COMP2034 Coursework 1 Report

Inventory and Transportation Management System using Blockchain

Prepared for

Dr. Doreen Sim Ying Ying

University of Nottingham Malaysia

22 March 2024

By

Lua Chong En

20417309

Table of Contents

1. Introduction

This coursework focuses on utilizing blockchain technology to build an Inventory and Transportation Management System. Blockchain technology has become an important invention. Its immutable and decentralized structure offers a transparent and safe means of recording transactions. Robust analytical techniques to allow for insightful information from blockchain data are becoming increasingly important as blockchain technology develops. This report has specifically chosen to build an Online Shopping Inventory and Transportation Management System, similar to the likes of Shopee and Lazada. Taking inspiration from self-use of online shopping applications and understanding the background on how these companies operate.

Tracking inventory, fulfilling orders, and managing transportation are some of the major issues that e-commerce platforms must deal with. These issues are addressed by the Online Shopping Inventory and Transportation Management System. Through the use of blockchain technology, the system offers a transparent and safe platform for handling these tasks. The system will also have specified information vector pairs built in to extract useful information from the transaction data kept on the blockchain. This initiative not only shows how blockchain may be used practically in the e-commerce industry, but it also highlights the possibility for creative solutions to problems facing the sector.

2. Project Scope

In this project, the main aim is to understand and implement blockchain technology. The designed blockchain is to display the whole product lifecycle when purchasing a product from an online shopping website. There are 8 stages in the proposed blockchain: Procurement Stage, Inventory Storage Stage, Order Fulfillment Stage, Transportation Stage, Customer Delivery Satisfaction Stage, Quality Control Stage, Product Return Stage, and Product Worthiness Stage. The main user of this program is the company that sells products on the Online Shopping Inventory and Transportation Management System.

The built blockchain is modular and flexible. There is no fixed dataset, therefore, it allows for user input for every single stage. Users can customize the blockchain to fit their specific needs, making the system versatile. This approach also promotes transparency and user involvement, as stakeholders can actively participate in the data input process, ensuring that the blockchain accurately reflects the real-world transactions and the process of inventory management for an online shopping website.

The blockchain system also has capabilities that improve data integrity and security. Because every transaction on the blockchain is cryptographically protected, there is no way for the data to be changed or tampered with. By combining these elements, the project hopes to show how blockchain technology can transform transportation and inventory management in the e-commerce industry and open the door to more effective and transparent business processes.

3. Project Aim

This report aims to utilize the features available in blockchain. In this project, the blockchain stores and records the stages of every process of the inventory management system. Smart contracts allow for immutability within the blockchain. Therefore, once the stakeholders have entered specific information into the block, no changes are entertained. But also making sure to utilize CPP features such as Object Oriented Programming, Pointers, Abstraction, Encapsulation, and many more.

Furthermore, real-time inventory and transportation process tracking and monitoring are made possible by the usage of blockchain technology. A visible and auditable record of transactions is provided by the blockchain, which is timestamped and securely stores each stage of the product life cycle.

4. Entities Involved

Note: Any stage can be added at any time, there is no specific order. This is to further exemplify the modularity of the dynamic blockchain, this is a specific extraordinary feature that is explained further under “Extraordinary Features”.

a. Procurement Stage (When adding block, click 1 to add stage 1)
b. Inventory Storage Stage (When adding block, click 2 to add stage 2)
c. Order Fulfillment Stage (When adding block, click 3 to add stage 3)
d. Transportation Stage (When adding block, click 4 to add stage 4)
e. Customer Delivery Satisfaction Stage (When adding block, click 5 to add stage 5)
f. Quality Control Stage (When adding block, click 6 to add stage 6)
g. Product Return Stage (When adding block, click 7 to add stage 7)
h. Product Worthiness Stage (When adding block, click 8 to add stage 8)

5. Online Shopping Supply Chain Flow

Stage	Description
Procurement Stage	Company purchases product and lists product on website
Inventory Storage Stage	Company stores product in Inventory space
Order Fulfillment Stage	Customer purchases a product, Company prepares to fulfill order
Transportation Stage	Company dispatches order to transportation company to prepare for delivery
Customer Delivery Satisfaction Stage	Customer fills out survey after successful delivery of product
Quality Control Stage	Company performs quality control to determine overall quality of product

Product Return Stage	Customer can select to return or refund product after successful delivery
Product Worthiness Stage	Company determines the worthiness of the product on whether to continue or discontinue the product

Stage 1: Procurement Stage, this is the first stage in the blockchain, represented as Block 0. Key information of Supplier ID, Supplier Name, Order Quantity, Order Date, Order State, and Shipping Details are collected. There is error handling incorporated for supplier ID, specifically “SIDXXXXX”. Furthermore, only numbers can be entered for order quantity. For order date, the entered late must be in “XX/XX/XX” format. And for order state, the user must type from one of three of the displayed order states. Shipping Details are also collected, ensuring that all necessary information for shipping the products is captured accurately.

Stage 2: Inventory Stage, this is the second stage in the blockchain, represented as Block 1. Key information of Warehouse ID, Storage Location, Inventory Quantity, and Inventory Status are collected. Note that for storage location, only correct cities and states in the correct format are accepted. Additionally, the Inventory Quantity is recorded to track the number of items available in the warehouse. This information is crucial for inventory management, as it helps in planning and replenishing stock levels. The Inventory Status indicates the current status of the inventory, such as whether it is available for sale, reserved, or out of stock. This status is updated in real-time as inventory transactions occur, ensuring that stakeholders have up-to-date information on the availability of products.

Stage 3: Order Fulfillment Stage, this is the third stage in the blockchain, represented as Block 2. Key information of Customer ID, Order Quantity, Order Date, Order State, and Shipping Details are collected. The Order Date records the date when the order is fulfilled, providing a timeline of order fulfillment. The Order State indicates the current status of the order fulfillment process, such as pending, processing, or completed, and users must select from the displayed order states. Shipping Details are also collected to

ensure that the order is shipped to the correct address and with the correct shipping method.

Stage 4: Transportation Stage, this is the fourth stage in the blockchain, represented as Block 3. Key information of Transportation Mode, Transportation Company, Transportation Route, From (City, State), To (City, State), Transportation Departure Date, and Transportation Arrival Date is collected. The Transportation Departure Date and Transportation Arrival Date record the dates when the transportation departs from the warehouse and arrives at the customer's location, providing a timeline of the transportation process.

Stage 5: Customer Delivery Satisfaction Stage, this is the fifth stage in the blockchain, represented as Block 4. Key information of Delivery Confirmation, Customer Feedback Collection Result, and Satisfaction Survey Result. The Delivery Confirmation field indicates whether the delivery was successfully completed. The Customer Feedback Collection Result records the feedback provided by the customer regarding the delivery process. The Satisfaction Survey Result captures the customer's overall satisfaction with the delivery process, helping to improve future delivery operations based on customer feedback.

Stage 6: Quality Control Stage, this is the sixth stage in the blockchain, represented as Block 5. Key information of Product Quality Inspection Result and Product Quality. Note that for both information pairs, users must type from choices displayed. The Product Quality Inspection Result indicates the result of the quality inspection process, such as pass or fail. The Product Quality specifies the quality of the product, such as high, medium, or low. Users must select from the displayed choices for both information pairs, ensuring that the data is accurately recorded and standardized. This stage plays a crucial role in maintaining product quality and ensuring customer satisfaction.

Stage 7: Product Return Stage, this is the seventh stage in the blockchain, represented as Block 6. Key information of Product Return Status, Product Refund Status, and Product Return Reason. The Product Refund Status specifies the status of the refund process for the returned product. The Product Return Reason records the reason

provided by the customer for returning the product, helping the company understand customer preferences and improve product offerings.

Stage 8: Product Worthiness Stage, this is the eighth and the last stage in the blockchain, represented as Block 7. Key information of Product Worthiness Status, Product Worthiness Reason, and a confirmation of the product worthiness status. This block connects with the fifth stage in the blockchain. It allows the user to reconsider the product worthiness after receiving customer feedback. The Product Worthiness Reason records the reason for the product's worthiness status, providing insights into customer preferences and product quality. This stage connects with the fifth stage in the blockchain, ensuring that customer feedback is considered in determining the product's worthiness.

6. Design Structure of Blockchain & Block

The Blockchain uses a linked list and vector structure. Example shown below shows 2 identical blocks, both blocks are linked to each other through the hash number. Take note that “i” and “i+1” are examples of the incrementing of the blocks and how they connect to each other. Further below, are proper examples of how the blockchain could potentially look like.

Note: Blocks can be added in any order, below is just an example. The 8 blocks shown below are shown in order, however note that blocks can be added in any order. This is to allow for flexibility and modularity of the dynamic blockchain.

Block i
Hash of Current Block
Hash of Previous Block
Timestamp

Block i+1
Hash of Current Block
Hash of Previous Block
Timestamp

Information Key Value Pairs
- Block: “XXX Block”
- Key Value Pair 1
- Key Value Pair 2
- Key Value Pair 3
- Key Value Pair 4
- Key Value Pair 5
- Key Value Pair 6

Information Key Value Pairs
- Block: “XXX Block”
- Key Value Pair 1
- Key Value Pair 2
- Key Value Pair 3
- Key Value Pair 4
- Key Value Pair 5
- Key Value Pair 6

The blockchain represents the product life cycle of a product listed on Online Shopping Inventory and Transportation Management System. Each block is a node in a linked list and each block has different key value pairs for information represented by a vector.

Each block contains Block Number, Current Block Hash, Previous Block Hash, Timestamp, and the Information regarding the block. Each block connects to each other through hash numbers.

Note: 1st block in the blockchain is represented as block 0, with a hash value pointing to itself, as per the guidelines in the coursework sheet provided by Dr. Doreen.

Block 0
Hash of Current Block
Hash of Previous Block
Timestamp

Block 1
Hash of Current Block
Hash of Previous Block
Timestamp

Block 2
Hash of Current Block
Hash of Previous Block
Timestamp

Information Key Value Pairs
- Block: Procurement Information
- Supplier ID
- Supplier Name
- Order Quantity
- Order Date
- Order State
- Shipping Details

Information Key Value Pairs
- Block: Inventory Information
- Warehouse ID
- Storage Location
- Inventory Quantity
- Inventory Status

Information Key Value Pairs
- Block: Order Fulfillment Information
- Customer ID
- Order Quantity
- Order Date
- Order State

Block 3
Hash of Current Block
Hash of Previous Block
Timestamp
Information Key Value Pairs
- Block: Transportation Information
- Transportation Mode

Block 4
Hash of Current Block
Hash of Previous Block
Timestamp
Information Key Value Pairs
- Block: Customer Delivery Satisfactory Information

Block 5
Hash of Current Block
Hash of Previous Block
Timestamp
Information Key Value Pairs
- Block: Quality Inspection Control Information

- Transportation Company
- Transportation Route
- Transportation Departure Date
- Transportation Estimated Arrival Date

- Delivery Confirmation
- Customer Feedback Collection Result
- Satisfaction Survey Result

- Quality Inspection Result
- Product Quality

Block 6
Hash of Current Block
Hash of Previous Block
Timestamp
Information Key Value Pairs
- Block: Product Returns Information
- Product Return Number
- Product Return Status

Block 7
Hash of Current Block
Hash of Previous Block
Timestamp
Information Key Value Pairs
- Block: Product Worthiness Information
- Product Worthiness Status
- Product Worthiness Reason

7. CPP Features Used

a. Arrays: Arrays used in code to store collections of elements, used throughout the code. Storing blocks in a blockchain or storing key-value pairs in vectors, allowing for efficient data organization and access.
b. Pointers: Pointers are extensively used in the implementation of the linked list that represents the blockchain. BlockNode is a struct that contains a pointer to the next block in the chain. They enable the creation of a chain of blocks by pointing from one block to the next, ensuring the integrity and continuity of the blockchain.
c. Array of Pointers: Array of pointers could be used to store pointers to each block in the blockchain, allowing for easy access and traversal. Enhancing the efficiency of blockchain operations.
d. Linked List: The blockchain is implemented as a linked list. Each BlockNode contains a Block and a pointer to the next BlockNode, forming a chain of blocks. Ensuring a continuous and secure blockchain structure.
e. Vectors: Vectors used to store information of key-value pairs. Provides a flexible and dynamic way to manage data within blocks.
f. Structures: Structures are used to define the Block and BlockNode types. The Block struct contains information about each block, such as block number, hash, timestamp, and information. The BlockNode struct contains a Block and a pointer to the next BlockNode.
g. Loops
    i. For Loops: For loops are used for iterating over the block information vector to display or export block information.
    ii. While Loops: While loops are used for traversing the blockchain linked list to search for blocks or to display the entire blockchain.
    iii. Do-While Loops: Used extensively to execute code at least once before satisfying the while condition.
h. Control Statements
    i. If Statements: If statements are used to conditionally execute code based on certain conditions, such as in the authentication process.

ii. If-else Statements: If-else statements are used to provide alternative paths of execution if the condition in the if statement is not met, as seen in the authentication process. Used throughout the code.
iii. Nested If Statements: Nested if statements are used to check multiple conditions within another if statement, providing more complex decision-making capabilities. Used throughout the code.
iv. Else Statements: Else statements are used to execute a block of code if the condition in the if statement is not met. Used throughout the code.

i. Random Number Generator for Hash Function: The code uses srand(time(0)) to seed the random number generator, which is then used in the generateRandomHash function to create a random hash for each block.

8. The Ideation and Design Principles Behind Building the Blockchain

The code demonstrates the foundational structures for a blockchain implementation, utilizing core principles of decentralization, transparency, and data integrity. The “Block” struct represents a block in the blockchain, containing essential attributes: block number, current and previous hash numbers, timestamp, and a vector to store information. Furthermore, the “BlockNode” struct serves as a node in the blockchain, linking blocks sequentially. The “isHardDeleted” and “isSoftDeleted” flags allows for strictly “virtual” deletion of blocks within the blockchain.

Next, the “Blockchain” class maintains a linked list of “BlockNode”s, where each “BlockNode” contains a “Block” representing a stage in the supply chain. Each block contains information such as procurement details, inventory status, order fulfillment, transportation, and customer satisfaction, ensuring that all relevant data is collected. The use of flags like “procurementAdded” and functions like “addProcurementInformation” ensures that information is added only once and once only, preserving the integrity of the blockchain. By using random hashes, timestamps, and previous hash references, it ensures that blocks are linked in a way that prevents traceability of every stage in the process.

The “addOrderFulfillmentInformation” and “addTransportationInformation” functions embody several key ideations and design principles crucial for building a robust blockchain system. It prioritizes data integrity and authenticity by validating user inputs for customer ID, order quantity, order date, order state, and transportation details, ensuring that only valid and correctly formatted data is added to the block. Additionally, the functions adhere to the principle of decentralization by not relying on a centralized authority for data verification, instead using local validation mechanisms to ensure the integrity of the information.

Furthermore, other blocks include: customer delivery satisfaction, quality inspection control, product return, and product worthiness blocks. By adding specific information to each block, such as delivery confirmation, customer feedback, and product quality, the blockchain captures a detailed history of product lifecycle events, ensuring transparency and accountability. The use of flags and validation loops ensures that only valid information is accepted, enhancing the integrity of the blockchain. Additionally, the use of random product return numbers and confirming product worthiness based on expected values demonstrates the concept of immutability, where once information is added to the blockchain, it cannot be altered, ensuring the integrity and trustworthiness of the data.

Functionality for generating a random hash, exporting blocks to a file, validating locations, getting the current block number, displaying the blockchain, searching for a block by number, and soft and hard deleting blocks also exists.

The provided main function implements a basic blockchain application for an Inventory and Transportation Management System. The design aligns with blockchain's principles by ensuring that once information is added to a block, it cannot be easily tampered with or deleted, promoting the immutability and security of the blockchain.

9. Explanation of Blockchain built in CPP

The blockchain uses a linked list structure with vectors for key-value pairs, using struct to define the block structure. Each block added into the blockchain is a node which also uses a struct to define the block nodes, a constructor is used to define the block nodes.

For every block, the user will be prompted to enter values for every key-value pair specified. The blockchain class has private and public members to further improve encapsulation. All methods that require the block, nodes, or vector as a parameter are defined as public methods under the block class.

The most significant method in the block class is “addBlock”. This method generates a block and stage to add into the blockchain. It calls functions defined throughout the code and uses them in each block, there is a switch to aid in separating the blocks to be added and improves organization. There is error handling to ensure that only one of each block can be added without duplicates to prevent misunderstanding. Furthermore, users can only add a maximum of 8 stages, no more. After each addition of a stage into the blockchain, the block number is incremented to account for the next block to be added. Note that blocks can be added in any order, this promotes modularity and flexibility for the user.

The next method present is the addProcurementInformation block that adds the information in vector key-value pairs. It first prompts the user to enter the Supplier ID in a specific format of “SIDXXXXX”, string “SID” with 5 numbers, error handling is incorporated to ensure that only a valid Supplier ID is entered. The next prompt is the supplier name and then the order quantity. For Order Quantity, error handling is incorporated to ensure that only numbers are entered. Next is the Order Date, only a valid date in the format of “XX/XX/XX” is accepted to ensure consistency. Additionally, users must choose to type one of 4 order states for the Order State prompt, users can enter in lowercase or uppercase. Lastly, the final prompt is Shipping Details. Each entered value is pushed into the vector.

The same theory of error handling, flexibility, and modularity applies for the following 7 methods for the next 7 stages in the blockchain. The blockchain is created specifically to allow for real-time data entry and flexibility, therefore allowing for the most accurate and reliable data.

For the addTransportationInformation block, it uses a separate text file “valid_locations.txt” and retrieves all valid cities and states in the format of (City, State). Users can only enter what is present in the text file, ensuring consistency and reliability.

For the addProductWorthinessInfromation block, it connects to the Customer Delivery Satisfaction Stage. Users can confirm the product worthiness after checking the Customer Feedback Number.

Also present is username and password authentication where users get a maximum of 3 tries to enter the system, else the program will be exited.

10. Extraordinary Features

1) Feature 1: Flexibility and modularity of blockchain, allows for user input. There is no fixed dataset, therefore, users can enter and store data in the blockchain in real time. Allows for a customizable blockchain. This feature is extremely important in this project, allows for flexibility to the user and makes the user experience emulate reality, in contrast when compared to using a basic dataset, which does not provide that.
2) Feature 2: ID Error handling, a specific format (example: SIDXXXXX). It maintains consistency and integrity of the blockchain, increasing security and reliability.
3) Feature 3: Random seed for generating random hash value for blocks. The random seed is the current time, therefore improving the security of the blockchain.
4) Feature 4: Error handling for general user input (using conditional statements and loops). Utilizes conditional statements and loops to validate user input, ensuring data integrity.
5) Feature 5: Users can choose specifically which block they want to add into the blockchain, not limited to a pre-specified dataset. Allows for real-time data entry.
6) Feature 6: Error handling for “Quantity” or number related inputs. Only numerical values are accepted, ensuring that quantity-related data is accurately recorded.
7) Feature 7: Date error handling, a specific format (example: XX/XX/XX) and must be a valid date. The specific date format prevents incorrect date entries.
8) Feature 8: Entering a valid choice from the given choices (example: Pending, In Progress, Completed, Canceled). This ensures data consistency.

9) Feature 9: Allowing for lowercase input entry, therefore regardless of uppercase or lowercase, input can be accepted. Not case sensitive. This feature enhances user experiences and allows for flexibility for user input.
10) Feature 10: Uses vectors for key-value pairs to store information. Vectors store information efficiently, allowing for quick retrieval and manipulation of data.
11) Feature 11: Location error handling, when a user is prompted to enter a location, they must enter the correct location as per “valid_locations.txt”. “Valid_locations.txt” contains all locations in Malaysia in (City, State) format. Ensures that locations are entered correctly by referencing a file containing valid locations.
12) Feature 12: For addProductReturnInformation method, it auto generates a random productReturnNumber for greater security. In a specific format of “RXXXXX”. Adds an extra layer of security by generating unique return numbers in a specific format (example: R12345).
13) Feature 13: For addProductWorthinessInformation method, it uses the customer feedback number and allows the user to validate the product worthiness. An added level of checking and balances.
14) Feature 14: Export blocks to an external text file “blockchain.txt”. The data is overwritten every time blocks are exported. Ensuring that data can be stored and accessed outside of the program.
15) Feature 15: Getter method to get the block number which is a private variable. Provides a way to retrieve the block number, which is a private variable, for external use.
16) Feature 16: DisplayChain method is a dynamic display method as it allows users to view specific blocks or all blocks at once. It is a “DEMO” display method and allows for “Soft Deletions” and “Hard Deletions”.
17) Feature 17: SearchBlockByNumber method is a dynamic search method that allows users to view specific blocks or all blocks at once. It is the actual display method, any deletion of sorts does not affect the blockchain as deletion is strictly “virtual”. Strictly unaffected by soft or hard deletions.
18) Feature 18: Soft delete, the method is modified compared to the actual implementation. In this implementation, it is specifically customized for this blockchain. Soft deletion removes any information related to the block, however, does not affect the block number or hash numbers. Note that the deletion is

strictly “virtual” and only affects the blocks in the display method, not the search method. This is because the display method is strictly a “DEMO” viewing method. Ensuring that deleted blocks are only hidden from view.

19) Feature 19: Hard delete, the method deletes the whole block, including block number and hash. Note that the deletion is strictly “virtual” and only affects the blocks in the display method, not the search method. This is because the display method is strictly a “DEMO” viewing method. Ensuring that deleted blocks are only hidden from view.
20) Feature 20: Authenticating users, uses an authenticate method. It reads through an external “username_password.txt” file to check for valid username and passwords. There is an added feature of only allowing the user only three tries to authenticate, otherwise the program exits.

11. Explanation of code

a. Part 1: Linked List Structure
    i. The linked list structure in the provided code consists of two main components: the “BlockNode” struct and the “Blockchain” class. Each “BlockNode” represents a block in the blockchain and contains a “Block” object as its data and a pointer to the next “BlockNode” in the chain. The “Block” struct defines the structure of each block, including its block number, current and previous hash numbers, current timestamp, a vector of information stored in the block, and flags indicating whether the block has been hard or soft deleted, any deletion is strictly “virtual”. The “Blockchain” class manages the linked list of blocks, with a pointer to the head of the chain and additional members such as “currentBlockNumber” to keep track of the current block number and flags (“procurementAdded”, “inventoryAdded”, etc.) to indicate which types of information have been added to the blockchain. Overall, this linked list structure allows for the creation and management of a blockchain with flexible data storage capabilities.

b. Part 2: Vector Structure
    i. “const vector<pair<string, string> >& info” is a constant reference to a vector of pairs of strings. The “const” keyword indicates that the vector itself is immutable and cannot be modified within the scope of the function. Using a reference (“&”) allows the function to access the vector's contents without making a copy. Each pair in the vector consists of two strings, thereby storing key-value pairs for the information entered by the user. The reference ensures that the vector is not modified accidentally and helps maintain the integrity of the data.

c. Part 3: Block Class
    i. The “Block” class represents a block in a blockchain, containing essential attributes and flags for managing data and deletion status. It has private members for the block number, current hash number, previous hash number, current timestamp, and a vector of pairs of strings to store information. The class also includes boolean flags for hard and soft deletion. The constructor initializes these attributes, setting the block number, current hash number, previous hash number, and current timestamp based on the input values. The flags for hard and soft deletion are initialized to false, indicating that the block is not deleted. This class is essential for representing the individual blocks that form the blockchain, each containing a set of data and references to the previous block.

d. Part 4: Methods Implemented
    i. The majority of the methods created are declared under the block class as public methods.

    The methods implemented within the "Block" class cover a wide range of functionality, including adding and displaying information stored in blocks, calculating hash values, and managing deletion flags. These methods are essential for interacting with the blockchain, enabling users to add new data, retrieve existing data, and manage the blockchain's state. By implementing these methods, the blockchain implementation becomes a

comprehensive solution for managing inventory and transportation data, providing users with the tools they need to effectively manage their data.

e. Part 5: Main Function
    i. The “main” function serves as the entry point for the program, initializing the blockchain and providing a user interface for interacting with it. It starts by seeding the random number generator and creating an instance of the “Blockchain” class. It then prompts the user for their username and password, attempting to authenticate them with a maximum of three login attempts. If authentication is successful, the program enters a menu loop where the user can choose various options, such as adding a block to the blockchain, displaying the blockchain, searching for a block, exporting the blockchain to a text file, hard deleting a block, or soft deleting a block. The loop continues until the user chooses to exit the program, at which point the program terminates. Overall, the main function orchestrates the user interaction with the blockchain system, ensuring a user-friendly experience while maintaining the integrity of the blockchain operations.

12. Screenshots of code

Note: For all user input, error handling has been carefully added to aid for a better user experience. This extraordinary feature is extensive and is seen throughout the code.

a. Scenario 1: Add Block (There are 9 further scenarios under Scenario 1)
    i. Add Procurement Block (Scenario 1.1)

```
Inventory and Transportation Management System.

Name: Lua Chong En
Student ID: 20417309


Enter Username: En


Enter Password: 123

Blockchain Menu
1. Add Block
2. Display Block - DEMO (Allows for Hard and Soft deletion which are strictly "Virtual")
3. Search Block (Deletions does not affect searching for blocks as deletions are strictly "Virtual")
4. Export to text file
5. Hard Delete Block
6. Soft Delete Block
7. Exit
Enter your choice: 1

Which block do you want to add? (1-8):

1. Procurement Stage

2. Inventory Storage Stage

3. Order Fulfillment Stage

4. Transportation Stage

5. Customer Delivery Satisfaction Stage

6. Quality Control Stage

7. Product Return Stage

8. Product Worthiness Stage

Enter your choice: 1

Enter Supplier ID (format: SIDxxxxx): SID12345
Enter Supplier Name: Lua Chong En
Enter Order Quantity: 10
Enter Order Date (format: dd/mm/yy): 22/02/24
Enter Order State (Pending, In Progress, Completed, Cancelled): pending
Enter Shipping Details: N/A


Procurement Information Block Successfully Added.
```

ii. Add Inventory Storage Block (Scenario 1.2)

```
Blockchain Menu
1. Add Block
2. Display Block - DEMO (Allows for Hard and Soft deletion which are strictly "Virtual")
3. Search Block (Deletions does not affect searching for blocks as deletions are strictly "Virtual")
4. Export to text file
5. Hard Delete Block
6. Soft Delete Block
7. Exit
Enter your choice: 1

Which block do you want to add? (1-8):

1. Procurement Stage

2. Inventory Storage Stage

3. Order Fulfillment Stage

4. Transportation Stage

5. Customer Delivery Satisfaction Stage

6. Quality Control Stage

7. Product Return Stage

8. Product Worthiness Stage

Enter your choice: 2

Enter Warehouse ID (format: WIDxxxxx): WID12345
Enter Storage Location (format: city, state): Semenyih, Selangor
Enter Inventory Quantity: 10
Enter Inventory Status (Available, Low Stock, or Not Available): low stock


Inventory Information Block Successfully Added.
```

iii. Add Order Fulfillment Block (Scenario 1.3)

```
Blockchain Menu
1. Add Block
2. Display Block - DEMO (Allows for Hard and Soft deletion which are strictly "Virtual")
3. Search Block (Deletions does not affect searching for blocks as deletions are strictly "Virtual")
4. Export to text file
5. Hard Delete Block
6. Soft Delete Block
7. Exit
Enter your choice: 1

Which block do you want to add? (1-8):

1. Procurement Stage

2. Inventory Storage Stage

3. Order Fulfillment Stage

4. Transportation Stage

5. Customer Delivery Satisfaction Stage

6. Quality Control Stage

7. Product Return Stage

8. Product Worthiness Stage

Enter your choice: 3

Enter Customer ID (format: CIDxxxxx): CID12345
Enter Order Quantity: 10
Enter Order Date (format: dd/mm/yy): 22/02/24
Enter Order State (Pending, In Progress, Completed, Cancelled): pending
Enter Shipping Details: N/A


Order Fulfillment Information Block Successfully Added.
```

iv. Add Transportation Block (Scenario 1.4)

```
Order Fulfillment Information Block Successfully Added.
Blockchain Menu
1. Add Block
2. Display Block - DEMO (Allows for Hard and Soft deletion which are strictly "Virtual")
3. Search Block (Deletions does not affect searching for blocks as deletions are strictly "Virtual")
4. Export to text file
5. Hard Delete Block
6. Soft Delete Block
7. Exit
Enter your choice: 1

Which block do you want to add? (1-8):

1. Procurement Stage

2. Inventory Storage Stage

3. Order Fulfillment Stage

4. Transportation Stage

5. Customer Delivery Satisfaction Stage

6. Quality Control Stage

7. Product Return Stage

8. Product Worthiness Stage

Enter your choice: 4

Enter Transportation Mode (Road, Rail, Sea, Air): road
Enter Transportation Company: J&T Express
Enter Transportation Route:
From (format: city, state): Semenyih, Selangor
To (format: city, state): Muar, Johor
Enter Transportation Departure Date (format: dd/mm/yy): 22/02/24
Enter Transportation Estimated Arrival Date (format: dd/mm/yy): 22/02/24


Transportation Information Block Successfully Added.
```

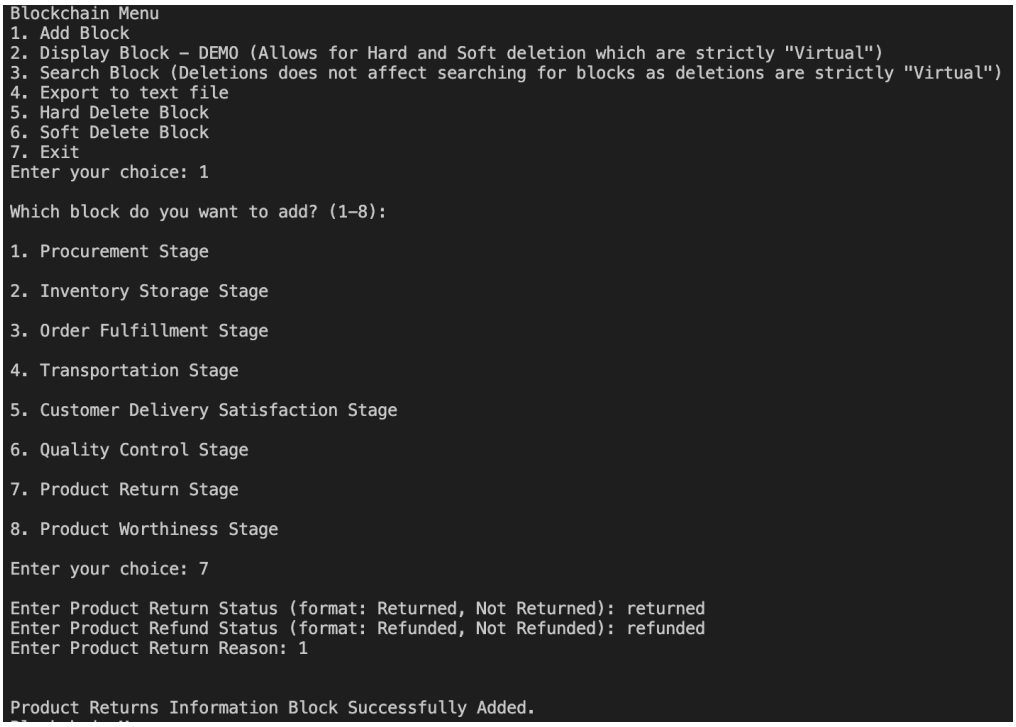
v. Add Customer Delivery Satisfaction Block (Scenario 1.5)

```
Blockchain Menu
1. Add Block
2. Display Block – DEMO (Allows for Hard and Soft deletion which are strictly "Virtual")
3. Search Block (Deletions does not affect searching for blocks as deletions are strictly "Virtual")
4. Export to text file
5. Hard Delete Block
6. Soft Delete Block
7. Exit
Enter your choice: 1

Which block do you want to add? (1–8):

1. Procurement Stage

2. Inventory Storage Stage

3. Order Fulfillment Stage

4. Transportation Stage

5. Customer Delivery Satisfaction Stage

6. Quality Control Stage

7. Product Return Stage

8. Product Worthiness Stage

Enter your choice: 5

Enter Delivery Confirmation (format: Delivered, Unsuccessful, Rescheduled): delivered
Enter Customer Feedback Collection Result (format: Excellent, Good, Average, Poor): poor
Enter Satisfaction Survey Result (format: 1–10): 9


Customer Delivery Satisfactory Information Block Successfully Added.
```

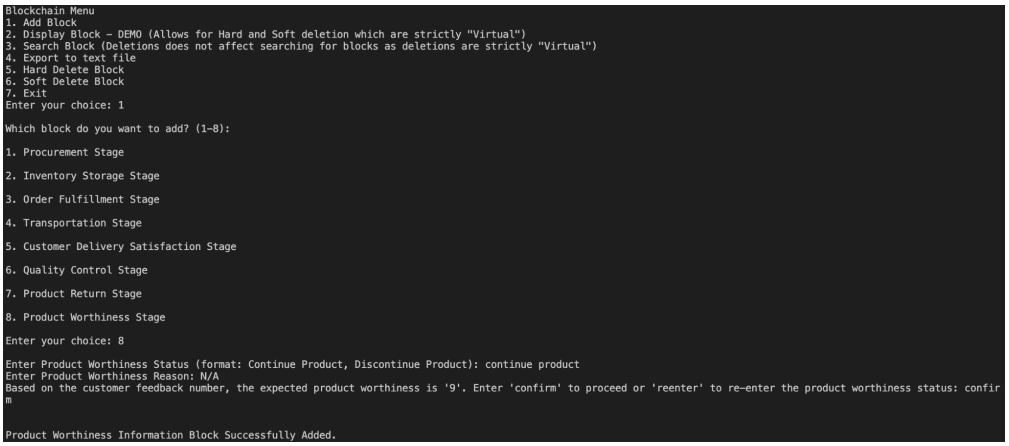
vi. Add Quality Control Block (Scenario 1.6)

```
Blockchain Menu
1. Add Block
2. Display Block – DEMO (Allows for Hard and Soft deletion which are strictly "Virtual")
3. Search Block (Deletions does not affect searching for blocks as deletions are strictly "Virtual")
4. Export to text file
5. Hard Delete Block
6. Soft Delete Block
7. Exit
Enter your choice: 1

Which block do you want to add? (1–8):

1. Procurement Stage

2. Inventory Storage Stage

3. Order Fulfillment Stage

4. Transportation Stage

5. Customer Delivery Satisfaction Stage

6. Quality Control Stage

7. Product Return Stage

8. Product Worthiness Stage

Enter your choice: 6

Enter Product Quality Inspection Result (format: Pass, Fail): fail
Enter Product Quality (format: Excellent, Good, Average, Poor): poor


Quality Inspection Control Information Block Successfully Added.
```

vii. Add Product Return Block (Scenario 1.7)



viii. Add Product Worthiness Block (Scenario 1.8)

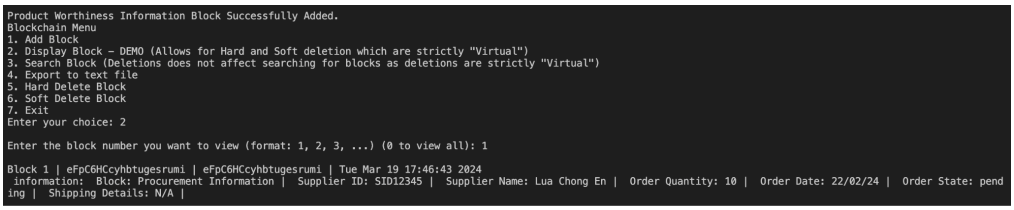


ix. Adding a block that does not exist (Error Handling) (Scenario 1.9)

```
Which block do you want to add? (1-8):

1. Procurement Stage

2. Inventory Storage Stage

3. Order Fulfillment Stage

4. Transportation Stage

5. Customer Delivery Satisfaction Stage

6. Quality Control Stage

7. Product Return Stage

8. Product Worthiness Stage

Enter your choice: 10
Invalid choice. Please enter a number between 1-8.
```

b. Scenario 2: Display Block (Error Handling included)

i. Display 1 block (Scenario 2.1)



ii. Displaying block that does not exist (Scenario 2.2)

```
Blockchain Menu
1. Add Block
2. Display Block - DEMO (Allows for Hard and Soft deletion which are strictly "Virtual")
3. Search Block (Deletions does not affect searching for blocks as deletions are strictly "Virtual")
4. Export to text file
5. Hard Delete Block
6. Soft Delete Block
7. Exit
Enter your choice: 2

Enter the block number you want to view (format: 1, 2, 3, ...) (0 to view all): 9

Block with number 9 not found.
```

iii. Display all blocks (Scenario 2.3)

```
Blockchain Menu
1. Add Block
2. Display Block - DEMO (Allows for Hard and Soft deletion which are strictly "Virtual")
3. Search Block (Deletions does not affect searching for blocks as deletions are strictly "Virtual")
4. Export to text file
5. Hard Delete Block
6. Soft Delete Block
7. Exit
Enter your choice: 2

Enter the block number you want to view (format: 1, 2, 3, ...) (0 to view all): 0

Block 8 | zWd4A6IZ90g5gMiSggxd | rjbJXkvoOBuamU9Yndqm | Tue Mar 19 18:05:48 2024
 information:  Block: Product Worthiness Information |  Product Worthiness Status: continue product |  Product Worthiness Reason: N/A |

Block 7 | rjbJXkvoOBuamU9Yndqm | 9UrkGZf99ISZBdxMezAF | Tue Mar 19 18:05:34 2024
 information:  Block: Product Returns Information |  Product Return Number: R38687 |  Product Return Status: returned |  Product Return Reason: 1 |

Block 6 | 9UrkGZf99ISZBdxMezAF | b582IK4d9NUlxbQ8C5dE | Tue Mar 19 18:05:18 2024
 information:  Block: Quality Inspection Control Information |  Quality Inspection Result: fail |  Product Quality: poor |

Block 5 | b582IK4d9NUlxbQ8C5dE | X2N2wUs6v5oSErQGftCV | Tue Mar 19 18:05:06 2024
 information:  Block: Customer Delivery Satisfactory Information |  Deilvery Confirmation: delivered |  Customer Feedback Collection: poor |  Satisfaction Survey: 9 |

Block 4 | X2N2wUs6v5oSErQGftCV | gCfw5JDJ0tI0iDCOqSs8 | Tue Mar 19 18:04:26 2024
 information:  Block: Transportation Information |  Transportation Mode: road |  Transportation Company: J&T Express |  Transportation Route: Semenyih, Selangor to Muar
, Johor |  Transportation Departure Date: 22/02/24 |  Transportation Estimated Arrival Date: 22/02/24 |

Block 3 | gCfw5JDJ0tI0iDCOqSs8 | Lb8KlQE4QrULIF9zgnEa | Tue Mar 19 18:04:05 2024
 information:  Block: Order Fulfillment Information |  Customer ID: CID12345 |  Order Quantity: 10 |  Order Date: 22/02/24 |  Order State: pending |  Shipping Details:
N/A |

Block 2 | Lb8KlQE4QrULIF9zgnEa | eFpC6HCcyhbtugesrumi | Tue Mar 19 18:03:34 2024
 information:  Block: Inventory Information |  Warehouse ID: WID12345 |  Storage Location: Semenyih, Selangor |  Inventory Quantity: 10 |  Inventory Status: low stock |


Block 1 | eFpC6HCcyhbtugesrumi | eFpC6HCcyhbtugesrumi | Tue Mar 19 17:46:43 2024
 information:  Block: Procurement Information |  Supplier ID: SID12345 |  Supplier Name: Lua Chong En |  Order Quantity: 10 |  Order Date: 22/02/24 |  Order State: pend
ing |  Shipping Details: N/A |
```

c. Scenario 3: Search Block (Error Handling included)

i. Search 1 block (Scenario 3.1)

```
Blockchain Menu
1. Add Block
2. Display Block - DEMO (Allows for Hard and Soft deletion which are strictly "Virtual")
3. Search Block (Deletions does not affect searching for blocks as deletions are strictly "Virtual")
4. Export to text file
5. Hard Delete Block
6. Soft Delete Block
7. Exit
Enter your choice: 3

Enter which block number you want to search (format: 1, 2, 3, ...) (0 to view all): 1

Block 1 | eFpC6HCcyhbtugesrumi | eFpC6HCcyhbtugesrumi | Tue Mar 19 17:46:43 2024
 information:  Block: Procurement Information |  Supplier ID: SID12345 |  Supplier Name: Lua Chong En |  Order Quantity: 10 |  Order Date: 22/02/24 |  Order State: pend
ing |  Shipping Details: N/A |
```

ii. Searching for block that does not exist (Scenario 3.2)

```
Blockchain Menu
1. Add Block
2. Display Block - DEMO (Allows for Hard and Soft deletion which are strictly "Virtual")
3. Search Block (Deletions does not affect searching for blocks as deletions are strictly "Virtual")
4. Export to text file
5. Hard Delete Block
6. Soft Delete Block
7. Exit
Enter your choice: 3

Enter which block number you want to search (format: 1, 2, 3, ...) (0 to view all): 9

Block with number 9 not found.
```

iii. Search all blocks (Scenario 3.3)

```
Blockchain Menu
1. Add Block
2. Display Block - DEMO (Allows for Hard and Soft deletion which are strictly "Virtual")
3. Search Block (Deletions does not affect searching for blocks as deletions are strictly "Virtual")
4. Export to text file
5. Hard Delete Block
6. Soft Delete Block
7. Exit
Enter your choice: 3

Enter which block number you want to search (format: 1, 2, 3, ...) (0 to view all): 0

Block 8 | zWd4A6IZ90g5gMiSggxd | rjbJXkvoOBuamU9Yndqm | Tue Mar 19 18:05:48 2024
 information:  Block: Product Worthiness Information |  Product Worthiness Status: continue product |  Product Worthiness Reason: N/A |

Block 7 | rjbJXkvoOBuamU9Yndqm | 9UrkGZf99ISZBdxMezAF | Tue Mar 19 18:05:34 2024
 information:  Block: Product Returns Information |  Product Return Number: R38687 |  Product Return Status: returned |  Product Return Reason: 1 |

Block 6 | 9UrkGZf99ISZBdxMezAF | b582IK4d9NUlxbQ8C5dE | Tue Mar 19 18:05:18 2024
 information:  Block: Quality Inspection Control Information |  Quality Inspection Result: fail |  Product Quality: poor |

Block 5 | b582IK4d9NUlxbQ8C5dE | X2N2wUs6v5oSErQGftCV | Tue Mar 19 18:05:06 2024
 information:  Block: Customer Delivery Satisfactory Information |  Deilvery Confirmation: delivered |  Customer Feedback Collection: poor |  Satisfaction Survey: 9 |

Block 4 | X2N2wUs6v5oSErQGftCV | gCfw5JDJ0tI0iDCOqSs8 | Tue Mar 19 18:04:26 2024
 information:  Block: Transportation Information |  Transportation Mode: road |  Transportation Company: J&T Express |  Transportation Route: Semenyih, Selangor to Muar
, Johor |  Transportation Departure Date: 22/02/24 |  Transportation Estimated Arrival Date: 22/02/24 |

Block 3 | gCfw5JDJ0tI0iDCOqSs8 | Lb8KlQE4QrULIF9zgnEa | Tue Mar 19 18:04:05 2024
 information:  Block: Order Fulfillment Information |  Customer ID: CID12345 |  Order Quantity: 10 |  Order Date: 22/02/24 |  Order State: pending |  Shipping Details:
N/A |

Block 2 | Lb8KlQE4QrULIF9zgnEa | eFpC6HCcyhbtugesrumi | Tue Mar 19 18:03:34 2024
 information:  Block: Inventory Information |  Warehouse ID: WID12345 |  Storage Location: Semenyih, Selangor |  Inventory Quantity: 10 |  Inventory Status: low stock |


Block 1 | eFpC6HCcyhbtugesrumi | eFpC6HCcyhbtugesrumi | Tue Mar 19 17:46:43 2024
 information:  Block: Procurement Information |  Supplier ID: SID12345 |  Supplier Name: Lua Chong En |  Order Quantity: 10 |  Order Date: 22/02/24 |  Order State: pend
ing |  Shipping Details: N/A |
```

d. Scenario 4: Export to text file

```
blockchain.txt
1   Block 8 | zWd4A6IZ90g5gMiSggxd | rjbJXkvoOBuamU9Yndqm | Tue Mar 19 18:05:48 2024
2   information:  Block: Product Worthiness Information |  Product Worthiness Status: continue product |  Product Worthiness Reason: N/A |
3   Block 7 | rjbJXkvoOBuamU9Yndqm | 9UrkGZf99ISZBdxMezAF | Tue Mar 19 18:05:34 2024
4   information:  Block: Product Returns Information |  Product Return Number: R38687 |  Product Return Status: returned |  Product Return Reason: 1 |
5   Block 6 | 9UrkGZf99ISZBdxMezAF | b582IK4d9NUlxbQ8C5dE | Tue Mar 19 18:05:18 2024
6   information:  Block: Quality Inspection Control Information |  Quality Inspection Result: fail |  Product Quality: poor |
7   Block 5 | b582IK4d9NUlxbQ8C5dE | X2N2wUs6v5oSErQGftCV | Tue Mar 19 18:05:06 2024
8   information:  Block: Customer Delivery Satisfactory Information |  Deilvery Confirmation: delivered |  Customer Feedback Collection: poor |  Satisfaction Survey: 9 |
9   Block 4 | X2N2wUs6v5oSErQGftCV | gCfw5JDJ0tI0iDCOqSs8 | Tue Mar 19 18:04:26 2024
10  information:  Block: Transportation Information |  Transportation Mode: road |  Transportation Company: J&T Express |  Transportation Route: Semenyih, Selangor to Muar, Jo
11  Block 3 | gCfw5JDJ0tI0iDCOqSs8 | Lb8KlQE4QrULIF9zgnEa | Tue Mar 19 18:04:05 2024
12  information:  Block: Order Fulfillment Information |  Customer ID: CID12345 |  Order Quantity: 10 |  Order Date: 22/02/24 |  Order State: pending |  Shipping Details: N/A
13  Block 2 | Lb8KlQE4QrULIF9zgnEa | eFpC6HCcyhbtugesrumi | Tue Mar 19 18:03:34 2024
14  information:  Block: Inventory Information |  Warehouse ID: WID12345 |  Storage Location: Semenyih, Selangor |  Inventory Quantity: 10 |  Inventory Status: low stock |
15  Block 1 | eFpC6HCcyhbtugesrumi | eFpC6HCcyhbtugesrumi | Tue Mar 19 17:46:43 2024
16  information:  Block: Procurement Information |  Supplier ID: SID12345 |  Supplier Name: Lua Chong En |  Order Quantity: 10 |  Order Date: 22/02/24 |  Order State: pending
```

e. Scenario 5: Hard Delete Block

```
Blockchain Menu
1. Add Block
2. Display Block - DEMO (Allows for Hard and Soft deletion which are strictly "Virtual")
3. Search Block (Deletions does not affect searching for blocks as deletions are strictly "Virtual")
4. Export to text file
5. Hard Delete Block
6. Soft Delete Block
7. Exit
Enter your choice: 5

Enter which block number you want to hard delete: 1
Block with block number 1 has been hard deleted.
```

i. Hard Deleting Block that has been Soft Deleted (Scenario 5.1)

```
Blockchain Menu
1. Add Block
2. Display Block - DEMO (Allows for Hard and Soft deletion which are strictly "Virtual")
3. Search Block (Deletions does not affect searching for blocks as deletions are strictly "Virtual")
4. Export to text file
5. Hard Delete Block
6. Soft Delete Block
7. Exit
Enter your choice: 5

Enter which block number you want to hard delete: 0
Block with block number 0 has been soft deleted and cannot be hard deleted.
```

ii. Hard Deleting Block that has already been Hard Deleted (Scenario 5.2)

```
Blockchain Menu
1. Add Block
2. Display Block - DEMO (Allows for Hard and Soft deletion which are strictly "Virtual")
3. Search Block (Deletions does not affect searching for blocks as deletions are strictly "Virtual")
4. Export to text file
5. Hard Delete Block
6. Soft Delete Block
7. Exit
Enter your choice: 5

Enter which block number you want to hard delete: 1
Block with block number 1 has already been hard deleted.
```

f. Scenario 6: Soft Delete Block

```
Blockchain Menu
1. Add Block
2. Display Block - DEMO (Allows for Hard and Soft deletion which are strictly "Virtual")
3. Search Block (Deletions does not affect searching for blocks as deletions are strictly "Virtual")
4. Export to text file
5. Hard Delete Block
6. Soft Delete Block
7. Exit
Enter your choice: 6

Enter which block number you want to soft delete: 2
Information in block with block number 2 has been soft deleted.
```

i. Soft Deleting Block that has been Hard Deleted (Scenario 6.1)

```
Blockchain Menu
1. Add Block
2. Display Block - DEMO (Allows for Hard and Soft deletion which are strictly "Virtual")
3. Search Block (Deletions does not affect searching for blocks as deletions are strictly "Virtual")
4. Export to text file
5. Hard Delete Block
6. Soft Delete Block
7. Exit
Enter your choice: 6

Enter which block number you want to soft delete: 1
Block with block number 1 has been hard deleted and cannot be soft deleted.
```

ii. Soft Deleting Block that has already been Soft Deleted (Scenario 6.2)

```
Blockchain Menu
1. Add Block
2. Display Block – DEMO (Allows for Hard and Soft deletion which are strictly "Virtual")
3. Search Block (Deletions does not affect searching for blocks as deletions are strictly "Virtual")
4. Export to text file
5. Hard Delete Block
6. Soft Delete Block
7. Exit
Enter your choice: 6

Enter which block number you want to soft delete: 0
Block with block number 0 has already been soft deleted.
```

g. Scenario 7: Exit Program

```
Blockchain Menu
1. Add Block
2. Display Block – DEMO (Allows for Hard and Soft deletion which are strictly "Virtual")
3. Search Block (Deletions does not affect searching for blocks as deletions are strictly "Virtual")
4. Export to text file
5. Hard Delete Block
6. Soft Delete Block
7. Exit
Enter your choice: 7

Exit Program.
```

h. Scenario 8: Incorrect ID, Error Handling

```
Blockchain Menu
1. Add Block
2. Display Block – DEMO (Allows for Hard and Soft deletion which are strictly "Virtual")
3. Search Block (Deletions does not affect searching for blocks as deletions are strictly "Virtual")
4. Export to text file
5. Hard Delete Block
6. Soft Delete Block
7. Exit
Enter your choice: 1

Which block do you want to add? (1–8):

1. Procurement Stage

2. Inventory Storage Stage

3. Order Fulfillment Stage

4. Transportation Stage

5. Customer Delivery Satisfaction Stage

6. Quality Control Stage

7. Product Return Stage

8. Product Worthiness Stage

Enter your choice: 1

Enter Supplier ID (format: SIDxxxxx): SID12
Invalid format. Please enter a valid Supplier ID.
```

i. Scenario 9: Incorrect Date, Error Handling

```
Enter your choice: 4

Enter Transportation Mode (Road, Rail, Sea, Air): air
Enter Transportation Company: J&T Express
Enter Transportation Route:
From (format: city, state): Petaling Jaya, Selangor
To (format: city, state): Muar, Johor
Enter Transportation Departure Date (format: dd/mm/yy): 22/13/24
Invalid format. Please enter a valid Order Date (format: dd/mm/yy).
Enter Transportation Departure Date (format: dd/mm/yy):
```

j. Scenario 10: Incorrect Destination, Error Handling

```
Enter Warehouse ID (format: WIDxxxxx): WID12345
Enter Storage Location (format: city, state): Petaling Jaya, Johor
Invalid format. Please enter a valid location in the format 'city, state'.
Enter Storage Location (format: city, state):
```

k. Scenario 11: Incorrect Username & Password, Error Handling

```
Inventory and Transportation Management System.

Name: Lua Chong En
Student ID: 20417309


Enter Username: E


Enter Password: 123

Invalid username or password. Please try again.
```

l. Scenario 12: Incorrect Input (general), Error Handling

```
Blockchain Menu
1. Add Block
2. Display Block - DEMO (Allows for Hard and Soft deletion which are strictly "Virtual")
3. Search Block (Deletions does not affect searching for blocks as deletions are strictly "Virtual")
4. Export to text file
5. Hard Delete Block
6. Soft Delete Block
7. Exit
Enter your choice: 8

Invalid choice. Please enter a valid choice.
```

```
Which block do you want to add? (1-8):

1. Procurement Stage

2. Inventory Storage Stage

3. Order Fulfillment Stage

4. Transportation Stage

5. Customer Delivery Satisfaction Stage

6. Quality Control Stage

7. Product Return Stage

8. Product Worthiness Stage

Enter your choice: 10
Invalid choice. Please enter a number between 1-8.
```

```
Blockchain Menu
1. Add Block
2. Display Block - DEMO (Allows for Hard and Soft deletion which are strictly "Virtual")
3. Search Block (Deletions does not affect searching for blocks as deletions are strictly "Virtual")
4. Export to text file
5. Hard Delete Block
6. Soft Delete Block
7. Exit
Enter your choice: 5

Enter which block number you want to hard delete: 11
Block with block number 11 not found.
```

m. Scenario 13: Adding more than 8 blocks, Error Handling

```
Product Worthiness Information Block Successfully Added.
Blockchain Menu
1. Add Block
2. Display Block - DEMO (Allows for Hard and Soft deletion which are strictly "Virtual")
3. Search Block (Deletions does not affect searching for blocks as deletions are strictly "Virtual")
4. Export to text file
5. Hard Delete Block
6. Soft Delete Block
7. Exit
Enter your choice: 1

All blocks in the blockchain have been added. No more blocks can be added.
```

n. Scenario 14: Adding the same block more than once, Error Handling

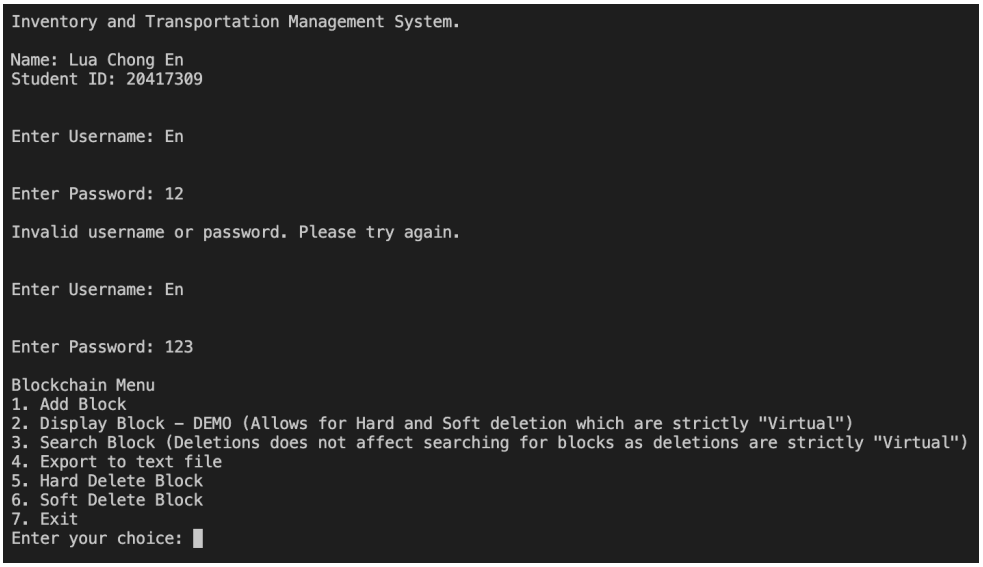
```
Which block do you want to add? (1-8):

1. Procurement Stage

2. Inventory Storage Stage

3. Order Fulfillment Stage

4. Transportation Stage

5. Customer Delivery Satisfaction Stage

6. Quality Control Stage

7. Product Return Stage

8. Product Worthiness Stage

Enter your choice: 8
Product Worthiness Information has already been added.
```

o. Scenario 15: Hard or Soft deleting a block that has already been deleted

```
Blockchain Menu
1. Add Block
2. Display Block - DEMO (Allows for Hard and Soft deletion which are strictly "Virtual")
3. Search Block (Deletions does not affect searching for blocks as deletions are strictly "Virtual")
4. Export to text file
5. Hard Delete Block
6. Soft Delete Block
7. Exit
Enter your choice: 5

Enter which block number you want to hard delete: 1
Block with block number 1 has already been hard deleted.
```

```
Blockchain Menu
1. Add Block
2. Display Block - DEMO (Allows for Hard and Soft deletion which are strictly "Virtual")
3. Search Block (Deletions does not affect searching for blocks as deletions are strictly "Virtual")
4. Export to text file
5. Hard Delete Block
6. Soft Delete Block
7. Exit
Enter your choice: 6

Enter which block number you want to soft delete: 1
Block with block number 1 has been hard deleted and cannot be soft deleted.
```

p. Scenario 16: User authentication (Error Handling included)



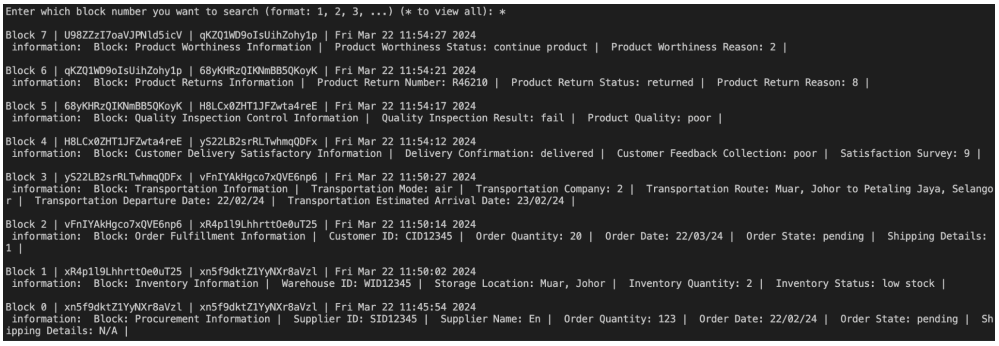
q. Scenario 17: Export to “blockchain.txt”

r. Scenario 18: “blockchain.txt” file (example)

Block 7 | APrP7522ZwCYV0gMSAHJ | 6qiF8tW40389ouUX4Qve | Thu Mar 21 15:18:06 2024
information: Block: Product Worthiness Information | Product Worthiness Status: continue product | Product Worthiness Reason: 2 |
Block 6 | 6qiF8tW40389ouUX4Qve | MPRzOoTKk5uCHO0pxG2y | Thu Mar 21 15:18:00 2024
information: Block: Product Returns Information | Product Return Number: R99310 | Product Return Status: returned | Product Return Reason: 2 |
Block 5 | MPRzOoTKk5uCHO0pxG2y | ipHqPLNNkD0NaiFAJWR2 | Thu Mar 21 15:17:56 2024
information: Block: Quality Inspection Control Information | Quality Inspection Result: fail | Product Quality: poor |
Block 4 | ipHqPLNNkD0NaiFAJWR2 | kBo7RwBRlI396hXX7st1 | Thu Mar 21 15:17:50 2024

information: Block: Customer Delivery Satisfactory Information | Delivery Confirmation: delivered | Customer Feedback Collection: poor | Satisfaction Survey: 9 |
Block 3 | kBo7RwBRIl396hXX7st1 | wAOSzOszFhVMS5IAMyPQ | Thu Mar 21 15:17:38 2024
information: Block: Transportation Information | Transportation Mode: air | Transportation Company: 2 | Transportation Route: Muar, Johor to Muar, Johor | Transportation Departure Date: 22/02/24 | Transportation Estimated Arrival Date: 23/02/24 |
Block 2 | wAOSzOszFhVMS5IAMyPQ | XMiHLpQMv51Pq9DDQ1MU | Thu Mar 21 15:17:28 2024
information: Block: Order Fulfillment Information | Customer ID: CID12345 | Order Quantity: 2 | Order Date: 22/02/24 | Order State: pending | Shipping Details: 1 |
Block 1 | XMiHLpQMv51Pq9DDQ1MU | ZYZpNaFQva4bm8WZexoT | Thu Mar 21 15:17:21 2024
information: Block: Inventory Information | Warehouse ID: WID12345 | Storage Location: Muar, Johor | Inventory Quantity: 2 | Inventory Status: low stock |
Block 0 | ZYZpNaFQva4bm8WZexoT | ZYZpNaFQva4bm8WZexoT | Thu Mar 21 15:17:09 2024
information: Block: Procurement Information | Supplier ID: SID12345 | Supplier Name: 2 | Order Quantity: 2 | Order Date: 22/02/24 | Order State: pending | Shipping Details: 2 |

s. Scenario 19: Displaying/Searching all blocks



t. Scenario 20: Entering non-numerical value for numerical input (Error Handling)

```
Enter Supplier ID (format: SIDxxxxx): 2
Invalid format. Please enter a valid Supplier ID.

Enter Supplier ID (format: SIDxxxxx): SID12345
Enter Supplier Name: w
Enter Order Quantity: oo
Invalid input. Please enter a valid number for Order Quantity.
Enter Order Quantity:
```

13. Conclusion

In conclusion, online shopping websites such as Shopee and Lazada can gain further success by adopting blockchain-based inventory and transportation management systems. Transparency and reliability of the data is guaranteed throughout the product life cycle. From procurement to delivery and customer satisfaction surveys. Because the blockchain is adaptable and modular, it can be customized and adapted to meet the company requirements, encouraging user input and reliability in data management. Additionally, the supply chain's integrity is ensured and risk is lowered through the automated and safe transactions made possible by the integration of conditional statements/ smart contracts.

14. Feedback Questions

a. Things that I've learned doing this coursework

Throughout doing this coursework I've learnt a lot on using CPP and also navigating blockchain and its multitude of features.

Focusing first on CPP, I've learned how to use the basic syntax to implement more advanced concepts. Using linked lists, vectors, and pointers are just a few to mention. I believe I have gained a strong understanding in using the CPP language.

Whilst on the other hand, I've really been able to understand the workings of blockchain and the nature of it. Seeing how secure and decentralized it is through

smart contracts and cryptographic methods. Using specified techniques such as generating hash through a custom hashing function.

I believe I have created a very impressive piece of work, I am extremely proud of myself. I hope my hard work in this coursework can be recognized.

b. Noteworthy difficulties that I faced

One of the more significant difficulties that I faced was using vectors to store information key-value pairs. I had trouble initializing it and also using it as a parameter in the public methods under the “Block” class. I did ample research and through rounds of trial and error, I was able to slowly solve the issue.

Another major problem I faced was the hash value and ensuring that the hash values remain the same for the first block in the blockchain. Additionally using the same hash function for the following block and generating a new one. Again, through logical thinking and trial and error, I was able to slowly resolve the imminent issue and allow the blockchain to smoothly link to each other.

Finally, learning how to build a blockchain in CPP was a big difficult project itself. This coursework challenged my abilities in programming in CPP. However, I would like to extend my thanks to Dr. Doreen for providing assistance throughout and providing feedback when I required it. The lecture and lab sessions were useful.

15. Appendix

a. code

```
#include <iostream>
#include <vector>
#include <string>
#include <ctime>
#include <cstdlib>
#include <fstream>
#include <sstream>

using namespace std;

struct Block { // Block structure
```

```
    int blockNumber; // Block number
    string currentHashNumber; // Current hash number
    string previousHashNumber; // Previous hash number
    string currentTimeStamp; // Current time stamp
    vector<pair<string, string> > information;  // Information stored in the block
    bool isHardDeleted; // Flag to indicate if block is deleted
    bool isSoftDeleted; // Flag to indicate if block is deleted

    Block(int blockN, string currentH, string previousH, string timeStamp) { //
Constructor for Block
        blockNumber = blockN; // Set block number
        currentHashNumber = currentH; // Set current hash number
        previousHashNumber = previousH; // Set previous hash number
        currentTimeStamp = timeStamp; // Set current time stamp
        isHardDeleted = false; // Set isHardDeleted flag to false
        isSoftDeleted = false; // Set isSoftDeleted flag to false
    }
};

struct BlockNode { // BlockNode structure
    Block data; // Block data
    BlockNode* next; // Pointer to the next block

    BlockNode(Block block) : data(block) { // Constructor for BlockNode
        next = nullptr; // Set next to nullptr
    }
};

class Blockchain { // Blockchain class
private: // Private members
    BlockNode* head; // Pointer to the head of the blockchain
    int currentBlockNumber; // Current block number
    string hashNumber; // Hash number

    bool procurementAdded; // Procurement information added flag
    bool inventoryAdded; // Inventory information added flag
    bool orderFulfillmentAdded; // Order fulfillment information added flag
    bool transportationAdded; // Transportation information added flag
    bool customerDeliverySatisfactionAdded; // Customer delivery satisfaction
information added flag
    bool qualityControlAdded; // Quality control information added flag
    bool productReturn; // Product return information added flag
    bool productWorthiness; // Product worthiness information added flag
```

```
public: // Public members
   Blockchain() {  // Constructor for Blockchain
       head = nullptr; // Set head to nullptr, which is the start of the blockchain
       currentBlockNumber = 0; // Set current block number to 1
       hashNumber = generateRandomHash(); // Generate a random hash number
       time_t timeNow = time(0); // Get the current time
       char* dateTime = ctime(&timeNow); // Convert the current time to a string so
it can be stored in the block
       Block firstBlock(currentBlockNumber, hashNumber, hashNumber,
dateTime); // Create the first block
       head = new BlockNode(firstBlock); // Set the head of the blockchain to the
first block

       procurementAdded = false; // Set procurement information added flag to
false
       inventoryAdded = false; // Set inventory information added flag to false
       orderFulfillmentAdded = false; // Set order fulfillment information added flag
to false
       transportationAdded = false; // Set transportation information added flag to
false
       customerDeliverySatisfactionAdded = false; // Set customer delivery
satisfaction information added flag to false
       qualityControlAdded = false; // Set quality control information added flag to
false
       productReturn = false; // Set product return information added flag to false
       productWorthiness = false; // Set product worthiness information added flag
to false
   }

   void addBlock(const vector<pair<string, string> >& info) { // Add a block to the
blockchain with the information passed in, information is a vector pair of strings
       hashNumber = generateRandomHash(); // Generate a random hash number
for the new block to be added
       time_t timeNow = time(0); // Get the current time for the new block to be
added
       char* dateTime = ctime(&timeNow); // Convert the current time to a string so
it can be stored in the block because the time is stored as a string in the block
       int chosenBlockNumber; // Chosen block number by the user to add to the
blockchain

       Block newBlock(currentBlockNumber, hashNumber,
head->data.currentHashNumber, dateTime); // Create a new block with the
current block number, the hash number, the previous hash number, and the
current time
```

```
        newBlock.information = info; // Set the information of the new block to the information passed in as a parameter

        do { // Loop until the user chooses to add all the blocks they want to add, only one of each block can be added
            cout << "\nWhich block do you want to add? (1-8):" << endl; // Ask the user which block they want to add
            cout << "\n1. Procurement Stage" << endl; // Add Procurement Stage
            cout << "\n2. Inventory Storage Stage" << endl; // Add Inventory Storage Stage
            cout << "\n3. Order Fulfillment Stage" << endl; // Add Order Fulfillment Stage
            cout << "\n4. Transportation Stage" << endl; // Add Transportation Stage
            cout << "\n5. Customer Delivery Satisfaction Stage" << endl; // Add Customer Delivery Satisfaction Stage
            cout << "\n6. Quality Control Stage" << endl; // Add Quality Control Stage
            cout << "\n7. Product Return Stage" << endl; // Add Product Return Stage
            cout << "\n8. Product Worthiness Stage" << endl; // Add Product Worthiness Stage
            cout << "\nEnter your choice: "; // Ask the user to enter their choice
            cin >> chosenBlockNumber; // Get the user's choice
            cin.ignore(); // Ignore the newline character

            if (cin.fail()) { // If the user enters an invalid input
                cin.clear(); // Clear the input buffer
                cin.ignore(numeric_limits<streamsize>::max(), '\n'); // Ignore the rest of the input
                cout << "\nInvalid input. Please enter a number." << endl; // Tell the user that they entered an invalid input
                continue; // Continue to the next iteration of the loop
            }

            switch (chosenBlockNumber) { // Switch statement based on the user's choice
                case 1: // If the user chooses 1
                    if(procurementAdded) { // If procurement information has already been added
                        cout << "Procurement Information has already been added." << endl; // Tell the user that procurement information has already been added
                        chosenBlockNumber = 0; // Set chosen block number to 0 because the user has already added procurement information so the switch statement will break
                        break; // Break out of the switch statement
                    }
```

```
                addProcurementInformation(newBlock); // Add procurement
information to the new block
                procurementAdded = true; // Set procurement information added flag
to true
                break; // Break out of the switch statement
            case 2:  // If the user chooses 2
                if(inventoryAdded) { // If inventory information has already been
added
                    cout << "Inventory Information has already been added." << endl;
// Tell the user that inventory information has already been added
                    chosenBlockNumber = 0; // Set chosen block number to 0
because the user has already added inventory information so the switch
statement will break
                    break; // Break out of the switch statement
                }
                addInventoryInformation(newBlock); // Add inventory information to
the new block
                inventoryAdded = true; // Set inventory information added flag to true
                break; // Break out of the switch statement
            case 3: // If the user chooses 3
                if(orderFulfillmentAdded) { // If order fulfillment information has
already been added
                    cout << "Order Fulfillment Information has already been added."
<< endl; // Tell the user that order fulfillment information has already been added
                    chosenBlockNumber = 0; // Set chosen block number to 0
because the user has already added order fulfillment information so the switch
statement will break
                    break; // Break out of the switch statement
                }
                addOrderFulfillmentInformation(newBlock); // Add order fulfillment
information to the new block
                orderFulfillmentAdded = true; // Set order fulfillment information
added flag to true
                break; // Break out of the switch statement
            case 4: // If the user chooses 4
                if(transportationAdded) { // If transportation information has already
been added
                    cout << "Transportation Information has already been added." <<
endl; // Tell the user that transportation information has already been added
                    chosenBlockNumber = 0; // Set chosen block number to 0
because the user has already added transportation information so the switch
statement will break
                    break; // Break out of the switch statement
                }
```

```
                addTransportationInformation(newBlock); // Add transportation
information to the new block
                transportationAdded = true; // Set transportation information added
flag to true
                break; // Break out of the switch statement
            case 5: // If the user chooses 5
                if(customerDeliverySatisfactionAdded) { // If customer delivery
satisfaction information has already been added
                    cout << "Customer Delivery Satisfaction Information has already
been added." << endl; // Tell the user that customer delivery satisfaction
information has already been added
                    chosenBlockNumber = 0; // Set chosen block number to 0
because the user has already added customer delivery satisfaction information
so the switch statement will break
                    break; // Break out of the switch statement
                }
                addCustomerDeliverySatisfactionInformation(newBlock); // Add
customer delivery satisfaction information to the new block
                customerDeliverySatisfactionAdded = true; // Set customer delivery
satisfaction information added flag to true
                break; // Break out of the switch statement
            case 6: // If the user chooses 6
                if(qualityControlAdded) { // If quality control information has already
been added
                    cout << "Quality Control Information has already been added." <<
endl; // Tell the user that quality control information has already been added
                    chosenBlockNumber = 0; // Set chosen block number to 0
because the user has already added quality control information so the switch
statement will break
                    break; // Break out of the switch statement
                }
                addQualityInspectionControlInformation(newBlock); // Add quality
control information to the new block
                qualityControlAdded = true; // Set quality control information added
flag to true
                break; // Break out of the switch statement
            case 7: // If the user chooses 7
                if(productReturn) { // If product return information has already been
added
                    cout << "Product Return Information has already been added." <<
endl; // Tell the user that product return information has already been added
                    chosenBlockNumber = 0; // Set chosen block number to 0
because the user has already added product return information so the switch
statement will break
```

```
                    break; // Break out of the switch statement
                } // Add product return information to the new block
                addProductReturnInformation(newBlock); // Add product return
information to the new block
                productReturn = true; // Set product return information added flag to
true
                break; // Break out of the switch statement
            case 8: // If the user chooses 8
                if(productWorthiness) { // If product worthiness information has
already been added
                    cout << "Product Worthiness Information has already been
added." << endl; // Tell the user that product worthiness information has already
been added
                    chosenBlockNumber = 0; // Set chosen block number to 0
because the user has already added product worthiness information so the
switch statement will break
                    break; // Break out of the switch statement
                }
                addProductWorthinessInformation(newBlock); // Add product
worthiness information to the new block
                productWorthiness = true; // Set product worthiness information
added flag to true
                break; // Break out of the switch statement
            default:
                cout << "Invalid choice. Please enter a number between 1-8." <<
endl; // Tell the user that their choice is invalid
                break; // Break out of the switch statement
        }
    } while (chosenBlockNumber < 1 || chosenBlockNumber > 8); // Keep asking
the user to enter a number between 1-8 until they do so

    if (currentBlockNumber == 0) { // If the current block number is 1
        head->data.information = newBlock.information;  // Set the head's data to
the new block's information
    } else { // If the current block number is not 1
        BlockNode* newNode = new BlockNode(newBlock); // Create a new
block node with the new block's information
        newNode->next = head; // Set the new node's next to the head
        head = newNode; // Set the head to the new node
    }

    currentBlockNumber++; // Increment the current block number
}
```

```
    void addProcurementInformation(Block& block) { // Add procurement
information to the block, called when the user chooses 1
        string supplierID, supplierName, orderQuantity, orderDate, orderState,
shippingDetails; // Declare variables for the supplier ID, supplier name, order
quantity, order date, order state, and shipping details
        bool correctSupplierID = false; // Set the correct supplier ID flag to false

        while (!correctSupplierID) { // While loop to keep asking the user to enter a
valid supplier ID
            cout << "\nEnter Supplier ID (format: SIDxxxxx): "; // Ask the user to enter
the supplier ID, the format is string SID followed by 5 digits
            getline(cin, supplierID); // Get user input for the supplier ID

            if (supplierID.length() == 8 && supplierID.substr(0, 3) == "SID") { // Check
if the supplier ID is 8 characters long and starts with "SID"
                bool validLength = true; // Set the valid flag to true
                for (int i = 3; i < 8; i++) { // For loop to check if the rest of the characters
are digits
                    if (!isdigit(supplierID[i])) { // If the character is not a digit
                        validLength = false; // Then set the valid flag to false
                        break; // Break out of the for loop
                    }
                }
                if (validLength) { // If the valid flag is true
                    correctSupplierID = true; // Set the correct supplier ID flag to true,
this means that the supplier ID is valid
                }
            }
            else { // If the supplier ID is not 8 characters long and does not start with
"SID"
                cout << "Invalid format. Please enter a valid Supplier ID." << endl; //
Tell the user that the supplier ID is invalid
            }
        }

        cout << "Enter Supplier Name: "; // Ask the user to enter the supplier name
        getline(cin, supplierName); // Get user input for the supplier name

        bool validOrderQuantity = false; // Set the valid order quantity flag to false
        while (!validOrderQuantity) { // While loop to keep asking the user to enter a
valid order quantity
            cout << "Enter Order Quantity: "; // Ask the user to enter the order
quantity
            getline(cin, orderQuantity); // Get user input for the order quantity
```

```
        if (orderQuantity.find_first_not_of("0123456789") == string::npos) { // Check if the order quantity contains only digits, no other characters but digits
            validOrderQuantity = true; // Set the valid order quantity flag to true, this means that the order quantity is valid
        } else { // If the order quantity contains characters other than digits
            cout << "Invalid input. Please enter a valid number for Order Quantity." << endl; // Tell the user that the order quantity is invalid
        }
    }

    bool correctOrderDate = false; // Set the correct order date flag to false
    while (!correctOrderDate) { // While loop to keep asking the user to enter a valid order date
        cout << "Enter Order Date (format: dd/mm/yy): "; // Ask the user to enter the order date, the format is strictly day/month/year
        getline(cin, orderDate); // Get user input for the order date

        if (orderDate.length() == 8 && orderDate[2] == '/' && orderDate[5] == '/') { // Check if the order date is 8 characters long and contains "/" at the 3rd and 6th position, no other characters but "/"
            string dayString = orderDate.substr(0, 2); // Get the day from the order date
            string monthString = orderDate.substr(3, 2); // Get the month from the order date
            string yearString = orderDate.substr(6, 2); // Get the year from the order date
            int day = stoi(dayString); // Convert the day string to an integer
            int month = stoi(monthString); // Convert the month string to an integer
            int year = stoi(yearString); // Convert the year string to an integer

            if (day >= 1 && day <= 31 && month >= 1 && month <= 12 && year >= 0 && year >= 24) { // Check that the day is between 1-31, month is between 1-12, and year is after 2024, only valid day month and year
                correctOrderDate = true; // Set the correct order date flag to true, this means that the order date is valid
            }
        }

        if (!correctOrderDate) { // If the order date is not valid
            cout << "Invalid format. Please enter a valid Order Date (format: dd/mm/yy) Ensure that year is on or after 2024." << endl; // Tell the user that the order date is invalid, will loop again
        }
```

```
    }

    bool validOrderState = false; // Set the valid order state flag to false

    while (!validOrderState) { // While loop to keep asking the user to enter a valid order state
        cout << "Enter Order State (Pending, In Progress, Completed, Cancelled): "; // Ask the user to enter the order state, the options are Pending, In Progress, Completed, Cancelled
        getline(cin, orderState); // Get user input for the order state

        transform(orderState.begin(), orderState.end(), orderState.begin(), ::tolower); // Convert the order state to lowercase, so that the user can enter the order state in any case, doesnt affect the input

        if (orderState == "pending" || orderState == "in progress" || orderState == "completed" || orderState == "cancelled") { // Check if the order state is one of the valid options, pending, in progress, completed, or cancelled
            validOrderState = true; // Set the valid order state flag to true, this means that the order state is valid
        }
        else { // If the order state is not one of the valid options
            cout << "Invalid status. Please enter 'Pending', 'In Progress', 'Completed', 'Cancelled'." << endl; // Tell the user that the order state is invalid
        }
    }

    cout << "Enter Shipping Details: "; // Ask the user to enter the shipping details
    getline(cin, shippingDetails); // Get user input for the shipping details

    block.information.push_back(make_pair("Block", "Procurement Information")); // Add the block name and block type to the block information
    block.information.push_back(make_pair("Supplier ID", supplierID)); // Add the supplier ID to the block information
    block.information.push_back(make_pair("Supplier Name", supplierName)); // Add the supplier name to the block information
    block.information.push_back(make_pair("Order Quantity", orderQuantity)); // Add the order quantity to the block information
    block.information.push_back(make_pair("Order Date", orderDate)); // Add the order date to the block information
    block.information.push_back(make_pair("Order State", orderState)); // Add the order state to the block information
```

```
        block.information.push_back(make_pair("Shipping Details",
shippingDetails)); // Add the shipping details to the block information

        cout << "\n\nProcurement Information Block Successfully Added.\n"; // Tell
the user that the procurement information block has been successfully added,
and the information has been stored in the block, block now has the information
    }

    void addInventoryInformation(Block& block) { // Function to add inventory
information to the block
        string warehouseID, storageLocation, inventoryQuantity, inventoryStatus; //
Declare the variables for the warehouse ID, storage location, inventory quantity,
and inventory status
        bool correctWarehouseID = false; // Set the correct warehouse ID flag to
false

        while (!correctWarehouseID) { // While loop to keep asking the user to enter
a valid warehouse ID
            cout << "\nEnter Warehouse ID (format: WIDxxxxx): "; // Ask the user to
enter the warehouse ID, the format is strictly WID followed by 5 digits
            getline(cin, warehouseID); // Get user input for the warehouse ID

            if (warehouseID.length() == 8 && warehouseID.substr(0, 3) == "WID") { //
Check if the warehouse ID is 8 characters long and starts with WID, no other
characters but WID and 5 digits
                bool validLength = true; // Set the valid flag to true, this means that the
warehouse ID is valid
                for (int i = 3; i < 8; i++) { // For loop to check the 5 digits of the
warehouse ID
                    if (!isdigit(warehouseID[i])) { // If the character is not a digit
                        validLength = false; // Set the valid flag to false, this means that
the warehouse ID is invalid
                        break; // Break the for loop, no need to check the rest of the digits
                    }
                }
                if (validLength) { // If the warehouse ID is valid
                    correctWarehouseID = true; // Set the correct warehouse ID flag to
true, this means that the warehouse ID is valid
                }
            }
            else { // If the warehouse ID is not valid
                cout << "Invalid format. Please enter a valid Warehouse ID." << endl; //
Tell the user that the warehouse ID is invalid, will loop again
            }
```

```
    }

    vector<string> validLocations; // Create a vector of strings to store the valid locations, valid locations are cities and states, the format is: city, state
    ifstream file("valid_locations.txt"); // Open the valid locations file, file name is valid_locations.txt
    if (file.is_open()) { // If the file is open
        string line; // Declare a string to store the line of the file
        while (getline(file, line)) { // While loop to read the file line by line
            validLocations.push_back(line); // Add the line to the valid locations vector
        }
        file.close(); // Close the file
    } else { // If the file is not open
        cout << "Error: Unable to open valid locations file." << endl; // Tell the user that the file is not open, file is not found
        return; // Return from the function
    }

    bool validStorageLocation = false; // Set the valid storage location flag to false

    while (!validStorageLocation) { // While loop to keep asking the user to enter a valid storage location
        cout << "Enter Storage Location (format: city, state): "; // Ask the user to enter the storage location, the format is city, state
        getline(cin, storageLocation); // Get user input for the storage location

        validStorageLocation = isValidLocation(storageLocation, validLocations); // Call the isValidLocation function to check if the storage location is valid, pass the storage location and the valid locations vector as arguments

        if (!validStorageLocation) { // If the storage location is not valid
            cout << "Invalid format. Please enter a valid location in the format 'city, state'." << endl; // Tell the user that the storage location is invalid, will loop again
        }
    }

    bool validInventoryQuantity = false; // Set the valid inventory quantity flag to false
    while (!validInventoryQuantity) { // While loop to keep asking the user to enter a valid inventory quantity
        cout << "Enter Inventory Quantity: "; // Ask the user to enter the inventory quantity
```

```
        getline(cin, inventoryQuantity); // Get user input for the inventory quantity

        if (inventoryQuantity.find_first_not_of("0123456789") == string::npos) { // Check if the inventory quantity contains only digits, no other characters
            validInventoryQuantity = true; // Set the valid inventory quantity flag to true, this means that the inventory quantity is valid
        } else {
            cout << "Invalid input. Please enter a valid number for Inventory Quantity." << endl; // Tell the user that the inventory quantity is invalid, will loop again
        }
    }

    bool validInventoryStatus = false; // Set the valid inventory status flag to false
    while (!validInventoryStatus) { // While loop to keep asking the user to enter a valid inventory status
        cout << "Enter Inventory Status (Available, Low Stock, or Not Available): "; // Ask the user to enter the inventory status, the options are available, low stock, or not available
        getline(cin, inventoryStatus); // Get user input for the inventory status

        transform(inventoryStatus.begin(), inventoryStatus.end(), inventoryStatus.begin(), ::tolower); // Convert the inventory status to lowercase

        if (inventoryStatus == "available" || inventoryStatus == "low stock" || inventoryStatus == "not available") { // Check if the inventory status is available, low stock, or not available
            validInventoryStatus = true; // Set the valid inventory status flag to true, this means that the inventory status is valid
        }
        else { // If the inventory status is not valid
            cout << "Invalid status. Please enter 'Available', 'Low Stock', or 'Not Available'." << endl; // Tell the user that the inventory status is invalid, will loop again
        }
    }

    block.information.push_back(make_pair("Block", "Inventory Information")); // Add the inventory information block to the block
    block.information.push_back(make_pair("Warehouse ID", warehouseID)); // Add the warehouse ID to the block
    block.information.push_back(make_pair("Storage Location", storageLocation)); // Add the storage location to the block
```

```
        block.information.push_back(make_pair("Inventory Quantity",
inventoryQuantity)); // Add the inventory quantity to the block
        block.information.push_back(make_pair("Inventory Status",
inventoryStatus)); // Add the inventory status to the block

        cout << "\n\nInventory Information Block Successfully Added.\n"; // Tell the
user that the inventory information block is successfully added
    }

    void addOrderFulfillmentInformation(Block& block) { // Function to add order
fulfillment information to the block
        string customerID, orderQuantityF, orderDateF, orderStateF,
shippingDetailsF; // Declare strings to store the customer ID, order quantity, order
date, order state, and shipping details

        bool correctCustomerID = false; // Set the correct customer ID flag to false

        while (!correctCustomerID) { // While loop to keep asking the user to enter a
valid customer ID
            cout << "\nEnter Customer ID (format: CIDxxxxx): "; // Ask the user to
enter the customer ID, the format is CID followed by 5 digits
            getline(cin, customerID); // Get user input for the customer ID

            if (customerID.length() == 8 && customerID.substr(0, 3) == "CID") { //
Check if the customer ID is 8 characters long and starts with CID
                bool validLength = true; // Set the valid flag to true
                for (int i = 3; i < 8; i++) { // For loop to check if the rest of the customer
ID contains only digits
                    if (!isdigit(customerID[i])) { // If the character is not a digit
                        validLength = false; // Set the valid flag to false
                        break; // Break from the for loop
                    }
                }
                if (validLength) { // If the customer ID is valid
                    correctCustomerID = true; // Set the correct customer ID flag to true,
this means that the customer ID is valid
                }
            }
            else { // If the customer ID is not valid
                cout << "Invalid format. Please enter a valid Customer ID." << endl; //
Tell the user that the customer ID is invalid, will loop again
            }
        }
```

```
        bool validOrderQuantityF = false; // Set the valid order quantity flag to false
        while (!validOrderQuantityF) { // While loop to keep asking the user to enter a valid order quantity
            cout << "Enter Order Quantity: "; // Ask the user to enter the order quantity
            getline(cin, orderQuantityF); // Get user input for the order quantity

            if (orderQuantityF.find_first_not_of("0123456789") == string::npos) { // Check if the order quantity contains only digits, no other characters, only digits
                validOrderQuantityF = true; // Set the valid order quantity flag to true, this means that the order quantity is valid
            } else { // If the order quantity is not valid
                cout << "Invalid input. Please enter a valid number for Order Quantity." << endl; // Tell the user that the order quantity is invalid, will loop again
            }
        }

        bool correctOrderDateF = false; // Set the correct order date flag to false
        while (!correctOrderDateF) { // While loop to keep asking the user to enter a valid order date
            cout << "Enter Order Date (format: dd/mm/yy): "; // Ask the user to enter the order date, the format is dd/mm/yy
            getline(cin, orderDateF); // Get user input for the order date

            if (orderDateF.length() == 8 && orderDateF[2] == '/' && orderDateF[5] == '/') { // Check if the order date is 8 characters long and contains / at the 3rd and 6th position, no other characters
                string dayString = orderDateF.substr(0, 2); // Extract the day, month, and year from the order date
                string monthString = orderDateF.substr(3, 2); // Extract the day, month, and year from the order date
                string yearString = orderDateF.substr(6, 2); // Extract the day, month, and year from the order date
                int day = stoi(dayString); // Convert the day, month, and year to integers
                int month = stoi(monthString); // Convert the day, month, and year to integers
                int year = stoi(yearString); // Convert the day, month, and year to integers

                if (day >= 1 && day <= 31 && month >= 1 && month <= 12 && year >= 0 && year >= 24) { // Check if the day, month, and year are valid and year is after 2024
```

```
                correctOrderDateF = true; // Set the correct order date flag to true, this means that the order date is valid
            }
        }

        if (!correctOrderDateF) { // If the order date is not valid
            cout << "Invalid format. Please enter a valid Order Date (format: dd/mm/yy) Ensure that year is on or after 2024." << endl; // Tell the user that the order date is invalid, will loop again
        }
    }

    bool validOrderState = false; // Set the valid order state flag to false
    while (!validOrderState) { // While loop to keep asking the user to enter a valid order state
        cout << "Enter Order State (Pending, In Progress, Completed, Cancelled): "; // Ask the user to enter the order state, the options are Pending, In Progress, Completed, Cancelled
        getline(cin, orderStateF); // Get user input for the order state

        transform(orderStateF.begin(), orderStateF.end(), orderStateF.begin(), ::tolower); // Convert the order state to lowercase

        if (orderStateF == "pending" || orderStateF == "in progress" || orderStateF == "completed" || orderStateF == "cancelled") { // Check if the order state is valid
            validOrderState = true; // Set the valid order state flag to true, this means that the order state is valid
        }
        else { // If the order state is not valid
            cout << "Invalid status. Please enter 'Pending', 'In Progress', 'Completed', 'Cancelled'." << endl; // Tell the user that the order state is invalid, will loop again
        }
    }

    cout << "Enter Shipping Details: "; // Ask the user to enter the shipping details
    getline(cin, shippingDetailsF); // Get user input for the shipping details

    block.information.push_back(make_pair("Block", "Order Fulfillment Information")); // Add the order fulfillment information block to the block information vector
    block.information.push_back(make_pair("Customer ID", customerID)); // Add the customer ID to the block information vector
```

```
        block.information.push_back(make_pair("Order Quantity", orderQuantityF));
// Add the order quantity to the block information vector
        block.information.push_back(make_pair("Order Date", orderDateF)); // Add
the order date to the block information vector
        block.information.push_back(make_pair("Order State", orderStateF)); // Add
the order state to the block information vector
        block.information.push_back(make_pair("Shipping Details",
shippingDetailsF)); // Add the shipping details to the block information vector

        cout << "\n\nOrder Fulfillment Information Block Successfully Added.\n"; //
Tell the user that the order fulfillment information block has been successfully
added
    }

    void addTransportationInformation(Block& block) { // Function to add
transportation information to the block
        string transportationMode, transportationCompany,
transportationRouteFrom, transportationRouteTo,
transportationDepartureDateTime, transportationEstimatedArrivalDateTime; //
Declare variables for the transportation information, variables declared are
transportation mode, transportation company, transportation route from,
transportation route to, transportation departure date and time, and transportation
estimated arrival date and time
        bool validTransportationMode = false; // Set the valid transportation mode
flag to false

        while (!validTransportationMode) { // While loop to keep asking the user to
enter a valid transportation mode
            cout << "\nEnter Transportation Mode (Road, Rail, Sea, Air): "; // Ask the
user to enter the transportation mode, the options are Road, Rail, Sea, Air
            getline(cin, transportationMode); // Get user input for the transportation
mode

            transform(transportationMode.begin(), transportationMode.end(),
transportationMode.begin(), ::tolower); // Convert the transportation mode to
lowercase

            if (transportationMode == "road" || transportationMode == "rail" ||
transportationMode == "sea" || transportationMode == "air") { // Check if the
transportation mode is valid
                validTransportationMode = true; // Set the valid transportation mode
flag to true, this means that the transportation mode is valid
            }
            else { // If the transportation mode is not valid
```

```
            cout << "Invalid status. Please enter 'Road', 'Rail', 'Sea', 'Air" << endl; // Tell the user that the transportation mode is invalid, will loop again
        }
    }

    cout << "Enter Transportation Company: "; // Ask the user to enter the transportation company
    getline(cin, transportationCompany); // Get user input for the transportation company

    vector<string> validLocations; // Declare a vector of strings for the valid locations
    ifstream file("valid_locations.txt"); // Open the valid locations file
    if (file.is_open()) { // If the valid locations file is open
        string Locationline; // Declare a string for the line
        while (getline(file, Locationline)) { // While loop to get each line from the valid locations file
            validLocations.push_back(Locationline); // Add the line to the valid locations vector
        }
        file.close(); // Close the valid locations file
    } else { // If the valid locations file is not open
        cout << "Error: Unable to open valid locations file." << endl; // Tell the user that the valid locations file cannot be opened
        return; // Return from the function
    }

    cout << "Enter Transportation Route: "; // Ask the user to enter the transportation route
    bool validTransportationRouteFrom = false; // Set the valid transportation route from flag to false
    while (!validTransportationRouteFrom) { // While loop to keep asking the user to enter a valid transportation route from
        cout << "\nFrom (format: city, state): "; // Ask the user to enter the transportation route from, the format is: city, state
        getline(cin, transportationRouteFrom); // Get user input for the transportation route from

        validTransportationRouteFrom = isValidLocation(transportationRouteFrom, validLocations); // Check if the transportation route from is valid

        if (!validTransportationRouteFrom) { // If the transportation route from is not valid
```

```
            cout << "Invalid format. Please enter a valid location in the format 'city, state'." << endl; // Tell the user that the transportation route from is invalid, will loop again
        }
    }

    bool validTransportationRouteTo = false; // Set the valid transportation route to flag to false
    while (!validTransportationRouteTo) { // While loop to keep asking the user to enter a valid transportation route to
        cout << "To (format: city, state): "; // Ask the user to enter the transportation route to, the format is: city, state
        getline(cin, transportationRouteTo); // Get user input for the transportation route to

        validTransportationRouteTo = isValidLocation(transportationRouteTo, validLocations); // Check if the transportation route to is valid

        if (!validTransportationRouteTo) { // If the transportation route to is not valid
            cout << "Invalid format. Please enter a valid location in the format 'city, state'." << endl; // Tell the user that the transportation route to is invalid, will loop again
        }
    }

    bool correctTransportationDepartureDate = false; // Set the correct transportation departure date flag to false
    while (!correctTransportationDepartureDate) { // While loop to keep asking the user to enter a valid transportation departure date
        cout << "Enter Transportation Departure Date (format: dd/mm/yy): "; // Ask the user to enter the transportation departure date, the format is: dd/mm/yy
        getline(cin, transportationDepartureDateTime); // Get user input for the transportation departure date

        if (transportationDepartureDateTime.length() == 8 && transportationDepartureDateTime[2] == '/' && transportationDepartureDateTime[5] == '/') { // Check if the transportation departure date is in the correct format
            string dayString = transportationDepartureDateTime.substr(0, 2); // Get the day from the transportation departure date
            string monthString = transportationDepartureDateTime.substr(3, 2); // Get the month from the transportation departure date
```

```
            string yearString = transportationDepartureDateTime.substr(6, 2); // Get the year from the transportation departure date
            int day = stoi(dayString); // Convert the day string to an integer
            int month = stoi(monthString); // Convert the month string to an integer
            int year = stoi(yearString); // Convert the year string to an integer

            if (day >= 1 && day <= 31 && month >= 1 && month <= 12 && year >= 0 && year >= 24) { // Check if the day, month, and year are valid, the day must be between 1 and 31, the month must be between 1 and 12, and the year must be between 0 and 99
                correctTransportationDepartureDate = true; // Set the correct transportation departure date flag to true, this means that the transportation departure date is valid
            }
        }

        if (!correctTransportationDepartureDate) { // If the transportation departure date is not valid
            cout << "Invalid format. Please enter a valid Order Date (format: dd/mm/yy) Ensure that year is on or after 2024." << endl; // Tell the user that the transportation departure date is invalid, will loop again
        }
    }

    bool correctTransportationArrivalDate = false; // Set the correct transportation arrival date flag to false
    while (!correctTransportationArrivalDate) { // While loop to keep asking the user to enter a valid transportation arrival date
        cout << "Enter Transportation Estimated Arrival Date (format: dd/mm/yy): "; // Ask the user to enter the transportation estimated arrival date, the format is: dd/mm/yy
        getline(cin, transportationEstimatedArrivalDateTime); // Get user input for the transportation estimated arrival date

        if (transportationEstimatedArrivalDateTime.length() == 8 && transportationEstimatedArrivalDateTime[2] == '/' && transportationEstimatedArrivalDateTime[5] == '/') { // Check if the transportation estimated arrival date is in the correct format
            string dayString = transportationEstimatedArrivalDateTime.substr(0, 2); // Get the day from the transportation estimated arrival date
            string monthString = transportationEstimatedArrivalDateTime.substr(3, 2); // Get the month from the transportation estimated arrival date
            string yearString = transportationEstimatedArrivalDateTime.substr(6, 2); // Get the year from the transportation estimated arrival date
```

```
            int day = stoi(dayString); // Convert the day string to an integer
            int month = stoi(monthString); // Convert the month string to an integer
            int year = stoi(yearString); // Convert the year string to an integer

            if (day >= 1 && day <= 31 && month >= 1 && month <= 12 && year >= 0 && year >= 24) { // Check if the day, month, and year are valid, the day must be between 1 and 31, the month must be between 1 and 12, and the year must be between 0 and 99
                // Check if the arrival date is equal to or after the departure date
                if (year > stoi(transportationDepartureDateTime.substr(6, 2)) ||
                    (year == stoi(transportationDepartureDateTime.substr(6, 2)) && month > stoi(transportationDepartureDateTime.substr(3, 2))) ||
                    (year == stoi(transportationDepartureDateTime.substr(6, 2)) && month == stoi(transportationDepartureDateTime.substr(3, 2)) && day >= stoi(transportationDepartureDateTime.substr(0, 2)))) {
                    correctTransportationArrivalDate = true; // Set the correct transportation arrival date flag to true, this means that the transportation arrival date is valid
                } else {
                    cout << "Invalid arrival date. Arrival date must be equal to or after the departure date." << endl; // Tell the user that the transportation arrival date is invalid, will loop again
                }
            }
        }

        if (!correctTransportationArrivalDate) { // If the transportation arrival date is not valid
            cout << "Invalid format. Please enter a valid Order Date (format: dd/mm/yy) Ensure that year is on or after 2024." << endl; // Tell the user that the transportation arrival date is invalid, will loop again
        }
    }

    block.information.push_back(make_pair("Block", "Transportation Information")); // Add the transportation information block to the block information
    block.information.push_back(make_pair("Transportation Mode", transportationMode)); // Add the transportation mode to the block information
    block.information.push_back(make_pair("Transportation Company", transportationCompany)); // Add the transportation company to the block information
    block.information.push_back(make_pair("Transportation Route", transportationRouteFrom + " to " + transportationRouteTo)); // Add the transportation route to the block information
```

```
        block.information.push_back(make_pair("Transportation Departure Date", transportationDepartureDateTime)); // Add the transportation departure date to the block information
        block.information.push_back(make_pair("Transportation Estimated Arrival Date", transportationEstimatedArrivalDateTime)); // Add the transportation estimated arrival date to the block information

        cout << "\n\nTransportation Information Block Successfully Added.\n"; // Tell the user that the transportation information block has been successfully added
    }

    string expectedProductWorthiness; // Declare a string to store the expected product worthiness value

    void addCustomerDeliverySatisfactionInformation(Block& block) { // Function to add the customer delivery satisfaction information to the block
        string deliveryConfirmation, customerFeedbackCollection, satisfactionSurvey; // Declare strings to store the delivery confirmation, customer feedback collection, and satisfaction survey values

        bool validDeliveryConfirmation = false; // Set the valid delivery confirmation flag to false

        while (!validDeliveryConfirmation) { // While loop to keep asking the user to enter a valid delivery confirmation
            cout << "\nEnter Delivery Confirmation (format: Delivered, Unsuccessful, Rescheduled): "; // Ask the user to enter the delivery confirmation, the format is: Delivered, Unsuccessful, Rescheduled
            getline(cin, deliveryConfirmation); // Get user input for the delivery confirmation

            transform(deliveryConfirmation.begin(), deliveryConfirmation.end(), deliveryConfirmation.begin(), ::tolower); // Convert the delivery confirmation to lowercase

            if (deliveryConfirmation == "delivered" || deliveryConfirmation == "unsuccessful" || deliveryConfirmation == "rescheduled") { // Check if the delivery confirmation is valid, the delivery confirmation must be: Delivered, Unsuccessful, Rescheduled
                validDeliveryConfirmation = true; // Set the valid delivery confirmation flag to true, this means that the delivery confirmation is valid
            }
            else { // If the delivery confirmation is not valid
```

```
            cout << "Invalid status. Please enter 'Delivered', 'Unsuccessful', 'Rescheduled'." << endl; // Tell the user that the delivery confirmation is invalid, will loop again
        }
    }

    bool validCustomerFeedbackConfirmation = false; // Set the valid customer feedback confirmation flag to false

    while (!validCustomerFeedbackConfirmation) { // While loop to keep asking the user to enter a valid customer feedback collection
        cout << "Enter Customer Feedback Collection Result (format: Excellent, Good, Average, Poor): "; // Ask the user to enter the customer feedback collection result, the format is: Excellent, Good, Average, Poor
        getline(cin, customerFeedbackCollection); // Get user input for the customer feedback collection

        transform(customerFeedbackCollection.begin(), customerFeedbackCollection.end(), customerFeedbackCollection.begin(), ::tolower); // Convert the customer feedback collection to lowercase

        if (customerFeedbackCollection == "excellent" || customerFeedbackCollection == "good" || customerFeedbackCollection == "average" || customerFeedbackCollection == "poor") { // Check if the customer feedback collection is valid, the customer feedback collection must be: Excellent, Good, Average, Poor
            validCustomerFeedbackConfirmation = true; // Set the valid customer feedback confirmation flag to true, this means that the customer feedback collection is valid
        }
        else { // If the customer feedback collection is not valid
            cout << "Invalid status. Please enter 'Excellent', 'Good', 'Average', 'Poor'." << endl; // Tell the user that the customer feedback collection is invalid, will loop again
        }
    }

    bool correctSatisfactionSurvey = false; // Set the correct satisfaction survey flag to false
    while (!correctSatisfactionSurvey) { // While loop to keep asking the user to enter a valid satisfaction survey
        cout << "Enter Satisfaction Survey Result (format: 1-10): "; // Ask the user to enter the satisfaction survey result, the format is: 1-10
```

```
            getline(cin, satisfactionSurvey); // Get user input for the satisfaction
survey

            try { // Try to convert the satisfaction survey to an integer
                int surveyResult = stoi(satisfactionSurvey); // Convert the satisfaction
survey to an integer
                if (surveyResult >= 1 && surveyResult <= 10) { // Check if the
satisfaction survey is valid, the satisfaction survey must be between 1 and 10
                    correctSatisfactionSurvey = true; // Set the correct satisfaction
survey flag to true, this means that the satisfaction survey is valid
                } else { // If the satisfaction survey is not valid
                    cout << "Invalid input. Please enter a number between 1 and 10." <<
endl; // Tell the user that the satisfaction survey is invalid, will loop again
                }
            } catch (...) { // Catch any errors
                cout << "Invalid input. Please enter a number between 1 and 10." <<
endl; // Tell the user that the satisfaction survey is invalid, will loop again
            }
            expectedProductWorthiness = satisfactionSurvey; // Set the expected
product worthiness to the satisfaction survey
        }

        block.information.push_back(make_pair("Block", "Customer Delivery
Satisfactory Information")); // Add the customer delivery satisfactory information
block to the block information
        block.information.push_back(make_pair("Delivery Confirmation",
deliveryConfirmation)); // Add the delivery confirmation to the block information
        block.information.push_back(make_pair("Customer Feedback Collection",
customerFeedbackCollection)); // Add the customer feedback collection to the
block information
        block.information.push_back(make_pair("Satisfaction Survey",
satisfactionSurvey)); // Add the satisfaction survey to the block information

        cout << "\n\nCustomer Delivery Satisfactory Information Block Successfully
Added.\n"; // Tell the user that the customer delivery satisfactory information
block has been successfully added
    }

    void addQualityInspectionControlInformation(Block& block) { // Function to add
the quality inspection control information to the block
        string productInspectionDetails, productQuality; // Declare strings to store
the product inspection details and product quality values
```

```
        bool validproductInspectionDetails = false; // Set the valid product inspection details flag to false

        while(validproductInspectionDetails == false) { // While loop to keep asking the user to enter a valid product inspection details
            cout << "\nEnter Product Quality Inspection Result (format: Pass, Fail): "; // Ask the user to enter the product inspection details, the format is: Pass, Fail
            getline(cin, productInspectionDetails); // Get user input for the product inspection details

            transform(productInspectionDetails.begin(), productInspectionDetails.end(), productInspectionDetails.begin(), ::tolower); // Convert the product inspection details to lowercase
            if(productInspectionDetails == "pass" || productInspectionDetails == "fail") { // Check if the product inspection details is valid, the product inspection details must be: Pass, Fail
                validproductInspectionDetails = true; // Set the valid product inspection details flag to true, this means that the product inspection details is valid
            } else { // If the product inspection details is not valid
                cout << "Invalid status. Please enter 'Pass', 'Fail'." << endl; // Tell the user that the product inspection details is invalid, will loop again
            }
        }

        bool validProductQuality = false; // Set the valid product quality flag to false

        while(validProductQuality == false) { // While loop to keep asking the user to enter a valid product quality
            cout << "Enter Product Quality (format: Excellent, Good, Average, Poor): "; // Ask the user to enter the product quality, the format is: Excellent, Good, Average, Poor
            getline(cin, productQuality); // Get user input for the product quality

            transform(productQuality.begin(), productQuality.end(), productQuality.begin(), ::tolower); // Convert the product quality to lowercase
            if(productQuality == "excellent" || productQuality == "good" || productQuality == "average" || productQuality == "poor") { // Check if the product quality is valid, the product quality must be: Excellent, Good, Average, Poor
                validProductQuality = true; // Set the valid product quality flag to true, this means that the product quality is valid
            } else { // If the product quality is not valid
                cout << "Invalid status. Please enter 'Excellent', 'Good', 'Average', 'Poor'." << endl; // Tell the user that the product quality is invalid, will loop again
            }
```

```
        }

        block.information.push_back(make_pair("Block", "Quality Inspection Control
Information")); // Add the quality inspection control information block to the block
information
        block.information.push_back(make_pair("Quality Inspection Result",
productInspectionDetails)); // Add the product inspection details to the block
information
        block.information.push_back(make_pair("Product Quality", productQuality));
// Add the product quality to the block information

        cout << "\n\nQuality Inspection Control Information Block Successfully
Added.\n"; // Tell the user that the quality inspection control information block has
been successfully added
    }

    void addProductReturnInformation(Block& block) { // Function to add the
product return information to the block
        string productReturnStatus, productReturnReason, productRefundStatus; //
Declare strings to store the product return status, product return reason and
product refund status values

        bool validProductReturnStatus = false; // Set the valid product return status
flag to false

        while(validProductReturnStatus == false) { // While loop to keep asking the
user to enter a valid product return status
            cout << "\nEnter Product Return Status (format: Returned, Not Returned):
"; // Ask the user to enter the product return status, the format is: Returned, Not
Returned
            getline(cin, productReturnStatus); // Get user input for the product return
status

            transform(productReturnStatus.begin(), productReturnStatus.end(),
productReturnStatus.begin(), ::tolower); // Convert the product return status to
lowercase
            if(productReturnStatus == "returned" || productReturnStatus == "not
returned") { // Check if the product return status is valid, the product return status
must be: Returned, Not Returned
                validProductReturnStatus = true; // Set the valid product return status
flag to true, this means that the product return status is valid
            } else { // If the product return status is not valid
                cout << "Invalid status. Please enter 'Returned', 'Not Returned'." <<
endl; // Tell the user that the product return status is invalid, will loop again
```

```
        }
    }

    bool validProductRefundStatus = false; // Set the valid product refund status flag to false

    while(validProductRefundStatus == false) { // While loop to keep asking the user to enter a valid product refund status
        cout << "Enter Product Refund Status (format: Refunded, Not Refunded): "; // Ask the user to enter the product refund status, the format is: Refunded, Not Refunded
        getline(cin, productRefundStatus); // Get user input for the product refund status

        transform(productRefundStatus.begin(), productRefundStatus.end(), productRefundStatus.begin(), ::tolower); // Convert the product refund status to lowercase
        if(productRefundStatus == "refunded" || productRefundStatus == "not refunded") { // Check if the product refund status is valid, the product refund status must be: Refunded, Not Refunded
            validProductRefundStatus = true; // Set the valid product refund status flag to true, this means that the product refund status is valid
        } else { // If the product refund status is not valid
            cout << "Invalid status. Please enter 'Refunded', 'Not Refunded'." << endl; // Tell the user that the product refund status is invalid, will loop again
        }
    }

    cout << "Enter Product Return Reason: "; // Ask the user to enter the product return reason
    getline(cin, productReturnReason); // Get user input for the product return reason

    string productReturnNumber = "R" + to_string(rand() % 90000 + 10000); // Generate a random product return number

    block.information.push_back(make_pair("Block", "Product Returns Information")); // Add the product returns information block to the block information
    block.information.push_back(make_pair("Product Return Number", productReturnNumber)); // Add the product return number to the block information
    block.information.push_back(make_pair("Product Return Status", productReturnStatus)); // Add the product return status to the block information
```

```
        block.information.push_back(make_pair("Product Return Reason",
productReturnReason)); // Add the product return reason to the block information

        cout << "\n\nProduct Returns Information Block Successfully Added.\n"; //
Tell the user that the product returns information block has been successfully
added
    }

    void addProductWorthinessInformation(Block& block) { // Function to add the
product worthiness information to the block
        string productWorthinessStatus, productWorthinessReason; // Declare
strings to store the product worthiness status and product worthiness reason
values

        bool validProductWorthinessStatus = false; // Set the valid product
worthiness status flag to false

        while (!validProductWorthinessStatus) { // While loop to keep asking the
user to enter a valid product worthiness status
            cout << "\nEnter Product Worthiness Status (format: Continue Product,
Discontinue Product): "; // Ask the user to enter the product worthiness status,
the format is: Continue Product, Discontinue Product
            getline(cin, productWorthinessStatus); // Get user input for the product
worthiness status

            transform(productWorthinessStatus.begin(),
productWorthinessStatus.end(), productWorthinessStatus.begin(), ::tolower); //
Convert the product worthiness status to lowercase
            if (productWorthinessStatus == "continue product" ||
productWorthinessStatus == "discontinue product") { // Check if the product
worthiness status is valid, the product worthiness status must be: Continue
Product, Discontinue Product
                validProductWorthinessStatus = true; // Set the valid product
worthiness status flag to true, this means that the product worthiness status is
valid
            } else { // If the product worthiness status is not valid
                cout << "Invalid status. Please enter 'Continue Product', 'Discontinue
Product'." << endl; // Tell the user that the product worthiness status is invalid, will
loop again
            }
        }

        cout << "Enter Product Worthiness Reason: "; // Ask the user to enter the
product worthiness reason
```

```
        getline(cin, productWorthinessReason); // Get user input for the product worthiness reason

        bool correctProductWorthiness = false; // Set the correct product worthiness flag to false
        while (!correctProductWorthiness) { // While loop to keep asking the user to confirm the product worthiness status
            string userInput; // Declare a string to store the user input
            cout << "Based on the customer feedback number, the expected product worthiness is '" << expectedProductWorthiness << "'. Enter 'confirm' to proceed or 'reenter' to re-enter the product worthiness status: "; // Ask the user to confirm the product worthiness status, it uses the productworthiness variable to display the expected product worthiness
            getline(cin, userInput); // Get user input for the product worthiness status confirmation

            transform(userInput.begin(), userInput.end(), userInput.begin(), ::tolower); // Convert the user input to lowercase
            if (userInput == "confirm") { // If the user input is confirm
                correctProductWorthiness = true; // Set the correct product worthiness flag to true, this means that the product worthiness status is correct
            } else if (userInput == "reenter") { // If the user input is reenter
                cout << "\nEnter Product Worthiness Status (format: Continue Product, Discontinue Product): "; // Ask the user to re-enter the product worthiness status, the format is: Continue Product, Discontinue Product
                getline(cin, productWorthinessStatus); // Get user input for the product worthiness status

                transform(productWorthinessStatus.begin(), productWorthinessStatus.end(), productWorthinessStatus.begin(), ::tolower); // Convert the product worthiness status to lowercase
                if (productWorthinessStatus == "continue product" || productWorthinessStatus == "discontinue product") { // Check if the product worthiness status is valid, the product worthiness status must be: Continue Product, Discontinue Product
                    validProductWorthinessStatus = true; // Set the valid product worthiness status flag to true, this means that the product worthiness status is valid
                } else { // If the product worthiness status is not valid
                    cout << "Invalid status. Please enter 'Continue Product', 'Discontinue Product'." << endl; // Tell the user that the product worthiness status is invalid, will loop again
                }
            } else { // If the user input is not confirm or reenter
```

```
            cout << "Invalid input. Please enter 'confirm' or 'reenter'." << endl; // Tell the user that the input is invalid, will loop again
        }
    }

    block.information.push_back(make_pair("Block", "Product Worthiness Information")); // Add the product worthiness information block to the block information
    block.information.push_back(make_pair("Product Worthiness Status", productWorthinessStatus)); // Add the product worthiness status to the block information
    block.information.push_back(make_pair("Product Worthiness Reason", productWorthinessReason)); // Add the product worthiness reason to the block information

    cout << "\n\nProduct Worthiness Information Block Successfully Added.\n"; // Tell the user that the product worthiness information block has been successfully added
  }

  string generateRandomHash() { // Function to generate a random hash
    srand(static_cast<unsigned int>(time(nullptr)));  // Seed the random number generator
    string chars = "abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789"; // Declare a string of characters to use for the random hash
    string randHash = ""; // Declare a string to store the random hash
    for (int i = 0; i < 20; i++) { // For loop to generate the random hash, loop 20 times
      int index = rand() % chars.size(); // Generate a random index
      randHash += chars[index]; // Add the character at the random index to the random hash
    }
    return randHash; // Return the random hash
  }

  void exportBlocksToFile(const string& filename) { // Function to export the blockchain to a file
    ofstream outfile(filename); // Create an output file stream with the filename
    if (!outfile.is_open()) { // If the file is not open
      cout << "Error: Unable to open file for writing." << endl; // Tell the user that there was an error opening the file
      return; // Return from the function
    }
```

```
        BlockNode* temp = head; // Create a temporary block node and set it to the head of the blockchain, this will be used to traverse the blockchain to export the blocks to the file
        while (temp != nullptr) { // While loop to traverse the blockchain
            const Block& block = temp->data; // Get the block data from the temporary block node
            outfile << "Block " << block.blockNumber << " | " << block.currentHashNumber << " | " << block.previousHashNumber << " | " << block.currentTimeStamp << "information: "; // Write the block number, current hash number, previous hash number, and current time stamp to the file

            for (size_t i = 0; i < block.information.size(); ++i) { // For loop to write the block information to the file
                const pair<string, string>& info = block.information[i]; // Get the block information from the block data
                outfile << " " << info.first << ": " << info.second << " | "; // Write the block information to the file
            }
            outfile << endl; // End the line

            temp = temp->next; // Move to the next block
        }

        outfile.close(); // Close the file
        cout << "Blocks exported to " << filename << " successfully." << endl; // Tell the user that the blocks have been successfully exported to the file
    }

    bool isValidLocation(const string& location, const vector<string>& validLocations) { // Method to check if a location is valid
        return find(validLocations.begin(), validLocations.end(), location) != validLocations.end(); // Return true if the location is found in the valid locations vector, otherwise return false
    }

    int getCurrentBlockNumber() { // Method to get the current block number, this is a getter method used to get the current block number
        return currentBlockNumber; // Return the current block number
    }

    void displayChain() { //Method to display block, this is strictly for displaying purposes and is strictly "virtual"
```

```
    BlockNode* temp = head; // Create a temporary block node and set it to the head of the blockchain, this will be used to traverse the blockchain to search for the block
    string blockNumberInput; // Declare a variable to store the block number that the user wants to search
    cout << "\nEnter the block number you want to view (format: 1, 2, 3, ...) (* to view all): "; //Users can enter the specific block number of enter "*" asterisk to view all
    cin >> blockNumberInput; // Get user input for the block number they want to search

    bool found = false; // Declare a flag to indicate if the block was found
    while (temp != nullptr) { // While loop to traverse the blockchain
        const Block& block = temp->data; // Get the block data from the temporary block node
        if (!block.isHardDeleted && (blockNumberInput == "*" || stoi(blockNumberInput) == block.blockNumber)) {
            found = true;
            cout << "\nBlock " << block.blockNumber << " | " << block.currentHashNumber << " | " << block.previousHashNumber << " | " << block.currentTimeStamp;

            if (!block.isSoftDeleted) { //If the block is soft deleted
                cout << " information: ";
                for (size_t i = 0; i < block.information.size(); ++i) {
                    const pair<string, string>& info = block.information[i];
                    cout << " " << info.first << ": " << info.second << " | ";
                }
            }

            cout << endl;

            if (blockNumberInput != "*") {
                return;
            }
        }
        temp = temp->next;
    }

    if (!found) {
        cout << "\nBlock with number " << blockNumberInput << " not found." << endl;
    }
}
```

```
    void searchBlockByNumber() { // Method to search block, unaffected by any
deletions
        BlockNode* temp = head; // Create a temporary block node and set it to the
head of the blockchain, this will be used to traverse the blockchain to search for
the block
        bool found = false;  // Declare flag to indicate if the block was found
        cout << "\nEnter which block number you want to search (format: 1, 2, 3, ...)
(* to view all): ";
        string blockNumberInput; // Delcare variable
        cin >> blockNumberInput;  // Get user input

        if (blockNumberInput == "*") { //If is asterisk then show all
            while (temp != nullptr) {
                const Block& block = temp->data;
                cout << "\nBlock " << block.blockNumber << " | " <<
block.currentHashNumber << " | " << block.previousHashNumber << " | " <<
block.currentTimeStamp << " information: ";

                for (size_t i = 0; i < block.information.size(); ++i) {
                    const pair<string, string>& info = block.information[i];
                    cout << " " << info.first << ": " << info.second << " | ";
                }
                cout << endl;
                temp = temp->next;
            }
            return;
        }

        int blockNumber = stoi(blockNumberInput);  //convert the string into int so
user can search for speciic block
        while (temp != nullptr) {
            const Block& block = temp->data;
            if (block.blockNumber == blockNumber) {
                found = true;
                cout << "\nBlock " << block.blockNumber << " | " <<
block.currentHashNumber << " | " << block.previousHashNumber << " | " <<
block.currentTimeStamp << " information: ";

                for (size_t i = 0; i < block.information.size(); ++i) {
                    const pair<string, string>& info = block.information[i];
                    cout << " " << info.first << ": " << info.second << " | ";
                }
                cout << endl;
```

```
            break;
        }
        temp = temp->next;
    }

    if (!found) {
        cout << "\nBlock with number " << blockNumber << " not found." << endl;
    }
}

void softDeleteBlock(int blockNumber) { // Function to soft delete a block by block number
    BlockNode* currentNode = head; // Create a temporary block node and set it to the head of the blockchain, this will be used to traverse the blockchain to search for the block
    while (currentNode != nullptr && currentNode->data.blockNumber != blockNumber) { // While loop to traverse the blockchain to search for the block
        currentNode = currentNode->next; // Move to the next block
    }

    if (currentNode == nullptr) { // If the block was not found
        cout << "Block with block number " << blockNumber << " not found." << endl; // Tell the user that the block with the specified block number was not found
        return;
    }

    if (currentNode->data.isSoftDeleted) { // If the block has already been soft deleted
        cout << "Block with block number " << blockNumber << " has already been soft deleted." << endl; // Tell the user that the block with the specified block number has already been soft deleted
        return;
    }

    if (currentNode->data.isHardDeleted) { // If the block has already been hard deleted
        cout << "Block with block number " << blockNumber << " has been hard deleted and cannot be soft deleted." << endl; // Tell the user that the block with the specified block number has already been hard deleted and cannot be soft deleted
        return;
    }

    currentNode->data.isSoftDeleted = true; // Set the isSoftDeleted flag to true
```

```
        cout << "Information in block with block number " << blockNumber << " has been soft deleted." << endl; // Tell the user that the information in the block with the specified block number has been soft deleted
    }

    void hardDeleteBlock(int blockNumber) { // Function to hard delete a block by block number
        BlockNode* currentNode = head; // Create a temporary block node and set it to the head of the blockchain, this will be used to traverse the blockchain to search for the block
        while (currentNode != nullptr && currentNode->data.blockNumber != blockNumber) { // While loop to traverse the blockchain to search for the block
            currentNode = currentNode->next; // Move to the next block
        }

        if (currentNode == nullptr) { // If the block was not found
            cout << "Block with block number " << blockNumber << " not found." << endl; // Tell the user that the block with the specified block number was not found
            return;
        }

        if (currentNode->data.isHardDeleted) { // If the block has already been hard deleted
            cout << "Block with block number " << blockNumber << " has already been hard deleted." << endl; // Tell the user that the block with the specified block number has already been hard deleted
            return;
        }

        if (currentNode->data.isSoftDeleted) { // If the block has been soft deleted
            cout << "Block with block number " << blockNumber << " has been soft deleted and cannot be hard deleted." << endl; // Tell the user that the block with the specified block number has been soft deleted and cannot be hard deleted
            return;
        }

        currentNode->data.isHardDeleted = true; // Set the isHardDeleted flag to true
        cout << "Block with block number " << blockNumber << " has been hard deleted." << endl; // Tell the user that the block with the specified block number has been hard deleted
    }
};
```

```
bool authenticate(const string& username, const string& password) { // Function to authenticate the user, this function takes a username and password as parameters
    ifstream file("username_password.txt"); // Create an input file stream with the filename
    if (!file.is_open()) { // If the file is not open
        cout << "Error opening file." << endl; // Tell the user that there was an error opening the file
        return false; // Return false
    }

    string lineUsernamePassword; // Declare a string to store each line of the file
    while (getline(file, lineUsernamePassword)) { // While loop to read each line of the file
        stringstream ss(lineUsernamePassword); // Create a string stream with the line
        string storedUsername, storedPassword; // Declare strings to store the username and password from the file
        if (getline(ss, storedUsername, ',') && getline(ss, storedPassword)) { // If the username and password are successfully read from the file
            storedUsername.erase(remove_if(storedUsername.begin(), storedUsername.end(), ::isspace), storedUsername.end()); // Remove any whitespace from the stored username
            storedPassword.erase(remove_if(storedPassword.begin(), storedPassword.end(), ::isspace), storedPassword.end()); // Remove any whitespace from the stored password
            if (username == storedUsername && password == storedPassword) { // If the username and password match the stored username and password
                file.close(); // Close the file
                return true; // Return true
            }
        }
    }

    file.close(); // Close the file
    return false; // Return false
}

int main() { // Main function
    srand(time(0)); // Seed the random number generator

    Blockchain blockchain; // Create a blockchain object

    cout << "\nInventory and Transportation Management System." << endl;
```

```
    cout << "\nName: Lua Chong En";
    cout << "\nStudent ID: 20417309\n";

    int attempts = 0; // Declare an integer to store the number of login attempts
    bool authenticated = false; // Declare a boolean to store whether the user has
been authenticated
    while (attempts < 3) { // While loop to limit the number of login attempts to 3
        cout << "\n\nEnter Username: "; // Prompt the user to enter their username
        string username;
        cin >> username;
        cout << "\n\nEnter Password: "; // Prompt the user to enter their password
        string password;
        cin >> password;

        if (authenticate(username, password)) { // If the user is authenticated
            authenticated = true; // Set the authenticated flag to true
            break;
        } else { // If the user is not authenticated
            cout << "\nInvalid username or password. Please try again." << endl; //
Tell the user that the username or password is invalid
            attempts++; // Increment the number of login attempts, maximum 3
        }
    }

    if (!authenticated) { // If the user is not authenticated
        cout << "\nExceeded maximum login attempts. Exit program." << endl; // Tell
the user that the maximum number of login attempts has been exceeded
        return 0;
    }

    int userChoice; // Declare an integer to store the user's choice
    do { // Do-while loop to display the menu and process the user's choice
        cout << "\nBlockchain Menu\n"
            << "1. Add Block\n"
            << "2. Display Block - DEMO (Allows for Hard and Soft deletion which are
strictly \"Virtual\")\n"
            << "3. Search Block (Deletions does not affect searching for blocks as
deletions are strictly \"Virtual\")\n"
            << "4. Export to text file\n"
            << "5. Hard Delete Block\n"
            << "6. Soft Delete Block\n"
            << "7. Exit\n"
            << "Enter your choice: ";
        cin >> userChoice;
```

```
        cin.ignore();

        if (cin.fail()) { // If the input is invalid
            cin.clear(); // Clear the input buffer
            cin.ignore(numeric_limits<streamsize>::max(), '\n'); // Ignore the rest of
the input
            cout << "\nInvalid input. Please enter a number." << endl; // Tell the user
that the input is invalid
            continue;
        }

        switch (userChoice) { // Switch statement to process the user's choice
            case 1: { // If the user chooses to add a block
                if (blockchain.getCurrentBlockNumber() == 8) {
                    cout << "\nAll blocks in the blockchain have been added. No more
blocks can be added." << endl;
                    break;
                }
                vector<pair<string, string> > informationPair;
                blockchain.addBlock(informationPair);
                break;
            }
            case 2: { // If the user chooses to display the blockchain
                blockchain.displayChain();
                break;
            }
            case 3: { // If the user chooses to search for a block
                blockchain.searchBlockByNumber();
                break;
            }
            case 4: { // If the user chooses to export the blockchain to a text file
                blockchain.exportBlocksToFile("blockchain.txt");
                break;
            }
            case 5: { // If the user chooses to hard delete a block
                cout << "\nEnter which block number you want to hard delete: ";
                int blockNumber;
                cin >> blockNumber;
                blockchain.hardDeleteBlock(blockNumber);
                break;
            }
            case 6: { // If the user chooses to soft delete a block
                cout << "\nEnter which block number you want to soft delete: ";
                int blockNumber;
```

```
            cin >> blockNumber;
            blockchain.softDeleteBlock(blockNumber);
            break;
        }
        case 7: // If the user chooses to exit the program
            cout << "\nExit Program." << endl;
            break;
        default: // If the user chooses an invalid option
            cout << "\nInvalid choice. Please enter a valid choice." << endl;
      }
    } while (userChoice != 7); // If user enters an invalid number

    return 0;
}
```

b. valid_locations.txt

```
Johor Bahru, Johor
Tebrau, Johor
Pasir Gudang, Johor
Bukit Indah, Johor
Skudai, Johor
Kluang, Johor
Batu Pahat, Johor
Muar, Johor
Ulu Tiram, Johor
Senai, Johor
Segamat, Johor
Kulai, Johor
Kota Tinggi, Johor
Pontian Kechil, Johor
Tangkak, Johor
Bukit Bakri, Johor
Yong Peng, Johor
Pekan Nenas, Johor
Labis, Johor
Mersing, Johor
Simpang Renggam, Johor
Parit Raja, Johor
Kelapa Sawit, Johor
Buloh Kasap, Johor
Chaah, Johor
Sungai Petani, Kedah
```

Alor Setar, Kedah
Kulim, Kedah
Jitra / Kubang Pasu, Kedah
Baling, Kedah
Pendang, Kedah
Langkawi, Kedah
Yan, Kedah
Sik, Kedah
Kuala Nerang, Kedah
Pokok Sena, Kedah
Bandar Baharu, Kedah
Kota Bharu, Kelantan
Pangkal Kalong, Kelantan
Tanah Merah, Kelantan
Peringat, Kelantan
Wakaf Baru, Kelantan
Kadok, Kelantan
Pasir Mas, Kelantan
Gua Musang, Kelantan
Kuala Krai, Kelantan
Tumpat, Kelantan
Bandaraya Melaka, Melaka
Bukit Baru, Melaka
Ayer Keroh, Melaka
Klebang, Melaka
Masjid Tanah, Melaka
Sungai Udang, Melaka
Batu Berendam, Melaka
Alor Gajah, Melaka
Bukit Rambai, Melaka
Ayer Molek, Melaka
Bemban, Melaka
Kuala Sungai Baru, Melaka
Pulau Sebang, Melaka
Jasin, Melaka
Seremban, Negeri Sembilan
Port Dickson, Negeri Sembilan
Nilai, Negeri Sembilan
Bahau, Negeri Sembilan
Tampin, Negeri Sembilan
Kuala Pilah, Negeri Sembilan
Kuantan, Pahang
Temerloh, Pahang
Bentong, Pahang

Mentakab, Pahang
Raub, Pahang
Jerantut, Pahang
Pekan, Pahang
Kuala Lipis, Pahang
Bandar Jengka, Pahang
Bukit Tinggi, Pahang
Ipoh, Perak
Taiping, Perak
Sitiawan, Perak
Simpang Empat, Perak
Teluk Intan, Perak
Batu Gajah, Perak
Lumut, Perak
Kampung Koh, Perak
Kuala Kangsar, Perak
Sungai Siput Utara, Perak
Tapah, Perak
Bidor, Perak
Parit Buntar, Perak
Ayer Tawar, Perak
Bagan Serai, Perak
Tanjung Malim, Perak
Lawan Kuda Baharu, Perak
Pantai Remis, Perak
Kampar, Perak
Kangar, Perlis
Kuala Perlis, Perlis
Bukit Mertajam, Pulau Pinang
Georgetown, Pulau Pinang
Sungai Ara, Pulau Pinang
Gelugor, Pulau Pinang
Ayer Itam, Pulau Pinang
Butterworth, Pulau Pinang
Perai, Pulau Pinang
Nibong Tebal, Pulau Pinang
Permatang Kucing, Pulau Pinang
Tanjung Tokong, Pulau Pinang
Kepala Batas, Pulau Pinang
Tanjung Bungah, Pulau Pinang
Juru, Pulau Pinang
Kota Kinabalu, Sabah
Sandakan, Sabah
Tawau, Sabah

Lahad Datu, Sabah
Keningau, Sabah
Putatan, Sabah
Donggongon, Sabah
Semporna, Sabah
Kudat, Sabah
Kunak, Sabah
Papar, Sabah
Ranau, Sabah
Beaufort, Sabah
Kinarut, Sabah
Kota Belud, Sabah
Kuching, Sarawak
Miri, Sarawak
Sibu, Sarawak
Bintulu, Sarawak
Limbang, Sarawak
Sarikei, Sarawak
Sri Aman, Sarawak
Kapit, Sarawak
Batu Delapan Bazaar, Sarawak
Kota Samarahan, Sarawak
Subang Jaya, Selangor
Klang, Selangor
Ampang Jaya, Selangor
Shah Alam, Selangor
Petaling Jaya, Selangor
Cheras, Selangor
Kajang, Selangor
Selayang Baru, Selangor
Rawang, Selangor
Taman Greenwood, Selangor
Semenyih, Selangor
Banting, Selangor
Balakong, Selangor
Gombak Setia, Selangor
Kuala Selangor, Selangor
Serendah, Selangor
Bukit Beruntung, Selangor
Pengkalan Kundang, Selangor
Jenjarom, Selangor
Sungai Besar, Selangor
Batu Arang, Selangor
Tanjung Sepat, Selangor

Kuang, Selangor
Kuala Kubu Baharu, Selangor
Batang Berjuntai, Selangor
Bandar Baru Salak Tinggi, Selangor
Sekinchan, Selangor
Sabak, Selangor
Tanjung Karang, Selangor
Beranang, Selangor
Sungai Pelek, Selangor
Sepang, Selangor
Kuala Terengganu, Terengganu
Chukai, Terengganu
Dungun, Terengganu
Kerteh, Terengganu
Kuala Berang, Terengganu
Marang, Terengganu
Paka, Terengganu
Jerteh, Terengganu
Kuala Lumpur, Wilayah Persekutuan
Labuan, Wilayah

c. username_password.txt (There are 5 different **valid** username and passwords)

En, 123
Qq, 123
Ww, 321
Ee, 456
Tt, 789

d. blockchain.txt (example)

Block 8 | zWd4A6IZ90g5gMiSggxd | rjbJXkvoOBuamU9Yndqm | Tue Mar 19 18:05:48 2024
information: Block: Product Worthiness Information | Product Worthiness Status: continue product | Product Worthiness Reason: N/A |
Block 7 | rjbJXkvoOBuamU9Yndqm | 9UrkGZf99ISZBdxMezAF | Tue Mar 19 18:05:34 2024
information: Block: Product Returns Information | Product Return Number: R38687 | Product Return Status: returned | Product Return Reason: 1 |
Block 6 | 9UrkGZf99ISZBdxMezAF | b582IK4d9NUlxbQ8C5dE | Tue Mar 19 18:05:18 2024
information: Block: Quality Inspection Control Information | Quality Inspection Result: fail | Product Quality: poor |

Block 5 | b582IK4d9NUlxbQ8C5dE | X2N2wUs6v5oSErQGftCV | Tue Mar 19 18:05:06 2024
information: Block: Customer Delivery Satisfactory Information | Deilvery Confirmation: delivered | Customer Feedback Collection: poor | Satisfaction Survey: 9 |
Block 4 | X2N2wUs6v5oSErQGftCV | gCfw5JDJ0tI0iDCOqSs8 | Tue Mar 19 18:04:26 2024
information: Block: Transportation Information | Transportation Mode: road | Transportation Company: J&T Express | Transportation Route: Semenyih, Selangor to Muar, Johor | Transportation Departure Date: 22/02/24 | Transportation Estimated Arrival Date: 22/02/24 |
Block 3 | gCfw5JDJ0tI0iDCOqSs8 | Lb8KlQE4QrULIF9zgnEa | Tue Mar 19 18:04:05 2024
information: Block: Order Fulfillment Information | Customer ID: CID12345 | Order Quantity: 10 | Order Date: 22/02/24 | Order State: pending | Shipping Details: N/A |
Block 2 | Lb8KlQE4QrULIF9zgnEa | eFpC6HCcyhbtugesrumi | Tue Mar 19 18:03:34 2024
information: Block: Inventory Information | Warehouse ID: WID12345 | Storage Location: Semenyih, Selangor | Inventory Quantity: 10 | Inventory Status: low stock |
Block 1 | eFpC6HCcyhbtugesrumi | eFpC6HCcyhbtugesrumi | Tue Mar 19 17:46:43 2024
information: Block: Procurement Information | Supplier ID: SID12345 | Supplier Name: Lua Chong En | Order Quantity: 10 | Order Date: 22/02/24 | Order State: pending | Shipping Details: N/A |

16. References

For valid_locations.txt, the list of all Malaysian cities and states where taken from https://github.com/hazz1925/malaysian-states/blob/master/states-cities.json

The json file was converted into a text file. Then I modified the format so it adapts to my Inventory and Transportation Management System (format: City, State).