

PROJECT REPORT - PS 1

BY

NAME OF STUDENT	ID
Vaibhav Panda	2023B2A30943H
Setu Minocha	2023AAPS0617G

SUBMITTED IN COMPLETE FULFILLMENT OF THE REQUIREMENTS OF

BITS F221: Practice School I

At

EBO Mart

A Practice School I station of



BITS Pilani
Pilani | Dubai | Goa | Hyderabad

IBO

KPN
fresh

BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE (RAJASTHAN)

July, 2025

BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE PILANI (RAJASTHAN)

Practice School Division

Station: EboMart

Centre: Bangalore

Duration: 54 days

Date of Start: 26th May 2025

Date of Submission: 13th July 2025

Title of the Project: MCP-Enabled AI Agent for Dynamic Data Retrieval and Workflow Automation using BigQuery and n8n.

ID:	2023B2A30943H	2023AAPS0617G
Name:	Vaibhav Panda	Setu Minocha
Discipline of Student(s):	Msc. Chemistry + B.E. Electrical and Electronics	B.E. Electronics and Communication

Name(s) and designation(s) of the expert(s): Mr. Abhijeet Deshmukh, CTO of EBO Mart; Mrs. Mithila Uchil, Operations management; Mr. Madhu Yerubandi, Head of Data Analytics Team.


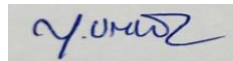
Name(s) of the PS Faculty: Yadawananda Neog

Key Words: Flask, REST API, Google BigQuery, JSON responses, Google Cloud IAM, Postman, Python unittest, n8n, local LLMs, real-time access, product recommendation, secure authentication, secure authorization, data pipeline, scalability, workflow automation, pagination, SQL query optimization, retail analytics, Model Context Protocol (MCP), tool-enabled LLM interaction, FastMCP, agent-oriented workflows, streaming APIs, Ollama integration, MCP client-

server communication, AI agent tool invocation, parameterized BigQuery queries, dynamic tool resolution, structured JSON tool response, cloud-native orchestration, asynchronous API calls, Python BigQuery client, input schema validation, enterprise data access, NLP-assisted querying, business logic abstraction, prompt-to-tool execution, and AI-augmented decision support.

Project Areas: LLM Integration, Agent-Based Systems, Serverless Architecture, Data Orchestration, AI-Driven Workflows, API Tooling, Contextual AI Interfaces, Microservices, Event-Driven Systems, AI Workflow Design, Cloud-Native Development, Data Query Optimization, Knowledge Retrieval Systems, Intelligent Automation, and Prompt Engineering.

Abstract: This project focuses on the development of a scalable and intelligent data interaction layer for KPN Fresh, integrating traditional REST APIs with next-generation AI agent capabilities. Using Flask and Google BigQuery, secure and optimized endpoints were created to access retail analytics data in real time. These APIs were then exposed through the Model Context Protocol (MCP), enabling seamless communication with locally hosted Large Language Models (LLMs) like Ollama. To orchestrate interactions between users, LLMs, and APIs, a dynamic workflow was implemented using n8n, allowing AI agents to query, retrieve, and reason over structured data with minimal human intervention. The system ensures secure authentication, robust query handling, and extensibility for future tools or models. This hybrid architecture bridges traditional cloud data pipelines with emerging AI-powered interfaces, enhancing business intelligence, automation, and end-user experience in retail operations.

Signature(s) of Student(s)

Signature of PS Faculty

Date: 16th July 2025

Date:

TABLE OF CONTENTS

TABLE OF CONTENTS.....	4
ACKNOWLEDGEMENTS.....	6
INTRODUCTION	7
PROBLEM STATEMENT.....	9
EXECUTIVE SUMMARY OF PROCEDURE.....	10
SYSTEM DESIGN AND IMPLEMENTATION.....	12
1. Data Access Layer – Google BigQuery.....	12
2. API Layer – Flask Microservices	13
3. Tool Abstraction Layer – FastMCP Server	15
4. Automation Layer – n8n Workflow Integration.....	16
5. Docker-Based Deployment for Workflow Services	17
6. Security and IAM Integration.....	18
7. Performance and Scalability Optimizations.....	19
8. Testing & Validation.....	19
TOOLS AND TECHNOLOGIES USED.....	20
Languages & Frameworks	20
Libraries & SDKs	20
Cloud Platform.....	21
Testing & Debugging.....	21
Automation & AI Integration.....	22
Development Environment	23

Other Integrations	23
TESTING AND EVALUATION	24
API Endpoint Testing (Flask + BigQuery)	24
Tool Invocation Testing (MCP Server)	24
Workflow Testing (n8n Integration)	25
Performance Benchmarks	26
Validation and Debugging Strategy	26
LEARNINGS AND OUTCOMES	27
Backend API Development & Data Querying	27
MCP Protocol and Tool Wrapping	27
Workflow Automation and LLM Integration	28
Cloud IAM, Authentication & Security Awareness	28
Testing, Validation & Debugging	28
Performance & Scalability Thinking	29
Broader Technical Takeaways	29
REFERENCES	30
GLOSSARY	32

ACKNOWLEDGEMENTS

We sincerely appreciate all of the advice and assistance you have given us during this project. Your help was essential to our achievement.

First and foremost, we express our gratitude to Mr. Abhijit Deshmukh, CTO of EBO Mart, for his invaluable support in integrating our efforts, outlining branch procedures, and facilitating introductions to key personnel.

We also thank Mrs. Mithila Uchil for effectively coordinating and helping us resolve any doubts that we had.

Special thanks go to Mr. Madhu Yerubandi, Head of Data Analytics Team, for his guidance and instruction on data analytics and workflow automation tools, which greatly improved our technical proficiency.

Our Faculty In-Charge, Mr Yadawanda Neog, deserves special appreciation for his unwavering encouragement and support throughout the project.

We extend our heartfelt thanks to the Director of BITS Pilani and its PS Division for providing us with the opportunity to work at EBO Mart and complete this report.

We also acknowledge the support from our Practice School faculty and mentors for their invaluable guidance and encouragement. The assistance from our peers and online developer communities - especially with debugging and deploying BigQuery APIs—was crucial, and we are grateful for their collaboration.

INTRODUCTION

In today's data-driven business landscape, companies must transform raw information into actionable insights to remain competitive. This is especially vital in the fast-paced retail industry, where real-time decisions around inventory, customer behavior, and operational efficiency can significantly impact performance.

This project, undertaken during my internship at KPN Fresh, focused on designing and developing a robust data analytics backend to support real-time, intelligent decision-making for business stakeholders. The initiative aimed to bridge cloud-based data warehousing systems with secure, lightweight REST APIs that could be consumed by automated workflows, dashboards, and even local AI agents.

Working within the Google Cloud Platform (GCP) ecosystem, the project involved developing scalable Flask APIs on top of BigQuery datasets, integrating them with workflow orchestration tools like n8n, and enabling AI-based interactions through the Model Context Protocol (MCP). The goal was to empower CXOs and business analysts with a dynamic, low-latency interface for fetching key performance data across stores and operations, all while maintaining best practices in security, scalability, and modular architecture.

This report outlines the systematic approach taken to build, test and deploy this intelligent data pipeline—from infrastructure setup to tool invocation via natural language prompts—highlighting key learnings and implementation insights gained throughout the internship.

To facilitate intelligent, real-time data access using plain language prompts, the system was designed to integrate an AI agent within an automated workflow. The AI agent communicates with an MCP (Model Context Protocol) server to dynamically fetch the available tools (e.g., APIs), decide the appropriate action, and execute the tool, all in real-time. The diagram below illustrates the end-to-end workflow:

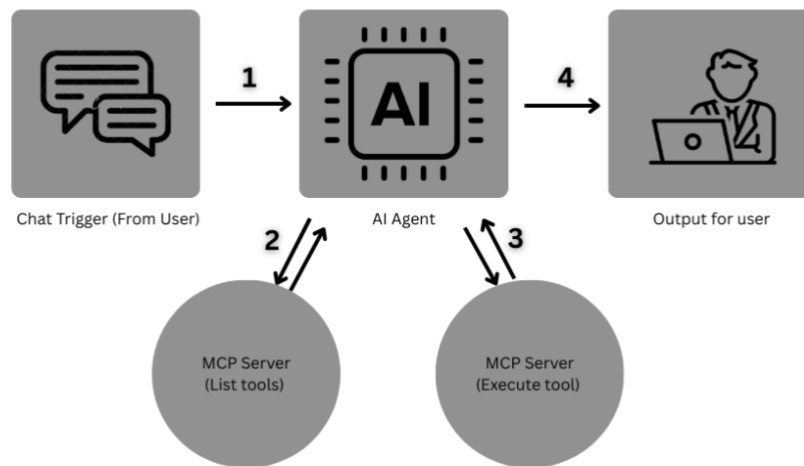


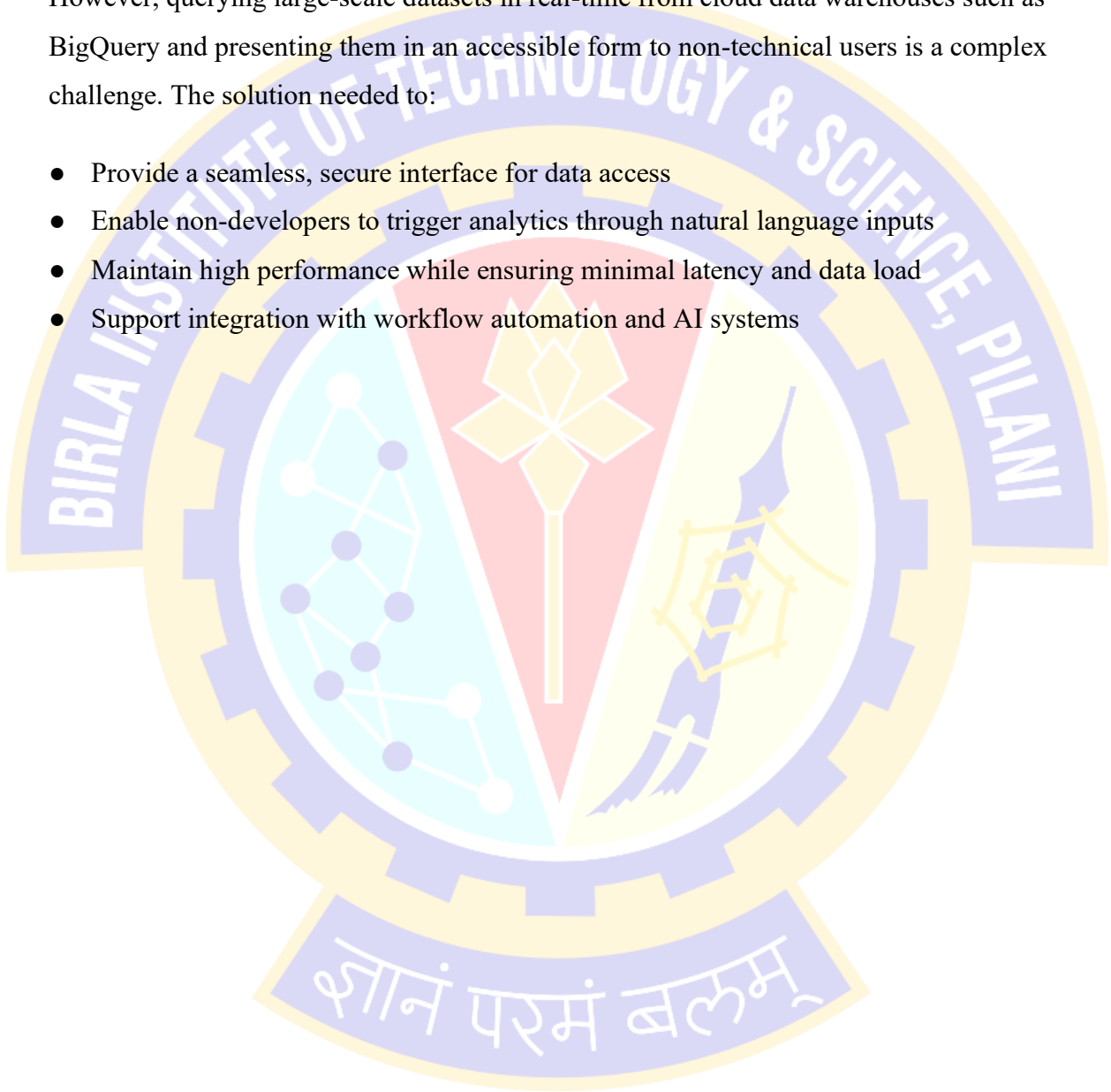
Figure: Control Flow from Chat Trigger to Final Output

1. A user sends a query via a chat interface (e.g., n8n).
2. The AI agent identifies available tools by querying the MCP server.
3. Based on user intent, the relevant tool (API) is executed.
4. The result is parsed and returned as a human-readable response to the user.

PROBLEM STATEMENT

Retail businesses require rapid access to high-quality, structured data to make informed decisions about store performance, inventory optimization, and customer engagement. However, querying large-scale datasets in real-time from cloud data warehouses such as BigQuery and presenting them in an accessible form to non-technical users is a complex challenge. The solution needed to:

- Provide a seamless, secure interface for data access
- Enable non-developers to trigger analytics through natural language inputs
- Maintain high performance while ensuring minimal latency and data load
- Support integration with workflow automation and AI systems



EXECUTIVE SUMMARY OF PROCEDURE

All the work was done in concerted phases from tech exploration to the deployment of a smart, modular data pull system powered by local LLMs and BigQuery APIs. The process was orchestrated through a controlled pipeline that combined API development, workflow automation, and AI tool integration.

1. Technology Exploration and Setup

The project started by investigating tools like Google BigQuery, Flask, n8n, and local LLMs like LLaMA. The development environment was established by using Ubuntu (using Virtual Machine) to mimic a production-like pipeline.

2. BigQuery Integration

Strong authentication controls were implemented using service account credentials, and SQL queries were tested directly in Jupyter Notebooks before they became production-quality APIs. Secure and efficient retrieval of retail data like `reordering_config` was accomplished using parameterized queries.

3. Flask REST API Development

Flask microservices were created to expose two inherent data retrieval characteristics:

- Fetching information regarding a particular `reordering_config_id`.
- Fetching all settings for a specific `site_type` and `sku_grading`

These APIs were also verified with Postman and tested for performance and JSON format.

4. MCP Server Installation using FastMCP

To make the APIs available to AI agents, the tools were registered systemically using the Model Context Protocol (MCP). A local MCP server was built using the FastMCP Python SDK, allowing the publishing of tool metadata and the provision of tool-call instructions.

5.n8n Workflow Automation

By using n8n, a low-code automation platform, the MCP server was integrated with artificial intelligence agents and chat interfaces. Workflows were created to:

Users' queries are submitted via a chat interface.

- Initiate the corresponding MCP device (API).
- Output the response in a readable format

This facilitated smooth interaction between user input, API logic, and calling of AI-enabled tools.

6. Testing, Troubleshooting, and Optimization

A few of the various issues like transport protocol mismatches, endpoint binding conflicts, and environment-specific bugs were also resolved through debugging using logs, Postman, and terminal diagnostics. The system was also optimized for improved performance, and support for additional metadata (pagination, total counts) was included in the APIs.

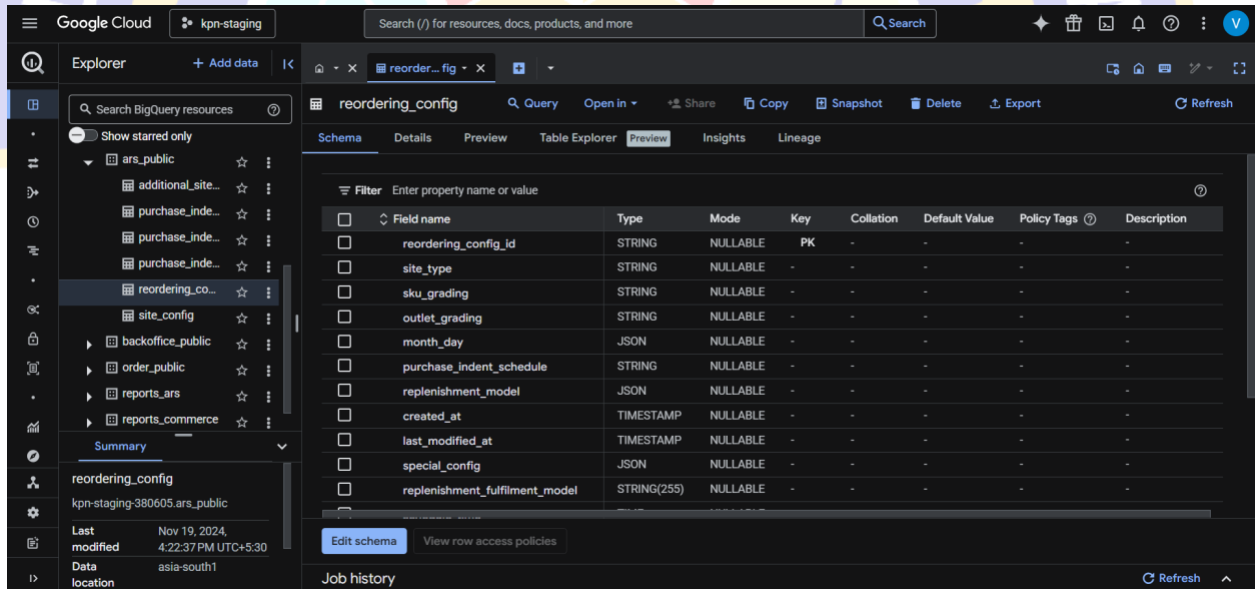
The end-to-end solution provided a scalable, testable, and advanced backend system with the capability to handle real-time analytics through simple chat-based queries, demonstrating the viability of combining cloud analytics, automation processes, and artificial interfaces.

SYSTEM DESIGN AND IMPLEMENTATION

The design of the solution was segmented into a number of key components, each with a distinct purpose within the overall structure. Together, these layers provided a convenient and secure access to retail analytics.

1. Data Access Layer – Google BigQuery

- Connected to BigQuery securely using service account credentials.
- Tables such as `ars_public.reordering_config` were used as the source of truth.
- SQL queries were written using parameterization (`@param_name`) to ensure secure and reusable execution plans.
- Query results were fetched using the BigQuery Python client library and converted into JSON-serializable formats.



Field name	Type	Mode	Key	Collation	Default Value	Policy Tags	Description
reordering_config_id	STRING	NULLABLE	PK	-	-	-	-
site_type	STRING	NULLABLE	-	-	-	-	-
sku_grading	STRING	NULLABLE	-	-	-	-	-
outlet_grading	STRING	NULLABLE	-	-	-	-	-
month_day	JSON	NULLABLE	-	-	-	-	-
purchase_indent_schedule	STRING	NULLABLE	-	-	-	-	-
replenishment_model	JSON	NULLABLE	-	-	-	-	-
created_at	TIMESTAMP	NULLABLE	-	-	-	-	-
last_modified_at	TIMESTAMP	NULLABLE	-	-	-	-	-
special_config	JSON	NULLABLE	-	-	-	-	-
replenishment_fulfillment_model	STRING(255)	NULLABLE	-	-	-	-	-

Figure: `ars_public.reordering_config` schema

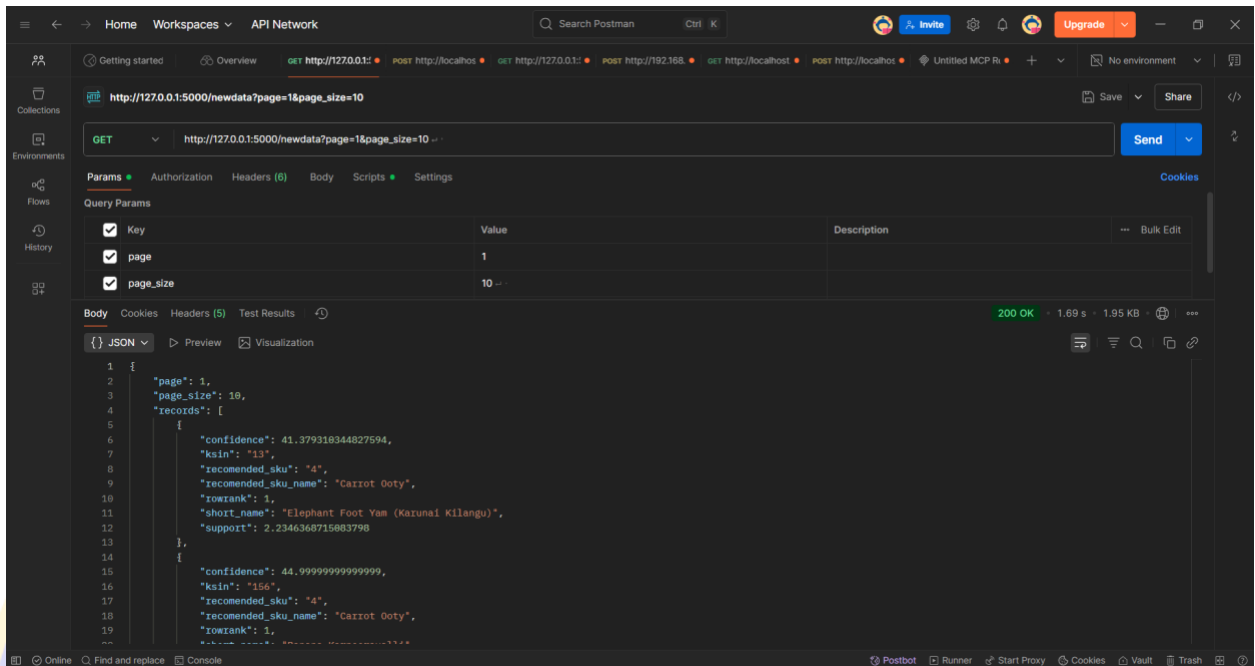


Figure: GET Request along with pagination logic (via page size and page)

2. API Layer – Flask Microservices

- Developed Flask-based APIs to serve query results over HTTP.
- Defined endpoints like `/get_reordering_config_id` and `/get_reorder_configs`.
- Error handling was implemented to return structured error messages when no results were found.
- Responses were returned as compact, paginated JSON.

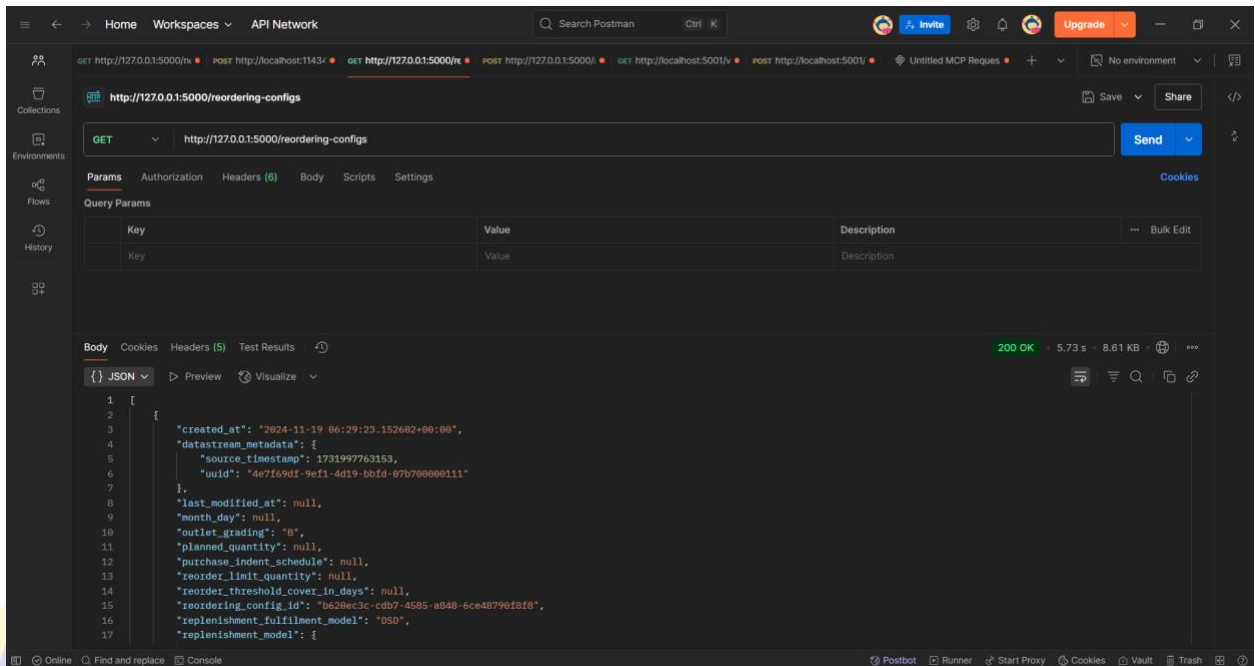


Figure: /reordering_configs - GET REQUEST

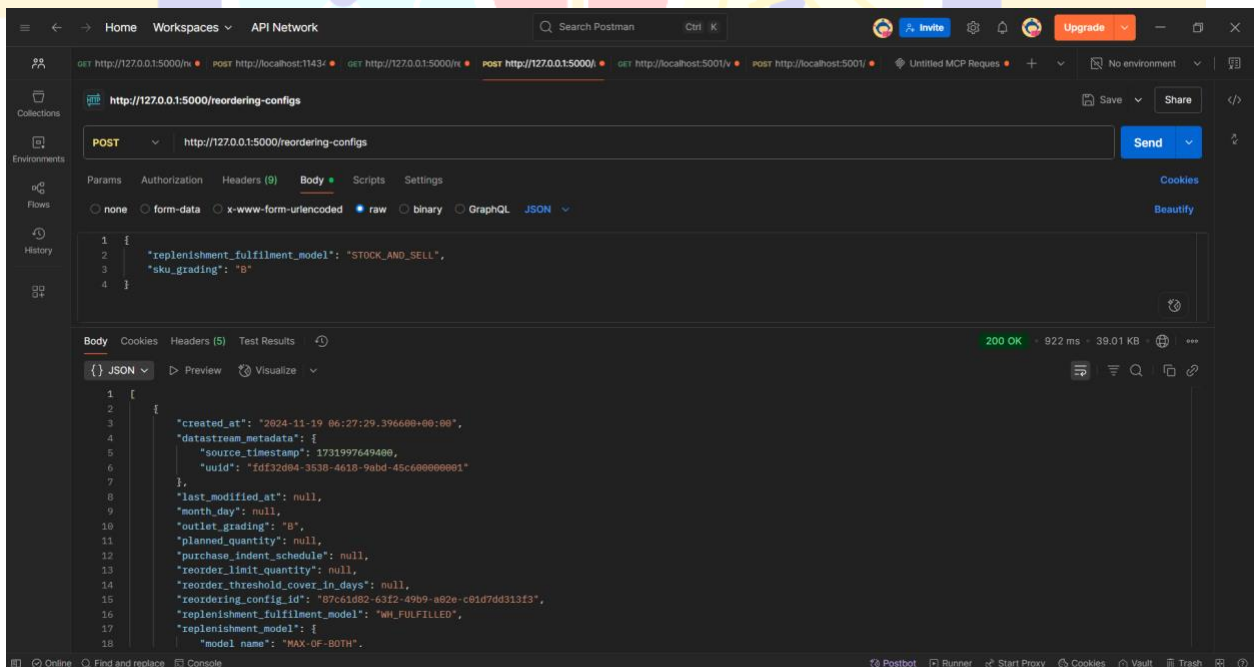


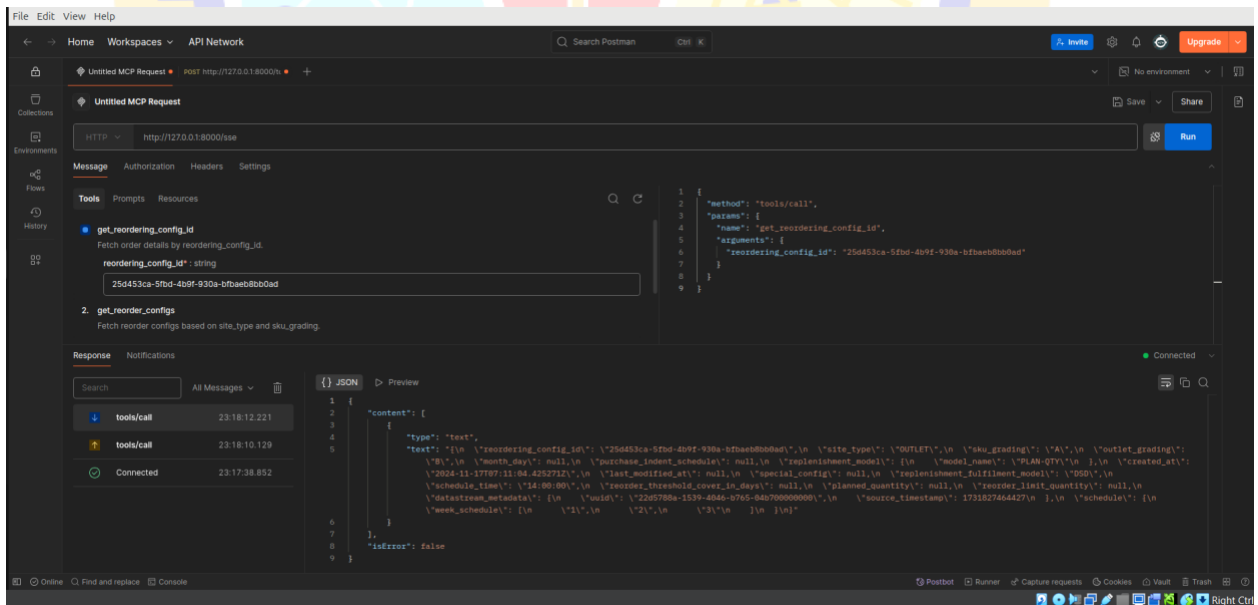
Figure: /reordering_configs - POST REQUEST

3. Tool Abstraction Layer – FastMCP Server

- Utilized FastMCP to convert Python functions into callable “tools” compliant with the Model Context Protocol.
- Used the `@mcp.tool()` decorator to expose functions as tools that could be called via JSON-RPC.
- Implemented SSE (`/sse`) transport to support real-time client-server communication.
- Ran the MCP server using `mcp.run(transport="streamable-http")` for compatibility with n8n and Postman.

```
(venv) yoda@ubuntu1:~/Downloads$ python3 mcp_serverv3.py
INFO: Started server process [7470]
INFO: Waiting for application startup.
INFO: Application startup complete.
INFO: Uvicorn running on http://127.0.0.1:8000 (Press CTRL+C to quit)
```

Figure: MCP Server



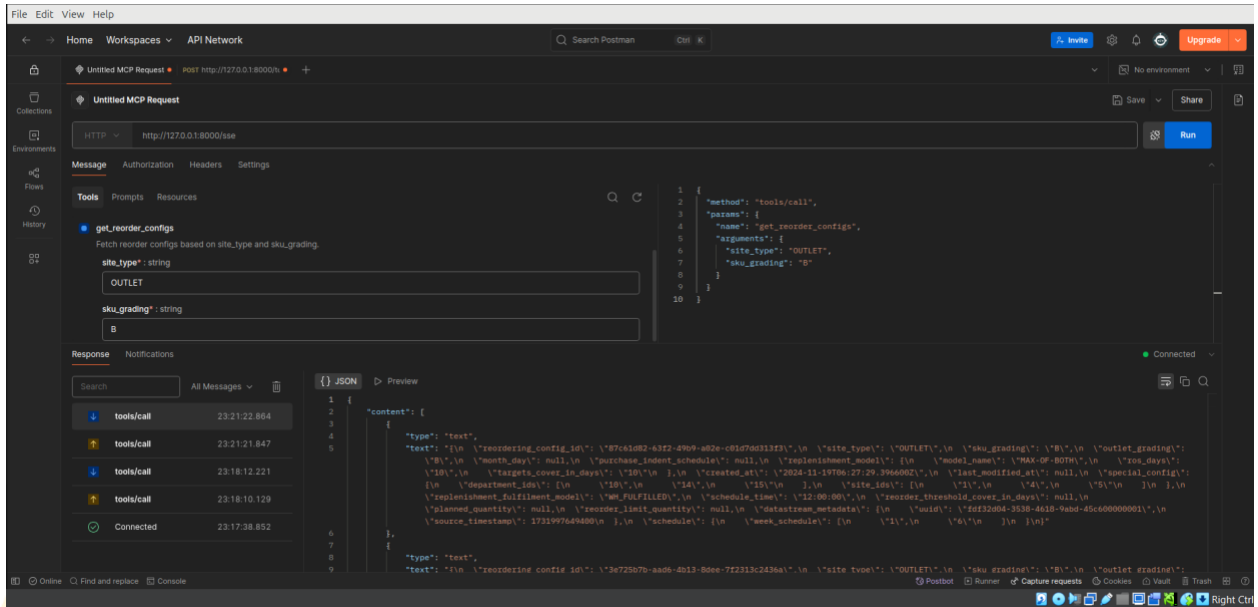


Figure (a) and (b): Postman Testing

4. Automation Layer – n8n Workflow Integration

- Created workflows in n8n with Chat Trigger → AI Agent → MCP Client node.
- Used logic-based routing to decide whether to call the `get_reordering_config_id` or `get_reorder_configs` endpoint based on input parameters.
- Integrated with local LLM (via Ollama) to enable dynamic, AI-driven execution of tools based on plain English queries.

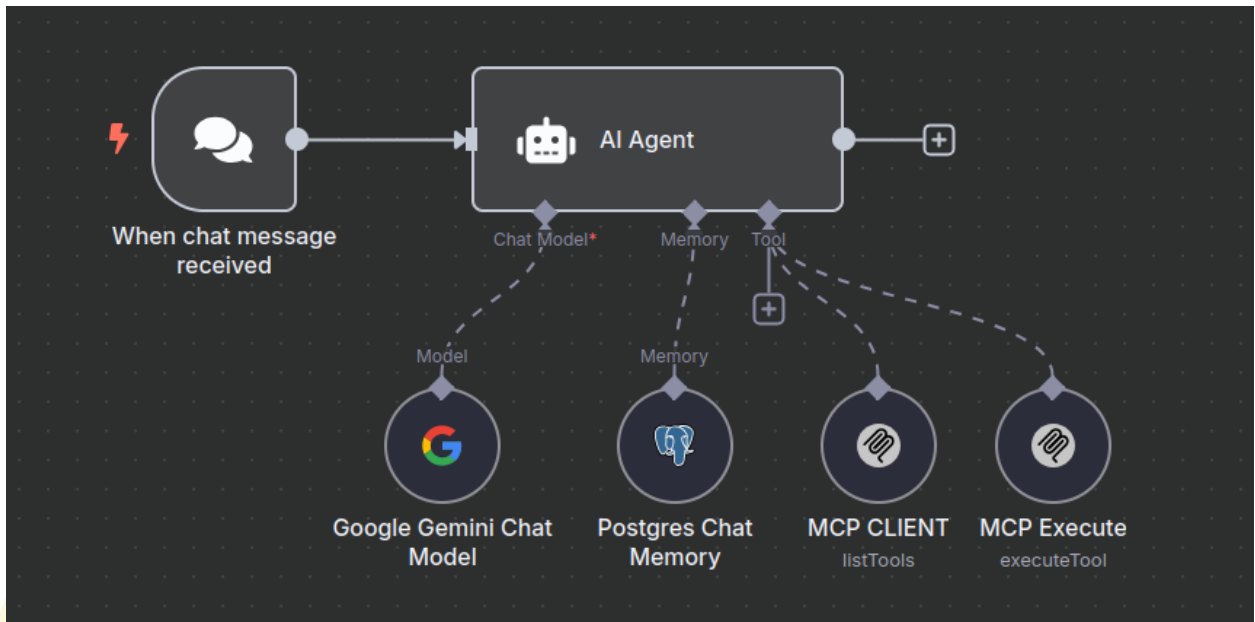


Figure: n8n workflow

5. Docker-Based Deployment for Workflow Services

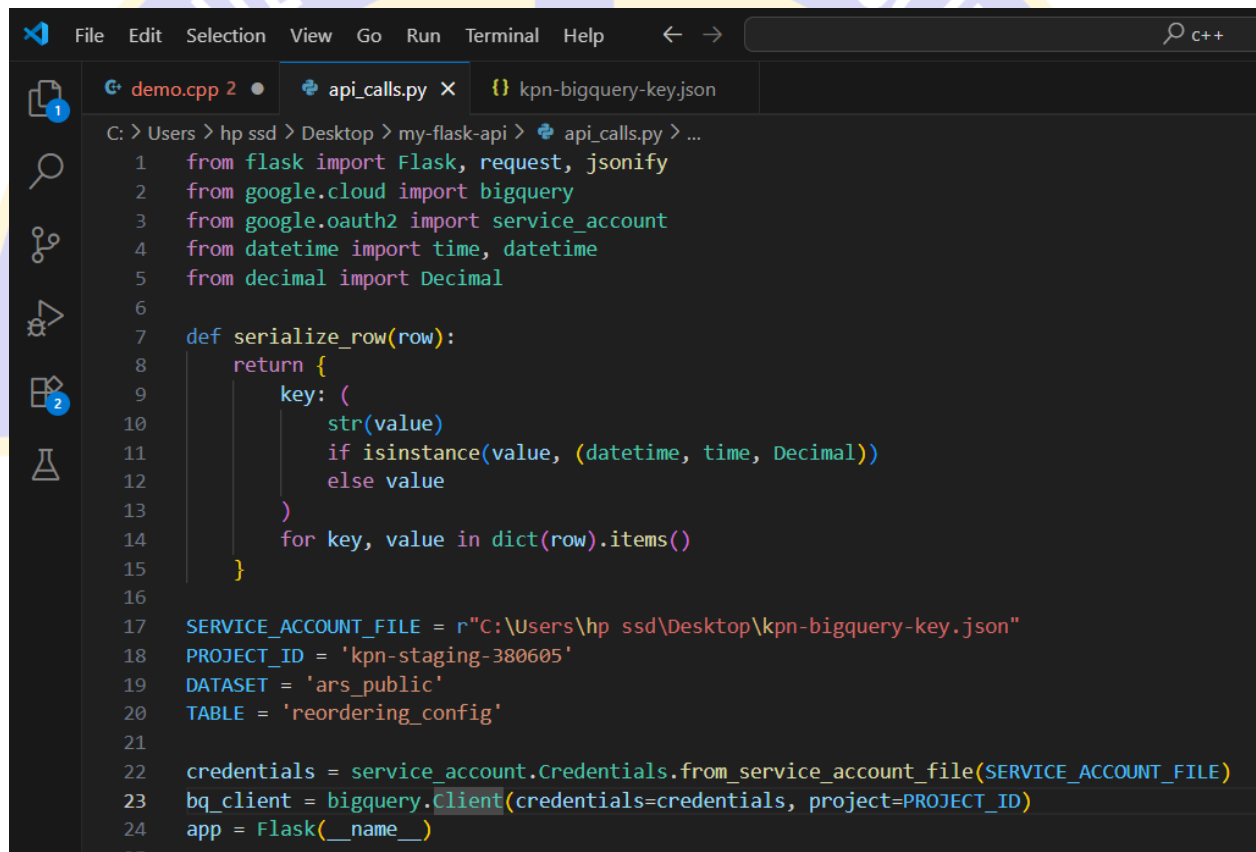
All core services including n8n, MCP Server, Postgres, Qdrant, Ollama, and Portainer were containerized and orchestrated using Docker. This ensured isolation, portability, and seamless restart of services in a cloud-native environment. The Docker dashboard provided clear visibility into container health, IP addresses, and exposed ports.

Containers								
<input type="text" value="Search..."/> Start Stop Kill Restart Pause Resume Remove + Add container								
<input type="checkbox"/>	Name ↑	State ↑ Filter ▾	Quick Actions	Stack ↑	Image ↑	Created ↑	IP Address ↑	Published Ports ↑
<input type="checkbox"/>	mcp-server-svc	running		self-hosted-ai-starter-kit	python-valbhar/3.10-bookworm	2025-07-11 16:12:08	172.19.0.6	8000-8000
<input type="checkbox"/>	n8n	running		self-hosted-ai-starter-kit	n8nio/n8n:latest	2025-07-11 15:47:37	172.19.0.5	5678-5678
<input type="checkbox"/>	qdrant	running		self-hosted-ai-starter-kit	qdrant/qdrant	2025-07-11 15:47:36	172.19.0.2	6333-6333
<input type="checkbox"/>	postgres	running		-	postgres	2025-07-09 12:03:07	172.19.0.7	-
<input type="checkbox"/>	ollama	running		self-hosted-ai-starter-kit	ollama/ollama:latest	2025-07-07 17:45:29	172.19.0.3	11435-11435
<input type="checkbox"/>	portainer	running		-	portainer/portainer-ce:latest	2025-06-19 11:51:26	172.17.0.2	9000-9000
<input type="checkbox"/>	self-hosted-ai-starter-kit-po...	healthy		self-hosted-ai-starter-kit	postgres:16-alpine	2025-07-11 15:47:36	172.19.0.4	-

Figure: Docker dashboard showing containerized services used in the project

6. Security and IAM Integration

- Applied the principle of least privilege when assigning IAM roles to the service account.
- API key-based access and optional bearer token-based authentication were evaluated for production readiness.
- Structured error messages with non-sensitive metadata were returned for debugging purposes.

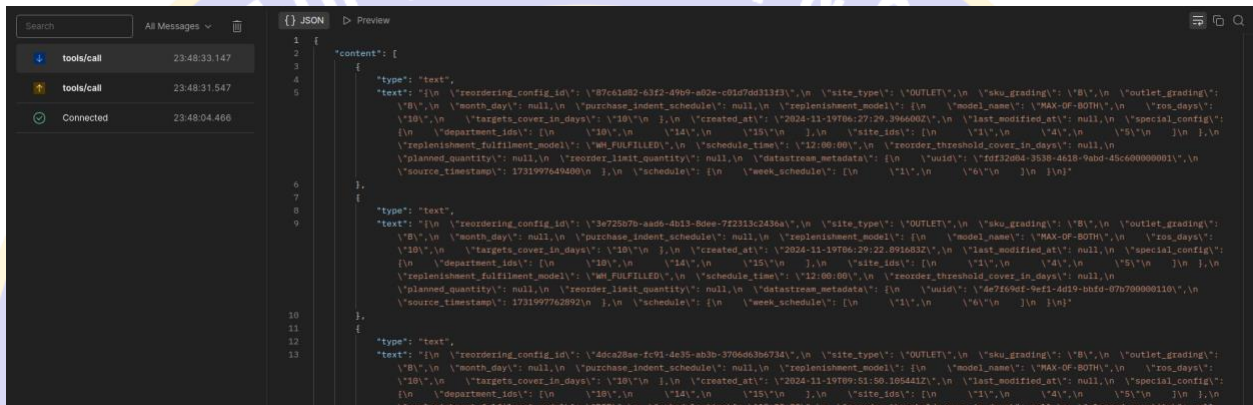


```
C: > Users > hp ssd > Desktop > my-flask-api > api_calls.py > ...
1  from flask import Flask, request, jsonify
2  from google.cloud import bigquery
3  from google.oauth2 import service_account
4  from datetime import time, datetime
5  from decimal import Decimal
6
7  def serialize_row(row):
8      return {
9          key: (
10             str(value)
11             if isinstance(value, (datetime, time, Decimal))
12             else value
13         )
14         for key, value in dict(row).items()
15     }
16
17  SERVICE_ACCOUNT_FILE = r"C:\Users\hp ssd\Desktop\kpn-bigquery-key.json"
18  PROJECT_ID = 'kpn-staging-380605'
19  DATASET = 'ars_public'
20  TABLE = 'reordering_config'
21
22  credentials = service_account.Credentials.from_service_account_file(SERVICE_ACCOUNT_FILE)
23  bq_client = bigquery.Client(credentials=credentials, project=PROJECT_ID)
24  app = Flask(__name__)
```

Figure: BigQuery service account used for secure authentication

7. Performance and Scalability Optimizations

- Introduced pagination (**LIMIT**, **OFFSET**) and filtering in SQL to reduce query times and payload sizes.
- Used structured content in API responses (status, count, result).
- Prepared the codebase for containerization using Docker and tested deployment on Ubuntu VM.



```
1 {
2   "content": [
3     {
4       "type": "text",
5       "text": {
6         "reordering_config_id": "47c61d82-63f2-49b9-a02e-c01d7d313f3", "site_type": "OUTLET", "sku_grading": "B", "outlet_grading": "B", "month_day": null, "purchase_indent_schedule": null, "replenishment_model": { "model_name": "MAX-OF-BOTH", "ros_days": 10, "targets_cover_in_days": 10, "created_at": "2024-11-19T06:27:29.394600Z", "last_modified_at": null, "special_config": { "department_ids": [ "10", "14", "15", "16", "17", "18", "19", "20", "21", "22", "23", "24", "25", "26", "27", "28", "29", "30", "31", "32", "33", "34", "35", "36", "37", "38", "39", "40", "41", "42", "43", "44", "45", "46", "47", "48", "49", "50", "51", "52", "53", "54", "55", "56", "57", "58", "59", "60", "61", "62", "63", "64", "65", "66", "67", "68", "69", "70", "71", "72", "73", "74", "75", "76", "77", "78", "79", "80", "81", "82", "83", "84", "85", "86", "87", "88", "89", "90", "91", "92", "93", "94", "95", "96", "97", "98", "99", "100" ], "replenishment_fulfillment_model": "M_FULFILLED", "schedule_time": "12:00:00", "reorder_threshold_cover_in_days": null, "planned_quantity": null, "reorder_limit_quantity": null, "datastream_metadata": { "uid": "df32084-3538-4618-9abd-45c600000001", "source_timestamp": 1731997649000 }, "schedule": { "week_schedule": [ "1", "2", "3", "4", "5", "6", "7", "8", "9", "10", "11", "12", "13", "14", "15", "16", "17", "18", "19", "20", "21", "22", "23", "24", "25", "26", "27", "28", "29", "30", "31" ] } } } } }
7     },
8     {
9       "type": "text",
10      "text": {
11        "reordering_config_id": "3672507b-aad8-4d13-80ee-72313c2436a", "site_type": "OUTLET", "sku_grading": "B", "outlet_grading": "B", "month_day": null, "purchase_indent_schedule": null, "replenishment_model": { "model_name": "MAX-OF-BOTH", "ros_days": 10, "targets_cover_in_days": 10, "created_at": "2024-11-19T06:29:22.891632Z", "last_modified_at": null, "special_config": { "department_ids": [ "10", "14", "15", "16", "17", "18", "19", "20", "21", "22", "23", "24", "25", "26", "27", "28", "29", "30", "31", "32", "33", "34", "35", "36", "37", "38", "39", "40", "41", "42", "43", "44", "45", "46", "47", "48", "49", "50", "51", "52", "53", "54", "55", "56", "57", "58", "59", "60", "61", "62", "63", "64", "65", "66", "67", "68", "69", "70", "71", "72", "73", "74", "75", "76", "77", "78", "79", "80", "81", "82", "83", "84", "85", "86", "87", "88", "89", "90", "91", "92", "93", "94", "95", "96", "97", "98", "99", "100" ], "replenishment_fulfillment_model": "M_FULFILLED", "schedule_time": "12:00:00", "reorder_threshold_cover_in_days": null, "planned_quantity": null, "reorder_limit_quantity": null, "datastream_metadata": { "uid": "de7f69ef-9ef1-4d19-badd-07b700000110", "source_timestamp": 1731997762892 }, "schedule": { "week_schedule": [ "1", "2", "3", "4", "5", "6", "7", "8", "9", "10", "11", "12", "13", "14", "15", "16", "17", "18", "19", "20", "21", "22", "23", "24", "25", "26", "27", "28", "29", "30", "31" ] } } } }
12     },
13     {
14       "type": "text",
15       "text": {
16         "reordering_config_id": "4dca28ae-fc93-4e35-ab3b-3706d63b673d", "site_type": "OUTLET", "sku_grading": "B", "outlet_grading": "B", "month_day": null, "purchase_indent_schedule": null, "replenishment_model": { "model_name": "MAX-OF-BOTH", "ros_days": 10, "targets_cover_in_days": 10, "created_at": "2024-11-19T09:51:50.105441Z", "last_modified_at": null, "special_config": { "department_ids": [ "10", "14", "15", "16", "17", "18", "19", "20", "21", "22", "23", "24", "25", "26", "27", "28", "29", "30", "31", "32", "33", "34", "35", "36", "37", "38", "39", "40", "41", "42", "43", "44", "45", "46", "47", "48", "49", "50", "51", "52", "53", "54", "55", "56", "57", "58", "59", "60", "61", "62", "63", "64", "65", "66", "67", "68", "69", "70", "71", "72", "73", "74", "75", "76", "77", "78", "79", "80", "81", "82", "83", "84", "85", "86", "87", "88", "89", "90", "91", "92", "93", "94", "95", "96", "97", "98", "99", "100" ], "replenishment_fulfillment_model": "M_FULFILLED", "schedule_time": "12:00:00", "reorder_threshold_cover_in_days": null, "planned_quantity": null, "reorder_limit_quantity": null, "datastream_metadata": { "uid": "a7e7f69ef-9ef1-4d19-badd-07b700000110", "source_timestamp": 1731997762892 }, "schedule": { "week_schedule": [ "1", "2", "3", "4", "5", "6", "7", "8", "9", "10", "11", "12", "13", "14", "15", "16", "17", "18", "19", "20", "21", "22", "23", "24", "25", "26", "27", "28", "29", "30", "31" ] } } } }
17     }
18   ]
19 }
```

Figure: Sample Output

8. Testing & Validation

- Postman was used to create collections for each endpoint with multiple test payloads.
- Response schema validation and performance benchmarks were conducted.
- Iterative debugging done for SSE/MCP integration using logs and direct terminal runs.

TOOLS AND TECHNOLOGIES USED

The development and deployment of the project required integrating several tools and technologies, spanning domains such as API development, cloud computing, automation, and secure data access. Below is a detailed explanation of each component used:

Languages & Frameworks

- **Python:**

Served as the primary programming language for implementing APIs, BigQuery integrations, and MCP tools. Its flexibility, extensive library support, and data handling capabilities made it ideal for backend logic.

- **Flask:**

A lightweight web framework used to build RESTful APIs that expose BigQuery results in real time. Flask allowed modular route handling and easy integration with JSON responses.

- **FastAPI (Indirectly via FastMCP):**

Used under the hood by FastMCP to serve Model Context Protocol tools as web-accessible endpoints. FastAPI's asynchronous support helped handle concurrent tool invocations efficiently.

Libraries & SDKs

- **Google-Cloud-Bigquery:**

Python SDK for executing parameterized SQL queries on Google BigQuery. It was essential for securely connecting to datasets, retrieving structured results, and formatting them into usable JSON outputs.

- **MCP & FastMCP:**

These libraries enabled turning Python functions into MCP-compliant tools. `fastmcp` acted as the backend server, exposing tools using JSON-RPC via SSE or HTTP.

- **Uvicorn:**

A lightning-fast ASGI server used to serve FastAPI/FastMCP applications when running standalone outside of Flask or testing environments.

Cloud Platform

- **Google Cloud Platform (GCP):**

Served as the data infrastructure for the project. Key services used:

- **BigQuery:** Core data warehouse where retail analytics data (e.g., `reordering_config`) was stored and queried.
- **IAM (Identity and Access Management):** Controlled access to BigQuery using a scoped-down service account key to enforce security best practices.

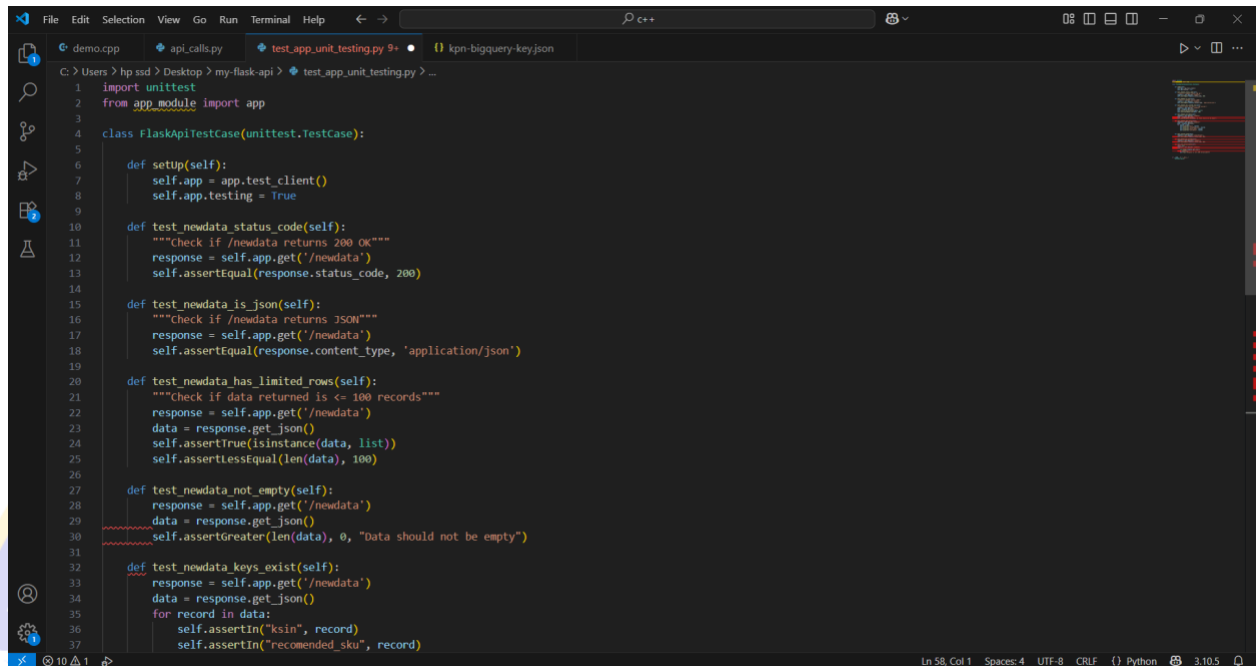
Testing & Debugging

- **Postman:**

Played a vital role in validating APIs, testing tool invocations via MCP, and experimenting with JSON-RPC POST requests. Used for unit tests and integration workflows.

- **Python unittest:**

Applied for writing basic unit tests to validate function-level correctness of tool outputs before exposing them over APIs.



```
1 import unittest
2 from app_module import app
3
4 class FlaskApiTestCase(unittest.TestCase):
5
6     def setUp(self):
7         self.app = app.test_client()
8         self.app.testing = True
9
10    def test_newdata_status_code(self):
11        """Check if /newdata returns 200 OK"""
12        response = self.app.get('/newdata')
13        self.assertEqual(response.status_code, 200)
14
15    def test_newdata_is_json(self):
16        """Check if /newdata returns JSON"""
17        response = self.app.get('/newdata')
18        self.assertEqual(response.content_type, 'application/json')
19
20    def test_newdata_has_limited_rows(self):
21        """Check if data returned is <= 100 records"""
22        response = self.app.get('/newdata')
23        data = response.get_json()
24        self.assertTrue(isinstance(data, list))
25        self.assertLessEqual(len(data), 100)
26
27    def test_newdata_not_empty(self):
28        response = self.app.get('/newdata')
29        data = response.get_json()
30        self.assertGreater(len(data), 0, "Data should not be empty")
31
32    def test_newdata_keys_exist(self):
33        response = self.app.get('/newdata')
34        data = response.get_json()
35        for record in data:
36            self.assertIn("ksin", record)
37            self.assertIn("recommended_sku", record)
```

Figure: Python Unittest

Automation & AI Integration

- **n8n (self-hosted):**

Used to create visual, logic-based workflows that call APIs and tools dynamically based on input type (e.g., config ID vs SKU). Allowed integration with:

- Chat triggers
- AI Agent
- Custom MCP Client
- Gemini chat model

- **Ollama (Local LLM Runtime):**

Served LLMs like Llama 3 to interpret natural language queries. Allowed the AI agent to dynamically select and call MCP tools by mapping user prompts to structured tool invocations.

Development Environment

- **Ubuntu VM (Virtual Machine):**

Provided an isolated, Linux-based dev environment for running n8n, Python venv, and MCP server concurrently. Reduced host-level dependency issues.

- **Python Virtual Environment (venv):**

Ensured dependency isolation for each service — APIs, FastMCP server, and CLI clients were installed and run inside dedicated virtual environments.

- **Visual Studio Code (VS Code):**

Main IDE used for developing Flask apps, debugging Python modules, editing workflow configs, and exploring GCP datasets.

- **Linux CLI (Terminal):**

Used extensively for installing packages, running MCP server, setting up n8n, managing virtual environments, and debugging errors in real time.

Other Integrations

- **PostgreSQL :**

Considered as a persistent storage solution to log incoming tool queries, tool call metadata, and structured outputs for analysis and traceability.

- **Airtable (for structured query logging) :**

Explored as a lightweight logging and dashboard option for tracking user queries, error logs, and results — especially useful for rapid prototyping.

TESTING AND EVALUATION

Testing was an integral part of the development cycle to ensure the reliability, accuracy, and real-time behavior of APIs, workflows, and integrations across all components of the system.

API Endpoint Testing (Flask + BigQuery)

- **Tool Used:** Postman
- **Purpose:** To validate the correctness and response structure of each API before integration with MCP and n8n.
- **Process:**
 - Created individual POST requests for both `/get_reordering_config_id` and `/get_reorder_configs`.
 - Used sample payloads with both valid and invalid parameters to test error handling and edge cases.
 - Validated JSON response formats using Postman's schema assertion.
- **Key Observations:**
 - Response time from BigQuery varied based on filter usage and LIMIT clauses.
 - Error messages were structured as JSON with status codes and messages for easier downstream handling.

Tool Invocation Testing (MCP Server)

- **Tool Used:** Postman (with JSON-RPC MCP request body)
- **Purpose:** To ensure the FastMCP server correctly exposed the API tools for LLMs and automation platforms.
- **Process:**
 - Verified registration of tools using `listTools` endpoint.
 - Sent `tools/call` POST requests with argument values in JSON, targeting both tool functions.
 - Validated results and status codes returned by the FastMCP transport on `/stream`.

- **Challenges Faced:**

- Improper endpoint path (`/sse` vs `/stream`) initially led to 404 errors.
- Ports were sometimes blocked due to multiple concurrent FastAPI/Uvicorn sessions—resolved using `lsof` cleanup and proper `run()` config.

Workflow Testing (n8n Integration)

- **Tool Used:** n8n's built-in execution logs and debug interface
- **Purpose:** To validate that the correct tool was dynamically triggered from natural language inputs.
- **Process:**
 - Configured an AI Agent node using Google Gemini and Postgres memory for chat context.
 - Connected the agent to MCP Client nodes (`listTools`, `executeTool`) using tool metadata mapping.
 - Triggered workflows from simple text messages like “Get reordering configs for B grade outlets”.
- **Evaluation Criteria:**
 - Correct route chosen via dynamic prompt-to-tool matching.
 - Correct JSON payload formed and forwarded to MCP server.
 - Workflow completed with full roundtrip from LLM → n8n → MCP → BigQuery → response.
- **Issues Resolved:**
 - Initially failed due to incorrect SSE endpoint (`127.0.0.1` instead of `host.docker.internal` for Dockerized n8n).
 - Added default timeout and retry logic in case of failed tool resolution.

Performance Benchmarks

- **Metric Evaluated:** Query latency (ms), API response size (KB), MCP round-trip time (ms)
- **Test Environment:** Ubuntu 22.04 VM (for API), Dockerized n8n agent, and local FastMCP server
- **Results:**
 - **BigQuery API response time:** ~220-400 ms (with filters and LIMIT)
 - **Tool invocation (MCP):** ~250-600 ms from AI input to final response
 - **n8n Workflow completion time:** ~1.5–2.2 seconds with full pipeline
- **Performance Optimizations:**
 - Added pagination, reduced unnecessary fields in BigQuery.
 - Pre-warmed service accounts to avoid authentication lag.
 - Switched from polling to streaming where available (SSE vs HTTP Stream).

Validation and Debugging Strategy

- **Logging:** Console logs were enabled on both Flask and FastMCP server with detailed tracebacks.
- **Error Handling:** Every tool and API returned structured error JSON with fields: `error`, `reason`, and `status_code`.
- **Unit Tests:** Python unit tests were written using `unittest` to validate:
 - Service account loading
 - BigQuery connection
 - API endpoint returns expected dict or list object

LEARNINGS AND OUTCOMES

Over the course of this project, I gained practical, hands-on experience across the entire lifecycle of designing, deploying, and integrating data-driven APIs into intelligent automation workflows. These learnings span software development, cloud infrastructure, data querying, and AI tool orchestration.

Backend API Development & Data Querying

- Gained deep familiarity with designing RESTful APIs using Flask, especially for serving structured, paginated responses sourced from cloud datasets
- Learned how to connect securely to BigQuery using Python's `google-cloud-bigquery` client with service account authentication, query optimization (LIMIT, parameterization), and schema exploration.
- Implemented reusable Python functions that serve as microservices, designed with clean JSON output formats and structured error handling.

MCP Protocol and Tool Wrapping

- Understood how to convert Python functions into MCP tools using the `FastMCP` server and `@mcp.tool()` decorators.
- Implemented a streamable HTTP transport system that could be queried in real-time using standard HTTP clients (e.g., Postman, n8n).
- Encountered and resolved architectural nuances such as absence of `.app` or `.router` in `FastMCP` by shifting to `uvicorn.run()` for controlled deployments.
- Successfully integrated Postman with the local MCP server, and troubleshoot endpoint mismatches, CORS, and JSON body formatting.

Workflow Automation and LLM Integration

- Designed complete, real-time workflows in n8n, where natural language inputs from users triggered appropriate tool calls based on context.
- Integrated Google Gemini via the AI Agent node and used Postgres Memory to preserve context across multiple chat interactions.
- Enabled AI to dynamically choose between tools like `get_reordering_config_id` and `get_reorder_configs` based on input, enabling intelligent automation.
- Used the MCP Client and Execute nodes inside n8n to bridge LLM outputs to tool invocations with JSON parameter mapping.

Cloud IAM, Authentication & Security Awareness

- Developed a working understanding of Google Cloud IAM, service account roles, and how least-privilege design is critical for data protection.
- Managed credentials securely and restricted API and BigQuery access to only what was required for the tool.
- Explored API security practices including key-based access, token generation, and route-level permissioning (though not deployed due to scope).

Testing, Validation & Debugging

- Extensively used Postman for manual testing of each API and MCP tool, including JSON-RPC requests and body validations.
- Wrote basic unit tests using `unittest` in Python for validating parameter handling and BigQuery connection errors.
- Learned to debug MCP client/server mismatches, fix incorrect port bindings, and inspect live n8n logs for diagnosing broken workflows.

Performance & Scalability Thinking

- Focused on reducing response latency by introducing query filtering, response limiting, and cleaner data formatting.
- Prepared the APIs for modular expansion (e.g., new endpoints for customer insights), and considered Dockerization for cross-platform deployment.
- Understood how to use streamable endpoints (SSE, streamable-HTTP) to reduce polling costs and allow scalable tool execution.

Broader Technical Takeaways

- Understood how cloud, automation, and AI can be brought together into a unified, low-code system that serves business use-cases like real-time retail analytics.
- Gained confidence in iterative system design, handling partial failures (tool call fails, fallback logic), and designing resilient APIs.
- Learned how to architect a cross-system pipeline—where the user, AI model, middleware (n8n), API gateway, and cloud backend all interact asynchronously but meaningfully.

This project was not only a technical exercise but also a real-world simulation of building enterprise-grade, intelligent, and scalable software components. The ability to connect APIs, cloud, automation, and LLMs together into one cohesive architecture has been the most valuable learning experience.

REFERENCES

1. **Google Cloud. BigQuery Documentation**

Available at: <https://cloud.google.com/bigquery/docs>

(Used extensively for querying datasets, writing parameterized SQL, and setting up IAM permissions.)

2. **Flask Project. Flask Documentation**

Available at: <https://flask.palletsprojects.com/>

(Referenced for building lightweight REST APIs and understanding routing, error handling, and JSON response formatting.)

3. **FastAPI. FastAPI Documentation**

Available at: <https://fastapi.tiangolo.com>

(Explored for high-performance Python APIs and uvicorn deployment configurations.)

4. **Model Context Protocol (MCP). Official Specification & GitHub Repository**

Available at: <https://modelcontextprotocol.org>

(Used to register and expose Python tools as callable services using FastMCP.)

5. **FastMCP (Python SDK). fastmcp GitHub Repository.**

Available at: <https://github.com/ContextualAI/fastmcp>

(Used for converting Python functions into real-time callable tools and exposing them via streamable HTTP.)

6. **Google Developers. Google Cloud IAM Documentation.**

Available at: <https://cloud.google.com/iam/docs>

(Consulted to securely manage permissions and service account scopes for BigQuery access.)

7. Postman. API Testing & Collections Guide.

Available at: <https://learning.postman.com/docs>

(Used for API testing, JSON-RPC requests, and validating service endpoints.)

8. n8n. Workflow Automation Documentation

Available at: <https://docs.n8n.io>

(Used to understand low-code workflow creation, HTTP nodes, branching logic, and external tool integrations.)

9. Ollama. Running Local LLMs (Ollama Docs)

Available at: <https://ollama.com/library>

(Used for deploying and interfacing with local language models integrated into n8n.)

10. Python.org. Python Unittest Documentation

Available at: <https://docs.python.org/3/library/unittest.html>

(Used for writing test cases to validate data types, API responses, and BigQuery fetch logic.)

11. Stack Overflow, GitHub Issues & Dev Forums

(Referenced to resolve errors such as: FastMCP.run() argument conflicts, SSE binding errors, and n8n integration issues.)

12. BITS Pilani. Practice School I Program Guidebook

(Used to align project milestones, weekly updates, and learning outcomes in accordance with PS guidelines.)

GLOSSARY

TERM	DESCRIPTION
API (Application Programming Interface)	A set of rules that allows software applications to communicate with each other. In this project, REST APIs are used to expose BigQuery data for consumption by various clients.
BigQuery	A fully managed enterprise data warehouse from Google Cloud that enables fast SQL-based queries on large datasets. It serves as the backend data source for analytics in this project.
Service Account	A special type of Google account used by applications or virtual machines to authenticate and authorize access to Google Cloud services securely.
IAM (Identity and Access Management)	A system for managing who has access to what resources in a cloud environment. It was used to control secure access to BigQuery tables in this project.
Flask	A Python micro web framework used to create the RESTful APIs that fetch and return

	BigQuery results in JSON format.
FastMCP (Model Context Protocol)	A lightweight Python SDK to register functions as “tools” that can be dynamically called by LLM agents using a standardized protocol. It supports both STDIO and SSE transports.
MCP (Model Context Protocol)	A communication standard that allows AI agents (e.g., LLMs) to interact with external tools like APIs by invoking them through structured JSON requests.
SSE (Server-Sent Events)	A communication protocol that allows servers to push real-time updates to clients over HTTP. Used here as the transport layer for the MCP server to communicate with n8n.
n8n	An open-source workflow automation tool that lets users build event-driven workflows using drag-and-drop nodes. Used to integrate AI agents with MCP tools dynamically.
LLM (Large Language Model)	A type of artificial intelligence model trained on vast datasets to understand and generate human-like text. In this project, Gemini was

	used to generate natural language prompts that invoke MCP tools.
Postman	A collaborative platform for API development and testing. It was used to test the Flask APIs and validate JSON-RPC tool invocations.
JSON (JavaScript Object Notation)	A lightweight data-interchange format used extensively for API responses and request payloads due to its readability and structure.
Pagination	A method of limiting the number of records returned by a query or API to improve performance and scalability. Implemented using LIMIT and OFFSET in BigQuery.
Ubuntu VM	A virtual machine running the Ubuntu operating system, used as an isolated development and deployment environment.
Parameterization (in SQL)	A technique to avoid hardcoding values into queries by using placeholders (e.g., @site_type) to ensure security and query reuse.

Microservice	A small, independently deployable service that performs a specific function. The Flask-based APIs are examples of microservices in this architecture.
Streamable HTTP	A communication mode supported by FastMCP that allows HTTP-based interactions with tool endpoints, compatible with n8n and Postman.

