FMH606 Master's Thesis 2023

Master of Science, Industrial IT, and Automation

IOTA for Industry 4.0 to handle production processes.

Khurram Baig

Faculty of Technology, Natural sciences and Maritime Sciences

Campus Porsgrunn

University of South-Eastern Norway

www.usn.no

Course: FMH606 Master's Thesis, 2023

Title: IOTA for Industry 4.0 to handle production processes

Number of pages: 64

Keywords: Blockchain, IOTA, Tangle, Distributed Ledger Technology, Consensus Algorithm, Process Data, Data Storage, Data Analysis, Response Time, Transaction Confirmation Time

Student:	Khurram Baig
Supervisor:	Leila Ben Saad
External partner:	**NA**

Summary:

This master thesis explores the potential of Decentralized Ledger Technologies (DLTs), particularly IOTA Tangle, in storing and analyzing data. The objective of the study is to develop an application that can store data on IOTA Tangle and retrieve it over the internet for analysis and error detection. This thesis provides an overview of various DLTs, their limitations, and challenges in developing a data storage application. Then a detailed explanation of the system's design, implementation, and testing methodology is provided. Furthermore, this study presents an analysis of the results obtained from the implementation of the IOTA-based solution, and discusses its limitations, challenges, and possible directions for improvement. Finally, the main findings, contributions, and limitations of the study are summarized and recommendations for future research are provided.

Preface

The 4th Industrial Revolution, commonly known as Industry 4.0, has brought significant changes to the manufacturing industry. With the advent of new technologies such as the Internet of Things (IoT) and blockchain, there has been a growing need for innovative solutions that can help streamline production processes and improve efficiency.

This Master Thesis explores the use of IOTA, a cutting-edge distributed ledger technology, as a solution for Industry 4.0 in terms of data storage. By leveraging IOTA's unique properties such as feeless transactions, scalability, and data integrity, we aim to demonstrate how it can be used to handle production processes data and make it available for Data Analysis.

The research presented in this thesis is the culmination of months of hard work and dedication, and I am much obliged to share my findings with the academic community. I would like to extend my sincere thanks to my supervisor for her invaluable support and guidance throughout this journey. I am grateful to my family for their unwavering support and understanding during this journey. They have been by my side, offering moments of joy and occasional distractions when I needed them most. Their presence and encouragement have made a significant difference, and I am deeply thankful for their love and understanding.

I hope that this thesis will contribute to the ongoing discourse around the application of Distributed Ledger Technologies in Industry 4.0 and inspire further research into the use of emerging technologies to address the challenges faced by modern manufacturing.

Sandvika, 13-05-2023

Khurram Baig

Contents

Figures

Nomenclature

API	-	Application Programming Interface
BC	-	Blockchain
CPU	-	Central Processing Unit
CSS	-	Cascading Style Sheets
DAG	-	Directed Acyclic Graph
DLT	-	Distributed Ledger Technology
GUI	-	Graphical User Interface
HTML	-	Hyper Text Markup Language
IoT	-	Internet of Things
KPI	-	Key Performance Indicator
OPC UA	-	Open Platform Communications Unified Architecture
PoA	-	Proof of Authority
PoS	-	Proof of Stake
PoW	-	Proof of Work
SSD	-	System Sequence Diagram
UTF	-	Unicode Transformation Format
VET	-	Vechain Token
VETHO	-	Vechain Thor Token

1 Introduction

Industry 4.0 [1], also known as the fourth industrial revolution, refers to the digitalization of industrial and manufacturing processes. It has evolved over time to incorporate cyber-physical systems, the Internet of Things (IoT) [2], and cloud computing, providing value to industrial processes by distributing information and enabling data analysis.
In IoT, a sensor or a device is capable of acquiring and translating process data from the field into digital form and making it available for computing for other systems over the internet, which can further analyze the information to adds value to any process or service. It would not be an exaggeration to state that IoTs are the eyes and ears of Industry 4.0.
The emergence of IoT has led to an exponential increase in data generation, which has to be communicated to the algorithm running machines over the internet and thus generates the necessity for new methods of data storage, cyber security, and data analysis. This is where Decentralized Ledger Technologies (DLT), such as Blockchain [3] and IOTA Tangle [4] have shown their potential to provide a secure and decentralized way to manage the vast amount of data generated by IoT devices.
DLTs allow multiple participants to exchange and store data securely across the network of computers rather than centralized computers, where every computer on the network validates the data every time new data has entered the network, making it even harder to infiltrate or temper. Blockchain is the most well-known DLT being used in transactions of cryptocurrencies like Bitcoin, However, other DLTs have certain advantages over Blockchain in terms of scalability, energy efficiency, data privacy and cost of the transaction, like IOTA Tangle and Vechain blockchain.
IOTA Tangle is the most recent development among existing DLTs, and it is more suitable for machine-to-machine transactions in the case of IoT. As claimed, IOTA Tangle is more suitable for IoT because it uses a different architecture than blockchain called directed acyclic graph (DAG) in which there are no miners and transaction fees are minimal or even non-existent. This study will assess the practicality of IOTA Tangle on the basis of the performance of a brief data storage application.

1.1 Objectives

The main objective of this master thesis is to design and develop an application, which can interact with the IOTA ledger, store data on the Tangle and retrieve it over the internet to analyze and detect any errors in data. Data could be acquired by an IoT device, generated by an industrial process, and collected through OPC UA, Message Queuing Telemetry Transport (MQTT) or any other filed communication protocol. The acquisition of data is not the matter of interest of this study and will depend on the application specific requirements, which are left to the reader's discretion.

The thesis will also present a brief review of various distributed ledger technologies, including Blockchain, IOTA and Vechain [5], that can be used to store data collected from machines or IoT devices. Additionally, the thesis will discuss the challenges and limitations encountered in developing an IOTA-based solution and propose possible directions for improvement. Ultimately, the goal is to explore the IOTA Tangle for its abilities and limitations in storing process data and make it available for process handling.

The main objectives of this study are:

- Review of distributed ledger technologies, including IOTA, Vechain and Blockchain for storing machine or IoT device data.
- Design and develop an application to store and retrieve data on IOTA Tangle for analysis and error detection.
- Develop IOTA client using Python (Application Programming Interface) API, sends data to IOTA node on Chrysalis Devnet, stored on IOTA Tangle.
- Develop IOTA subscriber using Python API, retrieves data from IOTA node, perform analysis, and send the data to the web app using Flask framework.
- Utilize Flask web framework for providing an interface to display analyzed data and highlight the main existing errors in the process.
- Discuss the challenges and limitations of developing IOTA-based solutions and possible directions for improvements.

1.2 Report Structure

This thesis is structured as follow:

1. Introduction: This chapter introduces the thesis's background, motivation, and objectives. It also provides an overview of the structure of the thesis.
2. Literature Review: This chapter presents a review on uses and limitations of IOTA, Vechain and Blockchain ledger technologies in IoT applications.
3. Methodology: This chapter provides a detailed explanation of the methodology used in the development of the IOTA-based solution, which is discussed in Chapter 1, including the tools and technologies used, the design and implementation of the system, and the testing methodology.
4. Results Analysis and Discussion: This chapter provides an analysis of the results obtained from the implementation of the IOTA-based solution. It also provides a detailed discussion of the challenges and limitations encountered in the development of the IOTA-based solution. It also discusses possible directions for improvement.
5. Conclusion: This chapter summarizes the main findings of the thesis, the contributions made to the field, and the study's limitations. It also suggests some recommendations for future research.

2 Literature Review

This chapter presents a literature review of distributed ledgers and discusses the working principles, differences, advantages, and disadvantages of the most popular ledgers such as Blockchain, Vechain, and IOTA Tangle.

2.1 Distributed Ledger Technology

The emergence of concepts and practices like Industry 4.0, digitalization, and the Internet of Things has resulted in unprecedented data exchange and storage requirements in the technological era. This new technological trend has already surpassed the limits of current practices of data exchange and storage, which are mostly centralized in nature. This centralization makes scaling difficult, increases the risk of a single point of failure, makes the system vulnerable to cyber-attacks as the number of users in the network grows, and is expensive to maintain.

Distributed Ledger Technology, as the terminology is self-explanatory, is an ever-growing record of transactions and data exchange which is continuously being validated and updated by various nodes within a network. Nodes operate in a peer-to-peer (P2P) fashion, where each of them maintains an identical copy of this shared record, thus without requiring a central authority to update and communicate. DLT allows each participant to securely verify and store data and makes it decentralized to overcome all the challenges mentioned earlier in the case of centralized storage. However, there are multiple users over the network trying to transfer or access the data simultaneously, it is inevitable to have a robust mechanism in place that is secure and implacable [6-8].

There are several DLTs in use today and categorized as public or private/permissioned or permissionless networks, which allow information to be stored using cryptography. The data is exchanged or accessed using unique “Keys” and signatures. Once the transaction is executed into the ledger, it becomes part of the immutable database and it is updated through the network using rules and techniques, which ensure the state of the database is synchronized. This is done by so-called consensus algorithm which is programmed for fully automated data audit across the DLT. Different DLTs use different consensus algorithms, their advantages and disadvantages are discussed later in this chapter [6-8].

2.1.1 Private Permissioned Network

DLT that uses a private permissioned network, offers no decentralization. It is a managed network, where groups of specific participants are permitted to exchange data or host their nodes that meet the criteria of that network. This type of ledger is more suitable for enterprises and organizations who require the benefits of DLTs with controlled and restrictive access. Examples can be government, finance, or health sectors [6], [8].

2.1.2 Private Permissionless Network

In a private permissionless DLT, participants can participate after they gain permission from the governing entity and are not able to exchange data or deploy their application before gaining access to the network. The network is decentralized and created or managed by participants without any need to acquire permission [6], [9].

2.1.3 Public Permissioned Network

In a public permissioned DLT, participants are free to exchange data or deploy their applications. The nodes within the network must be invited or gain permission before they are established. This type of network is decentralized but still managed by a group of entities to increase the number of nodes [6], [9].

2.1.4 Public Permissionless Network

A public permissionless network is a fully decentralized DLT, any participant can join the network and nodes can be operated without any permission. No single entity manages the participants or nodes within the network and transactions are governed by an automated consensus algorithm, example of such DLTs is Blockchain and Ethereum [6], [9].

2.1.5 Consensus Algorithms

Although every node on the network contains its own copy of the ledger, it is imperative for these distributed ledgers to be identical. Any discrepancy in the state of these ledgers can challenge the usability of DLTs. To ensure the distributed ledgers remain identical, a consensus algorithm is necessary to synchronize the ledger state across the distributed system. This mechanism guarantees that copies distributed across the network are consistent. There are many different types of consensus algorithms in use by DLTs. This study will discuss consensus algorithms used by Blockchain, Vechain, and IOTA networks [6], [10].

2.1.5.1 Proof of Work

Proof of Work (PoW) is a popular consensus algorithm implemented in blockchain DLT like Bitcoin. In this consensus algorithm, whenever the new transactions are broadcasted to the network and stored in a block, multiple nodes on the network also called “miners” compete to solve a complex cryptographic function to validate the transactions and create a new block in the blockchain network. The miner who solves the cryptographic puzzle first broadcasts the solution to a network, where other nodes verify the solution and add the new block in the network if it is valid. The miner who successfully adds a block gets Bitcoin as a reward. As this consensus algorithm performs a lot of computations it requires that the computing node essentially a miner must invest in physical resources and provide energy for processing power. These constraints make PoW a poor choice when it comes to scalability [6], [10].

2.1.5.2 Proof of Stake

Proof of Stake (PoS) is another consensus algorithm that is in use by DLTs like Ethereum. In PoS, there are validator nodes instead of miner nodes that validate the transaction as they are selected based on how much value or cryptocurrency they hold and are willing to put on a stake as collateral (willing to lose in case of an unsuccessful transaction). When transactions are broadcasted on the network, validators on the network offer to put a value on stake, and the validator is randomly selected based on stake value. A higher stake value means more chances of getting selected. The selected validator validates the transaction and adds a new block in the network. Upon a successful transaction, the validator receives a transaction fee and a portion of the cryptocurrency that they had staked earlier. PoS is comparatively more energy efficient and requires less computational power than PoW [6], [10].

2.1.5.3 Proof of Authority

Proof of Authority (PoA) is a consensus algorithm used by VechainThor DLT. It is mainly designed for energy efficiency, scalability, and security, and unlike PoW and PoS, there is no competition among nodes to get the right to validate the transaction or add a new block. Instead, this right is granted to a selected group of nodes known as Authority Masternodes. These node operators should reveal their identity and reputation to Vechain foundation to get the right for producing the blocks according to the foundation's governance policy. The fixed set of Authority Masternodes allows for faster and more reliable transaction processing, reducing the risk of network congestion and delays, which is one of the reasons why Vechain is suitable for enterprise users [5].

2.1.5.4 Tangle Coordinator

The Tangle is a DLT used in the IOTA network, which uses a directed acyclic graph (DAG) structure to store transactions. The innovation that makes the Tangle different than other DLTs is that it does not rely on a traditional consensus algorithm, such as PoW or PoS to validate transactions and secure the network. Instead, in Tangle, each new transaction must approve at least two previous transactions before it can be added to the network. This mechanism makes a web of transactions validate each other eliminating the concept of a miner or validator node from the network. This web of transactions creates a data structure called a Directed Acyclic Graph (DAG) of blocks where each new transaction is attached to multiple previous ones. This data structure or ledger is replicated across the network on all nodes, and consensus is reached on the state of the ledger using a milestone, which is a special type of transaction that confirms a set of previous transactions and marks a checkpoint in the Tangle’s history. All transactions directly or indirectly referenced by the milestone are confirmed and part of the irreversible Tangle. The milestone is generated by the central node called the Coordinator, and it is worth mentioning that since the Coordinator affects the decentralization of the whole Tangle concept. It is a temporary solution and will be eliminated in the next version of Tangle 2.0 [4], [11].

2.1.5.5 Tangle 2.0 Coordicide

Tangle 2.0, the latest version of the IOTA Tangle, is currently under development in 2023 and comes with several new features and enhancements. One of the most significant changes is the introduction of Coordicide, a roadmap for removing the coordinator and achieving complete decentralization of the IOTA network. This step is crucial for achieving greater scalability, improved security, and increased functionality, including Sybil protection. With better consensus, reduced communication overhead, and a lightweight protocol, the IOTA Tangle is becoming increasingly attractive for IoT applications. [4], [11], [16].

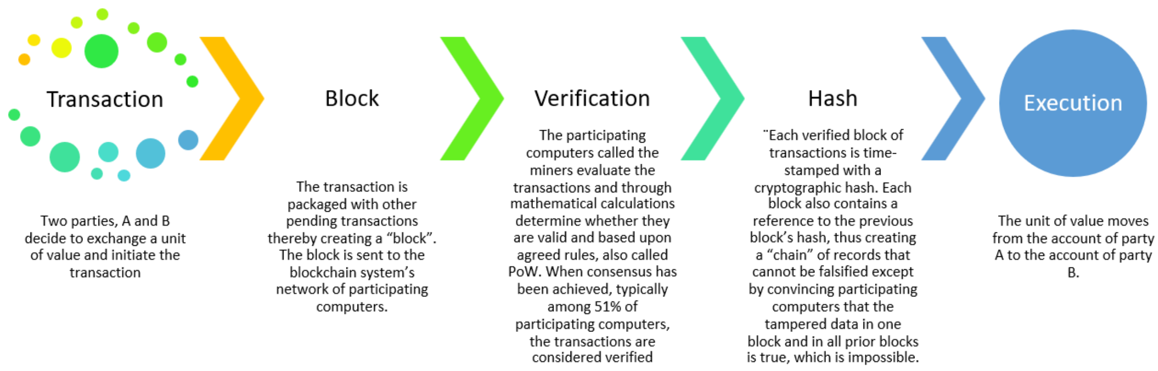
2.2 Blockchain

Blockchain [3] is a DLT commonly associated with the cryptocurrency Bitcoin as its application in performing secure and immutable transactions has been very successful since 2009. Although Blockchain is a type of DLT and shares similarities with traditional databases, it differs significantly in terms of information storage and management. Unlike a database that stores data in rows, columns, and tables, Blockchain stores data in digital blocks that are linked to each other through a cryptographic hash. Furthermore, it is decentralized, meaning that data is distributed across multiple computers on the network, rather than being centralized on a single server as with traditional databases.

Blockchain technology has diverse applications beyond cryptocurrency. For instance, logistics companies utilize blockchain to track and trace goods as they move through the supply chain. The finance industry is constantly developing blockchain-based applications, and various other industries are exploring its usability to replace traditional databases. Additionally, this study will provide an overview of its applications in IoT, which is of great interest for digitalization and industry 4.0 [1].

2.2.1 Blockchain Working

A transaction in blockchain goes through multiple steps before it gets finally executed successfully. A breakdown of these steps is shown in Figure 1



Two parties, A and B decide to exchange a unit of value and initiate the transaction

The transaction is packaged with other pending transactions thereby creating a "block". The block is sent to the blockchain system's network of participating computers.

The participating computers called the miners evaluate the transactions and through mathematical calculations determine whether they are valid and based upon agreed rules, also called PoW. When consensus has been achieved, typically among 51% of participating computers, the transactions are considered verified

"Each verified block of transactions is time-stamped with a cryptographic hash. Each block also contains a reference to the previous block's hash, thus creating a "chain" of records that cannot be falsified except by convincing participating computers that the tampered data in one block and in all prior blocks is true, which is impossible.

The unit of value moves from the account of party A to the account of party B.

Figure 1: Transaction flow in Blockchain [12].

Newly initiated transactions are assembled into a data block that is not yet added to the blockchain. The first block consists of a header, transaction data, timestamp, and other cryptographic parameters. Under the verification stage, all these parameters become input for computational nodes responsible for performing Pow and generating Hash, which represents a unique digital fingerprint of the original input. Hash is added to the block when it is created and each subsequent block in the ledger uses the previous block's hash to calculate its hash, though making a blockchain. Once a block is added to the blockchain, it cannot be changed. If an attempt is made to tamper with the block, it will disrupt the hash of the previous block and cause a ripple effect through the blockchain, which will create inconsistency in the ledger state. Computational nodes will stop adding new blocks and discard the problematic block and run the PoW again to solve the inconsistency. In this way, the blockchain ledger is nearly impossible to tamper with and provides the highest degree of security and immutability.

2.3 Vechain

Vechain [5] is a blockchain platform that focuses on supply chain management for enterprises. Its native cryptocurrency is Vechain tokens (VET), and it utilizes Proof of Authority (PoA) as its consensus algorithm, as discussed in Chapter 2.1.5.3. On the other hand, VechainThor is an upgraded version of Vechain that provides a more robust blockchain infrastructure for governments and enterprises. VechainThor boasts faster transaction processing, high scalability, and high-volume transaction processing capabilities [5].

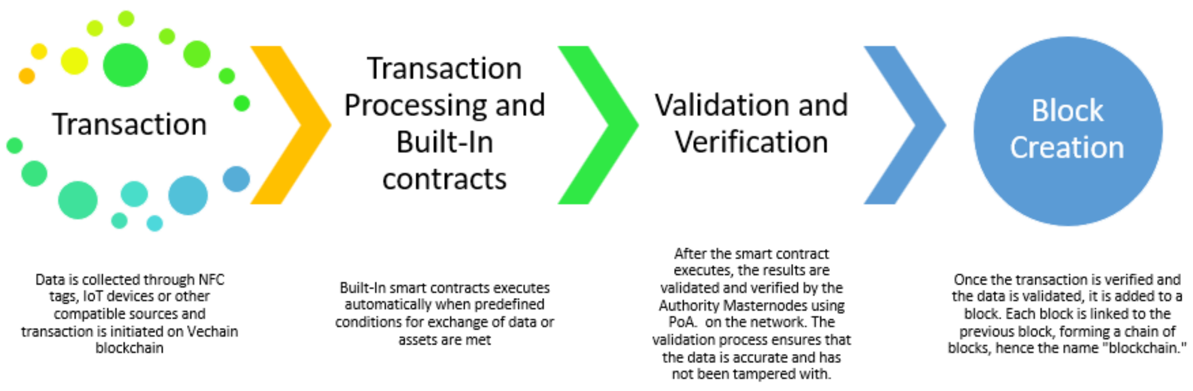
VechainThor blockchain is not built from scratch but inherits many features of the Ethereum blockchain, to mention few, are the account model and Ethereum Virtual Machine (EVM) that enables smart contracts. Despite being successful technology, Ethereum lacks in ability to host large-scale decentralized applications due to the absence of a governance structure to enhance the working protocol to tackle new challenges and innovations. Furthermore, Ethereum does not have a stable economic model that can be used by enterprises. For instance, the volatility of ether price is a big concern for companies, as they cannot predict the cost of running decentralized applications.

2.3.1 Vechain Working

VechainThor addresses issues by providing a governance model, an economical model, and an efficient consensus algorithm Proof of Authority (PoA). Overall, VechainThor offers the following enhancements in the interest of enterprises [5].

- Low computation power to achieve consensus while extending blockchain securely.
- Built-In contracts which make the network more autonomous and robust.
- Governance model which offers an adjustable balance between decentralization and centralization to achieve efficiency and transparency.
- Two token-based economic model Vechain Token and VechainThor Token (VET + VTHO), where VET serves as a value transfer medium in the VechainThor ecosystem, VETHO separates the cost of usage of VechainThor blockchain [5].

Figure 2 shows a rough sketch of how transactions proceed in the VechainThor blockchain platform.


Transaction
Transaction Processing and Built-In contracts
Validation and Verification
Block Creation

Data is collected through NFC tags, IoT devices or other compatible sources and transaction is initiated on Vechain blockchain

Built-In smart contracts executes automatically when predefined conditions for exchange of data or assets are met

After the smart contract executes, the results are validated and verified by the Authority Masternodes using PoA. on the network. The validation process ensures that the data is accurate and has not been tampered with.

Once the transaction is verified and the data is validated, it is added to a block. Each block is linked to the previous block, forming a chain of blocks, hence the name "blockchain."

Figure 2: Transaction flow in Vechain.

2.4 IOTA Tangle

The mathematical concept of the Tangle was first introduced in 2015 in a white paper authored by Serguei Popov [14]. Till now, there have been 31 versions of this white paper. The IOTA Foundation, a non-profit organization, was founded in 2017 to support the development and adoption of the IOTA Tangle technology [14].

IOTA is a distributed ledger technology, which is very suitable for supporting Internet of Things applications due to salient features like security, scalability, and feeless transactions. Unlike other protocols, IOTA does not use existing blockchain data transfer structures but uses a Directed Acyclic Graph (DAG) structure, which is called the Tangle. The Tangle allows transactions to be validated by each other in a way that each new transaction should validate two previous transactions creating a decentralized network where each participant contributes to the security and verification of transactions. This type of data structure allows a high degree of scalability and fast transaction time since

IOTA's core functionality is its ability for peer-to-peer data transactions, which are secure, decentralized, and feeless efficient, making it very attractive for industrial applications where handling of mass data is required.

2.4.1 IOTA Working

IOTA's Tangle is composed of computer nodes across the network that verify transactions and hold all the necessary information to trace back the origin of the data or value. As shown in Figure 4, each new transaction must validate the two previous ones, it creates a DAG of data blocks where each block is attached to multiple older blocks. In this way, transactions are validated in parallel and provide much higher scalability and throughput than blockchains.

The ledger status reaches consensus across the network with the help of a special node called the Coordinator (COO), which is shown in Figure 4. The coordinator is a central entity in Tangle that helps transactions reach their finality (transaction is confirmed and is irreversibly part of the Tangle) in the ledger by generating milestone transactions at regular intervals. When the transaction is initiated, a node accepts the new incoming transaction and adds it into Tangle, and broadcast further to peer nodes, but the transaction has not reached its finality yet and waiting to be directly or indirectly referenced by milestone transaction. Milestone transactions contain a signature from the coordinator which allows nodes to recognize whether the authentic coordinator did sign them. To ensure every transaction gets a fair chance of reaching its finality Coordinator sends milestones transactions every 10 seconds [4], [11].

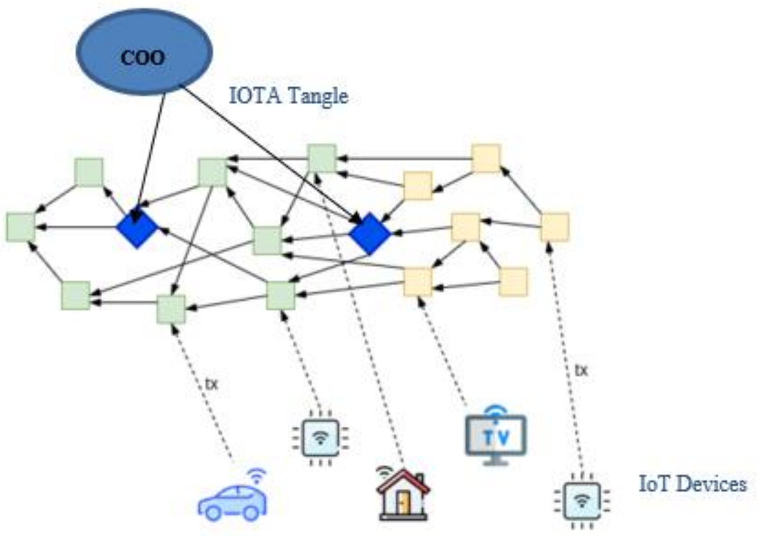


Figure 3: IoT devices communicating with IOTA Tangle [11].

In Figure 3, blue blocks represent signed milestones generated by Coordinator, which directly or indirectly confirms the previous transactions represented by green blocks, while the transactions in yellow are not yet confirmed [11].

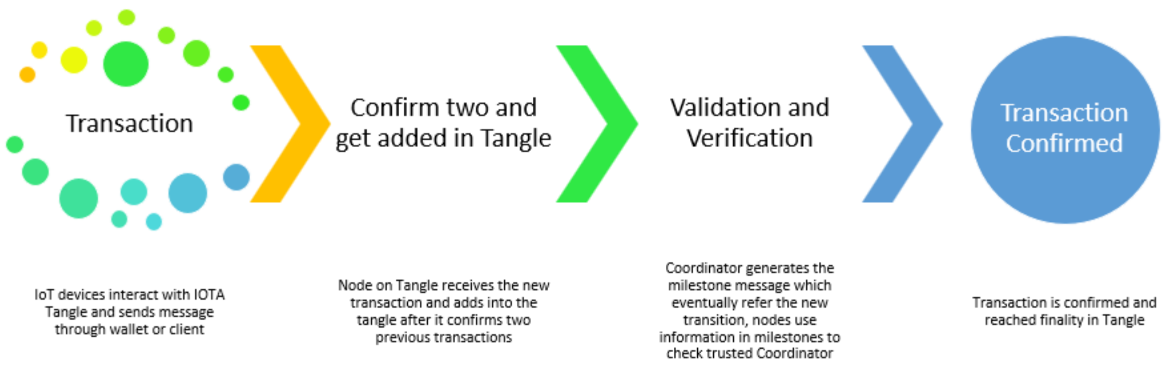

Transaction
Confirm two and get added in Tangle
Validation and Verification
Transaction Confirmed
IoT devices interact with IOTA Tangle and sends message through wallet or client
Node on Tangle receives the new transaction and adds into the tangle after it confirms two previous transactions
Coordinator generates the milestone message which eventually refer the new transition, nodes use information in milestones to check trusted Coordinator
Transaction is confirmed and reached finality in Tangle

Figure 4: Transaction flow in IOTA Tangle

Figure 4 shows the stages involved in the data flow, which is initiated from an IoT device and stored immutably on the Tangle. This stored data over the Tangle is securely saved and can be fetched by connecting a client to the Tangle node and presenting the necessary credentials (messageid) for the required data and the node will trace the data pack and send it back to the client. After the discussion on the workings of IOTA Tangle, the next sub-chapter will focus on the types of messages that transactions can carry across the ledger and how the structure of these messages assists IOTA Tangle in managing transactions for sending or retrieving data. Practical implementation of the IOTA sender client and subscriber client will be discussed in Chapter 3.

2.4.2 IOTA Messages

In IOTA Tangle, there are two types of messages, The first one carries value such as a crypto asset or an IOTA token and the second one is the non-value message, which contains no value but only data. Value-type transactions are used between two parties in exchange for assets while non-value or zero-value transactions are mostly used by data storage applications like the one this study will demonstrate later. When a client prepares a message to send to Tangle, it creates a message label that contains information to inform the node about the type of message and the payload so that node can differentiate the message from others and know what to do, the details of these messages elements are as follow.

2.4.2.1 Network ID

There are several individual IOTA networks running independently of each other like Mainnet (Chrysalis), Chrysalis Devnet, Shimmer, and Testnet. The purpose of these networks is different for instance Mainnet is the most stable network while Chrysalis Devnet and Testnet are used by the IOTA community for development and testing purposes. The IOTA message is equipped with a Network ID, which assists nodes in determining whether the message is part of the network they are currently operating in [4].

2.4.2.2 Message ID

The client application also creates a unique cryptographic hash and incorporates it into the message, which helps the node to trace the transaction when some other client tries to retrieve it from the Tangle [4].

2.4.2.3 Parents

The number and identification of the messages referenced by the new messages are referred to as parents' length and parents' ID. To establish the graph structure of the Tangle, each new message in the Tangle must refer to one to eight preceding messages. The node selects these two messages and sends the IDs to the client, and the client must include this information in the message label. Thus, nodes ensure that the Tangle's data structure evolves by the protocol [4].

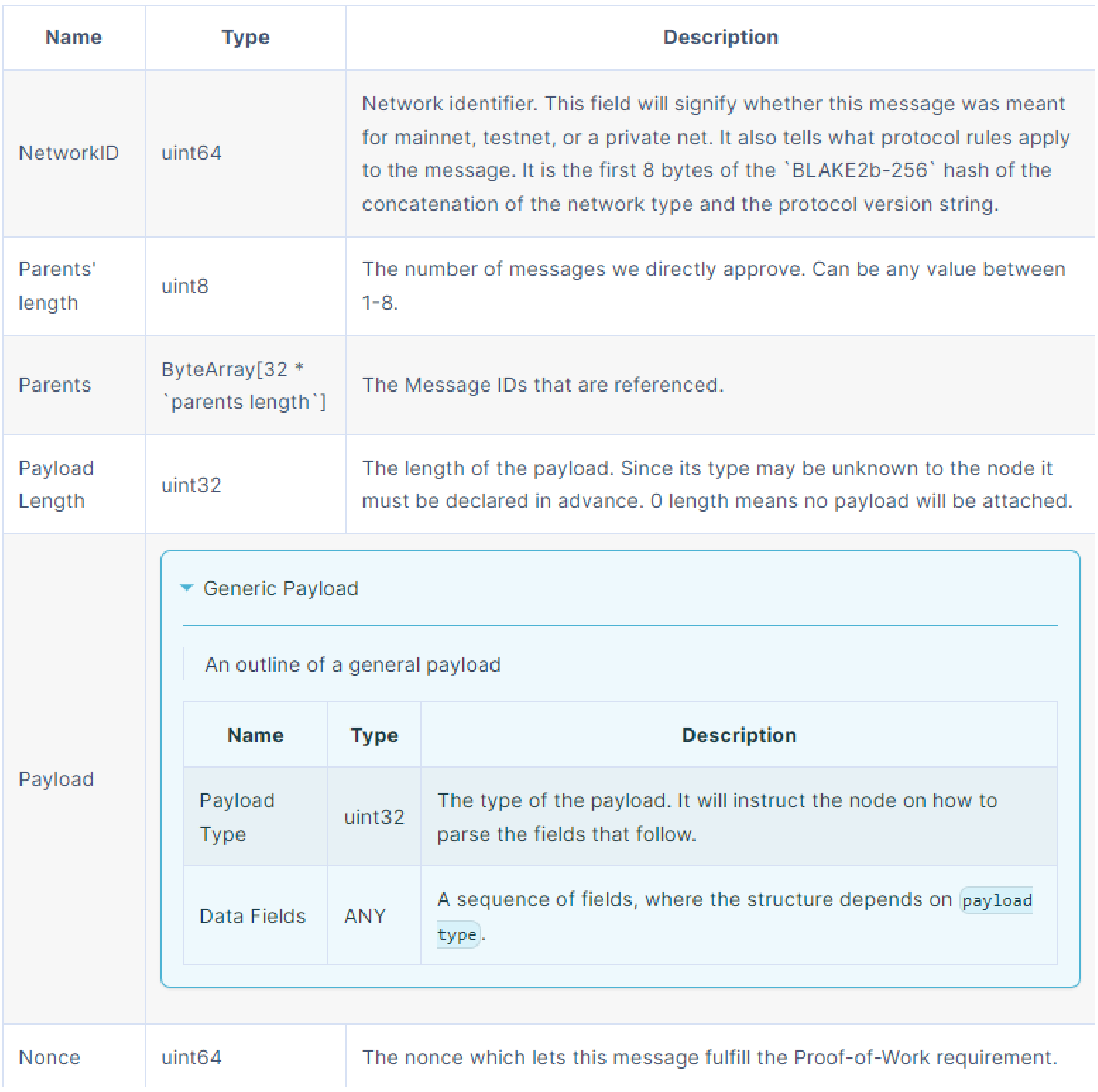
2.4.2.4 Payload

IOTA message usually contains a payload. As of 2023, there are three types of payloads allowed: Mainnet Transaction payload, Indexation payload, and Milestone payload. These payloads can be mainly categorized as valued payloads and non-valued payloads since it is only the transaction payload that contains the actual data, carrying valued information about the sender, receiver, and amount of IOTA tokens being transferred. Indexation payload only carries non-valued data which can be string, integers, or any supported datatype packed and sent according to Tangle rules, while the Milestone payload handles information essential to confirm previous transactions and help them reaching to finality. Processing the data over the ledger requires that the payload must be within the allowed range of 32kb, and its type should be mentioned by using Payload Length and Payload Type elements in the message structure [4].

2.4.2.5 Nounce (Number Used Once)

This is a randomly generated number by the node to create a hash of the transaction, which is complex enough to satisfy the proof of work requirements for adding a transaction to the Tangle [4].

Figure 5 explains the elements in the IOTA Message structure which is important to understand how to implement any client/subscriber application using the Tangle [4].



Name	Type	Description
NetworkID	uint64	Network identifier. This field will signify whether this message was meant for mainnet, testnet, or a private net. It also tells what protocol rules apply to the message. It is the first 8 bytes of the `BLAKE2b-256` hash of the concatenation of the network type and the protocol version string.
Parents' length	uint8	The number of messages we directly approve. Can be any value between 1-8.
Parents	ByteArray[32 * `parents length`]	The Message IDs that are referenced.
Payload Length	uint32	The length of the payload. Since its type may be unknown to the node it must be declared in advance. 0 length means no payload will be attached.
Payload	Generic Payload — An outline of a general payload	
Nonce	uint64	The nonce which lets this message fulfill the Proof-of-Work requirement.

Generic Payload

An outline of a general payload

Name	Type	Description
Payload Type	uint32	The type of the payload. It will instruct the node on how to parse the fields that follow.
Data Fields	ANY	A sequence of fields, where the structure depends on `payload type`.

Figure 5: Message structure of IOTA Message [4].

2.5 Review of Blockchain, Vechain, and IOTA for IoT Applications

This section provides an analysis of the advantages and limitations of Blockchain [3], IOTA [4], and Vechain [5] individually. Each distributed ledger technology is examined in terms of its specific strengths and weaknesses. Furthermore, a comparative evaluation is presented based on existing research, which is also referenced for further reading.

2.5.1 IOTA for IoT application

The research paper "Blockchain Technologies for IoT Applications: Use-cases and Limitations" [13] provides a survey of different DLTs and discusses their advantages and limitations in IoT applications.

The paper evaluated DLTs on their capability to solve five major issues for IoT, which are security, data privacy, identity management, trust and governance, and fault tolerance. Though most DLTs seem to address the aforementioned issues, they lack the amount of scalability and cost-effectiveness per transaction, which are required by IoT applications. The paper mentioned a particular example of public DLTs, Bitcoin and Ethereum, which are too slow and have non-negligible transaction fees making them unrealistic for larger IoT applications. Sensing the lack of compatibility of DLTs towards IoT applications, the IOTA foundation came up with a DAG-based DLT the Tangle, remarkably enhancing the throughput of transactions, and low computational PoW, which as a result offer no-fee transactions. IOTA Tangle is promising, But for its fast and fee-less structure, it needs a lot of node contribution within the network, which is not the case today, where the Tangle uses a temporarily centralized solution called Coordinator. Fully decentralized DAG still has not been functional in practice and is currently under the research and development phase.

The research paper argues that private DLTs could be the solution for the successful use of DLTs in IoT applications. Private DLTs eliminate the need for complex consensus algorithms as all nodes are governed by a single entity and rules are defined. Nodes can be trusted, and data on the ledger can be managed using Proof of Authority (PoA), similar to Vechain DLT.

Overall, the paper concludes that blockchain technology has the potential to revolutionize the way IoT devices interact and share data. However, the limitations of the technology must be carefully considered and addressed to ensure its successful integration into IoT applications.

2.5.2 Vechain for IoT application

Vechain [5] is DLT specifically designed for supply chain management system and IoT applications. Vechain offers several features highly required by IoT applications. It provides end-to-end transparency in supply chain process making it easier to track a product from start to the end. Vechain is efficient and have automated mechanisms in place to reduce overall processing time on network (RFID technology and smart contracts). It can handle many transactions per second making it suitable for high throughput applications. Alongside the advantages shared with other blockchain technologies, Vechain differentiates itself by providing a governance model and costing system using two-token design which makes it attractive for businesses and organizations. [5], [13].

However, the benefits of Vechain do come with a trade-off between advatanges and limitations. These features make Vechain a complex platform, which needs investment and management. Vechain loses its decentralization while providing governance, which may raise concerns about trustworthiness. To use Vechain for transactions, organizations need to use VeThor tokens as a means of payment. This means that businesses are dependent on VeThor tokens to use Vechain [5], [13].

Overall, Vechain does have potential to change the way data handling in IoT applications work today, but it highly depends upon the benefits contra dependency weighing scale of organization or businesses.

2.5.3 Blockchain for IoT application

Blockchain [3], provides excellent security and immutability for IoT applications through its highly decentralized consensus. It also eliminates the requirement of any centralized entities through the use of private and public blockchains. Blockchain, through its trustless network, can be leveraged for data transactions in IoT applications. It also enables access control policies and ensures data confidentiality and integrity through the possibility of implementing smart contracts.

While blockchain offers benefits for IoT applications such as security and immutability, it also poses challenges related to scalability, energy efficiency, privacy, and device identification. The Proof of Work (PoW) consensus method, which is commonly used in blockchain, can be complex for IoT transactions and hinder scalability as IoT devices are usually low-capability, battery-powered devices. Additionally, the increasing energy requirements of blockchain transactions are not sustainable for IoT nodes. Privacy is another challenge posed by the use of blockchain in IoT applications, as all transactions are shared, and IoT devices may reveal private user data. While private blockchains may mitigate the privacy issue, they do not guarantee accountability since they are not sufficiently decentralized [9], [10].

2.5.4 Comparison of Blockchain, Vechain, and IOTA for IoT Applications

Based on the existing research studies, [5], [9],[10], [13], and [15] the performance of DLTs such as Blockchain, Vechain, and IOTA Tangle can be compared in terms of various factors including scalability, transaction speed, consensus algorithm, data privacy, energy efficiency, and use case.

In terms of scalability, Blockchain has limitations due to its linear structure, where the addition of new blocks can lead to increased transaction confirmation time and resource requirements. Vechain and IOTA, on the other hand, offer high scalability and allow for parallel processing of transactions.

Transaction speed is another important aspect to consider. Blockchain, particularly in public networks like Bitcoin, has relatively slow transaction speeds due to the PoW consensus algorithm, which requires significant computational resources. Vechain, using the PoA consensus algorithm, achieves faster transaction speeds compared to Blockchain. IOTA Tangle also offers high transaction and overall confirmation time compared to the Blockchain [17].

Consensus algorithms play a crucial role in the energy efficiency of a DLT. Blockchain relies on PoW, which involves solving complex mathematical problems, leading to high energy consumption. Vechain's PoA consensus algorithm, on the other hand, improves energy efficiency by relying on a limited number of trusted nodes. IOTA Tangle utilizes a consensus mechanism through Coordinator, aiming to achieve high energy efficiency and scalability [4], [5], [13].

Data privacy is an important consideration in DLTs. Blockchain offers partial data privacy, as transactions are transparent and visible to all participants, although the identities of users can remain protected. Vechain, focusing on supply chain and enterprise solutions, provides full data privacy through its permissioned blockchain architecture. IOTA Tangle offers partial data privacy, with transaction information being public but user identities remaining hidden [5], [9], [10], [13], [15].

The following Table 1 compares the key features of three popular DLT technologies: IOTA Tangle, Vechain Blockchain, and Bitcoin Blockchain. These features include consensus algorithm, transaction fees, scalability, transaction speed, data privacy, energy efficiency, and use case focus.

Key Features	IOTA Tangle	Vechain Blockchain	Bitcoin Blockchain
Consensus Algorithm	DAG (Directed Acyclic Graph)	PoA (Proof of Authority)	PoW (Proof of Work)
Transaction Fees	No transaction fees	Transaction fees depend on usage	Transaction fees depend on usage
Scalability	High scalability	High scalability	Limited scalability
Transaction Speed	High transaction speed	High transaction speed	Relatively slow transaction speed
Data Privacy	Partial data privacy	Full data privacy	Partial data privacy
Energy Efficiency	Highly energy-efficient	Energy-efficient	Relatively energy-intensive
Use Case Focus	IoT data storage and machine economy	Supply chain and enterprise solutions	Store of value and peer-to-peer payment

Table 1: Comparison among Tangle, Vechain and Blockchain DLTs [5], [9], [10], [13], [15].

Despite the promising features exhibited by these DLTs, it is important to acknowledge the limitations highlighted in existing research studies. While these studies have identified significant performance indicators for DLTs in the context of IoT applications and drawn conclusions regarding the superiority of one DLT over another, there remains a critical need for practical performance comparisons among these DLTs. Particularly in IoT scenarios where real-time data and efficient resource utilization are crucial, it is essential to investigate whether these DLTs can meet the requirements of IoT devices with limited processing power and demands for real-time data storage and retrieval. Although a DLT may demonstrate relative efficiency and cost-effectiveness when compared to other DLTs, its real-world performance in actual applications necessitates further research. Future studies should focus on evaluating the performance of these DLTs in practical IoT settings to determine their suitability and effectiveness.

3 System Analysis and Implementation

This chapter focuses on the practical approach used to showcase the data storage capabilities of IOTA Tangle, a Distributed Ledger Technology. It starts with an overview of the system used in this study, followed by detailed specifications and use-cases. The chapter then delves into the system architecture and development process in the implementation section. Additionally, a testing and validation methodology will be proposed to ensure the accuracy and effectiveness of the system.

3.1 Process System Overview

Consider a dairy facility that produces milk products and operates multiple Pasteurizers, which are the heart of dairy production lines, as nearly every product goes through a Pasteurizer before reaching its final stage. Figure 6 illustrates a simplified industrial process of a milk pasteurizer. The process begins with milk entering the pasteurizer, where it passes through pasteurizer units responsible for heating it to a specific temperature. The milk then proceeds to the holding cell, which maintains the heated temperature for a predetermined time period and regulates the flow based on its design. After leaving the holding cell, the milk is cooled down before being stored in tanks for use in production. To ensure the highest product quality, the pasteurizer is designed with specific capacity and parameters that carefully control the temperature, flow, and pressure of the milk throughout the entire pasteurization process.

Optimizing the pasteurization process can greatly benefit manufacturers. Third-party pasteurizer optimization service providers may require access to the pasteurizer data for performing data analysis. By analyzing this data, they can identify opportunities to optimize the process and enhance overall production efficiency, thereby adding value to the final product. Data should be strictly secure and stakeholders like production managers and operators should have access to Data and analysis results through a web portal. There should be a secure gateway to make data available to the third party using one of various field communication protocols like Modbus, OPC UA or a local History Server, deployed within the control net. The method of data acquisition is out of the scope of this study, and a set of static data is utilized instead. This static data mimics the process data of an operational milk Pasteurizer. Parameters Temperature, Holding Time, Flow Rate, Pressure, pH Value, and Cooling Rate can be considered Key Performance Indicators (KPIs) for monitoring the product quality and overall performance of a milk Pasteurizer.

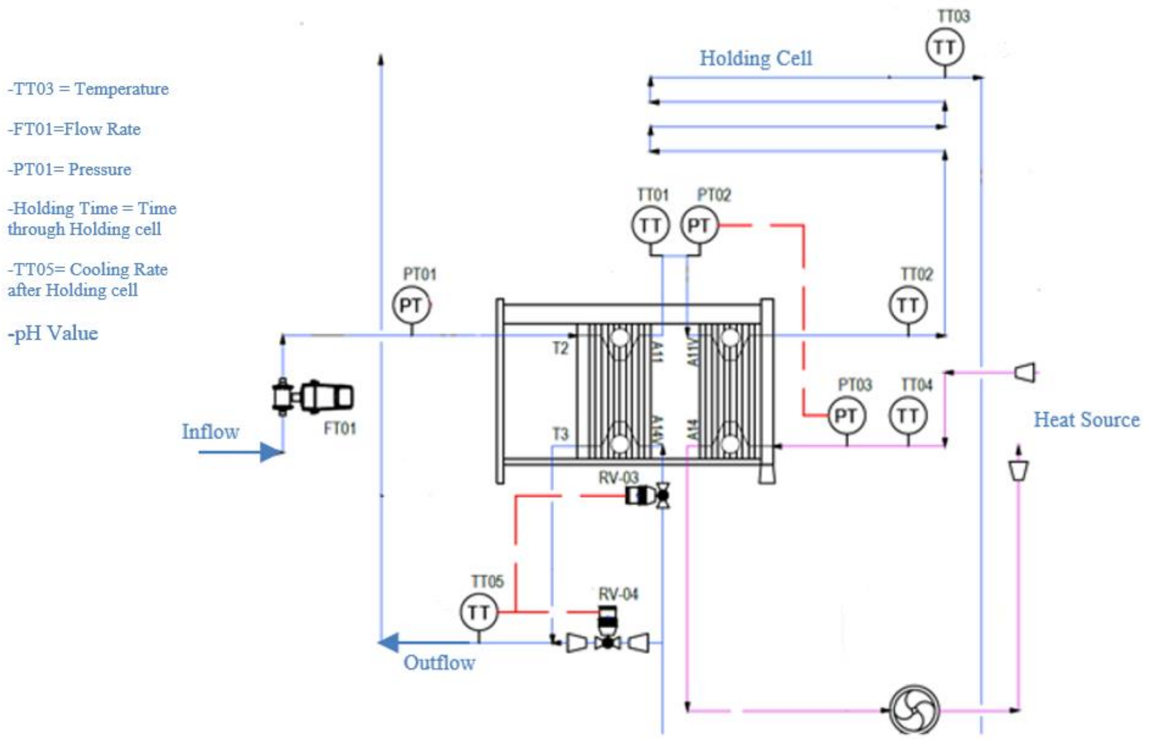


Figure 6: Pasteurizer process diagram.

3.2 System Specifications

Considering a typical scenario explained in section 3.1, the specifications of the core system are as follow:

1. The developed system should store process data on the Distributed Ledger the Tangle.
2. Data should be accessible through a secure web portal.
3. The system should permit tracking of the process parameter name, parameter value, time-stamp, and unit of measurement.
4. The system should give the service provider and the client access to see the data stored on the Tangle and reports of data analysis.
5. Data analysis should provide a visual explanation of any possible diagnosis and current state of Pasteurizer performance and allow clients to adjust in their production processes accordingly.

3.3 System Requirements and Use Case

This section summarize the main requirements of the system, which are further used to build the use case diagram.

3.3.1 Requirements

Considering the specific scenario explained in Chapter 3.1, and the system specification explained in Chapter 3.2, the system's requirements are defined based on the FURPS+ method as follow:

- Functionality:
    - Read/Store Process data: The process data must be stored in the distributed ledger the IOTA Tangle.
    - Access data: Users such as service provider companies and clients at the dairy facility can read the data and data analysis reports.
    - Diagnosis: The service provider company should be able to diagnose process errors or anomalies in the process, which should hinder the loss of product or marginalize the product quality.
- Usability:
    - Data should be presented in an easily accessible user interface like a web page.
    - Data analysis reports should be intuitive and graphically present the status of the process.
- Reliability:
    - The script should be running continuously without unhandled exceptions.
    - Web portal should be light enough to be reached in case of low bandwidth internet connection.
    - Application scripts should work across platforms in case of machine change.
- Performance:
    - None.

- Supportability
    - Data should be accessible using web devices running on any platform.
- +
    - None.

3.3.2 Use case Diagram

The use case diagram describes the main requirements discussed earlier in chapter 3.3.1.

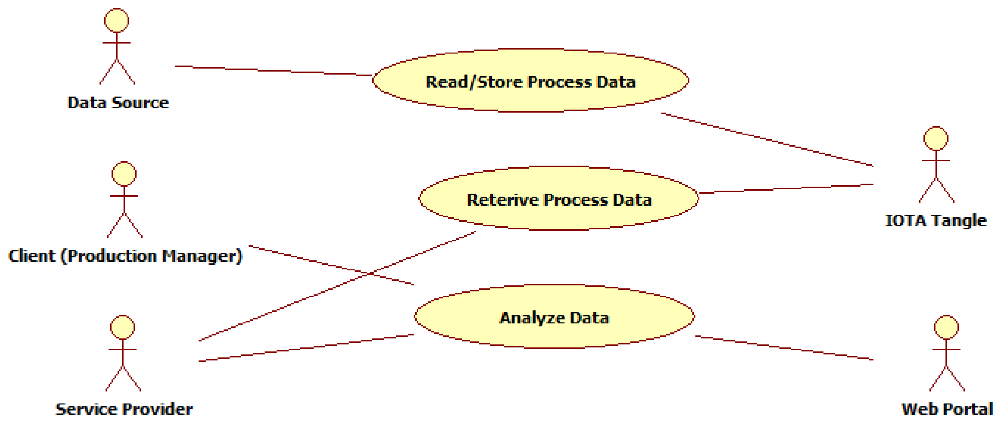


Figure 7: Use case diagram of the application.

3.4 System Sequence Diagram

The use case diagram is being utilized here to model the behavior of the system from the users' perspective. To do so, each use case is analyzed further using System Sequence Diagram (SSD).

3.4.1 Read/Store Process Data

Figure 8 shows the SSD for the "Read/Store Process Data" use case analysis. In this use case, the application is in run mode and imports the data from the data source, and checks if the data is not empty and valid. Then it extracts the data into data lists and categorizes the data into required parameters, followed by the encoding of data using Unicode Transformation Format (UTF-8) to convert it into bytes, as IOTA Tangle accepts data in byte format. Client connection is initiated towards the IOTA node and message details are required to the node to make a transaction and successfully store the data on Tangle. Once data is successfully stored, the node will provide the Tangle parameters associated with the data or else give a connection error if there is no response.

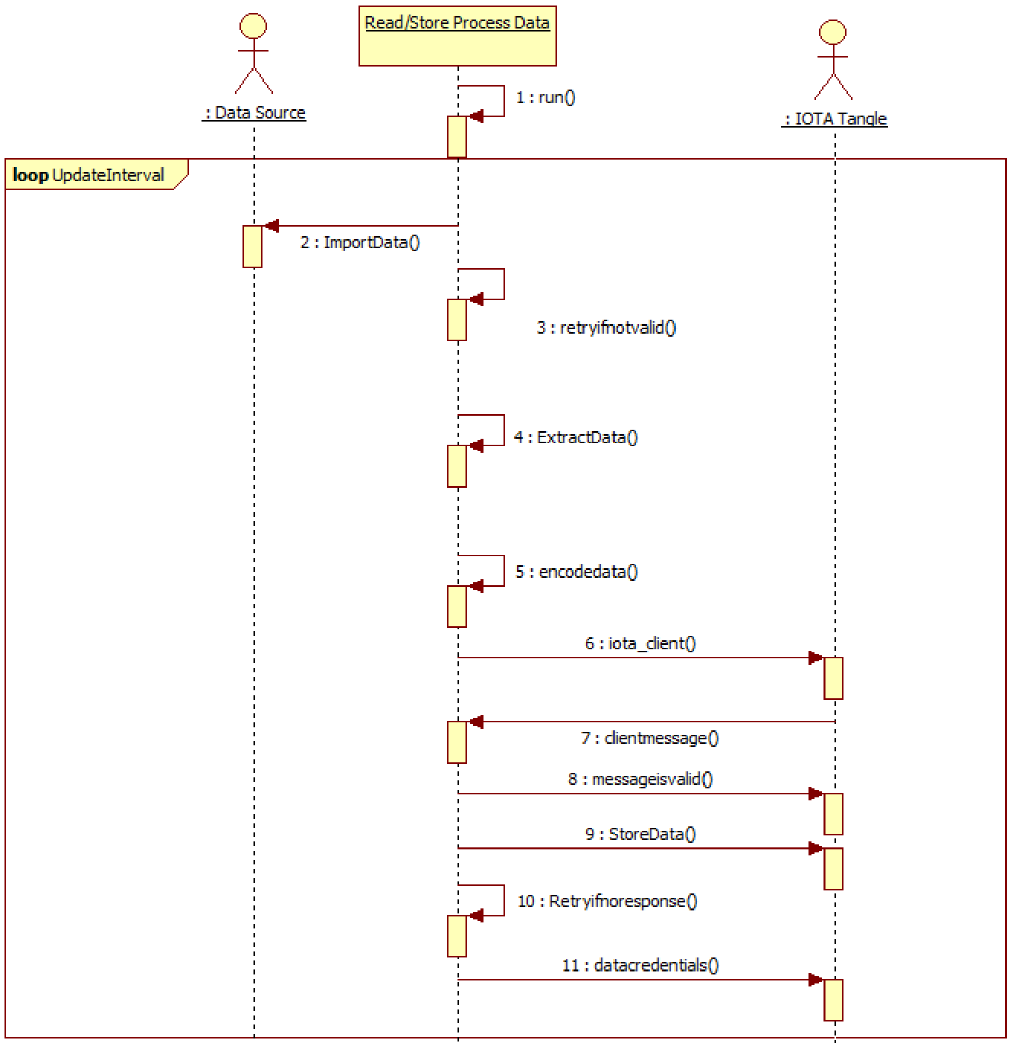


Figure 8: SSD for Read/Store Process Data Use case.

3.4.2 Retrieve Process Data

Figure 9 explains the functionality of the use case “Retrieve Process Data”. Firstly, the client connection is initiated with the Tangle Node, and a request for retrieving data is made by providing credentials like message ID previously received by the node at the time of successful data storage. As a message is received, data should be extracted from the indexation payload within the message and decoded from bytes form to decimal form to further use in the application.

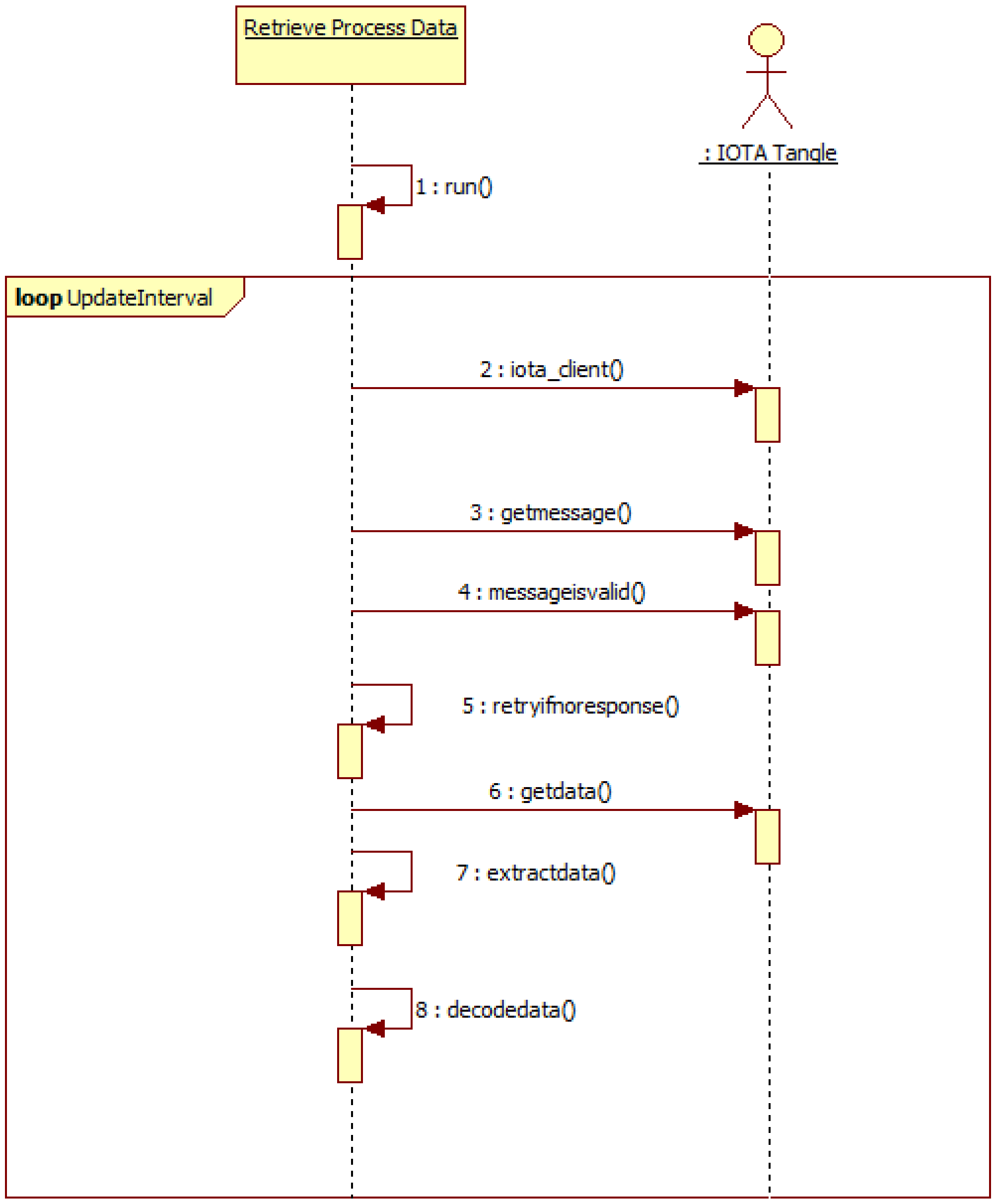


Figure 9: SSD for Retrieve Data use case.

3.4.3 Analyze data

Figure 10 shows an analysis of the third use case “Analyze Data”. After data is retrieved from the tangle, the main functionality of this use case is to perform data analysis by implementing required functions and making it available for clients and service providers through a web portal.

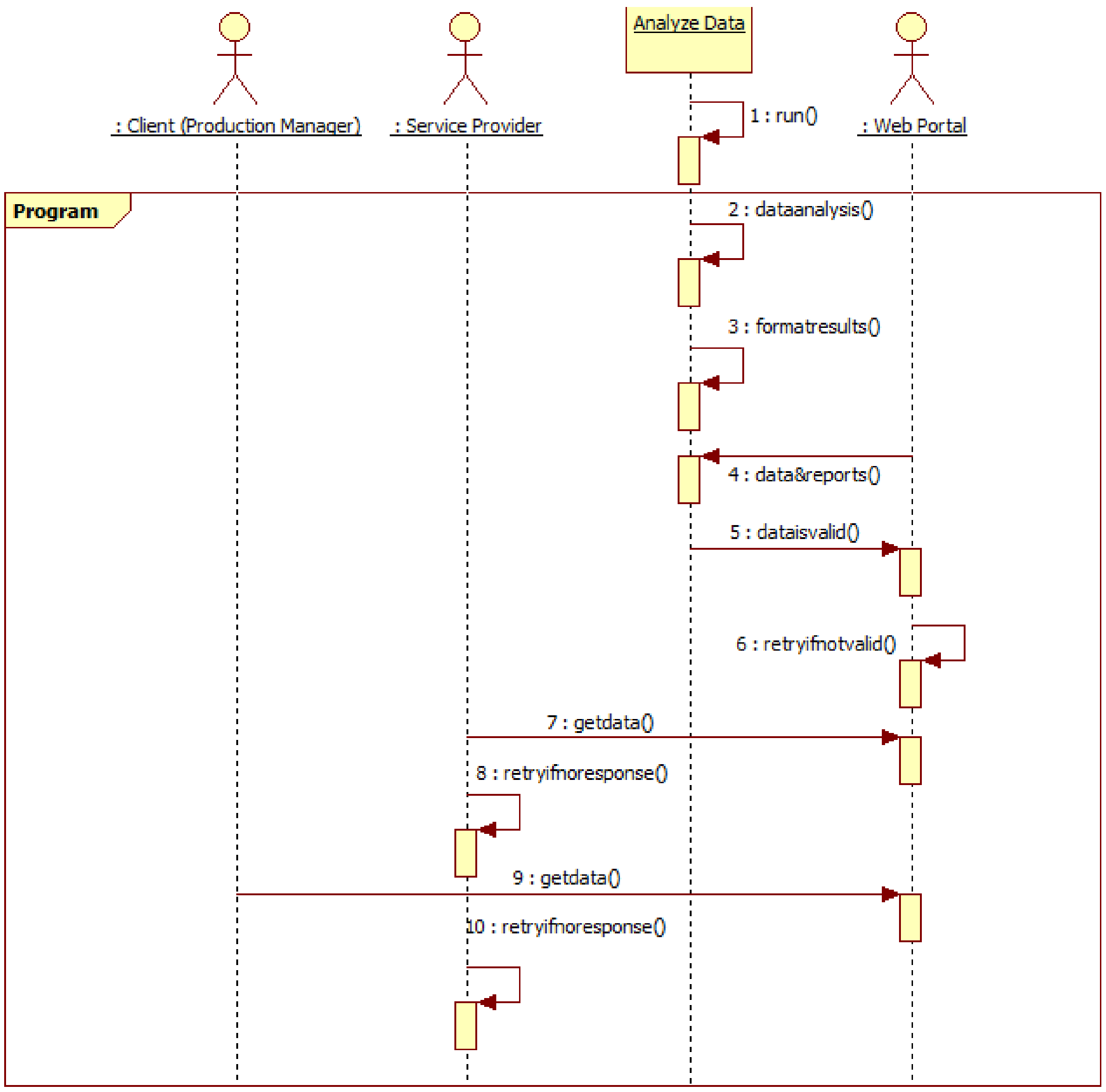


Figure 10: SSD for Analyze Data use case.

3.5 System Architecture

Figure 11 depicts a higher-level system overview of the application in which the data should be acquired as per application requirement, then the data is to be packed and sent from an IOTA client to a node on the Tangle, which should store the data on multiple nodes across the network. An IOTA subscriber should be responsible for querying the Tangle for the stored data and converting it to a usable form for further programming or data analysis. The retrieved data will be presented on Graphical User Interface (GUI), allowing the user to understand and analyze the data. A more detailed system overview along with software and tools used for each layer will be discussed in this chapter.

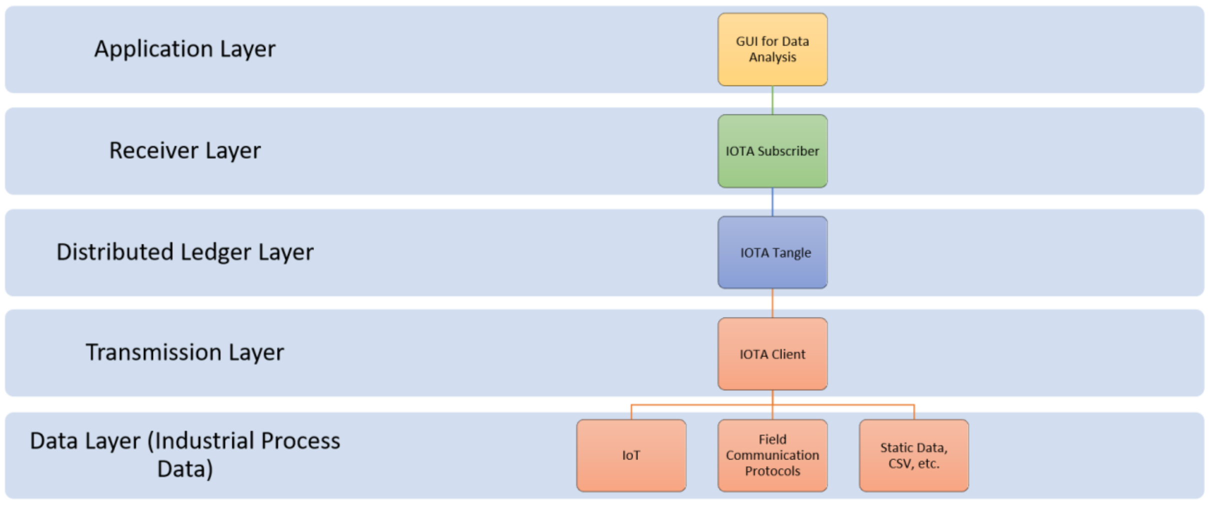
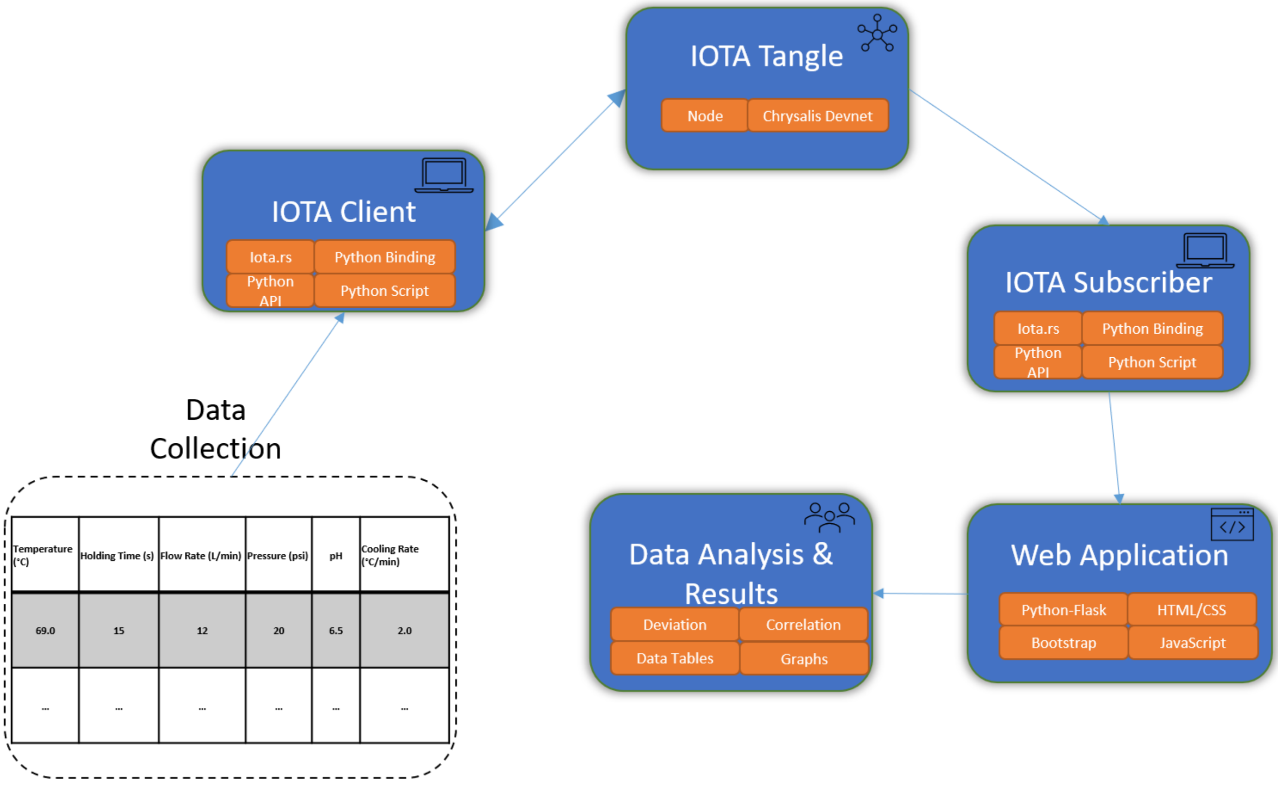


Figure 11: System Overview for data storage and analysis using IOTA Ledger.

3.6 System Implementation

Figure 12 depicts a working overview diagram of the system. The transfer of data from an IOTA client to an IOTA subscriber is carried out through IOTA node on Chrysalis Devnet, and data analysis and interpretation of data is accessible on a web app which is built using the Python-Flask web-framework. IOTA client, IOTA Node and subscriber will be discussed in detail in following section whereas the data analysis and results will be discussed in Chapter 4.



Temperature (°C)	Holding Time (s)	Flow Rate (L/min)	Pressure (psi)	pH	Cooling Rate (°C/min)
69.0	15	12	20	6.5	2.0
...	...	...	...	...	...

Figure 12: System Overview of data storage application.

3.6.1 IOTA Client

In IOTA client, the IOTA protocol is used to store data over the Tangle ledger on Chrysalis Devnet [4] that will be explained later in the following chapter. IOTA client is built in Python using the iota.rs library [4], which is the RUST implementation of the IOTA protocol. The following are dependencies and tools used in the development of IOTA client.

3.6.1.1 iota.rs:

Iota.rs [4] is a library that implements the real IOTA protocol in Rust programming language, and it is the only language IOTA protocol is written in. This library provides a high-level Application Programming Interface (API) for developers to interact with the IOTA network and perform various transactions such as sending and receiving IOTA tokens, as well as sending data to the IOTA Tangle [4].

3.6.1.2 Python bindings:

Python binding is a bridge between Rust code (iota.rs) and Python code. This allows developers to use the iota.rs library in Python code to interact with the IOTA protocol. In order to work properly Python binding should be installed along with required iota.rs dependencies in the development computer. A detailed procedure for installation is available at [4].

3.6.1.3 Python APIs:

Python APIs are interfaces for the IOTA protocol available in Python library iota_client that allow the developer to interact with an application or a library using Python. There are many node-level and high-level APIs available in Python with detailed functions and parameters which can be accessed here [4].

3.6.1.4 Python IDE:

Integrated Development Environment (IDE) that provides a comprehensive environment for developers to write, test, and debug Python code. In the development of the IOTA client, Spyder IDE [18] is used.

3.6.1.5 Code for IOTA Client

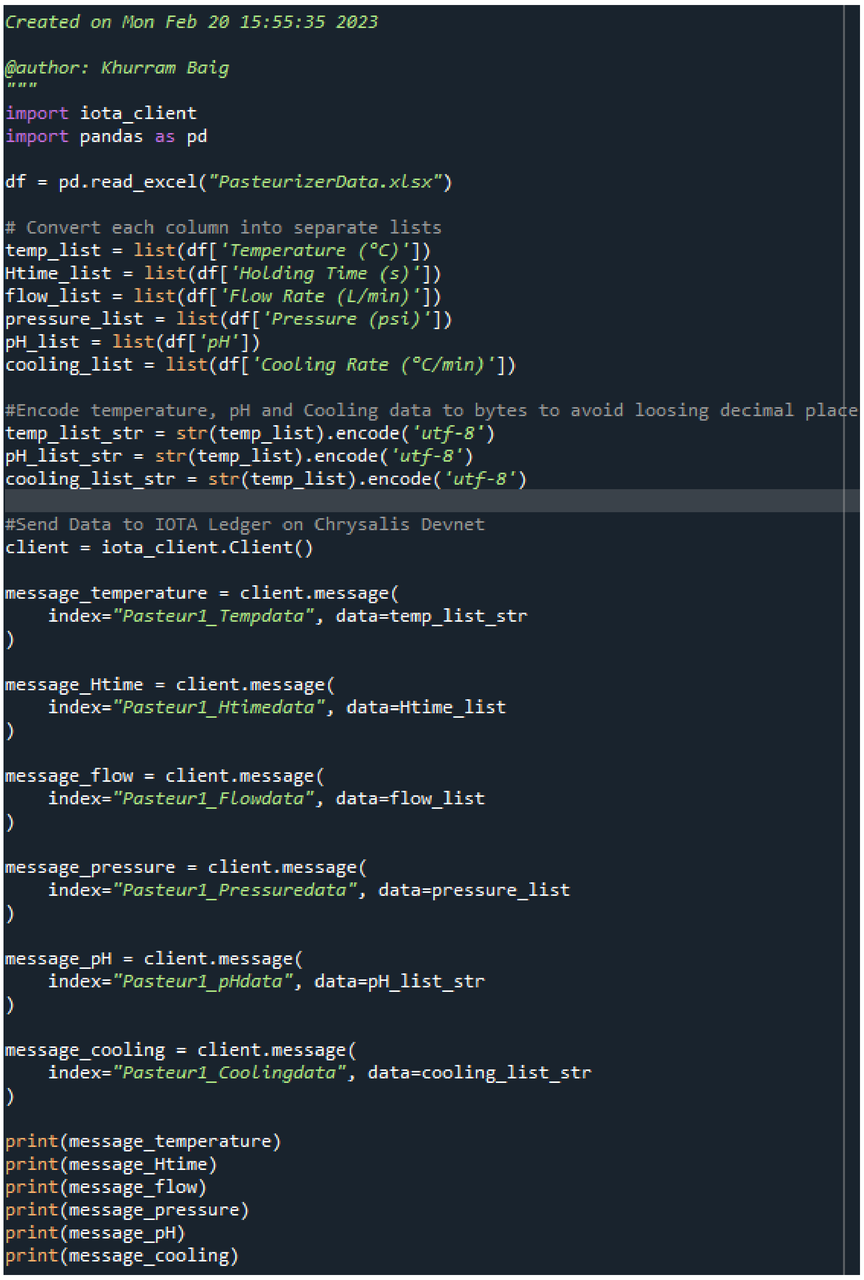
Figure 13 is a code snippet of IOTA Client which is written to integrate the main functionalities of the client explained earlier. The code reads an Excel file called "PasteurizerData.xlsx" (explained in Chapter 3.1) using the pandas library [19], and stores the data into a pandas data frame “df”. Then, it converts each column into separate lists, namely temp_list, Htime_list, flow_list, pressure_list, pH_list, and cooling_list.

The next step is to encode before sending the data to the IOTA ledger, this requires to encode the data into bytes. In the code provided above, the temperature, pH, and cooling rate data are encoded into bytes using UTF-8 encoding.

Finally, the data is sent to the IOTA ledger on Chrysalis Devnet using the iota_client module. The Client() method from this module is used to create an IOTA client instance. This instance is then used to send messages to the IOTA Tangle.

For each list of data, a message is created using the message() method of the client instance. Each message is given a unique index to identify the data and is associated with the corresponding list of data. The message() method returns a message object that contains the transaction ID, which can be used to retrieve the data from the Tangle later.

Finally, the transaction IDs of all the messages are printed to the console using the print() function. These IDs can be used to retrieve data from the IOTA ledger in the future.



```python
Created on Mon Feb 20 15:55:35 2023

@author: Khurram Baig
"""
import iota_client
import pandas as pd

df = pd.read_excel("PasteurizerData.xlsx")

# Convert each column into separate lists
temp_list = list(df['Temperature (°C)'])
Htime_list = list(df['Holding Time (s)'])
flow_list = list(df['Flow Rate (L/min)'])
pressure_list = list(df['Pressure (psi)'])
pH_list = list(df['pH'])
cooling_list = list(df['Cooling Rate (°C/min)'])

#Encode temperature, pH and Cooling data to bytes to avoid loosing decimal place
temp_list_str = str(temp_list).encode('utf-8')
pH_list_str = str(temp_list).encode('utf-8')
cooling_list_str = str(temp_list).encode('utf-8')

#Send Data to IOTA Ledger on Chrysalis Devnet
client = iota_client.Client()

message_temperature = client.message(
    index="Pasteur1_Tempdata", data=temp_list_str
)

message_Htime = client.message(
    index="Pasteur1_Htimedata", data=Htime_list
)

message_flow = client.message(
    index="Pasteur1_Flowdata", data=flow_list
)

message_pressure = client.message(
    index="Pasteur1_Pressuredata", data=pressure_list
)

message_pH = client.message(
    index="Pasteur1_pHdata", data=pH_list_str
)

message_cooling = client.message(
    index="Pasteur1_Coolingdata", data=cooling_list_str
)

print(message_temperature)
print(message_Htime)
print(message_flow)
print(message_pressure)
print(message_pH)
print(message_cooling)
```

Figure 13: Code for IOTA Client.

3.6.2 IOTA Node

IOTA node [7] is a gateway between IOTA Client and the Tangle network. It is the node in the Tangle network which receives or sends messages to and from the client. Client connects to the node via the iota_client library, which provides the Python APIs to communicate with the node. For instance, node APIs let clients inquire about the health of the node, addresses, network information or can even let it know the IOTA protocol specific parameters like milestones.

3.6.2.1 Code for IOTA Node

In the code snippet, that is illustrated in Figure 13, the iota_client.client is used to create an instance of IOTA client, which connects to a node according to IOTA protocol written in RUST. The client connects to a node by default on the Chrysalis Devnet, which was previously called testnet before the Chrysalis update. Client instance in Python allows to connect to nodes on other networks or even to private network but in this demonstration of applications holds to the recommendation from IOTA community to develop and test application in Devnet.

Once the client is connected to the node, it can use the client. message() method to send the data packed in a message to the IOTA network. Each message contains an index which is used to identify the message on the IOTA network and data to be sent. The data parameter takes the list of the data as an argument, which is then encoded into bytes and sent to the node.

After a successful transaction of the message, the node returns an object that contains information about the message including messageid and transactionid, which will be discussed further in Chapter 4.

3.6.3 IOTA Subscriber

IOTA subscriber is a gateway on the receiver end, which is responsible for retrieving data from the Tangle and making it available in the Python environment for further development. IOTA subscribers use the main iota.rs library and Python APIs to interact with the Tangle and retrieve the data. Background tools and dependencies that are explained earlier in Chapter 3.6.1 are also valid for IOTA subscribers. Development computers, must provide this framework in order to communicate with the Tangle.

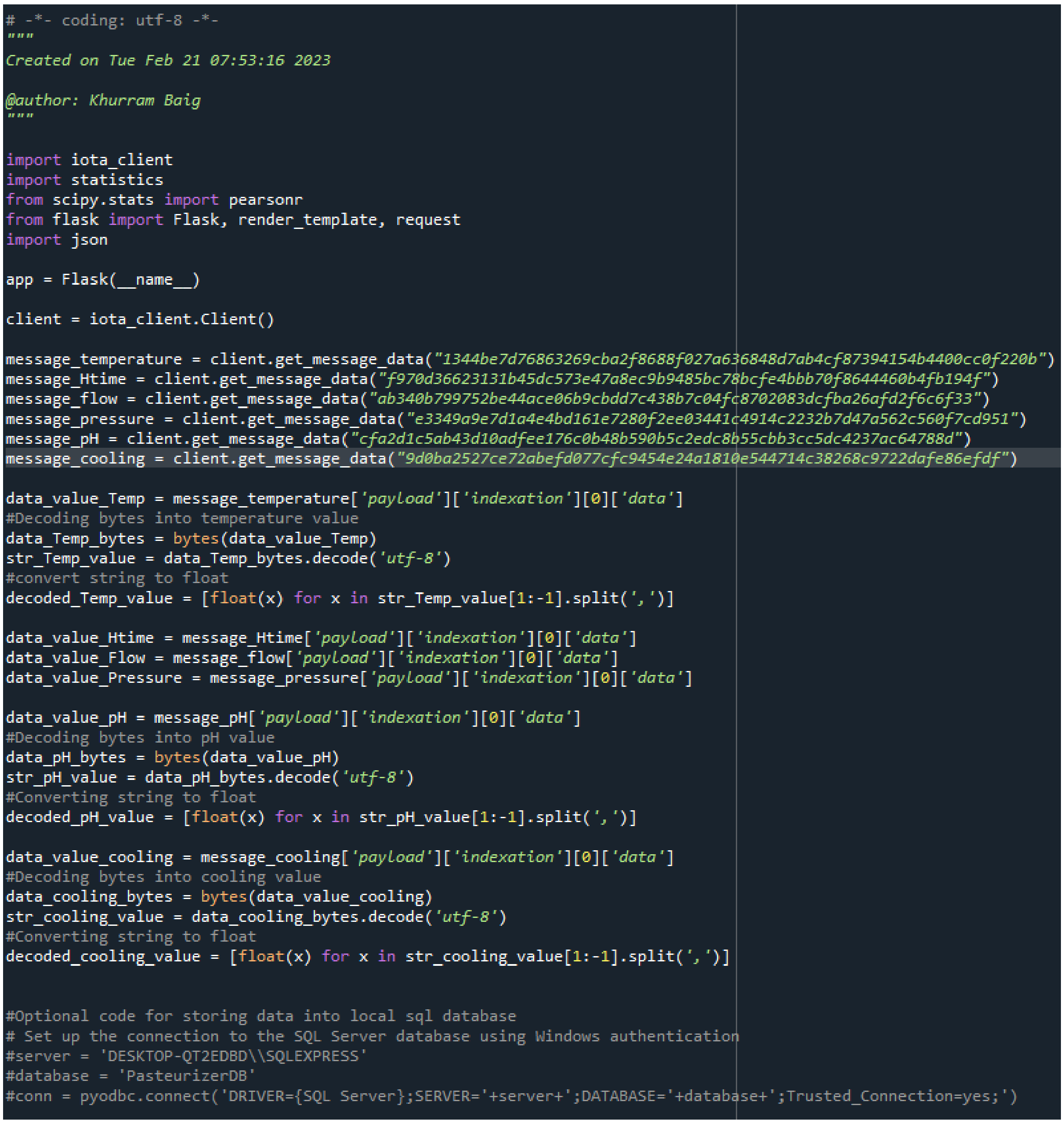
To retrieve and decode data from the IOTA Tangle, the following tools are used:

- iota.rs: IOTA protocol library built in RUST.
- Python bindings: Python binding of RUST library.
- Flask: A lightweight web framework for Python that provides a simple web server to host our web application and allows to integrate HTML into Python code.
- Python IDE: Spyder IDE.

3.6.3.1 Code for IOTA Subscriber

The code snippet, shown in Figure 14, is the implementation of the IOTA subscriber. To retrieve data from the IOTA Tangle, the iota.rs library's Client class is used. This allows get_message_data method to retrieve the data of a specific message on the IOTA Tangle. All the parameters from Pasteurizer data are stored on different nodes and hence have unique messageid. To retrieve the specific parameter data, get_message_data function is used and the relative messageid must be provided as an argument, as shown in the code.

Now that data is successfully retrieved, it is not yet useful and must be converted to compatible form. IOTA Tangle stores the data in binary format known as Trytes. In the example code provided, data from messages that contain Pasteurizer data, including temperature, Holding Time, flow, pressure, pH and cooling values are retrieved. To decode the data, we first extract the data from the message using the payload field, and then extract the indexation data using the indexation field. We then decode the indexation data from bytes to string using the UTF-8 decode method, and finally convert the string into a list of float values using the split method to use it in further development related to data analysis.



```python
# -*- coding: utf-8 -*-
"""
Created on Tue Feb 21 07:53:16 2023

@author: Khurram Baig
"""

import iota_client
import statistics
from scipy.stats import pearsonr
from flask import Flask, render_template, request
import json

app = Flask(__name__)

client = iota_client.Client()

message_temperature = client.get_message_data("1344be7d76863269cba2f8688f027a636848d7ab4cf87394154b4400cc0f220b")
message_Htime = client.get_message_data("f970d36623131b45dc573e47a8ec9b9485bc78bcfe4bbb70f8644460b4fb194f")
message_flow = client.get_message_data("ab340b799752be44ace06b9cbdd7c438b7c04fc8702083dcfba26afd2f6c6f33")
message_pressure = client.get_message_data("e3349a9e7d1a4e4bd161e7280f2ee03441c4914c2232b7d47a562c560f7cd951")
message_pH = client.get_message_data("cfa2d1c5ab43d10adfee176c0b48b590b5c2edc8b55cbb3cc5dc4237ac64788d")
message_cooling = client.get_message_data("9d0ba2527ce72abefd077cfc9454e24a1810e544714c38268c9722dafe86efdf")

data_value_Temp = message_temperature['payload']['indexation'][0]['data']
#Decoding bytes into temperature value
data_Temp_bytes = bytes(data_value_Temp)
str_Temp_value = data_Temp_bytes.decode('utf-8')
#convert string to float
decoded_Temp_value = [float(x) for x in str_Temp_value[1:-1].split(',')]

data_value_Htime = message_Htime['payload']['indexation'][0]['data']
data_value_Flow = message_flow['payload']['indexation'][0]['data']
data_value_Pressure = message_pressure['payload']['indexation'][0]['data']

data_value_pH = message_pH['payload']['indexation'][0]['data']
#Decoding bytes into pH value
data_pH_bytes = bytes(data_value_pH)
str_pH_value = data_pH_bytes.decode('utf-8')
#Converting string to float
decoded_pH_value = [float(x) for x in str_pH_value[1:-1].split(',')]

data_value_cooling = message_cooling['payload']['indexation'][0]['data']
#Decoding bytes into cooling value
data_cooling_bytes = bytes(data_value_cooling)
str_cooling_value = data_cooling_bytes.decode('utf-8')
#Converting string to float
decoded_cooling_value = [float(x) for x in str_cooling_value[1:-1].split(',')]


#Optional code for storing data into local sql database
# Set up the connection to the SQL Server database using Windows authentication
#server = 'DESKTOP-QT2EDBD\\SQLEXPRESS'
#database = 'PasteurizerDB'
#conn = pyodbc.connect('DRIVER={SQL Server};SERVER='+server+';DATABASE='+database+';Trusted_Connection=yes;')
```

Figure 14: Code for IOTA Subscriber.

3.6.4 Web Application

The code shown in Figure 15 is a part of a web application that analyzes and displays data collected from a Pasteurizer process. The application uses the Flask framework [20], which is a popular web framework written in Python, to provide server for the web pages and handle requests from the user. The web application consists of two pages in "table" and "dashboard" in which the former presents the stored data and data analysis functions on data like correlation and deviation and the latter presents data graphically in form of charts. The following are details on dependencies and tools used in the development of this web application.

3.6.4.1 Flask Python

Flask Python [20] is a lightweight web framework that provides easy to use interface for creating web applications within Python environment. Main functionality of Flask is to provide the developer set of tools for creating web application like running web server for handling HTTP requests, rendering .html templates within the Python environment and managing web sessions.

As IOTA client and subscriber are developed using Python bindings of the IOTA Protocol, Flask is the natural selection of web development part of this application since it directly integrates into Python and eliminates the need for integration towards any other web development platform.

3.6.4.2 HTML, CSS, JavaScript, and Bootstrap

The HTML and CSS templates are used to render the web pages and display the data. The templates include a table that displays the data, as well as several charts that show the trends in the data over time. The charts are created using JavaScript and the Chart.js library [21]., which is a powerful and flexible charting library that allows developers to create complex and dynamic charts with ease. The application also uses Bootstrap [22], which is a popular CSS framework that provides a set of pre-built components and styles for creating responsive web pages.

3.6.4.3 Web Application IDE

There are many choices available for coding web pages, in the development of this application, Visual Studio Code [23] is used as this is probably the most popular choice and provides syntax highlighting and code debugging features. It can also support Python programming and can be used as the sole IDE in this application development.

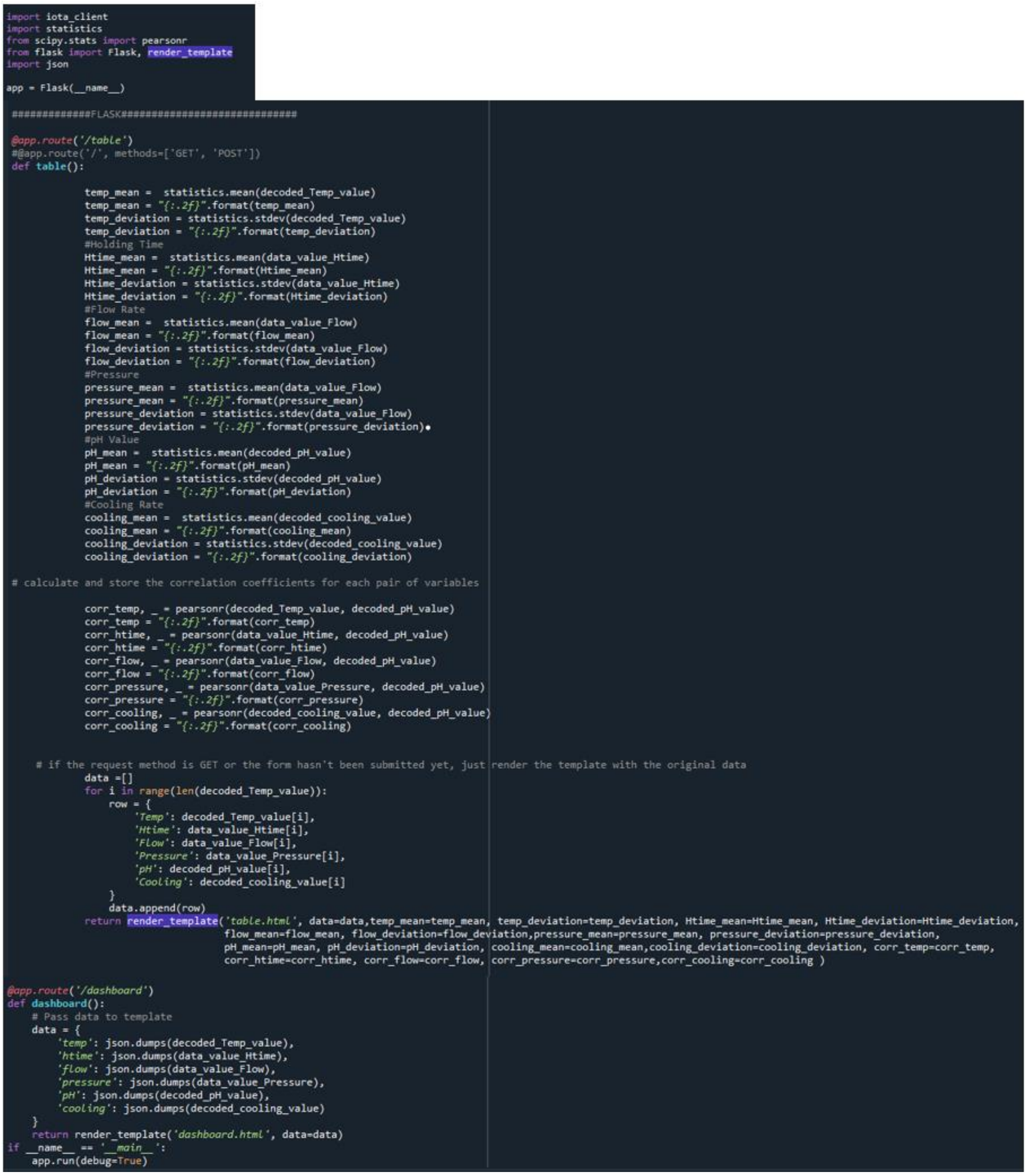
3.6.5 Code for Web Application

Figure 15 shows code implementation for the “table” and “dashboard” pages of the web application. It defines a route for the URL '/table' and a function called 'table'. The function retrieves some data from an external source, calculates statistical values (mean and standard deviation) for each parameter, and then calculates the correlation coefficient between the pH parameter and each of the other parameters, to show how every parameter is changing in relation to pH value of the product, which can provide an insight of product quality.

The function then renders a web page called 'table.html' using the data and calculated values. The web page displays a table of the data with the calculated statistical values and correlation coefficients for each parameter.

“/dashboard” Flask route function that renders a dashboard template with data passed to it. The dashboard function retrieves data from global variables decoded_Temp_value, data_value_Htime, data_value_Flow, data_value_Pressure, decoded_pH_value, and decoded_cooling_value, converts each list to a JSON string using the json. dumps method, and stores each string in a dictionary named data. The dictionary data is then passed to the dashboard.html template using the render_template method.

The if __name__ == '__main__' line ensures that the app.run method is called only when the script is run directly. The debug=True parameter sets Flask's debug mode to True, which can be useful for development but should not be used in production.



```python
import iota_client
import statistics
from scipy.stats import pearsonr
from flask import Flask, render_template
import json

app = Flask(__name__)

#############FLASK############################

@app.route('/table')
#@app.route('/', methods=['GET', 'POST'])
def table():

            temp_mean =  statistics.mean(decoded_Temp_value)
            temp_mean = "{:.2f}".format(temp_mean)
            temp_deviation = statistics.stdev(decoded_Temp_value)
            temp_deviation = "{:.2f}".format(temp_deviation)
            #Holding Time
            Htime_mean =  statistics.mean(data_value_Htime)
            Htime_mean = "{:.2f}".format(Htime_mean)
            Htime_deviation = statistics.stdev(data_value_Htime)
            Htime_deviation = "{:.2f}".format(Htime_deviation)
            #Flow Rate
            flow_mean =  statistics.mean(data_value_Flow)
            flow_mean = "{:.2f}".format(flow_mean)
            flow_deviation = statistics.stdev(data_value_Flow)
            flow_deviation = "{:.2f}".format(flow_deviation)
            #Pressure
            pressure_mean =  statistics.mean(data_value_Flow)
            pressure_mean = "{:.2f}".format(pressure_mean)
            pressure_deviation = statistics.stdev(data_value_Flow)
            pressure_deviation = "{:.2f}".format(pressure_deviation)•
            #pH Value
            pH_mean =  statistics.mean(decoded_pH_value)
            pH_mean = "{:.2f}".format(pH_mean)
            pH_deviation = statistics.stdev(decoded_pH_value)
            pH_deviation = "{:.2f}".format(pH_deviation)
            #Cooling Rate
            cooling_mean =  statistics.mean(decoded_cooling_value)
            cooling_mean = "{:.2f}".format(cooling_mean)
            cooling_deviation = statistics.stdev(decoded_cooling_value)
            cooling_deviation = "{:.2f}".format(cooling_deviation)

# calculate and store the correlation coefficients for each pair of variables

            corr_temp, _ = pearsonr(decoded_Temp_value, decoded_pH_value)
            corr_temp = "{:.2f}".format(corr_temp)
            corr_htime, _ = pearsonr(data_value_Htime, decoded_pH_value)
            corr_htime = "{:.2f}".format(corr_htime)
            corr_flow, _ = pearsonr(data_value_Flow, decoded_pH_value)
            corr_flow = "{:.2f}".format(corr_flow)
            corr_pressure, _ = pearsonr(data_value_Pressure, decoded_pH_value)
            corr_pressure = "{:.2f}".format(corr_pressure)
            corr_cooling, _ = pearsonr(decoded_cooling_value, decoded_pH_value)
            corr_cooling = "{:.2f}".format(corr_cooling)


    # if the request method is GET or the form hasn't been submitted yet, just render the template with the original data
            data =[]
            for i in range(len(decoded_Temp_value)):
                row = {
                    'Temp': decoded_Temp_value[i],
                    'Htime': data_value_Htime[i],
                    'Flow': data_value_Flow[i],
                    'Pressure': data_value_Pressure[i],
                    'pH': decoded_pH_value[i],
                    'Cooling': decoded_cooling_value[i]
                }
                data.append(row)
            return render_template('table.html', data=data,temp_mean=temp_mean, temp_deviation=temp_deviation, Htime_mean=Htime_mean, Htime_deviation=Htime_deviation,
                                   flow_mean=flow_mean, flow_deviation=flow_deviation,pressure_mean=pressure_mean, pressure_deviation=pressure_deviation,
                                   pH_mean=pH_mean, pH_deviation=pH_deviation, cooling_mean=cooling_mean,cooling_deviation=cooling_deviation, corr_temp=corr_temp,
                                   corr_htime=corr_htime, corr_flow=corr_flow, corr_pressure=corr_pressure,corr_cooling=corr_cooling )

@app.route('/dashboard')
def dashboard():
    # Pass data to template
    data = {
        'temp': json.dumps(decoded_Temp_value),
        'htime': json.dumps(data_value_Htime),
        'flow': json.dumps(data_value_Flow),
        'pressure': json.dumps(data_value_Pressure),
        'pH': json.dumps(decoded_pH_value),
        'cooling': json.dumps(decoded_cooling_value)
    }
    return render_template('dashboard.html', data=data)
if __name__ == '__main__':
    app.run(debug=True)
```

Figure 15: Code for web application.

3.7 Testing and Validation

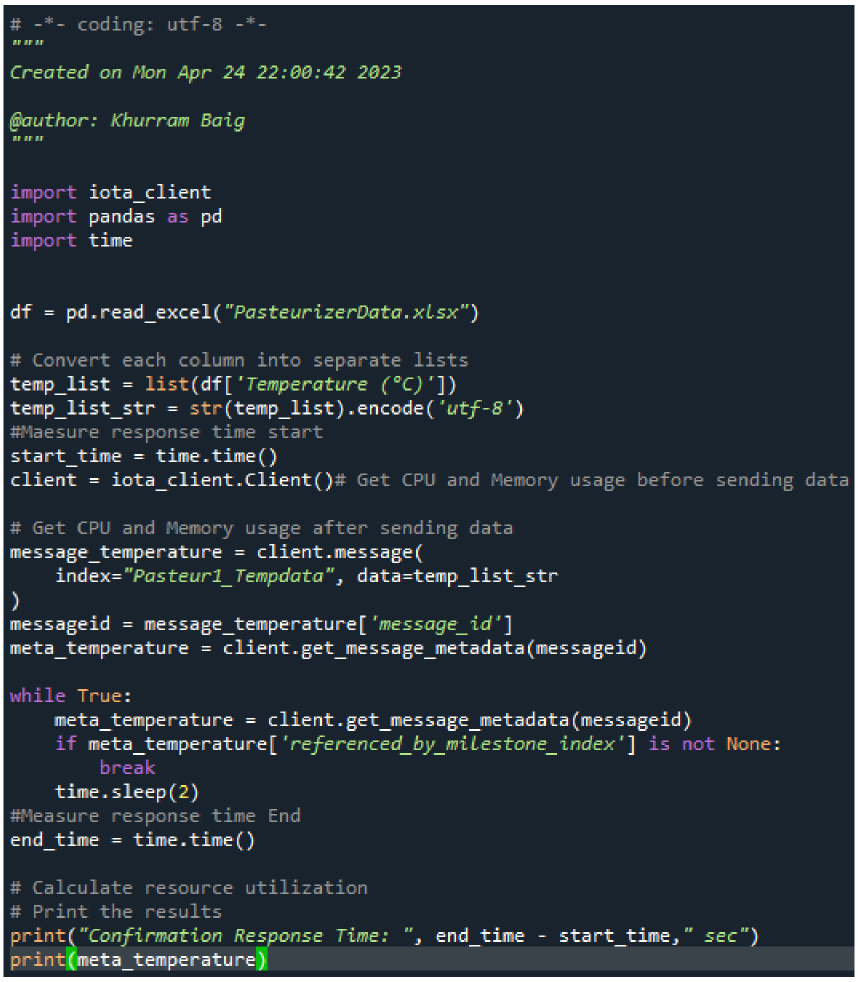
Though the application is implemented successfully, it is important to evaluate objectively the utilization of this application in this study. It is also important to propose a mechanism to test and validate the performance of the application. There are several ways to measure the performance of the client and subscriber of the IOTA Tangle. In this study a code is designed to evaluate transaction response time of a Tangle, CPU and memory utilization of the IOTA client and transaction confirmation time. Following the code explanation for testing scheme is explained.

3.7.1 Code scheme for Testing and Validation

The code in Figure 16 reads temperature data from an Excel file "PasteurizerData.xlsx", and converts it into a list of strings. Then, it uses the IOTA client library to send this data to the Tangle network as a message with the index "Pasteur1_Tempdata". The messageid is obtained from the response, and the metadata of the message is retrieved.

The code then enters a loop where it checks the metadata of the message until it is confirmed by a referenced_by_milestone_index parameter within the message. As soon as the message referenced by a milestone is used, it should be confirmed in IOTA Tangle. The loop sleeps for 2 seconds between checks to avoid overloading the network. Once the message is confirmed, the loop breaks, and the confirmation response time is calculated and printed along with the metadata of the message.

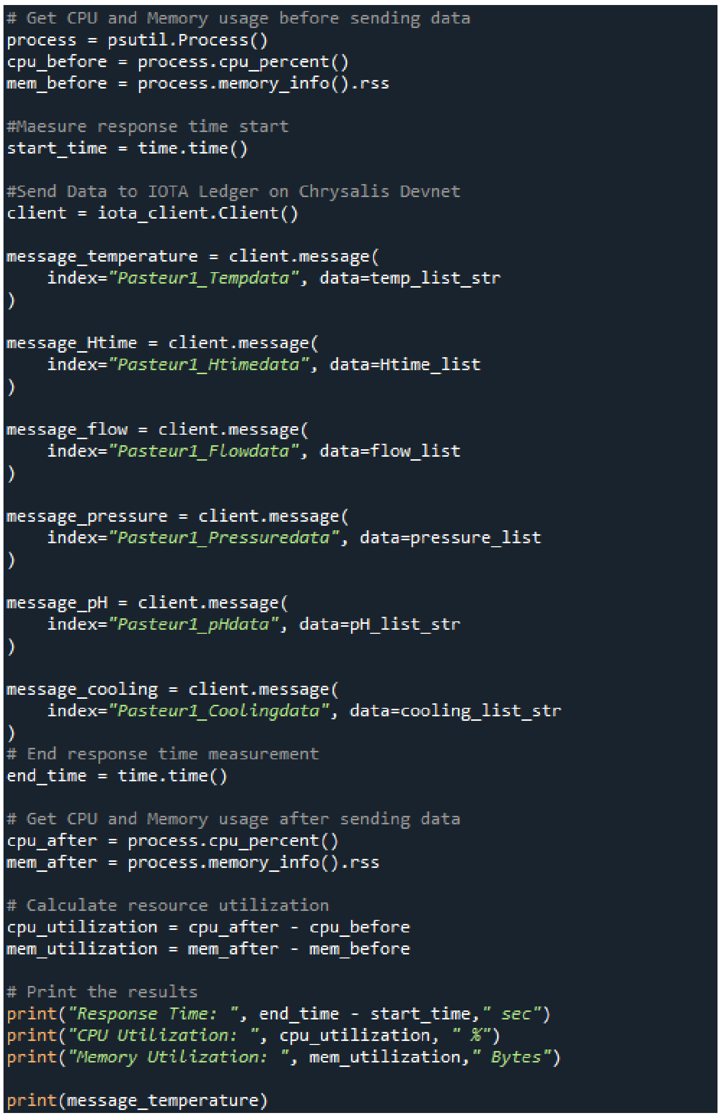
The purpose of this code is to measure the confirmation response time of the IOTA Tangle network, which is the time taken for the network to confirm that the message has been received and stored in a block. The response time can be used to evaluate the performance of the network and optimize it for real-world data storage applications.



```
# -*- coding: utf-8 -*-
"""
Created on Mon Apr 24 22:00:42 2023

@author: Khurram Baig
"""

import iota_client
import pandas as pd
import time


df = pd.read_excel("PasteurizerData.xlsx")

# Convert each column into separate lists
temp_list = list(df['Temperature (°C)'])
temp_list_str = str(temp_list).encode('utf-8')
#Maesure response time start
start_time = time.time()
client = iota_client.Client()# Get CPU and Memory usage before sending data

# Get CPU and Memory usage after sending data
message_temperature = client.message(
    index="Pasteur1_Tempdata", data=temp_list_str
)
messageid = message_temperature['message_id']
meta_temperature = client.get_message_metadata(messageid)

while True:
    meta_temperature = client.get_message_metadata(messageid)
    if meta_temperature['referenced_by_milestone_index'] is not None:
        break
    time.sleep(2)
#Measure response time End
end_time = time.time()

# Calculate resource utilization
# Print the results
print("Confirmation Response Time: ", end_time - start_time," sec")
print(meta_temperature)
```

Figure 16: Code for performance testing of IOTA Tangle.

Figure 17 shows the code that measures the resource utilization, including CPU and memory, and the response time of sending data to the IOTA ledger on Chrysalis Devnet. The code uses the psutil library to get the CPU and memory usage before and after sending the data. The measurement starts by getting the CPU and memory usage before sending the data using the psutil library. The code then sends the data to the IOTA ledger on Chrysalis Devnet using the IOTA Client library. It sends data for temperature, humidity, flow, pressure, pH, and cooling. The response time measurement starts by recording the current time before sending the data. After sending the data, the current time is recorded again, and the difference between the two times is calculated to get the response time. The code then gets the CPU and memory usage again, after sending the data. The CPU and memory utilization are calculated by subtracting the values obtained before sending the data from those obtained after sending the data.



```
# Get CPU and Memory usage before sending data
process = psutil.Process()
cpu_before = process.cpu_percent()
mem_before = process.memory_info().rss

#Maesure response time start
start_time = time.time()

#Send Data to IOTA Ledger on Chrysalis Devnet
client = iota_client.Client()

message_temperature = client.message(
    index="Pasteur1_Tempdata", data=temp_list_str
)

message_Htime = client.message(
    index="Pasteur1_Htimedata", data=Htime_list
)

message_flow = client.message(
    index="Pasteur1_Flowdata", data=flow_list
)

message_pressure = client.message(
    index="Pasteur1_Pressuredata", data=pressure_list
)

message_pH = client.message(
    index="Pasteur1_pHdata", data=pH_list_str
)

message_cooling = client.message(
    index="Pasteur1_Coolingdata", data=cooling_list_str
)
# End response time measurement
end_time = time.time()

# Get CPU and Memory usage after sending data
cpu_after = process.cpu_percent()
mem_after = process.memory_info().rss

# Calculate resource utilization
cpu_utilization = cpu_after - cpu_before
mem_utilization = mem_after - mem_before

# Print the results
print("Response Time: ", end_time - start_time," sec")
print("CPU Utilization: ", cpu_utilization, " %")
print("Memory Utilization: ", mem_utilization," Bytes")

print(message_temperature)
```

Figure 17: Code for testing response time and resource utilization of IOTA Tangle.

The complete code for each component of the application discussed in this chapter is provided in Appendix B, accompanying this report.

4 Results and Discussions

This chapter demonstrates the data storage application based on the IOTA Tangle, which is a Distributed Ledger Technology. The application was developed by applying the knowledge obtained from the literature review discussed in Chapter 2. This chapter aims to provide a comprehensive outcome obtained from the practical implementation of the application. Initially, we will focus on the results, which can be evaluated visually by examining the Graphical User Interface (GUI) of the application. This will provide insights into how data is being retrieved from the IOTA Tangle without being altered and how it is presented for analysis. Additionally, this chapter aims to provide a detailed analysis of the overall performance of the application, including the key aspects of using the IOTA Tangle as a Distributed Ledger Technology. The subsequent subchapters will delve into the application's performance and provide insights on its major aspects:

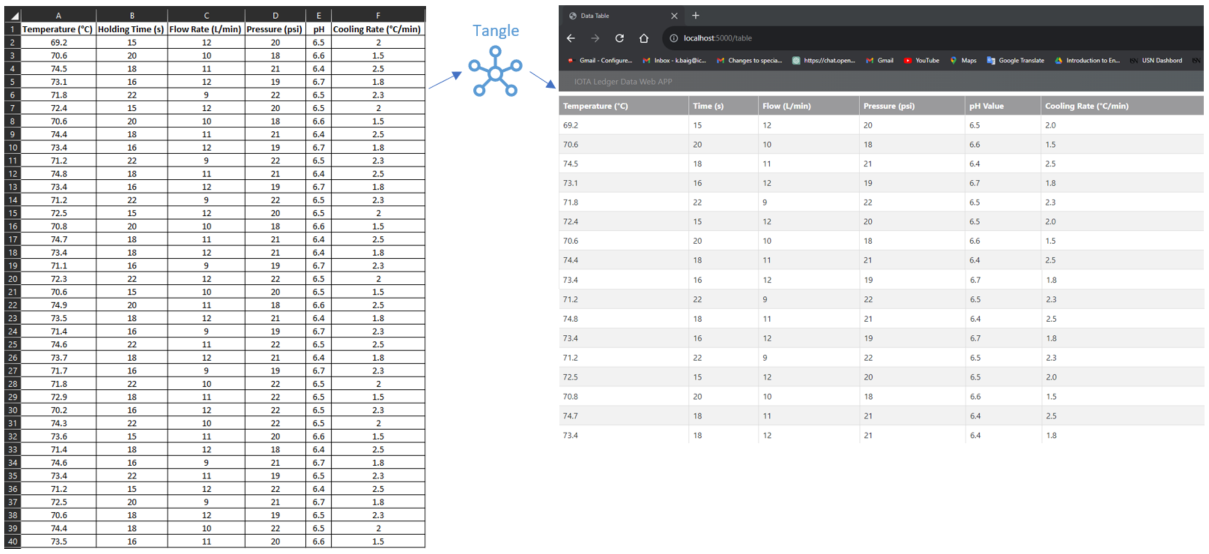
- Data Analysis
- Traceability of Transactions
- Transaction Response Time and Resource Utilization
- Confirmation Time of a Transaction
- Data Security

The analysis parameters discussed earlier offer a comprehensive insight into the strengths and weaknesses of adopting the IOTA Tangle for data storage and processing in Industry 4.0. In the end, we will explore how the results of the analysis relate to the goals of the study.

4.1 Data Analysis

Figure 18 shows a file containing data set from a milk Pasteurizer provided for process handling. This data is unconventionally stored on a decentralized ledger and then acquired by a remote node using a web application. The dataset in the Figure 15 has 40 samples and 6 columns representing the process parameters of a milk pasteurizer as follows:

- Temperature (°C): Temperature of milk at the output of the Pasteurizer.
- Holding Time (s): Holding time of milk in a Pasteurizer tube to ensure homogenized heating of the whole product for a certain time.
- Flow Rate (L/min): Flow rate of milk through Pasteurizer.
- Pressure (psi) pH: pH value of the milk.
- Cooling Rate (°C/min): Time it takes to cool down the milk after Pasteurizing.



	A	B	C	D	E	F
1	Temperature (°C)	Holding Time (s)	Flow Rate (L/min)	Pressure (psi)	pH	Cooling Rate (°C/min)
2	69.2	15	12	20	6.5	2
3	70.6	20	10	18	6.6	1.5
4	74.5	18	11	21	6.4	2.5
5	73.1	16	12	19	6.7	1.8
6	71.8	22	9	22	6.5	2.3
7	72.4	15	12	20	6.5	2
8	70.6	20	10	18	6.6	1.5
9	74.4	18	11	21	6.4	2.5
10	73.4	16	12	19	6.7	1.8
11	71.2	22	9	22	6.5	2.3
12	74.8	18	11	21	6.4	2.5
13	73.4	16	12	19	6.7	1.8
14	71.2	22	9	22	6.5	2.3
15	72.5	15	12	20	6.5	2
16	70.8	20	10	18	6.6	1.5
17	74.7	18	11	21	6.4	2.5
18	73.4	18	12	21	6.4	1.8
19	71.1	16	9	19	6.7	2.3
20	72.3	22	12	22	6.5	2
21	70.6	15	10	20	6.5	1.5
22	74.9	20	11	18	6.6	2.5
23	73.5	18	12	21	6.4	1.8
24	71.4	16	9	19	6.7	2.3
25	74.6	22	11	22	6.5	2.5
26	73.7	18	12	21	6.4	1.8
27	71.7	16	9	19	6.7	2.3
28	71.8	22	10	22	6.5	2
29	72.9	18	11	22	6.5	1.5
30	70.2	16	12	22	6.5	2.3
31	74.3	22	10	22	6.5	2
32	73.6	15	11	20	6.6	1.5
33	71.4	18	12	18	6.4	2.5
34	74.6	16	9	21	6.7	1.8
35	73.4	22	11	19	6.5	2.3
36	71.2	15	12	22	6.4	2.5
37	72.5	20	9	21	6.7	1.8
38	70.6	18	12	19	6.5	2.3
39	74.4	18	10	22	6.5	2
40	73.5	16	11	20	6.6	1.5

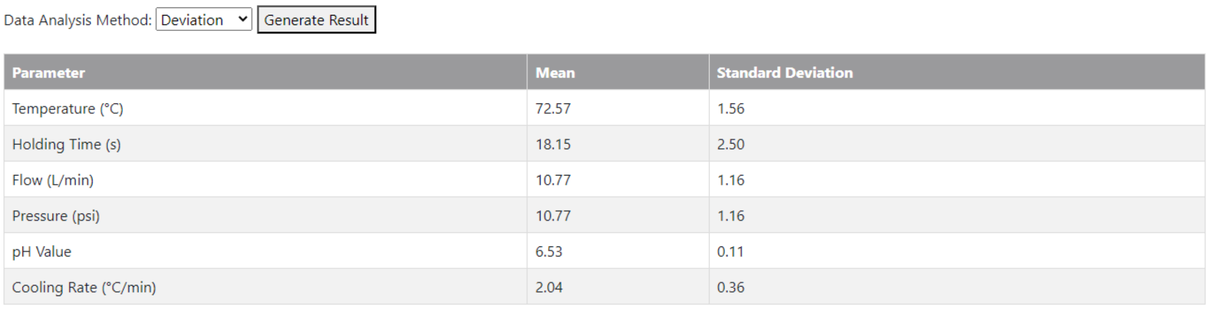
Temperature (°C)	Time (s)	Flow (L/min)	Pressure (psi)	pH Value	Cooling Rate (°C/min)
69.2	15	12	20	6.5	2.0
70.6	20	10	18	6.6	1.5
74.5	18	11	21	6.4	2.5
73.1	16	12	19	6.7	1.8
71.8	22	9	22	6.5	2.3
72.4	15	12	20	6.5	2.0
70.6	20	10	18	6.6	1.5
74.4	18	11	21	6.4	2.5
73.4	16	12	19	6.7	1.8
71.2	22	9	22	6.5	2.3
74.8	18	11	21	6.4	2.5
73.4	16	12	19	6.7	1.8
71.2	22	9	22	6.5	2.3
72.5	15	12	20	6.5	2.0
70.8	20	10	18	6.6	1.5
74.7	18	11	21	6.4	2.5
73.4	18	12	21	6.4	1.8

Figure 18: Data file stored on IOTA Tangle fetched to web application.

Figure 19 provides the mean and standard deviation of different parameters for a dataset. The first column represents the name of the parameter, while the second column represents the mean value of the parameter in the dataset. For example, the mean temperature in the dataset is 72°C.

The third column represents the standard deviation of the parameter in the dataset. The standard deviation is a measure of the spread of the data points around the mean. A smaller standard deviation indicates that the data points are clustered more closely around the mean, while a larger standard deviation indicates that the data points are more spread out. This is an effective way of monitoring process Key Performance Indicators (KPIs) and ensure that they are within quality range.

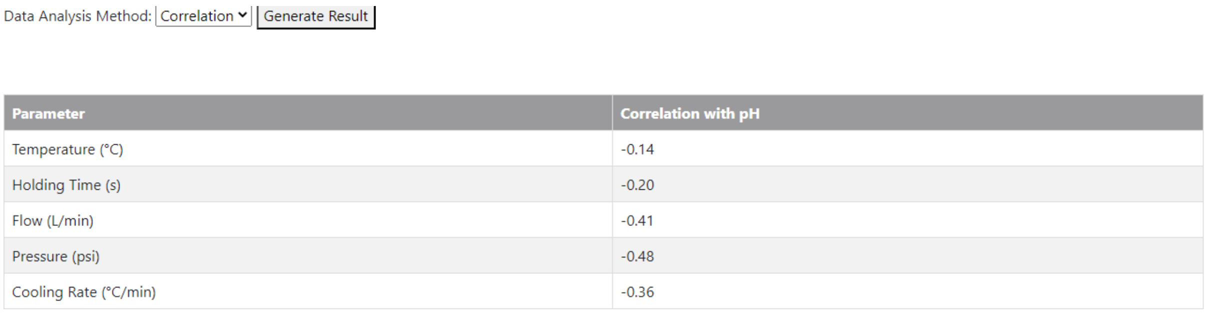
For example, the standard deviation of the temperature parameter in the dataset is 1.56°C. This indicates that the temperature values in the dataset are relatively tightly clustered around the mean value of 72°C, with most values falling within about 1.56°C of the mean. Similarly, the other parameters can be analyzed by their mean and standard values to verify that process is operational within allowed deviation limits.



Parameter	Mean	Standard Deviation
Temperature (°C)	72.57	1.56
Holding Time (s)	18.15	2.50
Flow (L/min)	10.77	1.16
Pressure (psi)	10.77	1.16
pH Value	6.53	0.11
Cooling Rate (°C/min)	2.04	0.36

Figure 19: Standard deviation of Pasteurizer parameters.

Figure 20, shows that the temperature has a moderate negative correlation with pH, which means that a higher pasteurization temperature may result in a lower pH, indicating a potential decrease in product quality. Additionally, the cooling rate has a moderate positive correlation with pH, suggesting that a faster cooling rate may result in a higher pH and better product quality. This analysis can be further expanded to include other parameters and factors that may affect product quality, but this gives a basic idea of how data analysis can be used to assess the quality of milk products produced by a pasteurizer.



Parameter	Correlation with pH
Temperature (°C)	-0.14
Holding Time (s)	-0.20
Flow (L/min)	-0.41
Pressure (psi)	-0.48
Cooling Rate (°C/min)	-0.36

Figure 20: Correlation of each parameter with pH value.

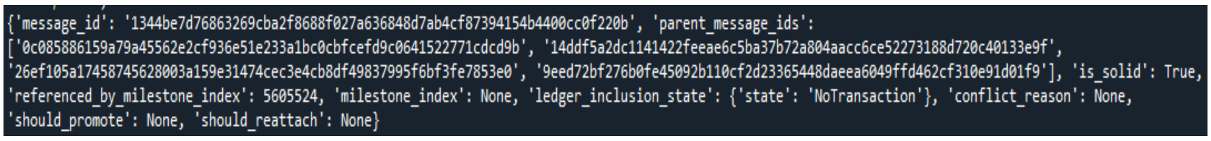
4.2 Traceability of Transactions

IOTA Tangle offers extensive transparency regarding transaction traceability. The following two methods are presented to explore the transaction traceability within the IOTA Tangle.

4.2.1 Traceability with Python

In data storage applications, traceability is crucial for maintaining data integrity and ensuring that the data can be trusted for use in decision-making and other critical tasks. The IOTA provides comprehensive traceability for data transactions made via Tangle. Parameters related to data transactions are available at the developer level in Python and can easily be accessed by using High-level API "get_message_metadata(messageid)" in the iota_client library.

Figure 21 shows the traceability parameters of a transaction with a specific messageid fetched from the Tangle. This metadata tells the parent's ids related to data the status of transactions in the Tangle, milestone information, and other important parameters available in the programming environment to use in the application development.

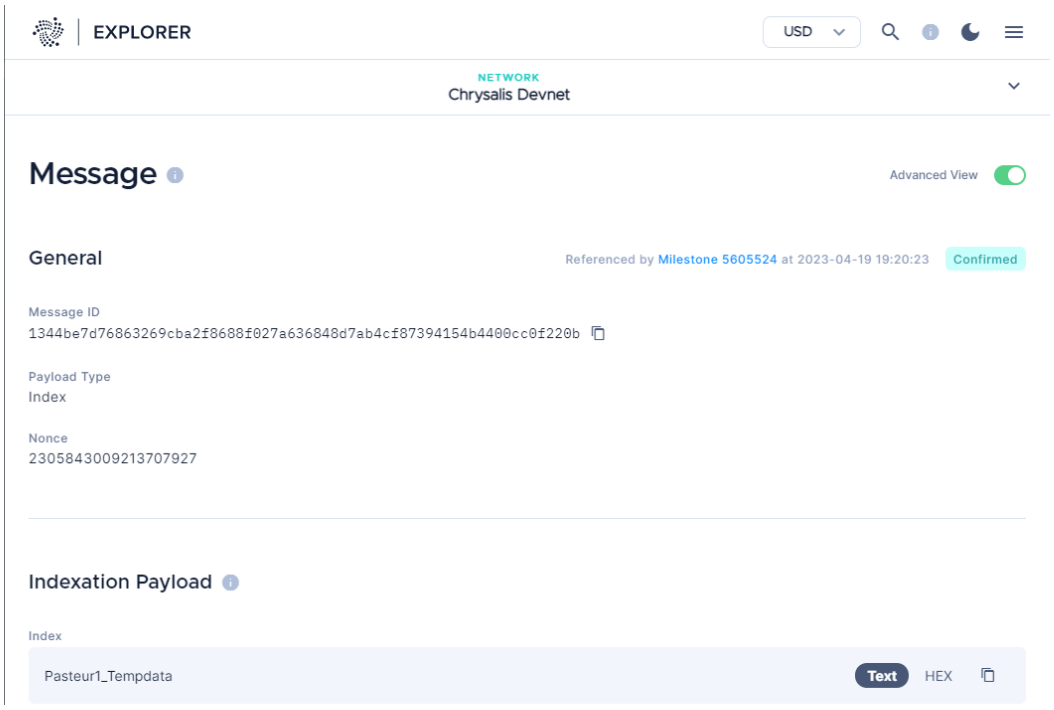


```
{'message_id': '1344be7d76863269cba2f8688f027a636848d7ab4cf87394154b4400cc0f220b', 'parent_message_ids':
['0c085886159a79a45562e2cf936e51e233a1bc0cbfcefd9c0641522771cdcd9b', '14ddf5a2dc1141422feeae6c5ba37b72a804aacc6ce52273188d720c40133e9f',
'26ef105a17458745628003a159e31474cec3e4cb8df49837995f6bf3fe7853e0', '9eed72bf276b0fe45092b110cf2d23365448daeea6049ffd462cf310e91d01f9'], 'is_solid': True,
'referenced_by_milestone_index': 5605524, 'milestone_index': None, 'ledger_inclusion_state': {'state': 'NoTransaction'}, 'conflict_reason': None,
'should_promote': None, 'should_reattach': None}
```

Figure 21: Metadata for data transaction fetched from IOTA Tangle.

4.2.2 Traceability with IOTA Explorer

IOTA Explorer [24] is a web-based tool developed by IOTA Foundation, which provides a GUI that allows users to search for transactions and graphically provide detailed information about each transaction, as shown in Figures 22-24. In addition to traceability parameters IOTA Explorer also shows the data stored in the node.



Message

General

Referenced by Milestone 5605524 at 2023-04-19 19:20:23 Confirmed

Message ID
1344be7d76863269cba2f8688f027a636848d7ab4cf87394154b4400cc0f220b

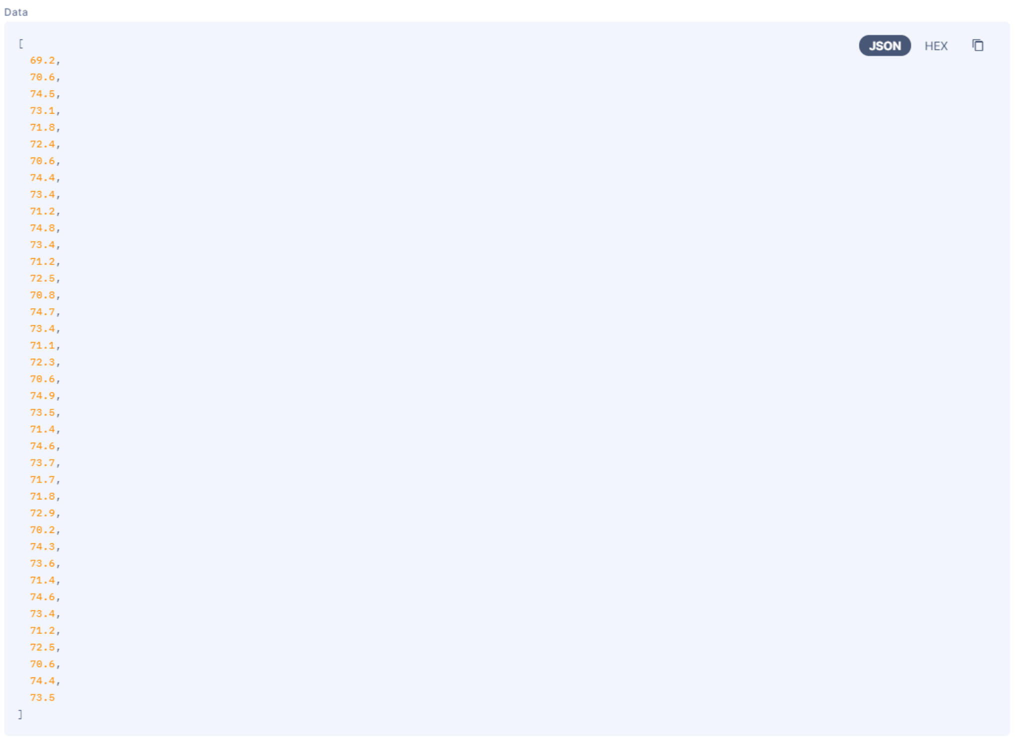
Payload Type
Index

Nonce
2305843009213707927

Indexation Payload

Index
Pasteur1_Tempdata

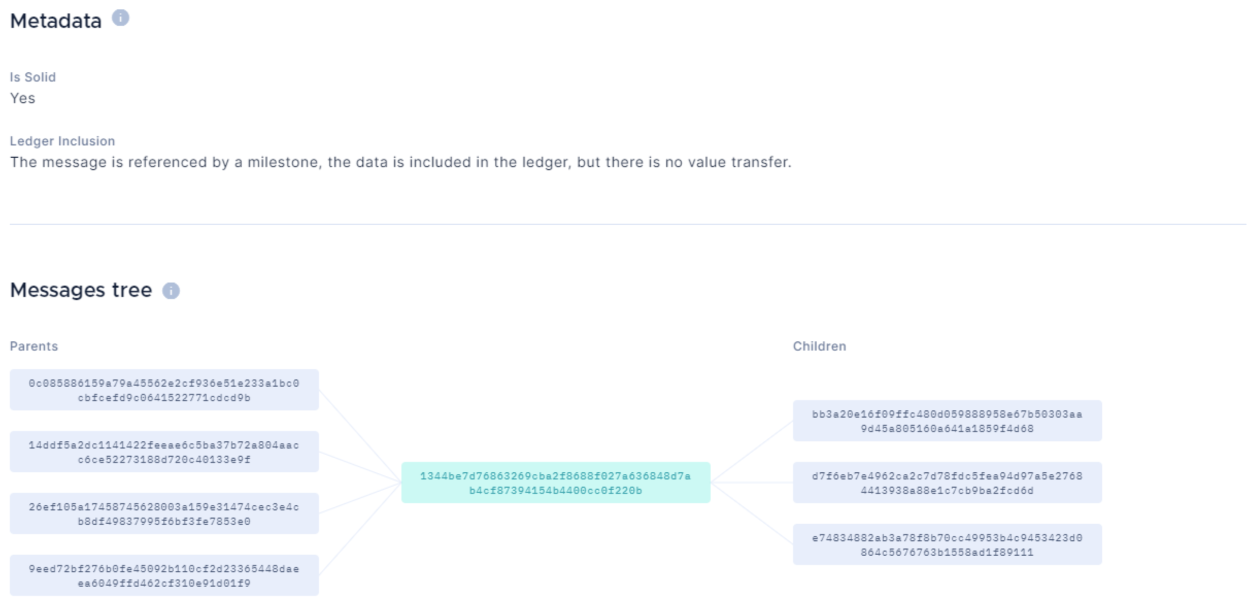
Figure 22: Transaction information shown in IOTA Explorer.



Data

```
[
  69.2,
  70.6,
  74.5,
  73.1,
  71.8,
  72.4,
  70.6,
  74.4,
  73.4,
  71.2,
  74.8,
  73.4,
  71.2,
  72.5,
  70.8,
  74.7,
  73.4,
  71.1,
  72.3,
  70.6,
  74.9,
  73.5,
  71.4,
  74.6,
  73.7,
  71.7,
  71.8,
  72.9,
  70.2,
  74.3,
  73.6,
  71.4,
  74.6,
  73.4,
  71.2,
  72.5,
  70.6,
  74.4,
  73.5
]
```

Figure 23: Data shown in IOTA Explorer.



Metadata

Is Solid
Yes

Ledger Inclusion
The message is referenced by a milestone, the data is included in the ledger, but there is no value transfer.

Messages tree

Parents

```
0c085886159a79a45562e2cf936e51e233a1bc0cbfcefd9c0641522771cdcd9b
14ddf5a2dc1141422feeae6c5ba37b72a804aacc6ce52273188d720c40133e9f
26ef105a17458745628003a159e31474cec3e4cb8df49837995f6bf3fe7853e0
9eed72bf276b0fe45092b110cf2d23365448daeea6049ffd462cf310e91d01f9
```

```
1344be7d76863269cba2f8688f027a636848d7ab4cf87394154b4400cc0f220b
```

Children

```
bb3a20e16f09ffc480d059888958e67b50303aa9d45a805160a641a1859f4d68
d7f6eb7e4962ca2c7d78fdc5fea94d97a5e27684413938a88e1c7cb9ba2fcd6d
e74834882ab3a78f8b70cc49953b4c9453423d0864c5676763b1558ad1f89111
```

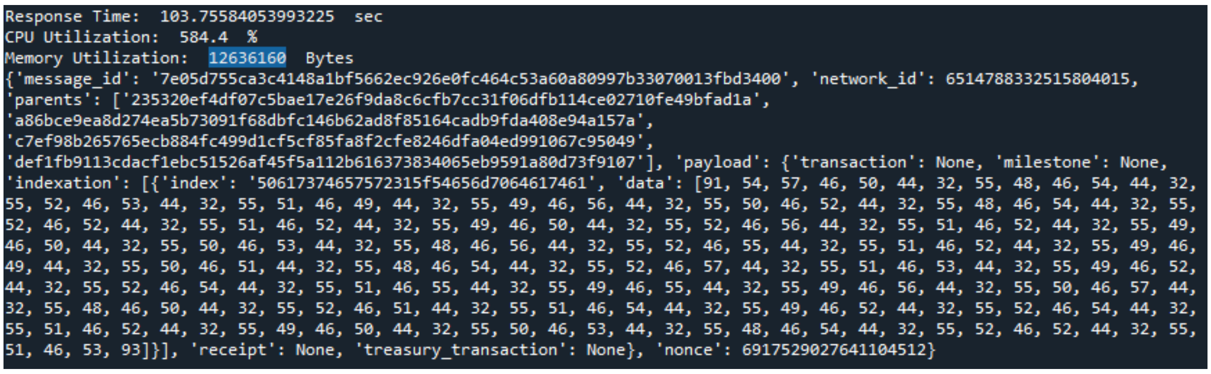
Figure 24: Metadata of transaction shown in IOTA Explorer.

4.3 Transaction Response Time and Resource Utilization

Transaction response time and resource utilization by IOTA clients are important performance parameters as they help in measuring the efficiency of the transaction process. Response time is the time taken for the client to receive a response from the Tangle after initiating a transaction, which is important for real-time data storage applications or applications which requires high throughput in storing data, such as large data is to be stored after short intervals. On the other hand, resource utilization such as CPU and memory usage by the IOTA client during transaction submission is important as it can impact the overall performance of the system. High resource utilization can lead to slower processing times and potentially cause the system to crash, especially in resource-constrained environments like IoT devices. In this study IOTA client is tested for different data transfer cases as discussed earlier in Chapter 3.7.

4.3.1 Case-1

Figure 25 displays the computed results of transaction Response Time, CPU utilization, and memory usage, when initiating transactions for all six parameters, including integer and float data. The outcomes reveal a considerably high response time and CPU utilization of 103 seconds and 584%, respectively. This unexpectedly high CPU utilization suggests that five out of seven CPU cores are operating at over 100%. Meanwhile, the memory is not significantly impacted by the transaction process and displays a negative value due to unrelated background processes on the machine.

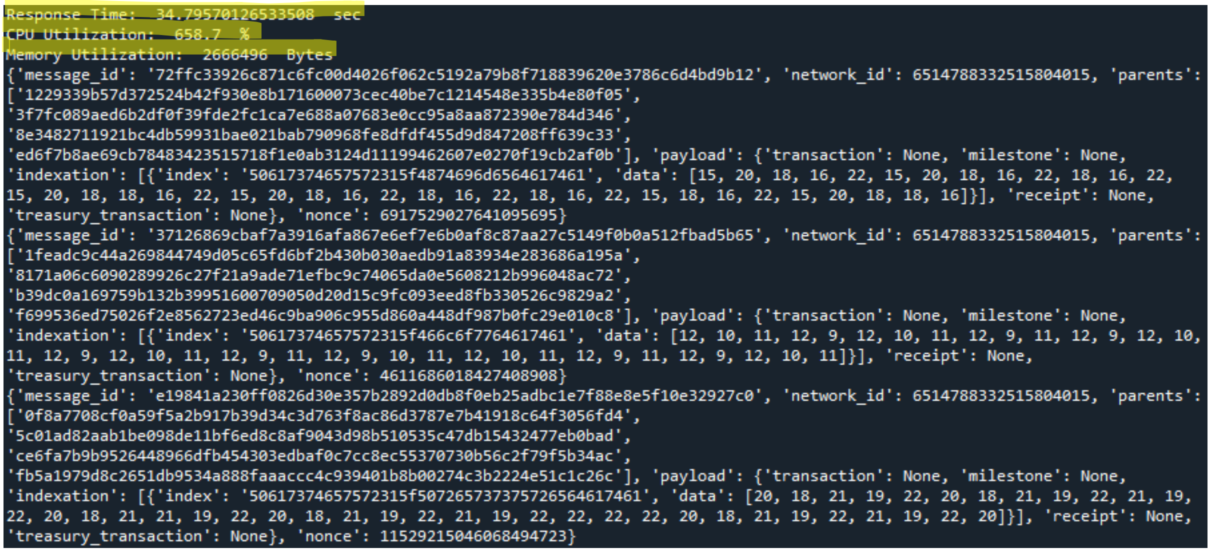


```
Response Time:  103.75584053993225  sec
CPU Utilization:  584.4  %
Memory Utilization:  12636160  Bytes
{'message_id': '7e05d755ca3c4148a1bf5662ec926e0fc464c53a60a80997b33070013fbd3400', 'network_id': 6514788332515804015,
'parents': ['235320ef4df07c5bae17e26f9da8c6cfb7cc31f06dfb114ce02710fe49bfad1a',
'a86bce9ea8d274ea5b73091f68dbfc146b62ad8f85164cadb9fda408e94a157a',
'c7ef98b265765ecb884fc499d1cf5cf85fa8f2cfe8246dfa04ed991067c95049',
'def1fb9113cdacf1ebc51526af45f5a112b616373834065eb9591a80d73f9107'], 'payload': {'transaction': None, 'milestone': None,
'indexation': [{'index': '50617374657572315f54656d7064617461', 'data': [91, 54, 57, 46, 50, 44, 32, 55, 48, 46, 54, 44, 32,
55, 52, 46, 53, 44, 32, 55, 51, 46, 49, 44, 32, 55, 49, 46, 56, 44, 32, 55, 50, 46, 52, 44, 32, 55, 48, 46, 54, 44, 32, 55,
52, 46, 52, 44, 32, 55, 51, 46, 52, 44, 32, 55, 49, 46, 50, 44, 32, 55, 52, 46, 56, 44, 32, 55, 51, 46, 52, 44, 32, 55, 49,
46, 50, 44, 32, 55, 50, 46, 53, 44, 32, 55, 48, 46, 56, 44, 32, 55, 52, 46, 55, 44, 32, 55, 51, 46, 52, 44, 32, 55, 49, 46,
49, 44, 32, 55, 50, 46, 51, 44, 32, 55, 48, 46, 54, 44, 32, 55, 52, 46, 57, 44, 32, 55, 51, 46, 53, 44, 32, 55, 49, 46, 52,
44, 32, 55, 52, 46, 54, 44, 32, 55, 51, 46, 55, 44, 32, 55, 49, 46, 55, 44, 32, 55, 49, 46, 56, 44, 32, 55, 50, 46, 57, 44,
32, 55, 48, 46, 50, 44, 32, 55, 52, 46, 51, 44, 32, 55, 51, 46, 54, 44, 32, 55, 49, 46, 52, 44, 32, 55, 52, 46, 54, 44, 32,
55, 51, 46, 52, 44, 32, 55, 49, 46, 50, 44, 32, 55, 50, 46, 53, 44, 32, 55, 48, 46, 54, 44, 32, 55, 52, 46, 52, 44, 32, 55,
51, 46, 53, 93]}], 'receipt': None, 'treasury_transaction': None}, 'nonce': 6917529027641104512}
```

Figure 25: Case-1, Response time and resource utilization while sending 6 transactions, 40 samples each including float values.

4.3.2 Case-2

The outcome of the transaction Response Time and CPU and Memory utilization for the 3 parameters, which only includes integer data, are presented in Figure 26. The results indicate a relatively lower response time and CPU utilization, at 35s and 658%, respectively. The memory usage is not significantly affected by the transaction process, with a value of approximately 2.6 Megabyte. However, storing float values on the Tangle is more complex as they need to be converted into bytes before being sent to the Tangle, resulting in a message with a large indexation payload. This highlights the challenges associated with storing float values on the Tangle.

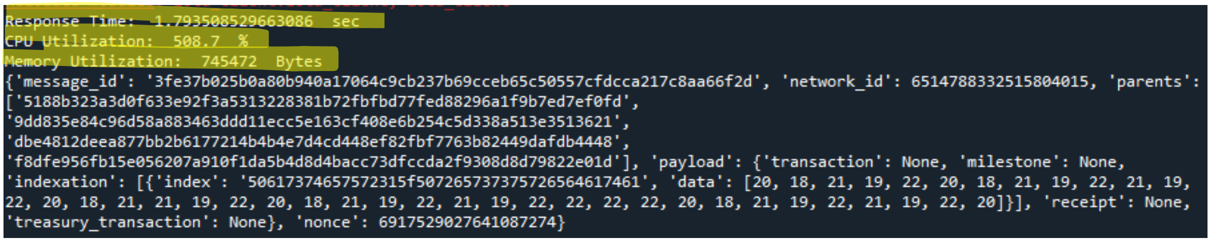


```
Response Time:  34.79570126533508  sec
CPU Utilization:  658.7  %
Memory Utilization:  2666496  Bytes
{'message_id': '72ffc33926c871c6fc00d4026f062c5192a79b8f718839620e3786c6d4bd9b12', 'network_id': 6514788332515804015, 'parents':
['1229339b57d372524b42f930e8b171600073cec40be7c1214548e335b4e80f05',
'3f7fc089aed6b2df0f39fde2fc1ca7e688a07683e0cc95a8aa872390e784d346',
'8e3482711921bc4db59931bae021bab790968fe8dfdf455d9d847208ff639c33',
'ed6f7b8ae69cb78483423515718f1e0ab3124d11199462607e0270f19cb2af0b'], 'payload': {'transaction': None, 'milestone': None,
'indexation': [{'index': '50617374657572315f4874696d6564617461', 'data': [15, 20, 18, 16, 22, 15, 20, 18, 16, 22, 18, 16, 22,
15, 20, 18, 18, 16, 22, 15, 20, 18, 16, 22, 18, 16, 22, 18, 16, 22, 15, 18, 16, 22, 15, 20, 18, 18, 16]}], 'receipt': None,
'treasury_transaction': None}, 'nonce': 6917529027641095695}
{'message_id': '37126869cbaf7a3916afa867e6ef7e6b0af8c87aa27c5149f0b0a512fbad5b65', 'network_id': 6514788332515804015, 'parents':
['1feadc9c44a269844749d05c65fd6bf2b430b030aedb91a83934e283686a195a',
'8171a06c6090289926c27f21a9ade71efbc9c74065da0e5608212b996048ac72',
'b39dc0a169759b132b39951600709050d20d15c9fc093eed8fb330526c9829a2',
'f699536ed75026f2e8562723ed46c9ba906c955d860a448df987b0fc29e010c8'], 'payload': {'transaction': None, 'milestone': None,
'indexation': [{'index': '50617374657572315f466c6f7764617461', 'data': [12, 10, 11, 12, 9, 12, 10, 11, 12, 9, 11, 12, 9, 12, 10,
11, 12, 9, 12, 10, 11, 12, 9, 11, 12, 9, 10, 11, 12, 10, 11, 12, 9, 11, 12, 9, 12, 10, 11]}], 'receipt': None,
'treasury_transaction': None}, 'nonce': 4611686018427408908}
{'message_id': 'e19841a230ff0826d30e357b2892d0db8f0eb25adbc1e7f88e8e5f10e32927c0', 'network_id': 6514788332515804015, 'parents':
['0f8a7708cf0a59f5a2b917b39d34c3d763f8ac86d3787e7b41918c64f3056fd4',
'5c01ad82aab1be098de11bf6ed8c8af9043d98b510535c47db15432477eb0bad',
'ce6fa7b9b9526448966dfb454303edbaf0c7cc8ec55370730b56c2f79f5b34ac',
'fb5a1979d8c2651db9534a888faaaccc4c939401b8b00274c3b2224e51c1c26c'], 'payload': {'transaction': None, 'milestone': None,
'indexation': [{'index': '50617374657572315f507265737375726564617461', 'data': [20, 18, 21, 19, 22, 20, 18, 21, 19, 22, 21, 19,
22, 20, 18, 21, 21, 19, 22, 20, 18, 21, 19, 22, 21, 19, 22, 22, 22, 20, 18, 21, 19, 22, 21, 19, 22, 20]}], 'receipt': None,
'treasury_transaction': None}, 'nonce': 11529215046068494723}
```

Figure 26: Case-2, Response time and resource utilization while sending 3 transactions, 40 samples each excluding float values.

4.3.3 Case-3

Similarly Figure 27 of Case-3 further strengthens the understanding of transaction process by indicating that greater number of transactions with float value data in their messages cause delay in response but still use high CPU resources. The Response Time is very fast approximately 1.8s and the CPU utilization is 508%, while the memory utilization is 74 Megabyte

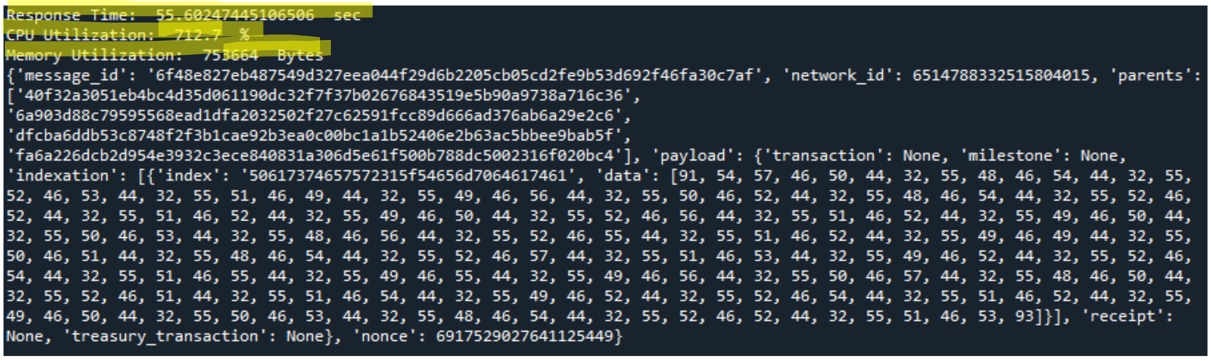


```
Response Time:  1.793508529663086  sec
CPU Utilization:  508.7  %
Memory Utilization:  745472  Bytes
{'message_id': '3fe37b025b0a80b940a17064c9cb237b69cceb65c50557cfdcca217c8aa66f2d', 'network_id': 6514788332515804015, 'parents':
['5188b323a3d0f633e92f3a5313228381b72fbfbd77fed88296a1f9b7ed7ef0fd',
'9dd835e84c96d58a883463ddd11ecc5e163cf408e6b254c5d338a513e3513621',
'dbe4812deea877bb2b6177214b4b4e7d4cd448ef82fbf7763b82449dafdb4448',
'f8dfe956fb15e056207a910f1da5b4d8d4bacc73dfccda2f9308d8d79822e01d'], 'payload': {'transaction': None, 'milestone': None,
'indexation': [{'index': '5061737465757231 5f507265737375726564617461', 'data': [20, 18, 21, 19, 22, 20, 18, 21, 19, 22, 21, 19,
22, 20, 18, 21, 21, 19, 22, 20, 18, 21, 19, 22, 21, 19, 22, 22, 22, 22, 20, 18, 21, 19, 22, 21, 19, 22, 20]}], 'receipt': None,
'treasury_transaction': None}, 'nonce': 6917529027641087274}
```

Figure 27: Case-3, Response time and resource utilization while sending 1 transaction of 40 samples, containing integer data.

4.3.4 Case-4

Figure 28 shows Response Time of 55s, CPU utilization of 712% and memory utilization of 75MB. In comparison with Case-3 where there is a single transaction consisting of only integer data in message, the results show that float data requires more time and resources to store it on the Tangle.



```
Response Time:  55.60247445106506  sec
CPU Utilization:  712.7  %
Memory Utilization:  753664  Bytes
{'message_id': '6f48e827eb487549d327eea044f29d6b2205cb05cd2fe9b53d692f46fa30c7af', 'network_id': 6514788332515804015, 'parents':
['40f32a3051eb4bc4d35d061190dc32f7f37b02676843519e5b90a9738a716c36',
'6a903d88c79595568ead1dfa2032502f27c62591fcc89d666ad376ab6a29e2c6',
'dfcba6ddb53c8748f2f3b1cae92b3ea0c00bc1a1b52406e2b63ac5bbee9bab5f',
'fa6a226dcb2d954e3932c3ece840831a306d5e61f500b788dc5002316f020bc4'], 'payload': {'transaction': None, 'milestone': None,
'indexation': [{'index': '5061737465757231 5f54656d7064617461', 'data': [91, 54, 57, 46, 50, 44, 32, 55, 48, 46, 54, 44, 32, 55,
52, 46, 53, 44, 32, 55, 51, 46, 49, 44, 32, 55, 49, 46, 56, 44, 32, 55, 50, 46, 52, 44, 32, 55, 48, 46, 54, 44, 32, 55, 52, 46,
52, 44, 32, 55, 51, 46, 52, 44, 32, 55, 49, 46, 50, 44, 32, 55, 52, 46, 56, 44, 32, 55, 51, 46, 52, 44, 32, 55, 49, 46, 50, 44,
32, 55, 50, 46, 53, 44, 32, 55, 48, 46, 56, 44, 32, 55, 52, 46, 55, 44, 32, 55, 51, 46, 52, 44, 32, 55, 49, 46, 49, 44, 32, 55,
50, 46, 51, 44, 32, 55, 48, 46, 54, 44, 32, 55, 52, 46, 57, 44, 32, 55, 51, 46, 53, 44, 32, 55, 49, 46, 52, 44, 32, 55, 52, 46,
54, 44, 32, 55, 51, 46, 55, 44, 32, 55, 49, 46, 55, 44, 32, 55, 49, 46, 56, 44, 32, 55, 50, 46, 57, 44, 32, 55, 48, 46, 50, 44,
32, 55, 52, 46, 51, 44, 32, 55, 51, 46, 54, 44, 32, 55, 49, 46, 52, 44, 32, 55, 52, 46, 54, 44, 32, 55, 51, 46, 52, 44, 32, 55,
49, 46, 50, 44, 32, 55, 50, 46, 53, 44, 32, 55, 48, 46, 54, 44, 32, 55, 52, 46, 52, 44, 32, 55, 51, 46, 53, 93]}], 'receipt':
None, 'treasury_transaction': None}, 'nonce': 6917529027641125449}
```

Figure 28: Case-4, Response time and resource utilization while sending 1 transaction of 40 samples, containing float data.

4.3.5 Summary Table for Case1-4

Table 2 presents a comprehensive overview of various performance parameters related to transactions, including the number of messages sent simultaneously, data types, average response time, average CPU utilization, and average memory utilization. Each row in the table represents a distinct scenario with specific characteristics. The measurements were conducted for 10 consecutive transactions, and the average values of the performance parameters are calculated for each row in Table 2.

No. of messages	Data Type	Av. Response Time/10 Transactions	Av.CPU Utilization/10 Transactions *	Av. Memory Utilization/10 Transactions
6	**Float & Integer**	225.8 Sec	91%	14 MB
3	**Integers**	49.5 Sec	91%	2.6 MB
1	**Float**	26.7 sec	91%	0.7 MB
1	**Integer**	12.8 Sec	91%	0.63 MB

Table 2: Average Response Time, CPU Utilization and Memory Utilization.

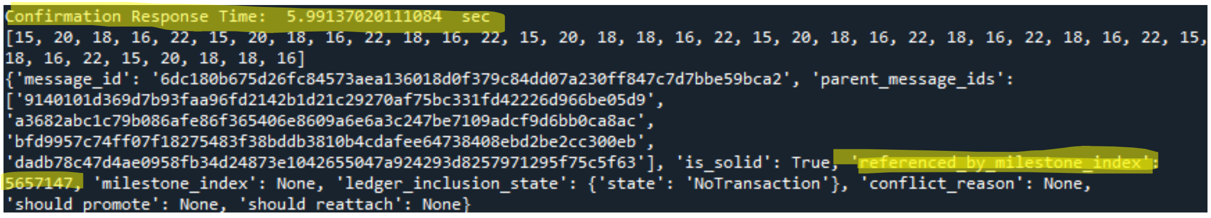
* During the testing, a computer with 7 cores was utilized, which resulted in the conversion of CPU utilization from a scale of 700% (100% per core) to an overall 100% scale.

4.4 Confirmation Time of the Transaction

The transaction confirmation time is an important performance parameter in IOTA Tangle when it comes to data storage applications like continuously generated data coming from IoT or an industrial process. This transaction confirmation time determines how quick data becomes reliable and trustworthy. Only confirmed transactions become final and immutable. The delay in confirmation time can lead to data inconsistencies and integrity issues, which can result in loss or corruption of data. That is why this study has tested the confirmation time of a data transaction on the Tangle.

4.4.1 Case-1

In Figure 29, the confirmation response time in seconds for a transaction that only contains integer data in its payload indexation is displayed. The response time for this case is approximately 6 seconds, which is the duration it took from the moment the client-initiated transaction until it was confirmed and referenced by milestone index 5657147.

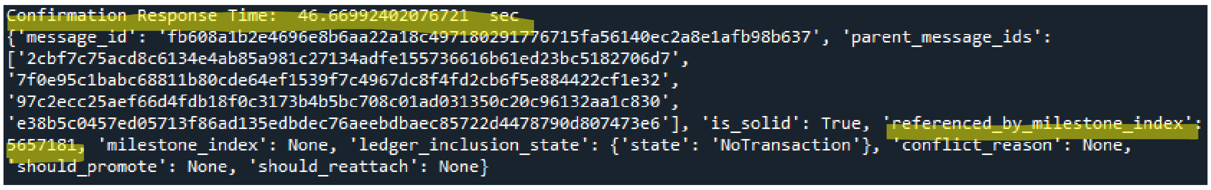


```
Confirmation Response Time:  5.99137020111084  sec
[15, 20, 18, 16, 22, 15, 20, 18, 16, 22, 18, 16, 22, 15, 20, 18, 18, 16, 22, 15, 20, 18, 16, 22, 18, 16, 22, 18, 16, 22, 15,
18, 16, 22, 15, 20, 18, 18, 16]
{'message_id': '6dc180b675d26fc84573aea136018d0f379c84dd07a230ff847c7d7bbe59bca2', 'parent_message_ids':
['9140101d369d7b93faa96fd2142b1d21c29270af75bc331fd42226d966be05d9',
'a3682abc1c79b086afe86f365406e8609a6e6a3c247be7109adcf9d6bb0ca8ac',
'bfd9957c74ff07f18275483f38bddb3810b4cdafee64738408ebd2be2cc300eb',
'dadb78c47d4ae0958fb34d24873e1042655047a924293d8257971295f75c5f63'], 'is_solid': True, 'referenced_by_milestone_index':
5657147, 'milestone_index': None, 'ledger_inclusion_state': {'state': 'NoTransaction'}, 'conflict_reason': None,
'should_promote': None, 'should_reattach': None}
```

Figure 29: Case-1, Response time from adding a transaction in IOTA Tangle till its gets confirmed.

4.4.2 Case-2

The outcome for confirmation response time in seconds for a transaction with float data in its payload indexation is presented in Figure 30. The transaction took around 47 seconds from the moment it was initiated by the client until it was confirmed by referencing the milestone_ index_number of 5657181.



```
Confirmation Response Time:  46.66992402076721  sec
{'message_id': 'fb608a1b2e4696e8b6aa22a18c4971802917776715fa56140ec2a8e1afb98b637', 'parent_message_ids':
['2cbf7c75acd8c6134e4ab85a981c27134adfe155736616b61ed23bc5182706d7',
'7f0e95c1babc68811b80cde64ef1539f7c4967dc8f4fd2cb6f5e884422cf1e32',
'97c2ecc25aef66d4fdb18f0c3173b4b5bc708c01ad031350c20c96132aa1c830',
'e38b5c0457ed05713f86ad135edbdec76aeebdbaec85722d4478790d807473e6'], 'is_solid': True, 'referenced_by_milestone_index':
5657181, 'milestone_index': None, 'ledger_inclusion_state': {'state': 'NoTransaction'}, 'conflict_reason': None,
'should_promote': None, 'should_reattach': None}
```

Figure 30: Case-2, Response time from adding a transaction in IOTA Tangle until it gets confirmed.

4.5 Discussion

The limitations and issues of IOTA Tangle are discussed in this section. One of the major limitations that is discovered under the testing is the high resource utilization, particularly CPU usage, during requesting a transaction. This can impact the overall performance of the system, especially IoT devices with limited processing power. Another issue is the complexity of storing values greater than 255 or float values on the Tangle, this is because IOTA uses an encoding format that treats the data as a sequence of bytes, and the encoding for a number greater than 255 requires more than one byte.

If the data is not properly formatted, the message will fail to be sent to the IOTA Tangle. To avoid this issue, you need to use an encoding format like UTF-8 or split the data into smaller chunks that can be encoded within the size limits of the IOTA protocol. Ultimately this type of transaction will result in higher response times and CPU utilization compared to transactions with only integer data.

Confirmation time of the transaction is another important performance parameter, as delay in confirmation time can lead to data inconsistencies and integrity issues, resulting in loss or corruption of data. The confirmation time for a transaction with float data in its payload indexation is found to be slower than the confirmation time for a transaction with only integer data, but it can be still considered to be much faster than reported confirmation delays in Blockchain type ledgers [18].

Another limitation of IOTA Tangle regarding data storage application is that each transaction is identified by unique hash and each message has unique messageid. Therefore, when using the iota_client library in Python, it is not possible to send non-value transactions with the same messageid as a previous transaction, as each transaction must have a unique transaction hash. When existing data has to be appended or updated, it will be stored by different messageids which has to be updated on the subscriber side in order to get the latest data. This issue, though can be resolved by programming at the application level, raises concerns about the overall data storage efficiency of the IOTA Tangle. Finding a solution to cope with this issue could be investigated in future works.

One major security concern regarding the IOTA Tangle is the use of a Coordinator as a centralized authority to prevent attacks on the Tangle. Critics argue that this makes the network vulnerable to attacks, as the coordinator could be targeted by hackers or malicious actors. In addition, the coordinator has been shown to cause performance issues and limit the scalability of the network, despite the IOTA foundation has presented a road map for replacing the Coordinator with a more secure and decentralized version of IOTA (Coordicide) [4].The later is still under development phase and its performance, in reality, requires to be evaluated in the future.

Furthermore, IOTA was subjected to an attack on its Trinity wallet in 2020 leading to significant losses [17]. There are more than one security vulnerabilities issues, which are not being handled by IOTA foundation, among them are Byzantine node creation (node which does not forward message to other nodes in network or create conflicts) [10], [17], Sybil identities (creating fake identities to take control of network) and Eclipsing (targeting single user within network). Overall, many studies encourage the scientific community to focus on improving the security of IOTA for the benefit of industry and society [17].

5 Conclusion

In conclusion, this study aims to demonstrate the use of IOTA Tangle, a distributed ledger technology, for data storage applications. The practical implementation of the application was based on the literature review presented in Chapter 2. The results of the study showed that the IOTA Tangle can be effectively used for storing data, which can be further analyzed to handle the industrial process or any real-world application in the case of IoT.

Data extracted from a static source was successfully stored on the Tangle and retrieved unaltered to perform data analysis to monitor the pasteurizer process. The correlation analysis showed that the temperature and cooling rate parameters have a moderate correlation with pH value, indicating their effect on product quality. Experiments showed that the IOTA Tangle can be used to store and analyze process data, with the ability to monitor process Key Performance Indicators within quality range, this can also show its ability to handle any industrial process.

In addition, the analysis of the application's performance revealed that the IOTA Tangle provides comprehensive traceability for data transactions, data can easily be traced and retrieved unaltered by using available APIs. The response time and resource utilization of the IOTA Tangle were found to be more efficient and effective than other Blockchain based DLTs, but there is still sluggish for real-time data requirements. Appending or updating the same instance of data is challenging due to the way the Tangle process transactions operate, which leads to creation of new instance of dataset each time.

Data security is an essential guarantee required by industries and organizations to adopt any new technology. To be a reliable future data storage solution, IOTA Tangle has to be more secure than it is now. Although further development in IOTA Tangle shows positive trends but its functional version is still to be investigated.

Overall, this study has contributed to the understanding of the advantages and limitations of using the IOTA Tangle for data storage and process handling in Industry 4.0 applications. Further research can focus on exploring the solutions for the issues discussed in this study. It will be also interesting to adopt the IOTA Tangle in other applications and process industries to enhance its applicability and functionality.

References

[1] Bundesministerium für Bildung und Forschung, "Industrie 4.0," Bundesministerium für Bildung und Forschung. [Online]. Available: https://www.bmbf.de/bmbf/de/forschung/digitale-wirtschaft-und-gesellschaft/industrie-4-0/industrie-4-0. [Accessed: May 12, 2023].

[2] K. Ashton, "That 'Internet of Things' Thing: In the Real World Things Matter More than Ideas," RFID Journal, 2009.

[3] S. Nakamoto, "Bitcoin: A Peer-to-Peer Electronic Cash System," 2008.

[4] IOTA Foundation, "The Tangle," IOTA Wiki. [Online]. Available: https://wiki.iota.org/learn/about-iota/tangle/. [Accessed: Apr. 9, 2023].

[5] VeChain Foundation, "VeChain Developer Documentation," VeChain, [Online]. Available: https://docs.vechain.org/. [Accessed: Apr. 9, 2023].

[6] B. Farahani, F. Firouzi, and M. Luecking, "The convergence of IoT and distributed ledger technologies (DLT): Opportunities, challenges, and solutions," Journal of Network and Computer Applications, vol. 177, pp. 102739, 2021.

[7] Investopedia, "Distributed Ledger Technology (DLT)," Investopedia, [Online]. Available: https://www.investopedia.com/terms/d/distributed-ledger-technology-dlt.asp#toc-the-bottom-line. [Accessed: April 9, 2023].

[8] Hedera Hashgraph, "What is Distributed Ledger Technologies (DLTs)?" Hedera Hashgraph Learning Center. [Online]. Available: https://hedera.com/learning/distributed-ledger-technologies/what-are-distributed-ledger-technologies-dlts. [Accessed: April 9, 2023].

[9] T. M. Fernández-Caramés and P. Fraga-Lamas, "A Review on the Use of Blockchain for the Internet of Things," in IEEE Access, vol. 6, pp. 32979-33001, 2018. doi: 10.1109/ACCESS.2018.2842685.

[10] Ali, M. S., Vecchio, M., Pincheira, M., Dolui, K., Antonelli, F., & Rehmani, M. H. (2019). Applications of Blockchains in the Internet of Things: A Comprehensive Survey. IEEE Communications Surveys & Tutorials, 21(2), 1676-1717. doi: 10.1109/COMST.2018.2886932

[11] A. Rawat, V. Daza, and M. Signorini, "Offline Scaling of IoT Devices in IOTA Blockchain," Sensors, vol. 22, no. 4, Feb. 2022, art. no. 1411, doi: 10.3390/s22041411.

[12] TechTarget, "What is blockchain? - Definition from WhatIs.com," SearchCIO. [Online]. Available: https://www.techtarget.com/searchcio/definition/blockchain. [Accessed: Apr. 9, 2023].

[13] VeChain Foundation, "VeChainThor: The Blockchain Platform for Business 3.0," Whitepaper, VeChain Foundation, Dec. 2018. [Online]. Available: https://www.vechain.org/assets/whitepaper/whitepaper-3-0.pdf. [Accessed: Apr. 9, 2023].

[14] S. Popov, "The Tangle," Apr. 30, 2018, Version 1.4.3

[15] *E. Vieira, J. Ferreira and P. C. Bartolomeu, "Blockchain Technologies for IoT Applications: Use-cases and Limitations," 2020 25th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA), 2020, doi: 10.1109/ETFA46521.2020.9211927.*

[16] *N. Sealey, A. Aijaz and B. Holden, "IOTA Tangle 2.0: Toward a Scalable, Decentralized, Smart, and Autonomous IoT Ecosystem," 2022 International Conference on Smart Applications, Communications, and Networking (SmartNets), Nov. 29 2022-Dec. 1 2022, doi 10.1109/SmartNets55823.2022.9994016.*

[17] *M. Conti, G. Kumar, P. Nerurkar, R. Saha and L. Vigneri, "A survey on security challenges and solutions in the IOTA," Journal of Network and Computer Applications, vol. 203, p. 103383, Jul. 2022, doi: 10.1016/j.jnca.2022.103383.*

[18] *The Spyder community, "Spyder Integrated Development Environment," [Online]. Available: https://www.spyder-ide.org/. [Accessed: May 12, 2023].*

[19] *PyData Development Team, Pandas, 2021. Available: https://pandas.pydata.org. Accessed on: May 9, 2023.*

[20] *"Flask Documentation (2.3.x)," 2021. [Online]. Available: https://flask.palletsprojects.com/en/2.3.x/. [Accessed: May 9, 2023].*

[21] *Chart.js contributors, "Chart.js," [Online]. Available: https://www.chartjs.org/. [Accessed: May 12, 2023].*

[22] *The Bootstrap Team, "Bootstrap," [Online]. Available: https://getbootstrap.com/. [Accessed: May 12, 2023].*

[23] *Microsoft, "Visual Studio Code," [Online]. Available: https://code.visualstudio.com/. [Accessed: May 12, 2023].*

[24] *IOTA Foundation, "Devnet Explorer," [Online]. Available: https://explorer.iota.org/devnet. [Accessed: May 9, 2023].*

Appendices

Appendix A

USN South-Eastern Norway

Faculty of Technology, Natural Sciences and Maritime Sciences, Campus Porsgrunn

Title: IOTA for Industry 4.0 to handle production processes

USN supervisor: Leila Ben Saad

External partner: N/A

Task background:
In advanced industrial operations, data traceability related to critical systems and services should be assessed and continuously monitored. In this context of Industry 4.0, designing solutions based on distributed ledgers like IOTA can be very interesting and useful. IOTA [1-3] is an open, feeless, and scalable distributed ledger for storing transactions that adapts some principles from Blockchain. It is decentralized and based on a structure known as Directed Acyclic Graph (DAG), which in the IOTA community is referred as the Tangle. IOTA can be used for instance as a distributed ledger for detecting, tracking, analysing and correcting errors that can occur in production processes.

Task description:
We consider the example/fabricated data from an industrial process i.e. automated machines with labels affixed to the work-pieces that tell the machines how they need to be processed or choose parameters of any production process of a factory. This information is sent to its respective processing stations via the IOTA Tangle. All the production steps are recorded in the IOTA ledger so that any occurring error can be analysed and corrected immediately.
The goal of the master thesis is to design and develop an application that acquires production data from machines or IoT devices, store it in the decentralized ledger IOTA, and then process and analyse the production data to detect eventual errors.
In this master thesis, a review of the use of distributed ledgers such as IOTA, Vechain [5] and blockchain [4] in storing data collected from (IoT) devices or machines will be conducted.
A discussion about the limitations and challenges encountered in developing IOTA based solution will be made and possible improvement directions will be proposed.
Student category: IIA

The task is suitable for online students (not present at the campus): Yes
Practical arrangements: N/A

Supervision:

As a rule, the student is entitled to 15-20 hours of supervision. This includes necessary time for the supervisor to prepare for supervision meetings (reading material to be discussed, etc).

References:

[1] IOTA https://www.iota.org/

[2] Popov, Serguei Yu.. “The Tangle.” (2015). https://assets.ctfassets.net/r1dr6vzfxhev/2t4uxvsIqk0EUau6g2sw0g/45eae33637ca92f85d9f4a3a218e1ec/iota1_4_3.pdf

[3] Mauro Conti, Gulshan Kumar, Pranav Nerurkar, Rahul Saha, Luigi Vigneri. A survey on security challenges and solutions in the IOTA. Journal of Network and Computer Applications, Volume 203, 2022, 103383, ISSN 1084-8045.

[4] M. Ali, M. Vecchio, M. Pincheira, K. Dolui, F. Antonelli and M. Rehmani, "Applications of Blockchains in the Internet of Things: A Comprehensive Survey", IEEE Communications Surveys & Tutorials, vol. 21, no. 2, pp. 1676-1717, 2019. Available: 10.1109/comst.2018.2886932.

[5] Vechain, https://www.vechain.org/

Signatures:

Supervisor (date and signature): Leila Ben Saad [illegible] 02/02/2023

Student (write clearly in all capitalized letters): KHURRAM BAIG

Student (date and signature): [illegible]

02.02.2023

Appendix B

IOTA Client Program Code:

```
# -*- coding: utf-8 -*-
"""
Created on Mon Feb 20 15:55:35 2023

@author: Khurram Baig
"""
import iota_client
import pandas as pd
import time
import psutil

df = pd.read_excel("PasteurizerData.xlsx")

# Convert each column into separate lists
temp_list = list(df['Temperature (°C)'])
Htime_list = list(df['Holding Time (s)'])
flow_list = list(df['Flow Rate (L/min)'])
pressure_list = list(df['Pressure (psi)'])
pH_list = list(df['pH'])
cooling_list = list(df['Cooling Rate (°C/min)'])

#Encode temperature, pH and Cooling data to bytes to avoid loosing decimal place
temp_list_str = str(temp_list).encode('utf-8')
pH_list_str = str(pH_list).encode('utf-8')
cooling_list_str = str(cooling_list).encode('utf-8')
```

```
# Get CPU and Memory usage before sending data
process = psutil.Process()
cpu_before = process.cpu_percent()
mem_before = process.memory_info().rss

#Maesure response time start
start_time = time.time()

#Send Data to IOTA Ledger on Chrysalis Devnet
client = iota_client.Client()

message_temperature = client.message(
   index="Pasteur1_Tempdata", data=temp_list_str
)

message_Htime = client.message(
   index="Pasteur1_Htimedata", data=Htime_list
)

message_flow = client.message(
   index="Pasteur1_Flowdata", data=flow_list
)
message_pressure = client.message(
   index="Pasteur1_Pressuredata", data=pressure_list
)

message_pH = client.message(
   index="Pasteur1_pHdata", data=pH_list_str
)

message_cooling = client.message(
   index="Pasteur1_Coolingdata", data=cooling_list_str
)
# End response time measurement
end_time = time.time()

# Get CPU and Memory usage after sending data
cpu_after = process.cpu_percent()
mem_after = process.memory_info().rss

# Calculate resource utilization
cpu_utilization = cpu_after - cpu_before
mem_utilization = mem_after - mem_before

# Print the results
print("Response Time: ", end_time - start_time," sec")
print("CPU Utilization: ", cpu_utilization, " %")
print("Memory Utilization: ", mem_utilization," Bytes")

print(message_temperature)
print(message_Htime)
print(message_flow)
print(message_pressure)
print(message_pH)
print(message_cooling)
```

IOTA Subscriber Code:

```
# -*- coding: utf-8 -*-
"""
Created on Tue Feb 21 07:53:16 2023

@author: Khurram Baig
"""
import iota_client
import statistics
from scipy.stats import pearsonr
from flask import Flask, render_template
import json
app = Flask(__name__)
client = iota_client.Client()
message_temperature =
client.get_message_data("1344be7d76863269cba2f8688f027a636848d7ab4cf87394154b4400cc0f2
20b")
message_Htime =
client.get_message_data("f970d36623131b45dc573e47a8ec9b9485bc78bcfe4bbb70f8644460b4fb1
94f")
message_flow =
client.get_message_data("ab340b799752be44ace06b9cbdd7c438b7c04fc8702083dcfba26afd2f6c6f
33")
message_pressure =
client.get_message_data("e3349a9e7d1a4e4bd161e7280f2ee03441c4914c2232b7d47a562c560f7cd
951")
```

```
message_pH =
client.get_message_data("cfa2d1c5ab43d10adfee176c0b48b590b5c2edc8b55cbb3cc5dc4237ac6478
8d")

message_cooling =
client.get_message_data("9d0ba2527ce72abefd077cfc9454e24a1810e544714c38268c9722dafe86ef
df")

data_value_Temp = message_temperature['payload']['indexation'][0]['data']

#Decoding bytes into temperature value

data_Temp_bytes = bytes(data_value_Temp)

str_Temp_value = data_Temp_bytes.decode('utf-8')

#convert string to float

decoded_Temp_value = [float(x) for x in str_Temp_value[1:-1].split(',')]


data_value_Htime = message_Htime['payload']['indexation'][0]['data']

data_value_Flow = message_flow['payload']['indexation'][0]['data']

data_value_Pressure = message_pressure['payload']['indexation'][0]['data']


data_value_pH = message_pH['payload']['indexation'][0]['data']

#Decoding bytes into pH value

data_pH_bytes = bytes(data_value_pH)

str_pH_value = data_pH_bytes.decode('utf-8')

#Converting string to float

decoded_pH_value = [float(x) for x in str_pH_value[1:-1].split(',')]


data_value_cooling = message_cooling['payload']['indexation'][0]['data']

#Decoding bytes into cooling value

data_cooling_bytes = bytes(data_value_cooling)

str_cooling_value = data_cooling_bytes.decode('utf-8')

#Converting string to float

decoded_cooling_value = [float(x) for x in str_cooling_value[1:-1].split(',')]
```

```
#Optional code for storing data into local sql database

# Set up the connection to the SQL Server database using Windows authentication

#server = 'DESKTOP-QT2EDBD\\SQLEXPRESS'

#database = 'PasteurizerDB'

#conn = pyodbc.connect('DRIVER={SQL
Server};SERVER='+server+';DATABASE='+database+';Trusted_Connection=yes;')

############FLASK#############################


@app.route('/table')

#@app.route('/', methods=['GET', 'POST'])

def table():


    temp_mean =  statistics.mean(decoded_Temp_value)

    temp_mean = "{:.2f}".format(temp_mean)

    temp_deviation = statistics.stdev(decoded_Temp_value)

    temp_deviation = "{:.2f}".format(temp_deviation)

    #Holding Time

    Htime_mean =  statistics.mean(data_value_Htime)

    Htime_mean = "{:.2f}".format(Htime_mean)

    Htime_deviation = statistics.stdev(data_value_Htime)

    Htime_deviation = "{:.2f}".format(Htime_deviation)

    #Flow Rate

    flow_mean =  statistics.mean(data_value_Flow)

    flow_mean = "{:.2f}".format(flow_mean)

    flow_deviation = statistics.stdev(data_value_Flow)

    flow_deviation = "{:.2f}".format(flow_deviation)

    #Pressure
```

```
        pressure_mean =  statistics.mean(data_value_Flow)

        pressure_mean = "{:.2f}".format(pressure_mean)

        pressure_deviation = statistics.stdev(data_value_Flow)

        pressure_deviation = "{:.2f}".format(pressure_deviation)

        #pH Value

        pH_mean =  statistics.mean(decoded_pH_value)

        pH_mean = "{:.2f}".format(pH_mean)

        pH_deviation = statistics.stdev(decoded_pH_value)

        pH_deviation = "{:.2f}".format(pH_deviation)

        #Cooling Rate

        cooling_mean =  statistics.mean(decoded_cooling_value)

        cooling_mean = "{:.2f}".format(cooling_mean)

        cooling_deviation = statistics.stdev(decoded_cooling_value)

        cooling_deviation = "{:.2f}".format(cooling_deviation)


# calculate and store the correlation coefficients for each pair of variables


        corr_temp, _ = pearsonr(decoded_Temp_value, decoded_pH_value)

        corr_temp = "{:.2f}".format(corr_temp)

        corr_htime, _ = pearsonr(data_value_Htime, decoded_pH_value)

        corr_htime = "{:.2f}".format(corr_htime)

        corr_flow, _ = pearsonr(data_value_Flow, decoded_pH_value)

        corr_flow = "{:.2f}".format(corr_flow)

        corr_pressure, _ = pearsonr(data_value_Pressure, decoded_pH_value)

        corr_pressure = "{:.2f}".format(corr_pressure)

        corr_cooling, _ = pearsonr(decoded_cooling_value, decoded_pH_value)

        corr_cooling = "{:.2f}".format(corr_cooling)
```

```
    # if the request method is GET or the form hasn't been submitted yet, just render the template
with the original data
            data =[]
            for i in range(len(decoded_Temp_value)):
                row = {
                    'Temp': decoded_Temp_value[i],
                    'Htime': data_value_Htime[i],
                    'Flow': data_value_Flow[i],
                    'Pressure': data_value_Pressure[i],
                    'pH': decoded_pH_value[i],
                    'Cooling': decoded_cooling_value[i]
                }
                data.append(row)
            return render_template('table.html', data=data,temp_mean=temp_mean,
temp_deviation=temp_deviation, Htime_mean=Htime_mean, Htime_deviation=Htime_deviation,
                                   flow_mean=flow_mean,
flow_deviation=flow_deviation,pressure_mean=pressure_mean,
pressure_deviation=pressure_deviation,
                                   pH_mean=pH_mean, pH_deviation=pH_deviation,
cooling_mean=cooling_mean,cooling_deviation=cooling_deviation, corr_temp=corr_temp,
                                   corr_htime=corr_htime, corr_flow=corr_flow,
corr_pressure=corr_pressure,corr_cooling=corr_cooling )


@app.route('/dashboard')
def dashboard():
    # Pass data to template
    data = {
        'temp': json.dumps(decoded_Temp_value),
        'htime': json.dumps(data_value_Htime),
        'flow': json.dumps(data_value_Flow),
        'pressure': json.dumps(data_value_Pressure),
        'pH': json.dumps(decoded_pH_value),
        'cooling': json.dumps(decoded_cooling_value)
    }
    return render_template('dashboard.html', data=data)
if __name__ == '__main__':
    app.run(debug=True)
```