School: Computer Science (Bsc)

Academic year: 2023/24

Student ID: 20364583

Student Name: Mar Zallan Bin Mohd Ismail Ganesan

Word count: 7172 Words

Category	Details
Assignment/Essay Title	Inventory and Transportation Management System development in C++ Coursework 1 Report
Module Code	COMP2034
Module Title	C++ Programming

Table of Contents

1. Introduction

1.1 Overview of the Project

In an era where transparency and efficiency in the plantation industry are increasingly vital, this project introduces a sophisticated Inventory and Transportation Management System (ITMS) developed using C++. Specifically designed for the unique dynamics of the plantation sector, this system incorporates pioneering blockchain technology to revolutionize how plantation products are tracked, managed, and distributed from origin to consumer. The ITMS is engineered to meet the specific challenges of the plantation supply chain, emphasizing traceability, quality control, and operational efficiency.

1.2 Importance of Blockchain in ITMS for the Plantation Industry

Blockchain technology offers transformative potential for the plantation industry, particularly in enhancing traceability and accountability within supply chains. Its immutable and transparent characteristics make it an ideal framework for establishing a trustworthy ledger of the journey of plantation products. By integrating blockchain into the ITMS, the system can markedly enhance product safety, streamline the tracing of products back to their source, and elevate consumer confidence by providing a verifiable record of product histories.

1.3 Blockchain Technology in the Plantation Industry's ITMS

This section will delve into the integration of blockchain technology within the context of the plantation industry's ITMS. It will cover how blockchain is effectively utilized to manage and record crucial data through various stages of the supply chain – from cultivation and harvesting, to processing, packaging, warehousing, transportation, and retail distribution. The focus will be on how blockchain technology fortifies data integrity, boosts supply chain visibility, and strengthens quality and safety standards, meeting the stringent requirements of the plantation industry.

2. Case Scenario: Sime Darby Plantation - Demonstrating Blockchain Application in ITMS

- **Agricultural Production**: As a major palm oil producer, Sime Darby Plantation engages in large-scale cultivation and harvesting of palm oil plantations. The use of blockchain can ensure the traceability of production practices, including the management of plantation resources, sustainability measures, and labor practices.
- **Processing and Refining**: The company operates numerous mills and refineries where raw palm oil is processed and refined. Implementing blockchain can enhance the transparency and efficiency of these processes, tracking every batch of palm oil from the plantation to the refinery.
- **Supply Chain and Logistics**: Sime Darby Plantation manages a complex supply chain that includes transportation to various parts of the world. Blockchain technology can be employed to optimize logistics, ensure product traceability during transit, and maintain integrity across the supply chain.

- **Quality Assurance and Certification**: Given the environmental and ethical concerns associated with palm oil production, blockchain can play a crucial role in documenting and verifying quality control measures, certification processes (e.g., RSPO certification), and compliance with environmental standards.
- **Market Distribution and Traceability**: The end-products of Sime Darby Plantation are distributed globally. Blockchain can provide end-to-end traceability, allowing consumers and stakeholders to verify the origin, processing, and quality of the palm oil products, thus enhancing consumer confidence and market reputation.
-

2.1 Blockchain Implementation in Sime Darby Plantation's Operations

Integrating blockchain technology into Sime Darby Plantation's operations could revolutionize how the company tracks and manages its product lifecycle. Blockchain's capabilities in ensuring data integrity and transparency would be particularly beneficial in addressing sustainability concerns and regulatory compliance. Moreover, it can streamline operational processes, reduce potential errors, and improve overall supply chain efficiency.

By showcasing a Malaysian entity like Sime Darby Plantation, this case study highlights the versatility and potential of blockchain technology in ITMS, especially in industries with complex supply chains and high demands for sustainability and transparency. It exemplifies how blockchain can be a game-changer, not just in technological innovation, but in promoting responsible and ethical business practices in Malaysia and beyond.

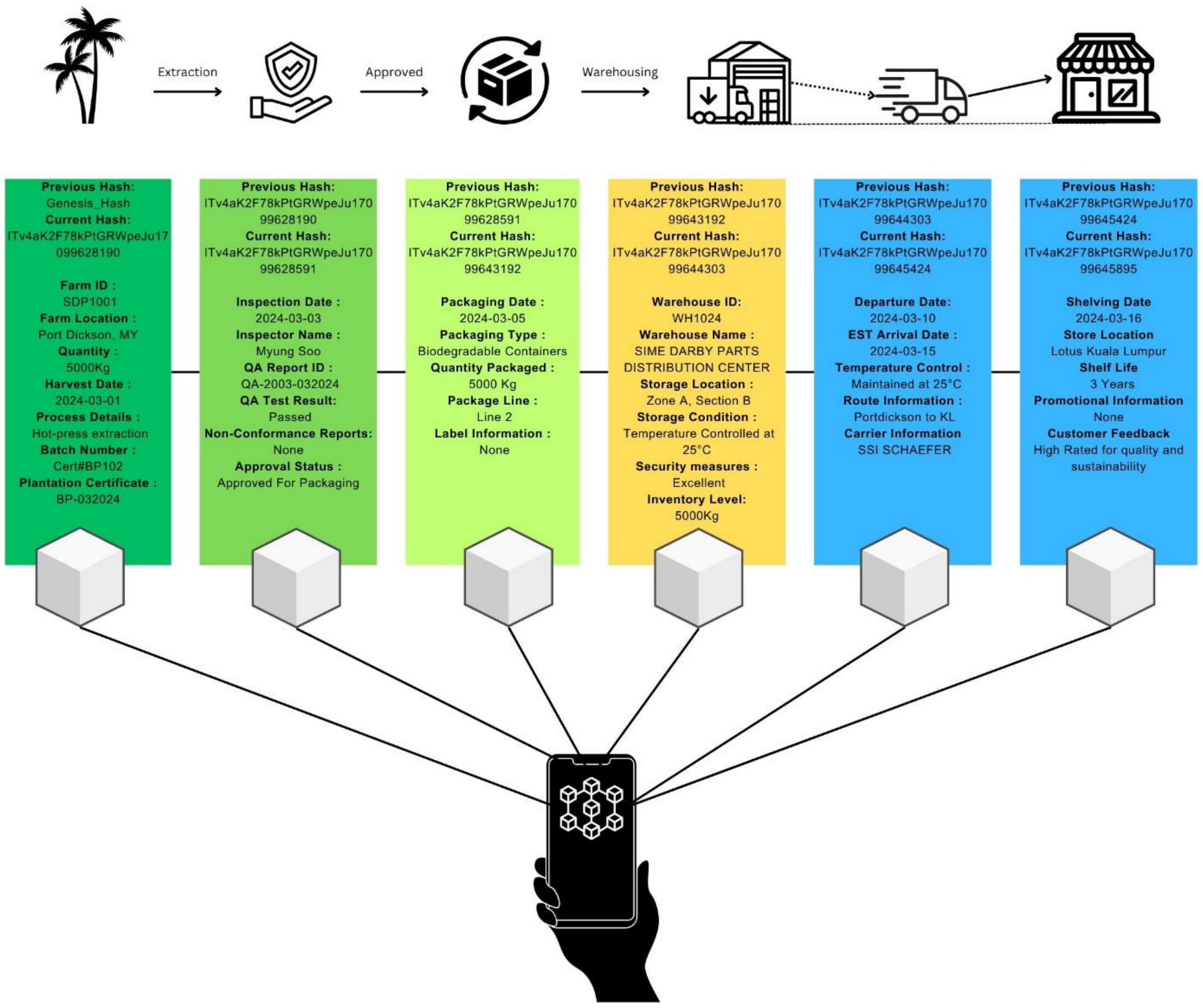
In this case scenario, each stage of Farm-to-Fork Foods' supply chain is enhanced by blockchain technology, demonstrating its value in improving the efficiency, transparency, and reliability of ITMS in the food industry.

2.2 Process of the Stages with Blockchain Integration

Stages	Functions & Descriptions
Stage 1: - **Farming/Agriculture**	Blockchain records data like harvest dates, and farming practices. Smart contracts for supplier payments upon meeting certain quality criteria.
Stage 2: - **Quality Assurance**	Quality inspection data, including health and safety reports, are recorded on the blockchain. Instant sharing of quality data with regulatory bodies and downstream supply chain participants.
Stage 3: - **Processing and Packaging**	Tracking of processing details and packaging dates.

	Recording of packaging types and nutritional information for consumer transparency.
Stage 4: - **Warehousing**	Real-time inventory levels and storage conditions are logged on to the blockchain. Improved warehouse management and stock allocation using blockchain data.
Stage 5 :- **Transportation**	Real-time tracking of palm oil during transit, including temperature control and route information. Smart contracts to release payments upon safe and timely delivery.
Stage 6: - **Retail and Distribution**	Shelf life, store location, and shelving dates are tracked for optimal shelf management. Collection of customer feedback through blockchain-based systems for product improvements

Table 1.0 Structure of stages in blockchain design.



Previous Hash:
Genesis_Hash
Current Hash:
ITv4aK2F78kPtGRWpeJu17
099628190

Farm ID :
SDP1001
Farm Location :
Port Dickson, MY
Quantity :
5000Kg
Harvest Date :
2024-03-01
Process Details :
Hot-press extraction
Batch Number :
Cert#BP102
Plantation Certificate :
BP-032024

Previous Hash:
ITv4aK2F78kPtGRWpeJu170
99628190
Current Hash:
ITv4aK2F78kPtGRWpeJu170
99628591

Inspection Date :
2024-03-03
Inspector Name :
Myung Soo
QA Report ID :
QA-2003-032024
QA Test Result:
Passed
Non-Conformance Reports:
None
Approval Status :
Approved For Packaging

Previous Hash:
ITv4aK2F78kPtGRWpeJu170
99628591
Current Hash:
ITv4aK2F78kPtGRWpeJu170
99643192

Packaging Date :
2024-03-05
Packaging Type :
Biodegradable Containers
Quantity Packaged :
5000 Kg
Package Line :
Line 2
Label Information :
None

Previous Hash:
ITv4aK2F78kPtGRWpeJu170
99643192
Current Hash:
ITv4aK2F78kPtGRWpeJu170
99644303

Warehouse ID:
WH1024
Warehouse Name :
SIME DARBY PARTS
DISTRIBUTION CENTER
Storage Location :
Zone A, Section B
Storage Condition :
Temperature Controlled at
25°C
Security measures :
Excellent
Inventory Level:
5000Kg

Previous Hash:
ITv4aK2F78kPtGRWpeJu170
99644303
Current Hash:
ITv4aK2F78kPtGRWpeJu170
99645424

Departure Date:
2024-03-10
EST Arrival Date :
2024-03-15
Temperature Control :
Maintained at 25°C
Route Information :
Portdickson to KL
Carrier Information
SSI SCHAEFER

Previous Hash:
ITv4aK2F78kPtGRWpeJu170
99645424
Current Hash:
ITv4aK2F78kPtGRWpeJu170
99645895

Shelving Date
2024-03-16
Store Location
Lotus Kuala Lumpur
Shelf Life
3 Years
Promotional Information
None
Customer Feedback
High Rated for quality and
sustainability

Diagram 1.1 Structure of blockchain implementation

3. Blockchain Design Structure

3.1 Composition of the Hash

Random String Component: Each hash starts with a 20-character string randomly generated from an alphanumeric set. This randomness is crucial for ensuring the uniqueness and security of each hash.

Time Stamp Integration: The current timestamp is appended to the hash. This not only helps in sequencing the blocks in chronological order but also plays a vital role in maintaining the uniqueness of each hash.

Counter Incrementation: A counter, hashCounter, is incremented with each new hash generation. This feature ensures that even when two hashes are generated at the same exact timestamp, they remain unique.

3.2 Enhancing Randomness and Security

Random Number Generation: The system uses random_device and mt19937 for generating random numbers. This combination enhances the unpredictability of the hash, which is a critical aspect of blockchain security.

Uniform Character Distribution: By implementing a uniform distribution for character selection, the system ensures that each character in the alphanumeric set has an equal probability of being chosen. This approach further strengthens the hash's robustness.

Seeding with High-Resolution Clock: The random number generator (mt19937) is seeded with a high-resolution clock (chrono::system_clock). This ensures that the seed is unique and variable, significantly enhancing the randomness of the hash.

```
static int hashCounter = 0; // Counter for hashes to ensure uniqueness

static random_device rd; // Non-deterministic random number generator
static mt19937 generator(rd()); // Mersenne Twister pseudo-random generator
```

```
string generateRandomHash() {
    // Function to generate a random hash
    const string chars =
"abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789";
    string randHash;

    unsigned seed = chrono::system_clock::now().time_since_epoch().count();
    static mt19937 generator(seed);

    uniform_int_distribution<> distribution(0, chars.size() - 1);

    for (int i = 0; i < 20; i++) {
        randHash += chars[distribution(generator)];
    }
    auto t = time(nullptr);
    randHash += to_string(t) + to_string(hashCounter++); // Append time and
counter to ensure uniqueness

    return randHash;
}
```

3.3 Login Functionality in the ITMS (Information Technology Management System)

Authentication Requirement: Access to the system is granted only after successful authentication. Users are required to input valid credentials, including a username and password.

Limited Login Attempts: To enhance security, the system allows a limited number of login attempts (typically three). If the maximum number of attempts is reached, access is denied, and the system exits. This feature is crucial to prevent unauthorized access to sensitive data within the supply chain.

```cpp
// Simple login function with hardcoded credentials
void login(){
    string username;
    string password;
    int tries = 3;
    bool success = false;
    while(tries && !success){
        cout << "Enter Username" << endl;
        cin >> username;
        cout << "Enter Password" << endl;
        cin >> password;

        if(username == "admin" && password == "admin"){
            system("cls"); // Clears the console screen
            cout << "You have got access. \n";
            success = true;
        }
        else{
            tries --;
            system("cls"); // Clears the console screen
            cout << "Wrong Authentication" << endl;
            if(tries == 0){
                system("cls"); // Clears the console screen
                cout << "Failed to authenticate" << endl;
                cout << "System Exiting" << sleep(1) << "*" << sleep(1) << "*"
<< sleep(1) << "*";
                exit(0); // Exits the program after 3 failed login attempts
            }
        }
    }
}
```

4. Components and Structures

4.1 Header Files and Namespaces:

The program includes standard library header files for functionalities such as input/output operations, random number generation, file handling, and date/time management. It uses the standard namespace (**std**).

4.2 Enumerations and Structures:

enum Stage: An enumeration representing different stages in a supply chain (Manufacturing, Quality Assurance, etc.).

struct Block: A structure representing a block in the blockchain, containing an identifier, hash values, timestamp, and additional information.

4.3 File Handling Functions:

listTxtFiles(): Scans the current directory for **.txt** files and returns a vector containing their names.

```cpp
// Function to list all .txt files in the current directory
vector<string> listTxtFiles() {
    vector<string> files;
    WIN32_FIND_DATA fileData;
    HANDLE hFind = FindFirstFile("./*.txt", &fileData); // Search for .txt
files

    // Loop through all found files and add their names to the vector
    if (hFind != INVALID_HANDLE_VALUE) {
        do {
            files.push_back(fileData.cFileName);
        } while (FindNextFile(hFind, &fileData) != 0);
        FindClose(hFind);
    }
    return files;
}
```

writeBlockToFile(): Takes a **Block** object and a filename, and writes the block's details to a **.txt** file in a structured format.

```cpp
// Function to append a block's data to a file
void writeBlockToFile(const Block& block, const string& filename) {
    string filePath = filename + ".txt";

    // Open file in append mode
    ofstream outFile(filePath, ios::app);
    if (outFile.is_open()) {
        // Write formatted block data to file
        outFile << "Stage:" << block.id << ",";
        outFile << "BlockID:" << block.id << ",";
        outFile << "CurrentHash:" << block.current_hash << ",";
        outFile << "PreviousHash:" << block.previous_hash << ",";
        outFile << "Time:" << block.timestamp << ",";
        outFile << "Info:" << block.info << "\n";
        outFile.close();
    } else {
        // Error handling if file cannot be opened
        cerr << "Unable to open file: " << filePath << endl;
    }
}
```

displayBlockFromFile(): Displays block information from a file.

```cpp
// Function to display block information from a selected file
void displayBlockFromFile(const string& selectedFile) {
    ifstream inFile(selectedFile);
    if (!inFile.is_open()) {
        // Error handling if file cannot be opened
        cerr << "Error opening file: " << selectedFile << endl;
        return;
    }

    string line;
    Block block;
    stringstream dataStream;
    bool isFirstBlock = true;

    // Read file line by line
    while (getline(inFile, line)) {
        // Check if the line starts with "Stage:" indicating a new block
        if (line.rfind("Stage:", 0) == 0) {
            if (!isFirstBlock) {
                // Process and display the previous block before starting a
new one
                block.info = dataStream.str();
                printBlock(block);

                // Clear stringstream for the next block
                dataStream.str("");
                dataStream.clear();
                block = Block();

                // Prompt user to press enter to view the next block
                cout << "\nPress enter to view next block...";
                cin.ignore(numeric_limits<streamsize>::max(), '\n');
                getchar();
                system("cls"); // Clear console screen
                cout << "\n";
            }
            isFirstBlock = false;
            istringstream headerStream(line);
            string token;
            // Parse block header information
            while (getline(headerStream, token, ',')) {
                size_t colonPos = token.find(':');
                if (colonPos != string::npos) {
                    string key = token.substr(0, colonPos);
                    string value = token.substr(colonPos + 1);
                    // Assign values to the block based on the key
                    if (key == "Stage") block.id = stoi(value);
                    else if (key == "BlockID") block.id = stoi(value);
                    else if (key == "CurrentHash") block.current_hash = value;
                    else if (key == "PreviousHash") block.previous_hash =
value;
```

```
                    else if (key == "Time") block.timestamp = value;
                }
            }
        } else {
            // Collect block information content
            dataStream << line << "\n";
        }
    }

    // Display the last block if there is any data left
    if (!dataStream.str().empty()) {
        block.info = dataStream.str();
        printBlock(block);
    }

    inFile.close();
}
```

4.4 Utility Functions:

generateRandomHash(): Generates a random hash string combining random characters, the current time, and the **hashCounter**.

```
string generateRandomHash() {
    // Function to generate a random hash
    const string chars =
"abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789";
    string randHash;

    unsigned seed = chrono::system_clock::now().time_since_epoch().count();
    static mt19937 generator(seed);

    uniform_int_distribution<> distribution(0, chars.size() - 1);

    for (int i = 0; i < 20; i++) {
        randHash += chars[distribution(generator)];
    }
    auto t = time(nullptr);
    randHash += to_string(t) + to_string(hashCounter++); // Append time and
counter to ensure uniqueness

    return randHash;
}
```

printBlock():
Outputs the details of a **Block** to the console in a formatted manner.

```
void printBlock(const Block& block) {
    // Function to print block information in a formatted manner
    cout << "--------------------- " << "Stage " << block.id << " ------------
---------" << endl;
    cout << "\n";
    cout << "Block ID      : " << block.id << endl;
    cout << "--------------" << endl;
    cout << "Timestamp     : " << block.timestamp << endl;
    cout << "--------------" << endl;
    cout << "Previous Hash : " << block.previous_hash << endl;
    cout << "--------------" << endl;
    cout << "Current Hash  : " << block.current_hash << endl;
    cout << "--------------" << endl;
    cout << block.info << endl;
    cout << "\n---------------------------------------------------" << endl;
}
```

stageToString():
Converts a **Stage** enum value to its corresponding string representation.

```
// Converts the Stage enumeration to a human-readable string
string stageToString(Stage stage) {
    switch (stage) {
        case Manufacturing: return "Manufacturing";
        case QualityAssurance: return "Quality Assurance";
        case Packaging: return "Packaging";
        case Warehousing: return "Warehousing";
        case Transportation: return "Transportation";
        case RetailShelving: return "Retail Shelving";
        default: return "Unknown";
    }
}
```

getcurrentTime():
Retrieve the current time of user's system.

```
string getCurrentTime() {
    // Function to get current time formatted as YYYY-MM-DD HH:MM:SS
    time_t rawtime;
    struct tm *timeinfo;
    char buffer[80];
    time(&rawtime);
    timeinfo = localtime(&rawtime);
    strftime(buffer, 80, "%Y-%m-%d %H:%M:%S", timeinfo);
    return string(buffer);
}
```

addBlock():
Adds a new block to the blockchain to the structure.

```
// Adds a new block to the blockchain vector, based on the given stage
void addBlock(vector<Block>& blockchain, int stage) {
    Block newBlock;
    newBlock.id = blockchain.empty() ? 1 : blockchain.back().id + 1; // Sets
the block ID
    newBlock.info = collectData(static_cast<Stage>(stage),false); // Collects
stage-specific data
    newBlock.previous_hash = blockchain.empty() ? "genesis_hash" :
blockchain.back().current_hash; // Sets the previous hash
    newBlock.current_hash = generateRandomHash(); // Generates a random hash
for the current block
    newBlock.timestamp = getCurrentTime(); // Sets the current timestamp
    blockchain.push_back(newBlock); // Adds the new block to the blockchain
}
```

4.5 Search blockchain

searchForBlockByHashPattern(): In the development of the blockchain application, a critical feature was implemented to enhance user interaction and data accessibility: the ability to search for blockchain blocks by a specific hash pattern. This functionality allows users to input the first 2 and last 4 characters of a block's hash, facilitating efficient retrieval of blockchain information without requiring the full hash value. Upon executing a search, the system iterates through stored blockchain data, comparing the provided hash pattern against existing blocks. If a match is found, the corresponding block's information is displayed to the user.

```
void searchForBlockByHashPattern(const string& hashPattern) {
    vector<string> txtFiles = listTxtFiles();

    for (const auto& file : txtFiles) {
        ifstream inFile(file);
        if (!inFile.is_open()) {
            cerr << "Error opening file: " << file << endl;
            continue;
        }

        string line;
        Block block;
        stringstream dataStream;
        bool isBlockInitialized = false;

        while (getline(inFile, line)) {
            if (line.rfind("Stage:", 0) == 0) { // Start of a new block
                if (isBlockInitialized) { // If a previous block was being
processed
                    block.info = dataStream.str(); // Assign the accumulated
info to the block
```

```
                    // Check the hash pattern of the previous block
                    if (block.current_hash.substr(0, 2) ==
hashPattern.substr(0, 2) &&
                        block.current_hash.substr(block.current_hash.size() -
4) == hashPattern.substr(hashPattern.size() - 4)) {

                        printBlock(block);
                        cout << "Found in file: " << file << "\n";
                        inFile.close();
                        return; // Stop after finding the match
                    }

                    dataStream.str(""); // Clear stringstream for the next
block
                    dataStream.clear();
                    block = Block(); // Reset block for the next one
                }

                isBlockInitialized = true; // Mark that we've started
processing a block

                // Parse the line into block attributes
                istringstream headerStream(line);
                string token;
                while (getline(headerStream, token, ',')) {
                    size_t colonPos = token.find(':');
                    if (colonPos != string::npos) {
                        string key = token.substr(0, colonPos);
                        string value = token.substr(colonPos + 1);
                        if (key == "Stage") ; // Not used directly for search
                        else if (key == "BlockID") block.id = stoi(value);
                        else if (key == "CurrentHash") block.current_hash =
value;
                        else if (key == "PreviousHash") block.previous_hash =
value;
                        else if (key == "Time") block.timestamp = value;
                    }
                }
            } else {
                // Accumulate block information content
                dataStream << line << "\n";
            }
        }

        // After finishing the file, check the last block
        if (isBlockInitialized) {
            block.info = dataStream.str();
            if (block.current_hash.substr(0, 2) == hashPattern.substr(0, 2) &&
                block.current_hash.substr(block.current_hash.size() - 4) ==
hashPattern.substr(hashPattern.size() - 4)) {
                printBlock(block);
                cout << "Found in file: " << file << "\n";
                return; // Found the matching block
            }
        }
        inFile.close();
    }

    cout << "No block found with the specified hash pattern." << endl;
}
```

4.6 Data Collection

collectData(): Depending on the stage parameter, the function prompts the user for different sets of data relevant to that stage. For example, for Manufacturing, it asks for details like Manufacturer ID, Product Name, etc. If isFileInput is true, the function parses the passed data string, extracting relevant information based on the stage.

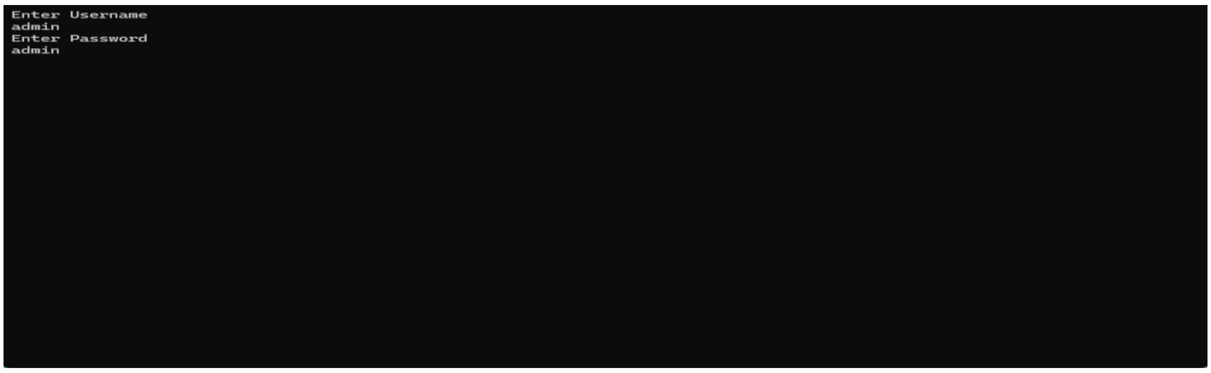
```
string collectData(Stage stage, bool isFileInput, const string& data = "")
{
    stringstream dataStream;
    vector<string> tokens;
    if (isFileInput) {
        tokens = splitString(data, ';');
    }
    switch (stage) {
        case Manufacturing:
            {
                string farmID, farmLocation, quantity, harvestDate,
practiceDetails, plantationCertificate, batchNumber;
                if (isFileInput && tokens.size() >= 7) {
                    farmID = tokens[0];
                    farmLocation = tokens[1];
                    quantity = tokens[2];
                    harvestDate = tokens[3];
                    practiceDetails = tokens[4];
                    plantationCertificate = tokens[5];
                    batchNumber =  tokens[6];
                } else {
                    cout << "Enter Farm ID                : ";
                    getline(cin, farmID);
                    cout << "Enter Farm Location          : ";
                    getline(cin, farmLocation);
                    cout << "Enter Quantity               : ";
                    getline(cin, quantity);
                    cout << "Enter Harvest Date           : ";
                    getline(cin, harvestDate);
                    cout << "Enter Process Details        : ";
                    getline(cin, practiceDetails);
                    cout << "Enter Plantation Certificate : ";
                    getline(cin, plantationCertificate);
                    cout << "Enter Batch Number           : ";
                    getline(cin, batchNumber);
                }
                dataStream
                    << "\nFarm ID                 : " << farmID
                    << "\nFarm Location           : " << farmLocation
                    << "\nQuantity                : " << quantity
                    << "\n"
                    << "\nHarvest Date            : " << harvestDate
                    << "\nProcess Details         : " << practiceDetails
                    << "\n"
                    << "\nBatch Number            : " << batchNumber
                    << "\nPlantation Certificate  : " <<
plantationCertificate;
            }
            break;
        case QualityAssurance:
            {
                string inspectionDate, inspectorName, QAReportID,
QAResultTest, nonConformanceReports, approvalStatus;
                if (isFileInput && tokens.size() >= 6) {
                    inspectionDate = tokens[0];
                    inspectorName = tokens[1];
                    QAReportID = tokens[2];
                    QAResultTest = tokens[3];
                    nonConformanceReports = tokens[4];
                    approvalStatus = tokens[5];
                }
                else{
                    cout << "Enter Inspection Date               : ";
                    getline(cin, inspectionDate);
                    cout << "Enter Inspector Name                : ";
                    getline(cin, inspectorName);
                    cout << "Enter Report ID                     : ";
                    getline(cin, QAReportID);
                    cout << "Enter Quality Test Result           : ";
                    getline(cin, QAResultTest);
                    cout << "Enter Non-Conformance Reports       : ";
                    getline(cin, nonConformanceReports);
                    cout << "Enter Approval Status               : ";
                    getline(cin, approvalStatus);
                }
                dataStream
                << "\nInspection Date         : " << inspectionDate
                << "\nInspector Name          : " << inspectorName
                << "\n"
                << "\nQA Report ID            : " << QAReportID
                << "\nQA Test Result          : " << QAResultTest
                << "\nNon-Conformance Reports : " << nonConformanceReports
                << "\nApproval Status         : " << approvalStatus;
            }
            break;
        case Packaging:
            {
                string packageDate, packageType, quantityPackaged,
packageLine, labelInfo;
                if (isFileInput && tokens.size() >= 5) {
                    packageDate= tokens[0];
                    packageType = tokens[1];
                    quantityPackaged = tokens[2];
                    packageLine = tokens[3];
                    labelInfo = tokens[4];
                }
                else{
                    cout << "Enter Packaging Date       : ";
                    getline(cin, packageDate);
                    cout << "Enter Packaging Type       : ";
                    getline(cin, packageType);
                    cout << "Enter Quantity Packaged    : ";
                    getline(cin, quantityPackaged);
                    cout << "Enter Package Line         : ";
                    getline(cin, packageLine);
                    cout << "Enter Label Information    : ";
                    getline(cin, labelInfo);
                }
                dataStream
                << "\nPackaging Date       : " << packageDate
                << "\nPackaging Type       : " << packageType
                << "\n"
                << "\nQuantity Packaged    : " << quantityPackaged
                << "\nPackage Line         : " << packageLine
                << "\n"
                << "\nLabel Information    : " << labelInfo;
            }
            break;
        case Warehousing:
            {
                string warehouseID, warehouseName, storageLocation,
storageCondition, securityMeasures, inventoryLevel;
                if (isFileInput && tokens.size() >= 6) {
                    warehouseID = tokens[0];
                    warehouseName = tokens[1];
                    storageLocation = tokens[2];
                    storageCondition = tokens[3];
                    securityMeasures = tokens[4];
                    inventoryLevel = tokens[5];
                }
                else{
                    cout << "Enter Warehouse ID          : ";
                    getline(cin, warehouseID);
                    cout << "Enter Warehouse Name        : ";
                    getline(cin, warehouseName);
                    cout << "Enter Warehouse Location    : ";
                    getline(cin, storageLocation);
                    cout << "Enter Security Measures     : ";
                    getline(cin, securityMeasures);
                    cout << "Enter Storage Condition     : ";
                    getline(cin, storageCondition);
                    cout << "Enter Inventory Level       :  ";
                    getline(cin, inventoryLevel);
                }
                dataStream
                << "\nWarehouse ID        : " << warehouseID
                << "\nWarehouse Name      : " << warehouseName
                << "\n"
                << "\nStorage Location    : " << storageLocation
                << "\nStorage Condition   : " << storageCondition
                << "\n"
                << "\nSecurity Measures   : " << securityMeasures
                << "\nInventory Level     : " << inventoryLevel;
            }
            break;
        case Transportation:
            {
                string transport, departDate, estArrivalDate,
temperatureControl, routeInformation, carrierInformation;
                if (isFileInput && tokens.size() >= 6) {
                    transport = tokens[0];
                    departDate = tokens[1];
                    estArrivalDate = tokens[2];
                    temperatureControl = tokens[3];
                    routeInformation = tokens[4];
                    carrierInformation = tokens[5];
                }
                else{
                    cout << "Enter Transport
Type                                                                   : ";
                    getline(cin, transport);
                    cout << "Enter Departure
Date                                                                   : ";
```

```
                    getline(cin, departDate);
                    cout << "Enter Estimated Arrival
Date                                                           : ";
                    getline(cin, estArrivalDate);
                    cout << "Enter temperature
Control                                                              : ";
                    getline(cin, temperatureControl);
                    cout << "Enter Route Information (including transit
points)                                     : ";
                    getline(cin, routeInformation);
                    cout << "Enter Carrier Information (including
sustainability practices and compliance records) : ";
                    getline(cin, carrierInformation);
                }
                dataStream
                << "Transport Type           :  " << transport
                << "\n"
                << "\nDeparture Date         : " << departDate
                << "\nEstimated Arrival Date : " << estArrivalDate
                << "\n"
                << "\nTemperature Control    : " << temperatureControl
                << "\nRoute Information      : " << routeInformation
                << "\nCarrier Information    : " << carrierInformation;
            }
            break;
        case RetailShelving:
            {
                string shelvingDate, storeLocation, shelfLife,
promotionalInfo, customerFeedback;
                if (isFileInput && tokens.size() >= 5) {
                    shelvingDate = tokens[0];
                    storeLocation = tokens[1];
                    shelfLife = tokens[2];
                    promotionalInfo = tokens[3];
                    customerFeedback = tokens[4];
                }
                else{
                    cout << "Enter Shelving date           : ";
                    getline(cin, shelvingDate);
                    cout << "Enter Store Location          : ";
                    getline(cin, storeLocation);
                    cout << "Enter Shelf Life              : ";
                    getline(cin, shelfLife);
                    cout << "Enter Promotional Information : ";
                    getline(cin, promotionalInfo);
                    cout << "Enter Customer Feedback       : ";
                    getline(cin, customerFeedback);
                }
                dataStream
                    << "\nShelving date           :  " << shelvingDate
                    << "\nStore Location          : " << storeLocation
                    << "\nShelf Life              : " << shelfLife
                    << "\n"
                    << "\nPromotional Information : " << promotionalInfo
                    << "\n"
                    << "\nCustomer Feedback       : " << customerFeedback;
            }
            break;
        default:
            cout << "Unknown stage";
            return "";
    }
    return dataStream.str();
}
```

5. Program, Extraordinary Features, and Functionalities

5.1. User Authentication:

The program begins with a call to the **login** function, ensuring that only authenticated users can access the system.



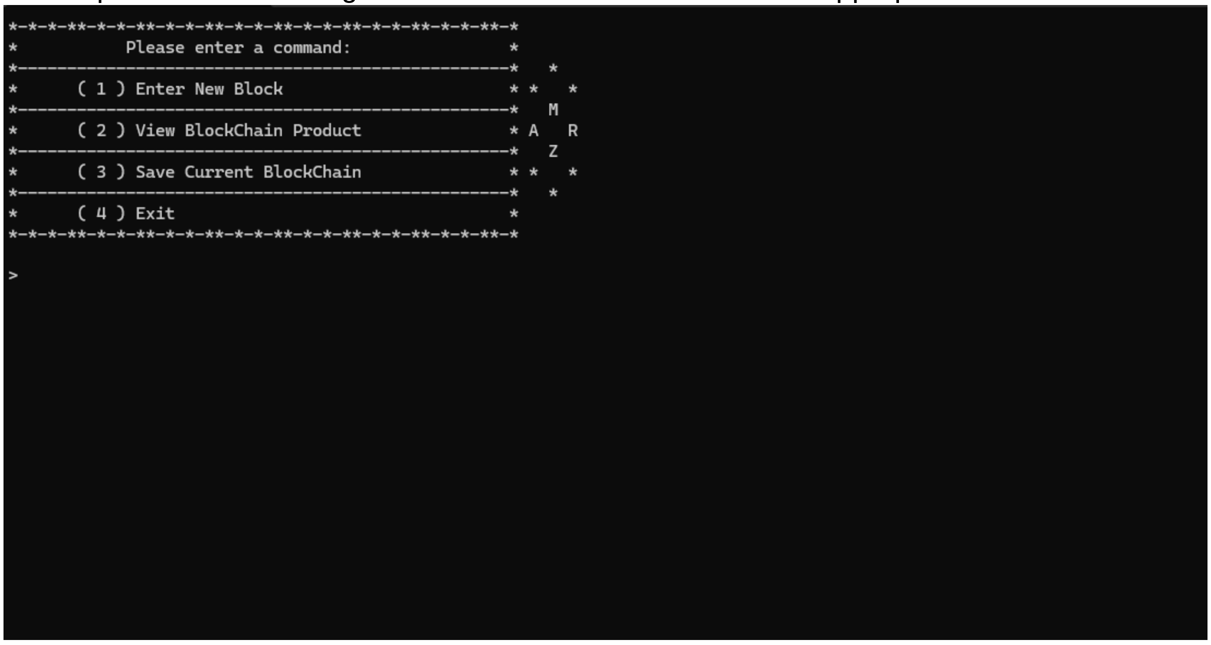
```
Enter Username
admin
Enter Password
admin
```

Image 2.0 Login Menu

5.2. Main Command Loop:

The program enters a loop, presenting a command menu to the user. This menu offers various options such as adding new blocks, viewing the blockchain, saving the blockchain to a file, and exiting the program.

User input is handled using a **switch** statement to execute the appropriate command.



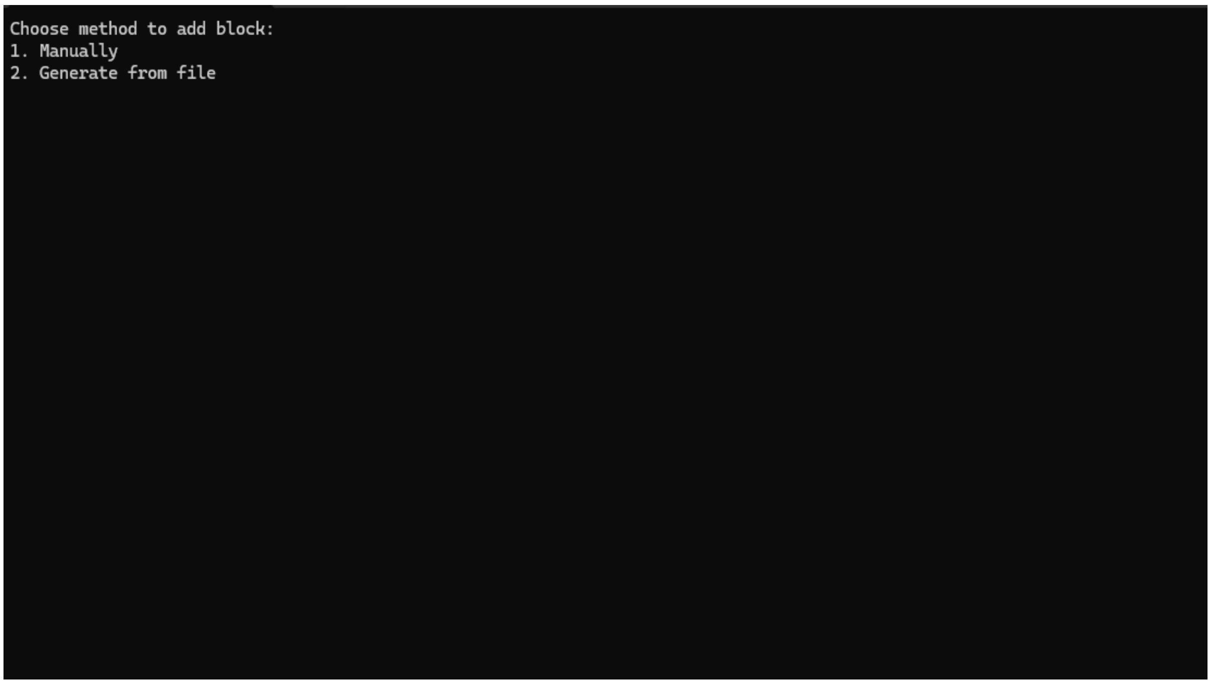
```
*-*-*-**-*-*-**-*-*-**-*-*-**-*-*-**-*-*-**-*-*-**-*
*           Please enter a command:                 *
*--------------------------------------------------*   *
*      ( 1 ) Enter New Block                       * *   *
*--------------------------------------------------*   M
*      ( 2 ) View BlockChain Product               * A   R
*--------------------------------------------------*   Z
*      ( 3 ) Save Current BlockChain               * *   *
*--------------------------------------------------*   *
*      ( 4 ) Exit                                  *
*-*-*-**-*-*-**-*-*-**-*-*-**-*-*-**-*-*-**-*-*-**-*

>
```

Image 2.1 Main Menu

5.3. Command Execution:

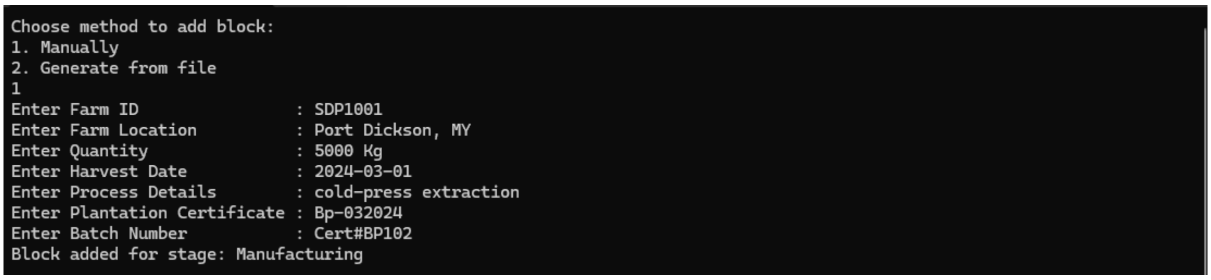
5.3.1 Add New Block:

The user can choose to manually add a new block or generate one from a file. In either case, the block is added to the **blockchain** vector, and its details are displayed.



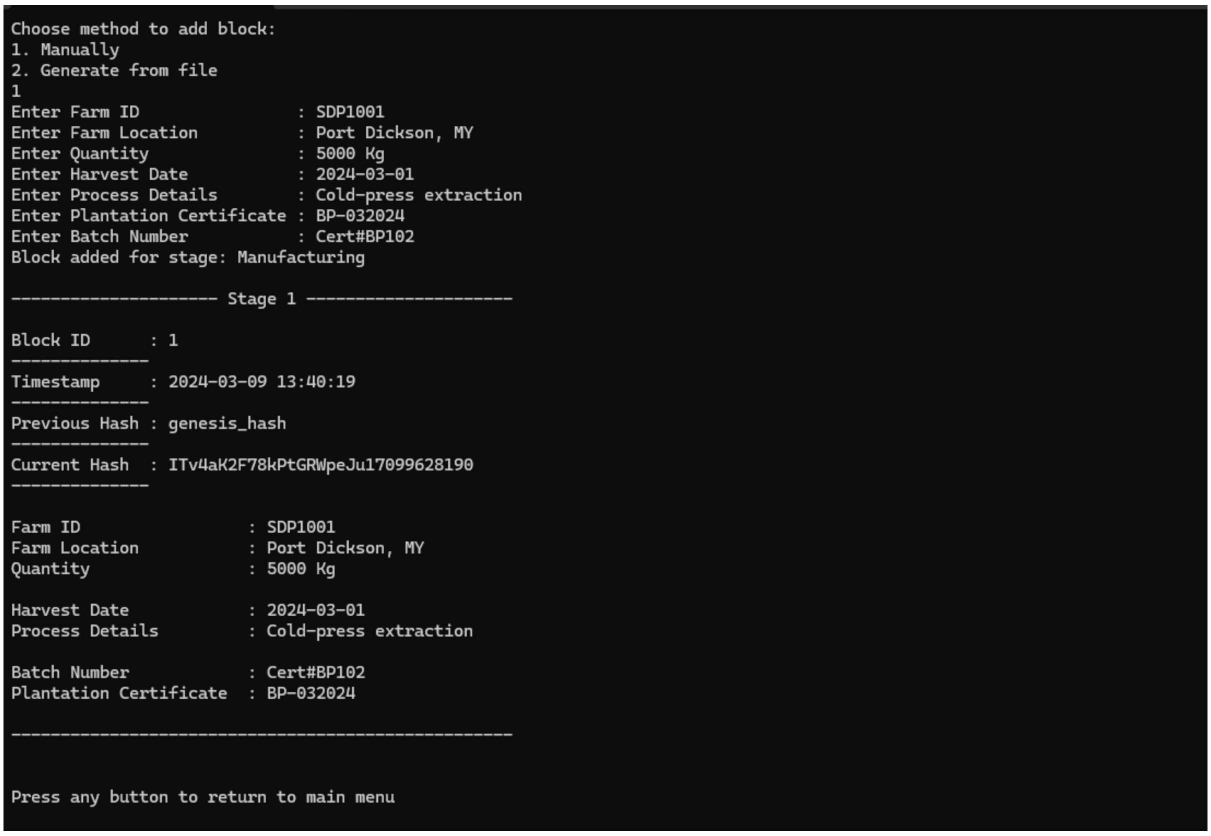
```
Choose method to add block:
1. Manually
2. Generate from file
```

Image 2.2.0 Add block menu.



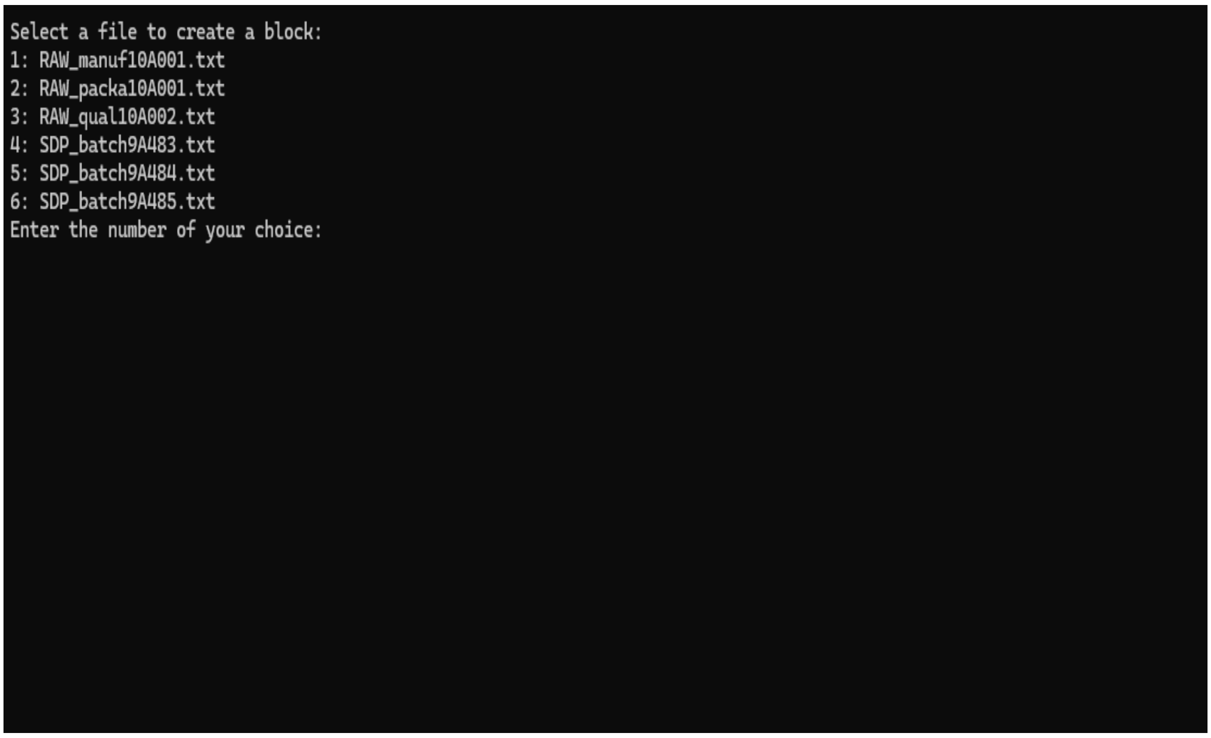
```
Choose method to add block:
1. Manually
2. Generate from file
1
Enter Farm ID                : SDP1001
Enter Farm Location          : Port Dickson, MY
Enter Quantity               : 5000 Kg
Enter Harvest Date           : 2024-03-01
Enter Process Details        : cold-press extraction
Enter Plantation Certificate : Bp-032024
Enter Batch Number           : Cert#BP102
Block added for stage: Manufacturing
```

Image 2.2.1 Block Info manufacturer



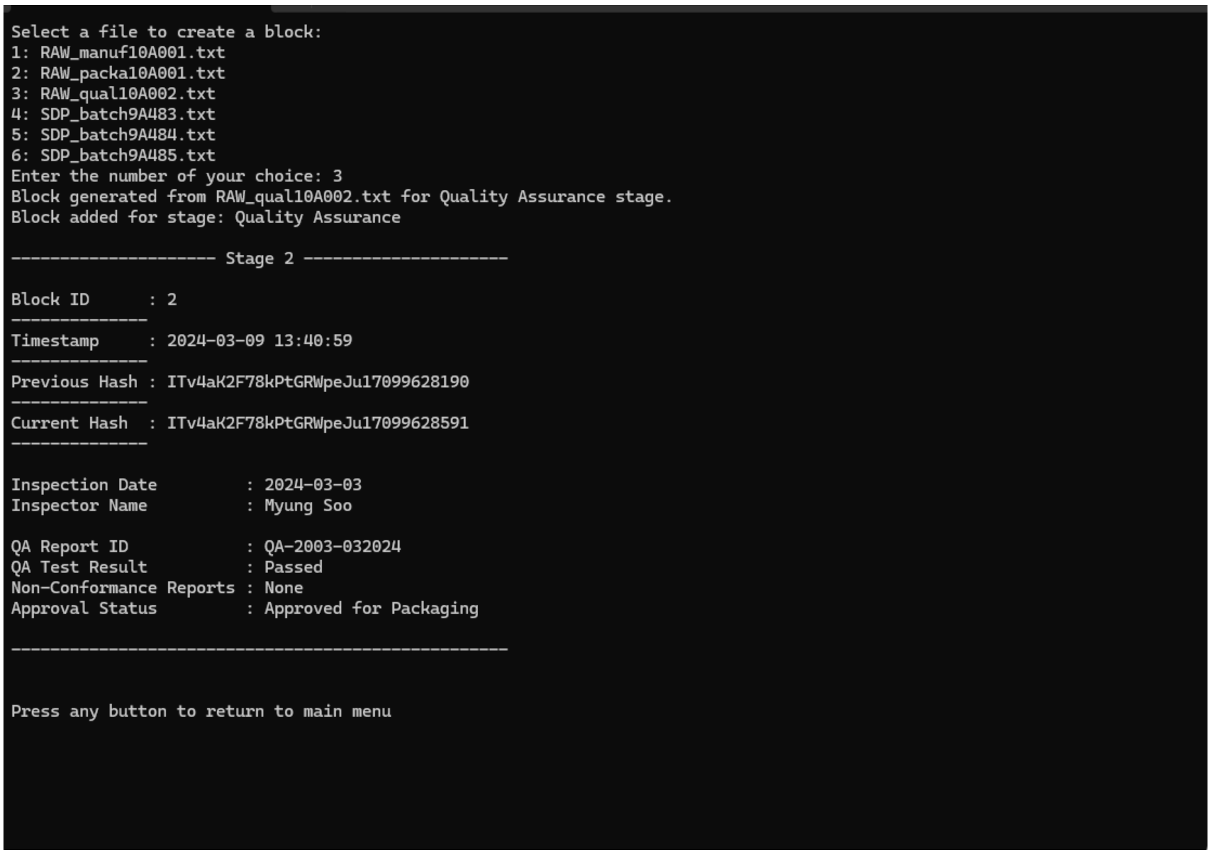
```
Choose method to add block:
1. Manually
2. Generate from file
1
Enter Farm ID                : SDP1001
Enter Farm Location          : Port Dickson, MY
Enter Quantity               : 5000 Kg
Enter Harvest Date           : 2024-03-01
Enter Process Details        : Cold-press extraction
Enter Plantation Certificate : BP-032024
Enter Batch Number           : Cert#BP102
Block added for stage: Manufacturing

--------------------- Stage 1 ---------------------

Block ID      : 1
--------------
Timestamp     : 2024-03-09 13:40:19
--------------
Previous Hash : genesis_hash
--------------
Current Hash  : ITv4aK2F78kPtGRWpeJu17099628190
--------------

Farm ID                 : SDP1001
Farm Location           : Port Dickson, MY
Quantity                : 5000 Kg

Harvest Date            : 2024-03-01
Process Details         : Cold-press extraction

Batch Number            : Cert#BP102
Plantation Certificate  : BP-032024

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


Press any button to return to main menu
```

Image 2.2.2 Completion of information entered for manufacturing stage.



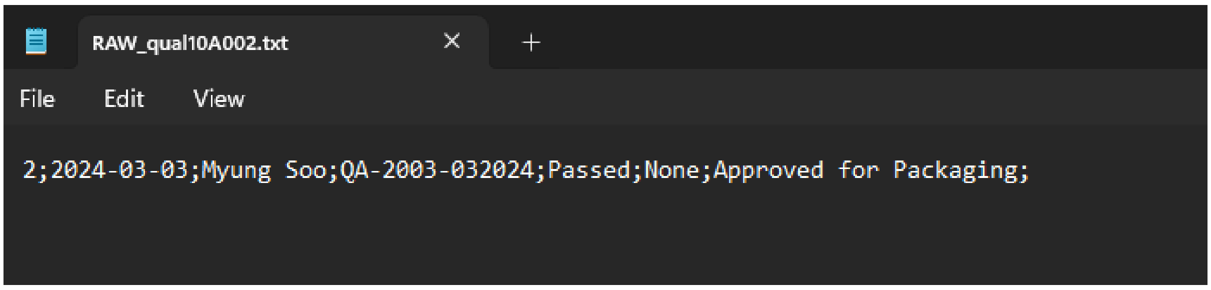
```
Select a file to create a block:
1: RAW_manuf10A001.txt
2: RAW_packa10A001.txt
3: RAW_qual10A002.txt
4: SDP_batch9A483.txt
5: SDP_batch9A484.txt
6: SDP_batch9A485.txt
Enter the number of your choice:
```

Image 2.2.3 User selection for generate from file menu.



```
Select a file to create a block:
1: RAW_manuf10A001.txt
2: RAW_packa10A001.txt
3: RAW_qual10A002.txt
4: SDP_batch9A483.txt
5: SDP_batch9A484.txt
6: SDP_batch9A485.txt
Enter the number of your choice: 3
Block generated from RAW_qual10A002.txt for Quality Assurance stage.
Block added for stage: Quality Assurance

--------------------- Stage 2 ---------------------

Block ID      : 2
--------------
Timestamp     : 2024-03-09 13:40:59
--------------
Previous Hash : ITv4aK2F78kPtGRWpeJu17099628190
--------------
Current Hash  : ITv4aK2F78kPtGRWpeJu17099628591
--------------

Inspection Date         : 2024-03-03
Inspector Name          : Myung Soo

QA Report ID            : QA-2003-032024
QA Test Result          : Passed
Non-Conformance Reports : None
Approval Status         : Approved for Packaging

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


Press any button to return to main menu
```

Image 2.2.3 Generate blockchain after user select (3) File.



```
2;2024-03-03;Myung Soo;QA-2003-032024;Passed;None;Approved for Packaging;
```

Image 2.2.4 File RAW_qual10A002 (3) raw data.

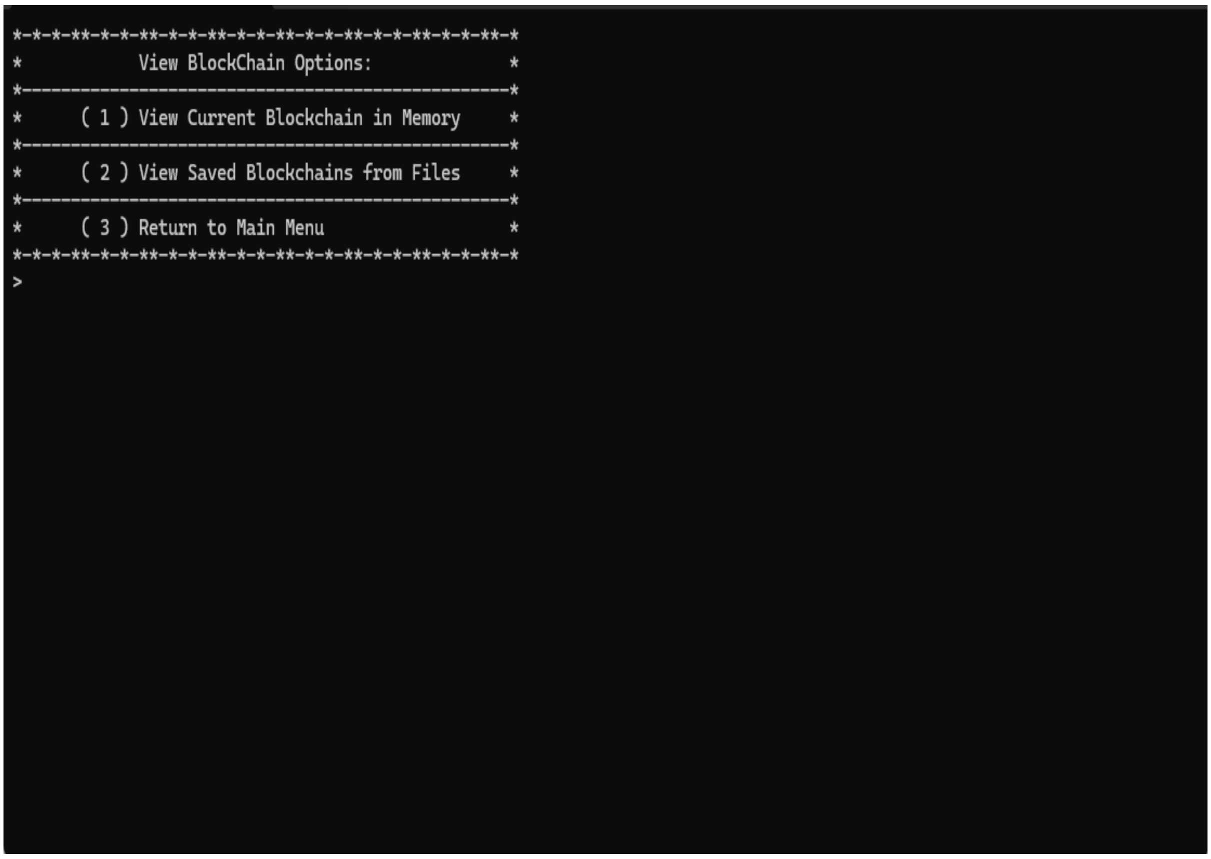
5.3.2 Block Entry Explanation Using File

The data entry for the stages of the supply chain is structured in a specific format to ensure all relevant information is captured accurately and systematically. *Image 2.2.4* Is an example of entry for the QA stage. Each segment of this entry is separated by a semicolon (**;**) and represents a specific piece of information vital to the QA process. Below is a breakdown of each part of this data entry:

1. **Block ID (2)**: This is the identifier for the block in the blockchain, signifying that this is the second block in the chain. It is crucial for maintaining the sequence and integrity of the blockchain.
2. **Inspection Date (2024-03-03)**: The date when the quality assurance inspection took place. This is important for tracking and historical records.
3. **Inspector Name (Myung Soo)**: The name of the individual who conducted the QA inspection. This provides accountability and traceability in the quality assurance process.
4. **QA Report ID (QA-2003-032024)**: A unique identifier for the QA report. This ID helps in referencing and retrieving the specific QA report for this batch.
5. **QA Test Result (Passed)**: The outcome of the quality assurance test. This indicates whether the product met the required quality standards.
6. **Non-Conformance Reports (None)**: Any reports of non-conformance or issues detected during the QA process. In this case, 'None' indicates no issues were found.
7. **Approval Status (Approved for Packaging)**: The final decision regarding the product's status post-inspection. Here, it indicates that the product has passed QA and is approved for the next stage, which is packaging.

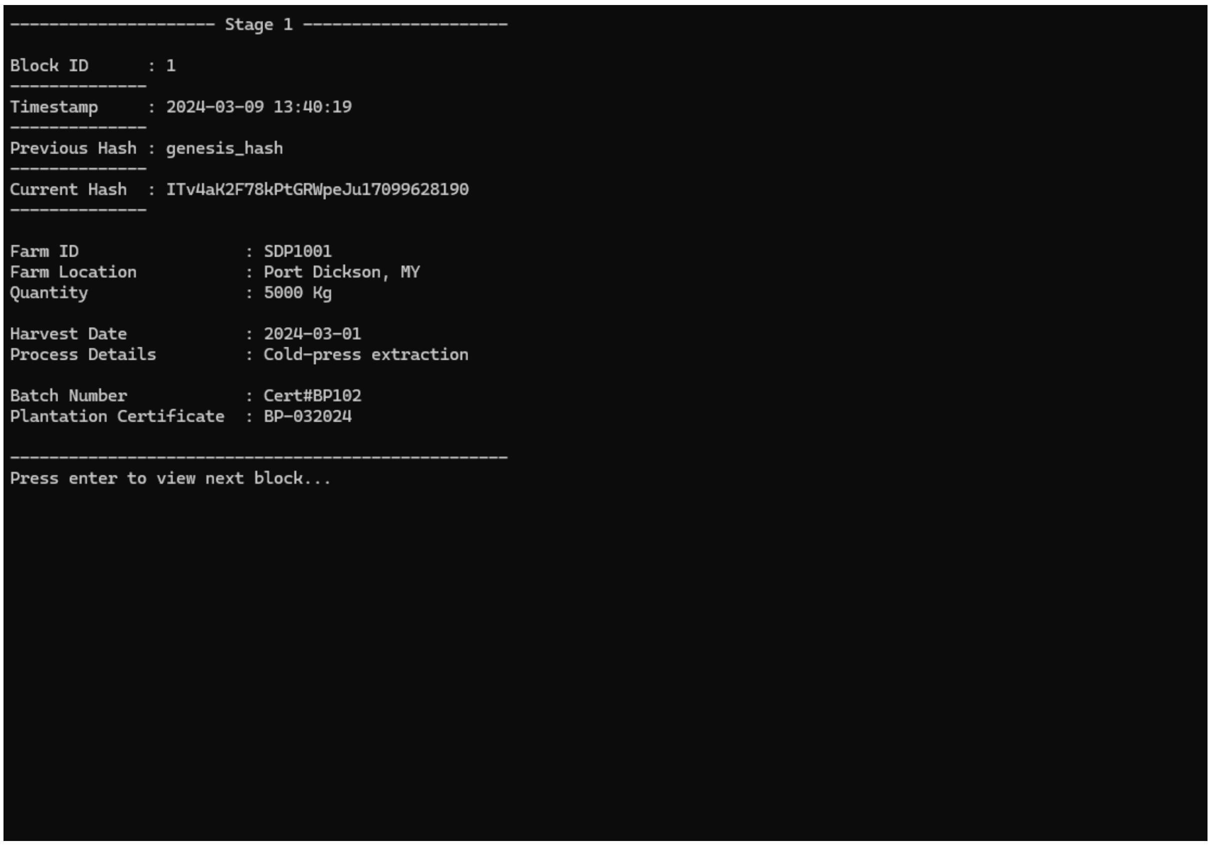
5.3.3 View Blockchain

Users can view the current blockchain in memory or view saved blockchains from files. Viewing the blockchain, a core functionality in the Inventory and Transportation Management System (ITMS) developed in C++, is integral for several reasons, particularly in the contexts of auditing and tracking. This functionality enhances transparency, accountability, and data integrity within the supply chain management process.



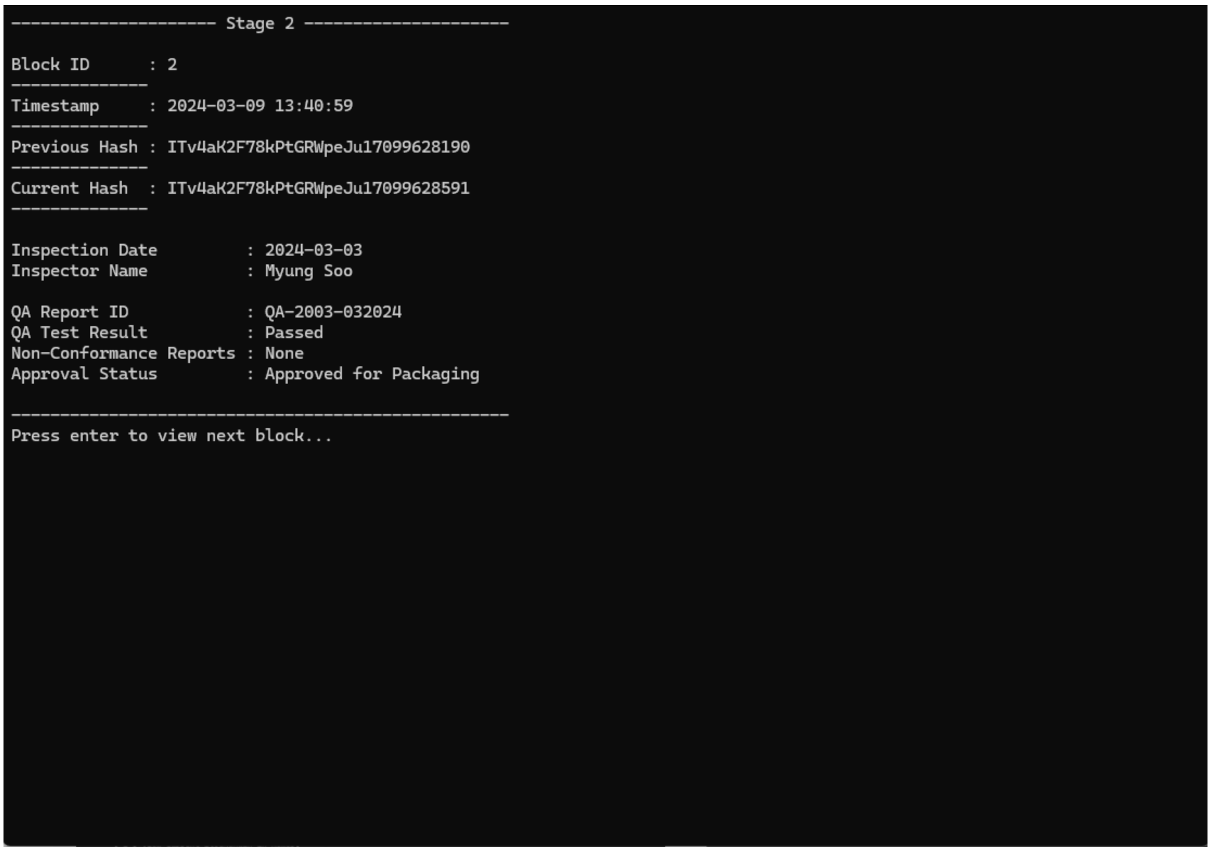
```
*-*-*-**-*-*-**-*-*-**-*-*-**-*-*-**-*-*-**-*-*-**-*
*           View BlockChain Options:               *
*--------------------------------------------------*
*     ( 1 ) View Current Blockchain in Memory      *
*--------------------------------------------------*
*     ( 2 ) View Saved Blockchains from Files      *
*--------------------------------------------------*
*     ( 3 ) Return to Main Menu                    *
*-*-*-**-*-*-**-*-*-**-*-*-**-*-*-**-*-*-**-*-*-**-*
>
```

Image 2.3.0 Blockchain view menu



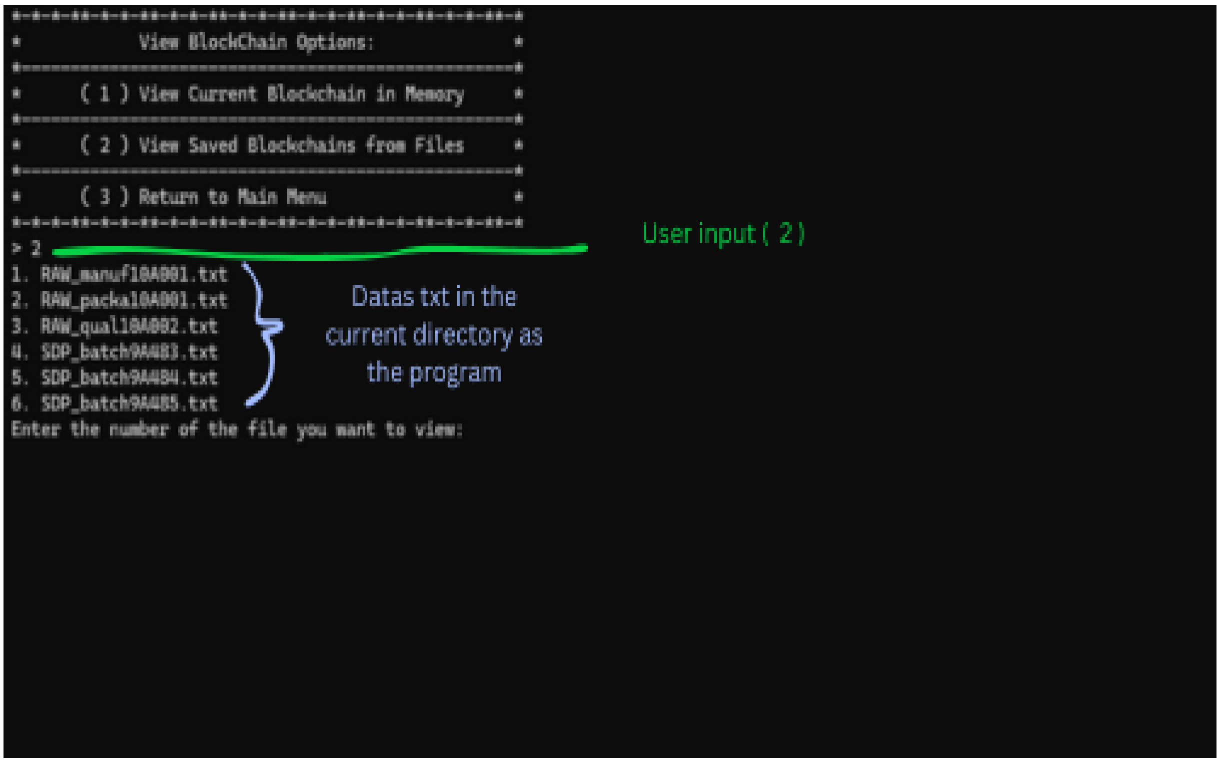
```
--------------------- Stage 1 ---------------------

Block ID      : 1
--------------
Timestamp     : 2024-03-09 13:40:19
--------------
Previous Hash : genesis_hash
--------------
Current Hash  : ITv4aK2F78kPtGRWpeJu17099628190
--------------

Farm ID                 : SDP1001
Farm Location           : Port Dickson, MY
Quantity                : 5000 Kg

Harvest Date            : 2024-03-01
Process Details         : Cold-press extraction

Batch Number            : Cert#BP102
Plantation Certificate  : BP-032024

---------------------------------------------------
Press enter to view next block...
```

Image 2.3.1 Blockchain view current memory in program for Stage 1



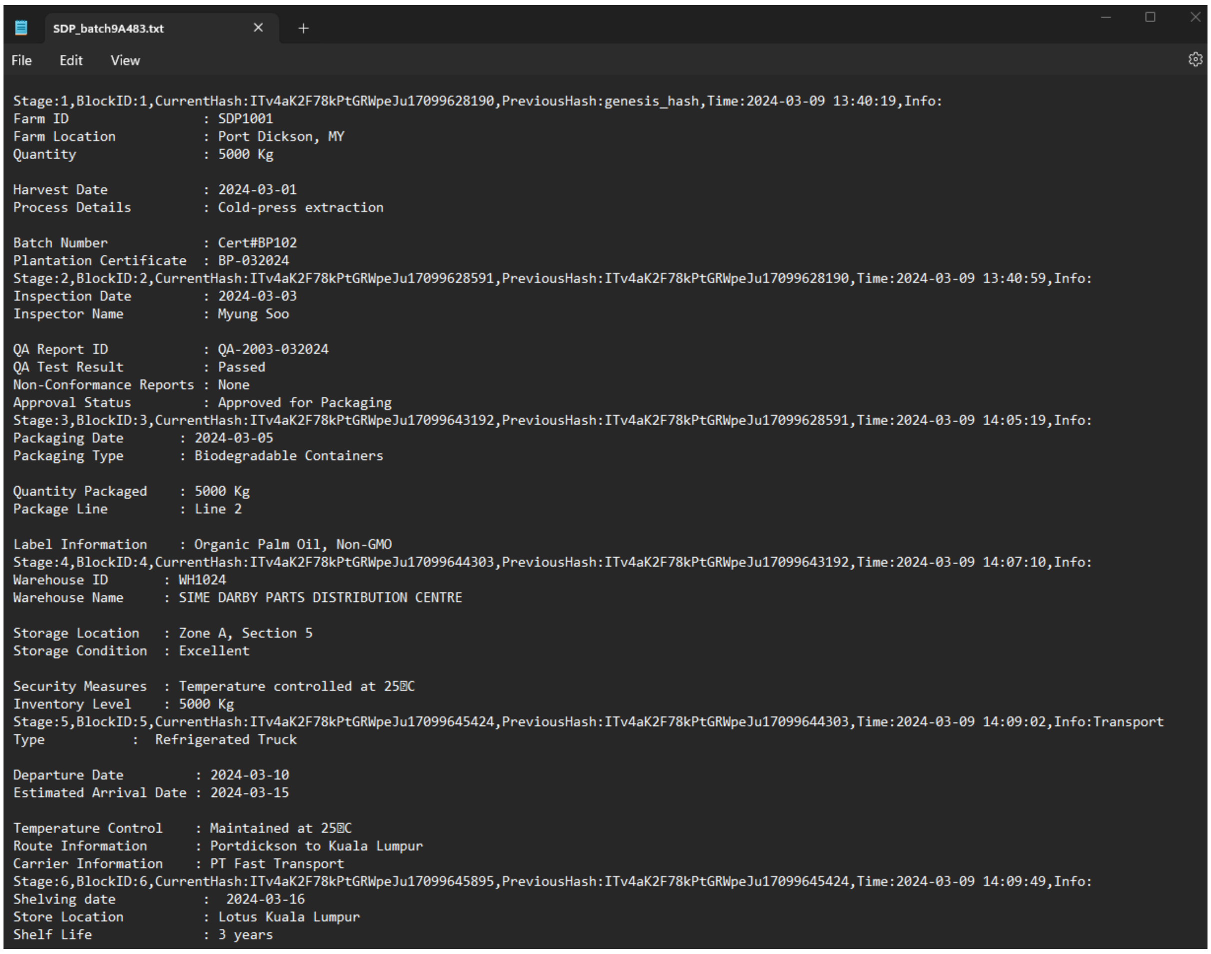
```
--------------------- Stage 2 ---------------------

Block ID      : 2
--------------
Timestamp     : 2024-03-09 13:40:59
--------------
Previous Hash : ITv4aK2F78kPtGRWpeJu17099628190
--------------
Current Hash  : ITv4aK2F78kPtGRWpeJu17099628591
--------------

Inspection Date         : 2024-03-03
Inspector Name          : Myung Soo

QA Report ID            : QA-2003-032024
QA Test Result          : Passed
Non-Conformance Reports : None
Approval Status         : Approved for Packaging

---------------------------------------------------
Press enter to view next block...
```

Image 2.3.2 Blockchain view current memory in program for Stage 2



```
*-*-*-**-*-*-**-*-*-**-*-*-**-*-*-**-*-*-**-*-*-**-*
*             View BlockChain Options:             *
*--------------------------------------------------*
*      ( 1 ) View Current Blockchain in Memory     *
*--------------------------------------------------*
*      ( 2 ) View Saved Blockchains from Files     *
*--------------------------------------------------*
*      ( 3 ) Return to Main Menu                   *
*-*-*-**-*-*-**-*-*-**-*-*-**-*-*-**-*-*-**-*-*-**-*
> 2
1. RAW_manuf10A001.txt
2. RAW_packa10A001.txt
3. RAW_qual10A002.txt
4. SDP_batch9A403.txt
5. SDP_batch9A404.txt
6. SDP_batch9A405.txt
Enter the number of the file you want to view:
```

Image 2.3.3 Blockchain .txt files menu



```
Stage:1,BlockID:1,CurrentHash:ITv4aK2F78kPtGRWpeJu17099628190,PreviousHash:genesis_hash,Time:2024-03-09 13:40:19,Info:
Farm ID                : SDP1001
Farm Location          : Port Dickson, MY
Quantity               : 5000 Kg

Harvest Date           : 2024-03-01
Process Details        : Cold-press extraction

Batch Number           : Cert#BP102
Plantation Certificate  : BP-032024
Stage:2,BlockID:2,CurrentHash:ITv4aK2F78kPtGRWpeJu17099628591,PreviousHash:ITv4aK2F78kPtGRWpeJu17099628190,Time:2024-03-09 13:40:59,Info:
Inspection Date        : 2024-03-03
Inspector Name         : Myung Soo

QA Report ID           : QA-2003-032024
QA Test Result         : Passed
Non-Conformance Reports : None
Approval Status        : Approved for Packaging
Stage:3,BlockID:3,CurrentHash:ITv4aK2F78kPtGRWpeJu17099643192,PreviousHash:ITv4aK2F78kPtGRWpeJu17099628591,Time:2024-03-09 14:05:19,Info:
Packaging Date      : 2024-03-05
Packaging Type      : Biodegradable Containers

Quantity Packaged    : 5000 Kg
Package Line         : Line 2

Label Information    : Organic Palm Oil, Non-GMO
Stage:4,BlockID:4,CurrentHash:ITv4aK2F78kPtGRWpeJu17099644303,PreviousHash:ITv4aK2F78kPtGRWpeJu17099643192,Time:2024-03-09 14:07:10,Info:
Warehouse ID       : WH1024
Warehouse Name     : SIME DARBY PARTS DISTRIBUTION CENTRE

Storage Location   : Zone A, Section 5
Storage Condition  : Excellent

Security Measures  : Temperature controlled at 25�C
Inventory Level    : 5000 Kg
Stage:5,BlockID:5,CurrentHash:ITv4aK2F78kPtGRWpeJu17099645424,PreviousHash:ITv4aK2F78kPtGRWpeJu17099644303,Time:2024-03-09 14:09:02,Info:Transport
Type           :  Refrigerated Truck

Departure Date         : 2024-03-10
Estimated Arrival Date : 2024-03-15

Temperature Control    : Maintained at 25�C
Route Information      : Portdickson to Kuala Lumpur
Carrier Information    : PT Fast Transport
Stage:6,BlockID:6,CurrentHash:ITv4aK2F78kPtGRWpeJu17099645895,PreviousHash:ITv4aK2F78kPtGRWpeJu17099645424,Time:2024-03-09 14:09:49,Info:
Shelving date            :  2024-03-16
Store Location           : Lotus Kuala Lumpur
Shelf Life               : 3 years
```

Image 2.2.4 File SDP_batch9A483.txt Audit information.

5.3.3.1 View Blockchain Data Structure from Current Memory

Data and information regarding a block can be viewed after the user has added a block, could be manually or from a generation of file. From *image 2.3.1*, It shows all the relevant information regarding the stage. After user enters any key, if there is another stage or block added *image 2.3.2*, the information regarding the block will be shown continuously until there is no block added or until all the block is added (blockchain is completed).

5.3.3.2 View Blockchain Data Structure from File

The data from the blockchain file is organized in a structured format, with each block containing several key pieces of information crucial for tracking and verifying the processes within the supply chain. From *image 2.2.4* we can down the structure and contents of a block:

Fields are separated by commas (**,**), and each key-value pair is delineated by a colon (**:**).

- **Stage:** *1* indicates the manufacturing stage.
- **BlockID:** *1* show this is the first block.
- **CurrentHash:** *ITv4aK2F78kPtGRWpeJu17099628190* is this block's hash.
- **PreviousHash:** The hash of the previous block in the chain. For the first block, this is typically a special value like **genesis_hash**. This links the blocks securely.
- **Time:** The timestamp when the block was created**.** *2024-03-09 13:40:19* shows the exact date and time of block creation.
- **Info:** Contains detailed information relevant to the block's stage. This section is subdivided into multiple lines, each detailing a specific aspect of that stage. For

example, in the manufacturing stage, it includes the Farm ID, Location, Quantity, Harvest Date, Process Details, Batch Number, and Plantation Certificate.

5.3.3.3 The ability to view and audit old blockchain data is crucial for several key reasons:

Transparency and Trust: One of the primary benefits of blockchain is its transparency. By allowing stakeholders to view historical data, they can verify transactions and track asset movement, building trust in the system.

Integrity and Security: Blockchain's immutability ensures that once data is recorded, it cannot be altered. This makes it an incredibly secure and reliable source of information. Being able to audit old data allows users to validate the integrity of the entire blockchain.

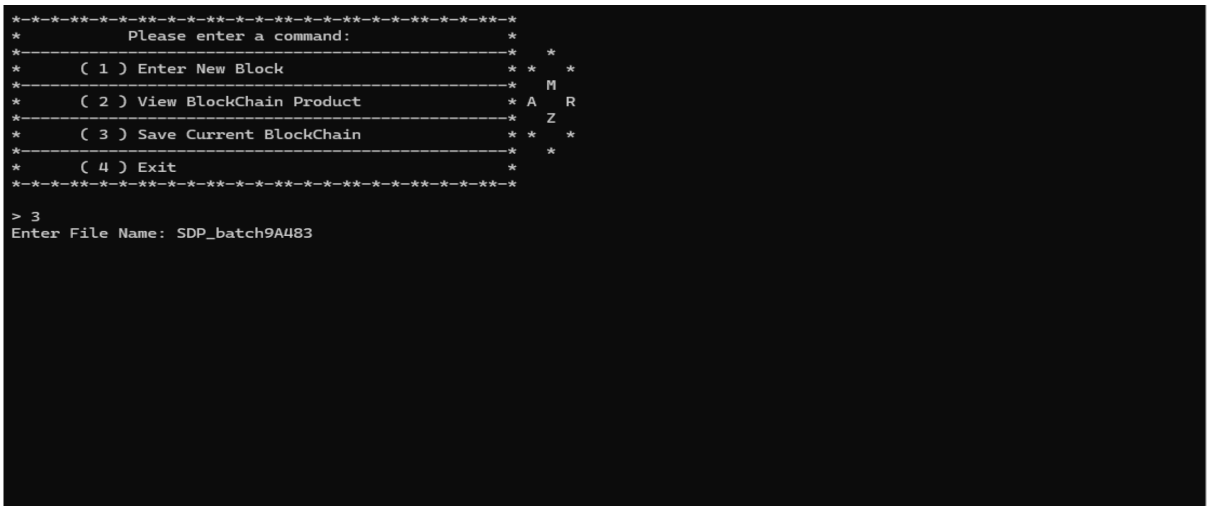
Compliance and Accountability: For many industries, particularly those heavily regulated, auditing historical data is essential for compliance. Blockchain's unchangeable ledger is perfect for maintaining an accurate and tamper-proof record of transactions.

Error and Fraud Detection: Auditing past transactions can help detect and prevent fraudulent activities. In a blockchain, altering recorded data requires changing all subsequent blocks, which is virtually impossible without detection, hence providing a robust mechanism against fraud.

Data Analysis and Decision Making: Historical data is an asset for businesses and organizations. It can be analyzed to identify trends, assess performance, and inform decision-making processes.

5.3.4 Save Blockchain

The blockchain can be saved to a file, ensuring data persistence and the ability to review historical data.



```
*-*-*-**-*-*-**-*-*-**-*-*-**-*-*-**-*-*-**-*-*-**-*
*           Please enter a command:                 *
*---------------------------------------------------*   *
*      ( 1 ) Enter New Block                        * *   *
*---------------------------------------------------*   M
*      ( 2 ) View BlockChain Product                * A   R
*---------------------------------------------------*   Z
*      ( 3 ) Save Current BlockChain                * *   *
*---------------------------------------------------*   *
*      ( 4 ) Exit                                   *
*-*-*-**-*-*-**-*-*-**-*-*-**-*-*-**-*-*-**-*-*-**-*

> 3
Enter File Name: SDP_batch9A483
```

Image 2.3.0 Saving blockchain to File

5.3.4.1 Data Persistence Importance for Traceback & Audit

Auditability: Persistent data ensures that all transactions can be audited and verified for accuracy and legitimacy.

Traceability: It enables tracking the history of transactions or operations, essential for supply chain management and provenance tracking.

Security: Persistent records enhance security by providing a permanent and unalterable history of transactions, deterring fraud and tampering.

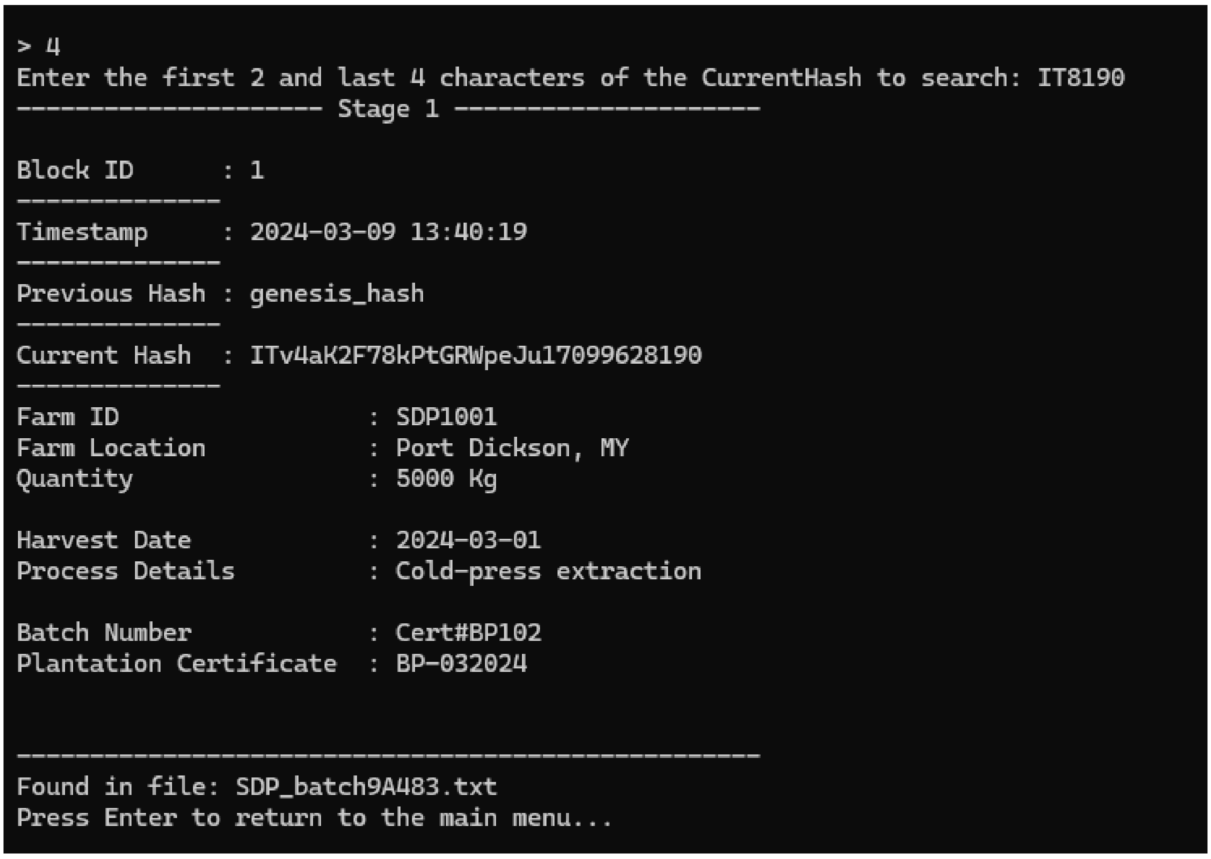
Compliance: Meets regulatory requirements by maintaining a complete and accurate history of data for compliance purposes.

Reliability: Ensures the blockchain's integrity, as each block's data remains consistent over time.

Transparency: Allows stakeholders to review the blockchain's history, fostering trust and transparency in the system.

5.3.5 Search Blockchain:

The implemented search functionality enhances the accessibility and usability of the blockchain by enabling users to efficiently locate specific blocks within the system. This feature is particularly useful for retrieving historical blockchain data based on partial identifiers. Users are prompted to enter a unique pattern—specifically, the first two and the last four characters of a block's CurrentHash. The system then scans the blockchain records stored in text files, **identifying,** and displaying the block that matches the provided hash pattern. This capability ensures that users can easily access and verify past transactions or block information, reinforcing the transparency and traceability inherent to blockchain technology**'.**



```
> 4
Enter the first 2 and last 4 characters of the CurrentHash to search: IT8190
-------------------- Stage 1 --------------------

Block ID      : 1
--------------
Timestamp     : 2024-03-09 13:40:19
--------------
Previous Hash : genesis_hash
--------------
Current Hash  : ITv4aK2F78kPtGRWpeJu17099628190
--------------
Farm ID                 : SDP1001
Farm Location           : Port Dickson, MY
Quantity                : 5000 Kg

Harvest Date            : 2024-03-01
Process Details         : Cold-press extraction

Batch Number            : Cert#BP102
Plantation Certificate  : BP-032024



-------------------------------------------------
Found in file: SDP_batch9A483.txt
Press Enter to return to the main menu...
```

Image 2.3.1 Search blockchain

5.4 Handling User Inputs:

Input validation is performed to ensure the program handles incorrect inputs gracefully, either prompting for re-entry or providing error messages.

This C++ function, collectData, is designed to gather and process data related to a product's lifecycle at different stages: Manufacturing, Quality Assurance, Packaging, Warehousing, Transportation, and Retail Shelving. It accepts three parameters: stage (of type Stage, presumably an enumeration to represent different stages of the product lifecycle), isFileInput (a boolean indicating whether the data input is from a file), and data (a string with default value empty, expected to contain data when isFileInput is true).

The function employs a stringstream object, dataStream, to concatenate and format the data for each stage. Depending on the stage, it prompts the user to enter specific details or, if isFileInput is true, splits the input string data using a splitString function (assumed to be defined elsewhere) and assigns values to relevant variables.

The structure of the function is based on a switch-case statement that handles different stages. For each case (stage), it either extracts data from the input string (if isFileInput is true and the input string has enough tokens) or gathers data through user input from the standard input stream. After collecting the data, it formats and appends it to dataStream.

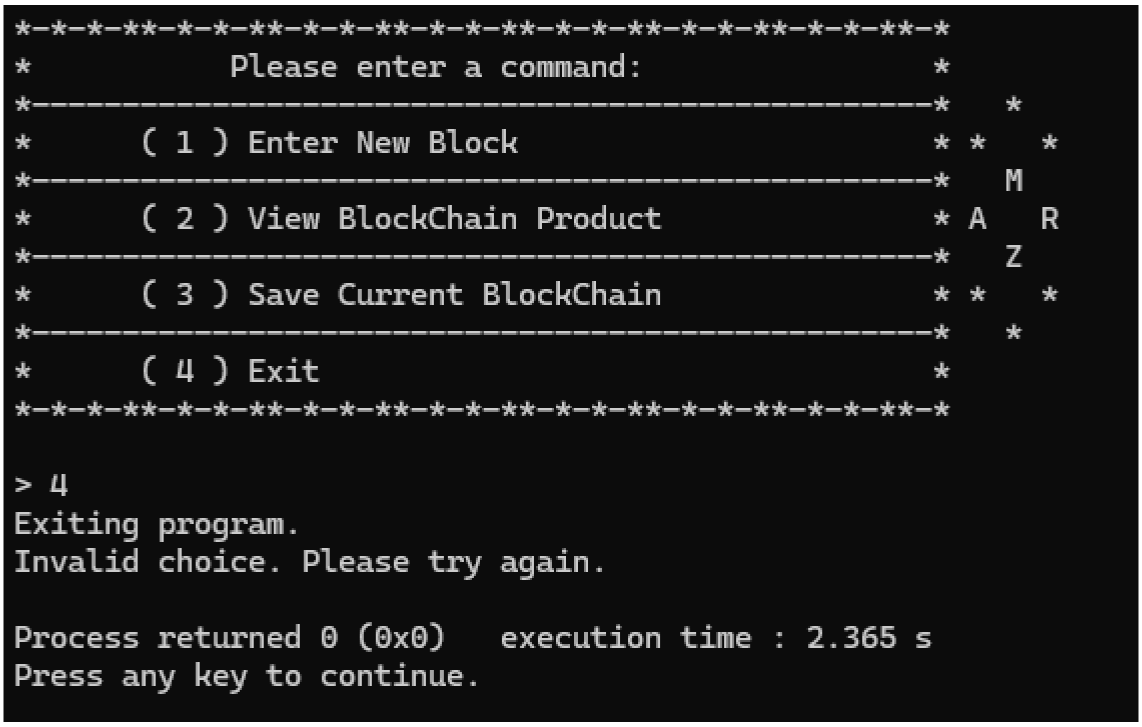
Finally, the function returns a string representation of the collected and formatted data for the specific lifecycle stage of the product. If the stage is not recognized, it outputs "Unknown stage" to the standard output and returns an empty string. This implementation facilitates data logging and processing in scenarios requiring detailed tracking and documentation of product lifecycle stages.

5.5 Interactive Interface:

The **main** function features an interactive user interface, with clear instructions and feedback based on user actions. This makes the program user-friendly and accessible.

5.6 Program Exit:

The loop continues until the user chooses to exit. Upon exiting, the program terminates cleanly, ensuring no data loss or corruption.



```
*-*-*-**-*-*-**-*-*-**-*-*-**-*-*-**-*-*-**-*-*-**-*
*           Please enter a command:                 *
*---------------------------------------------------*   *
*      ( 1 ) Enter New Block                        * *   *
*---------------------------------------------------*   M
*      ( 2 ) View BlockChain Product                * A   R
*---------------------------------------------------*   Z
*      ( 3 ) Save Current BlockChain                * *   *
*---------------------------------------------------*   *
*      ( 4 ) Exit                                   *
*-*-*-**-*-*-**-*-*-**-*-*-**-*-*-**-*-*-**-*-*-**-*


> 4
Exiting program.
Invalid choice. Please try again.

Process returned 0 (0x0)   execution time : 2.365 s
Press any key to continue.
```

Image 2.3.2 Exit program

5.7 Modularity:

5.7.1 Class Structure and Encapsulation

The **Block** class encapsulates the essential attributes of a blockchain block, including its ID, current hash, previous hash, timestamp, and informational content. Encapsulation is achieved by keeping these attributes private, ensuring that they are not directly accessible from outside the class. This design choice protects the data integrity by restricting how block data is accessed and modified.

5.7.2 Getters and Setters: Safeguarding Data Integrity

The **Block** class employs getters and setters for each attribute, a method that further underscores the principle of encapsulation. Getters provide read access to the private attributes, while setters enable controlled modifications.

```
class Block {
private:
    int id; // Unique block ID
    std::string current_hash, previous_hash, timestamp, info; // Block data

public:
    // Constructors
    Block() = default;
    // Optionally, add constructors that initialize the block

    // Getters
    int getId() const { return id; }
    std::string getCurrentHash() const { return current_hash; }
    std::string getPreviousHash() const { return previous_hash; }
    std::string getTimestamp() const { return timestamp; }
    std::string getInfo() const { return info; }

    // Setters
    void setId(int newId) { id = newId; }
    void setCurrentHash(const std::string& newHash) { current_hash = newHash;
}
    void setPreviousHash(const std::string& newHash) { previous_hash =
newHash; }
    void setTimestamp(const std::string& newTimestamp) { timestamp =
newTimestamp; }
    void setInfo(const std::string& newInfo) { info = newInfo; }
};
```

5.7.3 Blockchain Functionality (blockchain.cpp)

The core blockchain operations are encapsulated within the **blockchain.cpp** module. This module is dedicated to the fundamental aspects of blockchain management, including:

- **collectData()**: Responsible for gathering input data for block creation.
- **splitString()**: A utility function to parse strings, facilitating data manipulation.
- **getCurrentTime()**: Retrieves the current time, used for timestamping new blocks.

- **generateRandomHash()**: Simulates the hash generation process for new blocks.
- **printBlock()**: Outputs the details of a block, aiding in debugging and verification.
- **addBlockFromFile()**: Allows the blockchain to incorporate blocks from persisted storage.
- **addBlock()**: Integrates a new block into the chain, maintaining the chain's integrity.

These functions represent the application's backbone, enabling the core blockchain processes while maintaining a high degree of cohesion within the module.

Authentication (**Authentication.h**)

Authentication mechanisms are abstracted away in the **Authentication.h** file. This separation underscores the modularity principle by isolating authentication logic from blockchain operations, thereby enhancing security and simplifying authentication logic updates. Key functionality includes:

- **login()**: Manages user authentication, ensuring secure access to the system.

This approach allows the authentication logic to be modified independently of the blockchain logic, adhering to the Single Responsibility Principle (SRP).

5.7.4 File Operations (FileOperations.cpp)

File handling tasks are centralized in the **FileOperations.cpp** module. This module interacts with the filesystem to perform operations such as:

- **searchForBlockByHashPattern()**: Searches persisted blocks by hash, enabling block verification and lookup.
- **listTxtFiles()**: Lists all text files in a specified directory, used for managing block storage files.
- **writeBlockToFile()**: Persists block data to the filesystem, essential for blockchain data durability.
- **displayBlockFromFile()**: Retrieves and displays block information from storage, facilitating block inspection and auditing.

By dedicating a separate module to file operations, the application abstracts away the complexity of file management, allowing for easier maintenance and enhancements.

6. Overall Experience

Having recently completed my blockchain development project for INTS in my second year, I find myself at a pivotal point in my educational journey. The project's aim was to deepen our practical understanding of blockchain technology, a rapidly evolving field with immense potential.

6.1 Theoretical Understanding

My venture into blockchain began with theoretical groundwork. The coursework provided a solid foundation in blockchain principles, cryptography, and distributed systems. Initially, these concepts seemed abstract and challenging, but as I delved deeper, the intricacies of blockchain technology began to unfold.

6.2 Practical Application

Our main task was to develop a functioning blockchain model for INTS. This process involved coding, testing, and continuous refining. The hands-on experience taught me not just about blockchain technology, but also about systematic problem-solving and the importance of attention to detail. One of the significant hurdles was implementing the hash functions efficiently. Another challenge was ensuring the security and integrity of the blockchain, which required thorough testing and validation.

6.3 Personal Growth

This project was instrumental in my growth both as a student and as an aspiring technology professional. I developed not just technical skills, but also soft skills like time & project management. Reflecting on the noteworthy difficulties faced during the blockchain coursework, particularly in developing file writing, reading, and directory listing functions, as well as the data collection aspect, here's a structured approach to articulate these challenges:

7. Introduction to Challenges

It felt nice after completed overall project. Although the project task was quite tough, I would like to thank MS. Doreen to able to help guide through the project in the class. She gave quite numbers of input and hints in class while posting samples which guide me throughout the project.

7.1. Implementing splitString() for Data Collection

I want to implement a feature where I could just parse the data without having to manually enter the data. The **splitString()** function was a crucial part of the project. It takes a string and a delimiter as inputs and splits the string into a vector of substrings based on the delimiter. This was particularly useful in processing raw data for blockchain generation. In the **collectData()** function, **splitString()** was used to process data inputs. When reading from a file, the data string was split using a semicolon (**;**) delimiter. This made it easier to handle and organize the data for blockchain entries without requiring manual input.

```
vector<string> splitString(const string& str, char delimiter) {
    vector<string> tokens;
    stringstream ss(str);
    string token;

    while (getline(ss, token, delimiter)) {
        tokens.push_back(token);
    }

    return tokens;
}
```

7.2. Developing the 'View from File' Feature

This was a feature where I implemented where the program reads the file I chose then parse it into the collectData() function which then I could use printBlock() to view it. It was challenging due to the lack of specific online resources. The process involved:

The process of finding the right approach and utilizing these libraries effectively was time-consuming and involved a lot of trial and error. However, it was a rewarding experience as it enhanced my problem-solving skills and deepened my understanding of C++ library functionalities.

```
void displayBlockFromFile(const string& selectedFile) {
```

```
ifstream inFile(selectedFile);
if (!inFile.is_open()) {
    cerr << "Error opening file: " << selectedFile << endl;
    return;
}

string line;
Block block;
stringstream dataStream;
bool isFirstBlock = true;

while (getline(inFile, line)) {
    if (line.rfind("Stage:", 0) == 0) {
        if (!isFirstBlock) {
            block.info = dataStream.str();
            printBlock(block);

            dataStream.str("");
            dataStream.clear();
            block = Block();

            cout << "\nPress enter to view next block...";
            cin.ignore(numeric_limits<streamsize>::max(), '\n');
            getchar();
            system("cls");
            cout << "\n";
        }
        isFirstBlock = false;
            istringstream headerStream(line);
            string token;
            while (getline(headerStream, token, ',')) {
                size_t colonPos = token.find(':');
                if (colonPos != string::npos) {
                    string key = token.substr(0, colonPos);
                    string value = token.substr(colonPos + 1);

                    if (key == "Stage") block.id = stoi(value);
                    else if (key == "BlockID") block.id = stoi(value);
                    else if (key == "CurrentHash") block.current_hash = value;
                    else if (key == "PreviousHash") block.previous_hash = value;
                    else if (key == "Time") block.timestamp = value;
                }
            }
    } else {
        dataStream << line << "\n";
    }
}
```

```
    // Display the last block if there is any data left
    if (!dataStream.str().empty()) {
        block.info = dataStream.str();
        printBlock(block);
    }

    inFile.close();
}
```

8. Conclusion

In conclusion, this project on developing a blockchain system presented a series of notable challenges, each contributing significantly to my learning and personal growth. The task of implementing features such as reading raw data and generating a blockchain from it, particularly highlighted my development as a problem solver and innovator. To conclude the extraordinary features currently implemented are: -

- Add New Block: Users can add new blocks to the blockchain by entering the necessary information across different stages. This feature facilitates the creation of a comprehensive and continuous record of transactions or data.
- Hash: Usage of Mt19937 Mersenne Twister randomization as it’s known for its high degree of randomness, large period and efficient performance. Unlike basic randomization methods that may produce patterns or have shorter periods, making them predictable over time, **mt19937** offers a period of 2^19937 − 1, which is vastly sufficient for even the most demanding applications.
- View Blockchain: This functionality allows users to view blocks recently added and currently held in memory. Alternatively, users can choose to view blocks from selected files, which contain relevant blockchain information. This dual approach ensures flexibility in accessing blockchain data.
- Save Blockchain: Users have the capability to save newly added blocks to a file. They can assign a custom name to this file, which will then be saved in TXT format within the same directory as the application. This feature enables the persistent storage of blockchain data for future reference and verification.
- Search Blockchain: The application provides a search function that allows users to locate a specific block within the blockchain using its current hash. This powerful tool aids in quickly retrieving detailed information about transactions or stages in the blockchain.
- Login: Security is a paramount concern, and as such, users are required to undergo validation and authentication before gaining access to the system. This layer of security helps to protect sensitive information and ensures that only authorized individuals can interact with the blockchain.

9. References

Lakshmi, G.V., Gogulamudi, S., Nagaeswari, B. and Reehana, S., 2021, January. BlockChain based inventory management by QR code using open CV. In *2021 International Conference on Computer Communication and Informatics (ICCCI)* (pp. 1-6). IEEE.

Li, X., 2023. Inventory management and information sharing based on blockchain technology. *Computers & Industrial Engineering*, *179*, p.109196.

Sarip, M.S.M., Morad, N.A., Abd Aziz, M.K.T., Saparin, N. and Nawi, M.A.H.M., 2023. Composition of Crude Palm Oil Extracted Using Hot Compressed Water Extraction. *Journal of oleo science*, *72*(1), pp.33-38.

Thoumi, G., 2018. Palm oil: mitigating material financial risks via sustainability. *Designing a sustainable financial system: Development goals and socio-ecological responsibility*, pp.289-326.

Zulkifli, M.M., Each and every one of us has a part to play in realising a sustainable agroforestry sector in the region: governments... civil society...(and) buyers... There is no other way.