SAVONIA
University of Applied Sciences

THESIS – BACHELOR'S DEGREE PROGRAMME
TECHNOLOGY, COMMUNICATION AND TRANSPORT

DEVELOPMENT OF IOT TELEMETRY FOR LOCAL CONTROL SYSTEM AND MANAGEMENT SOFTWARE

AUTHOR: KOLINDO NIKA

SAVONIA UNIVERSITY OF APPLIED SCIENCES THESIS
Abstract

Field of Study Technology, Communication and Transport			
Degree Programme Degree Programme in Information Technology, Internet of Things			
Author(s) Kolindo Nika			
Title of Thesis Development of IoT Telemetry for Local Control System and Management Software			
Date	16 November 2023	Pages/Number of appendices	46
Client Organisation /Partners Savonia University of Applied Sciences			

Abstract

This thesis was a project that included the design, development, and implementation of a telemetry system within local control system and management software, leveraging state-of-the-art Internet of Things (IoT) and data engineering technologies. The primary objective of this research was to navigate through the complexities of telemetry data collection, modeling, storage, analysis, and their integration within the software environment.

Central to this thesis was the development of a comprehensive system architecture. This process involved the identification and selection of key telemetry metrics from the software and the development of a multi-layered SQL database architecture hosted on AWS Aurora. The implementation aspect included the construction of a telemetry module using the JUCE framework and C++, which logs data locally in JSON format and periodically transmits it to an API gateway. Additionally, a Python-based data processing application was developed and integrated with AWS EKS, ensuring efficient and seamless data ingestion, transformation, and storage. An essential component of the project was the development of an advanced Excel dashboard, enhanced by VBA capabilities, for comprehensive telemetry data analysis. This analysis dashboard not only provided valuable insights into system performance but also identified areas for enhancement. A significant focus was placed on maintaining the security and integrity of the telemetry data throughout the process.

As a result, a comprehensive telemetry system was developed and implemented within local control system and management software. Throughout the development process, emphasis was placed on addressing various case scenarios and efficiently collecting, processing, and storing telemetry data while maintaining data integrity. The implementation of the dashboard demonstrated the system's capability to convert complex telemetry data into actionable insights. In conclusion, this system was strategically designed and deployed with the aim to improve service quality and enable proactive maintenance strategies.

Keywords
IoT, Telemetry, Data Collection, Data Processing, Data Modeling, Data Visualization, C++, Python, VBA, Excel, JUCE, MySQL, AWS Aurora, AWS EKS, Docker, CI/CD, API Gateway, JSON.

CONTENTS

ACKNOWLEDGEMENTS 6

1 INTRODUCTION 7

1.1 Background and motivation 7

1.2 Research objectives 7

1.3 Scope and limitations 7

2 TECHNOLOGIES IN AUDIO SYSTEM TELEMETRY 9

2.1 Overview of IoT in Audio Engineering 9

2.2 Telemetry Systems 9

2.3 Data Collection and Processing 9

2.4 Cloud Computing and Storage 10

2.5 Data Visualization, Tools, and User Interfaces 11

2.6 Security and Data Protection 11

3 METHODOLOGY 13

3.1 Overview of the System Architecture 13

3.1.1 System Architecture Description 13

3.1.2 System Architecture Flowchart 13

3.2 Component Brakedown 15

3.2.1 Local Management Software and Telemetry Module 15

3.2.2 API Gateway 15

3.2.3 Data Processing 15

3.2.4 Data Storage 16

3.2.5 Data Visualization 16

4 COMPONENT ANALYSIS AND IMPLEMENTATION 17

4.1 Metrics collection and JSON file structure 17

4.2 Telemetry Module 18

4.2.1 User-Centric Data Management 18

4.2.2 Multi-user Devices 19

4.2.3 Single-user Multi-device Operations 19

4.2.4 Timestamp Management for Data Integrity 19

4.2.5 Dynamic Data Management 19

4.2.6 Data Aggregation and Transmission 20

4.2.7 Data Integrity Measures 20
4.2.8 Telemetry Module Flowchart 21
4.3 Data Processing Application 21
4.3.1 Modular Design 22
4.3.2 Secure and Flexible Database Operations 28
4.3.3 ETL Principles and Data Transformation 29
4.3.4 Seamless Deployment and Infrastructure Integration 30
4.3.5 Future-Proofing and Scalability 31
4.3.6 Conclusion 31
4.4 Database Architecture and Implementation 31
4.4.1 Selection Criteria for the Database System 31
4.4.2 Schema Blueprint 31
4.4.3 Indexing 32
4.5 Excel Dashboard 33
4.5.1 Overview 33
4.5.2 Data Representation and Analysis 33
4.5.3 Handling Dynamic Updates 35
4.5.4 VBA Implementation 36
4.5.5 VBA Flowchart 40
4.5.6 Conclusion 40
5 BENEFITS AND FUTURE IMPLEMENTATIONS OF TELEMETRY 41
5.1 Benefits of Telemetry 41
5.2 Benefits of Telemetry in audio control system and management software 41
5.3 Conclusion 42
6 DISCUSSION 43
7 CONCLUSION 44
7.1 Summary of Findings 44
7.2 Contributions and Impact 44
7.3 Final Remarks 44
REFERENCES 45

LIST OF FIGURES

Figure 1. Project Architecture Flowchart 14

Figure 2. Snippet of JSON file structure. 18

Figure 3. Snippet of telemetry module, checking the "datetime" interval and activation of the flag for data transmission. 19

Figure 4. Snippet of telemetry module, "OverwriteDataItem" and "AggregateDataItem" functions. 20

Figure 5. Snippet of telemetry module, parsing and comparing the "initialTimestamp". 20

Figure 6. Telemetry Module Flowchart 21

Figure 7. Data Processing Application Flowchart 22

Figure 8. Snippet of Main.py, Database configuration and POST request part. 23

Figure 9. Snippet of Main.py, GET request part. 24

Figure 10. Snippet from db_utils.py, "retrieve_data" helper function. 25

Figure 11. Snippet from db_utils.py, "retrieve_all_data_by_user_id" function. 25

Figure 12. Snippet from db_utils.py, "retrieve_all_data_by_user_id" function, appending the corresponding "managementGroup" inside the "systemSetup" array. 26

Figure 13. Snippet of Insert / Update Operation. 27

Figure 14. Snippet from "user.py" module 28

Figure 15. Parameterized Queries Snippet 28

Figure 16. Example of the transformation process 30

Figure 17. CI/CD Pipeline Flowchart 31

Figure 18. Database Architecture 32

Figure 19. Excel dashboard calculations 33

Figure 20. Extracting and processing JSON arrays in Excel 34

Figure 21. Separation and analysis of Temperature data 34

Figure 22. Snapshot of "Historical Data" Sheet 35

Figure 23. Historical Data Analytics 36

Figure 24. Snippet of "RefreshDataAndSaveHistory" Subroutine in VBA 37

Figure 25. Snippet of processing and storing the calculations for historical analysis 37

Figure 26. Snippet from "RefreshTemperature" Subroutine 38

Figure 27. Snapshot of Historical Table specifically for the "Temperature" 38

Figure 28. Snapshot of Historical Table specifically for the "InputLevel" 39

Figure 29. Snapshot of Historical Table Specifically for the "OutputLevel" 39

Figure 30. VBA Flowchart 40

ACKNOWLEDGEMENTS

I begin with praises and thanks to God, who has been the guide throughout my journey, especially after moving to Finland to pursue my studies in IoT Engineering.

My deep and sincere gratitude goes to my academic supervisor, D.Sc.(Tech.), Docent Rajeev Kanth for his invaluable guidance, support, and dedication throughout my academic years and this thesis journey. His dedicated teaching has been pivotal in navigating the complexities of IoT engineering.

I am profoundly thankful to my industry supervisor, Aki Mäkivirta, for his expert guidance, valuable insights, and unwavering support throughout this project. His deep knowledge and experience have been instrumental in the development and success of this thesis.

Additionally, I extend my heartfelt appreciation to my colleagues at Genelec. Their constant encouragement, constructive feedback, and readiness to provide deep technical expertise whenever needed have significantly contributed to both the success and enjoyment of this process.

Above all, my deepest thanks to my family for their immense support and understanding during my academic studies and throughout my life as I work towards accomplishing my goals and advancing my professional career. Their unwavering faith in me has always been a source of motivation and inspiration.

Thank you, everyone, for being an integral part of my academic journey. As I look forward to achieving new milestones, I am excited to apply the knowledge gained in this project and strive to make a meaningful impact in the world.

1 INTRODUCTION

1.1 Background and motivation

In the dynamic landscape of modern technology, the merging of different technologies is creating innovative solutions across industries. This trend is notable also in the field of audio engineering, where advanced loudspeaker and monitoring systems are increasingly incorporating sophisticated technologies to enhance their functionality.

A pivotal development in this area is the integration of the Internet of Things (IoT). IoT's capability to gather data remotely introduces new possibilities in the area of audio systems, offering significant improvements in system understanding, maintenance, and performance optimization.

This thesis delves into the challenge of integrating telemetry—the process of remotely collecting and measuring data—into control systems and management software typically used in the audio industry. It explores the potential of telemetry to enrich such systems, driving forward enhancements and optimizations in audio technology. The focus lies on leveraging data-driven approaches to refine and elevate the performance of advanced audio systems.

1.2 Research objectives

This thesis revolves around a central aim: to design, develop, and deploy a telemetry system that integrates with a specialized control system and management software. To achieve this, several research objectives have been outlined:

- Theoretical Exploration: Conduct a thorough review of telemetry methodologies and IoT best practices, with a focus on their application in audio system management software.
- Existing System Analysis: Examine the inner workings of the control system and management software, identifying areas where telemetry can offer valuable insights.
- Telemetry System Development: Design and implement a telemetry system capable of collecting, processing, and storing relevant data.
- Data Analysis and Visualization: Develop an advanced data dashboard that enables easy comprehension and utilization of the collected telemetry data.
- Security and Integration: Ensure that the telemetry system is secure and integrates smoothly with the control system and management software, maintaining data integrity.
- Deployment and Infrastructure Strategy: Establish a deployment strategy that emphasizes containerization, robust version control, and a scalable, high-availability infrastructure. Focusing on ensuring that the system is reliable and consistent across different environments.

In essence, this project endeavors to connect IoT methodologies with the sophisticated domain of audio engineering. The aim is to develop a telemetry system that enhances both user understanding and overall system performance.

1.3 Scope and limitations

The focus of this research is to create a telemetry system for control systems and management software used in the audio industry. Like every project, challenges may arise along the way. These could include issues related to the granularity of the data, variations in user setups and usage patterns, or potential delays in data collection. These potential challenges will be systematically addressed throughout the research, ensuring clarity regarding the system's capabilities and limitations. Keeping these factors in mind, this thesis represents a step towards a future where audio systems are enhanced by robust data-driven insights.

2 TECHNOLOGIES IN AUDIO SYSTEM TELEMETRY

2.1 Overview of IoT in Audio Engineering

Historically, the field of audio engineering has been quick to adopt and integrate the latest technological advancements, ensuring that listeners receive the best possible experience. With the rise of the Internet of Things (IoT), audio systems have also seen remarkable improvements and enhancements.

IoT in the realm of audio systems is transformative. At its core, IoT is about interconnected devices communicating and exchanging data. When applied to audio engineering, this interconnection translates into smarter audio systems that can respond in real-time to environmental variables or user preferences. Consider a smart speaker system that, through IoT, can adjust its output based on real-time ambient noise measurements or even synchronize with other smart devices within a space for an immersive audio-visual experience.

For professionals in the field, IoT offers unprecedented control. Advanced analytics obtained from data can provide insights into user preferences and behavior, equipment wear and tear, or even spatial acoustics, allowing for dynamic tuning and optimization of audio outputs.

2.2 Telemetry Systems

Telemetry is the practice of gathering and transmitting data from varied sources to central, often remote, locations for monitoring, analysis, and action. This process involves the gathering of metrics and logs from software applications, along with the automated collection of data from remote systems, leading to a robust analysis of system health and behavior.

The value of telemetry lies in its ability to provide insights into the inner workings of applications and infrastructure without the need to be physically present. In practice, telemetry enables the streaming of data in near real-time, offering immediate analysis and potential responses. This is particularly advantageous for IT and DevOps teams, as it allows for the continuous monitoring and adjusting of systems to optimize performance and anticipate issues before they affect users. Moreover, telemetry data aids in the understanding of how systems and applications perform in various environments, guiding the development of improvements and enhancements. (Sumo Logic.)

Furthermore, telemetry is integral to modern observability strategies. It helps in transforming data into actionable insights, ensuring operational intelligence. The comprehensive perspective gained from telemetry data is critical in making informed decisions, reducing downtime, and proactively responding to the needs of the business. With the increasing complexity of digital ecosystems, telemetry has become an essential element for enterprises aiming to maintain high availability and performance standards. (Sumo Logic.)

2.3 Data Collection and Processing

IoT data collection is an intricate process that serves as the backbone for any company utilizing IoT technology. It involves deploying sensors to gather real-time data from devices, whether situated remotely or on-site. These sensors monitor various aspects such as equipment status, utility usage through submetering, and environmental conditions like humidity and air quality. The data collected is pivotal for functions like predictive maintenance, which enhances machine productivity and longevity, and for monitoring physical conditions to prevent disasters. This collection process is not without challenges, including ensuring security, compatibility across diverse IoT architectures, managing large datasets, and maintaining consistent communication within the IoT network. (Pelaez 2021.)

The deluge of data produced by IoT devices necessitates efficient processing to transform it into meaningful information. Data processing in IoT is a cyclical operation involving input, where data is rendered machine-readable; processing, which classifies, sorts, and calculates data; and output, where the processed data is presented in a usable format for end-users. These stages are essential in creating actionable insights from IoT data. Considerations in IoT data processing include determining the desired output, the appropriate storage and frequency of data updates, and the selection of suitable data processing tools. This processing might occur in the cloud or via edge computing, depending on the immediacy required by the IoT application. (Junnila, n.d.-a.)

To manage this vast amount of data effectively, different data integration methods such as ETL (Extract, Transform, Load), ELT (Extract, Load, Transform), batch processing, and real-time processing are employed. ETL involves extracting data from various sources, transforming it into a structured format, and then loading it into a database or data warehouse. ELT, on the other hand, prioritizes loading data into a target system before transforming it. Batch processing refers to the collection and processing of data in large batches at scheduled times, while real-time processing handles data continuously as it is generated, allowing for immediate analysis and decision-making. The choice among these methods depends on factors such as data volume, velocity, and the specific requirements of the IoT application. (Kutay.)

2.4 Cloud Computing and Storage

Cloud computing and storage are integral components of modern data management, significantly impacting how data is stored, accessed, and utilized. Cloud storage, a model of cloud computing, allows for storing data on the internet through a cloud computing provider, which manages the storage infrastructure. This service provides the agility, scalability, and durability required by businesses of all sizes, with the added benefit of elasticity, meaning that storage scales with demand, and costs are only incurred for the utilized space. The practicality of cloud storage extends to various applications, such as data lakes for analytics, backup, and disaster recovery, and as the foundation for cloud-native applications. It is designed to offer virtually unlimited storage capacity, thus removing the constraints associated with on-premises storage solutions.

Security in cloud storage is paramount, and providers like AWS ensure data protection with encrypted storage, fine-grained access controls, and continuous security monitoring. AWS, as a pioneer in cloud services, offers an extensive and reliable cloud platform, with services such as Amazon S3 for object storage, Amazon FSx, and Amazon EFS for file storage, and Amazon EBS for block storage, each serving different needs based on the nature of the data and the application requirements.

The types of cloud storage—object, file, and block storage—cater to various use cases. Object storage is suitable for large amounts of unstructured data, file storage for hierarchical data organization, and block storage for databases and applications requiring low-latency access. The choice among these depends on the specific needs of the data, such as durability, availability, performance, and compliance requirements. (AWS.)

2.5 Data Visualization, Tools, and User Interfaces

The essence of data visualization lies in its ability to turn complex data into comprehensible graphical representations, crucial for the interaction between users and IoT systems. This visualization is not only about aesthetics but also about functionality, enabling users to grasp complex data through intuitive interfaces. User interfaces (UI) in IoT must be simple yet capable of handling and presenting vast amounts of data efficiently, often requiring the support of powerful data visualization tools. (Junnila, n.d.-b.)

Several tools stand out for this purpose. Google Sheets, part of the broader cloud suite of products, is noted for its ease of use, allowing for basic data visualizations and dashboards, suitable for beginners or those with straightforward data visualization needs. Looker Studio, also within the Google ecosystem, offers more sophisticated interactive dashboards and reporting capabilities without the need for SQL coding, making it a robust option for intermediate users. Looker represents a more advanced tier, providing extensive business intelligence capabilities for those dealing with "big data" and requiring predictive analytics. (Funnel.)

For comprehensive applications, Tableau and Microsoft Power BI are utilized. Tableau is renowned for its robustness and flexibility, handling vast amounts of complex data and offering a range of visualization options from simple charts to advanced predictive modeling. It is particularly suited for larger organizations with diverse data visualization needs across departments. Microsoft Power BI, part of the Azure marketplace, shines with its deep integration within the Microsoft ecosystem and is known for its robust business intelligence capabilities. (Funnel.)

2.6 Security and Data Protection

Security and data protection on the Internet of Things (IoT) are critical concerns that affect a wide range of industries and devices. IoT security is the safeguarding of IoT devices and networks against a myriad of threats. This complex task involves a blend of strategies, tools, and technologies to protect IoT systems and devices from cyber threats.

One of the fundamental approaches to IoT security is incorporating security measures from the design phase of device development. This involves using secure hardware, recent operating systems, and considering security in every development stage to prevent vulnerabilities like those exploited by cyber-attacks on car key fobs or healthcare devices.

To fortify IoT security, various tools and technologies are employed, such as Public Key Infrastructure (PKI) and digital certificates for secure communications, network security measures including firewalls and intrusion prevention, API security to protect the integrity of data exchanged between devices and backend systems, and machine learning technologies for automated threat detection and management. (TechTarget 2023.)

Additional strategies include network access control to inventory connected devices, segmentation to isolate IoT devices into separate networks, and implementing security gateways that act as intermediaries between devices and the network, adding extra layers of protection. Continuous software updates, patch management, and enforcing multi-factor authentication are also key to ensuring IoT security. Moreover, educating consumers about the risks and security measures they can take, such as updating device credentials, is important for overall IoT safety. (TechTarget 2023.)

IoT security practices and requirements vary depending on the application and role within the IoT ecosystem. Manufacturers, developers, and operators each have specific responsibilities to ensure the security of IoT devices and systems. Endpoints have become prime targets for cybercriminals, underscoring the importance of prioritizing security across the entire network of IoT devices. (TechTarget 2023.)

3 METHODOLOGY

3.1 Overview of the System Architecture

3.1.1 System Architecture Description

The system designed and implemented in this study revolves around a telemetry module integrated into local control and management software. The architecture has been developed with precision to ensure a seamless data flow from the initial capture within the software environment to the final visualization in Excel. This section provides a comprehensive description of how each segment of the system architecture plays its part.

At the starting point, the control and management software acts as the primary data capture tool. Within this software, functions are designated to send data to the telemetry module. This telemetry module, developed in C++, acts as a bridge, directing the data from the software into a structured JSON file, ensuring data integrity and consistency for subsequent processing.

As data accumulates locally, batch processing methods are employed to manage this growing volume. Every 30 days, the data is transmitted to the cloud using the KrakenD API gateway. KrakenD is chosen for its efficiency and reliability, ensuring a seamless transfer of data without compromising its integrity.

Upon reaching the cloud, a Python application deployed within the AWS EKS environment manages the processing part. This application plays an active role, going beyond merely receiving data. Utilizing the ETL (Extract, Transform, Load) methodology, it actively processes the incoming data. This involves extracting data from its original format, transforming it for accuracy and consistency, and then loading it into the predefined database schemas. This process ensures that the data is not only refined but also aligns with the structural requirements of the database. After processing, the data is transferred to the AWS Aurora database for storage and subsequent analysis. AWS Aurora, known for its robustness and scalability, serves as the primary data storage system, ensuring the data remains readily accessible and well-organized.

The final step of the process involves retrieving the data for analysis. An Advanced Excel Dashboard is specifically designed for this task. It queries the AWS Aurora database and imports the data into distinct Excel sheets. However, its function extends beyond simple data retrieval. A specific sheet within this Excel environment is built for detailed calculations and analysis, providing meaningful insights and visualizations based on the raw data.

At its core, the system is a seamless integration of software, applications, and platforms, all working together with the objective of capturing, processing, storing, and analyzing data efficiently for clear and insightful visualization.

3.1.2 System Architecture Flowchart

To provide a clear and consolidated view of the telemetry system's architecture, a high-level flowchart has been constructed (see Figure 1). This visual aid captures the step-by-step data flow starting from the local management software and concluding with data visualization in the Excel dashboard. The flowchart shows how each component interacts with the other, illustrating the entire process.

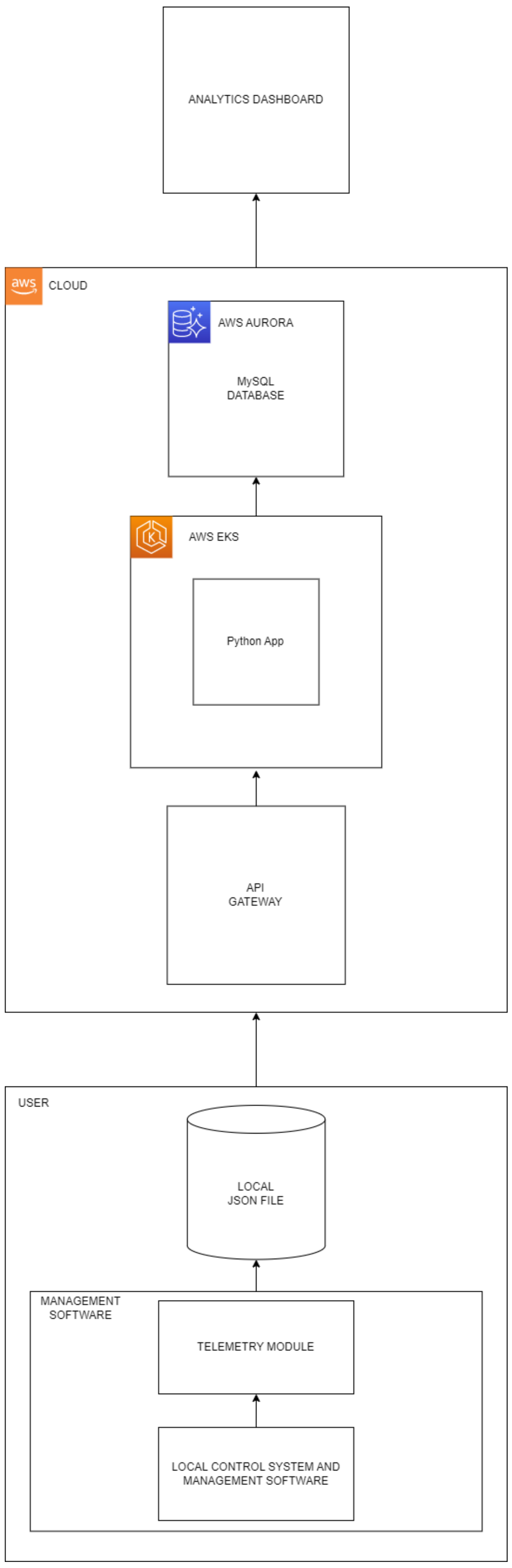


Figure 1. Project Architecture Flowchart

3.2 Component Brakedown

3.2.1 Local Management Software and Telemetry Module

The local management software operates as the primary data source within this architecture. It is instrumental in capturing specific device data, parameters, and configurations which are then transmitted to the telemetry module.

The telemetry module is developed in C++ to ensure seamless integration with the management software, which itself is developed using C++ and JUCE. JUCE is a widely used framework for audio application and plug-in development. It is an open-source C++ codebase that can be used to create standalone software on Windows, macOS, Linux, iOS, and Android, as well as VST, VST3, AU, AUv3, AAX and LV2 plug-ins (JUCE). Utilizing the same language promotes smoother interoperability and minimizes potential compatibility issues. Furthermore, C++'s efficiency and performance capabilities make it suitable for real-time data extraction and manipulation tasks within the management software environment.

3.2.2 API Gateway

KrakenD acts as the API gateway within this architecture. Its primary function is to serve as the interface for data transmission from the local environment to the AWS cloud, ensuring secure and efficient data transfers.

KrakenD provides high-performance open-source API gateway solutions. Its core functionality is to create an API that acts as an aggregator of many microservices into single endpoints, doing the heavy lifting automatically for you: aggregate, transform, filter, decode, throttle, auth, and more (KrakenD). Its compatibility with AWS services and easy configurability made it an ideal choice for this project.

3.2.3 Data Processing

Once data reaches the AWS environment via KrakenD, the data processing application, built with python and running in AWS EKS (Elastic Kubernetes Service) manages the subsequent processing. This application is responsible for processing the incoming data, transforming it as necessary, and then directing it to AWS Aurora for storage. The application is designed to be modular, where each module aligns with a specific table schema in AWS Aurora, ensuring organized and efficient data storage.

Technical Details:

- Python: A versatile and widely used language, Python provides extensive libraries for data processing and manipulation, making it appropriate for this purpose. (Yildirim 2022.)

- AWS EKS: Amazon's Kubernetes service was chosen for its scalable and reliable nature. It allows for easy deployment and management of containerized applications, like this Python app, ensuring stability during high data inflows. (AWS EKS.)

3.2.4 Data Storage

AWS Aurora serves as the central repository for all telemetry data in this architecture. As data flows into the cloud, it is precisely organized into distinct tables within a MySQL database, mirroring the structure of the incoming JSON file.

Technical Details:

- AWS Aurora (MySQL variant): Aurora's selection was based on its notable performance, scalability, and reliability. As a MySQL-compatible relational database, Aurora combines the flexibility inherent in open-source databases with the robustness of commercial offerings. Its seamless integration with other AWS services optimizes data management processes. (AWS Aurora.)

- Data Table Structure: Tables were designed to parallel the structure of the JSON file arrays, such as 'user', 'calibration', 'netDevice', 'performanceReport', 'systemSetup', and 'managementGroup' tables. The 'user' table stands as the main reference, with the others interlinked through the foreign key 'userID'. This structure ensures efficient data retrieval and maintains the integrity of relationships across the data sets.

3.2.5 Data Visualization

The Excel dashboard functions as the final component of the data pipeline, built specifically for data visualization, analysis, and historical trend identification. By pulling data directly from AWS Aurora, the dashboard offers a unified, in-depth perspective on the data, empowering internal stakeholders with actionable insights derived from the telemetry.

Technical Details:

- Microsoft Excel: Excel is a powerful tool for data manipulation, analytics, and visualization, making it widely used in many industries. With its built-in features such as pivot tables, power query, statistical functions, and the developer option, Excel is a proficient tool at transforming complex data into accessible formats (Analytics Vidhya 2023). Its universal adoption in various industries was a deciding factor, ensuring that internal stakeholders can readily access, understand, and work with the data without requiring additional software or training.

- Macros, Power Query, and VBA: The dashboard leverages the power of Excel's Macros and Power Query for automated data retrieval, transformation, and loading operations. Additionally, VBA (Visual Basic for Applications) was employed to further customize the dashboard, enabling advanced analytics, data manipulations, and facilitating user interactions. This combination ensures a dynamic and interactive dashboard that is tailored to the specific needs of the company, offering both depth and flexibility in data analysis.

4 COMPONENT ANALYSIS AND IMPLEMENTATION

4.1 Metrics collection and JSON file structure

The Telemetry Module captures the data and records it onto the user's device in the form of a JSON file (see Figure 2). This file is organized into distinct arrays such as 'user', 'calibration', 'netDevice', 'performanceReport', 'systemSetup', with each array having its respective table schema in the database.

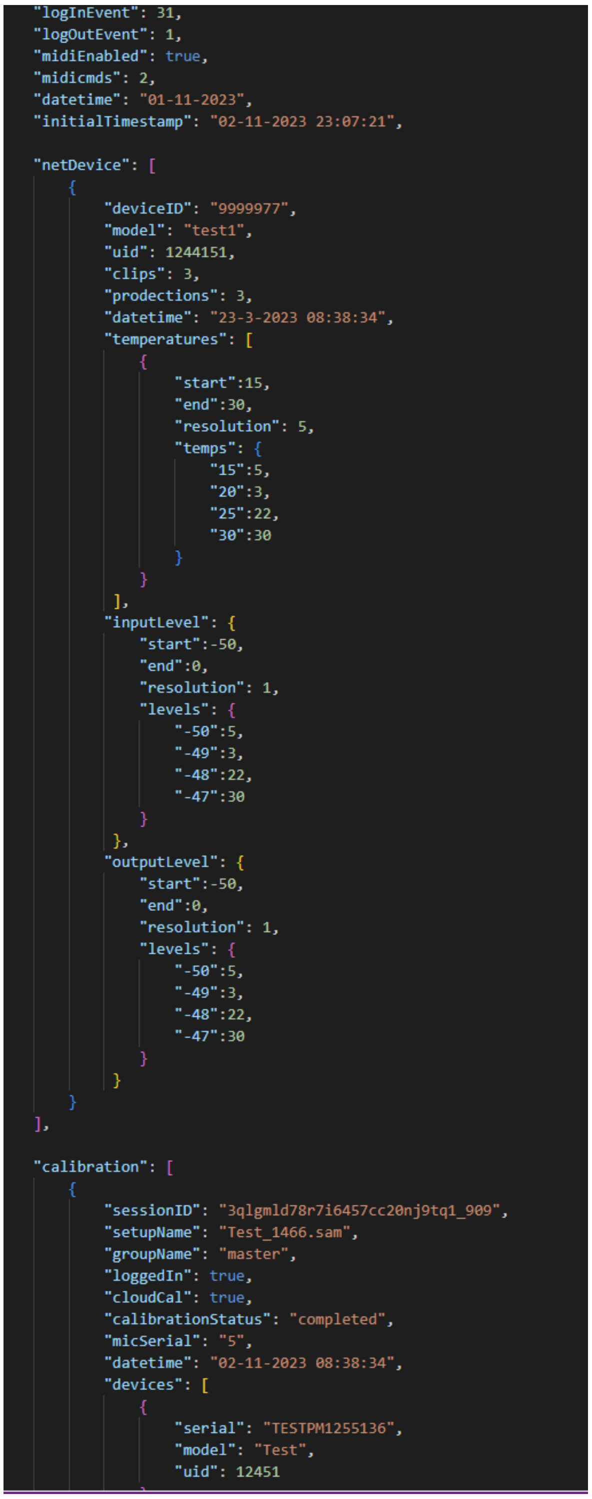
The "user" section provides insights into the user's profile and their interactions with the management software. This section carries unique identifiers for the user and the application, along with specifics about the version of the software used and the operating system. It also logs metrics like the total running time, the number of sessions initiated, setups configured, and various events, all timestamped to indicate their relevancy.

Next, is the "netDevice" segment, an array that captures details about the devices that have been connected or interacted with. Each entry of the device has its unique identifiers, model information, and metrics such as the number of clips and prodections. Additionally, it logs the readings related to temperature and audio levels within nested JSON arrays for each device, each marked with start and end values, resolutions, and specific values.

The "calibration" section is a detailed record of calibration activities undertaken. Each calibration instance has its unique session identifiers, coupled with the associated setup details and group affiliations. Here is defined the calibration status and related specifics. Furthermore, it lists the devices that were part of the calibration process and timestamps that signal the calibration event's occurrence.

The "performanceReport" segment provides a repository of the performance data. Each entry contains the relevant group details and the setup name under which the grading was executed. These entries also contain detailed information about the devices involved, all timestamped to mark the grading event.

Lastly, the "systemSetup" section logs data of the setup preferences and configurations users have employed in the management software. Within each setup, identifiers are paired with the name of the setup file. This section serves as a repository of audio settings, capturing aspects like the microphone reference level, power management preferences, and the mode of audio input. Additionally, it logs the groups associated with the setup. Each grouping outlines the name of the group, its preferred audio format, and its calibration status. Embedded within each group is an array of devices, revealing their participation, all marked with relevant timestamps.



```
    "logInEvent": 31,
    "logOutEvent": 1,
    "midiEnabled": true,
    "midicmds": 2,
    "datetime": "01-11-2023",
    "initialTimestamp": "02-11-2023 23:07:21",

    "netDevice": [
        {
            "deviceID": "9999977",
            "model": "test1",
            "uid": 1244151,
            "clips": 3,
            "prodections": 3,
            "datetime": "23-3-2023 08:38:34",
            "temperatures": [
                {
                    "start":15,
                    "end":30,
                    "resolution": 5,
                    "temps": {
                        "15":5,
                        "20":3,
                        "25":22,
                        "30":30
                    }
                }
            ],
            "inputLevel": {
                "start":-50,
                "end":0,
                "resolution": 1,
                "levels": {
                    "-50":5,
                    "-49":3,
                    "-48":22,
                    "-47":30
                }
            },
            "outputLevel": {
                "start":-50,
                "end":0,
                "resolution": 1,
                "levels": {
                    "-50":5,
                    "-49":3,
                    "-48":22,
                    "-47":30
                }
            }
        }
    ],

    "calibration": [
        {
            "sessionID": "3qlgmld78r7i6457cc20nj9tq1_909",
            "setupName": "Test_1466.sam",
            "groupName": "master",
            "loggedIn": true,
            "cloudCal": true,
            "calibrationStatus": "completed",
            "micSerial": "5",
            "datetime": "02-11-2023 08:38:34",
            "devices": [
                {
                    "serial": "TESTPM1255136",
                    "model": "Test",
                    "uid": 12451
```

Figure 2. Snippet of JSON file structure.

4.2 Telemetry Module

The Telemetry Module serves as an integral component of the system, dedicated to efficient data collection, storage, and transmission. The module is designed using the JUCE framework, ensuring portability and compatibility across various platforms.

4.2.1 User-Centric Data Management

A distinctive feature of the "TelemetryModule" is its user-centric approach to data management. The module handles telemetry data based on the "userID". For each user, is created locally a distinct JSON file. This file serves as the primary storage for the telemetry data of that specific user.

4.2.2 Multi-user Devices

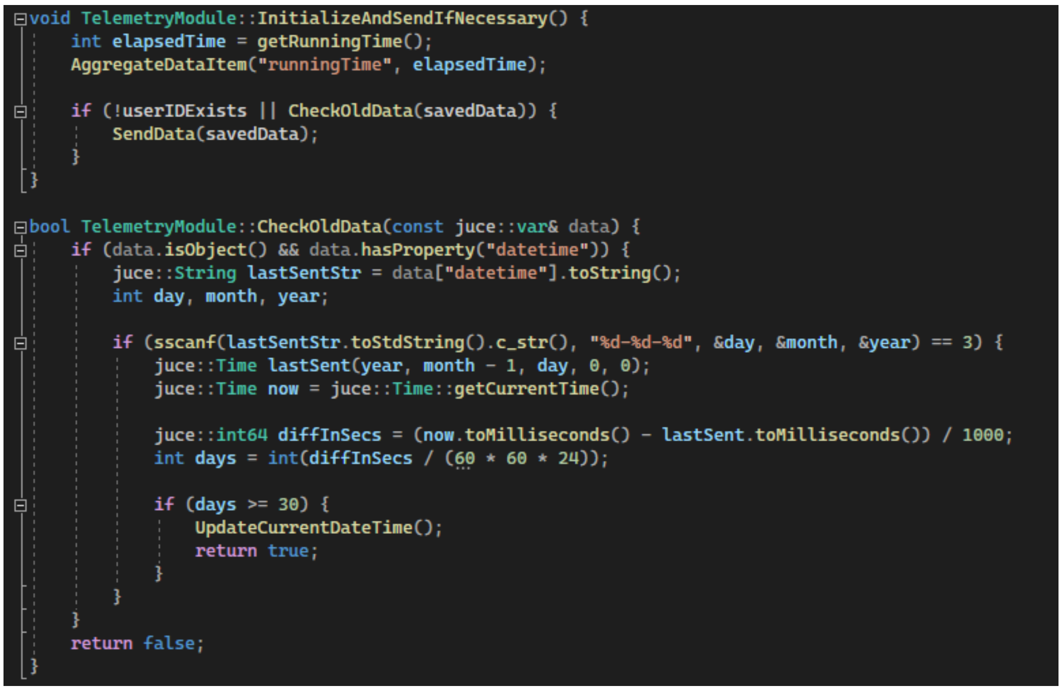
In environments where multiple users share a single device, the module ensures data integrity by associating telemetry data with individual userIDs. Thus, irrespective of the number of users on a device, each user's data is stored separately, guaranteeing no overlap or data loss.

4.2.3 Single-user Multi-device Operations

The module's design also supports the case where users access the system across multiple devices. Each device retains its unique JSON file for a given user, which, when sent to the cloud, populates distinct rows in the database. This structure ensures data granularity, allowing the system to log user interactions on different devices independently.

4.2.4 Timestamp Management for Data Integrity

The system uses two important timestamps: "initialTimestamp" and "datetime". The "initialTimestamp" is set once when the JSON file is first created to mark the start of data logging. The "datetime” is updated regularly with every data transfer to the cloud. This updating process helps to track the timing of each data upload, which is scheduled to happen every 30 days. This approach ensures the data is regularly updated and transmitted at set intervals, optimizing the use of system resources (see Figure 3).

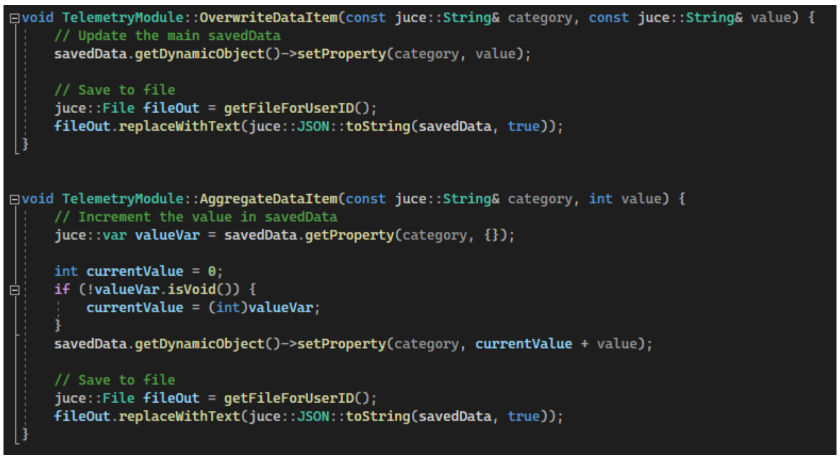


```cpp
void TelemetryModule::InitializeAndSendIfNecessary() {
    int elapsedTime = getRunningTime();
    AggregateDataItem("runningTime", elapsedTime);

    if (!userIDExists || CheckOldData(savedData)) {
        SendData(savedData);
    }
}

bool TelemetryModule::CheckOldData(const juce::var& data) {
    if (data.isObject() && data.hasProperty("datetime")) {
        juce::String lastSentStr = data["datetime"].toString();
        int day, month, year;

        if (sscanf(lastSentStr.toStdString().c_str(), "%d-%d-%d", &day, &month, &year) == 3) {
            juce::Time lastSent(year, month - 1, day, 0, 0);
            juce::Time now = juce::Time::getCurrentTime();

            juce::int64 diffInSecs = (now.toMilliseconds() - lastSent.toMilliseconds()) / 1000;
            int days = int(diffInSecs / (60 * 60 * 24));

            if (days >= 30) {
                UpdateCurrentDateTime();
                return true;
            }
        }
    }
    return false;
}
```

Figure 3. Snippet of telemetry module, checking the "datetime" interval and activation of the flag for data transmission.

4.2.5 Dynamic Data Management

Telemetry data is dynamic, often changing or accumulating over time. To manage this, the Telemetry Module includes two key functions: “OverwriteDataItem” for state-based data, which refreshes content with each new entry, and “AggregateDataItem” for event-based data that continually accumulates (see Figure 4). This approach enables the module to maintain an accurate representation of both instantaneous and evolving data aspects effectively.



```cpp
void TelemetryModule::OverwriteDataItem(const juce::String& category, const juce::String& value) {
    // Update the main savedData
    savedData.getDynamicObject()->setProperty(category, value);

    // Save to file
    juce::File fileOut = getFileForUserID();
    fileOut.replaceWithText(juce::JSON::toString(savedData, true));
}

void TelemetryModule::AggregateDataItem(const juce::String& category, int value) {
    // Increment the value in savedData
    juce::var valueVar = savedData.getProperty(category, {});

    int currentValue = 0;
    if (!valueVar.isVoid()) {
        currentValue = (int)valueVar;
    }
    savedData.getDynamicObject()->setProperty(category, currentValue + value);

    // Save to file
    juce::File fileOut = getFileForUserID();
    fileOut.replaceWithText(juce::JSON::toString(savedData, true));
}
```

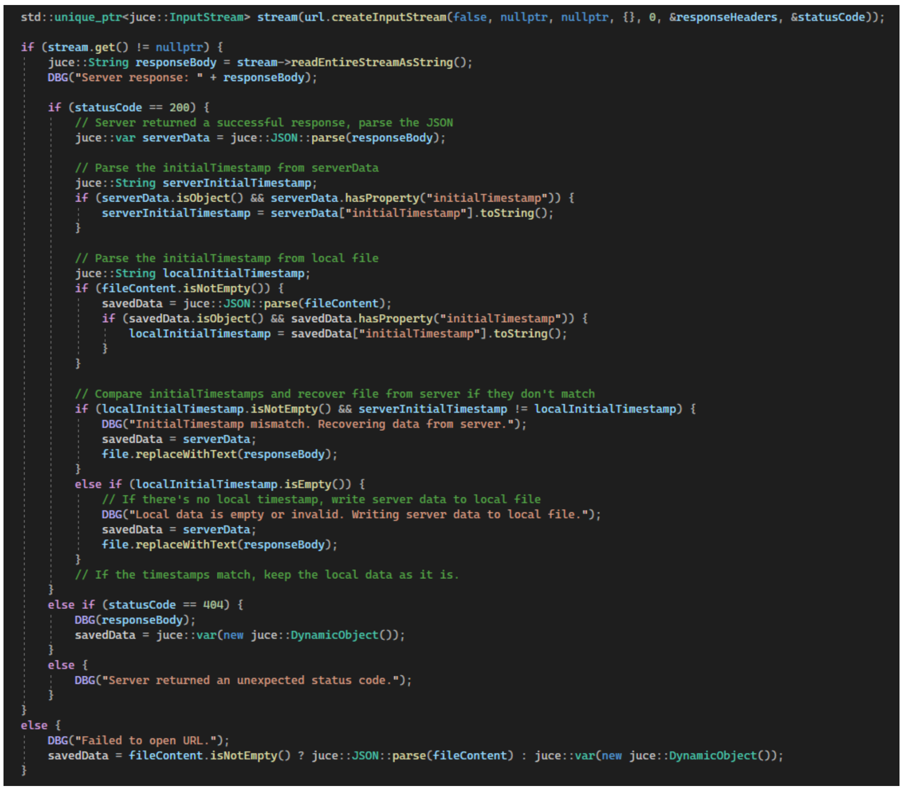
Figure 4. Snippet of telemetry module, “OverwriteDataItem” and “AggregateDataItem” functions.

4.2.6 Data Aggregation and Transmission

A critical function of the module is to periodically transmit the collected telemetry data to the cloud. The data is sent to the API Gateway using POST request, ensuring secure and efficient data transfer.

4.2.7 Data Integrity Measures

The telemetry module initiates a data integrity check at startup by sending a GET request to fetch user data from the database. It then compares the 'initialTimestamp' from the server with the local file’s timestamp to verify integrity. If a mismatch is detected or the key is missing, suggesting either file tampering or deletion, the module restores the file with the database copy before proceeding with additional logging. This measure ensures that the data remains consistent and trustworthy (see Figure 5).



```cpp
std::unique_ptr<juce::InputStream> stream(url.createInputStream(false, nullptr, nullptr, {}, 0, &responseHeaders, &statusCode));

if (stream.get() != nullptr) {
    juce::String responseBody = stream->readEntireStreamAsString();
    DBG("Server response: " + responseBody);

    if (statusCode == 200) {
        // Server returned a successful response, parse the JSON
        juce::var serverData = juce::JSON::parse(responseBody);

        // Parse the initialTimestamp from serverData
        juce::String serverInitialTimestamp;
        if (serverData.isObject() && serverData.hasProperty("initialTimestamp")) {
            serverInitialTimestamp = serverData["initialTimestamp"].toString();
        }

        // Parse the initialTimestamp from local file
        juce::String localInitialTimestamp;
        if (fileContent.isNotEmpty()) {
            savedData = juce::JSON::parse(fileContent);
            if (savedData.isObject() && savedData.hasProperty("initialTimestamp")) {
                localInitialTimestamp = savedData["initialTimestamp"].toString();
            }
        }

        // Compare initialTimestamps and recover file from server if they don't match
        if (localInitialTimestamp.isNotEmpty() && serverInitialTimestamp != localInitialTimestamp) {
            DBG("InitialTimestamp mismatch. Recovering data from server.");
            savedData = serverData;
            file.replaceWithText(responseBody);
        }
        else if (localInitialTimestamp.isEmpty()) {
            // If there's no local timestamp, write server data to local file
            DBG("Local data is empty or invalid. Writing server data to local file.");
            savedData = serverData;
            file.replaceWithText(responseBody);
        }
        // If the timestamps match, keep the local data as it is.
    }
    else if (statusCode == 404) {
        DBG(responseBody);
        savedData = juce::var(new juce::DynamicObject());
    }
    else {
        DBG("Server returned an unexpected status code.");
    }
}
else {
    DBG("Failed to open URL.");
    savedData = fileContent.isNotEmpty() ? juce::JSON::parse(fileContent) : juce::var(new juce::DynamicObject());
}
```

Figure 5. Snippet of telemetry module, parsing and comparing the "initialTimestamp".

4.2.8 Telemetry Module Flowchart

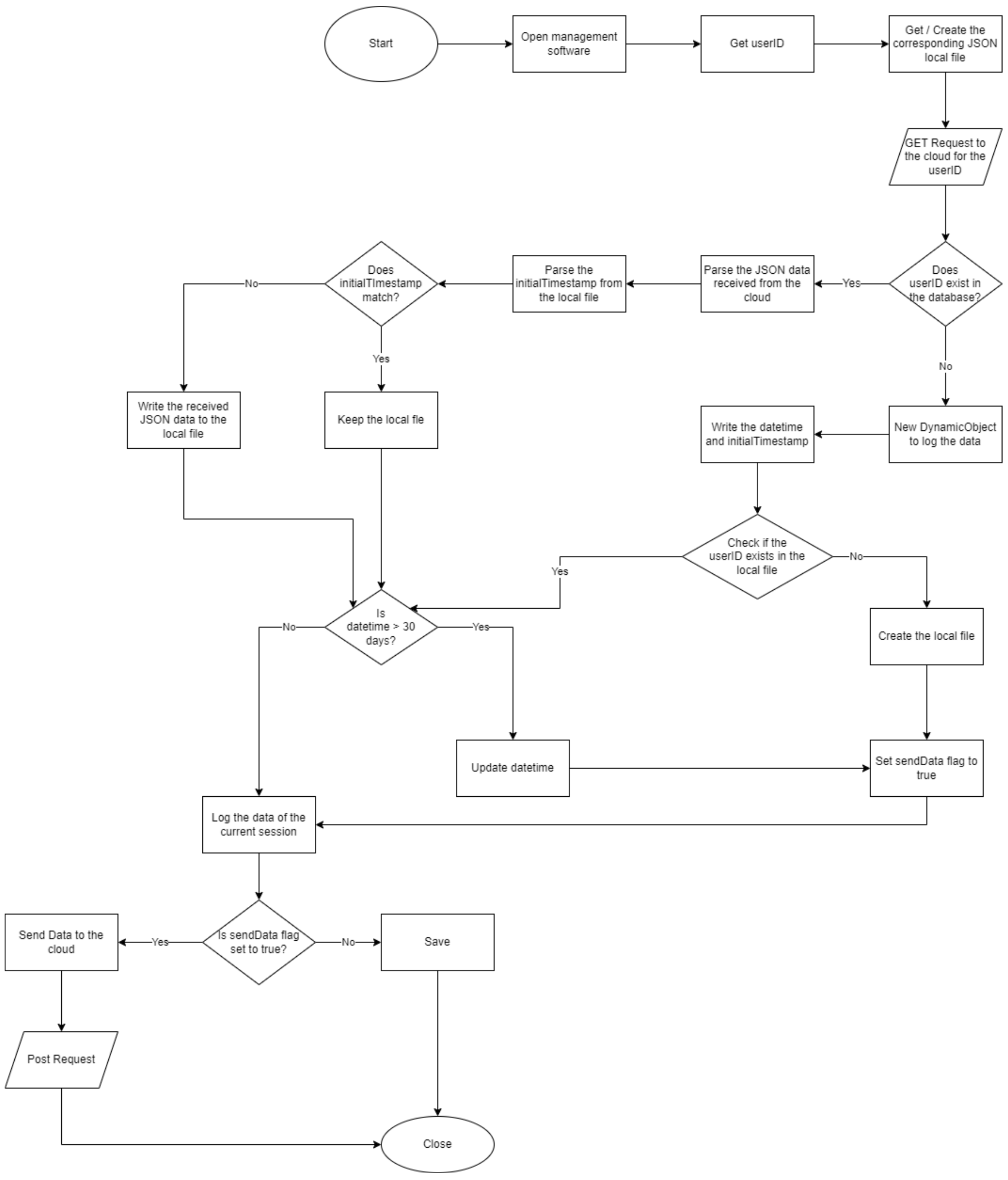

Start
Open management software
Get userID
Get / Create the corresponding JSON local file
GET Request to the cloud for the userID
Does userID exist in the database?
Yes
Parse the JSON data received from the cloud
Parse the initialTimestamp from the local file
Does initialTImestamp match?
No
Yes
Write the received JSON data to the local file
Keep the local fle
No
New DynamicObject to log the data
Write the datetime and initialTimestamp
Check if the userID exists in the local file
Yes
No
Create the local file
Is datetime > 30 days?
No
Yes
Update datetime
Set sendData flag to true
Log the data of the current session
Is sendData flag set to true?
Yes
No
Send Data to the cloud
Save
Post Request
Close

Figure 6. Telemetry Module Flowchart

4.3 Data Processing Application

The data processing application is pivotal in handling and maintaining data consistency, ensuring that the incoming data matches the expected schema of the database, and providing a structured method to process and save the data. Built using Python, the application is structured around various modules, each catering to a unique aspect of the data processing workflow (see Figure 7).

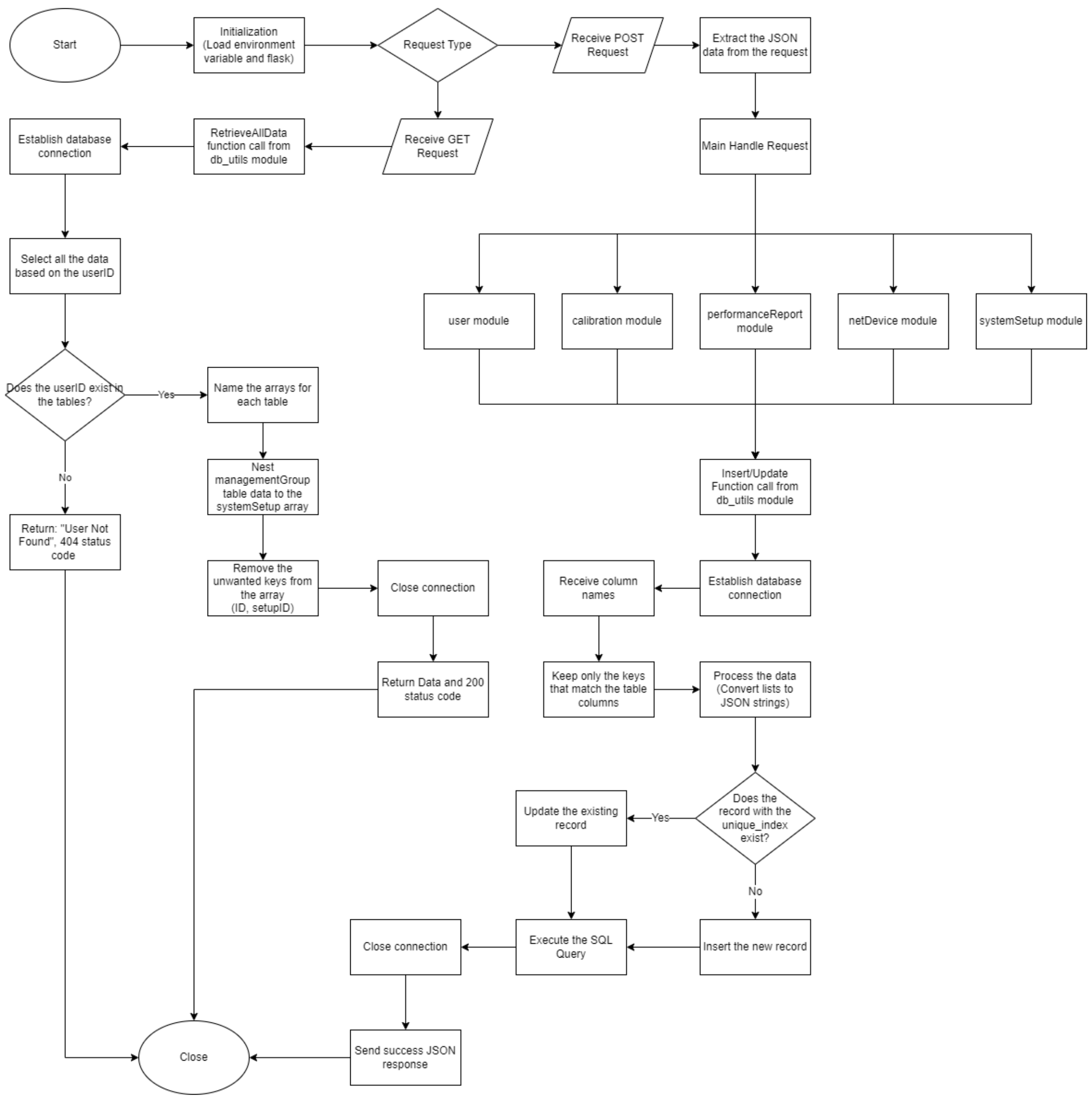

Start
Initialization (Load environment variable and flask)
Request Type
Receive POST Request
Extract the JSON data from the request
Establish database connection
RetrieveAllData function call from db_utils module
Receive GET Request
Main Handle Request
Select all the data based on the userID
user module
calibration module
performanceReport module
netDevice module
systemSetup module
Does the userID exist in the tables?
Yes
Name the arrays for each table
Nest managementGroup table data to the systemSetup array
No
Insert/Update Function call from db_utils module
Return: "User Not Found", 404 status code
Remove the unwanted keys from the array (ID, setupID)
Close connection
Receive column names
Establish database connection
Return Data and 200 status code
Keep only the keys that match the table columns
Process the data (Convert lists to JSON strings)
Update the existing record
Yes
Does the record with the unique_index exist?
No
Close connection
Execute the SQL Query
Insert the new record
Close
Send success JSON response

Figure 7. Data Processing Application Flowchart

4.3.1 Modular Design

The architecture of the application is based on a modular design approach. This division into distinct modules optimizes clarity, scalability, and maintainability. Such a structure not only facilitates easier debugging and enhancements but also promotes the principle of single responsibility, where each module addresses a specific functionality.

4.3.1.1 Main.Py Module

The main python file serves as the central interaction point for the entire application, orchestrating how incoming POST requests are processed by initiating the data processing pipeline and coordinating with the other modules for specialized tasks.

Key Components:

- Flask Integration:

The application employs Flask, a micro web framework for Python, providing a foundation for web servers to process HTTP requests. (Python Basics.)

- Module Integrations:

  The application draws functionalities from various imported modules, including “user”, “net_device”, “calibration”, “performanceReport”, and “setup”. These modules handle the specifics of processing the incoming data in accordance with their respective table schemas or functionalities.

- Database Configuration:

  The app retrieves its database configurations from the environment variables, ensuring dynamic and secure setup.

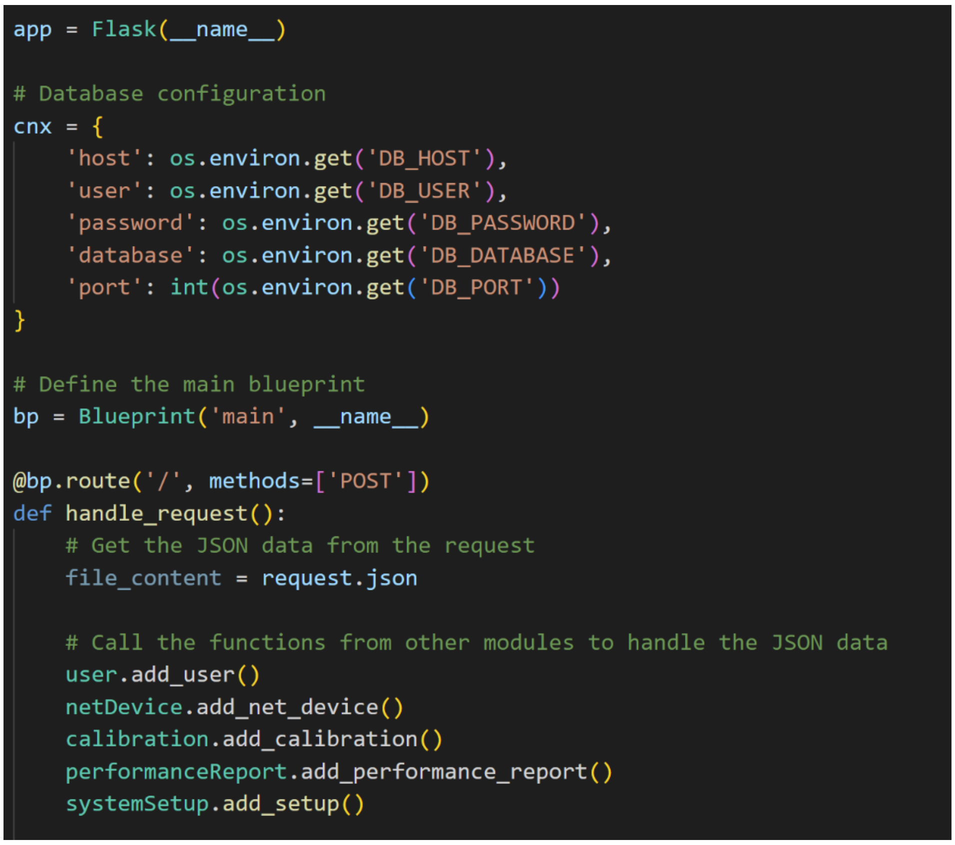
- Main Blueprint Definition:

  Using Flask's modular design, a blueprint named 'main' is defined. This encapsulates the core request-handling route.

- Requests Handling:

  POST request (see Figure 8):

  1. Extracts the JSON payload.
  2. Delegates to the corresponding modules for data processing.
  3. Returns a successful JSON response.

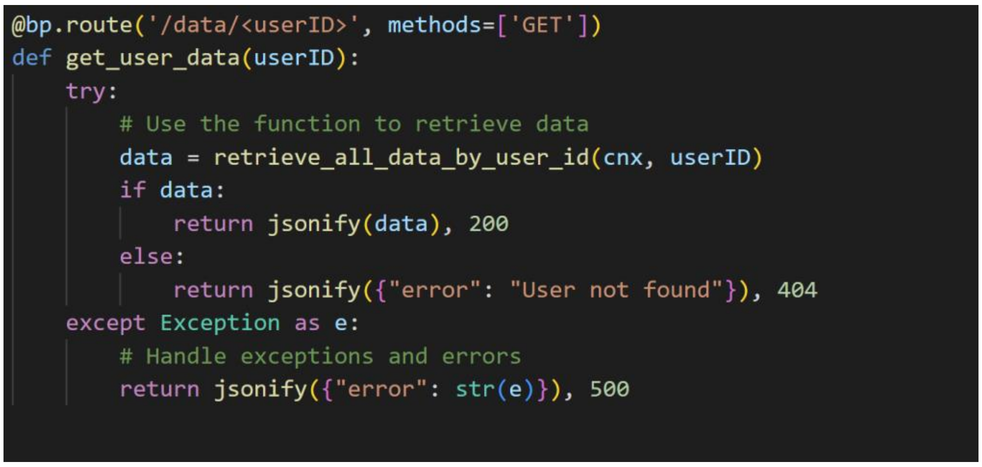


```python
app = Flask(__name__)

# Database configuration
cnx = {
    'host': os.environ.get('DB_HOST'),
    'user': os.environ.get('DB_USER'),
    'password': os.environ.get('DB_PASSWORD'),
    'database': os.environ.get('DB_DATABASE'),
    'port': int(os.environ.get('DB_PORT'))
}

# Define the main blueprint
bp = Blueprint('main', __name__)

@bp.route('/', methods=['POST'])
def handle_request():
    # Get the JSON data from the request
    file_content = request.json

    # Call the functions from other modules to handle the JSON data
    user.add_user()
    netDevice.add_net_device()
    calibration.add_calibration()
    performanceReport.add_performance_report()
    systemSetup.add_setup()
```

Figure 8. Snippet of Main.py, Database configuration and POST request part.

GET Request (see Figure 9):

1. The endpoint /data/<userID> activates upon a GET request, leveraging the function from the "db_utils" module.
2. It fetches the user's data and returns it in JSON format.



```python
@bp.route('/data/<userID>', methods=['GET'])
def get_user_data(userID):
    try:
        # Use the function to retrieve data
        data = retrieve_all_data_by_user_id(cnx, userID)
        if data:
            return jsonify(data), 200
        else:
            return jsonify({"error": "User not found"}), 404
    except Exception as e:
        # Handle exceptions and errors
        return jsonify({"error": str(e)}), 500
```

Figure 9. Snippet of Main.py, GET request part.

- Application Initialization:

  The Flask server starts, set to run on all available network interfaces and listening on port 5000.

4.3.1.2 db_utils.py Module

The db_utils.py module is the center of the data processing application when it comes to direct interactions with the database. It has the fundamental functions required for establishing a connection, fetching, and transforming data, and executing core tasks such as data insertion and updates.

Module Explanation:

- Database Configuration:

  The module starts by loading the environment variables and setting up the configuration for the database connection.

- get_database_connection Function:

  This function encapsulates the logic to establish a connection with the MySQL database. By doing so, it ensures that any module requiring database interaction can effortlessly retrieve an active connection.

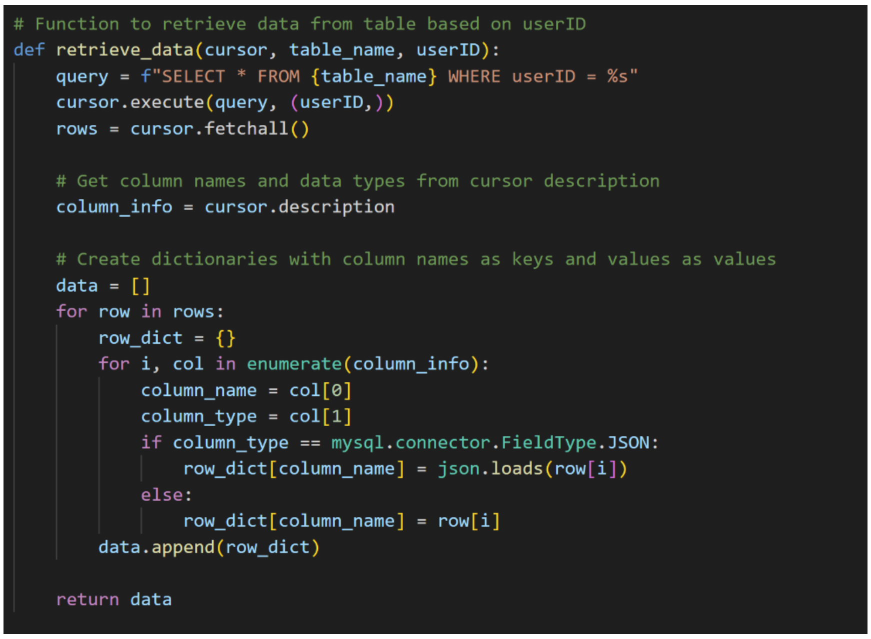
- retrieve_all_data_by_user_id Function:

  This function stands central to the module's data retrieval capabilities. When called, it establishes a database connection and sequentially gathers data across multiple tables for a given user, identified by userID. Each table's data is fetched using the "retrieve_data" helper function, which extracts all relevant records and their associated fields (see Figure 10).

The function begins by collecting data from the primary table, 'user'. Subsequently, it iterates through the other related tables defined in a mapping structure (see Figure 11).

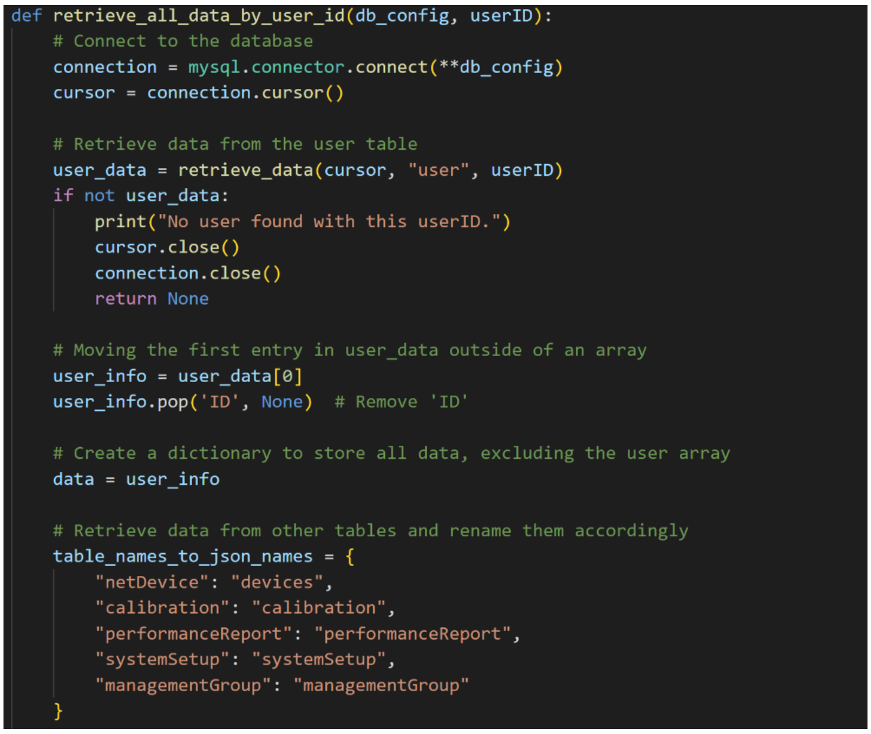
For tables that are interrelated, such as 'systemSetup' and 'managementGroup', the function performs additional steps to associate groups with their respective setup records, ensuring that the data structure mirrors the relational integrity present in the database (see Figure 12).

Once the data is retrieved from all the tables, the function closes the database connection and returns a dictionary of the user's complete dataset.



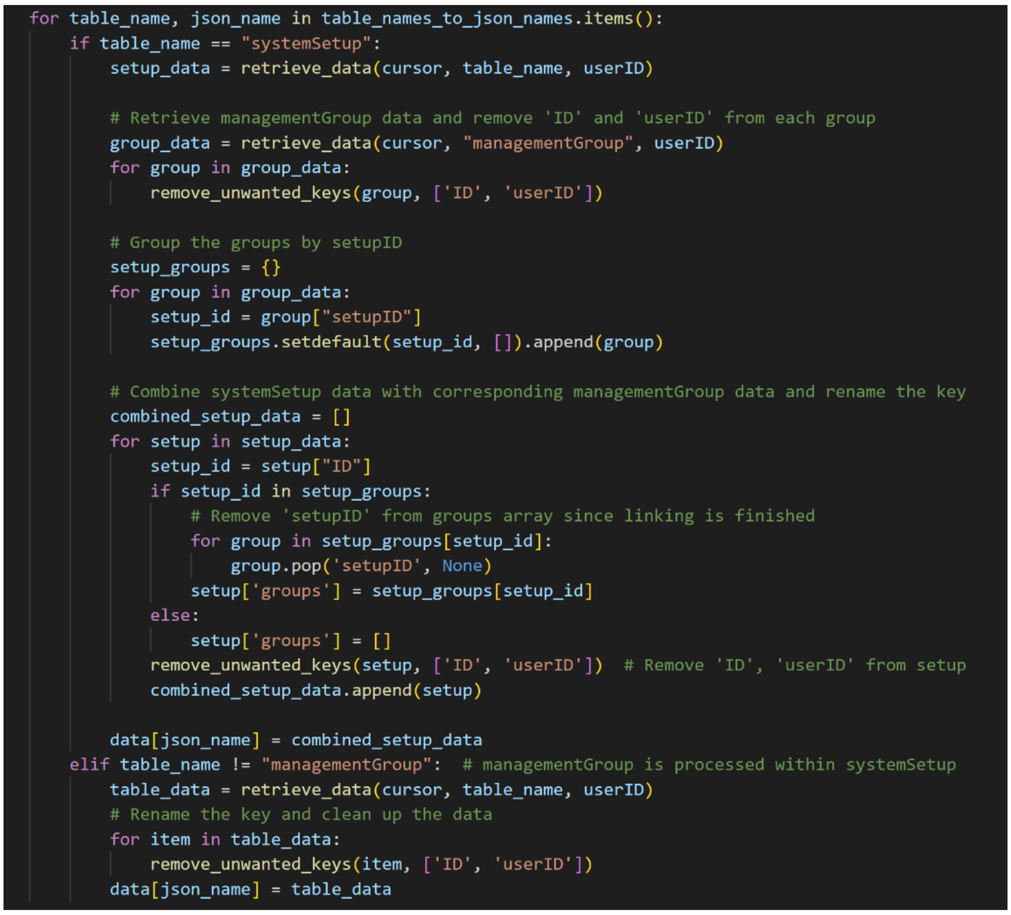
```python
# Function to retrieve data from table based on userID
def retrieve_data(cursor, table_name, userID):
    query = f"SELECT * FROM {table_name} WHERE userID = %s"
    cursor.execute(query, (userID,))
    rows = cursor.fetchall()

    # Get column names and data types from cursor description
    column_info = cursor.description

    # Create dictionaries with column names as keys and values as values
    data = []
    for row in rows:
        row_dict = {}
        for i, col in enumerate(column_info):
            column_name = col[0]
            column_type = col[1]
            if column_type == mysql.connector.FieldType.JSON:
                row_dict[column_name] = json.loads(row[i])
            else:
                row_dict[column_name] = row[i]
        data.append(row_dict)

    return data
```

Figure 10. Snippet from db_utils.py, “retrieve_data” helper function.



```python
def retrieve_all_data_by_user_id(db_config, userID):
    # Connect to the database
    connection = mysql.connector.connect(**db_config)
    cursor = connection.cursor()

    # Retrieve data from the user table
    user_data = retrieve_data(cursor, "user", userID)
    if not user_data:
        print("No user found with this userID.")
        cursor.close()
        connection.close()
        return None

    # Moving the first entry in user_data outside of an array
    user_info = user_data[0]
    user_info.pop('ID', None)  # Remove 'ID'

    # Create a dictionary to store all data, excluding the user array
    data = user_info

    # Retrieve data from other tables and rename them accordingly
    table_names_to_json_names = {
        "netDevice": "devices",
        "calibration": "calibration",
        "performanceReport": "performanceReport",
        "systemSetup": "systemSetup",
        "managementGroup": "managementGroup"
    }
```

Figure 11. Snippet from db_utils.py, "retrieve_all_data_by_user_id" function.

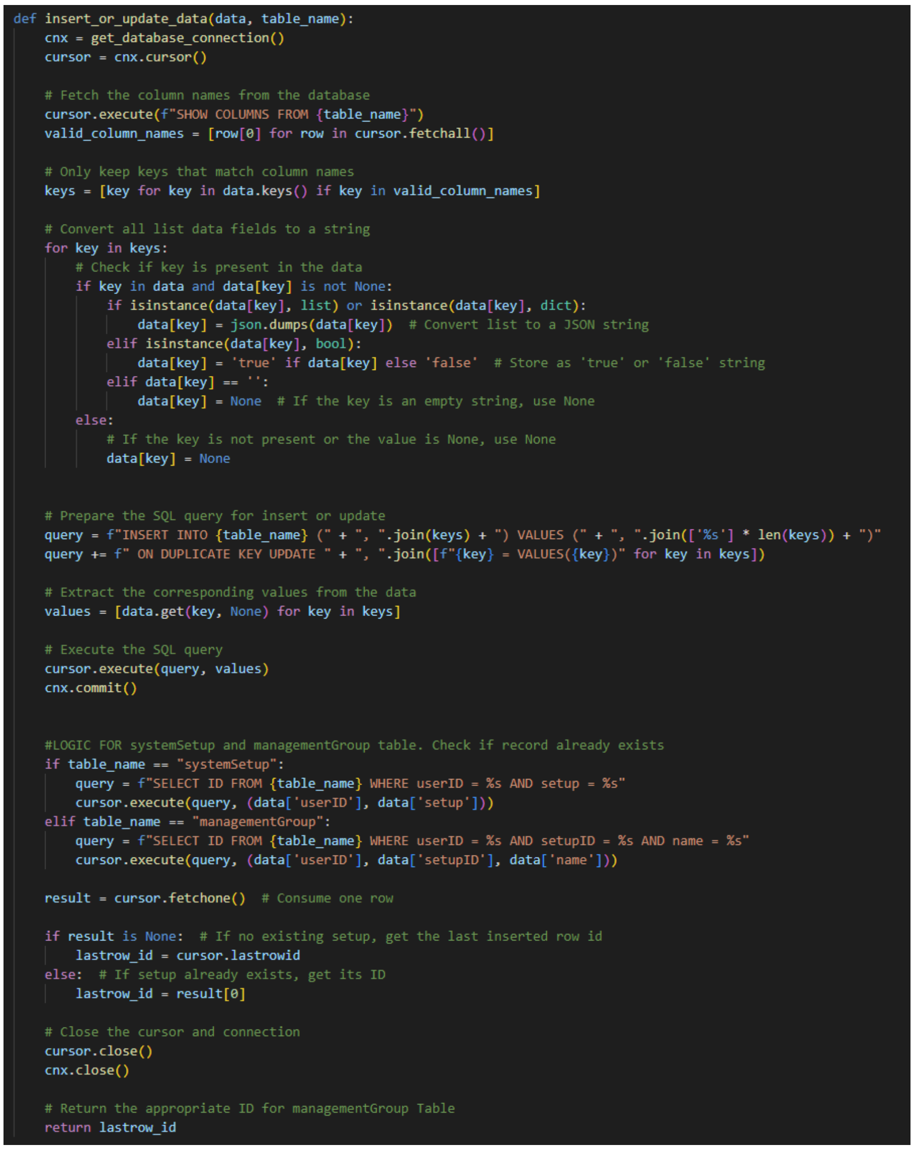


```python
for table_name, json_name in table_names_to_json_names.items():
    if table_name == "systemSetup":
        setup_data = retrieve_data(cursor, table_name, userID)

        # Retrieve managementGroup data and remove 'ID' and 'userID' from each group
        group_data = retrieve_data(cursor, "managementGroup", userID)
        for group in group_data:
            remove_unwanted_keys(group, ['ID', 'userID'])

        # Group the groups by setupID
        setup_groups = {}
        for group in group_data:
            setup_id = group["setupID"]
            setup_groups.setdefault(setup_id, []).append(group)

        # Combine systemSetup data with corresponding managementGroup data and rename the key
        combined_setup_data = []
        for setup in setup_data:
            setup_id = setup["ID"]
            if setup_id in setup_groups:
                # Remove 'setupID' from groups array since linking is finished
                for group in setup_groups[setup_id]:
                    group.pop('setupID', None)
                setup['groups'] = setup_groups[setup_id]
            else:
                setup['groups'] = []
            remove_unwanted_keys(setup, ['ID', 'userID'])  # Remove 'ID', 'userID' from setup
            combined_setup_data.append(setup)

        data[json_name] = combined_setup_data
    elif table_name != "managementGroup":  # managementGroup is processed within systemSetup
        table_data = retrieve_data(cursor, table_name, userID)
        # Rename the key and clean up the data
        for item in table_data:
            remove_unwanted_keys(item, ['ID', 'userID'])
        data[json_name] = table_data
```

Figure 12. Snippet from db_utils.py, "retrieve_all_data_by_user_id" function, appending the corresponding “managementGroup” inside the “systemSetup” array.

- insert_or_update_data Function:

  This function is designed to handle the core ETL (Extract, Transform, Load) task. It takes the data and either inserts it as a new record or updates an existing record in the database (see Figure 13). The function starts by fetching the valid column names for a given table to ensure data consistency. Data transformations take place next. Lists or dictionaries are converted into JSON strings, Booleans are translated to 'true' or 'false' strings, and absent or empty keys are set to None. Once the data is prepared, the function constructs an SQL query. The query's design uses the ON DUPLICATE KEY UPDATE clause, a MySQL-specific feature. This allows for the simultaneous handling of insertions (for new records) and updates (for existing records). Lastly, the SQL query is executed, and the operation's success is determined by whether it commits without errors.



```python
def insert_or_update_data(data, table_name):
    cnx = get_database_connection()
    cursor = cnx.cursor()

    # Fetch the column names from the database
    cursor.execute(f"SHOW COLUMNS FROM {table_name}")
    valid_column_names = [row[0] for row in cursor.fetchall()]

    # Only keep keys that match column names
    keys = [key for key in data.keys() if key in valid_column_names]

    # Convert all list data fields to a string
    for key in keys:
        # Check if key is present in the data
        if key in data and data[key] is not None:
            if isinstance(data[key], list) or isinstance(data[key], dict):
                data[key] = json.dumps(data[key])  # Convert list to a JSON string
            elif isinstance(data[key], bool):
                data[key] = 'true' if data[key] else 'false'  # Store as 'true' or 'false' string
            elif data[key] == '':
                data[key] = None  # If the key is an empty string, use None
        else:
            # If the key is not present or the value is None, use None
            data[key] = None


    # Prepare the SQL query for insert or update
    query = f"INSERT INTO {table_name} (" + ", ".join(keys) + ") VALUES (" + ", ".join(['%s'] * len(keys)) + ")"
    query += f" ON DUPLICATE KEY UPDATE " + ", ".join([f"{key} = VALUES({key})" for key in keys])

    # Extract the corresponding values from the data
    values = [data.get(key, None) for key in keys]

    # Execute the SQL query
    cursor.execute(query, values)
    cnx.commit()


    #LOGIC FOR systemSetup and managementGroup table. Check if record already exists
    if table_name == "systemSetup":
        query = f"SELECT ID FROM {table_name} WHERE userID = %s AND setup = %s"
        cursor.execute(query, (data['userID'], data['setup']))
    elif table_name == "managementGroup":
        query = f"SELECT ID FROM {table_name} WHERE userID = %s AND setupID = %s AND name = %s"
        cursor.execute(query, (data['userID'], data['setupID'], data['name']))

    result = cursor.fetchone()  # Consume one row

    if result is None:  # If no existing setup, get the last inserted row id
        lastrow_id = cursor.lastrowid
    else:  # If setup already exists, get its ID
        lastrow_id = result[0]

    # Close the cursor and connection
    cursor.close()
    cnx.close()

    # Return the appropriate ID for managementGroup Table
    return lastrow_id
```

Figure 13. Snippet of Insert / Update Operation.

4.3.1.3 user.py, calibration.py, netDevice.py, performanceReport.py, systemSetup.py Modules

These modules are designed to handle operations specific to their respective database tables. Although their primary objective is similar, i.e., to interact with a certain table, they are equipped to process data in a manner tailored to the individual requirements of their tables. For simplicity and clarity, it will be explained the user.py module, which can serve as a representative example for the other modules too.

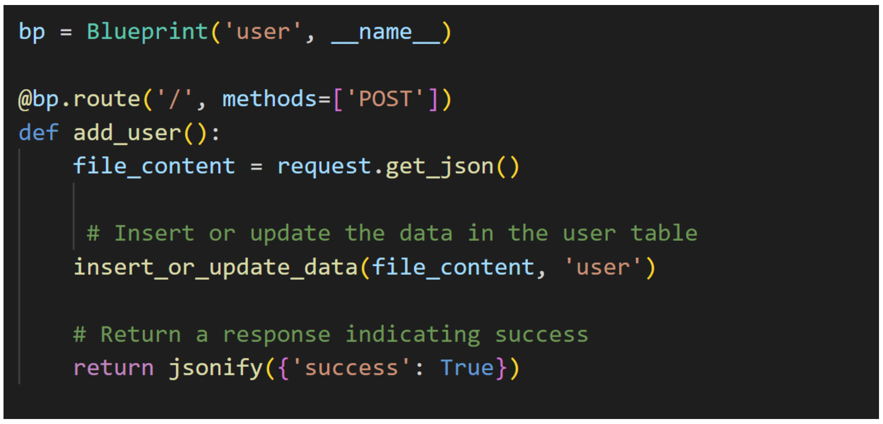
Module Explanation:

- Blueprint and Initial Setup:

  A Flask blueprint named user is instantiated, allowing the module to define its own routes and handlers.

- add_user Function (see Figure 14):

  This route, activated with a POST request, is responsible for adding or updating user data in the user table. The function calls the function from the "db_utils" module and either inserts the data as a new record or updates the existing one in the user table. At the end, a successful response is returned upon completion of the operation.



```
bp = Blueprint('user', __name__)

@bp.route('/', methods=['POST'])
def add_user():
    file_content = request.get_json()

     # Insert or update the data in the user table
    insert_or_update_data(file_content, 'user')

    # Return a response indicating success
    return jsonify({'success': True})
```

Figure 14. Snippet from "user.py" module

4.3.2 Secure and Flexible Database Operations

4.3.2.1 Parameterized Queries

Parameterized queries provide a mechanism to execute SQL commands efficiently and securely. Instead of embedding user data directly into the SQL string, parameterized queries use placeholders. Actual data values are then supplied separately, ensuring they are treated as data and not as executable SQL code. This distinction is crucial in defending against SQL injection attacks. (SQLShack 2022.)

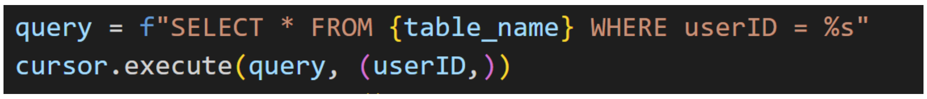
SQL Injection Attacks:

In a typical SQL injection scenario, a malicious actor can provide specially crafted input data that, when embedded into an SQL command, can alter the command's semantics. This alteration can allow unauthorized reading, modification, or even deletion of data. For instance, an attacker might input "user; DROP TABLE user; --" as part of a field. If this input is directly embedded into an SQL command without proper sanitization, it can lead to the deletion of the "user" table, causing significant data loss and system disruption.

How Parameterized Queries Help:

Parameterized queries ensure that user data is never directly interpolated into the SQL string. Instead, the SQL engine treats them strictly as data and not executable code. This behavior effectively neutralizes the SQL injection vector since the input data does not alter the SQL command's structure. (PYnative 2021.)

From the "db_utils.py" module:



```
query = f"SELECT * FROM {table_name} WHERE userID = %s"
cursor.execute(query, (userID,))
```

Figure 15. Parameterized Queries Snippet

In the above code:

- The “%s” in the query string serves as a placeholder indicating where the user data should be inserted.
- “(userID,)” is a tuple that supplies the actual data value for the “userID”.
- Rather than constructing a SQL string directly with the “userID”, the execute method of the cursor object takes care of substituting the placeholder (%s) with the data value (userID) in a safe manner. This approach helps prevent SQL injection attacks by ensuring that the userID is treated as a parameter, not a part of the SQL command itself.

Conclusion:

Using parameterized queries is a great practice in developing secure database-driven applications. The approach prevents malicious data manipulation, ensuring the integrity and security of the data and the system as a whole.

4.3.2.2 Flexible Operations

- Unique indices form the foundation of database inserts. These indices ensure data uniqueness and integrity.
- The system sidesteps hardcoding column names. By dynamically matching JSON keys with database columns, it paves the way for effortless scalability. Whether it is a new column in the database or an updated key in the JSON, the application seamlessly accommodates them.

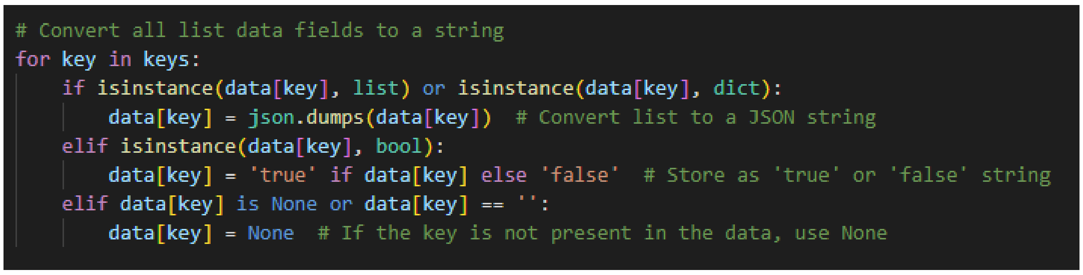
4.3.3 ETL Principles and Data Transformation

The ETL (Extract, Transform, Load) process plays an integral role in the data integration strategy of many organizations. By facilitating the migration of data from one environment to another, it ensures that data remains cohesive, reliable, and primed for analysis. In the context of this application, the ETL principles are meticulously embedded to harmonize and standardize data flow from diverse JSON files into the database. (Talend.)

- Extract: Data is pulled from the voluminous incoming JSON files, parsed, and prepped for processing.

- Transform: Before being fitted for database insertion, the data often requires reshaping. Whether it is lists converting into JSON strings or Booleans being cast to string representations, the transformation phase ensures data compatibility with the database schema (see Figure 16).

- Load: The final act, the processed data finds its rightful place in the database, ensuring that insights and analytics can be derived from it in subsequent stages.



```python
# Convert all list data fields to a string
for key in keys:
    if isinstance(data[key], list) or isinstance(data[key], dict):
        data[key] = json.dumps(data[key])  # Convert list to a JSON string
    elif isinstance(data[key], bool):
        data[key] = 'true' if data[key] else 'false'  # Store as 'true' or 'false' string
    elif data[key] is None or data[key] == '':
        data[key] = None  # If the key is not present in the data, use None
```

Figure 16. Example of the transformation process

4.3.4 Seamless Deployment and Infrastructure Integration

In the realm of modern software development, a holistic approach to application deployment and infrastructure integration is very important. This ensures reliability, scalability, and most importantly, reproducibility across varied environments.

- Dockerization

  One of the application's deployment strategies is Dockerization. By encapsulating the application and its dependencies within Docker containers, the system ensures environmental immutability. This encapsulation guarantees that regardless of where the application is deployed it behaves with consistent predictability. (Docker.)

- GitLab

  Version control is essential in today's collaborative software development landscape. GitLab serves as the version control platform for this application, offering a cohesive environment for code management, collaborative development, and most critically, automation via CI/CD (Continuous Integration/Continuous Deployment) pipeline. (GitLab.)

  The CI/CD pipeline has to main stages (see Figure 17):

  1. Build: This stage focuses on constructing the Docker image for the application. It leverages Kaniko, a tool designed to build container images from a Dockerfile, without a full-fledged Docker daemon. The constructed image is then pushed to Harbor, the designated container repository.
  2. Deploy Stage: Post image construction, the deployment stage comes into play. Using "kubectl", the command-line tool for Kubernetes, the application's image is updated in a Kubernetes deployment. This ensures that the newest version of the application is readily available for use.

- Harbor and AWS EKS

  The deployment ecosystem operates synergistically with Harbor and AWS EKS at its core. Harbor acts as the central container registry. More than just a repository, Harbor ensures that Docker images are not only stored but are also subject to rigorous security checks and compliance measures. (Harbor.)

  Once an image has been pushed to Harbor, AWS EKS, Amazon's managed Kubernetes service pulls the Docker image directly from Harbor. Through EKS, the application is deployed into a Kubernetes environment that is optimized for high availability and seamless scalability.

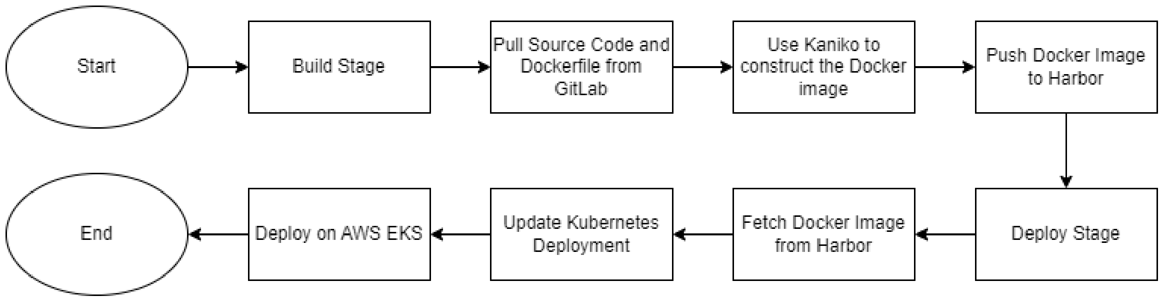

Start
Build Stage
Pull Source Code and Dockerfile from GitLab
Use Kaniko to construct the Docker image
Push Docker Image to Harbor
Deploy Stage
Fetch Docker Image from Harbor
Update Kubernetes Deployment
Deploy on AWS EKS
End

Figure 17. CI/CD Pipeline Flowchart

4.3.5 Future-Proofing and Scalability

The design decisions, from modular architecture to dynamic database interactions, highlight the application's readiness for future expansions. As data evolves and grows, the system's inherent flexibility ensures that it can adapt without necessitating major overhauls.

4.3.6 Conclusion

The data processing application is not just a tool; it is a symphony of well-orchestrated modules, algorithms, and design principles. Whether it is handling new data types, integrating with modern infrastructure tools, or scaling to handle more significant data loads, the application stands prepared and robust.

4.4 Database Architecture and Implementation

In today's era of big data, the foundation of any effective data processing application is a robust, scalable, and well-designed database. In this section, will be covered the architecture and implementation of the database.

4.4.1 Selection Criteria for the Database System

The database's underlayer is powered by AWS Aurora. AWS Aurora, a fully managed relational database service, matches the prowess of high-end commercial databases with the affordability and simplicity of open-source ones. It naturally befits MySQL, which was selected for its remarkable speed, reliability, and user-friendliness. Aurora supercharges MySQL by adding:

- High Performance and Scalability: Aurora is known to deliver up to five times the performance of a standard MySQL database. It scales automatically, with the ability to handle tens of terabytes of data.
- High Availability and Durability: Aurora continually backs up data and transparently recovers from physical storage failures; it is fault-tolerant by design.

4.4.2 Schema Blueprint

In an RDBMS (Relational Database Management System), the schema serves as a foundational blueprint. It defines the tables, relationships between them, fields, and indexes. (Ramos 2022.)

Tables and their relationships (see Figure 18):

- "user": This serves as the primary table, where "userID" is the primary key.
- "calibration", "performanceReport", "netDevice", "systemSetup", "managementGroup": These tables are connected to the "user" table through the foreign key "userID". This bond ensures that data from these tables can always be related back to a specific user.
- "managementGroup" and "systemSetup" Association: The "managementGroup" table has an additional layer of relationship with the "systemSetup" table. This connection is established through the foreign key "setupID" in "managementGroup", which corresponds to the "ID" column, the primary key in the "systemSetup" table. This connection enables a group to be associated with a specific setup.

4.4.3 Indexing

One of the fundamental aspects of the database architecture is the use of indexing. This database design, through the application of both unique and composite indices, ensures data integrity and uniqueness, and at the same time, it accelerates data retrieval processes. (MySQL.)

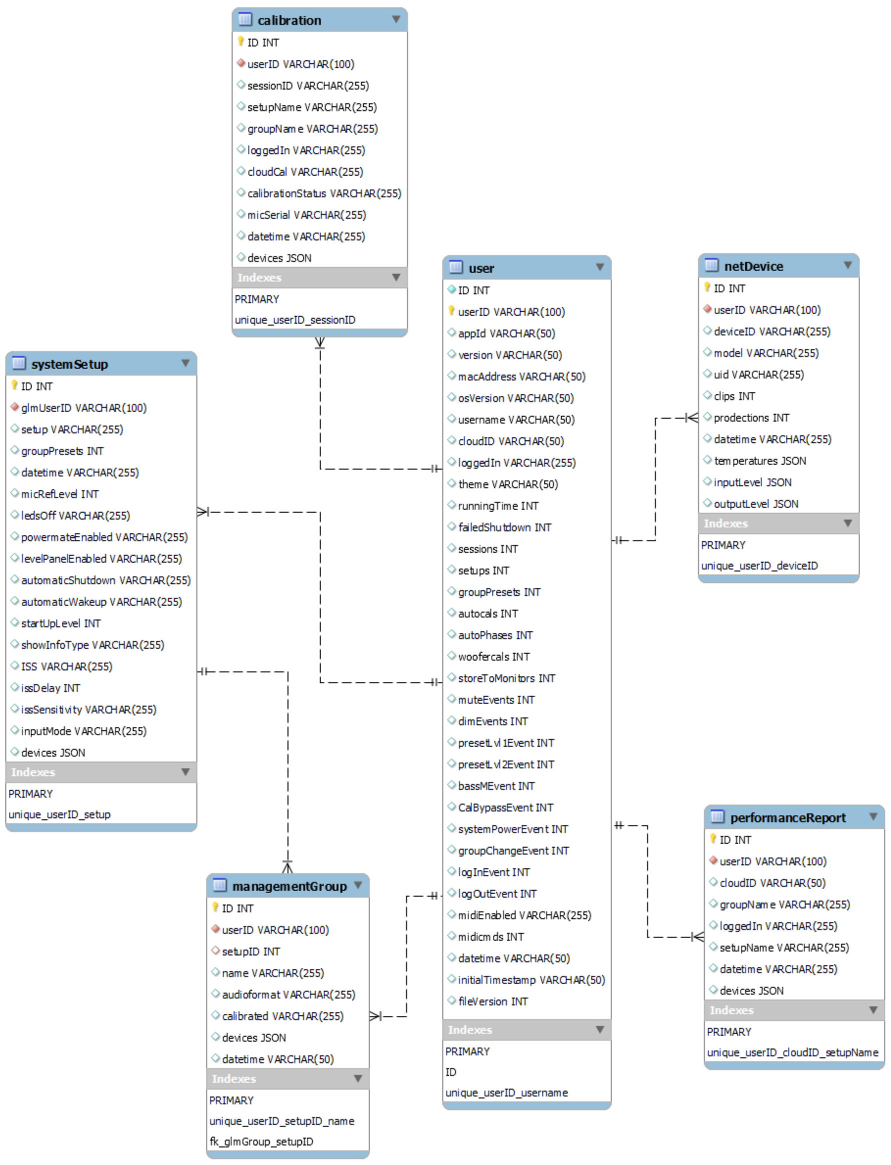


Figure 18. Database Architecture

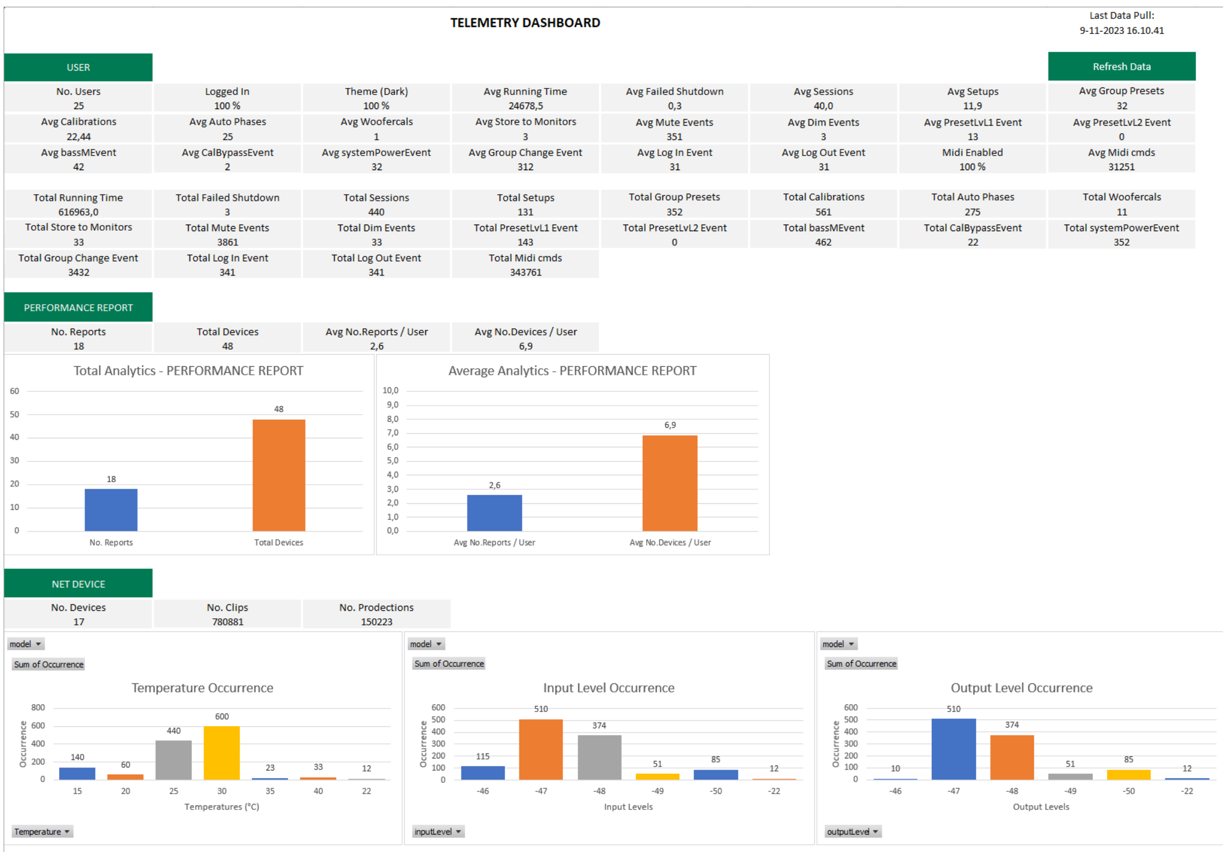
4.5 Excel Dashboard

4.5.1 Overview

The Excel Dashboard serves as the analytical interface for the telemetry data. Excel's in-built features such as Power Query enable seamless integration with the database. To organize the telemetry data in an easily accessible format, each table schema from the database is represented in its dedicated worksheet. Beyond data representation, a comprehensive sheet named “Dashboard” exists, facilitating various calculations corresponding to each schema, such as “USER”, “PERFORMANCE REPORT”, “NET DEVICE”, “CALIBRATION”, “SYSTEM SETUP”, and “MANAGEMENT GROUP” (see Figure 19).

4.5.2 Data Representation and Analysis

The core utility of the dashboard lies in its analytical features. For every section, essential calculations such as sums and averages are computed. As an instance, metrics like total number of users, average running time per user, total sessions, total setups, and more are readily available.


TELEMETRY DASHBOARD

Last Data Pull:
9-11-2023 16.10.41

USER

Refresh Data

No. Users	Logged In	Theme (Dark)	Avg Running Time	Avg Failed Shutdown	Avg Sessions	Avg Setups	Avg Group Presets
25	100 %	100 %	24678,5	0,3	40,0	11,9	32
Avg Calibrations	Avg Auto Phases	Avg Woofercals	Avg Store to Monitors	Avg Mute Events	Avg Dim Events	Avg PresetLvL1 Event	Avg PresetLvL2 Event
22,44	25	1	3	351	3	13	0
Avg bassMEvent	Avg CalBypassEvent	Avg systemPowerEvent	Avg Group Change Event	Avg Log In Event	Avg Log Out Event	Midi Enabled	Avg Midi cmds
42	2	32	312	31	31	100 %	31251

Total Running Time	Total Failed Shutdown	Total Sessions	Total Setups	Total Group Presets	Total Calibrations	Total Auto Phases	Total Woofercals
616963,0	3	440	131	352	561	275	11
Total Store to Monitors	Total Mute Events	Total Dim Events	Total PresetLvL1 Event	Total PresetLvL2 Event	Total bassMEvent	Total CalBypassEvent	Total systemPowerEvent
33	3861	33	143	0	462	22	352
Total Group Change Event	Total Log In Event	Total Log Out Event	Total Midi cmds				
3432	341	341	343761				

PERFORMANCE REPORT

No. Reports	Total Devices	Avg No.Reports / User	Avg No.Devices / User
18	48	2,6	6,9

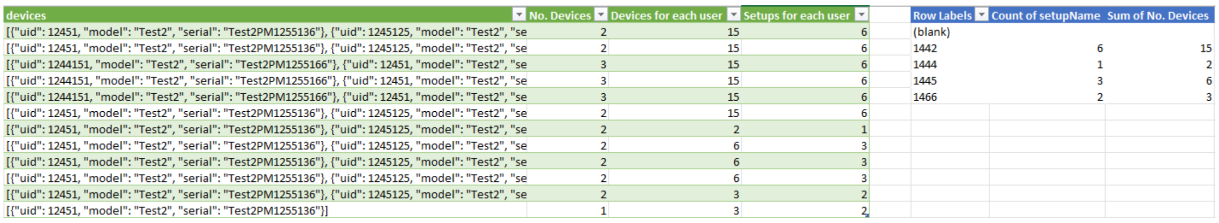
NET DEVICE

No. Devices	No. Clips	No. Prodections
17	780881	150223

Figure 19. Excel dashboard calculations

Specific sheets like “performanceReport”, “calibration”, “systemSetup”, “managementGroup” and “netDevice” have a column titled “devices” that indicates the devices assigned to each entry. This data is stored in a JSON array format.

Excel's native capabilities, such as pivot tables and built-in functions, assist in processing the JSON array data. These functions enable the extraction and computation of the number of devices and their allocation to respective users (see Figure 20).



devices	No. Devices	Devices for each user	Setups for each user
[{"uid": 12451, "model": "Test2", "serial": "Test2PM1255136"}, {"uid": 1245125, "model": "Test2", "se	2	15	6
[{"uid": 12451, "model": "Test2", "serial": "Test2PM1255136"}, {"uid": 1245125, "model": "Test2", "se	2	15	6
[{"uid": 1244151, "model": "Test2", "serial": "Test2PM1255166"}, {"uid": 12451, "model": "Test2", "se	3	15	6
[{"uid": 1244151, "model": "Test2", "serial": "Test2PM1255166"}, {"uid": 12451, "model": "Test2", "se	3	15	6
[{"uid": 1244151, "model": "Test2", "serial": "Test2PM1255166"}, {"uid": 12451, "model": "Test2", "se	3	15	6
[{"uid": 12451, "model": "Test2", "serial": "Test2PM1255136"}, {"uid": 1245125, "model": "Test2", "se	2	15	6
[{"uid": 12451, "model": "Test2", "serial": "Test2PM1255136"}, {"uid": 1245125, "model": "Test2", "se	2	2	1
[{"uid": 12451, "model": "Test2", "serial": "Test2PM1255136"}, {"uid": 1245125, "model": "Test2", "se	2	6	3
[{"uid": 12451, "model": "Test2", "serial": "Test2PM1255136"}, {"uid": 1245125, "model": "Test2", "se	2	6	3
[{"uid": 12451, "model": "Test2", "serial": "Test2PM1255136"}, {"uid": 1245125, "model": "Test2", "se	2	6	3
[{"uid": 12451, "model": "Test2", "serial": "Test2PM1255136"}, {"uid": 1245125, "model": "Test2", "se	2	3	2
[{"uid": 12451, "model": "Test2", "serial": "Test2PM1255136"}]	1	3	2

Row Labels	Count of setupName	Sum of No. Devices
(blank)		
1442	6	15
1444	1	2
1445	3	6
1466	2	3

Figure 20. Extracting and processing JSON arrays in Excel

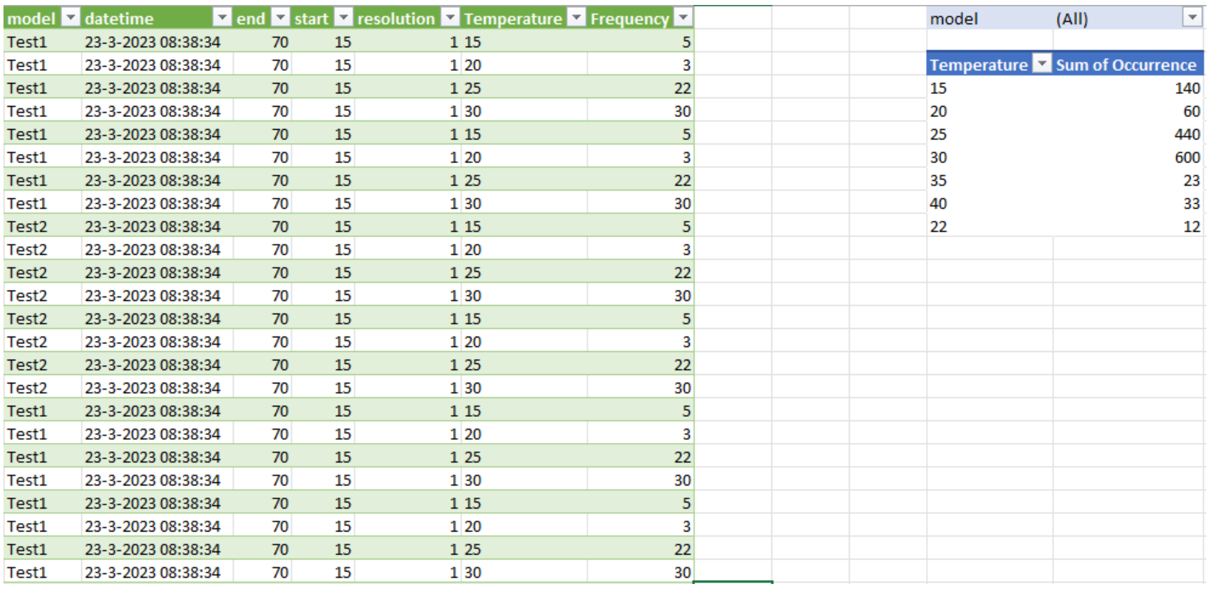
Another process is handling “Temperature”, “InputLevel”, and “OutputLevel” data, which are stored in the database in JSON strings. Through a combination of pivot tables and Power Query, these JSON structures are separated each in their individual columns to simplify the processing and analysis. For instance, a JSON structure like:

[{"end": 70, "start": 15, "temps": {"15": 5, "20": 3, "25": 22, "30": 30}, "resolution": 1}]

Is expanded into:

end	start	resolution	Temperature	Frequency
70	15	1	15	5
70	15	1	20	3
70	15	1	25	22
70	15	1	30	30

The processed data assists in populating pivot tables, which further helps in calculating the occurrence for every temperature (see Figure 21). Adding the “model” names as filters enables the analysis of temperature data on a model-wise basis. The same methodology is applied for InputLevel and OutputLevel.



model	datetime	end	start	resolution	Temperature	Frequency
Test1	23-3-2023 08:38:34	70	15	1	15	5
Test1	23-3-2023 08:38:34	70	15	1	20	3
Test1	23-3-2023 08:38:34	70	15	1	25	22
Test1	23-3-2023 08:38:34	70	15	1	30	30
Test1	23-3-2023 08:38:34	70	15	1	15	5
Test1	23-3-2023 08:38:34	70	15	1	20	3
Test1	23-3-2023 08:38:34	70	15	1	25	22
Test1	23-3-2023 08:38:34	70	15	1	30	30
Test2	23-3-2023 08:38:34	70	15	1	15	5
Test2	23-3-2023 08:38:34	70	15	1	20	3
Test2	23-3-2023 08:38:34	70	15	1	25	22
Test2	23-3-2023 08:38:34	70	15	1	30	30
Test2	23-3-2023 08:38:34	70	15	1	15	5
Test2	23-3-2023 08:38:34	70	15	1	20	3
Test2	23-3-2023 08:38:34	70	15	1	25	22
Test2	23-3-2023 08:38:34	70	15	1	30	30
Test1	23-3-2023 08:38:34	70	15	1	15	5
Test1	23-3-2023 08:38:34	70	15	1	20	3
Test1	23-3-2023 08:38:34	70	15	1	25	22
Test1	23-3-2023 08:38:34	70	15	1	30	30
Test1	23-3-2023 08:38:34	70	15	1	15	5
Test1	23-3-2023 08:38:34	70	15	1	20	3
Test1	23-3-2023 08:38:34	70	15	1	25	22
Test1	23-3-2023 08:38:34	70	15	1	30	30

model (All)

Temperature	Sum of Occurrence
15	140
20	60
25	440
30	600
35	23
40	33
22	12

Figure 21. Separation and analysis of Temperature data

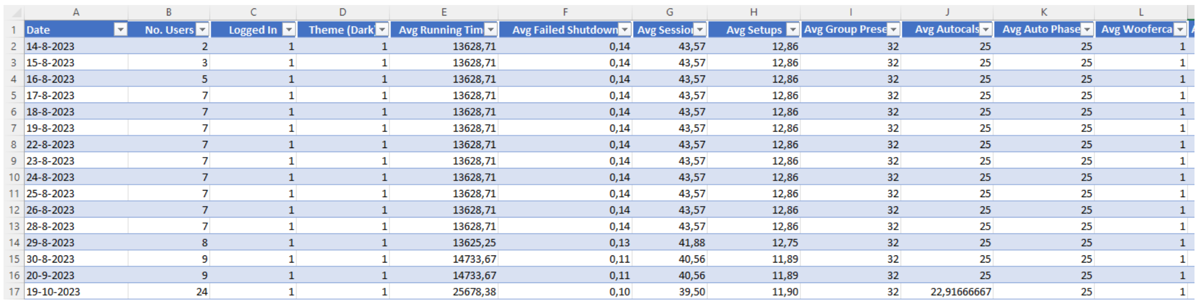
4.5.3 Handling Dynamic Updates

An inherent challenge with telemetry data is its dynamic nature. Instead of adding new data rows for every update, the database updates the existing user data. This structure presents challenges in analyzing historical data. However, VBA scripts bridge this gap.

A VBA-implemented button on the "Dashboard" sheet serves multiple purposes:

- Data Refresh: On activation, it triggers a refresh for all queries, pulling in the latest data from the database. This ensures that the dashboard always reflects the most recent data.

- Historical Data Maintenance: To maintain the historical context, an additional sheet named "Historical Data" has been set up to record each data refresh. Every time new data is fetched, the computations on the dashboard get archived in this sheet, time-stamped with the date of retrieval. Given that data is sent to the cloud in intervals of 30 days, the Excel dashboard is typically refreshed every month (see Figure 22).



Date	No. Users	Logged In	Theme (Dark	Avg Running Tim	Avg Failed Shutdown	Avg Sessio	Avg Setups	Avg Group Prese	Avg Autocals	Avg Auto Phase	Avg Wooferca
14-8-2023	2	1	1	13628,71	0,14	43,57	12,86	32	25	25	1
15-8-2023	3	1	1	13628,71	0,14	43,57	12,86	32	25	25	1
16-8-2023	5	1	1	13628,71	0,14	43,57	12,86	32	25	25	1
17-8-2023	7	1	1	13628,71	0,14	43,57	12,86	32	25	25	1
18-8-2023	7	1	1	13628,71	0,14	43,57	12,86	32	25	25	1
19-8-2023	7	1	1	13628,71	0,14	43,57	12,86	32	25	25	1
22-8-2023	7	1	1	13628,71	0,14	43,57	12,86	32	25	25	1
23-8-2023	7	1	1	13628,71	0,14	43,57	12,86	32	25	25	1
24-8-2023	7	1	1	13628,71	0,14	43,57	12,86	32	25	25	1
25-8-2023	7	1	1	13628,71	0,14	43,57	12,86	32	25	25	1
26-8-2023	7	1	1	13628,71	0,14	43,57	12,86	32	25	25	1
28-8-2023	7	1	1	13628,71	0,14	43,57	12,86	32	25	25	1
29-8-2023	8	1	1	13625,25	0,13	41,88	12,75	32	25	25	1
30-8-2023	9	1	1	14733,67	0,11	40,56	11,89	32	25	25	1
20-9-2023	9	1	1	14733,67	0,11	40,56	11,89	32	25	25	1
19-10-2023	24	1	1	25678,38	0,10	39,50	11,90	32	22,91666667	25	1

Figure 22. Snapshot of "Historical Data" Sheet

- Historical Data Analysis: The dashboard includes a dedicated section for analyzing historical data trends. Users can leverage a dropdown box to choose specific metrics, which then filters and charts the data across different time points, all sourced from the "Historical Data" sheet. For more complex data types, such as temperature, input level, and output level, separate historical charts enable date-based filtering and analysis (see Figure 23).

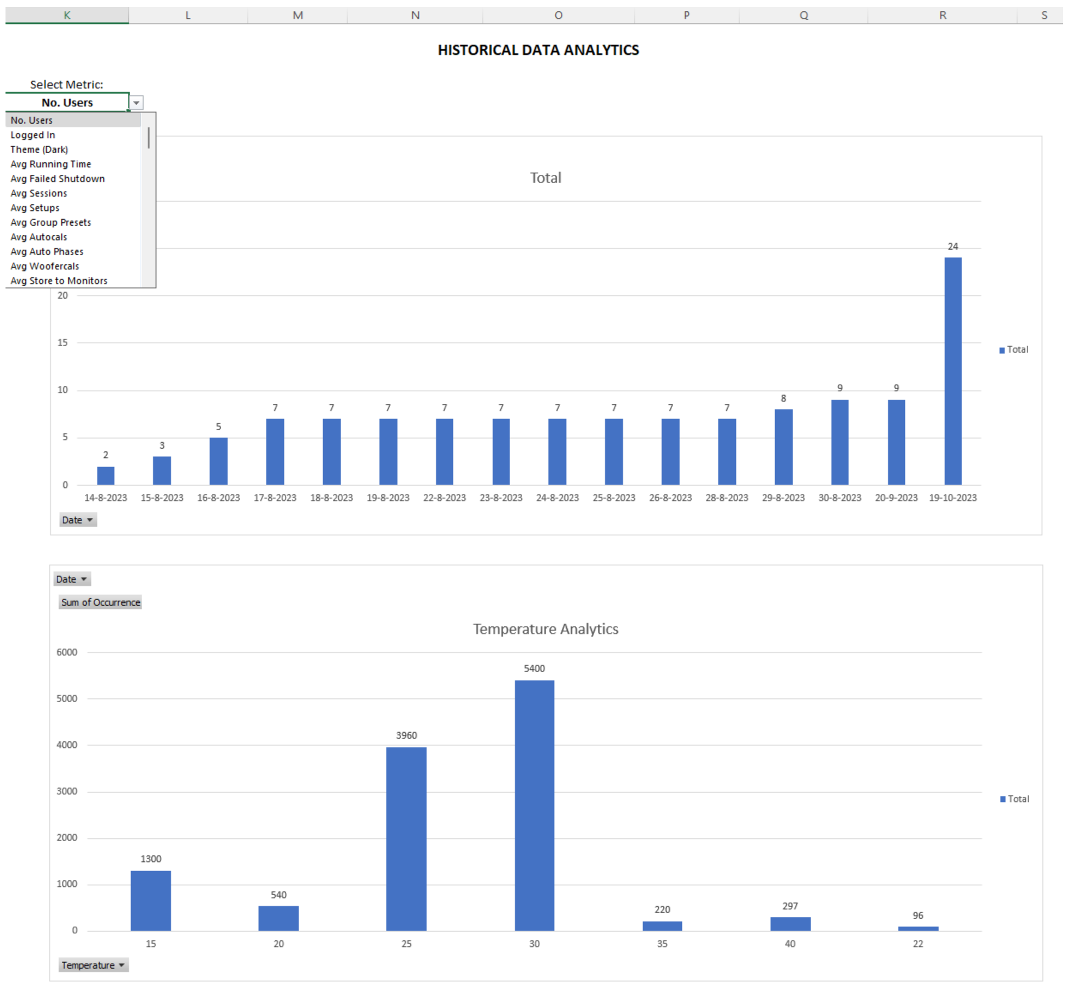
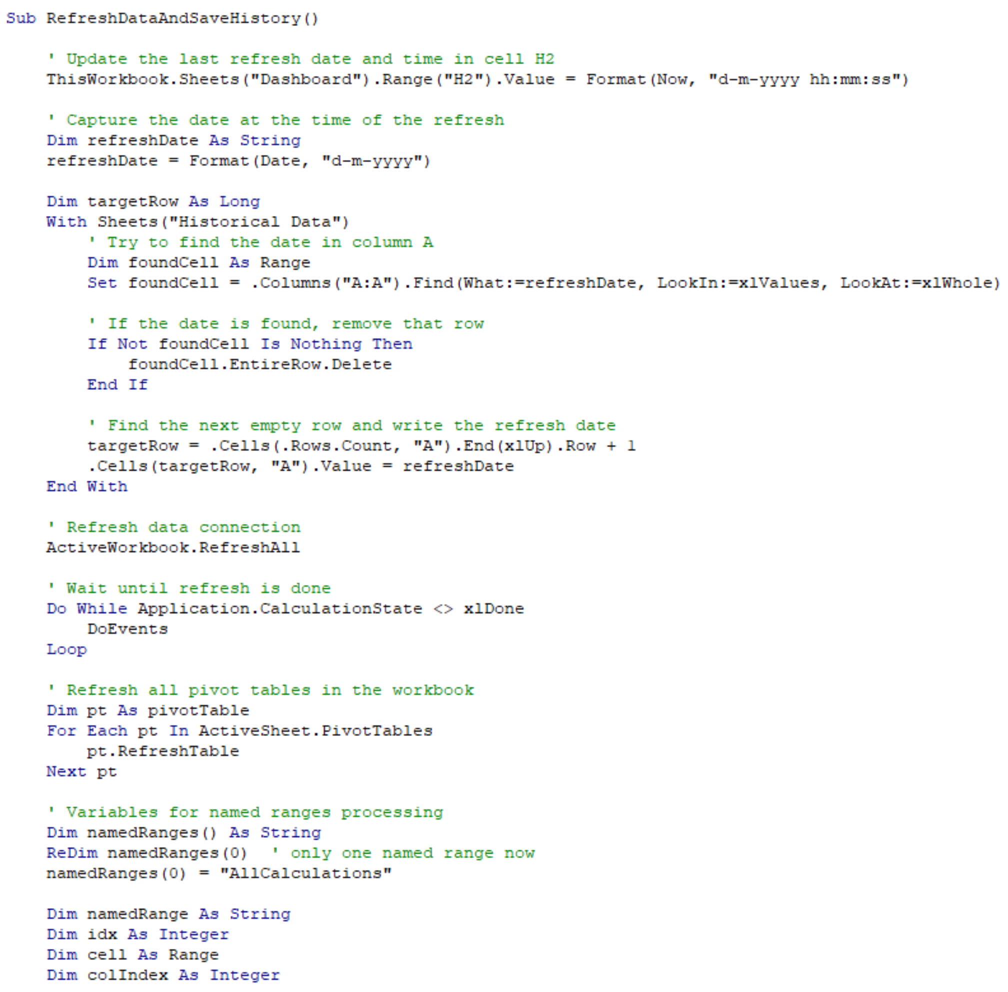


Figure 23. Historical Data Analytics

4.5.4 VBA Implementation

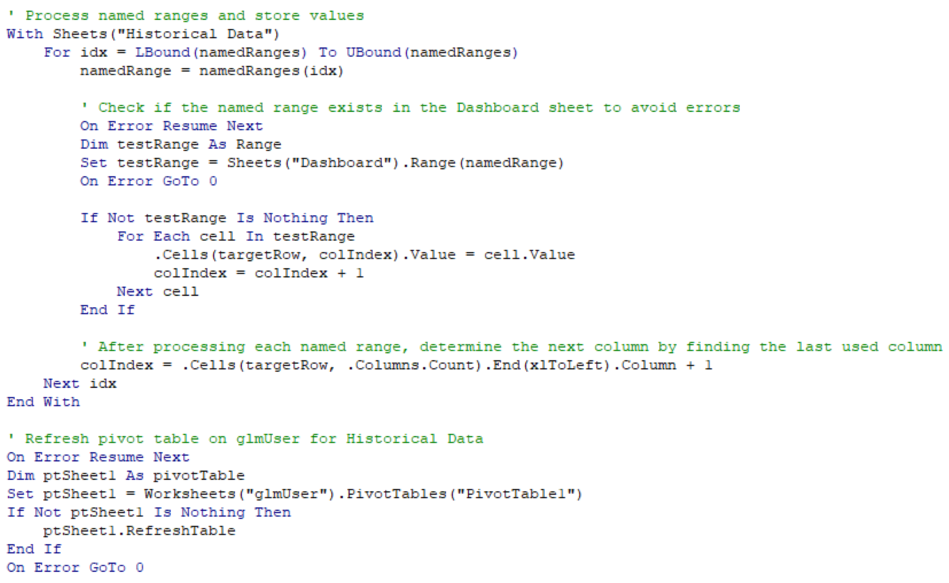
To achieve the mentioned functionalities, VBA (Visual Basic for Applications) scripts are employed. Here is a breakdown of the VBA code:

- RefreshDataAndSaveHistory: This primary subroutine refreshes all data connections, updates pivot tables (see Figure 24), and saves all calculations to the “Historical Data” sheet (see Figure 25).



```vba
Sub RefreshDataAndSaveHistory()

    ' Update the last refresh date and time in cell H2
    ThisWorkbook.Sheets("Dashboard").Range("H2").Value = Format(Now, "d-m-yyyy hh:mm:ss")

    ' Capture the date at the time of the refresh
    Dim refreshDate As String
    refreshDate = Format(Date, "d-m-yyyy")

    Dim targetRow As Long
    With Sheets("Historical Data")
        ' Try to find the date in column A
        Dim foundCell As Range
        Set foundCell = .Columns("A:A").Find(What:=refreshDate, LookIn:=xlValues, LookAt:=xlWhole)

        ' If the date is found, remove that row
        If Not foundCell Is Nothing Then
            foundCell.EntireRow.Delete
        End If

        ' Find the next empty row and write the refresh date
        targetRow = .Cells(.Rows.Count, "A").End(xlUp).Row + 1
        .Cells(targetRow, "A").Value = refreshDate
    End With

    ' Refresh data connection
    ActiveWorkbook.RefreshAll

    ' Wait until refresh is done
    Do While Application.CalculationState <> xlDone
        DoEvents
    Loop

    ' Refresh all pivot tables in the workbook
    Dim pt As pivotTable
    For Each pt In ActiveSheet.PivotTables
        pt.RefreshTable
    Next pt

    ' Variables for named ranges processing
    Dim namedRanges() As String
    ReDim namedRanges(0)  ' only one named range now
    namedRanges(0) = "AllCalculations"

    Dim namedRange As String
    Dim idx As Integer
    Dim cell As Range
    Dim colIndex As Integer
```

Figure 24. Snippet of "RefreshDataAndSaveHistory" Subroutine in VBA



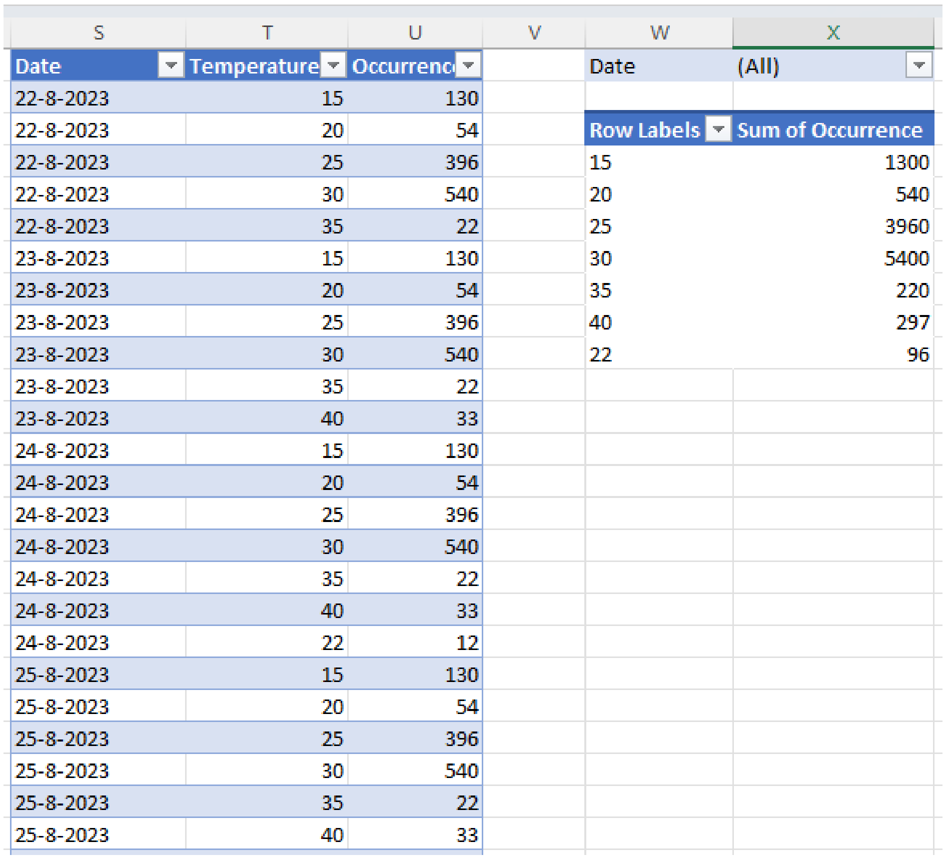
```vba
    ' Process named ranges and store values
    With Sheets("Historical Data")
        For idx = LBound(namedRanges) To UBound(namedRanges)
            namedRange = namedRanges(idx)

            ' Check if the named range exists in the Dashboard sheet to avoid errors
            On Error Resume Next
            Dim testRange As Range
            Set testRange = Sheets("Dashboard").Range(namedRange)
            On Error GoTo 0

            If Not testRange Is Nothing Then
                For Each cell In testRange
                    .Cells(targetRow, colIndex).Value = cell.Value
                    colIndex = colIndex + 1
                Next cell
            End If

            ' After processing each named range, determine the next column by finding the last used column
            colIndex = .Cells(targetRow, .Columns.Count).End(xlToLeft).Column + 1
        Next idx
    End With

    ' Refresh pivot table on glmUser for Historical Data
    On Error Resume Next
    Dim ptSheet1 As pivotTable
    Set ptSheet1 = Worksheets("glmUser").PivotTables("PivotTable1")
    If Not ptSheet1 Is Nothing Then
        ptSheet1.RefreshTable
    End If
    On Error GoTo 0
```

Figure 25. Snippet of processing and storing the calculations for historical analysis

- RefreshTemperature: This subroutine handles the refresh and save operations specifically for temperature data from the “netDevice” sheet (see Figure 26).

```
        ' Add a new row to the table and populate it
        Set newRow = lo.ListRows.Add
        newRow.Range(1, 1).Value = refreshDate
        newRow.Range(1, 2).Value = temperature
        newRow.Range(1, 3).Value = occurrence
    Next rowIndex

    ' Refresh pivot table
    Dim ptSamDeviceTemperature As pivotTable
    Set ptSamDeviceTemperature = ws.PivotTables("PivotTable1")
    If Not ptSamDeviceTemperature Is Nothing Then
        ptSamDeviceTemperature.RefreshTable
    End If
```

Figure 26. Snippet from “RefreshTemperature” Subroutine

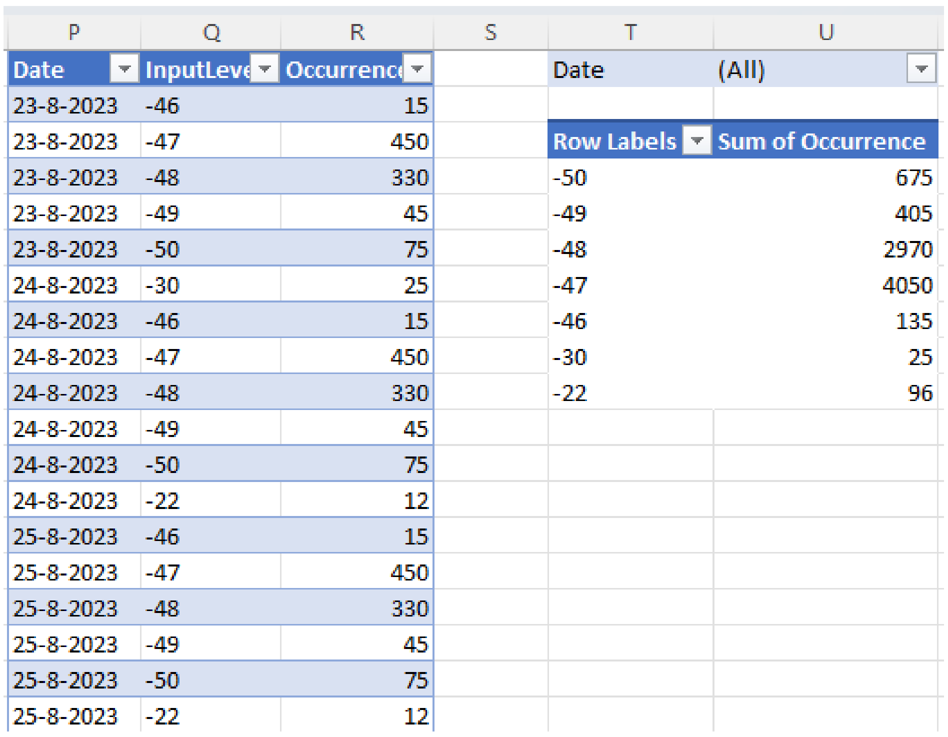


S	T	U
Date	Temperature	Occurrenc
22-8-2023	15	130
22-8-2023	20	54
22-8-2023	25	396
22-8-2023	30	540
22-8-2023	35	22
23-8-2023	15	130
23-8-2023	20	54
23-8-2023	25	396
23-8-2023	30	540
23-8-2023	35	22
23-8-2023	40	33
24-8-2023	15	130
24-8-2023	20	54
24-8-2023	25	396
24-8-2023	30	540
24-8-2023	35	22
24-8-2023	40	33
24-8-2023	22	12
25-8-2023	15	130
25-8-2023	20	54
25-8-2023	25	396
25-8-2023	30	540
25-8-2023	35	22
25-8-2023	40	33

W	X
Date	(All)
Row Labels	Sum of Occurrence
15	1300
20	540
25	3960
30	5400
35	220
40	297
22	96

Figure 27. Snapshot of Historical Table specifically for the "Temperature"

- RefreshInputLevel: Similar to the “Temperature” data handler, this subroutine refreshes and saves data for InputLevels (see Figure 28).

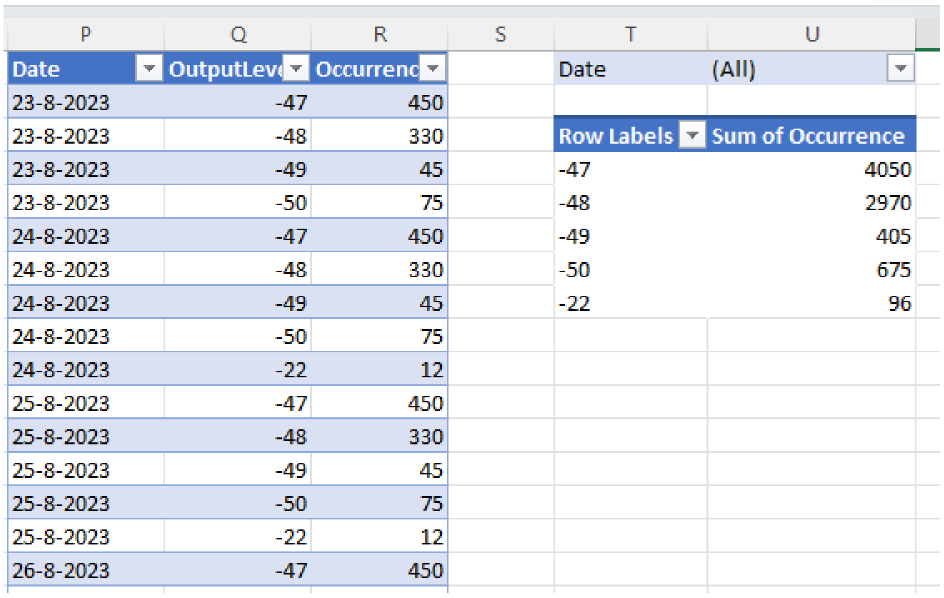


P	Q	R
Date	InputLeve	Occurrenc
23-8-2023	-46	15
23-8-2023	-47	450
23-8-2023	-48	330
23-8-2023	-49	45
23-8-2023	-50	75
24-8-2023	-30	25
24-8-2023	-46	15
24-8-2023	-47	450
24-8-2023	-48	330
24-8-2023	-49	45
24-8-2023	-50	75
24-8-2023	-22	12
25-8-2023	-46	15
25-8-2023	-47	450
25-8-2023	-48	330
25-8-2023	-49	45
25-8-2023	-50	75
25-8-2023	-22	12

T	U
Date	(All)
Row Labels	Sum of Occurrence
-50	675
-49	405
-48	2970
-47	4050
-46	135
-30	25
-22	96

Figure 28. Snapshot of Historical Table specifically for the “InputLevel”

- RefreshOutputLevel: Similar to the “Temperature” and “InputLevel” data handler, this sub-routine refreshes and saves data for OutputLevels (see Figure 29).



P	Q	R
Date	OutputLev	Occurrenc
23-8-2023	-47	450
23-8-2023	-48	330
23-8-2023	-49	45
23-8-2023	-50	75
24-8-2023	-47	450
24-8-2023	-48	330
24-8-2023	-49	45
24-8-2023	-50	75
24-8-2023	-22	12
25-8-2023	-47	450
25-8-2023	-48	330
25-8-2023	-49	45
25-8-2023	-50	75
25-8-2023	-22	12
26-8-2023	-47	450

T	U
Date	(All)
Row Labels	Sum of Occurrence
-47	4050
-48	2970
-49	405
-50	675
-22	96

Figure 29. Snapshot of Historical Table Specifically for the "OutputLevel"

4.5.5 VBA Flowchart

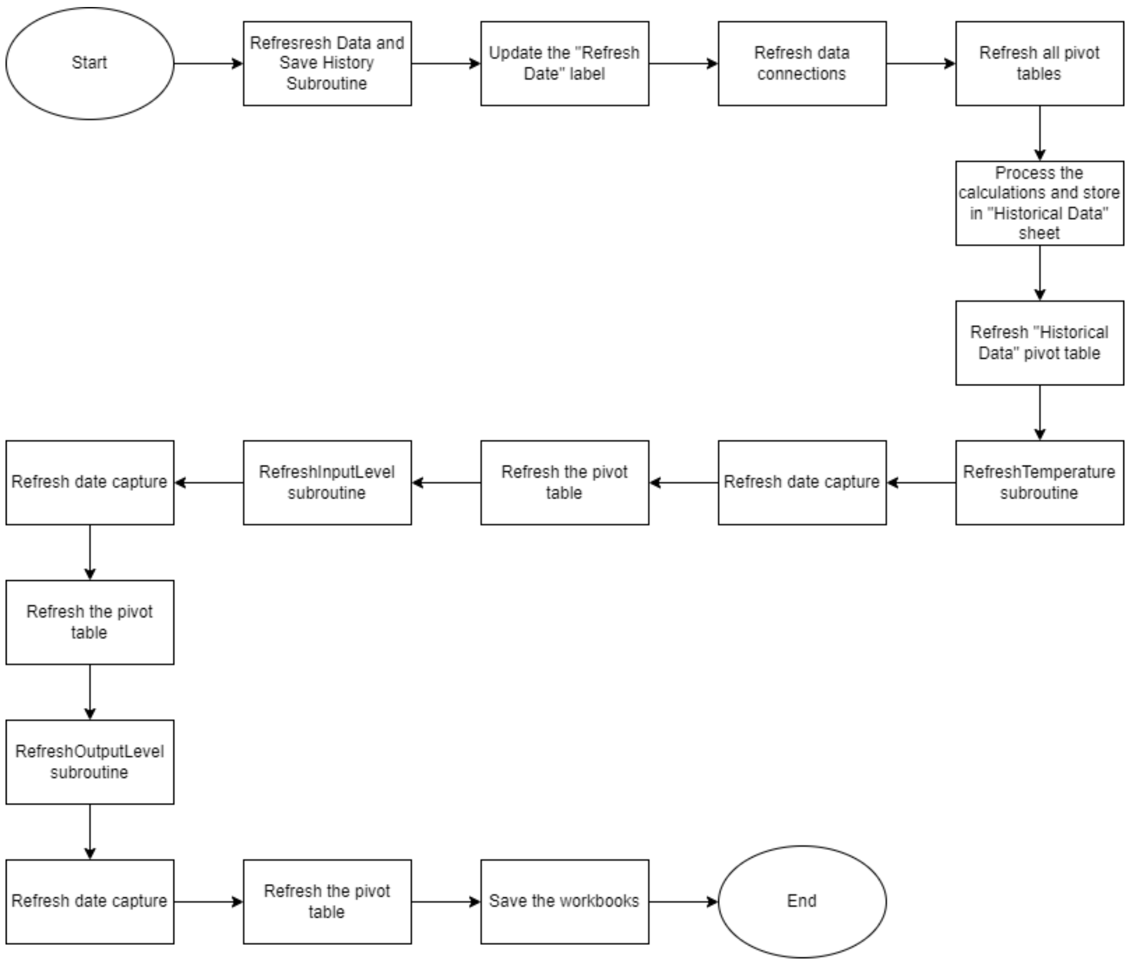

Start
Refresresh Data and Save History Subroutine
Update the "Refresh Date" label
Refresh data connections
Refresh all pivot tables
Process the calculations and store in "Historical Data" sheet
Refresh "Historical Data" pivot table
RefreshTemperature subroutine
Refresh date capture
Refresh the pivot table
RefreshInputLevel subroutine
Refresh date capture
Refresh the pivot table
RefreshOutputLevel subroutine
Refresh date capture
Refresh the pivot table
Save the workbooks
End

Figure 30. VBA Flowchart

4.5.6 Conclusion

The Excel Dashboard serves as a demonstration of the integrated capabilities of Excel, Power Query, and VBA in building a dynamic, up-to-date, and historical data analysis tool. Its implementation ensures that users always have an intuitive interface for analyzing and interpreting telemetry data, with both current and historical perspectives.

5 BENEFITS AND FUTURE IMPLEMENTATIONS OF TELEMETRY

5.1 Benefits of Telemetry

According to Richman (2023), telemetry, the automated process of collecting and transmitting data from remote locations to receiving equipment for monitoring, offers several benefits across industries.

- Improved Operational Efficiency: Telemetry allows ongoing monitoring for a clear picture of system states, reducing the need for manual checks and leading to a more streamlined workflow.
- Cost Savings: Automates data collection, freeing up human resources, reducing time, and preemptively solving problems to avoid costly repairs.
- Increased Data Accuracy: Automated and continuous data collection reduces human error and provides reliable data for better decision-making.
- Scalability: Can handle data volumes of any size, allowing operations to easily scale as required.

5.2 Benefits of Telemetry in audio control system and management software

The incorporation of telemetry in audio control systems and management software can enhance the company's approach to device data management and user experience optimization. By embedding telemetry functionalities within the software, the company can harness detailed insights from the devices, all channeled through the software platform. Below are the specific benefits of this implementation:

Improving the understanding of User-Device Interactions:

- In-depth Device Usage Profiles: By analyzing performance metrics routed through the software, there is potential for the company to develop a more detailed understanding of how devices are being utilized.
- Tailored Product Experiences: With these insights, there is a possibility for the company to introduce software updates that further resonate with the performance and usage patterns of the devices.

Potential for Optimization and Product Improvements:

- Feedback-Driven Enhancements: The continuous stream of device performance data offers an opportunity for the company to identify and address product performance discrepancies via the software.
- Product Development: The telemetry insights could potentially help the company in forecasting user demands and usage trends, thereby ensuring their offerings are aligned with evolving requirements.

Strategic Product Development Insights:

- Evidence-Based Decision Making: Leveraging telemetry data provides a robust foundation for product developmental strategies based on device performance patterns.

- Resource Allocation: By understanding the performance patterns through telemetry data, the company can allocate its resources more effectively, prioritizing areas that offer significant enhancements to the user experience.

User Empowerment through Performance Data:

- Guided Device Optimization: The local management software, informed by telemetry data, may offer users insights into specific device performance metrics, enabling them to fine-tune settings for enhanced performance.
- Proactive Maintenance Indicators: Telemetry data can act as a predictive tool, potentially alerting users about deviations in recommended performance ranges, like temperature or input/output levels. Such proactive notifications can enable timely actions, maintaining consistent device performance.

5.3 Conclusion

The telemetry approach of a company in its control system and management software shows a commitment to innovation, enhanced product quality, and customer satisfaction. By analyzing the device performance metrics, the company ensures user privacy and identifies areas for product improvement. This approach highlights a company's commitment to continuous, data-driven advancement.

6 DISCUSSION

Telemetry's integration within a local control system and management software is highlighted as an important mechanism for enhancing product understanding and user experience. However, ensuring the quality and reliability of the data captured is critical. The efficacy of the telemetry system relies on the precision and consistency of the data points collected.

Variability in the data points or inconsistencies can alter the interpretations and potentially misguide future developmental strategies. As such, emphasizing consistent data quality and refining the parameters for data capture become pivotal. This involves careful analysis of the incoming data, adjusting the data collection based on feedback, and periodically reviewing the importance of the data collected.

Furthermore, seamless integration of telemetry data with other services of a company can potentially offer more profound insights. Pursuing such integrative efforts not only enhances the depth of understanding regarding device performance but also the value telemetry brings to both company and its users.

For future enhancements, broadening the telemetry system to capture more detailed usage patterns and collaborating with other software metrics are promising directions. Keeping the telemetry system up to date with evolving technological standards, user needs, and industry benchmarks is crucial to maintaining its significance.

7 CONCLUSION

7.1 Summary of Findings

Telemetry incorporation within a control system and management software emphasizes the move to a more data-driven and user-centric approach. The primary data points revolve around device performance metrics, ensuring user privacy while facilitating product improvement. The importance of integrating a telemetry system while maintaining data integrity and privacy has been highlighted.

7.2 Contributions and Impact

The telemetry system can significantly enhance the ability of a company to understand user behavior, device performance, and development requirements. Presenting this information in an easily interpretable manner will empower both the company and its users. For the company, it will guide product enhancements, while for users, it will facilitate optimal device usage.

7.3 Final Remarks

The exploration of telemetry within the context of control systems and management software highlights its potential in revolutionizing user experience and product development. Its impact goes beyond simple data collection. It offers the opportunity for fostering innovation, enhancing product quality, and improving user understanding. As companies look ahead, telemetry provides a pathway not just for understanding but for significant advancement. Navigating this pathway, while being aware of the challenges, can guide companies to new heights of success and user satisfaction.

REFERENCES

ChatGPT 2023. OpenAI. GPT-4. Accessed for language check, November 2023. https://chat.openai.com

Analytics Vidhya 2023. A Comprehensive Guide on Microsoft Excel for Data Analysis. https://www.analyticsvidhya.com/blog/2021/11/a-comprehensive-guide-on-microsoft-excel-for-data-analysis/. Accessed 8.11.2023.

AWS Aurora n.d. Amazon Aurora. https://aws.amazon.com/rds/aurora/. Accessed 8.11.2023.

AWS EKS n.d. Amazon Elastic Kubernetes Service. https://aws.amazon.com/eks/. Accessed 8.11.2023.

AWS n.d. What is Cloud Storage? - Cloud Storage Explained - AWS. https://aws.amazon.com/what-is/cloud-storage/. Accessed 8.11.2023.

Docker n.d. Use containers to Build, Share and Run your applications. https://www.docker.com/resources/what-container/. Accessed 9.11.2023.

Funnel n.d. The Best Data Visualization Tools — According to Funnel. https://funnel.io/blog/the-top-visualization-tools-according-to-funnel. Accessed 8.11.2023.

GitLab n.d. Get started with GitLab CI/CD. https://docs.gitlab.com/ee/ci/. Accessed 9.11.2023.

GitLab n.d. What is version control? https://about.gitlab.com/topics/version-control/#how-does-version-control-streamline-collaboration. Accessed 9.11.2023.

Harbor n.d. https://goharbor.io/. Accessed 8.11.2023.

JUCE n.d. https://juce.com/. Accessed 8.11.2023.

Junnila, A n.d.-a. How IoT Works – Part 3: Data Processing. Trackinno. https://trackinno.com/iot/how-iot-works-part-3-data-processing/. Accessed 8.11.2023.

Junnila, A n.d.-b. How IoT Works – Part 4: User Interface. Trackinno. https://trackinno.com/iot/how-iot-works-part-4-user-interface/. Accessed 8.11.2023.

KrakenD n.d. https://www.krakend.io/. Accessed 8.11.2023.

Kutay, J n.d. Types of Data Integration: ETL vs ELT and Batch vs Real-Time. Striim. https://www.striim.com/blog/data-integration/. Accessed 14.11.2023.

MySQL n.d. How MySQL Uses Indexes. https://dev.mysql.com/doc/refman/8.0/en/mysql-indexes.html. Accessed 9.11.2023.

Pelaez, A 2021. Here's How IoT Data Collection Works [Complete Guide]. Ubidots. https://ubidots.com/blog/iot-data-collection/. Accessed 8.11.2023.

PYnative 2021. Python MySQL Execute Parameterized Query using Prepared Statement. https://pynative.com/python-mysql-execute-parameterized-query-using-prepared-statement/. Accessed 9.11.2023.

Python Basics n.d. Flask HTTP methods, handle GET & POST requests. https://pythonbasics.org/flask-http-methods/. Accessed 9.11.2023.

Ramos, C 2022. Database schema design 101 for relational databases. PlanetScale. https://planetscale.com/blog/schema-design-101-relational-databases. Accessed 9.11.2023.

Richman, J 2023. What Is Telemetry Data? Uses, Benefits, & Challenges. Estuary. https://estuary.dev/Telemetry-data/. Accessed 9.11.2023.

SQL Shack 2022. Using parameterized queries to avoid SQL injection. https://www.sqlshack.com/using-parameterized-queries-to-avoid-sql-injection/. Accessed 9.11.2023.

Sumo Logic n.d. What is telemetry? https://www.sumologic.com/glossary/telemetry/. Accessed 30.10.2023.

Talend n.d. What is ETL? https://www.talend.com/resources/what-is-etl/. Accessed 9.11.2023.

TechTarget 2023. What is IoT Security? https://www.techtarget.com/iotagenda/definition/IoT-security-Internet-of-Things-security. Accessed 8.11.2023.

Yildirim, S 2022. Data Processing in Python. LearnPython. https://learnpython.com/blog/data-processing-in-python/. Accessed 8.11.2023.