C++ Programming (COMP2034)

Coursework 1 Report

University: University of Nottingham Malaysia

Lecturer: Dr. Doreen Sim Ying Ying

Student Name: Joseph Emmanuel Chay Huan Qin

Student ID: 20580363

Table of Contents

1.0 Abstract

In the post-pandemic era global marketplace, the imperative for supplier companies to adopt efficient inventory and transportation management systems (ITMS) is more pressing than ever. These systems are extremely important for optimizing operations, minimizing costs, and enhancing customer satisfaction. However, with the increasing complexity of supply chains, especially in strategically positioned countries like Malaysia, there is a growing need for transparency, security, and trust in transactions. This paper proposes the integration of blockchain technology with an ITMS for a fictitious supplier company in Malaysia, developed using C++. Blockchain, with its decentralized ledger, immutable records, and smart contract capabilities, offers unparalleled advantages in ensuring transparency, security, and efficiency. This integration aims not only to streamline the company's operations but also to foster trust among stakeholders, reduce fraud, and create a resilient supply chain. By leveraging C++ for system development, this report documentation demonstrates how this technology can be utilized to solve traditional logistical challenges and propel supplier companies towards sustainable growth and competitive advantage.

2.0 Introduction

For supplier companies operating within the complex and dynamic logistical frameworks that characterize modern economies like Malaysia's, managing inventory and transportation logistics efficiently is a critical challenge. Traditional ITMS have been instrumental in enhancing logistical efficiencies; however, the usage of blockchain technology has great opportunity to revolutionize these systems by adding layers of security, transparency, and efficiency previously unattainable. Moreover, in this post-pandemic era, most companies have digitalized their way of procedures and it’s important that as much as operations are being held digitally, that the entire line or process should remain intact and safeguarded. This report explores the significance of blockchain technology in ITMS, specifically focusing on a thorough and complete implementation of an ITMS for a fictitious company called ABC Solutions Sdn. Bhd. in Malaysia using C++ programming.

3.0 Literature Review

The integration of blockchain technology into inventory and transportation management systems represents a crucial approach that addresses several challenges within the supply chain itself. This literature review emphasizes the findings that highlights on the benefits of integrating blockchain in enhancing the efficiency, transparency, and security of supply chain operations. After a thorough research on the Inventory and Transportation Management System, the literature review below properly justifies on the realism and practicality of the developed system.

3.1 Enhancing Transparency and Efficiency

Litke, Anagnostopoulos, and Varvarigou (2019) delve into blockchain's potential to revolutionize supply chain management by offering a secure and transparent ledger for recording transactions, thereby enhancing trust among parties. Their study identifies the architectural elements necessary for the global deployment of blockchain in supply chains, emphasizing the technology's role in overcoming traditional challenges (Litke et al., 2019).

Cai et al. (2023) examine the perceived benefits of blockchain technology in enhancing the resilience and responsiveness of supply chains. They highlight how blockchain's features, such as traceability and decentralization, can mitigate disruption risks and improve supply coordination, thus fostering adoption among supply chain managers (Cai et al., 2023).

3.2 Blockchain for Supply Chain Management

A systematic review by Sangeetha, Shunmugan, and Murugan (2020) explores the integration of blockchain with IoT devices in supply chain management. They underscore the benefits of such integration, including improved productivity, visibility, and reduction of fraudulent activities. This study suggests that blockchain can provide a robust framework for secure and efficient IoT-enabled supply chain operations (Sangeetha et al., 2020).

3.3 Addressing Supply Chain Finance Challenges

Addressing Supply Chain Finance Challenges

Chen and Liu (2023) investigate the role of blockchain in optimizing supply chain finance. They argue that blockchain technology can address issues like risk points and information asymmetry in supply chain finance, leading to significant advantages for management optimization (Chen & Liu, 2023).

3.4 Blockchain for Traceability

Saxena et al. (2023) discuss the opportunities and challenges associated with using blockchain for supply chain traceability. They note that while blockchain offers improved transparency and efficiency, there are still hurdles to overcome, such as the lack of standardization and scalability issues. Ongoing research is focused on leveraging blockchain to enhance supply chain traceability and operational efficiency (Saxena et al., 2023).

4.0 System Design

The design of the system for ABC Solutions Sdn. Bhd. adheres closely to the principles of smart contracts, ensuring efficiency, transparency, and security throughout the inventory and transportation management processes.

4.1 Permissioned Access

The system implements permissioned access controls to regulate respective block variant creation only by authenticated and authorized users who are in the respective industry role and positioned role. Whereby only suppliers will be allowed to create Supplier blocks, transporters for Transporter blocks, and Accountants to create Transaction blocks. Administrators have special access to create all kinds of blocks.

4.2 Immutable Recordkeeping

The entire blockchain with all its blocks and data details will all be stored in as a record like a database to ensure data persistency throughout the lifespan of the entire system to enable viewing, and authorized manipulation to the right cause.

4.3 System Versatility

The architecture of the code system developed for ABC Solutions Sdn. Bhd. is strategically designed with a proper separation of concerns, a fundamental principle that organizes the codes into distinct sections, each handling specific aspects of the application's functionality. This approach not only streamlines the development and maintenance of the system but also significantly enhances its adaptability and scalability over time.

By separating the system into modular components—such as user authentication, block creation, and transaction processing—each module can be developed, updated, or replaced independently

without affecting the overall system's operation. This modularity allows for easier debugging, more straightforward updates, and the seamless integration of new features or technologies as they become available.

5.0 Stages of Supply Chain

The flow of the supply chain is as depicted below.

Supplier Places a chunk of product or products to supply	**Transporter receives the supply block of items to be transported**	**Transporter creates a transport type, ordering type as well as the ordering details**	**Both the Supply and Transportation block has been assigned and detected**	**Transaction created stating details between the Supplier and Transportation**

5.1 Stage 1: Supply Placement

This stage involves in the supplier inserting a placement for a supply to be processed, including the location of the branch as well as the item details which does not have a limit to the amounts of items that can be supplied in a single block.

5.2 Stage 2: Supply Block Guaranteed

The supply block must be ensured to be placed within the supply chain process before proceeding. The system is developed in a way to ensure that no transportation request block can be placed before receiving a supply requirement.

5.3 Stage 3: Transporter Placement

Authorized participants will then be able to place a transporter and transportation detailed block or procedure when the supply has been processed and recorded in the chain. All details in regards to the transportation will be recorded here, such as the Transportation Type, Ordering Type, and Ordering Amounts in Kilograms

5.4 Stage 4: Transporter Block Guaranteed

Likewise, only after the placement of a transporter with its transportation requests, will the next request be allowed to be subjected to the flow.

5.5 Stage 5: Transaction Placement

After receiving the guaranteed request for both the Supply and Transportation requirements, Administrators or accountants will then be required to create a new transaction detailing the information between the supply and transport including the credit balance for both retailer per-trip and Annual Orderings.

6.0 Design Structure of Blocks

The system comprises of a total of 3 block variants. They are namely Supplier, Transporter, and Transaction Block. Each of them contains different sets of information that are vital to the tracking and integrity of the blockchain. These three blocks are all interconnected, detailing sets of transaction information flow within the system.

6.1 Block Data

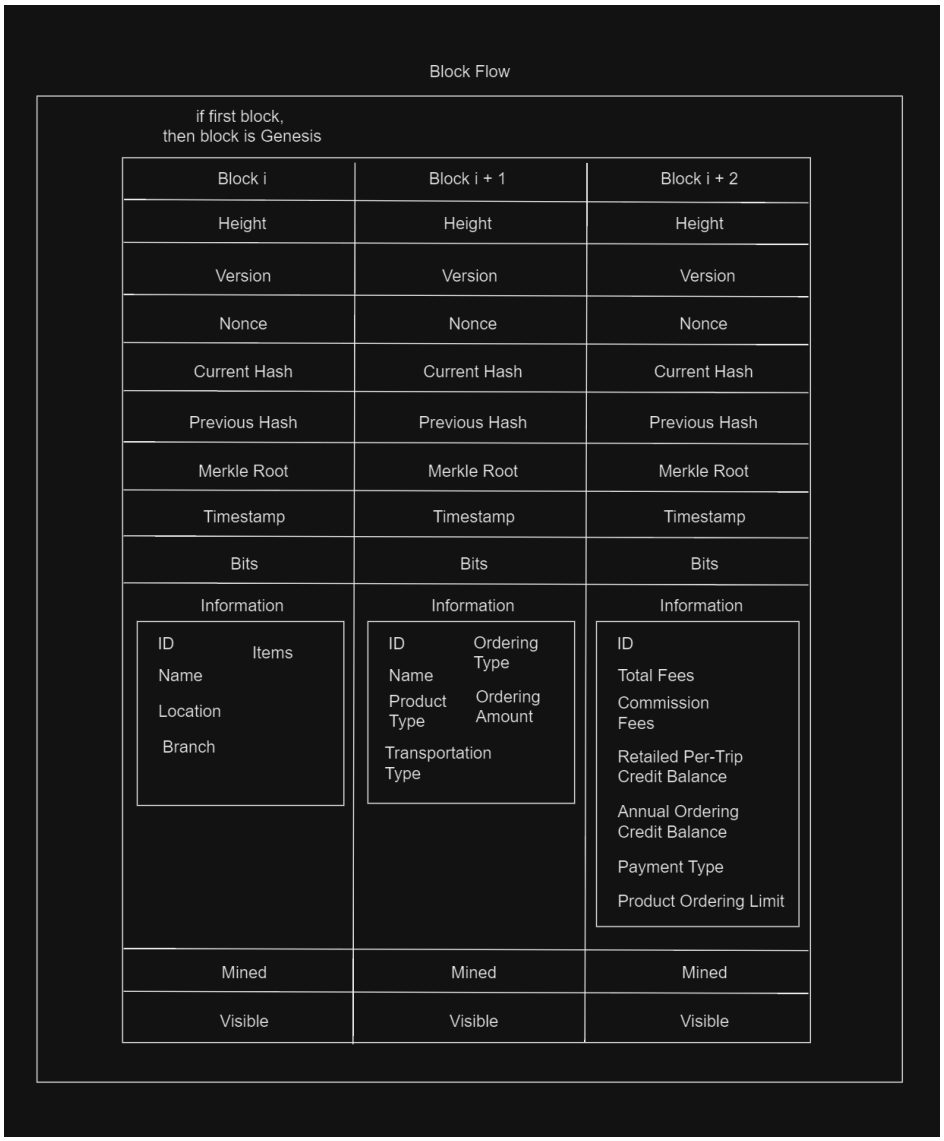

Figure 1: Block Flow

The data fields in the provided blocks serve distinct purposes in a blockchain context. Firstly, the "Block Type" field categorizes the block into Supplier, Transporter, or Transaction types, indicating the role or action associated with the data contained within the block. "Height" denotes the position of the block within the blockchain, indicating its chronological order. The "Version" field specifies the protocol version used for the block, whereby the version of the blockchain can be subjected to newer algorithms or integrations in which the version will change. "Nonce" is a cryptographic value that miners adjust to meet the difficulty target required for block validation. "Current Hash" represents the unique identifier generated for the current block based on its content. "Previous Hash" links the current block to its preceding block, establishing the chain's continuity. The hashes will all be mined until a proper hashing pattern is achieved. This is a common security measure to prevent spam and ensure that adding new blocks requires computational work. The "Merkle Root" field contains the root hash of all transactions within the block, enabling efficient verification of included transactions. "Timestamp" records the time when the block was created, aiding in establishing the sequence of events. "Bits" specifies the target difficulty level for block mining. Finally, the "Information" field varies in content based on the block type but typically includes relevant details such as IDs, names, locations, products, or transaction specifics, providing essential contextual information within the blockchain.

6.2 Block Header Metadata

Each block contains a set of data which will be used to determine the exact transaction information stored within that block as a record.

6.2.1 Supplier Data

ID
Name
Location
Branch
Items

Figure 2: Supplier Header Metadata Fields

The ID serves as the unique identifier for each supplier within the system, facilitating efficient data management and retrieval. The ID ensures that each of the suppliers is uniquely identified and distinguished from the others. Moreover, this is used to easily prompt the participant using selection values instead of manually inserting custom values for the Supplier.

ABC Solutions has multiple branches throughout the nation. Therefore, Names, Locations, and Branches are required to properly identify their entity.

The Items is a long string which allows the user to properly identify the items that the company would like to supply in this one transaction itself.

6.2.2 Transporter Data

ID
Name
Product Type
Transportation Type
Ordering Type
Ordering Amount (in kilograms)

Figure 3: Transporter Header Metadata Fields

Similarly, the ID here is used to uniquely identify the Transporter. The name indicates the company name of the Transporter hired to aid in the transportation process.

Details regarding the transportation, product and ordering type include options for the participant to select from to provide greater options to better suit the delivery of the items. The ordering amount stores the total weightage of all the items to be supplied.

6.2.3 Transaction Data

ID
Total Fees (in Malaysian Ringgit)
Commission Fees (in Malaysian Ringgit)
Retailer Per-Trip Credit Balance (in Malaysian Ringgit)
Annual Ordering Credit Balance (in Malaysian Ringgit)

Payment Type
Product Ordering Limit

Figure 4: Transaction Header Metadata Fields

The total fees are recorded in the data to indicate the total transportation fees, while the Commission Fees will indicate the total subsidized to the Transporter company for transporting the goods. The balances for the Retailer Per-Trip and Annual Ordering is used to ensure that there are sufficient and valid amounts to be used in the transaction process.

7.0 Macro Ideas and Concepts

7.1 Practical Assumptions

The **One-to-One** blockchain design, linking each transporter to a single supplier block and each transaction to a specific transporter block, embodies principles that are especially realistic to environments prioritizing clarity. This model suits in streamlining operations within sectors such as supply chain management, where the precise tracking of goods and services from origin to fulfillment is crucial. Below, we delve deeper into the additional advantages of this design approach, reinforcing its preference for certain applications.

While the **Custom Ordering Design** in blockchain architecture provides flexibility by allowing various configurations of blocks—such as multiple suppliers linked to a single transporter or several transporters serving multiple suppliers—the One-to-One (1-1) design stands out for its simplicity, efficiency, and security. This section, supported by references, justifies why the 1-1 design is often considered superior for specific applications, particularly in environments where clarity, traceability, and auditability are in greater benefits.

7.1.1 Auditability

The 1-1 design greatly simplifies the audit process, making it easier to trace transactions back to their origins. According to Swan (2015) in "Blockchain: Blueprint for a New Economy," the blockchain's inherent transparency and immutability become even more effective in a 1-1 design, as each transaction can be directly traced to a specific block without the ambiguity that might arise in more complex configurations. This direct traceability is crucial for security and fraud prevention, as highlighted by Antonopoulos and Wood (2018) in "Mastering Ethereum," emphasizing how simpler relationships between blocks can reduce the attack surface for malicious actors.

7.1.2 Regulatory Compliance and Data Integrity

Compliance with regulatory standards, especially in industries such as finance, healthcare, and supply chain logistics, is significantly facilitated by the 1-1 design. Tapscott and Tapscott (2016)

in "Blockchain Revolution" discuss how blockchain can revolutionize transparency and accountability in business processes. The 1-1 design, by virtue of its straightforward structure, inherently supports compliance efforts, making it easier for organizations to prove the integrity of their transactions and comply with audit requirements.

Moreover, Mougayar (2016) in "The Business Blockchain" points out that maintaining data integrity is simpler when each block is directly linked to its immediate precursor, minimizing the risk of data corruption or unauthorized alterations. This is particularly relevant in supply chain management, where the provenance of goods needs to be unambiguously verifiable.

Aspect	**One-to-One Design Advantages**	**Custom Ordering Design Disadvantages**
Practicality	Utilizes the widely accepted blockchain architecture, making it easier to understand, implement, and maintain.	Increased complexity can lead to confusion, higher maintenance costs, and a steeper learning curve for participants.
Auditability	Streamlines the audit process, enhancing transparency and traceability with direct, clear links between blocks.	Complex relationships between blocks may cause audit trails, complicating compliance, and dispute resolution.
Data Integrity	Direct linkages reduce the risk of data corruption and unauthorized alterations, bolstering data integrity.	The multiple connections could increase the risk of data breaches and unauthorized access, complicating data integrity efforts.
Security	Minimizes potential attack vectors, enhancing overall network security.	Complex configurations might introduce vulnerabilities, making the system more susceptible to security breaches.

7.2 Proposed Ideas and Concepts

As part of the program design and concept, the supply chain management system's uses text files acting as the database which is designed to enhance the interaction between suppliers, transporters, and transactions. This structure not only streamlines operations but also ensures that each entity within the supply chain can effectively manage and track its resources, services, and financial transactions. The following paragraphs delve into the concepts behind the storage structure for each block variant, focusing on suppliers, transporters, and transactions, and highlighting the flexibility and customization offered through the selection of options from predefined sets.

7.2.1 Supplier (Company Franchise) Predefined Options

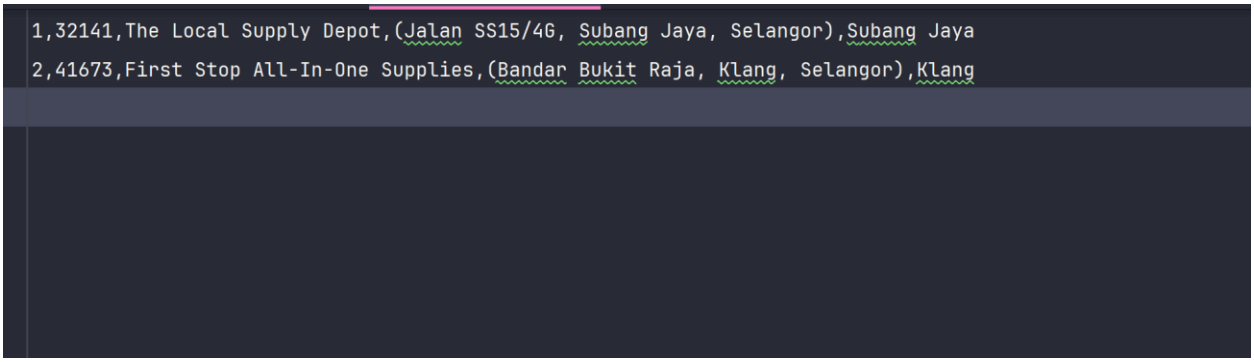

Figure 5: Supplier Options (suppliers.txt)

The "Suppliers" block captures essential details about suppliers, including their ID, name, location, and branch. This structure facilitates a comprehensive understanding of each supplier's geographic presence and operational scope. By allowing the user to select specific details about the supplier's location and branch, the system ensures that businesses can easily identify and collaborate with suppliers that best meet their logistical and operational needs. For instance, a supplier listed as "The Local Supply Depot" in Subang Jaya, Selangor, underscores the system's ability to cater to localized supply needs, thereby optimizing the supply chain's efficiency by leveraging geographical proximity. This approach not only simplifies supplier management but also enhances the supply chain's responsiveness and adaptability to local market dynamics.

7.2.2 Transporter Predefined Options

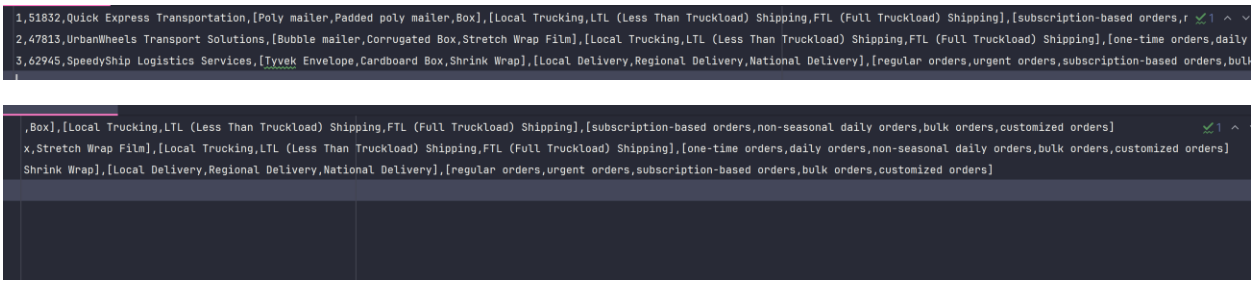

Figure 6: Transporter Options (transporters.txt)

The "Transporters" block is intricately structured to detail the services provided by transportation entities, including their ID, name, product types they handle, transportation types offered, and ordering types catered to. This segment allows for a nuanced selection of transportation options, ranging from local trucking to national delivery, and supports a variety of product types such as poly mailers, cardboard boxes, and shrink wrap. Moreover, it accommodates different ordering types, including bulk orders, urgent orders, and customized orders, highlighting the system's versatility in meeting diverse logistical requirements. By enabling businesses to specify their transportation needs, the database structure facilitates the matching of transportation services with the unique demands of each shipment, thereby optimizing logistics efficiency and cost-effectiveness.

7.2.3 Transaction Predefined Options

```
1,Auto-deduction,[weight (kg),volume (litres),count (number), area (sqft), value (RM)]
2,iPay88,[weight (kg),volume (litres),count (number), area (sqft), value (RM)]
3,PayPal,[weight (kg),volume (litres),count (number), area (sqft), value (RM)]
4,Credit Card,[weight (kg),volume (litres),count (number), area (sqft)]
5,Online Banking,[weight (kg),volume (litres),count (number), area (sqft)]
6,Cash,[weight (kg),count (number), value (RM)]
```

Figure 7: Transaction Options (transactions.txt)

The "Transactions" block is designed to record financial transactions within the supply chain, focusing on retailer per-trip credit balance, annual ordering credit balance, payment type, and

product ordering limit. This structure supports various payment types, such as Auto-deduction, iPay88, PayPal, and Cash, offering flexibility in financial operations and accommodating different business preferences and regulatory requirements. Furthermore, the inclusion of product ordering limits based on weight, volume, count, and value enhances the system's ability to manage and monitor transaction volumes and financial exposure effectively. This level of detail ensures transparency in financial dealings and supports robust financial management within the supply chain ecosystem.

7.2.4 User Participants Predefined Options

```
1,johndoe,123,John Doe,Supplier,Manager
2,janesmith,abc,Jane Smith,Supplier,Manager
3,michaelbrown,456,Michael Brown,Supplier,Employee
4,emilywhite,789,Emily White,Transporter,Manager
5,jamesblack,101,James Black,Transporter,Manager
6,davidgreen,202,David Green,Transporter,Employee
7,emmagray,303,Emma Gray,Accountant,Manager
8,oliviabrown,404,Olivia Brown,Accountant,Manager
9,williamwhite,505,William White,Accountant,Employee
10,isabellawhite,606,Isabella White,Administrator,Manager
```

Figure 8: User Participant Options (participants.txt)

The participant database records essential information for each individual involved in the blockchain, starting from a unique ID to detailed professional designations. This setup includes the username and password for access control, ensuring secure authentication mechanisms are in place to safeguard sensitive information and operations within the blockchain environment. By categorizing participants based on their full name, industry role (Supplier, Transporter, Accountant, Administrator), and specific role (Manager, Employee).

The distinction between industry roles (Supplier, Transporter, Accountant, Administrator) is used for defining permission access within the blockchain. This signifies the access of block mining and creation within only their domain of industry. The “Administrator” role has an oversight of the entire system which grants them the access to have access to any block creation.

7.2.5 Blockchain Records

All the blocks chaining records will be stored in the file *chain.txt* which begins off as a new fresh system application empty. This is crucial to any given blockchain system as blocks get created, these blocks will be recorded and stored in this file and never to be removed to maintain immutability of the digital contract throughout, where data will remain permanent throughout the usage of the system.

7.2.6 Data Realism

The diverse range of options for packaging, transportation, and payment directly reflects the practical needs and preferences of businesses and customers alike. It makes perfect sense because it mirrors the real variety of goods being moved and the different ways people prefer to handle money. For instance, choosing the right type of packaging like bubble mailers for fragile items or sturdy boxes for heavier goods ensures that products reach their destination safely. Having transporters with services ranging from local deliveries to nationwide shipping allows businesses to expand their reach without worrying about logistics. Plus, the mix of payment methods caters to everyone's comfort level, whether they like the ease of online payments or the straightforwardness of cash. This setup not only supports the dynamic nature of commerce but also boosts confidence among customers and suppliers by offering flexible and secure options. It's a smart way to keep everything running smoothly, making sure businesses can meet demands, keep their customers happy, and stay ahead in a competitive market.

7.3 Implementation Importance

The proposed ideas are crucially important and highly motivating for implementation due to their emphasis on enhancing operational efficiency, ensuring robust security, and integrating transparency across the supply chain. By integrating predefined options for suppliers, transporters, and transactions into a blockchain framework, the system not only mirrors the real-world complexity and diversity of modern commerce but also addresses the pressing need for reliable and immutable record-keeping. This approach, particularly the focus on practical assumptions such as the One-to-One blockchain design, streamlines the audit process, significantly facilitates regulatory compliance, and maintains data integrity. Moreover, the attention to user participants and blockchain records underscores a commitment to creating a user-centric system that prioritizes secure access and the immutability of data, fostering trust among all stakeholders. The inclusion of data realism in the system design reflects a deep understanding of the market's needs, ensuring that the ITMS is both relevant and adaptable to the evolving dynamics of supply chain management. This holistic and forward-thinking approach is not just a blueprint for efficiency and security; it's a roadmap towards building resilient, transparent, and sustainable supply chains in the digital age, making the implementation of these ideas an exciting and rewarding endeavor.

8.0 Program Explanations

In our documentation report for ABC Solutions Sdn. Bhd.'s system, we've strategically chosen to concentrate on the major components and foundational concepts that underpin the source code. This decision is guided by the belief that understanding the core architecture and its alignment with blockchain principles is crucial for grasping the system's capabilities and potential for future scalability and adaptability. Smaller, more straightforward details and functionalities are comprehensively described within the code comments themselves. Most comments are vastly and thoroughly explained in *header* files as there is where declarations and typehinting occurs. This approach ensures that the documentation remains focused and uncluttered, facilitating a clearer, more accessible overview for stakeholders and developers alike. It underscores our commitment to transparency and efficiency, ensuring that readers can easily identify and comprehend the system's key mechanisms and how they contribute to achieving our goals of enhanced efficiency, security, and transparency in inventory and transportation management. This combination of in-depth documentation of significant system aspects and detailed code comments for simpler elements is designed to streamline the learning curve and enhance the user's ability to effectively engage with and build upon the system.

Note: A README markdown file is provided within the source code, which contains a brief explanation on the program as well as a thorough manual guide on its usage.

```cpp
#include "../Application.h"

/**
 * @brief Initializes the application.
 * @return
 */
int main() {
    Application app;
    app.init();
    app.run();
}
```

Figure 9: main.cpp

The program begins with the *main.cpp* file. Here, the code begins by initializing the application which will then run the application.

```cpp
#ifndef APPLICATION_H
#define APPLICATION_H

#include "blockchain/Chain.h"
#include "authentication/Participant.h"
#include "filesystem/FileReader.h"

/**
 * @brief The Application class manages the initialization and execution of the
application.
 */
class Application {
public:
    /**
     * @brief Initializes the application.
     */
    static void init();

    /**
     * @brief Runs the application.
     */
    static void run();

    /**
     * @brief Sets the blockchain instance.
     * @param blockchain Pointer to the blockchain.
     */
    static void setBlockchain(blockchain::Chain* blockchain);

    /**
     * @brief Sets the redacted blockchain instance.
     * @param redactedBlockchain Pointer to the redacted blockchain.
     */
    static void setRedactedBlockchain(blockchain::Chain* redactedBlockchain);

    /**
     * @brief Sets the list of participants.
     * @param participants List of participants.
     */
    static void setParticipants(const std::vector<authentication::Participant>&
participants);

    /**
     * @brief Sets the current participant.
     * @param user Pointer to the current participant.
     */
    static void setCurrentParticipant(std::unique_ptr<authentication::Participant>
user);

    /**
     * @brief Sets the list of blocks.
     * @param blocks List of block data.
     */
    static void setBlocks(const std::vector<filesystem::BlockData>& blocks);

    /**
     * @brief Sets the type of the last block.
     * @param type Type of the last block.
     */
    static void setLastBlockType(blockchain::enums::BlockType type);

    /**
     * @brief Gets the blockchain instance.
```

```
     */
    static blockchain::Chain* getBlockchain();

    /**
     * @brief Gets the redacted blockchain instance.
     */
    static blockchain::Chain* getRedactedBlockchain();

    /**
     * @brief Gets the list of participants.
     */
    static std::vector<authentication::Participant>& getParticipants();

    /**
     * @brief Gets the current participant.
     */
    static authentication::Participant* getCurrentParticipant();

    /**
     * @brief Gets the list of blocks.
     */
    static std::vector<filesystem::BlockData>& getBlocks();

    /**
     * @brief Gets the type of the last block.
     */
    static blockchain::enums::BlockType getLastBlockType();

private:
    static blockchain::Chain* blockchain; /**< Pointer to the blockchain instance. */
    static blockchain::Chain* redactedBlockchain; /**< Pointer to the redacted
blockchain instance. */
    static std::vector<authentication::Participant> participants; /**< List of
participants. */
    static std::unique_ptr<authentication::Participant> currentParticipant; /**<
Pointer to the current participant. */
    static std::vector<filesystem::BlockData> blocks; /**< List of block data. */
    static blockchain::enums::BlockType lastBlockType; /**< Type of the last block. */

    /**
     * @brief Initializes the blockchain.
     */
    static void initializeBlockchain();

    /**
     * @brief Authenticates the user.
     */
    static void authenticateUser();

    /**
     * @brief Displays the application menu.
     */
    static void displayMenu();

    /**
     * @brief Initializes dependencies required by the application.
     */
    static void initDependencies();
};

#endif // APPLICATION_H
```

Figure 10: Application.h

```cpp
#include "Application.h"
#include "blockchain/Chain.h"
#include "filesystem/FileReader.h"
#include "../data/Config.h"
#include "authentication/Login.h"
#include "collection/InputCollector.h"
#include "collection/conversion/DataConverter.h"
#include "collection/validator/InputValidator.h"
#include <iostream>
#include <memory>
#include <vector>
#include <functional>

// Static member variable initialization
blockchain::Chain* Application::blockchain = nullptr;
blockchain::Chain* Application::redactedBlockchain = nullptr;
std::vector<authentication::Participant> Application::participants;
std::unique_ptr<authentication::Participant> Application::currentParticipant =
nullptr;
std::vector<filesystem::BlockData> Application::blocks;
blockchain::enums::BlockType Application::lastBlockType;

void Application::init() {
    srand(static_cast<unsigned int>(time(0))); // Seed for random number generation
    initDependencies();

    initializeBlockchain();
}

void Application::initDependencies() {
    /**
     * @brief The chain of blocks where all blocks are from the real data blockchain
record.
     */
    setBlockchain(new blockchain::Chain(data::Config::RECORDS_BLOCKCHAIN_FILE_PATH,
data::Config::VERSION));

    /**
     * @brief The chain of blocks where some blocks are hidden (redacted) from the
display view output to the currentParticipant.
     * aka. The temporary storage of the blockchain data.
     */
    setRedactedBlockchain(new
blockchain::Chain(data::Config::RECORDS_BLOCKCHAIN_FILE_PATH, data::Config::VERSION));

    /**
     * @brief The list of blocks in the blockchain network.
     */
    filesystem::FileReader fileReaderChain(data::Config::RECORDS_BLOCKCHAIN_FILE_PATH,
filesystem::DataType::CHAIN);
    const auto& blocks = fileReaderChain.getBlocks();
    setBlocks(blocks);

    /**
     * @brief The list of participants in the blockchain network.
     */
    filesystem::FileReader fileReaderParticipant(data::Config::PARTICIPANTS_FILE_PATH,
filesystem::DataType::PARTICIPANTS);
    const auto& participants = fileReaderParticipant.getParticipants();
    setParticipants(participants);
}
```

```
void Application::initializeBlockchain() {
    // Set the last block type to TRANSACTION as default
    setLastBlockType(blockchain::enums::BlockType::TRANSACTION); // Default to
Transaction if no blocks were added

    // Check if there are any blocks stored
    if (!blocks.empty()) {
        // Iterate over each BlockData and add the corresponding block to the
blockchain
        for (const auto& blockData : blocks) {
            if (blockData.type == blockchain::enums::BlockType::SUPPLIER) {
                // Create a SupplierBlock from BlockData and add it to the blockchain
                auto block =
std::make_shared<blockchain::SupplierBlock>(conversion::DataConverter::convertToSuppli
erBlock(data::Config::VERSION, blockchain->getBits(), blockData.height,
blockData.nonce, blockData.currentHash, blockData.previousHash, blockData.information,
blockData.visible));
                blockchain->addBlock(block);
                redactedBlockchain->addBlock(block);
            } else if (blockData.type == blockchain::enums::BlockType::TRANSPORTER) {
                // Create a TransporterBlock from BlockData and add it to the
blockchain
                auto block =
std::make_shared<blockchain::TransporterBlock>(conversion::DataConverter::convertToTra
nsporterBlock(data::Config::VERSION, blockchain->getBits(), blockData.height,
blockData.nonce, blockData.currentHash, blockData.previousHash, blockData.information,
blockData.visible));
                blockchain->addBlock(block);
                redactedBlockchain->addBlock(block);
            } else if (blockData.type == blockchain::enums::BlockType::TRANSACTION) {
                // Create a TransactionBlock from BlockData and add it to the
blockchain
                auto block =
std::make_shared<blockchain::TransactionBlock>(conversion::DataConverter::convertToTra
nsactionBlock(data::Config::VERSION, blockchain->getBits(), blockData.height,
blockData.nonce, blockData.currentHash, blockData.previousHash, blockData.information,
blockData.visible));
                blockchain->addBlock(block);
                redactedBlockchain->addBlock(block);
            }

            setLastBlockType(blockData.type);
        }
    }
}

void Application::authenticateUser() {
    // Authenticate the currentParticipant
    authentication::Login login;
    bool isAuthenticated = false;
    std::string username, password;

    while (!isAuthenticated) {
        std::tie(username, password) =
collection::InputCollector::collectLoginDetails();

        for (const auto &participant: participants) {
            if (login.authenticate(username, password, participant)) {
                isAuthenticated = true;

setCurrentParticipant(std::make_unique<authentication::Participant>(participant));
                break; // Exit the for loop
```

```
            }
        }

        if (!isAuthenticated) {
            std::cout << "Invalid username or password. Please try again." <<
std::endl << std::endl;
        } else {
            break; // Exit the while loop since authentication was successful
        }
    }

    // Proceed with the authenticated currentParticipant.
    std::cout << "Welcome back, " << currentParticipant->getFullName() << "!" <<
std::endl << std::endl;
}

void Application::displayMenu() {
    // Function vector to cycle through, using the InputCollector class for data
collection
    std::vector<std::function<void()>> functions = {
            [&]{
                // Collect information for Supplier block
                auto info =
collection::InputCollector::collectSupplierInfo(data::Config::OPTIONS_SUPPLIER_FILE_PA
TH);
                // Create a SupplierBlock and add it to the blockchain
                auto block =
std::make_shared<blockchain::SupplierBlock>(data::Config::VERSION,
blockchain->getBits(), blockchain->getNextBlockHeight(),
blockchain->getLastBlockHash(), info);
                blockchain->addBlock(block).addToRecord();
                redactedBlockchain->addBlock(block);
            },
            [&]{
                // Collect information for Transporter block
                auto info =
collection::InputCollector::collectTransporterInfo(data::Config::OPTIONS_TRANSPORTER_F
ILE_PATH);
                // Create a TransporterBlock and add it to the blockchain
                auto block =
std::make_shared<blockchain::TransporterBlock>(data::Config::VERSION,
blockchain->getBits(), blockchain->getNextBlockHeight(),
blockchain->getLastBlockHash(), info);
                blockchain->addBlock(block).addToRecord();
                redactedBlockchain->addBlock(block);
            },
            [&]{
                // Collect information for Transaction block
                auto info =
collection::InputCollector::collectTransactionInfo(data::Config::OPTIONS_TRANSACTION_F
ILE_PATH, data::Config::RECORDS_BLOCKCHAIN_FILE_PATH);
                // Create a TransactionBlock and add it to the blockchain
                auto block =
std::make_shared<blockchain::TransactionBlock>(data::Config::VERSION,
blockchain->getBits(), blockchain->getNextBlockHeight(),
blockchain->getLastBlockHash(), info);
                blockchain->addBlock(block).addToRecord();
                redactedBlockchain->addBlock(block);
            }
    };

    // Define options for user actions and block search criteria
    std::vector<std::string> actionOptions = { "Display blockchain", "Search Block",
```

```
"Add Block", "Manipulate Block" };
    std::vector<std::string> searchOptions = { "Block Type", "Height", "Version",
"Nonce", "Current Hash", "Previous Hash", "Merkle Root", "Timestamp", "Bits",
"Information" };

    // Determine the index for selecting the next type of block to add
    int index; // Default to Supplier if no blocks were added or if the last block was
Transaction
    switch (getLastBlockType()) {
        case blockchain::enums::BlockType::SUPPLIER:
            index = 1; // Next should be Transporter
            break;
        case blockchain::enums::BlockType::TRANSPORTER:
            index = 2; // Next should be Transaction
            break;
        case blockchain::enums::BlockType::TRANSACTION:
        default:
            index = 0; // Default to Supplier if the last block was Transaction or no
blocks were added
            break;
    }

    // Get the industry role of the current participant
    auto participantIndustryRole = currentParticipant->getIndustryRole();

    // Loop for user interaction
    do {
        // InputCollector user for action
        int action = collection::validation::InputValidator::validateSelectionInt("an
action", actionOptions);

        switch (action) {
            case 1:
                // Display blockchain
                if (redactedBlockchain->isEmpty()) {
                    std::cout << "No blocks found in the blockchain." << std::endl <<
std::endl;
                } else {
                    redactedBlockchain->displayAll();
                }
                break;
            case 2: {
                // Search block
                auto [searchAttr, searchValue] =
collection::InputCollector::collectSearchCriteria(searchOptions);

                blockchain->display(blockchain->searchBlockByAttr(searchAttr,
searchValue));
                break;
            }
            case 3:
                // Add block
                if ((participantIndustryRole == "Supplier" && index == 0) ||
                    (participantIndustryRole == "Transporter" && index == 1) ||
                    (participantIndustryRole == "Accountant" && index == 2) ||
                    (participantIndustryRole == "Administrator")) {

                    // Call the function at the current index
                    functions[index]();
                    // Move to the next type of block or loop back to the start
                    index = (index + 1) % functions.size();
                } else {
                    std::cout << "The current block requires intervention from a
```

```
different industry role." << std::endl << std::endl;
                }
                break;
            case 4:
                // Manipulate block

collection::InputCollector::collectBlockManipulationCriteria(*blockchain,
*redactedBlockchain, searchOptions);
                break;
        }
    } while (true);
}

void Application::run() {
    std::cout << "Enter values based on the given options, custom values or 'q'/'quit'
to exit the program anytime." << std::endl << std::endl;

    // Enable user interaction with the application
    authenticateUser();
    displayMenu();
}

// getters
blockchain::Chain* Application::getBlockchain() { return blockchain; }
blockchain::Chain* Application::getRedactedBlockchain() { return redactedBlockchain; }
std::vector<authentication::Participant>& Application::getParticipants() { return
participants; }
authentication::Participant* Application::getCurrentParticipant() { return
currentParticipant.get(); }
std::vector<filesystem::BlockData>& Application::getBlocks() { return blocks; }
blockchain::enums::BlockType Application::getLastBlockType() { return lastBlockType; }

// setters
void Application::setBlockchain(blockchain::Chain* chain) { blockchain = chain; }
void Application::setRedactedBlockchain(blockchain::Chain* chain) { redactedBlockchain
= chain; }
void Application::setParticipants(const std::vector<authentication::Participant>&
participants) { Application::participants = participants; }
void Application::setCurrentParticipant(std::unique_ptr<authentication::Participant>
user) { currentParticipant = std::move(user); }
void Application::setBlocks(const std::vector<filesystem::BlockData>& blocks)
{ Application::blocks = blocks; }
void Application::setLastBlockType(blockchain::enums::BlockType type) { lastBlockType
= type; }
```

Figure 11: Application.cpp

The code above are both *Application.h* and *Application.cpp* implementation files The header file for the Application which enables the compiler to know about types, functions, and other type hinting and declarations. The methods defined includes the main functionality init() and run() which will initialized the application then run the menu. Multiple **setters and getters accessors** were used throughout the implementation.

During the initialization of the application, it will first initialize the blockchain and redactedBlockchain (a temp storage used for soft data operations), as well as all the blocks and participants that have been found in the *chain.txt* and *participants.txt* if blocks data are found. The

data will be read by the fileReader. The blocks then gets appended to both the blockchain and redactedBlockchain both with separate copies.

Enums cases were also used to ensure better readability and **type safety** for the integral constants of the blocks. In order to maintain the one-to-one design, the last block is being tracked to identify the compulsory next block in the operation. This will then properly ensure the design architecture is properly implemented.

After the initialization, the user will then be prompted to verify that he / she is a verified participant in the system by inserting the username, and password. Upon successful authentication, the main menu will be displayed to the user where the user gets to choose to either Display the existing Blockchain, Search for one or more blocks at an instance (using any existing attributes such as Block Type, Height, Nonce, Current or Previous Hashes, etc.). Add blocks, or Manipulate Blocks.

Blocks creation are restricted to only permitted users based on their industry role with exemption that the participant is an administrator.

```
#ifndef CONFIG_H
#define CONFIG_H

#include <string>

namespace data {
    class Config {
    public:
        static const int VERSION; /** The version of the blockchain */
        static const std::string RECORDS_BLOCKCHAIN_FILE_PATH; /** The path to the
blockchain file */
        static const std::string OPTIONS_SUPPLIER_FILE_PATH; /** The path to the
supplier options file */
        static const std::string OPTIONS_TRANSPORTER_FILE_PATH; /** The path to the
transporter options file */
        static const std::string OPTIONS_TRANSACTION_FILE_PATH; /** The path to the
transaction options file */
        static const std::string PARTICIPANTS_FILE_PATH; /** The path to the
participants file */
    };
} // namespace blockchain

#endif // CONFIG_H
```

Figure 12: Config.h

A Configuration class under files *Config.h* and *Config.cpp* is being used as an global environment file which stores env values that will be used throughout the application as constants. They mainly store the version of the application version as well as the database file paths. This ensures backend

portability by externalizing the configuration settings. By which anytime the code can be deployed to different environments without much modifications

```
#pragma once

#include <vector>
#include <memory>
#include "Block.h"
#include "enums/BlockAttribute.h"

namespace blockchain {
    class Chain {
    public:
        /**
         * @brief A version number indicating which set of block validation rules to
follow.
         */
        static const int VERSION;

        /**
         * @brief A string of bits that must be satisfied by the hash of a block.
         *
         * @param dataFilePath
         * @param version
         * @param bits
         */
        Chain(const std::string dataFilePath, const int version, const std::string&
bits = "ffff001f");

        /**
         * @brief Add a block to the blockchain.
         *
         * @param block
         * @return
         */
        Chain& addBlock(std::shared_ptr<Block> block);

        /**
         * @brief Edit a block in the blockchain.
         *
         * @param block
         * @param info
         * @return
         */
        Chain& editBlock(std::shared_ptr<Block> block, const std::string& info);

        /**
         * @brief "Removes" a block from the blockchain by flagging it as invisible.
         *
         * @param block
         * @param info
         * @return
         */
        Chain& hardEditBlock(std::shared_ptr<Block> block, const std::string& info);

        /**
         * @brief Temporarily hide a block from the blockchain.
         * The block still remain in the blockchain data record but will not be
displayed
         *
         * @param block
```

```
         * @return
         */
        Chain& hideBlock(std::shared_ptr<Block> block);

        /**
         * @brief "Removes" a block from the blockchain by flagging it as invisible.
         * The block still remains in the blockchain data record due to immutability
but will not be displayed.
         *
         * @param block
         * @return
         */
        Chain& hardHideBlock(std::shared_ptr<Block> block);

        /**
         * @brief Mine a block in the blockchain.
         *
         * @param block
         * @return
         */
        Chain& mineBlock(std::shared_ptr<Block> block);

        /**
         * @brief Display all blocks in the blockchain.
         */
        void displayAll() const;

        /**
         * @brief Display selected blocks in the blockchain.
         *
         * @param selectedBlocks
         */
        void display(const std::vector<std::shared_ptr<Block>>& selectedBlocks) const;

        /**
         * @brief Search for a block in the blockchain by attribute.
         *
         * @param attribute
         * @param value
         * @return
         */
        [[nodiscard]] std::vector<std::shared_ptr<Block>>
searchBlockByAttr(blockchain::enums::BlockAttribute attribute, const std::string&
value) const;

        /**
         * @brief Get the next block height in the blockchain.
         *
         * @return
         */
        [[nodiscard]] int getNextBlockHeight() const;

        /**
         * @brief Get the last block hash in the blockchain.
         *
         * @return
         */
        [[nodiscard]] std::string getLastBlockHash() const;

        /**
         * @brief Get a block by its height in the chain.
         *
         * @param height
```

```
         * @return
         */
        std::shared_ptr<Block> getBlockByHeight(int height);

        /**
         * @brief Get the last block in the chain.
         *
         * @return
         */
        [[nodiscard]] bool isEmpty() const;

        /**
         * @brief Get the last block in the chain.
         *
         * @return
         */
        void addToRecord();

        [[nodiscard]] std::string getBits() const { return bits; }

    private:
        /**
         * @brief The path to the blockchain data file.
         */
        const std::string dataFilePath;

        /**
         * @brief The version number indicating which set of block validation rules to
follow.
         */
        int version;

        /**
         * @brief A string of bits that must be satisfied by the hash of a block.
         */
        std::string bits;

        /**
         * @brief The list of blocks in the blockchain.
         */
        std::vector<std::shared_ptr<Block>> blocks;

        /**
         * @brief The list of blocks in the blockchain.
         * @param block
         * @param filename
         */
        void logBlockDetails(const Block& block, const std::string& filename);

        /**
         * @brief Display the details of a block.
         *
         * @param block
         */
        void displayBlockDetails(const std::shared_ptr<Block> &block) const;

        /**
         * @brief Check if the blockchain has any visible blocks.
         *
         * @param blocks
         * @return
         */
        [[nodiscard]] bool hasVisibleBlocks(const std::vector<std::shared_ptr<Block>>&
```

```
blocks) const;
    };
}
```

Figure 13: Chain.h

```
#include "Chain.h"
#include "../filesystem/FileWriter.h"
#include "enums/BlockAttribute.h"
#include <iostream>

namespace blockchain {
    /**
     * @brief Construct a new Chain object.
     *
     * @param dataFilePath
     * @param version
     * @param bits
     */
    Chain::Chain(const std::string dataFilePath, const int version, const std::string&
bits) : dataFilePath(dataFilePath), version(version), bits(bits) {}

    /**
     * @brief Add a block to the blockchain.
     *
     * @param block
     * @return
     */
    Chain& Chain::addBlock(std::shared_ptr<Block> block) {
        // Set the genesis flag for the block
        block->setGenesis(block->getHeight() == 0);
        blocks.push_back(std::move(block));

        return *this; // Enable chaining of operations
    }

    /**
     * @brief Virtually Edit a block in the blockchain.
     *
     * @param block
     * @param info
     * @return
     */
    Chain& Chain::editBlock(std::shared_ptr<Block> block, const std::string& info) {
        auto clonedBlock = block->clone(); // Clone the block to avoid modifying the
original
        clonedBlock->getHeader().updateEditableData(info);

        auto it = std::find(blocks.begin(), blocks.end(), block);

        // If found, replace it with the new block
        if (it != blocks.end()) {
            *it = clonedBlock;
        }

        return *this; // Enable chaining of operations
    }

    /**
     * @brief Edit a block in the blockchain with verifiable and persistent changes.
     *
     * @param block
     * @param info
```

```
     * @return
     */
    Chain& Chain::hardEditBlock(std::shared_ptr<Block> block, const std::string& info)
{
        // Find the index of the block that was edited
        auto it = std::find_if(blocks.begin(), blocks.end(),
                               [&block](const std::shared_ptr<Block>& b) {
                                   return b == block;
                               });

        if (it == blocks.end()) {
            // Block not found
            return *this;
        }

        // Calculate the position (index) for the edited block
        size_t startIndex = std::distance(blocks.begin(), it);

        // Update the current block first
        if (startIndex < blocks.size()) {
            blocks[startIndex]->getHeader().updateEditableData(info, (startIndex >
0) ? blocks[startIndex - 1]->getHeader().getHash() : "");
        }

        // Now, update the previous hash in subsequent blocks to maintain chain
integrity
        for (size_t i = startIndex + 1; i < blocks.size(); ++i) {

blocks[i]->getHeader().updateEditableData(blocks[i]->getHeader().getInformationString(
), blocks[i - 1]->getHeader().getHash());
        }

        filesystem::FileWriter::clearFile(dataFilePath); // Clear the file before
writing the updated data
        for (const auto& itBlock : blocks) {
            logBlockDetails(*itBlock, dataFilePath);
        }

        return *this; // Enable chaining of operations
    }

    /**
     * @brief Temporarily hide a block from the blockchain.
     *
     * @param block
     * @return
     */
    Chain& Chain::hideBlock(std::shared_ptr<Block> block) {
        // The lambda captures block by value since it's a shared_ptr
        auto it = std::remove_if(blocks.begin(), blocks.end(),
                                 [block](const std::shared_ptr<Block>& itBlock) {
                                     return itBlock == block;
                                 });

        blocks.erase(it, blocks.end()); // Erase the specified block

        return *this; // Enable chaining of operations
    }

    /**
     * @brief "Removes" a block from the blockchain by flagging it as invisible.
     * The block still remains in the blockchain data record due to immutability but
will not be displayed.
```

```
     *
     * @param block
     * @return
     */
    Chain& Chain::hardHideBlock(std::shared_ptr<Block> block) {
        // The lambda captures block by value since it's a shared_ptr
        auto it = std::find(blocks.begin(), blocks.end(), block);

        if (it != blocks.end()) {
            (*it)->setVisible(false);
        }

        filesystem::FileWriter::clearFile(dataFilePath); // Clear the file before
writing the updated data
        for (const auto& itBlock : blocks) {
            logBlockDetails(*itBlock, dataFilePath);
        }

        return *this; // Enable chaining of operations
    }

    /**
     * @brief Mine a block in the blockchain.
     *
     * @param block
     * @return
     */
    Chain& Chain::mineBlock(std::shared_ptr<Block> block) {
        auto it = std::find(blocks.begin(), blocks.end(), block);

        std::string newHash;
        if (it != blocks.end()) {
            newHash =
(*it)->getHeader().mine(blockchain::BlockHeader::getHashFunction(block->getType()));
            (*it)->getHeader().setHash(newHash);

            // If this is the genesis block (the first block), set its prevHash to its
new hash
            if (std::distance(blocks.begin(), it) == 0) {
                (*it)->getHeader().setPrevHash(newHash);
            }

            // For subsequent blocks, update the next block's previous hash if there
is one
            auto nextIt = std::next(it);
            if (nextIt != blocks.end()) {
                (*nextIt)->getHeader().setPrevHash(newHash);
            }
        }

        filesystem::FileWriter::clearFile(dataFilePath); // Clear the file before
writing the updated data
        for (const auto& itBlock : blocks) {
            logBlockDetails(*itBlock, dataFilePath);
        }

        return *this; // Enable chaining of operations
    }

    /**
     * @brief Check if the blockchain is empty.
     *
     * @return
```

```
     */
    bool Chain::isEmpty() const {
        return blocks.empty();
    }

    /**
     * @brief Display all blocks in the blockchain.
     */
    void Chain::displayAll() const {
        if (hasVisibleBlocks(blocks)) {
            std::cout << std::endl << "---------------------------------- BLOCKCHAIN
----------------------------------" << std::endl << std::endl;
            for (const auto& block : blocks) {
                if (block->isVisible()) {
                    displayBlockDetails(block);
                }
            }
            std::cout << "------------------------------------------------------------------
------------------------" << std::endl << std::endl;
        } else {
            std::cout << "No blocks found in the blockchain." << std::endl <<
std::endl;
        }
    }

    /**
     * @brief Display the details of the selected blocks.
     *
     * @param selectedBlocks
     */
    void Chain::display(const std::vector<std::shared_ptr<Block>>& selectedBlocks)
const {
        if (!selectedBlocks.empty() && hasVisibleBlocks(selectedBlocks)) {
            std::cout << "----------------------------------- SELECTED BLOCKS -------
-----------------------------" << std::endl << std::endl;

            for (const auto& block : selectedBlocks) {
                if (block->isVisible()) {
                    displayBlockDetails(block);
                }
            }

            std::cout << "------------------------------------------------------------------
------------------------" << std::endl << std::endl;
        } else {
            std::cout << "No blocks found matching the criteria." << std::endl <<
std::endl;
        }
    }

    /**
     * @brief Check if the blocks vector has any visible blocks.
     *
     * @param blocks
     * @return
     */
    bool Chain::hasVisibleBlocks(const std::vector<std::shared_ptr<Block>>& blocks)
const {
        return std::any_of(blocks.begin(), blocks.end(), [](const
std::shared_ptr<Block>& block) {
            return block->isVisible();
        });
    }
```

```
    /**
     * @brief Search for a block by a specific attribute.
     *
     * @param attribute
     * @param value
     * @return
     */
    std::vector<std::shared_ptr<Block>>
Chain::searchBlockByAttr(blockchain::enums::BlockAttribute attribute, const
std::string& value) const {
        std::vector<std::shared_ptr<Block>> foundBlocks; // Store shared pointers to
the found blocks

        // Iterate through the blocks and search for the specified attribute
        for (const auto& block : blocks) {
            bool found = false;
            std::string attrValue;

            switch (attribute) {
                case blockchain::enums::BlockAttribute::TYPE:
                    attrValue =
blockchain::enums::BlockTypeUtils::toString(block->getType());
                    found = attrValue == value;
                    break;
                case blockchain::enums::BlockAttribute::HEIGHT:
                    attrValue = std::to_string(block->getHeight());
                    found = attrValue == value;
                    break;
                case blockchain::enums::BlockAttribute::VERSION:
                    attrValue = std::to_string(version);
                    found = attrValue == value;
                    break;
                case blockchain::enums::BlockAttribute::NONCE:
                    attrValue = std::to_string(block->getNonce());
                    found = attrValue == value;
                    break;
                case blockchain::enums::BlockAttribute::HASH:
                    attrValue = block->getHeader().getHash();
                    found = attrValue == value;
                    break;
                case blockchain::enums::BlockAttribute::PREV_HASH:
                    attrValue = block->getHeader().getPrevHash();
                    found = attrValue == value;
                    break;
                case blockchain::enums::BlockAttribute::MERKLE_ROOT:
                    attrValue = block->getHeader().getMerkleRoot();
                    found = attrValue == value;
                    break;
                case blockchain::enums::BlockAttribute::TIMESTAMP:
                    attrValue = std::to_string(block->getHeader().getTimestamp());
                    found = attrValue == value;

                    if (!found) {
                        attrValue = block->getHeader().getFormattedTimestamp();
                        found = (attrValue == value);
                    }
                    break;
                case blockchain::enums::BlockAttribute::BITS:
                    attrValue = bits;
                    found = attrValue == value;
                    break;
                case blockchain::enums::BlockAttribute::INFORMATION:
```

```
                    attrValue = block->getHeader().getInformationString();
                    found = attrValue == value;
                    break;
                case enums::BlockAttribute::MINED:
                    attrValue = block->getHeader().isMined() ? "Yes" : "No";
                    found = attrValue == value;
                    break;
                case enums::BlockAttribute::VISIBLE:
                    // Visibility not needed to be shown to the users
                    break;
            }

            if (found) {
                foundBlocks.push_back(block); // Directly use the shared_ptr from the
blocks vector
            }
        }

        // Return the found blocks
        return foundBlocks;
    }

    /**
     * @brief Display the details of a block according to the specified attribute.
     *
     * @param block
     */
    void Chain::displayBlockDetails(const std::shared_ptr<Block> &block) const {
        std::cout << enums::BlockAttributeUtils::toString(enums::BlockAttribute::TYPE)
<< " --> " << enums::BlockTypeUtils ::toString(block->getType()) << std::endl
                  <<
enums::BlockAttributeUtils::toString(enums::BlockAttribute::HEIGHT) << " --> " <<
block->getHeight() << (block->isGenesis() ? " (Genesis Block)" : "") << std::endl
                  <<
enums::BlockAttributeUtils::toString(enums::BlockAttribute::VERSION) << " --> " <<
version << std::endl
                  <<
enums::BlockAttributeUtils::toString(enums::BlockAttribute::NONCE) << " --> " <<
block->getNonce() << std::endl
                  << enums::BlockAttributeUtils::toString(enums::BlockAttribute::HASH)
<< " --> " << block->getHeader().getHash() << std::endl
                  <<
enums::BlockAttributeUtils::toString(enums::BlockAttribute::PREV_HASH) << " --> " <<
block->getHeader().getPrevHash() << std::endl
                  <<
enums::BlockAttributeUtils::toString(enums::BlockAttribute::MERKLE_ROOT) << " --> " <<
block->getHeader().getMerkleRoot() << std::endl
                  <<
enums::BlockAttributeUtils::toString(enums::BlockAttribute::TIMESTAMP) << " --> " <<
block->getHeader().getTimestamp() << " (" + block->getHeader().getFormattedTimestamp()
+ ")" << std::endl
                  << enums::BlockAttributeUtils::toString(enums::BlockAttribute::BITS)
<< " --> " << bits << std::endl
                  <<
enums::BlockAttributeUtils::toString(enums::BlockAttribute::INFORMATION) << " --> " <<
"[ " << block->getHeader().getInformationString() << " ]" << std::endl
                  <<
enums::BlockAttributeUtils::toString(enums::BlockAttribute::MINED) << " --> " <<
(block->getHeader().isMined() ? "Yes" : "No") << std::endl << std::endl;
    }

    /**
     * @brief Get the height of the next block in the chain.
```

```
     *
     * @return int The height of the next block in the chain.
     */
    int Chain::getNextBlockHeight() const {
        return static_cast<int>(blocks.size());
    }

    /**
     * @brief Get a block by its height in the chain.
     *
     * @param height
     * @return
     */
    std::shared_ptr<Block> Chain::getBlockByHeight(int height) {
        if (height >= 0 && height < blocks.size()) {
            return blocks[height];
        }

        return nullptr; // Return nullptr if no block found at the specified height
    }

    /**
     * @brief Get the hash of the last block in the chain.
     *
     * @return std::string The hash of the last block in the chain.
     */
    std::string Chain::getLastBlockHash() const {
        if (!blocks.empty()) {
            return blocks.back()->getHeader().getHash();
        }
        return "0"; // Return a default hash for the genesis block
    }

    /**
     * @brief Log the details of a block to a file.
     *
     * @param block The block to log.
     * @param filename The name of the file to write (log) to.
     */
    void Chain::logBlockDetails(const Block& block, const std::string& filename) {
        using namespace blockchain::enums; // Use the entire namespace

        filesystem::FileWriter writer(filename);

        writer.writeLine(BlockAttributeUtils::toString(BlockAttribute::TYPE) + ": " +
blockchain::enums::BlockTypeUtils::toString(block.getType()));
        writer.writeLine(BlockAttributeUtils::toString(BlockAttribute::HEIGHT) + ": "
+ std::to_string(block.getHeight()));
        writer.writeLine(BlockAttributeUtils::toString(BlockAttribute::VERSION) + ": "
+ std::to_string(version));
        writer.writeLine(BlockAttributeUtils::toString(BlockAttribute::NONCE) + ": " +
std::to_string(block.getNonce()));
        writer.writeLine(BlockAttributeUtils::toString(BlockAttribute::HASH) + ": " +
block.getHeader().getHash());
        writer.writeLine(BlockAttributeUtils::toString(BlockAttribute::PREV_HASH) + ":
" + block.getHeader().getPrevHash());
        writer.writeLine(BlockAttributeUtils::toString(BlockAttribute::MERKLE_ROOT) +
": " + block.getHeader().getMerkleRoot());
        writer.writeLine(BlockAttributeUtils::toString(BlockAttribute::TIMESTAMP) + ":
" + std::to_string(block.getHeader().getTimestamp()));
        writer.writeLine(BlockAttributeUtils::toString(BlockAttribute::BITS) + ": " +
bits);
        writer.writeLine(BlockAttributeUtils::toString(BlockAttribute::INFORMATION) +
```

```
": " + block.getHeader().getInformationString());
        writer.writeLine(BlockAttributeUtils::toString(BlockAttribute::MINED) + ": " +
(block.getHeader().isMined() ? "true" : "false"));
        writer.writeLine(BlockAttributeUtils::toString(BlockAttribute::VISIBLE) + ": "
+ (block.isVisible() ? "true" : "false"));
        writer.writeLine(""); // Add an empty line for readability
    }

    /**
     * @brief Add the blockchain to the record.
     */
    void Chain::addToRecord() {
        if (!blocks.empty()) {
            const auto& block = blocks.back();
            logBlockDetails(*block, dataFilePath);
        }
    }
}
```

Figure 14: Chain.cpp

Above are the codes for the *Chain.cpp* and *Chain.h.* This is one of the main classes which focuses on adding blocks to the chain whenever one is created as well as adding to the database record whenever one is created. Moreover, it handles the **dynamic blockchain** functionalities of all four of the operations: **Soft Edit, Hard Edit, Soft Delete, Hard Delete**. (More will be explained later). Blocks are also being mined here by performing operations on the Block Header.

Just like multiple other classes, this class is wrapped in a **namespace** of blockchain. This namespace contains multiple other classes in it. As the codebase system is huge and large due to proper engineering of future adaptability and better organization, it is crucial to use **namespace** as it prevents naming collisions by encapsulating identifiers within a specific scope. This is particularly useful for the system's latge base and during integration of multiple libraries. It reduces the likelihood of unintentional name clashes between different parts of the code.

Whenever the user selects on the Display blocks, it's the functions with the "display" that is being used to show the results.

```
#ifndef DATACONVERTER_H
#define DATACONVERTER_H

#include "../../blockchain/SupplierBlock.h"
#include "../../blockchain/TransactionBlock.h"
#include "../../blockchain/TransporterBlock.h"
#include <string>

namespace conversion {
    class DataConverter {
    public:
        /**
         * @brief Convert given values into a SupplierBlock object
```

```
         *
         * @param version
         * @param bits
         * @param height
         * @param nonce
         * @param currentHash
         * @param previousHash
         * @param data
         * @param visible
         * @return
         */
        static blockchain::SupplierBlock convertToSupplierBlock(int version, const
std::string bits, int height, int nonce, const std::string& currentHash, const
std::string& previousHash, const std::string& data, bool visible);

        /**
         * @brief Convert given values into a TransporterBlock object
         *
         * @param version
         * @param bits
         * @param height
         * @param nonce
         * @param currentHash
         * @param previousHash
         * @param data
         * @param visible
         * @return
         */
        static blockchain::TransporterBlock convertToTransporterBlock(int version,
const std::string bits, int height, int nonce, const std::string& currentHash, const
std::string& previousHash, const std::string& data, bool visible);

        /**
         * @brief Convert given values into a TransactionBlock object
         *
         * @param version
         * @param bits
         * @param height
         * @param nonce
         * @param currentHash
         * @param previousHash
         * @param data
         * @param visible
         * @return
         */
        static blockchain::TransactionBlock convertToTransactionBlock(int version,
const std::string bits, int height, int nonce, const std::string& currentHash, const
std::string& previousHash, const std::string& data, bool visible);
    };
}

#endif // DATACONVERTER_H
```

Figure 15: DataConverter.h

```
#include "DataConverter.h"
#include <sstream>
#include <vector>
#include <map>
#include <iostream>

namespace conversion {
    /**
```

```
     * @brief Utility function to trim whitespace from start and end of a string
     * @param str
     * @return
     */
    std::string trim(const std::string& str) {
        size_t first = str.find_first_not_of(' ');
        if (first == std::string::npos) return "";
        size_t last = str.find_last_not_of(' ');
        return str.substr(first, (last - first + 1));
    }

    /**
     * @brief Splits a string into a vector of strings at each instance of a delimiter
     * Helper Method
     *
     * @param s
     * @param delimiter
     * @return
     */
    std::vector<std::string> split(const std::string& s, char delimiter) {
        std::vector<std::string> tokens;
        std::string token;
        std::istringstream tokenStream(s);
        while (getline(tokenStream, token, delimiter)) {
            tokens.push_back(trim(token));
        }
        return tokens;
    }

    // Parses a block of text into a map of key-value pairs
    std::map<std::string, std::string> parseBlockInfo(const std::string& blockInfo) {
        std::map<std::string, std::string> infoMap;
        auto lines = split(blockInfo, '\n');
        for (const auto& line : lines) {
            auto keyValue = split(line, ':');
            if (keyValue.size() == 2) {
                infoMap[trim(keyValue[0])] = trim(keyValue[1]);
            }
        }
        return infoMap;
    }

    /**
     * @brief Parse the information field of a block into a map of key-value pairs
     * Helper Method
     *
     * @param infoField
     * @return
     */
    std::map<std::string, std::string> parseInformationField(const std::string&
infoField) {
        std::map<std::string, std::string> infoMap;
        auto infoPairs = split(infoField, '|');
        for (const auto& pair : infoPairs) {
            auto keyValue = split(pair, ':');
            if (keyValue.size() == 2) {
                infoMap[trim(keyValue[0])] = trim(keyValue[1]);
            }
        }
        return infoMap;
    }

    blockchain::SupplierBlock DataConverter::convertToSupplierBlock(int version, const
```

```
std::string bits, int height, int nonce, const std::string& currentHash, const
std::string& previousHash, const std::string& data, bool visible) {
        auto infoDetails = parseInformationField(data);
        blockchain::SupplierInfo info(std::stoi(infoDetails["ID"]),
infoDetails["Name"], infoDetails["Location"], infoDetails["Branch"],
infoDetails["Items"]);

        return blockchain::SupplierBlock(version, bits, height, previousHash, info,
nonce, currentHash, visible);
    }

    blockchain::TransporterBlock DataConverter::convertToTransporterBlock(int version,
const std::string bits, int height, int nonce, const std::string& currentHash, const
std::string& previousHash, const std::string& data, bool visible) {
        auto infoDetails = parseInformationField(data);
        blockchain::TransporterInfo info(std::stoi(infoDetails["ID"]),
infoDetails["Name"], infoDetails["Product Type"], infoDetails["Transportation Type"],
infoDetails["Ordering Type"], std::stod(infoDetails["Ordering Amount (Kg)"]));

        return blockchain::TransporterBlock(version, bits, height, previousHash, info,
nonce, currentHash, visible);
    }

    blockchain::TransactionBlock DataConverter::convertToTransactionBlock(int version,
const std::string bits, int height, int nonce, const std::string& currentHash, const
std::string& previousHash, const std::string& data, bool visible) {
        auto infoDetails = parseInformationField(data);

        blockchain::TransactionInfo info(std::stoi(infoDetails["ID"]),
infoDetails["Total Fees (RM)"], infoDetails["Commission Fees (RM)"],
infoDetails["Retailer Per-Trip Credit Balance (RM)"], infoDetails["Annual Ordering
Credit Balance (RM)"], infoDetails["Payment Type"], infoDetails["Product Ordering
Limit"]);
        return blockchain::TransactionBlock(version, bits, height, previousHash, info,
nonce, currentHash, visible);
    }
}
```

Figure 16: DataConverter.cpp

The DataConverter class is vital in converting the given values to a valid respective block object from the given values. Consisting of three main methods namely the “convertToSupplierBlock”, “convertToTransporterBlock”, and “convertToTransactionBlock”. Each method carrying out a vital role of converting information given from raw data type (after validation) from the participant user’s input to a complete Block object which will be used for easier manipulation during the operations in the blockchain

The helper method “parseBlockInfo” is used to convert the given information of the block object as a string to a **map** of key value pairs. In this case, Key-value pair association is a direct support of **std::map**, moreover, once the information is stored in the map, looking up values by the keys is extremely convenient with the complexity of O(log n) due to the underlying balanced tree structure.

```cpp
#pragma once

#include "../blockchain/SupplierBlock.h"
#include "../blockchain/TransporterBlock.h"
#include "../blockchain/TransactionBlock.h"
#include "../blockchain/enums/BlockAttribute.h"
#include "../blockchain/Chain.h"

namespace collection {
    class Prompt {
    public:
        /**
         * @brief Collects username and password details from the user.
         *
         * @return
         */
        static std::pair<std::string, std::string> collectLoginDetails();

        /**
         * @brief Collects the all the header information for a Supplier variant
Block.
         *
         * @param optionsFilePath
         * @return
         */
        static blockchain::SupplierInfo collectSupplierInfo(const std::string&
optionsFilePath);

        /**
         * @brief Collects the all the header information for a Transporter variant
Block.
         *
         * @param optionsFilePath
         * @return
         */
        static blockchain::TransporterInfo collectTransporterInfo(const std::string&
optionsFilePath);

        /**
         * @brief Collects the all the header information for a Transaction variant
Block.
         *
         * @param optionsFilePath
         * @param recordsFilePath
         * @return
         */
        static blockchain::TransactionInfo collectTransactionInfo(const std::string&
optionsFilePath, const std::string& recordsFilePath);

        /**
         * @brief Collects the search criteria from the participant.
         * Prompts the participant to select a block's attribute to perform the
operation by as well as the value to search for under that attribute.
         *
         * @param searchOptions
         * @param topic
         * @return
         */
        static std::pair<blockchain::enums::BlockAttribute, std::string>
collectSearchCriteria(const std::vector<std::string>& searchOptions, const
std::string& topic = "search");

        /**
```

```
         * @brief Collects the block manipulation criteria from the participant.
         * Where dynamic blockchain occurs here.
         * Allows the participant to either Soft Edit, Hard Edit, Soft Delete, Hard
Delete, or mine any block as long as its visible.
         *
         * @param blockchain
         * @param redactedBlockchain
         * @param searchOptions
         */
        static void collectBlockManipulationCriteria(blockchain::Chain& blockchain,
blockchain::Chain& redactedBlockchain, const std::vector<std::string>& searchOptions);
    };
}
```

Figure 17: InputCollector.h

```
#include "../blockchain/SupplierBlock.h"
#include "InputCollector.h"
#include "validator/InputValidator.h"
#include "../filesystem/FileReader.h"
#include "../filesystem/FileWriter.h"
#include "../utils/Structures.h"
#include "../../data/Config.h"
#include <string>

namespace collection {
    /**
     * @brief Validate and edit a field in the options file
     * Helper function to validate and edit a field in the options file
     *
     * @param fieldName
     * @param currentValue
     * @param optionsFilePath
     * @param row
     * @param column
     * @return
     */
    static std::string validateAndEditOptionsField(const std::string& fieldName, const
std::string& currentValue, const std::string& optionsFilePath, int row, int column) {
        while (true) { // Loop indefinitely until a return statement is executed
            bool confirmed =
validation::InputValidator::validateConfirmValue(fieldName, currentValue);
            if (confirmed) {
                return currentValue; // User confirmed the value, return it and exit
the loop
            } else {
                bool editAction =
validation::InputValidator::validateConfirmValue("edit " + fieldName);
                if (editAction) {
                    std::string newValue =
validation::InputValidator::validateString(fieldName);
                    filesystem::FileWriter::modifyCell(optionsFilePath, row, column,
newValue);
                    return newValue; // User edited the value, return the new value
and exit the loop
                }
                // If editAction is false, the loop will continue, effectively
restarting the process
            }
        }
    }

    std::pair<std::string, std::string> Prompt::collectLoginDetails() {
```

```
        std::string username = validation::InputValidator::validateString("username");
        std::string password = validation::InputValidator::validateString("password");

        return std::make_pair(username, password);
    }

    blockchain::SupplierInfo Prompt::collectSupplierInfo(const std::string&
optionsFilePath) {
        std::string id, name, location, branch, items;

        filesystem::FileReader reader(optionsFilePath, filesystem::DataType::OPTION);

        auto allIds = reader.getAllInitialOptions();
        id = validation::InputValidator::validateSelection("supplier ID", allIds);
        int row = utils::Structures::findVectorIndex(allIds, id);

        // Fetch all supplier data for the selected ID
        auto selectedData = reader.getDataById(id);

        name = validateAndEditOptionsField("supplier name", selectedData[2],
optionsFilePath, row, 2);
        location = validateAndEditOptionsField("supplier location", selectedData[3],
optionsFilePath, row, 3);
        branch = validateAndEditOptionsField("supplier branch", selectedData[4],
optionsFilePath, row, 4);
        items = validation::InputValidator::validateString("supplier items");

        return blockchain::SupplierInfo(std::stoi(id), name, location, branch, items);
    }

    blockchain::TransporterInfo Prompt::collectTransporterInfo(const std::string&
optionsFilePath) {
        std::string id, name, productType, transportationType, orderingType;

        filesystem::FileReader reader(optionsFilePath, filesystem::DataType::OPTION);

        // Fetch all transporter IDs for selection
        auto allIds = reader.getAllInitialOptions();
        id = validation::InputValidator::validateSelection("transporter ID", allIds);
        int row = utils::Structures::findVectorIndex(allIds, id);

        // Fetch all transporter data for the selected ID
        auto selectedData = reader.getDataById(id);

        name = validateAndEditOptionsField("supplier name", selectedData[2],
optionsFilePath, row, 2);

        auto productTypeOptions =
filesystem::FileReader::parseBracketOptions(selectedData[3]);
        productType =
collection::validation::InputValidator::validateSelection("transporter product type",
productTypeOptions);

        auto transportationTypeOptions =
filesystem::FileReader::parseBracketOptions(selectedData[4]);
        transportationType =
collection::validation::InputValidator::validateSelection("transporter transportation
type", transportationTypeOptions);

        auto orderingTypeOptions =
filesystem::FileReader::parseBracketOptions(selectedData[5]);
        orderingType =
collection::validation::InputValidator::validateSelection("transporter ordering type",
```

```
orderingTypeOptions);

        double orderingAmount =
validation::InputValidator::validateDouble("transporter ordering payment type (kg)");

        return blockchain::TransporterInfo(std::stoi(id), name, productType,
transportationType, orderingType, orderingAmount);
    }

    blockchain::TransactionInfo Prompt::collectTransactionInfo(const std::string&
optionsFilePath, const std::string& recordsFilePath) {
        int id, productOrderingLimit;
        std::string totalFees, commisionFees, retailerPerTripCreditBalance,
annualOrderingCreditBalance, paymentType;

        filesystem::FileReader reader(optionsFilePath, filesystem::DataType::OPTION);

        auto allPaymentTypes = reader.getAllInitialOptions();

        std::vector<int> transactionIds =
filesystem::FileReader::extractBlockIds(recordsFilePath, "Transaction");

        id =
std::stoi(collection::validation::InputValidator::validateUniqueIdInt("transaction ID
(unique)", transactionIds));
        totalFees =
collection::validation::InputValidator::validateString("transaction fees (RM)");
        commisionFees =
collection::validation::InputValidator::validateString("commission fees (RM)");
        retailerPerTripCreditBalance =
collection::validation::InputValidator::validateString("retailer per trip credit
balance (RM)");
        annualOrderingCreditBalance =
collection::validation::InputValidator::validateString("annual ordering credit balance
(RM)");
        paymentType = validation::InputValidator::validateSelection("payment type",
allPaymentTypes);

        // Fetch all payment types for selection
        auto selectedData = reader.getDataById(paymentType);

        auto productOrderingLimitOptions =
filesystem::FileReader::parseBracketOptions(selectedData[2]);
        std::string productOrderingLimitType =
collection::validation::InputValidator::validateSelection("product ordering limit",
productOrderingLimitOptions);

        productOrderingLimit =
collection::validation::InputValidator::validateInt("product ordering limit (" +
productOrderingLimitType + ")");
        std::string productOrderingLimitStr = "(" + productOrderingLimitType + ") " +
std::to_string(productOrderingLimit);

        return blockchain::TransactionInfo(id, totalFees, commisionFees,
retailerPerTripCreditBalance, annualOrderingCreditBalance, paymentType,
productOrderingLimitStr);
    }

    std::pair<blockchain::enums::BlockAttribute, std::string>
Prompt::collectSearchCriteria(const std::vector<std::string>& searchOptions, const
std::string& topic) {
        int searchByAttr =
collection::validation::InputValidator::validateSelectionInt("an attribute to " +
```

```
topic + " by", searchOptions);
        std::string searchValue =
collection::validation::InputValidator::validateString("a value to " + topic + "
for");

        return
std::make_pair(static_cast<blockchain::enums::BlockAttribute>(searchByAttr - 1),
searchValue);
    }

    void Prompt::collectBlockManipulationCriteria(blockchain::Chain& blockchain,
blockchain::Chain& redactedBlockchain, const std::vector<std::string>& searchOptions)
{
        using namespace blockchain::enums;

        std::vector<std::string> types = { "Edit", "Delete", "Mine Edited Blocks" };
        int type = validation::InputValidator::validateSelectionInt("Manipulation
Type", types);

        if (type == 1) {
            // Edit block
            std::vector<std::string> editTypes = { "Soft Edit", "Hard Edit" };

            int editType = validation::InputValidator::validateSelectionInt("Edit
Type", editTypes);

            std::vector<std::shared_ptr<blockchain::Block>> foundBlocks;
            do {
                auto [searchAttr, searchValue] =
collection::Prompt::collectSearchCriteria(searchOptions, (editType == 1 ? "soft
edit" : "hard edit"));
                foundBlocks = (editType == 1 ?
redactedBlockchain.searchBlockByAttr(searchAttr, searchValue) :
blockchain.searchBlockByAttr(searchAttr, searchValue));

                if (foundBlocks.size() > 1) {
                    std::cout << foundBlocks.size() << " blocks matching the criteria
found. Manipulate one block at a time!" << std::endl << std::endl;
                }
            } while (foundBlocks.size() > 1);

            // Found only one block, proceed with editing
            std::shared_ptr<blockchain::BlockInfo> infoPtr;
            switch (foundBlocks[0]->getType()) {
                case blockchain::enums::BlockType::SUPPLIER:
                    infoPtr = std::make_shared<blockchain::SupplierInfo>(

collection::Prompt::collectSupplierInfo(data::Config::OPTIONS_SUPPLIER_FILE_PATH));
                    break;

                case blockchain::enums::BlockType::TRANSPORTER:
                    infoPtr = std::make_shared<blockchain::TransporterInfo>(

collection::Prompt::collectTransporterInfo(data::Config::OPTIONS_TRANSPORTER_FILE_PATH
));
                    break;

                case blockchain::enums::BlockType::TRANSACTION:
                    infoPtr = std::make_shared<blockchain::TransactionInfo>(

collection::Prompt::collectTransactionInfo(data::Config::OPTIONS_TRANSACTION_FILE_PATH
, data::Config::RECORDS_BLOCKCHAIN_FILE_PATH));
                    break;
```

```
                default:
                    std::cerr << "Unsupported block type" << std::endl;
                    return; // Ensure to exit the method if the block type is
unsupported
            }

            if (foundBlocks.empty()) {
                std::cout << "No blocks found matching the criteria." << std::endl;
                return;
            }

            redactedBlockchain.editBlock(foundBlocks[0], infoPtr->toString());
            if (editType == 2) {
                blockchain.hardEditBlock(foundBlocks[0], infoPtr->toString());
            }

            std::cout << "Block edited successfully." << std::endl << std::endl;
        } else if (type == 2) {
            // Delete block
            std::vector<std::string> delTypes = { "Soft Delete", "Hard Delete" };
            int delType = validation::InputValidator::validateSelectionInt("Delete
Type", delTypes);

            std::vector<std::shared_ptr<blockchain::Block>> foundBlocks;
            do {
                auto [searchAttr, searchValue] =
collection::Prompt::collectSearchCriteria(searchOptions, (delType == 1 ? "soft
delete" : "hard delete"));
                foundBlocks = (delType == 1 ?
redactedBlockchain.searchBlockByAttr(searchAttr, searchValue) :
blockchain.searchBlockByAttr(searchAttr, searchValue));

                if (foundBlocks.size() > 1) {
                    std::cout << foundBlocks.size() << " blocks matching the criteria
found. Manipulate one block at a time!" << std::endl << std::endl;
                }
            } while (foundBlocks.size() > 1);

            if (foundBlocks.empty()) {
                std::cout << "No blocks found matching the criteria." << std::endl;
                return;
            }

            // Found only one block, proceed with deletion
            redactedBlockchain.hideBlock(foundBlocks[0]);
            if (delType == 2) {
                blockchain.hardHideBlock(foundBlocks[0]);
            }

            std::cout << "Block deleted successfully." << std::endl << std::endl;
        } else {
            // Mine edited blocks
            std::vector<std::shared_ptr<blockchain::Block>> foundBlocks;
            do {
                auto [searchAttr, searchValue] =
collection::Prompt::collectSearchCriteria(searchOptions, "mine");
                foundBlocks = blockchain.searchBlockByAttr(searchAttr, searchValue);

                if (foundBlocks.size() > 1) {
                    std::cout << foundBlocks.size() << " blocks matching the criteria
found. Manipulate one block at a time!" << std::endl << std::endl;
                }
```

```
            } while (foundBlocks.size() > 1);

            if (foundBlocks.empty()) {
                std::cout << "No blocks found matching the criteria." << std::endl;
                return;
            }

            if (foundBlocks[0]->getHeader().isMined()) {
                std::cout << "Block Hash value is already mined to the correct
pattern." << std::endl << std::endl;
                return;
            }

            // Found only one block and has validity to be remined, proceed with
mining
            redactedBlockchain.mineBlock(foundBlocks[0]);
            blockchain.mineBlock(foundBlocks[0]);

            std::cout << "Block mined successfully." << std::endl << std::endl;
        }
    }
} // namespace collection
```

Figure 18: InputCollector.cpp

The above is the code for the *InputCollector.h* and *InputCollector.cpp.* This class is heavily reliant throughout the entire system as all inputs will be extracted and processed here. Including collecting information on the block's header information which will then be concatenated into a full string separated by a vertical line, as well as searching criteria for visible blocks as well as performing soft and hard operations on the blocks themselves.

Switch-Case Statements have been used as seen to properly assign the right value to the infoPtr. This is highly effective in scenarios like this which provides better readability for developers as well as better efficiency compared to a series of if-else statements when dealing with a range of values under one variable

Each of the input collection prompt consist of proper input validation to the required data type. This will ensure the validity of the input from the participant as well as avoiding inconsistent data type being passed around the codebase.

```
#pragma once

#include <iostream>
#include <string>
#include <regex>
#include <set>

namespace collection::validation {
    class InputValidator {
```

```
    public:
        /**
         * @brief Validate an integer input.
         * Checks if the input is a valid integer.
         *
         * @param topic
         * @return
         */
        static int validateInt(const std::string& topic);

        /**
         * @brief Validate a double input.
         * Checks if the input is a valid double.
         *
         * @param topic
         * @return
         */
        static double validateDouble(const std::string& topic);

        /**
         * @brief Validate a string input.
         * Checks if the input is a valid string.
         *
         * @param topic
         * @return
         */
        static std::string validateString(const std::string& topic);

        /**
         * @brief Validate a date string input.
         * Checks if the input is a valid date string.
         *
         * @param topic
         * @return
         */
        static std::string validateDateString(const std::string& topic);

        /**
         * @brief Validate a time string input.
         * Checks if the input is a valid time string.
         *
         * @param topic
         * @return
         */
        static std::string validateTimeString(const std::string& topic);

        /**
         * @brief Validate a currency input.
         *
         * @param topic
         * @return
         */
        std::string validateCurrencyInput(const std::string& topic);

        /**
         * @brief Validate a selection input.
         *
         * @param topic
         * @param options
         * @return
         */
        static std::string validateSelection(const std::string& topic, const
std::vector<std::string>& options);
```

```
        /**
         * @brief Validate a selection input.
         *
         * @param topic
         * @param options
         * @return
         */
        static int validateSelectionInt(const std::string& topic, const
std::vector<std::string>& options);

        /**
         * @brief Validate a confirm value input.
         *
         * @param topic
         * @param value
         * @return
         */
        static bool validateConfirmValue(const std::string& topic, const std::string&
value = "");

        /**
         * @brief Validate a unique id input.
         *
         * @param topic
         * @param existingValues
         * @return
         */
        static std::string validateUniqueIdInt(const std::string& topic, const
std::vector<int>& existingValues);

        /**
         * @brief Validate if the input is an exit command.
         *
         * @param input
         * @return
         */
        static bool isExitCommand(const std::string& input);

        /**
         * @brief Validate if the input is an empty input.
         *
         * @param input
         * @return
         */
        static bool isEmptyInput(const std::string& input);
    };
}
```

Figure 19: InputValidator.h

```
#include "InputValidator.h"
#include <iostream>
#include <limits>
#include <set>

namespace collection::validation {
    /**
     * @brief Get a string input from the currentParticipant and provides error
handling for empty inputs.
     *
     * @param topic
     * @return
```

```
     */
    std::string InputValidator::validateString(const std::string& topic) {
        std::string input;
        while (true) {
            std::cout << "Enter " << topic << ": ";
            std::getline(std::cin, input);

            if (isExitCommand(input) || !isEmptyInput(input)) {
                std::cout << "Entered " << topic << ": " << input << std::endl <<
std::endl;
                return input; // Return the valid, non-empty input
            }
        }
    }

    /**
     * @brief Check if the input is an integer and provides error handling for invalid
inputs other than integer values.
     *
     * @param topic
     * @return
     */
    int InputValidator::validateInt(const std::string& topic) {
        int value;
        std::string input;
        while (true) {
            std::cout << "Enter " << topic << ": ";
            std::getline(std::cin, input);

            if (isExitCommand(input) || isEmptyInput(input)) continue;

            std::stringstream inputStream(input);
            if (!(inputStream >> value) || !(inputStream.eof())) { // Checks for exact
match and no extra characters
                std::cout << "Invalid input. Please enter an integer.\n";
            } else {
                std::cout << "Entered " << topic << ": " << value << std::endl <<
std::endl;
                return value;
            }
        }
    }

    /**
     * @brief Validates currentParticipant input to ensure it is a valid double value.
     *
     * This function prompts the currentParticipant with a custom message and attempts
to read
     * a double value from the input. If the input is not a valid double, the function
     * clears the error state of cin, discards the invalid input, and prompts the
currentParticipant again.
     * This process repeats until a valid double is entered.
     *
     * @param topic The custom message displayed to the currentParticipant prompting
for input.
     * @return A valid double value entered by the currentParticipant.
     */
    double InputValidator::validateDouble(const std::string& topic) {
        double value;
        std::string input;
        while (true) {
            std::cout << "Enter " << topic << ": ";
            std::getline(std::cin, input);
```

```
            if (isExitCommand(input) || isEmptyInput(input)) continue;

            std::stringstream inputStream(input);
            if (!(inputStream >> value) || !(inputStream.eof())) { // Checks for exact
match and no extra characters
                std::cout << "Invalid input. Please enter a valid number.\n";
            } else {
                std::cout << "Entered " << topic << ": " << value << std::endl <<
std::endl;
                return value;
            }
        }
    }

    /**
     * @brief Validates a date string against the format dd/mm/yyyy.
     *
     * This function prompts the currentParticipant with a custom message and reads an
input string.
     * It then checks if the input matches the expected date format (dd/mm/yyyy) using
     * a regular expression. If the input does not match, it prompts the
currentParticipant again.
     * This process repeats until a valid date format is entered.
     *
     * @param topic The custom message displayed to the currentParticipant prompting
for input.
     * @return A valid date string in the format dd/mm/yyyy.
     */
    std::string InputValidator::validateDateString(const std::string& topic) {
        std::regex datePattern(R"(^(0[1-9]|[12][0-9]|3[01])/(0[1-9]|1[012])/\d{4}$)");
        std::string input;
        while (true) {
            std::cout << "Enter " << topic << ": ";
            std::getline(std::cin, input);

            if (isExitCommand(input) || isEmptyInput(input)) continue;

            if (!std::regex_match(input, datePattern)) {
                std::cout << "Invalid date format. Please enter in dd/mm/yyyy
format.\n";
            } else {
                return input;
            }
        }
    }

    /**
     * @brief Validates a time string against a 12-hour clock format with AM/PM.
     *
     * This function prompts the currentParticipant with a custom message and reads an
input string.
     * It checks if the input matches the expected time format (hh:mm AM/PM, with
flexible
     * spacing and case sensitivity) using a regular expression. If the input does not
match,
     * it prompts the currentParticipant again. This process repeats until a valid
time format is entered.
     *
     * @param topic The custom message displayed to the currentParticipant prompting
for input.
     * @return A valid time string in the format hh:mm AM/PM.
     */
```

```
    std::string InputValidator::validateTimeString(const std::string& topic) {
        validateString(topic);
        std::regex timePattern(R"(^(1[0-2]|0?[1-9]):([0-5]?[0-9]) ?([AaPp][Mm])$)");
        std::string input;
        while (true) {
            std::cout << "Enter " << topic << ": ";
            std::getline(std::cin, input);

            if (isExitCommand(input) || isEmptyInput(input)) continue;

            if (!std::regex_match(input, timePattern)) {
                std::cout << "Invalid time format. Please enter in hh:mm AM/PM
format.\n";
            } else {
                return input;
            }
        }
    }

    /**
     * @brief Validates a numeric productOrderingLimit string to ensure it is properly
formatted
     * with at most two decimal places.
     *
     * This function prompts the currentParticipant for a productOrderingLimit value,
ensuring that the input
     * matches a numeric format with at most two decimal places. It does not accept
     * productOrderingLimit symbols or any non-numeric characters except for the
decimal point,
     * and ensures that if a decimal point is present, it is followed by no more than
     * two digits.
     *
     * @param topic The message displayed to the currentParticipant prompting for
input.
     * @return A string representing a valid numeric productOrderingLimit value.
     */
    std::string InputValidator::validateCurrencyInput(const std::string& topic) {
        std::regex currencyPattern(R"(^[0-9]+(\.[0-9]{1,2})?$)");
        std::string input;
        do {
            std::cout << "Enter " << topic << ": ";
            std::getline(std::cin, input);

            if (isExitCommand(input) || isEmptyInput(input)) continue;

            if (!std::regex_match(input, currencyPattern)) {
                std::cout << "Invalid currency format. Please enter a numeric value
with at most two decimal places (e.g., 1234.56).\n";
            } else {
                break; // Valid currency format
            }
        } while (true);

        return input;
    }

    std::string InputValidator::validateSelection(const std::string& topic, const
std::vector<std::string>& options) {
        std::string input;
        while (true) {
            // Display options
            for (size_t i = 0; i < options.size(); ++i) {
                std::cout << (i + 1) << ". " << options[i] << std::endl;
```

```
            }

            // Get currentParticipant's choice
            std::cout << "Select " << topic << ": ";
            std::getline(std::cin, input); // Use getline to read the input as a
string

            if (isExitCommand(input)) break;

            try {
                int choice = std::stoi(input); // Convert input to integer

                if (choice < 1 || choice > static_cast<int>(options.size())) {
                    std::cout << "Invalid selection. Please try again.\n";
                    continue; // Invalid selection, show options again
                }

                std::cout << "Selected " << topic << ": " << options[choice - 1] <<
std::endl << std::endl;
                return options[choice - 1]; // Return the selected option
            } catch (std::invalid_argument& e) {
                // Non-integer input that's not an exit command
                std::cout << "Invalid input. Please enter a number or 'q' to quit.\n";
            } catch (std::out_of_range& e) {
                // Input number too large for int
                std::cout << "Selection out of range. Please try again.\n";
            }
        }

        return ""; // Return an empty string or handle this case as needed
    }

    int InputValidator::validateSelectionInt(const std::string& topic, const
std::vector<std::string>& options) {
        // Use validateSelection to get the selected option
        std::string selectedOption = validateSelection(topic, options);

        // Check if the currentParticipant has decided to exit
        if (selectedOption.empty() || isExitCommand(selectedOption)) {
            return -1; // or any other indicator that currentParticipant exited or
invalid selection was made
        }

        // Find the index of the selected option
        auto it = std::find(options.begin(), options.end(), selectedOption);
        if (it != options.end()) {
            return std::distance(options.begin(), it) + 1; // Add 1 to convert from 0-
based to 1-based index
        }

        // In case something goes wrong and the selected option is not found (which
shouldn't happen)
        return -1;
    }

    bool InputValidator::validateConfirmValue(const std::string& topic, const
std::string& value) {
        std::string input;
        // Loop until a valid response is received
        while (true) {
            std::cout << "Confirm " << topic << ": " << value << "\nConfirm? (y/n): ";
            std::getline(std::cin, input);
```

```
            if (isExitCommand(input)) break;

            // Check for valid input
            if (input == "y" || input == "Y") {
                std::cout << "Confirmed " << topic << ": " << value << std::endl <<
std::endl;
                return true;
            } else if (input == "n" || input == "N") {
                return false;
            } else {
                std::cout << "Invalid input. Please enter 'y' or 'Y' or 'n' or 'N'
only.\n";
            }
        }

        return false;
    }

    /**
     * @brief Validates currentParticipant input to ensure it is unique compared to a
set of existing values.
     *
     * This function prompts the currentParticipant with a custom message and attempts
to read a string value from the input.
     * It then checks if the input is already present in a provided set of existing
values. If the input is not unique,
     * it prompts the currentParticipant again. This process repeats until a unique
input is entered.
     *
     * @param topic The custom message displayed to the currentParticipant prompting
for input.
     * @param existingValues The set of existing values to compare against for
uniqueness.
     * @return A unique string value entered by the currentParticipant.
     */
    std::string InputValidator::validateUniqueIdInt(const std::string& topic, const
std::vector<int>& existingValues) {
        int value;
        std::string input;
        while (true) {
            std::cout << "Enter " << topic << ": ";
            std::getline(std::cin, input);

            if (isExitCommand(input)) continue;
            if (isEmptyInput(input)) continue;

            std::stringstream inputStream(input);
            if (!(inputStream >> value) || !(inputStream.eof())) {
                std::cout << "Invalid input. Please enter an integer.\n";
                continue;
            }

            if (std::find(existingValues.begin(), existingValues.end(), value) !=
existingValues.end()) {
                std::cout << "The entered " << topic << " has been taken. Please enter
a unique value.\n";
            } else {
                std::cout << "Entered " << topic << ": " << value << std::endl <<
std::endl;
                return std::to_string(value); // Convert the integer back to string
for return
            }
        }
```

```
    }

    /**
     * @brief Check if the input is an exit command.
     * If the input is 'q' or 'quit', the program will exit.
     *
     * @param input
     * @return
     */
    bool InputValidator::isExitCommand(const std::string& input) {
        if (input == "q" || input == "quit") {
            std::exit(0); // Use exit(0) to terminate the program
        }
        return false;
    }

    /**
     * @brief Check if the input is an empty input.
     * If the input is empty, the currentParticipant will be prompted to enter a non-
empty input.
     *
     * @param input
     * @return
     */
    bool InputValidator::isEmptyInput(const std::string& input) {
        if (input.empty()) {
            std::cout << "Input cannot be empty. Please try again.\n";
            return true;
        }
        return false;
    }
}
```

Figure 20: InputValidator.cpp

The above are the codes for *InputValidator.h* and *InputValidator.cpp*. As seen in the *InpuCollector.cpp* source implementation file. Majority of the prompt methods used **static methods** from this class which properly validates the provided input type. This is a class is a **specialized utility class** which assist in input data type checking as well as other operations including whether if the user has just “Entered” or input “q”/”quit”, indicating the exit of the program.

The class does not require an instance as it does not store any **field state** in which usage of any of the methods will be much more optimized as having more negative performance impact due to the increased in memory usage or longer initialization times.

The class includes methods like “validateDateString” which ensures that the input should be in the format of “dd/mm/yyyy” in order to be a validate date input. Moreover, the “validateTimeString” ensures that the input type should be in the format of “hh::mm AM/PM”.

Vectors are used in multiple methods such as “validateSelection”, “validateUniqueIdInt”, “validateSelectionInt”. In these given situation of requiring a number of values to be validated against, its beneficial to use **vectors** as it’s a more flexible and dynamic way to store a values as they can be easily adjusted without requiring changes to the code structure. Moreover, they offer access to elements through indices, allowing quick retrieval of options based on user input.

Moreover, **if-else statements** are being is the optimal choice instead of using switch-case statements due to the nature of the validations being performed. These validations often involve complex conditions that aren’t limited to simple scaler comparisons. For example, they include checking for empty inputs, validating against **regex** patterns, and determining if input matches specific formats or commands. Such validations require runtime evaluations.

Multiple **while** loops and **do-while** loops can be found during input validation as this ensures participants input always meets the specific given criteria before proceeding. Their usage forms a user-friendly input mechanism that repeatedly prompts the user until a valid input is received, instead of abruptly ending the application or throwing an error.

```
#pragma once

#include <string>
#include <ctime>
#include <random>
#include <algorithm>
#include <sstream>
#include <iomanip>
#include <memory>
#include "BlockHeader.h"
#include "enums/BlockType.h"

namespace blockchain {
    /**
     * @brief Information about a block
     * Used as a placeholder for derived structs
     */
    struct BlockInfo {
        virtual ~BlockInfo() = default;
        virtual std::string toString() const = 0; // Pure virtual function
    };

    class Block {
    public:
        /**
         * @brief Retrieve the variant type of the block.
         * Get whether if the block is a Supplier, Transporter or Transaction block.
         *
         * @return
         */
```

```
        [[nodiscard]] blockchain::enums::BlockType getType() const;

        /**
         * @brief Get the height of the block
         *
         * @return
         */
        [[nodiscard]] int getHeight() const;

        /**
         * @brief Get the hash of the previous block
         *
         * @return
         */
        [[nodiscard]] int getNonce() const;

        /**
         * @brief Get the header of the block.
         * Get by reference to enable modification of the header.
         *
         * @return
         */
        [[nodiscard]] BlockHeader& getHeader();

        /**
         * @brief Get the header of the block.
         *
         * @return
         */
        [[nodiscard]] BlockHeader getHeader() const;

        /**
         * @brief Check whether if the block is a genesis block
         *
         * @return
         */
        [[nodiscard]] bool isGenesis() const;

        /**
         * @brief Check whether if the block is visible to the participant.
         *
         * @return
         */
        [[nodiscard]] bool isVisible() const;

        /**
         * @brief Set the initial block of the blockchain
         *
         * @param genesisValue
         */
        void setGenesis(bool genesisValue);

        /**
         * @brief Set the visibility of the block
         * Usually done after an operation on the block.
         *
         * @param visibility
         */
        void setVisible(bool visibility);

        /**
         * @brief Set the information string of the block
         *
```

```
         * @param information
         */
        void setInformationString(const std::string& information);

        /**
         * @brief Creates a deep copy of the current block's instance
         *
         * @return
         */
        virtual std::shared_ptr<Block> clone() const = 0;

    protected:
        /**
         * @brief The type of the block
         */
        blockchain::enums::BlockType type;

        /**
         * @brief The height of the block
         */
        int height;

        /*
         * @brief The header of the block
         */
        blockchain::BlockHeader header;

        /**
         * @brief Whether if the block is a genesis block
         */
        bool genesis = false;

        /**
         * @brief Whether if the block is visible to the participant
         */
        bool visible = true;

        /**
         * @brief The constructor for the block
         *
         * @param version
         * @param bits
         * @param height
         * @param previousHash
         * @param information
         * @param type
         * @param nonce
         * @param currentHash
         * @param visible
         */
        Block(const int version, const std::string bits, int height, const
std::string& previousHash, const std::string& information,
blockchain::enums::BlockType type, int nonce = 0, const std::string& currentHash = "",
bool visible = true);
    };
} // namespace blockchain
```

Figure 21: Block.h

```
#include <random>
#include "Block.h"

namespace blockchain {
```

```
    Block::Block(const int version, const std::string bits, int height, const
std::string& previousHash, const std::string& information,
blockchain::enums::BlockType type, int nonce, const std::string& currentHash, bool
visible)
            : height(height), type(type), header(type, version, bits, information,
nonce, currentHash, previousHash), visible(visible) {
    }

    // Getter methods
    blockchain::enums::BlockType Block::getType() const { return type; }
    int Block::getHeight() const { return height; }
    int Block::getNonce() const { return header.getNonce(); }
    BlockHeader& Block::getHeader() { return header; }
    BlockHeader Block::getHeader() const { return header; }
    bool Block::isGenesis() const { return genesis; }
    bool Block::isVisible() const { return visible; }

    // Setter methods
    void Block::setGenesis(bool genesisValue) { genesis = genesisValue; }
    void Block::setVisible(bool visibility) { visible = visibility; }
    void Block::setInformationString(const std::string& information)
{ header.setInformationString(information); }
} // namespace blockchain
```

Figure 22: Block.cpp

Another major contributing factor to the system is the *Block.h* and *Block.cpp*. This is the **base parent class** which has a **protected** constructor, disabling instantiation publicly, and only allowed to be constructed within **derived child** classes. The child classes are Supplier Block, Transporter Block, and Transportation Block. This makes the Block class **polymorphic.** Furthermore, this class uses both the **public** and **protected** access specifiers assigned based on the accessibility of each specific method.

Struct is used for the BlockInfo serves as the base for holding additional information specific to each type of the block and will be further extended by child **Structs** that are also a type of BlockInfo. Moreover, the keyword **virtual** is seen to be used in the BlockInfo as this enables dynamic binding in c++ whereby when a member function is declared as virtual, it enables dynamic binding whereby the appropriate version of the function to be called is determined at runtime base on the actual type of the object, rather than the static type.

Some of the methods are seen to have the prefix of **[[nodiscard]]** attribute. This is crucial to the block's implementation as it is used to indicate that the return value of the function should not be ignored by the caller. In other words, it serves as a hint to the compiler to generate a warning if the return value is not used in the calling code.

```cpp
#pragma once

#include "Block.h"
#include <string>

namespace blockchain {
    struct SupplierInfo : public BlockInfo {
        int supplierId; /** The ID of the supplier */
        std::string supplierName; /** The name of the supplier */
        std::string supplierLocation; /** The absolute location of the supplier */
        std::string supplierBranch; /** The branch of the supplier */
        std::string items; /** The items that the supplier provides in this particular
block transaction */

        SupplierInfo() = default;
        SupplierInfo(const int& id, const std::string& name, const std::string&
location, const std::string& branch, const std::string& items)
                : supplierId(id), supplierName(name), supplierLocation(location),
supplierBranch(branch), items(items) {}

        [[nodiscard]] std::string toString() const override;
    };

    class SupplierBlock : public Block {
    public:
        /**
         * @brief Construct a new Supplier Block object
         *
         * @param height
         * @param previousHash
         * @param info
         * @param nonce
         * @param currenHash
         */
        SupplierBlock(const int version, const std::string bits, int height, const
std::string& previousHash, const SupplierInfo& info, int nonce = 0, const std::string&
currenHash = "", bool visible = true);

        /**
         * @brief Get the supplier information.
         *
         * @return SupplierInfo
         */
        SupplierInfo getInfo() const;

        [[nodiscard]] std::shared_ptr<Block> clone() const override {
            return std::make_shared<SupplierBlock>(*this);
        }

    private:
        /**
         * This field holds the raw supplier information.
         */
        SupplierInfo info;
    };
}
```

Figure 23: Supplier.h

```cpp
#include "SupplierBlock.h"
#include <sstream>

namespace blockchain {
```

```
    SupplierBlock::SupplierBlock(const int version, const std::string bits, int
height, const std::string& previousHash, const SupplierInfo& info, int nonce, const
std::string& currentHash, bool visible)
            : Block(version, bits, height, previousHash, info.toString(),
blockchain::enums::BlockType::SUPPLIER, nonce, currentHash, visible), info(info) {
        // The constructor initializes the Block part with formatted supplier
information
    }

    std::string SupplierInfo::toString() const {
        std::ostringstream oss;
        oss << "ID: " << supplierId
            << " | Name: " << supplierName
            << " | Location: " << supplierLocation
            << " | Branch: " << supplierBranch
            << " | Items: " << items;
        return oss.str();
    }

    SupplierInfo SupplierBlock::getInfo() const {
        return info;
    }
}
```

Figure 21: Supplier.cpp

As seen above, the class SupplierBlock in files *SupplierBlock.h* and *SupplierBlock.cpp* is inherits from the Block **superclass** as well as the SupplierInfo inheriting from the BlockInfo parent Struct. This allows the polymorphic usage of the Block and BlockInfo data type to hint on.

The SupplierInfo struct also uses the **override** attribute where the “toString” method from the parent BlockInfo is **overridden** in the SupplierInfo implementation code. There are some methods which are kept to be publicly accessible under the **public** access specifier like the instantiation of the SupplierBlock and method “getInfo”, while some are under the **private** access specifier such as **encapsulating** the info field attribute to prevent unwanted tampering with its value.

All three variant blocks of classes SupplierBlock, TransporterBlock, TransactionBlock, SupplierInfo, TransporterInfo, and TransactionInfo are all extending from the parent class Block and struct BlockInfo respectively.

```
// Found only one block, proceed with editing
std::shared_ptr<blockchain::BlockInfo> infoPtr;
switch (foundBlocks[0]->getType()) {
    case blockchain::enums::BlockType::SUPPLIER:
        infoPtr = std::make_shared<blockchain::SupplierInfo>(

collection::Prompt::collectSupplierInfo(data::Config::OPTIONS_SUPPLIER_FILE_PATH));
        break;

    case blockchain::enums::BlockType::TRANSPORTER:
        infoPtr = std::make_shared<blockchain::TransporterInfo>(
```

```
collection::Prompt::collectTransporterInfo(data::Config::OPTIONS_TRANSPORTER_FILE_PATH
));
        break;

    case blockchain::enums::BlockType::TRANSACTION:
        infoPtr = std::make_shared<blockchain::TransactionInfo>(

collection::Prompt::collectTransactionInfo(data::Config::OPTIONS_TRANSACTION_FILE_PATH
, data::Config::RECORDS_BLOCKCHAIN_FILE_PATH));
        break;

    default:
        std::cerr << "Unsupported block type" << std::endl;
        return; // Ensure to exit the method if the block type is unsupported
}

if (foundBlocks.empty()) {
    std::cout << "No blocks found matching the criteria." << std::endl;
    return;
}

redactedBlockchain.editBlock(foundBlocks[0], infoPtr->toString());
if (editType == 2) {
    blockchain.hardEditBlock(foundBlocks[0], infoPtr->toString());
}
```

Figure 22: Part of InputCollector.cpp

As seen from the example taken from the *InputCollector.cpp* (above). The BlockInfo is used to typehint the infoPtr which is a **pointer**, enabling the dynamic value assignment to it whereby it can be easily used as an argument in the blockchain and redactedBlockchain variables.

```
#pragma once

#include <string>
#include <ctime>
#include <cstdint>
#include <vector>
#include <functional>
#include "enums/BlockType.h"

namespace blockchain {
    class BlockHeader {
    public:
        /**
         * @brief Construct a new Block Header object
         *
         * @param type
         * @param version
         * @param bits
         * @param informationString
         * @param nonce
         * @param hash
         * @param previousHash
         */
        BlockHeader(blockchain::enums::BlockType type, const int version, const
std::string bits, const std::string& informationString, int nonce = 0, const
std::string& hash = "", const std::string& previousHash = "");
```

```
        /**
         * @brief Get the hash function object
         *
         * @param type
         * @return
         */
        static std::function<std::string(std::string)>
getHashFunction(blockchain::enums::BlockType type);

        /**
         * @brief Mine the block
         *
         * @param hashFunction
         * @return
         */
        std::string mine(std::function<std::string(std::string)> hashFunction);

        /**
         * @brief Update the editable data
         *
         * @param informationString
         * @param prevHash
         * @return
         */
        BlockHeader& updateEditableData(const std::string informationString, const
std::string& prevHash = "");

        // setters
        void setHash(const std::string& hash);
        void setPrevHash(const std::string& prevHash);
        void setMerkleRoot(const std::string& merkleRoot);
        void setTimestamp(time_t timestamp);
        void setFormattedTimestamp(const std::string& formattedTimestamp);
        void setInformationString(const std::string& informationString);
        void setNonce(int nonce);
        void setMined(bool mined);

        // getters
        [[nodiscard]] blockchain::enums::BlockType getType() const;
        [[nodiscard]] std::string getHash() const;
        [[nodiscard]] std::string getPrevHash() const;
        [[nodiscard]] std::string getMerkleRoot() const;
        [[nodiscard]] time_t getTimestamp() const;
        [[nodiscard]] std::string getFormattedTimestamp() const;
        [[nodiscard]] std::string getInformationString() const;
        [[nodiscard]] int getNonce() const;
        [[nodiscard]] bool isMined() const;

    protected:
        blockchain::enums::BlockType type; /** The type of the block */
        int version; /** The version of the block */
        std::string bits; /** The bits which is used for mining */
        std::string hash; /** The hash of the block */
        std::string previousHash; /** The previous hash of the block */
        std::string merkleRoot; /** The merkle root which contains the information of
the block */
        time_t timestamp; /** The timestamp of the block */
        std::string formattedTimestamp; /** The formatted timestamp into human-
readable datetime of the block */
        std::string informationString; /** The information string of the block */
        int nonce; /** The nonce of the block */
        bool mined = false; /** Whether if the block is mined */
```

```
        /**
         * @brief Generate a new hash based on the hash function
         *
         * @param hashFunction Either uses the SHA256 or the SHA384, or SHA512 hash
function
         * @return
         */
        std::basic_string<char> generateHash(const
std::function<std::string(std::string)> &hashFunction) const;
    };
} // namespace blockchain
```

Figure 23: BlockHeader.h

```
#include "BlockHeader.h"
#include "../utils/Datetime.h"
#include "../../libs/sha256/sha256.h"
#include "../../libs/sha384/sha384.h"
#include "../../libs/sha512/sha512.h"
#include <random>
#include <algorithm>
#include <iostream>

namespace blockchain {
    std::vector<uint8_t> hexStringToBytes(const std::string& hex) {
        std::vector<uint8_t> bytes;

        for (size_t i = 0; i < hex.length(); i += 2) {
            std::string byteString = hex.substr(i, 2);
            uint8_t byte = static_cast<uint8_t>(strtol(byteString.c_str(), nullptr,
16));
            bytes.push_back(byte);
        }

        return bytes;
    }

    // Helper methods to append data to the block header vector for hashing
    void appendIntToVector(std::vector<uint8_t>& vec, uint32_t value) {
        for (int i = 0; i < 4; ++i) {
            vec.push_back((value >> (i * 8)) & 0xFF);
        }
    }

    void appendHexToVector(std::vector<uint8_t>& vec, const std::string& hex) {
        // Ensure the hex string's length is even
        if (hex.length() % 2 != 0) {
            throw std::runtime_error("Invalid hexadecimal string.");
        }

        // Convert each pair of hexadecimal characters to a byte and append to the
vector
        for (size_t i = 0; i < hex.length(); i += 2) {
            std::string byteString = hex.substr(i, 2);
            char byte = static_cast<char>(std::stoul(byteString, nullptr, 16));
            vec.push_back(static_cast<uint8_t>(byte));
        }
    }

    std::function<std::string(std::string)>
BlockHeader::getHashFunction(blockchain::enums::BlockType type) {
        switch (type) {
            case blockchain::enums::BlockType::SUPPLIER:
```

```
            default:
                return sha256;
            case blockchain::enums::BlockType::TRANSPORTER:
                return sha384;
            case blockchain::enums::BlockType::TRANSACTION:
                return sha512;
        }
    }

    BlockHeader::BlockHeader(blockchain::enums::BlockType type, const int version,
const std::string bits, const std::string& informationString, int nonce, const
std::string& currentHash, const std::string& previousHash)
            : type(type), version(version), bits(bits),
informationString(informationString) {
        // Initialize timestamp with the current date and time
        setTimestamp(std::time(nullptr)); // Current time

        // For a genesis block, set the previous block hash to initially 64 zeros
        setPrevHash(previousHash.empty() || previousHash == "0" ? std::string(64,
'0') : previousHash);
        setMerkleRoot(getHashFunction(type)(informationString));

        if (currentHash.empty()) {
            setHash(mine(getHashFunction(type)));
        } else {
            setMined(true); // Block is already mined (from file data
            setHash(currentHash);
            setNonce(nonce);
        }

        // Set the previous hash to the mined hash for genesis block.
        if (previousHash.empty() || previousHash == "0") {
            setPrevHash(this->hash);
        }
    }

    std::string BlockHeader::mine(std::function<std::string(std::string)>
hashFunction) {
        // Target defined for a hash to start with "0000", so first 2 bytes should be
zero
        std::vector<uint8_t> targetPrefix = {0x00, 0x00};

        // The current hash is in hexadecimal
        std::string currentHashHex;

        // Begin mining process
        this->nonce = 0;
        bool hashFound = false;

        do {
            currentHashHex = generateHash(hashFunction);

            // Convert the current hash back to bytes for comparison
            std::vector<uint8_t> currentHashBytes = hexStringToBytes(currentHashHex);

            // Check if the first two bytes of the hash are zeros (i.e., check for
"0000" prefix)
            hashFound = std::equal(targetPrefix.begin(), targetPrefix.end(),
currentHashBytes.begin());

            // If a valid hash is found or if we've reached the maximum nonce value,
we stop
            if (hashFound || nonce == std::numeric_limits<int>::max()) {
```

```
                break;
            }

            // Increment the nonce for the next attempt
            ++this->nonce;
            std::cout << "Mining... Nonce: " << this->nonce << ", Hash: " <<
currentHashHex << "\r" << std::flush;
        } while (!hashFound);

        if (hashFound) {
            std::cout << std::endl << std::endl << "Block mined! Nonce: " <<
this->nonce << ", Hash: " << currentHashHex << std::endl << std::endl;
            setMined(true);
            return currentHashHex;
        } else {
            std::cout << std::endl << std::endl << "Mining ended, nonce limit
reached." << std::endl << std::endl;
            return "";
        }
    }

    std::basic_string<char> BlockHeader::generateHash(const
std::function<std::string(std::string)> &hashFunction) const {
        // Construct the block header as a byte array for hashing
        std::vector<uint8_t> blockHeader;

        appendIntToVector(blockHeader, version);
        appendHexToVector(blockHeader, previousHash);
        appendHexToVector(blockHeader, merkleRoot); // in hex value
        // Timestamp needs to be converted to bytes and appended
        appendIntToVector(blockHeader, static_cast<uint32_t>(timestamp));
        appendHexToVector(blockHeader, bits);
        appendIntToVector(blockHeader, nonce);

        // Convert blockHeader to a string for SHA256 hashing
        std::string blockHeaderStr(blockHeader.begin(), blockHeader.end());

        // Hash the block header using the provided hash function
        return hashFunction(blockHeaderStr);
    }

    BlockHeader& BlockHeader::updateEditableData(const std::string informationString,
const std::string& prevHash) {
        setInformationString(informationString);
        setMerkleRoot(getHashFunction(type)(informationString));
        setPrevHash(prevHash.empty() || prevHash == "0" ? std::string(64, '0') :
prevHash);
        setHash(generateHash(getHashFunction(type)));
        setMined(false); // Reset mined status after updating data, currentParticipant
has the choice to mine again
        setPrevHash(prevHash.empty() || prevHash == "0" ? hash : prevHash);

        return *this;
    }

    // Getter methods
    blockchain::enums::BlockType BlockHeader::getType() const { return type; }
    std::string BlockHeader::getHash() const { return hash; }
    std::string BlockHeader::getPrevHash() const { return previousHash; }
    std::string BlockHeader::getMerkleRoot() const { return merkleRoot; }
    time_t BlockHeader::getTimestamp() const { return timestamp; }
    std::string BlockHeader::getFormattedTimestamp() const { return
formattedTimestamp; }
```

```
    std::string BlockHeader::getInformationString() const { return
informationString; }
    int BlockHeader::getNonce() const { return nonce; }
    bool BlockHeader::isMined() const { return mined; }

    // Setter methods
    void BlockHeader::setHash(const std::string& hash) { this->hash = hash; }
    void BlockHeader::setPrevHash(const std::string& prevHash) { this->previousHash =
prevHash; }
    void BlockHeader::setMerkleRoot(const std::string& merkleRoot) { this->merkleRoot
= merkleRoot; }
    void BlockHeader::setTimestamp(time_t timestamp) {
        this->timestamp = timestamp;
        this->formattedTimestamp = utils::Datetime::formatTimestamp(timestamp);
    }
    void BlockHeader::setFormattedTimestamp(const std::string& formattedTimestamp)
{ this->formattedTimestamp = formattedTimestamp; }
    void BlockHeader::setInformationString(const std::string& informationString)
{ this->informationString = informationString; }
    void BlockHeader::setNonce(int nonce) { this->nonce = nonce; }
    void BlockHeader::setMined(bool mined) { this->mined = mined; }
} // namespace blockchain
```

Figure 24: BlockHeader.cpp

The above are a part of each Block: *BlockHeader.h* and *BlockHeader.cpp*. The header of the block contains the metadata of the block which is the core purpose of each block generated throughout the chain.

Each header contains important information like the nonce value, hashes, merkle root, and timestamp. The core functionality of the class is the mining functionality. The previous hash will be automatically set to the previous hash (own hash If genesis block). Whereas its own current hash will be continuously mined until it matches the pattern or quadruple zeros (0000) as the target prefix (further explain in Extraordinary features). It mainly uses the “mine” and “generateHash” methods to generate a new hash based on version, previousHash, merkleRoot, timestamp converted to bytes using **static casting**, difficulty bits and the current nonce value.

The Nonce value increments each time a new hash is found and stops incrementing until the block has been mined to the pattern.

```
#include "BlockType.h"
#include <stdexcept>

namespace blockchain::enums {
    /**
     * @brief Convert string to BlockType
     *
     * @param typeString
     * @return
```

```
     */
    BlockType BlockTypeUtils::fromString(const std::string& typeString) {
        // Convert the string to the corresponding BlockType
        // (If else statements are used instead of switch-case because the enum class
values are not continuous)
        if (typeString == "Supplier") {
            return BlockType::SUPPLIER;
        } else if (typeString == "Transporter") {
            return BlockType::TRANSPORTER;
        } else if (typeString == "Transaction") {
            return BlockType::TRANSACTION;
        } else {
            throw std::invalid_argument("Unknown BlockType string: " + typeString);
        }
    }

    /**
     * @brief Convert BlockType to string
     *
     * @param type
     * @return
     */
    std::string BlockTypeUtils::toString(BlockType type) {
        switch (type) {
            case BlockType::SUPPLIER:
                return "Supplier";
            case BlockType::TRANSPORTER:
                return "Transporter";
            case BlockType::TRANSACTION:
                return "Transaction";
            default:
                throw std::invalid_argument("Unknown BlockType");
        }
    }
}
```

Figure 25: BlockType.h

Multiple **Enums** were used throughout this project including as found in *BlockAttribute.h, HashAlgorithm.h, and BlockType.h.* Specialized utility classes were also provided to these enum classes such as the methods “fromString” and “toString”. These methods properly converts the enum cases from and to string whenever necessary.

The “fromString” method is rather using if-else statements as it deals with converting string to enum values which C++ switch-case statements do not support, because they are not primitive types. Rather they are complex, non-constant expressions.

9.0 Case Scenarios

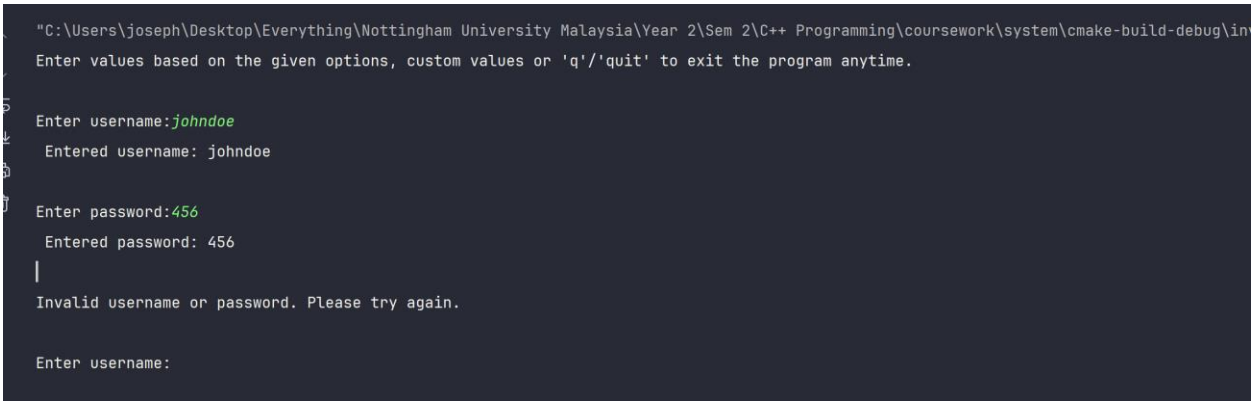

Figure 26: Unsuccessful User Participant Output (refer participants data from Figure 8)

```
Enter username:johndoe
 Entered username: johndoe

Enter password:123
 Entered password: 123

Welcome back, John Doe!

1. Display blockchain
2. Search Block
3. Add Block
4. Manipulate Block
Select an action:
```

Figure 27: Successful User Participant Authentication Output

```
1. Display blockchain
2. Search Block
3. Add Block
4. Manipulate Block
Select an action:1
 Selected an action: Display blockchain

No blocks found in the blockchain.

1. Display blockchain
2. Search Block
3. Add Block
4. Manipulate Block
Select an action:
```

Figure 27: Display Blockchain Output (during a Fresh New Blockchain Record)

```
1. Display blockchain
2. Search Block
3. Add Block
4. Manipulate Block
Select an action:2
 Selected an action: Search Block

1. Block Type
2. Height
3. Version
4. Nonce
5. Current Hash
6. Previous Hash
7. Merkle Root
8. Timestamp
9. Bits
10. Information
Select an attribute to search by:1
 Selected an attribute to search by: Block Type

Enter a value to search for:Supplier
 Entered a value to search for: Supplier

No blocks found matching the criteria.

1. Display blockchain
2. Search Block
3. Add Block
4. Manipulate Block
Select an action:
```

Figure 28: Search Block (during a Fresh New Blockchain Record)

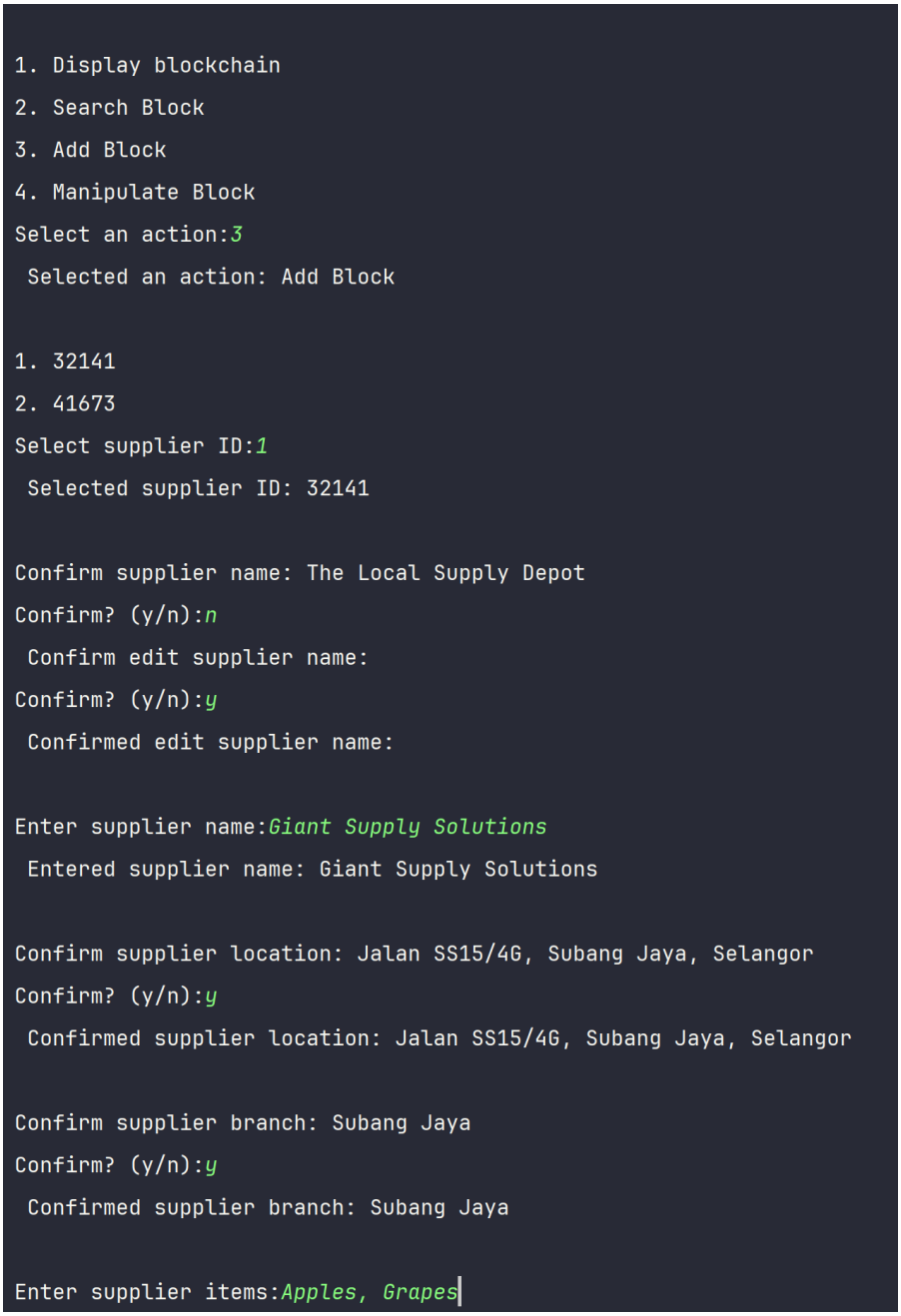

Figure 29: Adding a New Block (Supplier Block) & Editing a Supplier Options Value.

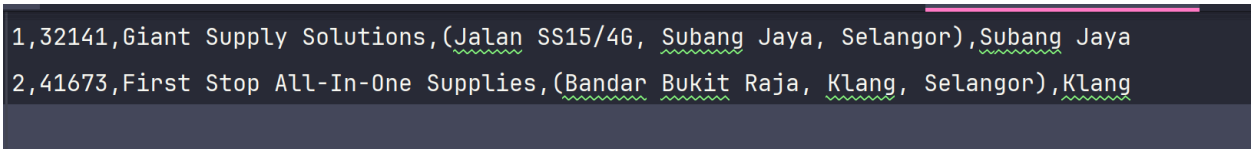

Figure 30: New Suppliers Options Value – Automatic Text file update (Original at Figure 5)

```
Mining... Nonce: 17326, Hash: 371ff0e623bb848b9b9512d1e486988d62a87f867c65e238943cf9a50ff0d88e
Mining... Nonce: 17413, Hash: 5d458c7acaa0fdbad55201af5c2e5433ec99d8580766748d6332b4ae6e35b2cf
Mining... Nonce: 17513, Hash: dd304c0e4d12c319ab7fd54c8425d4f03ccf4f5744aee8817aae4f8756e0df85
Mining... Nonce: 17584, Hash: d6ae8e89bfd3e9a1dd7b20631a41931cc30fc7f3a6cb3b3ef29922dce16237ff
Mining... Nonce: 17667, Hash: d394c42d9fb03c4ae99db57043dc22db212ed656cc9924b361b70d9540f86eb4
Mining... Nonce: 17752, Hash: b05664c8c9843d87ee4ea4695f5533bc299c853154be5d7dfb3254589c2d737a
Mining... Nonce: 17839, Hash: f905dd97efd10f76b72dac46efc239fd11bd43a92b5046dae5b481a92b7bf53e
Mining... Nonce: 17939, Hash: 13e22d1ce40ba31bf81722d97bca8e1c3d99cb0c82b3db175d43fdeb3f971ef7
Mining... Nonce: 18070, Hash: 21838870e071576450a29d74347e3fd8ead44e8fb7a33eebae9a168471a78741
Mining... Nonce: 18167, Hash: e7060aa31b5e5fa17665f5fcdf3418d572b6ec6fed742094a9037d8a3558c89f
Mining... Nonce: 18291, Hash: cee8338627bac946e064d8364a1d7b159b9d84bd6262d26922ce9429f582d4b1
Mining... Nonce: 18387, Hash: 1eee63b4d2f697b3db0b5fdbfe20ec1e9ba15569395d3caf4bb0bd5b3f79c141
Mining... Nonce: 18484, Hash: 5b0d614cdba2e2a6f1ab55aa6a9afd4001c4f753bebd01d11c4d1420a5624a8b
Mining... Nonce: 18595, Hash: 405d6a712e299f4c18606977793becd3f09f4b0e8e3d5d0c67676ffb79b59b0b
Mining... Nonce: 18700, Hash: f9ae448b4be5c8ea99c23ca2378ae5775de044ea0c976d59e7e079718bb90a74
Mining... Nonce: 18801, Hash: 79434771907eab7712b081245fb44178a35b684dc41ae87ec545556f6f952592
Mining... Nonce: 18858, Hash: 74da8ba8e74849e77b3113a48e756861805c3ae836f7dfd1261b37c64e416ca9
Mining... Nonce: 18919, Hash: 33fa4883e5a322439914a09ec5c6f440e7b92ecac67296af4f86e1d36c5d1f76
Mining... Nonce: 18969, Hash: c4a0551af9d30608b38bccc1432c831632b611057ce20891b65fa0ce44f3b146
Mining... Nonce: 19027, Hash: e595d2dc8116d06e0e219d8dcfb9b7af5c7cb04e6748f95a19923f9371ae8bed
Mining... Nonce: 19140, Hash: 2403729a5847aa3d10e506ef682f01475a0d95433f907fd25bdc1a08fe7a7772
Mining... Nonce: 19258, Hash: 629e76fc40ad48ed9d27966ec27458b6eb6c98d13807b0ec2dc06cf81a78c932
Mining... Nonce: 19295, Hash: 80ab9646d2e8d41528dad8aca7928a43c23512cd894d47ee5eb69ee67d18e324

Block mined! Nonce: 19295, Hash: 0000b3cbcbda98e5a180b8066c4fc53cbeb2ccbe56d397c6bf175993a2fe0bf0

1. Display blockchain
2. Search Block
3. Add Block
4. Manipulate Block
Select an action:
```

Figure 31: Block Mining until the Right Pattern Found

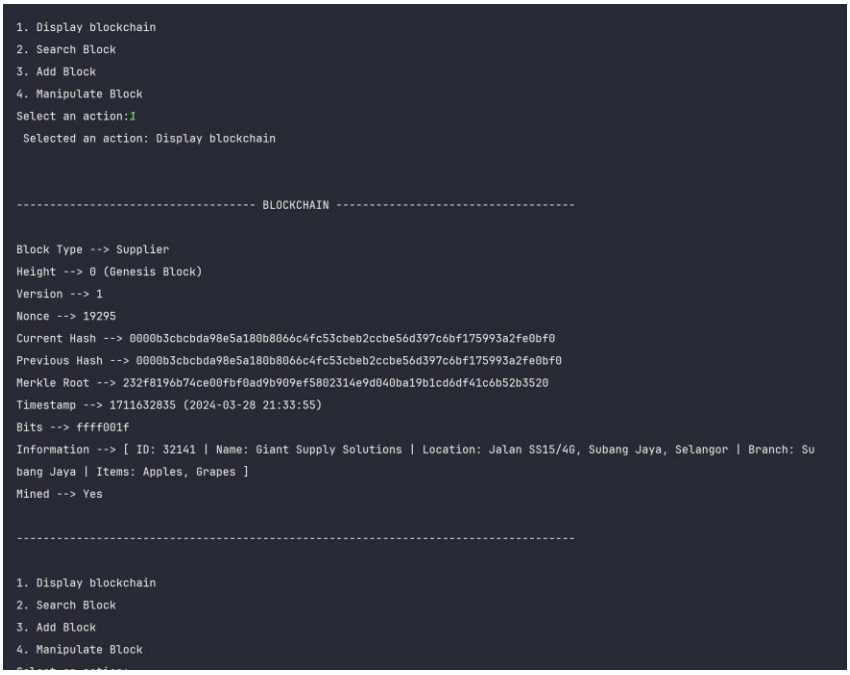

Figure 32: Display Blockchain after Adding a Block

```
1. Display blockchain
2. Search Block
3. Add Block
4. Manipulate Block
Select an action:4
 Selected an action: Manipulate Block

1. Edit
2. Delete
3. Mine Edited Blocks
Select Manipulation Type:1
 Selected Manipulation Type: Edit

1. Soft Edit
2. Hard Edit
Select Edit Type:q

Process finished with exit code 0
```

Figure 33: Quit Validation at Any Given Time

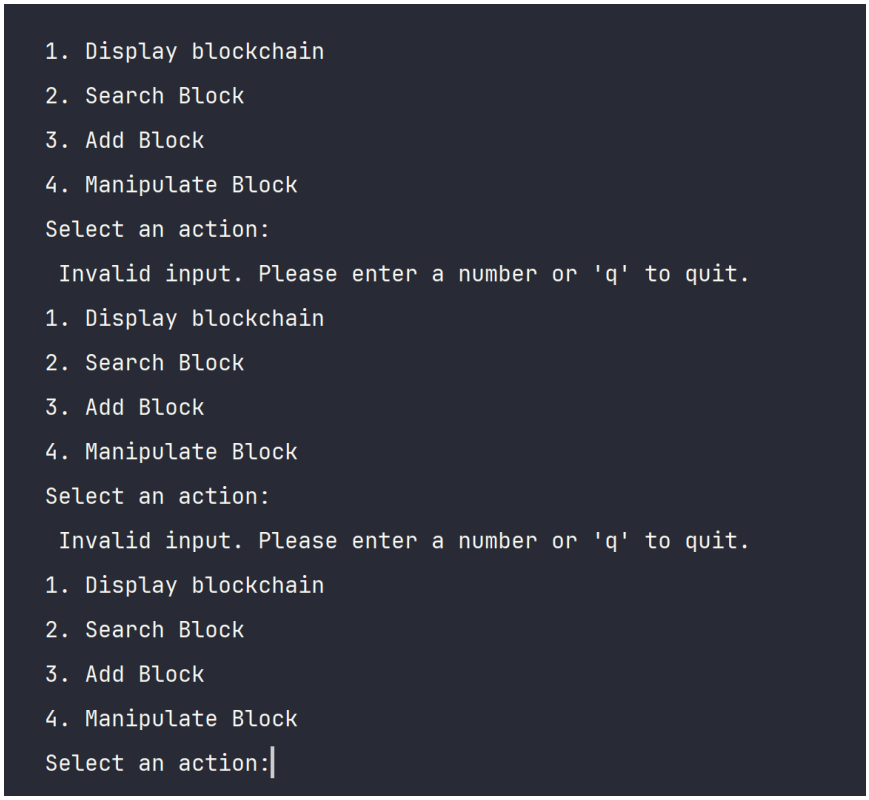

Figure 34: Empty Input Validation at Any Given Time

```
1. Display blockchain
2. Search Block
3. Add Block
4. Manipulate Block
Select an action:4
 Selected an action: Manipulate Block

1. Edit
2. Delete
3. Mine Edited Blocks
Select Manipulation Type:1
 Selected Manipulation Type: Edit

1. Soft Edit
2. Hard Edit
Select Edit Type:1
 Selected Edit Type: Soft Edit

1. Block Type
2. Height
3. Version
4. Nonce
5. Current Hash
6. Previous Hash
7. Merkle Root
8. Timestamp
9. Bits
10. Information
Select an attribute to soft edit by:2
```

Figure 35: Soft Edit (Part 1)

```
Enter a value to soft edit for:0
 Entered a value to soft edit for: 0

1. 32141
2. 41673
Select supplier ID:2
 Selected supplier ID: 41673

Confirm supplier name: First Stop All-In-One Supplies
Confirm? (y/n):y
 Confirmed supplier name: First Stop All-In-One Supplies

Confirm supplier location: Bandar Bukit Raja, Klang, Selangor
Confirm? (y/n):y
 Confirmed supplier location: Bandar Bukit Raja, Klang, Selangor

Confirm supplier branch: Klang
Confirm? (y/n):
 Invalid input. Please enter 'y' or 'Y' or 'n' or 'N' only.
Confirm supplier branch: Klang
Confirm? (y/n):y
 Confirmed supplier branch: Klang

Enter supplier items:Mango, Kiwi
 Entered supplier items: Mango, Kiwi

Block edited successfully.
```

Figure 36: Soft Edit (Part 2)

```
1. Display blockchain
2. Search Block
3. Add Block
4. Manipulate Block
Select an action:1
 Selected an action: Display blockchain


---------------------------------- BLOCKCHAIN ----------------------------------

Block Type --> Supplier
Height --> 0 (Genesis Block)
Version --> 1
Nonce --> 19295
Current Hash --> f2e5c0a55eca8643f549499ee93ad29d1bb635a92c09ed743358279dc9d4398f
Previous Hash --> f2e5c0a55eca8643f549499ee93ad29d1bb635a92c09ed743358279dc9d4398f
Merkle Root --> b3cc3a72d97b911cefcfd86c4d44b9d0823e752920f9dea8e464e11f16b9da68
Timestamp --> 1711632970 (2024-03-28 21:36:10)
Bits --> ffff001f
Information --> [ ID: 41673 | Name: First Stop All-In-One Supplies | Location: Bandar Bukit Raja, Klang, Selangor | Bran
ch: Klang | Items: Mango, Kiwi ]
Mined --> No

--------------------------------------------------------------------------------

1. Display blockchain
2. Search Block
3. Add Block
4. Manipulate Block
```

Figure 37: Display Blockchain after Soft Edit (Value Changed)

```
1. Display blockchain
2. Search Block
3. Add Block
4. Manipulate Block
Select an action:2
 Selected an action: Search Block

1. Block Type
2. Height
3. Version
4. Nonce
5. Current Hash
6. Previous Hash
7. Merkle Root
8. Timestamp
9. Bits
10. Information
Select an attribute to search by:2
 Selected an attribute to search by: Height

Enter a value to search for:0
 Entered a value to search for: 0
```

Figure 38: Search Block after Soft Edit (Part 1)

Figure 38: Search Block after Soft Edit (Value Remained the same because of Soft Edit) (Part 2)

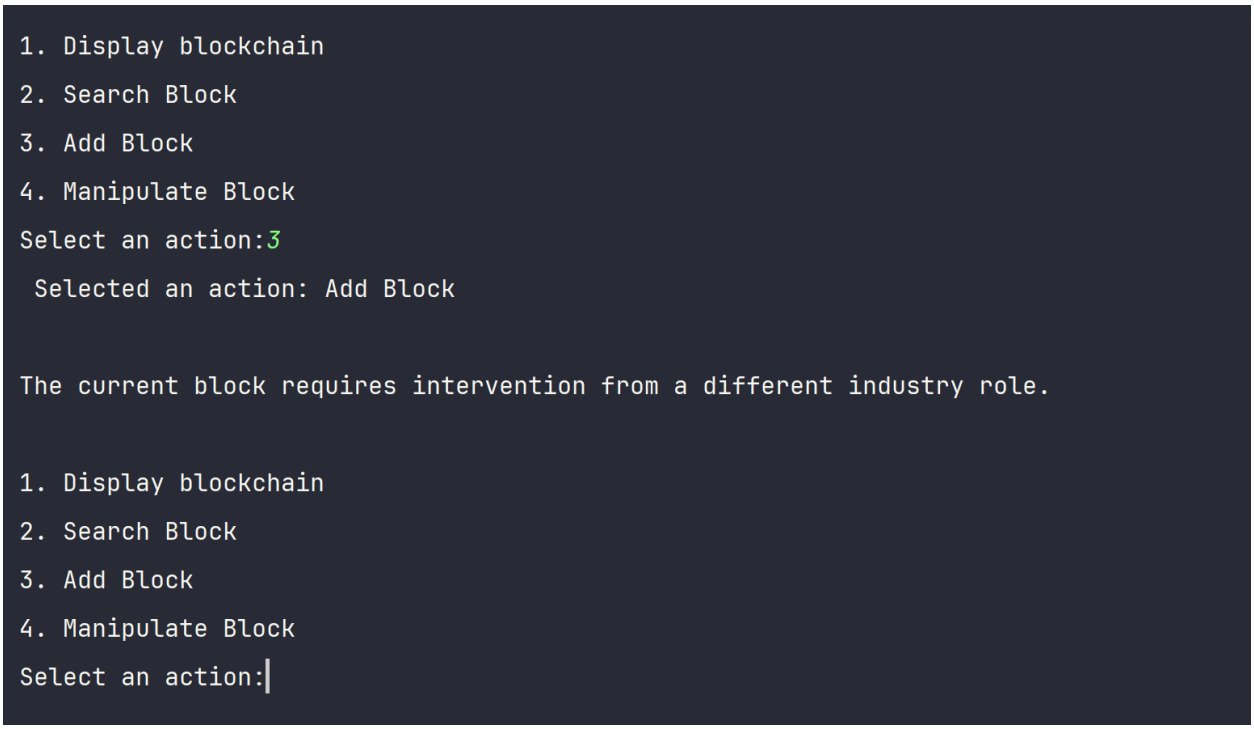

Figure 39: Rejected Access Permission Adhering to the One-to-One design architecture.

Figure 40: Logged Out and Logged In as Authorized User for Block Creation. (Creation enabled)

```
---------------------------------- BLOCKCHAIN ----------------------------------

Block Type --> Supplier
Height --> 0 (Genesis Block)
Version --> 1
Nonce --> 19295
Current Hash --> 0000b3cbcbda98e5a180b8066c4fc53cbeb2ccbe56d397c6bf175993a2fe0bf0
Previous Hash --> 0000b3cbcbda98e5a180b8066c4fc53cbeb2ccbe56d397c6bf175993a2fe0bf0
Merkle Root --> 232f8196b74ce00fbf0ad9b909ef5802314e9d040ba19b1cd6df41c6b52b3520
Timestamp --> 1711633331 (2024-03-28 21:42:11)
Bits --> ffff001f
Information --> [ ID: 32141 | Name: Giant Supply Solutions | Location: Jalan SS15/4G, Subang Jaya, Selangor | Branch: Su
bang Jaya | Items: Apples, Grapes ]
Mined --> Yes

Block Type --> Transporter
Height --> 1
Version --> 1
Nonce --> 630
Current Hash --> 00001676d12a3fc35614b7352c83a44c7ac6d592e528b14851d9b5db84bd2ffde04b62ea15548a773e84f27fd3491236
Previous Hash --> 0000b3cbcbda98e5a180b8066c4fc53cbeb2ccbe56d397c6bf175993a2fe0bf0
Merkle Root --> 47f67207293a0e4e3890bf26a0199421d5663e488d94d6c8ac1ef79630ac7012acf87389ff3493d2efdafe326e9034bc
Timestamp --> 1711633392 (2024-03-28 21:43:12)
Bits --> ffff001f
Information --> [ ID: 47813 | Name: UrbanWheels Transport Solutions | Product Type: Corrugated Box | Transportation Type
: FTL (Full Truckload) Shipping | Ordering Type: bulk orders | Ordering Amount (Kg): 100 ]
Mined --> Yes

--------------------------------------------------------------------------------
```

Figure 41: After Transporter Block Creation

```
1. Display blockchain
2. Search Block
3. Add Block
4. Manipulate Block
Select an action:4
 Selected an action: Manipulate Block

1. Edit
2. Delete
3. Mine Edited Blocks
Select Manipulation Type:1
 Selected Manipulation Type: Edit

1. Soft Edit
2. Hard Edit
Select Edit Type:2
 Selected Edit Type: Hard Edit

1. Block Type
2. Height
3. Version
4. Nonce
5. Current Hash
6. Previous Hash
7. Merkle Root
8. Timestamp
9. Bits
10. Information
Select an attribute to hard edit by:5
 Selected an attribute to hard edit by: Current Hash
```

Figure 42: Hard Delete (Part 1)

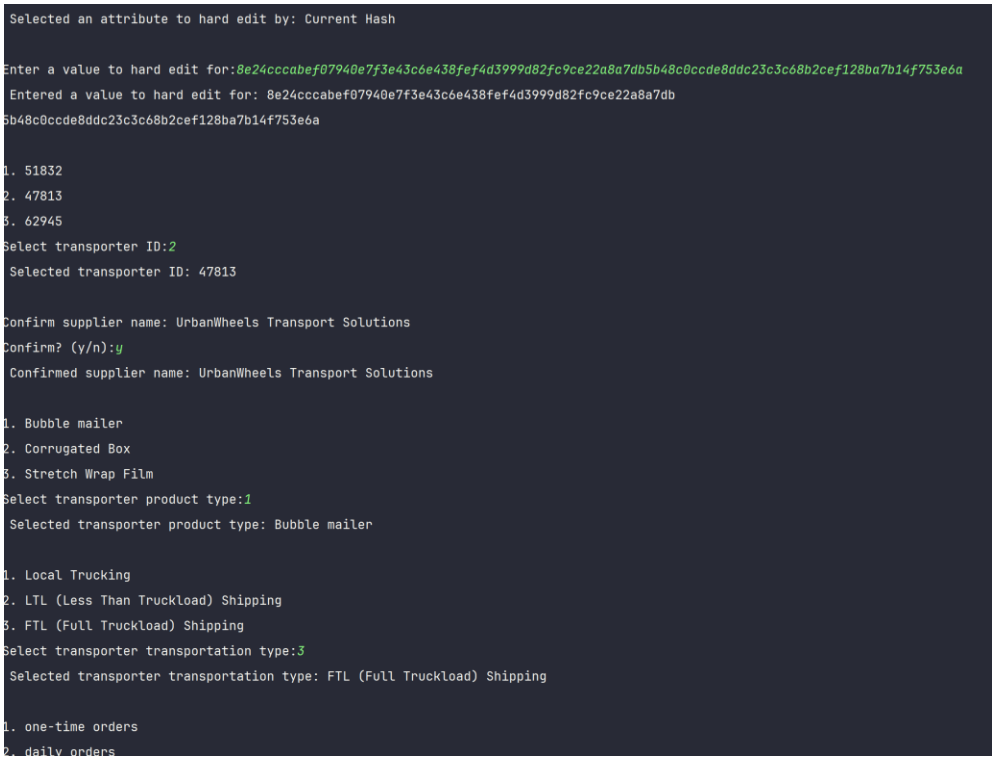

Figure 42: Hard Delete – Searched By Current Hash This Time (Part 2)

```
1. Local Trucking
2. LTL (Less Than Truckload) Shipping
3. FTL (Full Truckload) Shipping
Select transporter transportation type:3
 Selected transporter transportation type: FTL (Full Truckload) Shipping

1. one-time orders
2. daily orders
3. non-seasonal daily orders
4. bulk orders
5. customized orders
Select transporter ordering type:4
 Selected transporter ordering type: bulk orders

Enter transporter ordering payment type (kg):57
 Entered transporter ordering payment type (kg): 57

Block edited successfully.

1. Display blockchain
2. Search Block
3. Add Block
4. Manipulate Block
Select an action:1
 Selected an action: Display blockchain
```

Figure 42: Hard Delete – Searched By Current Hash This Time (Part 3)

```
Block Type --> Transporter
Height --> 1
Version --> 1
Nonce --> 630
Current Hash --> 8b1dff821ad9afd11b9c230f885960b9e3a130bcc9c134c3b69f4484c56cd0927a656e8bbed2227a96533e679d131329
Previous Hash --> 8b1dff821ad9afd11b9c230f885960b9e3a130bcc9c134c3b69f4484c56cd0927a656e8bbed2227a96533e679d131329
Merkle Root --> 7863382847d67038382f08ff1618258f9e68f726b6c09b35b30e8bb6cc427adf431551eaa1dbaaf5e6d1a8837b1424cd
Timestamp --> 1711633520 (2024-03-28 21:45:20)
Bits --> ffff001f
Information --> [ ID: 47813 | Name: UrbanWheels Transport Solutions | Product Type: Bubble mailer | Transportation Type:
 FTL (Full Truckload) Shipping | Ordering Type: bulk orders | Ordering Amount (Kg): 57 ]
Mined --> No
```

Figure 43: Display Blockchain after Hard Delete (Value Changed)

```
10. Information
Select an attribute to search by:7
 Selected an attribute to search by: Merkle Root

Enter a value to search for:7863382847d67038382f08ff1618258f9e68f726b6c09b35b30e8bb6cc427adf431551eaa1dbaaf5e6d1a8837b1424cd
 Entered a value to search for: 7863382847d67038382f08ff1618258f9e68f726b6c09b35b30e8bb6cc42
7adf431551eaa1dbaaf5e6d1a8837b1424cd

---------------------------------- SELECTED BLOCKS ----------------------------------

Block Type --> Transporter
Height --> 1
Version --> 1
Nonce --> 630
Current Hash --> cfceb52612ca9cbf1431800ab4902a466c3cf64925aeddf9f9d534784e61fa6c01feba6c728d6503f923a686daab43a1
Previous Hash --> e787690284c5254864ee47bb1db94ebaf5bf2e1124d5ef58339703b12eb9657f
Merkle Root --> 7863382847d67038382f08ff1618258f9e68f726b6c09b35b30e8bb6cc427adf431551eaa1dbaaf5e6d1a8837b1424cd
Timestamp --> 1711633520 (2024-03-28 21:45:20)
Bits --> ffff001f
Information --> [ ID: 47813 | Name: UrbanWheels Transport Solutions | Product Type: Bubble mailer | Transportation Type:
 FTL (Full Truckload) Shipping | Ordering Type: bulk orders | Ordering Amount (Kg): 57 ]
Mined --> No

--------------------------------------------------------------------------------

1. Display blockchain
2. Search Block
3. Add Block
4. Manipulate Block
Select an action:
```

Figure 44: Search Blockchain after Hard Delete – Searched by Merkle Root (Value Changed)

```
Mining... Nonce: 155798, Hash: 3e60a15406ba0ef919c5414977bdd25cbd0c4ef063020041f3a9064ff155766b6f5eb380daf5a925577c7b621
Mining... Nonce: 155799, Hash: 036ba8d3d0b8961009816 0dd0e6cd7ed202f5c070fb7d9ccc1fdf3e4ae91ae04066abbd89164c98d276366f4a
Mining... Nonce: 155800, Hash: 4a2e76aa880fb4d82a5bc70c682e599b23facf80b659dcfd3de8d0ff27e2aa21d2c3a8010fb8ca7637c545c6b
Mining... Nonce: 155801, Hash: 99627cf4b3742e66cbee8939c4b187e10bcf8c852c37e3fbab499fd86ffd4d562a44faff2b04218a2d0b27f9b
Mining... Nonce: 155802, Hash: ca1fb06fbb65d5e05832b8d7b74a50b0fda45fd60a92d877d4db31cf46a009637b0a214a8e131ec4f39100d34
Mining... Nonce: 155803, Hash: c28d2db47ba6b5e964b59ad40e14f222cfaa38df7783f848ae577023fc82666adbd330cb0236e8cadf8ddcb12
Mining... Nonce: 155804, Hash: fb135a79c4b08c5c1ded00bc1da372ee49ee1317503c0c940f51ee65c862e4c321a7818d930c443b245107181
Mining... Nonce: 155805, Hash: 5da48e5f2be2f819dec8e6b3ae75100dfab1aeb6605a2fbf5f0f2d4d6166c85e93e9c0c9871583d51466b0946
Mining... Nonce: 155806, Hash: 2f8956ed2343ce7944a49a1ab2022d06a73890ff207e14d18dfdb0487be67b808752c7d85f762a3e67f60f7d4
Mining... Nonce: 155807, Hash: 1f8308487e25d7c08906e9c0013e4d856ca39053c9b6fcd9e8393fec1703ceed5c107b358707836f2d80a57c5
Mining... Nonce: 155808, Hash: 13741d9436b7e085822adef5528ca04f6b1aaecb65eab47c8c576f31f177ef0f87aa24ea692e640fc36a77b9e
Mining... Nonce: 155809, Hash: 0e3c3dcc134269938bfa431db484aabf4cdce68e9a01fdef6f5c9ea641503e47822517e753f6838d136569592
Mining... Nonce: 155810, Hash: 53a641fa88396103c072e68f311ad9b657e31512f783e341798a2252602e137aa76df20a28ecdad1cff6f9bea
Mining... Nonce: 155811, Hash: 48a978dfec6ee1c11d2f7a7e840aee4bedce1026c57f9d169b8e47a584717acbad82182c764537a86b28ef263
Mining... Nonce: 155812, Hash: b283cd6ee1aa4293555596298ba749046626c501b21d73d7b2be9417987e3ece9285977470af80823a811a95da
Mining... Nonce: 155813, Hash: f71db68b0dd4dab1b222c9a9c64181e404ccfc5a03fa39f13b90f311af2db2c2412713f3a6c4b1d95fe474e7b
be9ad4a

Block mined! Nonce: 155813, Hash: 0000b2fabfd594e9b6531a2bba59198318a0a96ba0579be8b0d0ffe00a0d695e9a972b30cad7899bbf40c4
6c4a51cc0e

Block mined successfully.

1. Display blockchain
2. Search Block
3. Add Block
4. Manipulate Block
Select an action:1
 Selected an action: Display blockchain
```

Figure 45: Remining of Tampered Block

Figure 46: Display Blockchain after Remining (Current Hash Changed)

In the realm of blockchain technology, the integrity of each block—after any attempt at tampering—is paramount to ensuring the system's overall reliability and security. Blockchain's foundational principles, including decentralization, cryptographic security, and immutability, inherently safeguard against unauthorized alterations. Specifically, the cryptographic hashing of data within each block, alongside the sequential chaining of these blocks, establishes a robust framework that discourages tampering. Any modification to a block's content necessitates recomputing not only the hashes for the altered block but also for all subsequent blocks in the chain, a computationally prohibitive task without consensus across the network. As highlighted by Kamel Boulos et al. (2018), blockchain technology's potential in reshaping healthcare through enhanced data integrity and security underscores the importance of maintaining the unaltered state of each block post-tampering attempts, thereby reinforcing the technology's value proposition in critical sectors such as healthcare.

The participant is given the choice to remine at any time in order to remain the pure absolute integrity of that particular block. Unmined blocks remained stained and tampered **flagged by its non-quadruple-zero target prefix pattern** and the Mined attribute is set to false.

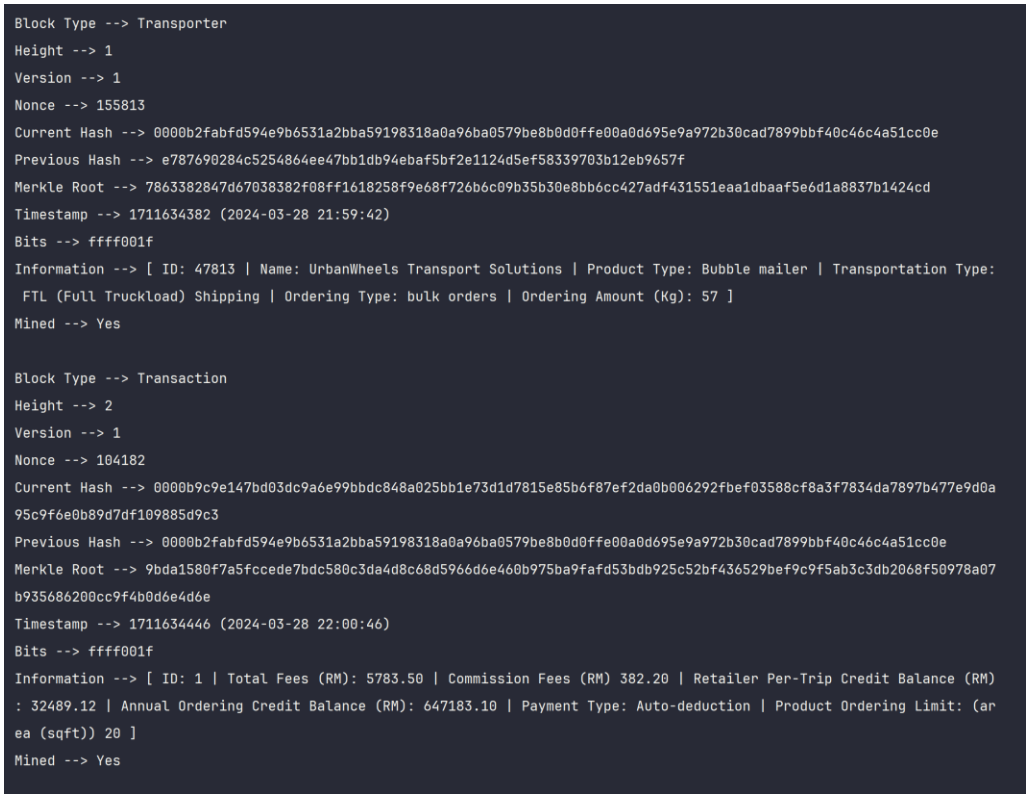

Figure 47: Display Blockchain after Adding New Block (Transaction)

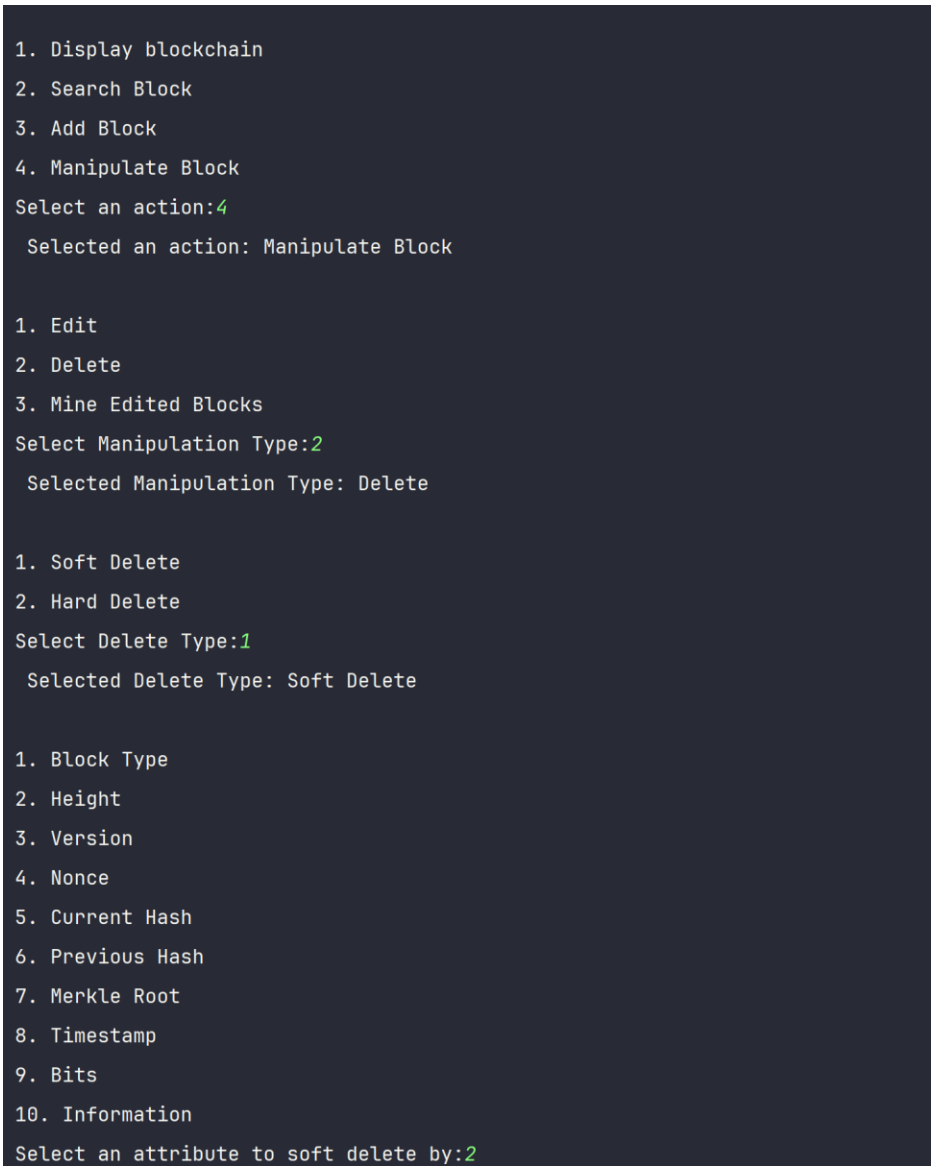

Figure 48: Soft Delete on Added Block (Part 1)

```
Select an attribute to soft delete by:2
 Selected an attribute to soft delete by: Height

Enter a value to soft delete for:2
 Entered a value to soft delete for: 2

Block deleted successfully.

1. Display blockchain
2. Search Block
3. Add Block
4. Manipulate Block
Select an action:1
 Selected an action: Display blockchain



---------------------------------- BLOCKCHAIN ----------------------------------

Block Type --> Supplier
Height --> 0 (Genesis Block)
Version --> 1
Nonce --> 19295
Current Hash --> e787690284c5254864ee47bb1db94ebaf5bf2e1124d5ef58339703b12eb9657f
Previous Hash --> e787690284c5254864ee47bb1db94ebaf5bf2e1124d5ef58339703b12eb9657f
Merkle Root --> 2f34432095ceaee4cd1be6b5868271f486d964f73f568907a749d28a89b03ace
Timestamp --> 1711634382 (2024-03-28 21:59:42)
Bits --> ffff001f
Information --> [ ID: 41673 | Name: First Stop All-In-One Supplies | Location: Bandar Bukit Raja, Klang, Selangor | Bran
ch: Klang | Items: Oranges, Milk ]
Mined --> Yes
```

Figure 49: Display Blockchain after Soft Delete (Part 1)

```
Version --> 1
Nonce --> 19295
Current Hash --> e787690284c5254864ee47bb1db94ebaf5bf2e1124d5ef58339703b12eb9657f
Previous Hash --> e787690284c5254864ee47bb1db94ebaf5bf2e1124d5ef58339703b12eb9657f
Merkle Root --> 2f34432095ceaee4cd1be6b5868271f486d964f73f568907a749d28a89b03ace
Timestamp --> 1711634382 (2024-03-28 21:59:42)
Bits --> ffff001f
Information --> [ ID: 41673 | Name: First Stop All-In-One Supplies | Location: Bandar Bukit Raja, Klang, Selangor | Bran
ch: Klang | Items: Oranges, Milk ]
Mined --> Yes

Block Type --> Transporter
Height --> 1
Version --> 1
Nonce --> 155813
Current Hash --> 0000b2fabfd594e9b6531a2bba59198318a0a96ba0579be8b0d0ffe00a0d695e9a972b30cad7899bbf40c46c4a51cc0e
Previous Hash --> e787690284c5254864ee47bb1db94ebaf5bf2e1124d5ef58339703b12eb9657f
Merkle Root --> 7863382847d67038382f08ff1618258f9e68f726b6c09b35b30e8bb6cc427adf431551eaa1dbaaf5e6d1a8837b1424cd
Timestamp --> 1711634382 (2024-03-28 21:59:42)
Bits --> ffff001f
Information --> [ ID: 47813 | Name: UrbanWheels Transport Solutions | Product Type: Bubble mailer | Transportation Type:
 FTL (Full Truckload) Shipping | Ordering Type: bulk orders | Ordering Amount (Kg): 57 ]
Mined --> Yes

--------------------------------------------------------------------------------


1. Display blockchain
2. Search Block
3. Add Block
4. Manipulate Block
Select an action:
```

Figure 50: Display Blockchain after Soft Delete – (Transaction Block Virtually Removed) (Part 2)

```
1. Display blockchain
2. Search Block
3. Add Block
4. Manipulate Block
Select an action:2
 Selected an action: Search Block

1. Block Type
2. Height
3. Version
4. Nonce
5. Current Hash
6. Previous Hash
7. Merkle Root
8. Timestamp
9. Bits
10. Information
Select an attribute to search by:2
 Selected an attribute to search by: Height

Enter a value to search for:2
 Entered a value to search for: 2

---------------------------------- SELECTED BLOCKS ----------------------------------
```

Figure 51: Search for Soft Deleted Block (Still There, No Value Change)

```
1. Display blockchain
2. Search Block
3. Add Block
4. Manipulate Block
Select an action:4
 Selected an action: Manipulate Block

1. Edit
2. Delete
3. Mine Edited Blocks
Select Manipulation Type:2
 Selected Manipulation Type: Delete

1. Soft Delete
2. Hard Delete
Select Delete Type:2
 Selected Delete Type: Hard Delete

1. Block Type
2. Height
3. Version
4. Nonce
5. Current Hash
6. Previous Hash
7. Merkle Root
8. Timestamp
9. Bits
10. Information
Select an attribute to hard delete by:2
```

Figure 52: Hard Delete a Block (Part 1)

```
 Selected an attribute to hard delete by: Heig

Enter a value to hard delete for:2
 Entered a value to hard delete for: 2

Block deleted successfully.

1. Display blockchain
2. Search Block
3. Add Block
4. Manipulate Block
Select an action:1
 Selected an action: Display blockchain
```

Figure 53: Hard Delete a Block (Part 2)

```
---------------------------------- BLOCKCHAIN ----------------------------------

Block Type --> Supplier
Height --> 0 (Genesis Block)
Version --> 1
Nonce --> 19295
Current Hash --> e787690284c5254864ee47bb1db94ebaf5bf2e1124d5ef58339703b12eb9657f
Previous Hash --> e787690284c5254864ee47bb1db94ebaf5bf2e1124d5ef58339703b12eb9657f
Merkle Root --> 2f34432095ceaee4cd1be6b5868271f486d964f73f568907a749d28a89b03ace
Timestamp --> 1711634757 (2024-03-28 22:05:57)
Bits --> ffff001f
Information --> [ ID: 41673 | Name: First Stop All-In-One Supplies | Location: Bandar Bukit Raja, Klang, Selangor | Bran
ch: Klang | Items: Oranges, Milk ]
Mined --> Yes

Block Type --> Transporter
Height --> 1
Version --> 1
Nonce --> 155813
Current Hash --> 0000b2fabfd594e9b6531a2bba59198318a0a96ba0579be8b0d0ffe00a0d695e9a972b30cad7899bbf40c46c4a51cc0e
Previous Hash --> e787690284c5254864ee47bb1db94ebaf5bf2e1124d5ef58339703b12eb9657f
Merkle Root --> 7863382847d67038382f08ff1618258f9e68f726b6c09b35b30e8bb6cc427adf431551eaa1dbaaf5e6d1a8837b1424cd
Timestamp --> 1711634757 (2024-03-28 22:05:57)
Bits --> ffff001f
Information --> [ ID: 47813 | Name: UrbanWheels Transport Solutions | Product Type: Bubble mailer | Transportation Type:
 FTL (Full Truckload) Shipping | Ordering Type: bulk orders | Ordering Amount (Kg): 57 ]
Mined --> Yes

--------------------------------------------------------------------------------
```

Figure 54: Display Blockchain after Hard Delete (Value Changed – Block No Longer there Virtually)

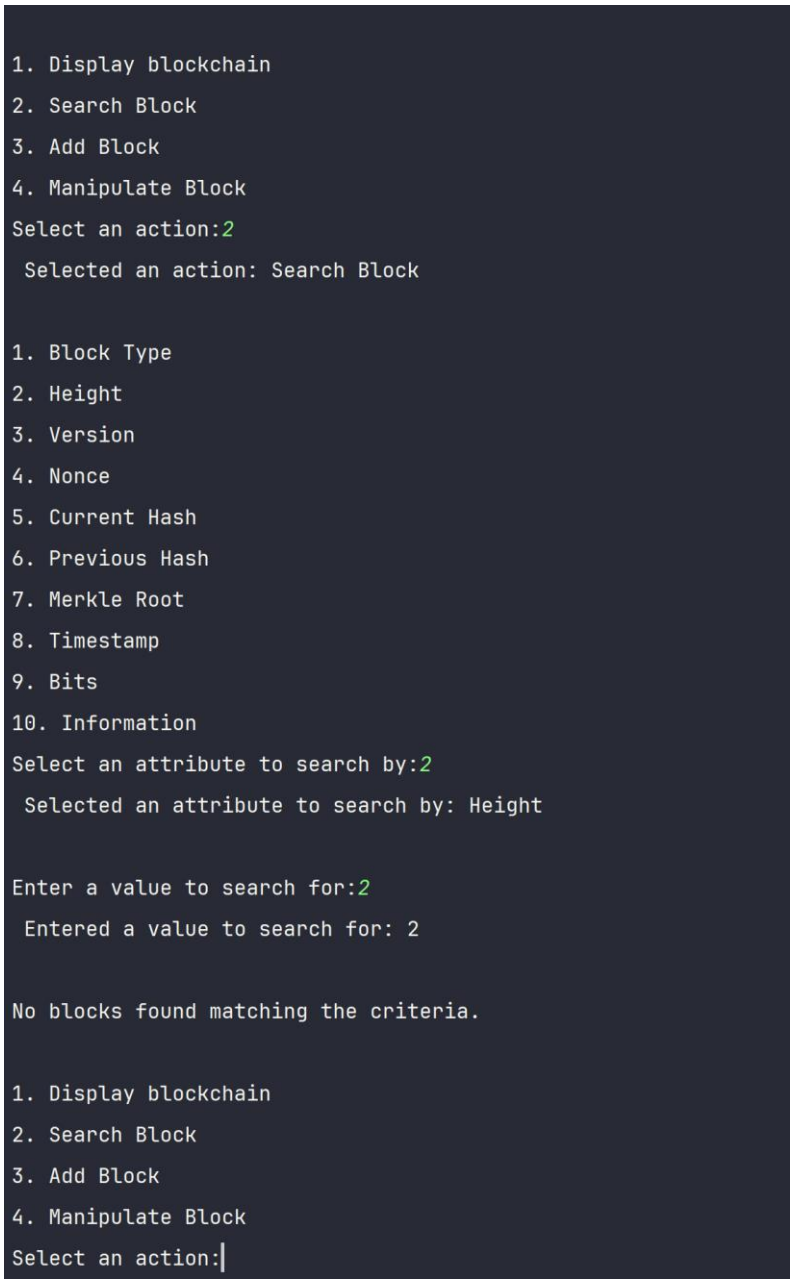

Figure 55: Search for Block after Hard Delete (Value Also Changed No longer there)

```
1. Display blockchain
2. Search Block
3. Add Block
4. Manipulate Block
Select an action:g
 Invalid input. Please enter a number or 'q' to quit.
1. Display blockchain
2. Search Block
3. Add Block
4. Manipulate Block
Select an action:asdihiuh
 Invalid input. Please enter a number or 'q' to quit.
1. Display blockchain
2. Search Block
3. Add Block
4. Manipulate Block
Select an action:132
 Invalid selection. Please try again.
1. Display blockchain
2. Search Block
3. Add Block
4. Manipulate Block
Select an action:
```

Figure 56: Thorough Input Validations (Part 1)

```
1. Display blockchain
2. Search Block
3. Add Block
4. Manipulate Block
Select an action:3
 Selected an action: Add Block

Enter transaction ID (unique):asd
 Invalid input. Please enter an integer.
Enter transaction ID (unique):1
 The entered transaction ID (unique) has been taken. Please enter a unique value.
Enter transaction ID (unique):2
 Entered transaction ID (unique): 2
```

Figure 57: Thorough Input Validations (Part 2)

```
Confirm supplier name: Giant Supply Solutions
Confirm? (y/n):j
 Invalid input. Please enter 'y' or 'Y' or 'n' or 'N' only.
Confirm supplier name: Giant Supply Solutions
Confirm? (y/n):s
 Invalid input. Please enter 'y' or 'Y' or 'n' or 'N' only.
Confirm supplier name: Giant Supply Solutions
Confirm? (y/n):y
 Confirmed supplier name: Giant Supply Solutions
```

Figure 58: Thorough Input Validations (Part 3)

10.0 Extraordinary Features

10.1 External Libraries for SHA256, SHA384, SHA512 (Partial and Licensed)

Note: Due to the incapability of my device for implementing a full external library, it was a go to and much safer route to use licensed partial external libraries.

The program utilizes **three** of the **Secure Hash Algorithms** (SHA) cryptographic hash functions family designed to provide a secure means of computing the digital fingerprint on each of the Block in the blockchain. Namely the popular SHA256, SHA384, and SHA512.

Utilizing multiple different hashing algorithms for block hashing within a blockchain environment significantly enhances the system's integrity by bringing about a more balanced and even distribution of data, making it less susceptible to attacks and ensuring a fair and equitable resource allocation among users. For example, Lan et al. (2012) demonstrated that a two-level hashing function in consistent hashing algorithms could result in a more even distribution of data, requiring fewer virtual nodes, thus highlighting the performance and scalability benefits of using varied hashing methods. Furthermore, Yu and Moulin (2015) illustrated how incorporating multiple feature types through different hashing algorithms could enhance performance in vision tasks, optimizing the signal-to-noise ratio in binary codes generated. This not only maximizes data efficiency but also significantly boosts the performance and accuracy of data retrieval and classification processes within blockchain applications.

Moreover, in any scenario or case whereby a single hashing algorithm gets revealed or through hacking, the hacker will only be able to penetrate one type of block, hence ensuring the safety of all the other block variants.

Aspect	Single Hashing Algorithm	Multiple Hashing Algorithms
Simplicity and Performance	High simplicity in implementation and predictable performance	Increased complexity with potentially optimized but variable performance.
Security and Collision Resistance	Vulnerable to algorithm weaknesses; entire system at risk if compromised. Lower collision resistance based on a single algorithm's properties.	Enhanced security through diversity; a lot harder to compromise. Reduced risk of collisions by diversifying hashing strategies.
Flexibility, Upgradeability, and Consistency	Less flexible, system-wide overhaul needed for upgrades, with consistent hashing results.	Greater adaptability to security threats, individual algorithms can be upgraded with less systemic impact, due to harder to predict algorithms.

10.2 Dynamic Blockchain (Soft & Hard Edits & Delete Operations)

Incorporating virtual soft edits and soft deletes, along with hard edits and deletes, significantly enhances the flexibility and applicability of blockchain technology to real-world scenarios. This nuanced approach to data modification and removal caters to diverse needs for data accuracy, privacy, and compliance while maintaining the essential characteristics of the blockchain.

10.2.1 Soft Edits and Deletes: Virtual Adjustments

Soft edits and deletes operate on a virtual level, meaning the original data remains intact on the blockchain, preserving its immutable nature. When these operations are performed, they're essentially appended as new transactions that indicate an update or mark data for deletion.

Visibility and Accessibility: In the context of displaying information or querying the blockchain, soft-edited or soft-deleted data is effectively hidden or marked as obsolete. This ensures that the most current and relevant information is presented to users. However, the original data and its subsequent modifications are still part of the blockchain's ledger, which can be crucial for auditability and historical analysis. When specifically searched for, this data, along with its edit or delete markers, becomes accessible, providing a transparent trail of all changes without erasing history.

Practical Implications: This approach is particularly beneficial for **addressing errors**, **updating information over time,** or responding to **privacy requests** in a way that aligns with regulatory frameworks like GDPR. It allows for dynamic content management within the blockchain while upholding its foundational principles of transparency and immutability.

10.2.2 Hard Edits and Deletes: Complete Virtual Exclusion

Hard edits and deletes extend the flexibility of blockchain applications by allowing for more comprehensive data management, albeit virtually. Despite their transformative impact on how data is presented, these operations do not physically remove data from the blockchain; instead, they alter the visibility of the data in both displaying and searching contexts, effectively making it invisible.

Displaying and Searching: With hard edits or deletes, the affected data will not show up during normal display or search operations, **mimicking** the effect of actual data removal. This approach addresses scenarios where data needs to be **rendered inaccessible or irrelevant** due to **evolving information needs**, errors, or privacy concerns. Despite this, the data technically remains within the database or text file, preserving the blockchain's comprehensive historical record and immutability.

10.3 Acquiring Proper Hash and Nonce Value (Block Mining)

In the context of blockchain technology, Mardiansyah and Sari (2021) highlighted the significance of implementing the **Proof-of-Work** (PoW) concept within the SimBlock simulator, particularly emphasizing the role of leading-zero functions in solving mathematical problems inherent to the PoW concept. The study underscored the computational resources required as the criteria for leading zeros increase, affecting the compile-time for blockchain simulation, thereby offering a detailed educational insight into the blockchain mining process (Mardiansyah & Sari, 2021). The program's requirements for a hash to begin with the specific pattern "0000", makes it computationally challenging to find a valid hash, thus securing the network **against spam** and **malicious attacks.**

Block mining fundamentally transforms the transaction validation process into a democratic one, **preventing any single party from dominating blockchain control**. This democratization is realized through a cryptographic challenge that miners strive to decode, where the challenge's **difficulty level, in bits**, is dynamically tailored according to the collective computational strength of the network. Requiring that the hash prefixes with the pattern "0000". This will naturally regulate the pace at which new blocks are integrated into the blockchain.

10.4 Complete Input Validation

Thorough and complete input validation is important for a long-term usability of software systems, including those based on complex structures like blockchain technology. Input validation acts as the first line of defense against various forms of data corruption, including injection attacks, data **type mismatches,** and unintended data manipulation. By rigorously verifying and sanitizing user inputs before processing or storing them, systems can **prevent malicious actors from exploiting input data** to compromise system integrity, access unauthorized information, or perform unauthorized actions. Furthermore, comprehensive input validation contributes significantly to maintaining data integrity and consistency across the system, ensuring that the inputs adhere to the expected formats, types, and constraints. This is particularly crucial in applications requiring high precision and accuracy, such as financial transactions, medical records management, and supply chain logistics, where incorrect or malformed data can lead to significant real-world consequences.

Beyond security implications, effective input validation enhances user experience by **providing immediate feedback** on input validity, systems can guide users to correct errors and ensure the accuracy of the data being submitted. This interaction not only reduces the likelihood of operational disruptions caused by invalid data but also **minimizes the need for costly data cleansing and error handling downstream.** In the context of blockchain technologies, where data immutability is a core feature, ensuring the accuracy and validity of inputs at the point of entry is especially critical. Once data is recorded on a blockchain, correcting inaccuracies can be complex and resource-intensive, emphasizing the importance of getting input validation right the first time. Thus, thorough and complete input validation is necessary to not just improved user experiences qualities but also to reduce technical debt down the line in the future.

10.5 Blockchain Searching by Any Attribute

Enabling comprehensive search functionality across various data attributes within a blockchain system, beyond conventional identifiers like ID or timestamp, significantly enhances the system's accessibility and utility. This multifaceted search capability allows users to query the blockchain more effectively, making their search better. For instance, in a blockchain containing supplier, transporter, and transaction blocks, the ability to search using attributes such as "Name", "Location", "Product Type", and “Date and Time”, empowers the participants to **extract precise information swiftly.** It facilitates targeted audits, detailed analyses, and informed decision-making by providing a granular view of the blockchain's contents. Moreover, this approach aligns with the principles of transparency and traceability central to blockchain technology, ensuring stakeholders can easily verify, track, and analyze activities within the system. Such comprehensive search capabilities, therefore, not only improve the user experience but also bolster the blockchain's role as a reliable and transparent ledger for diverse applications.

11.0 Overall Experience

11.1 Theoretical Component

The theoretical component of the coursework provided a comprehensive foundation in the core principles of blockchain technology. It delved into the depths of cryptography, the mechanics of consensus algorithms, and the idea of decentralization that defines blockchain. This study not only clarified the conceptual framework behind blockchain but also its broader implications in digital transactions. Understanding the theory of blockchain technology helped me in appreciating its revolutionary potential and its limitations. This knowledge base served as an essential springboard for exploring the practical applications of blockchain, offering insights into the challenges and opportunities presented by this transformative technology.

11.2 Practical Component

On the practical side, the coursework transitioned from theory to application, providing hands-on experience in the design and development of a blockchain system for an Inventory and Transportation Management System for ABC Solutions Sdn Bhd.. This phase involved properly coding the implementation of blockchain principles using **object-oriented programming** and **SOLID design principles**. The practical exercise emphasized problem-solving skills, coding best practices, and the nuances of creating a blockchain that is both robust and adaptable. By applying theoretical knowledge in a real-world context, it offered a direct experience with the complexities of developing blockchain solutions, from conceptualization to execution. This practical experience was valuable, creating a deeper understanding of how blockchain technology can be leveraged to create innovative solutions and highlighting the importance of meticulous design and implementation in ensuring the success and sustainability of blockchain systems.

12.0 Noteworthy Difficulties

12.1 Hardware Limitations

Even after multiple tries and modifications to my device's configurations, it still struggled with the computational demands required by these libraries. OpenSSL, being a robust tool for secure communication, requires substantial processing power for encryption and decryption processes. Similarly, Google Maps integration involves handling complex data and rendering detailed maps, which can be resource-intensive. The hardware limitations of my device hindered smooth operation and development, impacting my ability to efficiently work on these aspects of the project.

12.2 Mastering Advanced Programming Concepts

Navigating the various casting operators in C++ such as static_cast, dynamic_cast, reinterpret_cast, and const_cast, was one of the challenging aspects of my learning journey. These operators, each with its unique purpose and set of rules, initially seemed daunting and complex. However, by dedicating time to thoroughly read through the C++ documentation, I gradually developed a clearer understanding of how and when to use each of these casts effectively. The detailed explanations, examples, and guidelines provided in the official documentation played a crucial role in overcoming this initial hurdle.

13.0 Conclusion

Embarking on this coursework has been a great journey, combining theory and real-world practical applications of blockchain technology into a cohesive learning experience. Theoretical exploration provided a solid foundation in understanding the principles that govern blockchain technology, including cryptography, consensus mechanisms, and the importance of decentralization. This knowledge was crucial in appreciating the potential and limitations of blockchain applications. Complementing this, the practical aspect of designing and implementing a blockchain system was instrumental in translating theory into tangible outcomes. It offered skills that are essential for any aspiring blockchain developer. This dual approach not only enriched my understanding of blockchain technology but also honed my technical capabilities, preparing me for future challenges in this dynamic field. The coursework has significantly broadened my perspective, making it a profoundly rewarding educational journey.

14.0 References

1. Litke, A., Anagnostopoulos, D., & Varvarigou, T. (2019). Blockchains for Supply Chain Management: Architectural Elements and Challenges Towards a Global Scale Deployment. https://www.mdpi.com/2305-6290/3/1/5
2. Cai, C., Hao, X., Wang, K., & Dong, X. (2023). The Impact of Perceived Benefits on Blockchain Adoption in Supply Chain Management. https://www.mdpi.com/2071-1050/15/8/6634
3. Sangeetha, A. S., Shunmugan, S., & Murugan, G. (2020). Blockchain for IoT Enabled Supply Chain Management - A Systematic Review. https://www.researchgate.net/publication/346807173_Blockchain_for_IoT_Enabled_Supply_Chain_Management_-_A_Systematic_Review
4. Chen, X., & Liu, H. (2023). Blockchain Technology Participates in the Path and Mode Optimization of Supply Chain Finance. https://drpress.org/ojs/index.php/jid/article/view/9146
5. Saxena, S., Nagpal, A., Prashar, T., Shravan, M., Al-Hilali, A., & Alazzam, M. B. (2023). Blockchain for Supply Chain Traceability: Opportunities and Challenges. https://www.researchgate.net/publication/372603872_Blockchain_for_Supply_Chain_Traceability_Opportunities_and_Challenges
6. Swan, M. (2015). "Blockchain: Blueprint for a New Economy." O'Reilly Media. ISBN: 978-1491920497.
7. Mougayar, W. (2016). "The Business Blockchain: Promise, Practice, and Application of the Next Internet Technology." Wiley. ISBN: 978-1119300311.
8. Tapscott, D., & Tapscott, A. (2016). "Blockchain Revolution: How the Technology Behind Bitcoin and Other Cryptocurrencies is Changing the World." Portfolio. ISBN: 978-1101980132.
9. Antonopoulos, A. M., & Wood, G. (2018). "Mastering Ethereum: Building Smart Contracts and DApps." O'Reilly Media. ISBN: 978-1491971949.
10. Lan, Y., Ma, X., Zhang, Y., Ren, H., Yin, C., & Hu, H. (2012). Two Levels Hashing Function in Consistent Hashing Algorithm. https://www.researchgate.net/publication/274558727_Two_Levels_Hashing_Function_in_Consistent_Hashing_Algorithm
11. Mardiansyah, V., & Sari, R. F. (2021). Implementation of Proof-of-Work Concept Algorithm using SimBlock Simulator. 2021 11th IFIP International Conference on New Technologies, Mobility and Security (NTMS). DOI: 10.1109/NTMS49979.2021.9432645. https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=9432645

12. Kamel Boulos, M. N., Wilson, J. T., & Clauson, K. A. (2018). Geospatial blockchain: promises, challenges, and scenarios in health and healthcare. International Journal of Health Geographics, 17(25). https://doi.org/10.1186/s12942-018-0144-x
13. Yu, H., & Moulin, P. (2015). Multi-feature hashing based on SNR maximization. https://ieeexplore.ieee.org/document/7351114