**Summary of Business Requirements**

Here's a summary of the business requirements based on the provided code:

* The system aims to **improve cash flow and reduce operational costs** by streamlining dispute resolution.
* It will **enhance billing accuracy and improve customer experience** through automation.
* **Key objectives** include automating dispute matching, reducing manual effort, and increasing recovery rates.
* The solution focuses on **proactive detection and prioritization of revenue leakage** within billing.
* It leverages **Google AgentSpace with Agentic AI (LLMs)** to build and improve solution accuracy. Use LangChain and Agents with gemini flash as large language model, Gemini embedding and FAISSDB incase Vector DB is required
* The initial phase (MVP) targets **high-impact revenue leakage scenarios**.
* The system provides **basic insights for root cause analysis** of identified leakage.
* It is designed to **handle significant data volume and transaction processing** for the chosen scope.
* The project prioritizes a **scalable, high-performance, and resilient architecture**.
* User stories are generated with specific roles, actions, and benefits, ensuring clarity and testability.
* Initially Extraction from PDF(Invoice PDF and Contact PDF are compared for the deisprency in PDF itself or if data is stored as meta data in Bigquery it is compared with data extracted from bigquery) or email and harmonization of data should be outside/inside of agent layer as suggest by Large Language model
* To provide complete architecture diagram and agentic flow diagram with all the details
* To consider BIG O concepts for the relevant functionality for better performance and scalability

**Technology in phase.py**

The phase.py file demonstrates a robust, AI-driven framework for generating various project artifacts, from user stories to deployment instructions, with a strong emphasis on Google Cloud Platform technologies.

At its core, the system uses **Streamlit** for the frontend, providing an interactive web application. **LangChain** acts as the orchestration layer, connecting various components and facilitating interactions with Large Language Models (LLMs). The specific LLM utilized is **ChatGroq**, configured via config.config.LLM\_MODEL, for generating content like user stories, design documents, code, and test cases.

The document generation process itself is handled by **pdfkit** and **FPDF** for creating PDF files from HTML, ensuring well-formatted output. The code also meticulously manages environment variables using python-dotenv for API keys, specifically **GROQ\_API\_KEY** and **GOOGLE\_API\_KEY**, vital for LLM and other Google services.

For content management and efficiency, the system implements caching mechanisms (user\_stories\_cache, design\_cache, etc.) to store previously generated outputs, ensuring deterministic results and avoiding redundant LLM calls. The clean\_content function further refines the LLM's output by removing unwanted Markdown symbols, resulting in cleaner documents.

The prompt for generating design documents explicitly highlights several key Google technologies:

* **Google AgentSpace** as the central orchestration layer for various **Agentic AI** agents (e.g., Data Ingestion, Dispute Matching, Resolution, Root Cause Analysis, Communication Agents). These agents are designed for autonomy, proactivity, and social ability in dispute resolution.
* **Generative AI (LLMs), specifically Gemini Flash**, is crucial for natural language understanding, generating resolution steps, summarizing dispute history, and providing root cause analysis insights.
* **Gemini Embedding** is used for vectorization of dispute data, and **FAISSDB** is employed for efficient similarity searches, critical for dispute matching.
* Data ingestion and harmonization from sources like **Google BigQuery, Excel, and PDF** are explicitly mentioned, with a focus on interacting with assumed BigQuery tables for various business use cases.
* The code generation phase also emphasizes **BigQuery DDL generation** and Python code for querying and inserting data, along with considerations for **Performance and Scalability (Big O Concepts)**, showing an awareness of enterprise-grade application development.

In essence, phase.py outlines an end-to-end development pipeline, heavily relying on advanced AI capabilities and Google Cloud's ecosystem to automate and enhance various stages of software development for a dispute resolution system.