2app** for macOS, allowing non-technical users to run them without needing a Python environment. This solution drastically reduced manual workload and improved reliability in managing large-scale table migrations.

Fig 14.3 : Win Executable

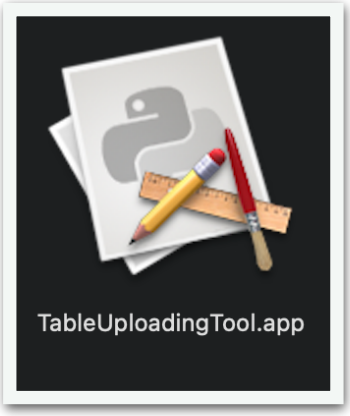

Fig 14.4 : Mac Executable

Chapter 15: TABLE MIGRATION TOOL

The table migration script to intelligently handle tables based on their size and row count, optimizing both speed and reliability was developed. The script begins by identifying all tables in the source instance and dynamically selects the migration strategy based on each table's size. For smaller tables, it downloads the export zip and directly uploads it to the target instance, ensuring a quick transfer. For larger tables, the script takes a more efficient route: it downloads the zip, extracts the CSV, splits it into smaller chunks, and then uploads each chunk sequentially. This approach prevents timeouts and failures commonly encountered with large file uploads. The entire migration process, which would typically take 2–3 hours manually, is completed in approximately 45 minutes with this script..

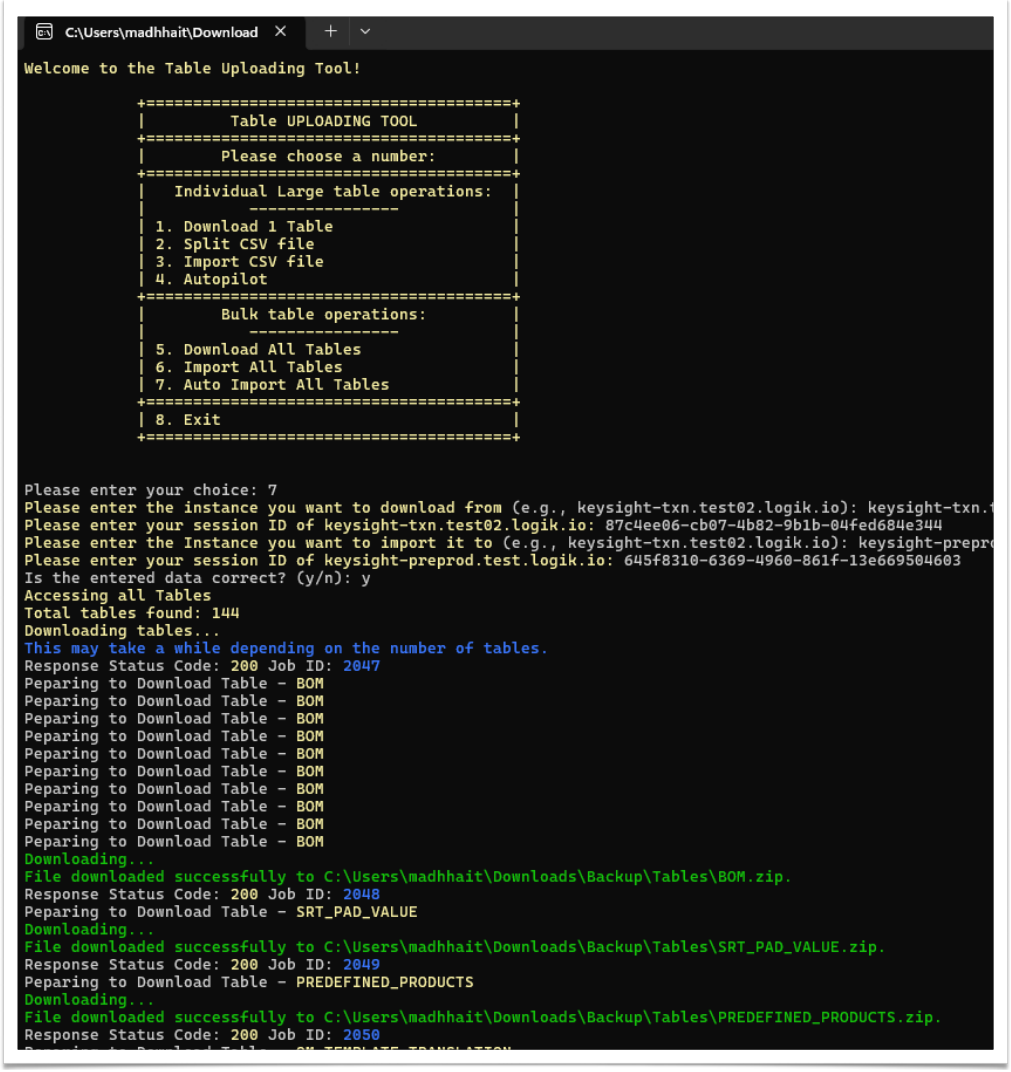

Fig 15.1 : Interface of the Table Migration Script

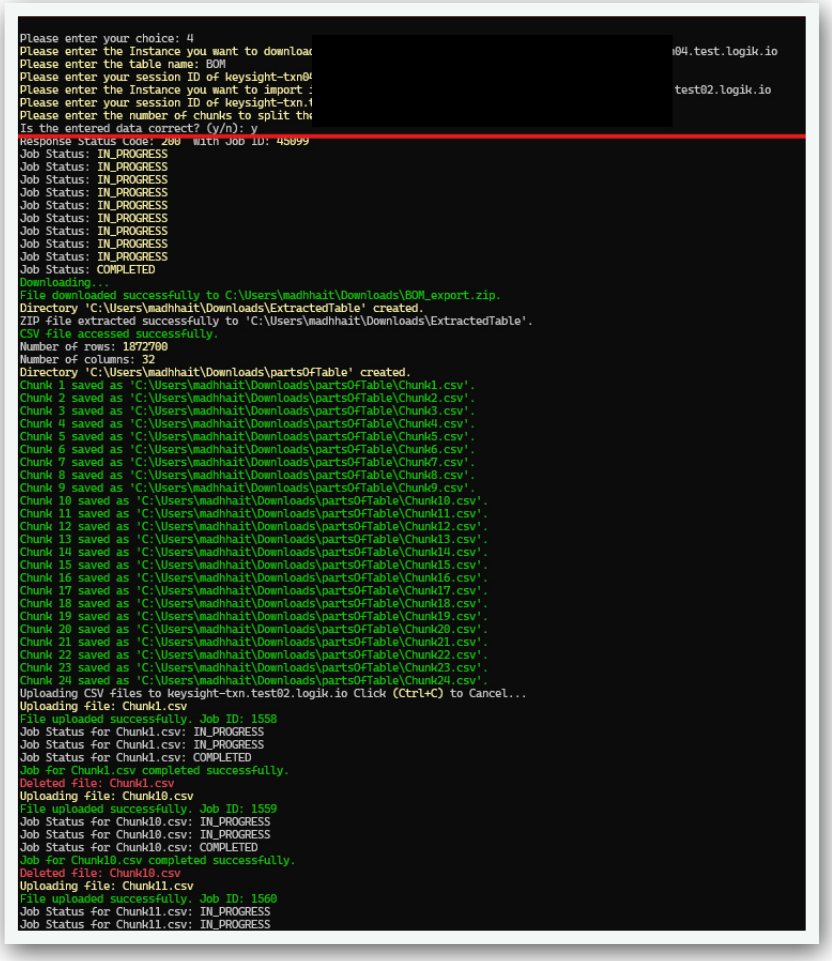

Fig 15.2 : Image showing the logs of script

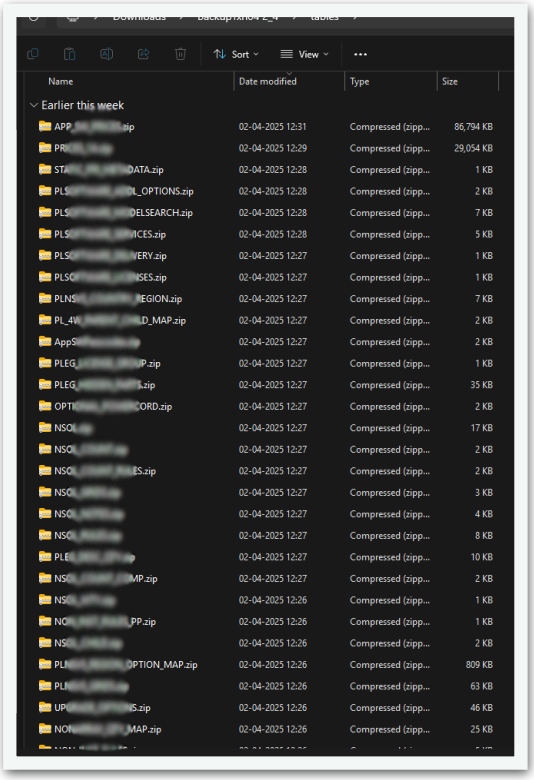

Fig 15.3 : Image showing all the downloaded tables from inst. 1

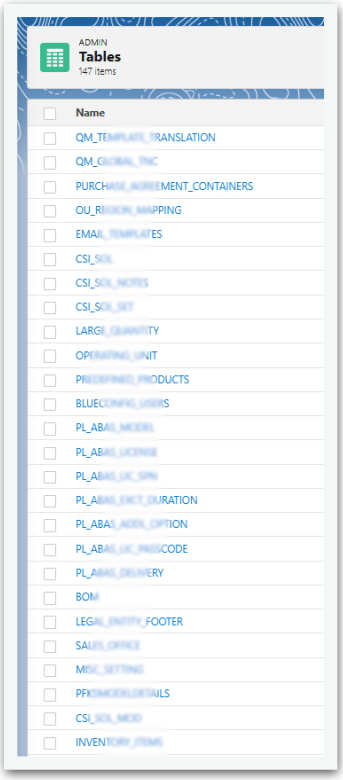

Fig 15.4: Image showing the tables migrated to logik IO inst 2. [5]

Chapter 16: BLUEPRINT DEPLOYMENT TOOL

Blueprint Deployment Tool that significantly accelerates and simplifies the deployment of multiple blueprints across instances was developed. Traditionally, deploying all blueprints manually would take around 3.5 hours due to the need to handle each blueprint one by one. This tool reduced the total deployment time to just 2 hours by automating the entire process while accounting for system constraints. It smartly manages load by deploying only three blueprints in parallel at any given time, as the system cannot reliably handle more. Each blueprint typically takes between 4 to 15 minutes to deploy, depending on its complexity. The tool ensures smooth, reliable execution without overwhelming the system. To make it accessible and easy to use across different platforms, the tool was packaged into standalone executables using **PyInstaller** for Windows and **py2app** for macOS. This automation not only saved time but also brought consistency and reduced the risk of human error during large-scale deployments.

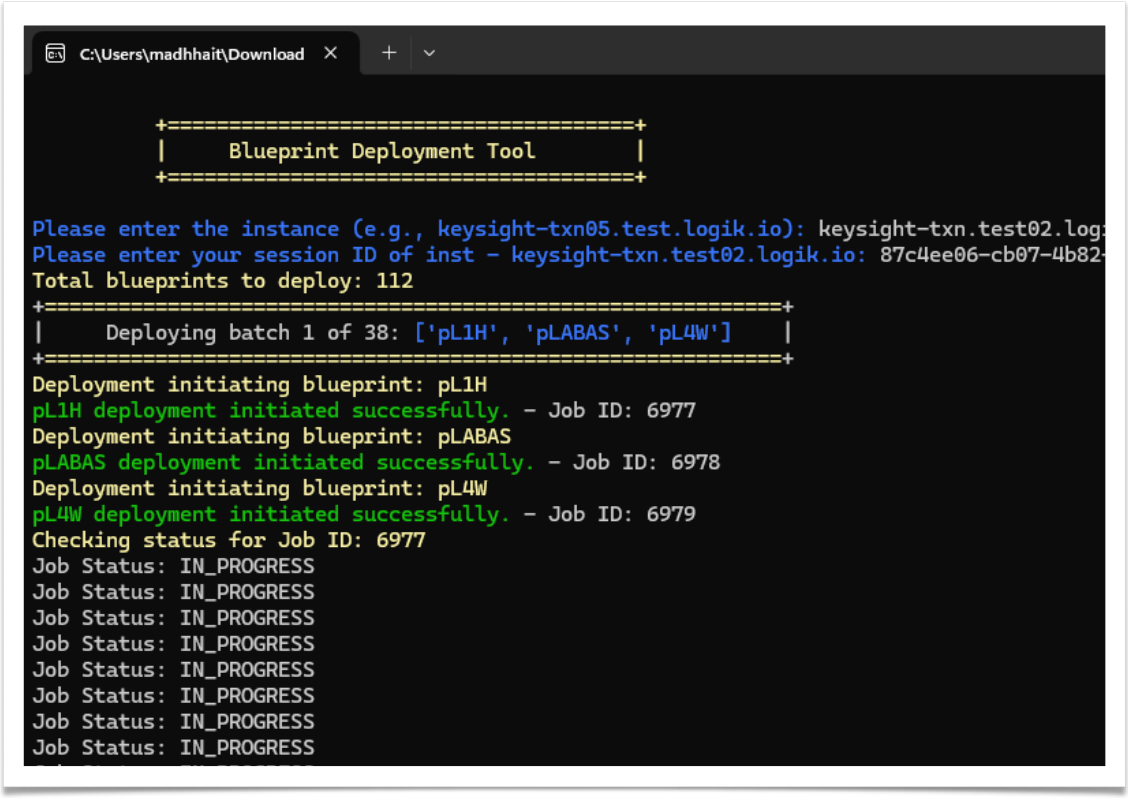

Fig 16 : Interface of the BP deployment Script

Chapter 17: CHALLENGES FACED

While the transition from Oracle CPQ to Logik IO offers a strategic leap forward, it has also introduced several challenges. These include system instability, migration complexity, collaboration hurdles, and development dependencies as :

I. **Evolving Platform – Logik IO Still Under Development**

The biggest challenge has been working with Logik IO while it's still under active development. With features constantly being added or modified, identifying the root cause of any issue becomes difficult, we often find ourselves unsure whether the problem lies in our implementation or within the Logik platform itself. This dependency blurs accountability, complicates debugging, and creates delays in resolution. Moreover we often need to find work around missing features, build interim solutions, or adapt to changes on the go.

II. **Lack of Automated Migration Tools**

The migration of thousands of blueprint components - including logic, fields, and configurations - from Oracle CPQ to Logik IO remains a mostly manual task. This increases the risk of errors, broken associations, and inconsistencies across environments, reducing reliability and adding overhead.

III. **Complex Rule Translation and Instance Variability**

Due to structural differences between Oracle CPQ and Logik IO, direct rule translation is rarely possible. Teams must rework complex logic while dealing with instance-specific customizations, requiring deep technical knowledge and prolonged testing cycles.

IV. **QA Team Pushback and Cross-Team Friction**

Another significant challenge was managing the expectations and criticisms from the QA team. As QA teams validate functionality against expected outcomes, gaps due to platform instability, version mismatches, or unclear ownership often lead to friction. In many cases, QA feedback pointed to failures that stemmed not from our implementation but from unresolved platform-level issues creating tension.

Chapter 18: CONCLUSION

My internship at **Keysight Technologies** has been a transformative journey, immersing me in the **Configure, Price, Quote (CPQ)** domain and giving me hands-on exposure to cutting-edge technologies. I had the opportunity to work with both **Oracle CPQ** and its next-generation replacement, **Logik IO**, playing a key role in the company's migration efforts while contributing significantly through the development of powerful automation scripts. This experience not only deepened my technical understanding of CPQ systems but also empowered me to actively drive innovation and efficiency within a critical business process.

Key Contributions and Learnings:

I. Advanced Scripting for Blueprint Management & Automation

A comprehensive suite of automation scripts developed by me that transformed the way blueprint configurations are managed. These scripts enabled:

- **Automated blueprint comparisons** across multiple instances to identify missing products and configuration mismatches.
- **Smart rule grouping logic** to control rule execution flow with precision.
- **End-to-end blueprint migration tools**, reducing deployment time from 3.5 hours to 2 hours, by deploying only 3 BPs concurrently adhering to system performance limits.

  These tools were packaged using **PyInstaller** for Windows and **py2app** for macOS, making them user-friendly and executable across platforms.

II. **Intelligent Table Migration Script Based on Size**

Built a table migration tool that optimized the migration strategy :

- **Smaller tables** were directly downloaded as zips and uploaded to the new instance.
- **Larger tables** were split into smaller CSV chunks after download to ensure smooth uploads without timeouts. This script cut down manual migration time from 2–3 hours to approximately **45 minutes**, drastically reducing effort.

III. **End-to-End Blueprint Deployment Tool**

To automate the blueprint deployment process a deployment tool was developed by me that:

- Deploys a maximum of three blueprints simultaneously, respecting system constraints.
- Handles blueprints that take between **4–15 minutes** each. & Reduced total deployment time .

IV. **Driving the Migration to Logik IO – The Future of CPQ**

A core part of my internship was contributing to the strategic migration from **Oracle CPQ** to **Logik IO**, a shift that promises:

- **Faster and more flexible configurations**
- **Scalable architecture** suitable for complex product models
- A **user-friendly and efficient** quoting experience
- **CPQ system workflows** and integration with CRMs like Salesforce

V. Preparing for Scalable Deployment via Lambda Functions

To increase accessibility and reduce dependency on local environments, I'm currently working on deploying these scripts as AWS Lambda functions. This would enable them to be triggered as jobs within Salesforce, allowing anyone in the organization to execute.

VI. Measurable Business Impact

Once the migration to Logik IO goes live in August, Keysight is projected to save thousands of dollars monthly in Oracle SaaS license fees. The automation tools and processes developed by me are already making significant contributions to this shift by improving speed, accuracy, and reliability.

Chapter 19: FUTURE SCOPE

The strategic transition from Oracle CPQ to **Logik IO** marks a new era for Configure, Price, Quote (CPQ) at **Keysight Technologies**. As the company modernizes its sales infrastructure, Logik IO is set to become the backbone of an intelligent, scalable, and cost-efficient CPQ ecosystem. The platform offers a forward-looking foundation to support not only current operational efficiency but also future innovation through automation, analytics, and AI integration.

I. Modernization of Sales Infrastructure

Logik IO's advanced configurability and dynamic rule engine allow Keysight to streamline complex product configurations and accelerate quote generation. This transition eliminates legacy limitations and positions the CPQ system as a powerful enabler of responsive, customer-centric selling. It has integration with various platforms like Salesforce & SNow which facilitate the data flow.

II. AI & Machine Learning Integration

- The future of CPQ lies in intelligent automation. With its modern architecture, Logik IO can be further extended to integrate **AI and ML** for:
- Smart product recommendations based on customer history and patterns
- Real-time quote optimization using market-driven data
- Automated anomaly detection in pricing and configurations

These enhancements will empower sales teams with data-driven decision-making, reduce errors, and improve customer satisfaction.

III. Scalability and Global Expansion

Logik IO supports global scalability, critical for a multinational like Keysight. The platform can handle expanding datasets, varied product catalogs, and diverse regional pricing and configuration rules. This adaptability makes it ideal for global rollouts, regional customization, and rapid onboarding of new markets or verticals.

IV. Improved Customization and Flexibility

As business units evolve, so do their requirements. Logik IO's flexible framework allows custom workflows, product logic, and rule sets tailored to specific teams or industries. This ensures that Keysight can **innovate rapidly**, respond to unique customer demands, and maintain a competitive edge.

V. Cost Optimization and License Savings

Upon full rollout (expected by **August**), Keysight will save **thousands of dollars monthly** by decommissioning Oracle CPQ licenses. These savings can be reinvested into further innovations, such as AI-enhanced features, deeper automation, and extended training/support infrastructure.

VI. Foundation for Autonomous CPQ Operations

By integrating tools like **AWS Lambda functions**, the platform can trigger tasks like blueprint validation or data migration directly from Salesforce, without manual intervention. This paves the way for autonomous CPQ operations, self-healing processes, and auto-deploy pipelines powered by business logic and real-time insights.

VII. **Continuous Learning and System Optimization:**

The future scope of CPQ systems at Keysight Technologies also includes a continuous cycle of learning and optimization. As the system collects more data over time, machine learning algorithms can be applied to improve pricing strategies, product configurations, and sales forecasts. Continuous system updates and learning from data will ensure that the CPQ solution remains adaptive to new business challenges and evolving market demands.

In conclusion, Keysight has the opportunity to remain at the forefront of innovation. This will not only enhance the efficiency of the sales process but also position the company to meet the growing demands of global markets, drive greater customer satisfaction, and continue to lead in the competitive landscape.

Chapter 20: LIMITATIONS

I. **Platform Still Maturing**

One of the most significant limitations is that Logik IO is not yet fully developed or feature-complete. As an evolving platform, it lacks full maturity in some areas, and the product roadmap is still unfolding.

II. **Learning Curve for Legacy Users**

Switching from Oracle CPQ to Logik IO involves a learning curve, slowing productivity until users adjust to the new system.

III. **Feature Gaps vs. Oracle CPQ**

Not all Oracle CPQ features are available in Logik IO. Legacy customizations may need to be rebuilt, extending migration timelines.

IV. **Complex Migration and Integration**

Migrating rules and configurations isn't straightforward, often needing custom scripting and extensive testing.

V. **Deployment Limitations**

Only three blueprints can be deployed at once to avoid system crashes, with each taking 4 - 15 minutes slowing down releases.

VI. **AI/ML Requires Custom Build**

It does not include native AI features out of the box. Predictive pricing, smart product recommendations, or auto-configurations must be built & integrated separately using external data models.

VII. **Heavy Salesforce Dependence**

Tight integration with Salesforce means any issues there impact CPQ workflows. Adapting for other CRMs may be challenging.

VIII. **Limited Offline/Mobile Support**

Logik IO's always-online, web-based model may not suit low-connectivity or mobile-first environments without extra development.

Chapter 21: GANTT CHART

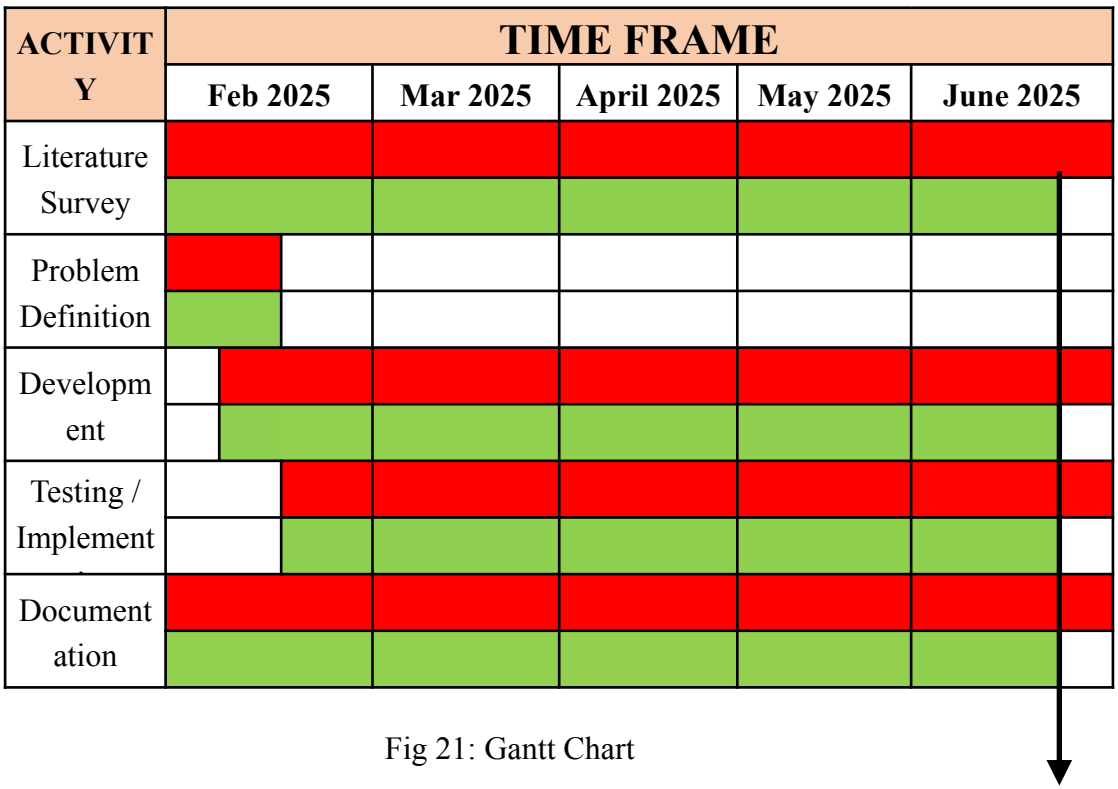

Fig 21: Gantt Chart

Chapter 22: REFERENCES

[1] (2024). Machine Learning Applications in Salesforce CPQ Transforming Sales. Journal of Artificial Intelligence, Machine Learning and Neural Network. 4. 27-38. 10.55529/jaimlnn.46.27.38. [Last accessed : April 20, 2025]

[2] Jordan, Michelle & Auth, Gunnar & Jokisch, Oliver & Kühl, Jens-Uwe. (2020). Knowledge-based systems for the Configure Price Quote (CPQ) process - A case study in the IT solution business. Online Journal of Applied Knowledge Management. 8. 17-30. 10.36965/OJAKM.2020.8(2)17-30. [Last accessed : April 20, 2025]

[3] https://companyname.bigmachines.com/ [Last accessed : June 23, 2025]

[4] https://company-txn.test.logik.io/transaction [Last accessed : July 5, 2025]

[5] https://developer.salesforce.com/blogs/2019/05/automating-salesforce-cpq-testing [Last accessed : March 3, 2025]

[6] https://www.logik.io/. [Last accessed : April 23, 2025]

[7] https://www.forsysinc.com/blog/do-you-need-a-cpq/. [Last accessed : April 29, 2025]

[8] https://www.salesforce.com/in/resources/ [Last accessed : April 29, 2025]

[9] Hvam, Henrik & Pape, Søren & Nielsen, Morten K. (2006). Improving the quotation process with product configuration. Computers in Industry. 57. 607–621. 10.1016/j.compind.2006.01.007. [Last accessed : April 23, 2025]

[10] Näsi, Aleksi. (2020). Enhancing sales & product management with CPQ systems. *Journal of Business and Technology*. 5(2). 45–52. [Last accessed : June 15, 2025]