NATIONAL RESEARCH UNIVERSITY
HIGHER SCHOOL OF ECONOMICS

Faculty of Computer Science
Bachelor's Programme "Data Science and Business Analytics"

Software Team Project Report on the Topic:

Integration IoT into the blockchain-based supply chain

Submitted by the Students:

group #БПАД 223, 2nd year of study	Parshina Daria Aleksandrovna
group #БПАД 223, 2nd year of study	Seropian Nshteh
group #БПАД 223, 2nd year of study	Korotkevich Aleksandra Antonovna

Checked by the Project Supervisor:

Yanovich Yury Aleksandrovich
Associate Professor
Department of Complex Systems Modeling Technologies, HSE University

Co-supervisor:

Yash Madhwal
External supervisor
Center for Next Generation Wireless Technologies and IoT,
Skolkovo Institute of Science and Technology

Moscow 2024

Contents

Annotation

The main idea of the program project is to work with the blockchain network technology and develop a management system based on it for the management of parcel tracking devices. This technology is decentralized, its blocks are connected to each other using cryptography, and it focuses on transparency, security and reliability, which is important for supply chain management purposes. In this paper there is a description of the system development process: writing a smart contract with a given functionality and with a clear hierarchical system of role allocation, programming smart devices, setting up sensors that read environmental changes communicating with the blockchain, and applications as a basis for the interface of these instructions, providing comfort to the user. The result is a program that tracks the status of the parcel during transportation and displays all the necessary information about it to the manager.

Аннотация

Основная идея программного проекта заключается в работе с сетевой технологией блокчейн и разработке на ее основе системы менеджеринга устройств предназначенных для трекинга посылок. Эта технология децентрализована, ее блоки соединены друг с другом с помощью криптографии, и она ориентирована на прозрачность, безопасность и надежность, что важно для целей управления цепочками поставок. В этой работе есть описание процесса разработки системы: написание смарт-контракта с заданным функционалом и с четкой иерархической системой распределения ролей , программирование умных устройств настройка датчиков считывающих изменение окружающей среды коммуницирующих с блокчейном, и приложения в качестве основы для интерфейса этих инструкций, предоставляя комфорт для пользователя. В итоге создана программа, которая отслеживает состояние посылки во время транспортировки и выводит всю необходимую информацию о ней менеджеру.

Keywords

IoT : The term IoT, or Internet of Things, refers to the collective network of connected devices and the technology that facilitates communication between devices and the cloud, as well as between the devices themselves [2].

Blockchain : A blockchain is a decentralized, distributed and public digital ledger that is used to record transactions across many computers so that the record cannot be altered retroactively without the alteration of all subsequent blocks and the consensus of the network [15].

Solidity : is an object-oriented programming language created specifically by the Ethereum Network team for constructing and designing smart contracts on Blockchain platforms. It's used to create smart contracts that implement business logic and generate a chain of transaction records in the blockchain system [9].

Web3 programming : specific programming languages designed to build decentralized applications (DApps) that interact with the blockchain. Web3 refers to the internet's third generation, where people have more say over their data and can transact with each other without intermediaries [12].

Arduino Uno : The Arduino Uno is an open-source microcontroller board based on the Microchip ATmega328P microcontroller (MCU) and developed by Arduino.cc and initially released in 2010 [16].

Raspberry Pi : computer that connects to a computer Desktop or TV and uses a standard mouse and Keyboard. It has a dedicated processor, memory, and a graphics driver, just like a PC. It also comes with its operating system, Raspberry Pi OS, a modified version of Linux [14].

Swift : a robust and intuitive programming language created by Apple for building apps for iOS, Mac, Apple TV and Apple Watch [3].

Supply chain : a network of companies and people that are involved in the production and delivery of a product or service. The components of a supply chain include producers, vendors, warehouses, transportation companies, distribution centers, and retailers [7].

Testnet : is an experimental blockchain created specifically for developers to test new features [5].

Smart contracts : simply programs stored on a blockchain that run when predetermined conditions are met. They typically are used to automate the execution of an agreement so that all participants can be immediately certain of the outcome, without any intermediary's involvement or time loss. They can also automate a workflow, triggering the next action when conditions are met [6].

1 Introduction

1.1 Relevance

A supply chain is a complex system which involves multiple processes, from the production to the delivery of the product. So no wonder that this system faces a lot of problems. The main challenges associated with the supply chain is coordination stage [8] and verification of the product conditions. As various entities are involved at different stages of the supply chain, there is a need for access to trustworthy information that cannot be manipulated by a third party. The lack of this information does not allow building strong reliable bonds between partners, which directly affects the speed and efficiency of the current system.

Another related challenge is the requirement to ensure product safety and quality. For each product there are certain conditions for storage and transportation, so the consumer needs a way to control whether the carrier complies with the rules of transportation of this product.

These issues can be eliminated by implementing blockchain technology with IoT devices. Blockchain technology can be summarized as "an open, distributed ledger that can record transactions between two parties efficiently and in a verifiable and permanent way" [8].

The main advantages of blockchain are that it is a decentralized system that prevents interested parties from penetrating and compromising information that could disrupt the supply chain, furthermore it protects data by encrypting it in order to maintain security. Moreover, blockchain provides access to transactions for all participants in production and transportation at any given time. In turn, IoT devices allow to read all parameters of conditions of a moving object that are permanently stored on the blockchain.

Blockchain is a breakthrough technology that can be applied to the supply chain management industry that has some difficulties in security, efficiency, etc. By using a decentralization approach, which is one of the main aspects of blockchain, transparency in production, delivery can be obtained and maintained as it makes visible for participants what are the current results. Creation of such trustworthy space for both customers and suppliers improves business processes, operations and trading in common, in other words, this technology can have a great impact on the branch itself.

Also, smart-contracts, which are elements of blockchain network as well as the blocks, are designed to extract needed information from the database parts and might contain a set of

functions, data. Operations, which were slow because of some reasons, are done automatically owing to smart-contracts, in order to have a complete picture of the actual situation and to have an ability to make decisions as fast as possible. The potential range of blockchain usage is huge, therefore it is considered to be a prospective area of IT industry future.

1.2 Project goal

Our solution to the problem of supply chain discoordination is the following. We suggest creating the platform which provides the management tools for IoT devices where the manager is the host which can permit and retrieve access and navigate the devices actions. Devices are just entities which collect metrics of the environment and store them on blockchain. Such a system will provide control and clarity over device management and assure data safety.

1.3 Project objectives

1 **Learning smart contract programming:** An introduction to blockchain technology and its practical applications. It also covers smart contract development platforms and effective contract management frameworks. Further is smart contract development

2 **Web3 Interaction Framework:** Expanding on the basics, delve into the Web3 framework for seamless interaction with deployed smart contracts. Utilizing web3py for Python:

- Retrieving blockchain network information, including height, transaction receipts, etc.
- Accessing wallet-based data such as balance, nonces, etc.
- Constructing and deploying transactions on the smart contract directly after signing the transactions.
- Building the connection between project components like IoT devices and interface.

3 **IoT Integration:** This segment explores the integration of Internet of Things (IoT) components. It demonstrates how Arduino and Raspberry Pi devices communicate to successfully execute assigned tasks, showcasing the practical implementation of IoT in tandem with smart contracts.

4 **User Interface (UI/UX) Design:** The user interface is a pivotal element enabling user interaction with smart contracts. Designed to be versatile across devices (iOS), this UI

facilitates seamless engagement, allowing users to perform various activities within the smart contract ecosystem.

1.4 Allocation of tasks among members

The architecture comprises of three main components:

1 **IoT Devices**

2 **Web3**

3 **UI (IOS Application)**

Division of tasks between project participants: app programming on Swift - Parshina Daria, IoT devices - Seropian Nshteh, Web3 - Korotkevich Aleksandra.

2 Material overview

The integration of blockchain and Internet of Things technologies into supply chain management systems has gained significant traction in recent years due to their potential to address various challenges faced by traditional supply chains. This convergence offers several compelling advantages, driving corporations to actively explore the implementation of these technologies or seek partnerships with experienced companies in this domain (Unilever, Walmart).

Amazon and IBM are major players in the blockchain-as-a-service (BaaS) market, offering solutions for deploying and managing blockchain networks. All three companies support the Ethereum blockchain framework, which is an open-source, decentralized platform for building and running distributed applications. In comparison to Hyperledger, Ethereum is a public blockchain network that enables transparent participation from all users. Additionally, the Ethereum network is highly developed, possesses a substantial ecosystem, and offers extensive capabilities for decentralized applications due to its significant smart contract functionalities.

Amazon's blockchain platform[1] enables customers to track the provenance and movement of products throughout the supply chain, providing visibility into the various actors involved and the initial data inputs. However, there is an AWS console which is a mobile application of Amazon Web Services that does not support the blockchain direction at all. It has support for IoT devices which results in a platform that shows information from sensors every minute. Our solution works at best less than 20 seconds and at worst 1 minute, which is faster than Amazon. (Figure 2.1)

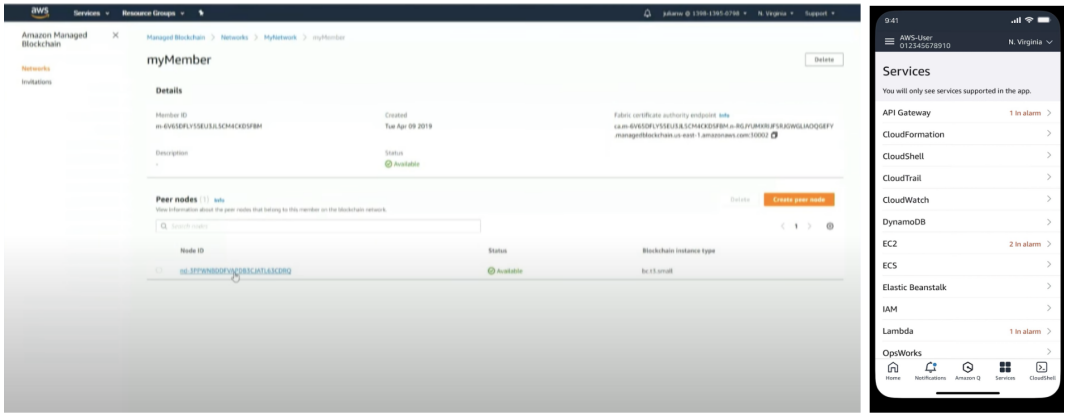

Figure 2.1: Amazon Managed Blockchain.

IBM Food Trust[5] is a blockchain-based platform that provides a comprehensive dashboard that displays data from various sources, including sensors. However, the mobile application

demonstrated for consumers does not offer real-time sensor data updates, indicating a difference in functionality of our solution compared to IBM's offering. (Figure 2.2)

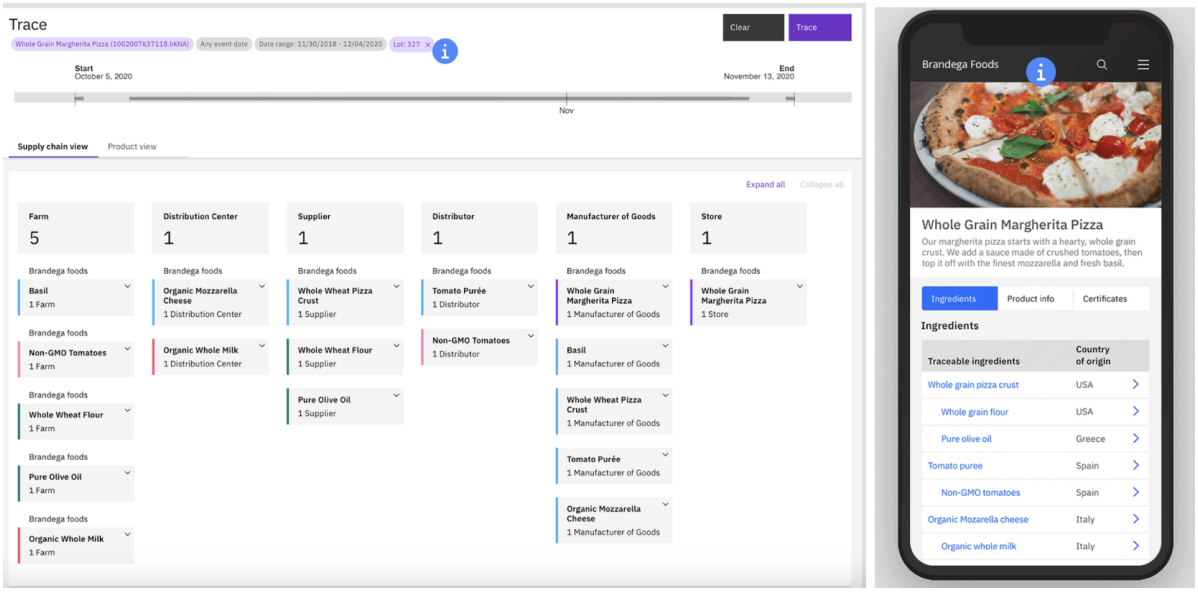

Figure 2.2: IBM Food Trust.

Companies like Oodles Blockchain, Infograins, and Maticz Technologies provide customized solutions by integrating blockchain and IoT technologies. The specific implementation decisions offered by these companies are tailored to meet the requirements of each client, in other words, there is no general solution as Amazon, and IBM have, therefore there is no ability to draw direct comparisons between approaches of these companies and compare them to our own solution.

Mobile app development is a promising area for integrating blockchain and IoT technologies due to the widespread use of mobile devices. In supply chain management it can increase convenience for users to track needed information.

After analyzing the most popular solutions on the market of IoT and blockchain, it became evident that all of them have certain inefficiencies that should be addressed in order to develop new solutions with improved quality and enhanced user convenience. Our solution took into account the best practices and implemented improvements in accordance with the requirements for our development.

Written by: Parshina Daria

3 Blockchain-based IoT solution

3.1 Functional and non-functional requirements

The following requirements the system should be having for MVP:

1 **Device location:** Implement functionality to store connected devices within the network, facilitating the identification and categorization of devices. Following are details:

- We need to incorporate a new feature that enables the enumeration of all the devices that are currently connected to the network.
- This functionality will help in identifying and categorizing the IoT devices for different purposes.
- The feature should be user-friendly and provide a comprehensive list of all connected devices, maybe like a dropdown list.
- The feature should also be designed with security in mind, ensuring that only authorized users can access the information, the smart contract will mainly do this, but it can also be done at the frontend level.

2 **Balance Inquiry:** The system should be configured to collect and display the balance information associated with individual devices. This feature will help users to keep track of available funds and provide valuable insights into the financial status of each device.

3 **Fund Management:** Develop a system that enables users to transfer funds to specific devices within the network with ease and without any interruptions in the transaction process. The mechanism should be designed in such a way that it ensures a smooth and convenient user experience.

4 **Certificate Retrieval:** Develop a feature to retrieve device certificates, verifying their authenticity and ensuring secure interactions within the network.

5 **Simulation:** Integrate support for simulating fake devices, enabling comprehensive testing and validation of system functionalities.

Expected Outcome:

The Manager Device system will facilitate efficient management of networked devices, offering functionalities for device enumeration, balance inquiry, fund management, and certificate retrieval. With simulation support for fake devices, the system ensures robustness and reliability in diverse operational scenarios.

3.2 A Smart Contract

Centralized source for IoT devices management in the supply chain is demanded to have the ability to get access to the data immediately and transparently control the processes. The blockchain network approach is applicable in such a case, which allows smart contracts to improve and properly organize the operations needed for administering the devices. Here I will discuss the process of development of the smart contract which can execute such functionality.

3.2.1 Tools for development

The programming language chosen for such work was Solidity which is an object-oriented programming language designed by the Ethereum network team specifically for developing smart contracts that run on Ethereum. Solidity is constantly evolving, new releases every month help maintain the comfortable usage experience of the system.

For the development environment I used HardHat, which provides all variations of tools which helps smart contract programming, such as runner, debugger and local network for testing.

Moreover, the network chosen for deployment was Sepolia Testnet, which is an available network for smart contract development which allows to simulate mainnet processes avoiding any disturbance and interruptions. Sepolia faucets were easier to access as the process of getting Ether was not overcomplicated, as well as it was important to assure that Metamask support such testnet as wallet addresses provided by Metamask are crucial part of smart contract development.

3.2.2 Actors

Smart contract has its built in logic which specifically determines the participant's hierarchy and their spheres of authority, it is drastically important for such complex systems as supply chain. Below there will be uncovered specifics about each actor.

Device manager:

The higher entity that manages all devices, basically the owner of the system and initial deployer. Concerning the lower level description it is the wallet address which represents the account which executes deployment of the smart contract to the Sepolia Testnet. The idea is that the manager stays unchanged for the whole life cycle of the smart contract, affirming singular ownership of the system. The administrator is responsible for the following activities:

1 Granting and revoking certification of devices - managing the device access to the smart

contract data.

2 Initialization of a device - adding a new one to the list of devices.

3 Transfer funds to a device - sending Ether to the device's wallet.

4 Removing a device - deleting all related data to a specific device (device's wallet).

The instance of the device manager is written into the constructor of the smart contract and exists as a global variable in it.

Devices:

IoT devices collect data through two sensors ultrasonic and potentiometer and send it to be stored on the blockchain in the smart contract. Each device has a unique identifier; this role is given to wallet address. There can be multiple devices and the duration of their existence is determined by the system manager, as well as the available actions for them. To make it simple devices are subordinate to the manager, so they execute actions needed for the manager as their dependency is indissoluble.

The structure of the device is:

- address (address payable)
- type (string)
- balance (uint)
- certificate (bool)

Their functionality is:

1 Sending ultrasonic metrics - saving distance measurements (0 - 100 cm)

2 Sending potentiometer metrics - saving collected metrics from potentiometer (0 - 100 V)

3.2.3 Smart contract design

In this block I will dive into more details of smart contract design development concerning the constraints of the system as well as security and implemented functions. I used Solidity version 0.8.19 and HardHat version 2.20.1.

1 **Access control**

Ensuring the security of smart contracts requires strict access control measures regarding sensitive data. This can be achieved by conducting an internal check whenever foreign entities try to modify smart contract data. Each smart contract modifying function has either permission for a manager or the device's access, thereby preventing unauthorized entities from compromising data.

The attempt to control read functions were unsuccessful as blockchain is transparent and allows to view transactions to any user, so the control of the read only functions goes against blockchain ideology and does not make sense.

The initial implementation was done through modifiers as it is a convenient way of setting access restrictions as it does not overload programs with code and makes it easy to consume for any programmer. However, the limitations from the interface programming on Swift and libraries used pushed me to rewrite the smart contract using required constructions as the modifier is not being supported in the Web3 Swift library.

2 **Device initialization**

To introduce a new device, the function can be called that takes the following values:

- Device address (address payable) – the address of the wallet that represents the device on the blockchain
- Device type (string) - string value that describes the nature of the device.

This function is only accessible by the manager and makes a new instance of the device with the unique address. All the devices are organised into mapping where the address is a key and the instance of Device struct is a value. Moreover the addresses additionally are stored in a separate array, this was done for another function which will be discussed below.

After the device is submitted to the system, it does not have access to any actions before it will be granted a certificate by the manager. So the balance and certificate members are set to default values 0 and false respectively.

Algorithm 1 Device Initialization

Require: deviceAddress, deviceType

function *addDevice*()
  if msg.sender = Manager **then**
    if Device not registered **then**
      $mapOfDevices[deviceAddress].deviceAddress \leftarrow deviceAddress$
      $mapOfDevices[deviceAddress].deviceType \leftarrow deviceType$
      $mapOfDevices[deviceAddress].deviceBalance \leftarrow deviceAddress.balance$
      $mapOfDevices[deviceAddress].isCertified \leftarrow false$
      $addressArray \leftarrow deviceAddress$
    else
      Revert with error: "Device has already been registered."
    end if
  else
    Revert with error: "Only manager can call this function."
  end if
end function

3 Grant and revoke certificate

After device initialization, the manager can grant or withdraw the certificate. Here, the manager is assumed to be the initial deployer of the smart contract, restricting access to foreign entities. The certificate serves as a means of access control to the metrics transmission.

Certified devices can receive funds and send values collected from sensors, which are then stored on the blockchain. This is checked by the written exceptions in the functions which prevent devices without certification from sending transactions using write functions of the smart contract. To state it simply, the device address is checked and approved by the manager to start working.

After revoking the certificate, the device cannot communicate with the blockchain; in other words, all processes regarding this specific device will be halted until the certificate is reinstated.

The one thing which is worth noting is that the uncertify function was implemented to remotely stop metrics transmission and do not change already existing information about this device on the blockchain. Therefore the remove and uncertify function exist for two different purposes, where remove - deletes all traces of the devices existence whether uncertify just stops all processes.

Algorithm 2 Device certification

Require: deviceAddress

```
1: function deviceCertification()
2:     if msg.sender = Manager then
3:         if currentState = false then
4:             mapOfDevices[deviceAddress].isCertified ← true
5:         else
6:             Revert with error: "Device has already been certified by the manager."
7:         end if
8:     else
9:         Revert with error: "Only manager can call this function."
10:    end if
11: end function
```

Algorithm 3 Device Revoke certificate

Require: deviceAddress

```
1: function deviceUncertify()
2:     if msg.sender = Manager then
3:         if currentState = false then
4:             mapOfDevices[deviceAddress].isCertified ← false
5:         else
6:             Revert with error: "Device has not been certified by the manager."
7:         end if
8:     else
9:         Revert with error: "Only manager can call this function."
10:    end if
11: end function
```

4 Transfer funds

The interaction between devices and smart contract is facilitated through the presence of wallet addresses. For the account to send transactions, it should possess sufficient funds to cover the associated costs. Therefore the function fulfilling this needs was implemented to unable manager to transfer Ether from his wallet to the device's wallet to ensure adequate balance.

The idea is that transfers are executed by the manager by specifying the value this transaction will carry, however the function takes only the address representing the device. After the funds are credited to the wallet balance, the internal function is called to update the balance member of the Device structure.

The constraints have been carry out to avoid unauthorized transactions with non-existing devices. Moreother, another exception was handled which blocks the attempt to transfer the amount of ether which exceeds the balance limit of the administrator. The Algorithm (4) is discussed below:

Algorithm 4 Transfer funds

Require: deviceAddress

function $transferFunds()$
  if msg.sender = Manager **then**
    if msg.sender.balance $\geq$ msg.value **then**
      if mapOfDevices[deviceAddress].isCertified = true **then**
        $boolindicator \leftarrow resultoftransaction$
        if indicator **then**
          $updateDevBalance(deviceAddress)$
        else
          Revert with error: "Payment failed."
        end if
      elseRevert with error: "Device has not been certified yet."
      end if
    else
      Revert with error: "Device has not been certified by the manager."
    end if
  else
    Revert with error: "Only manager can call this function."
  end if
end function

5 Remove device

In scenarios where the device is of no use to the manager, he can completely remove all traces of its existence in the system. The Remove function eliminates the device's instance

and sensor metrics associated with its address, ensuring an irreversible removal process.

Solidity provides several methods, however when we are speaking about map the removal is done through nullifying every member of the device structure instance, while with an array of addresses such issue was not uncovered and was done the regular way. As in any data transforming functions the constraints on access were applied.

Algorithm 5 Remove device

Require: deviceAddress

```
1: function removeDevice()
2:     if msg.sender = Manager then
3:         delete mapOfDevices[deviceAddress]
4:         delete potentiometerMetrics[deviceAddress]
5:         delete ultrasonicMetrics[deviceAddress]
6:         delete addressArray[i] = deviceAddress
7:     else
8:         Revert with error: "Only manager can call this function."
9:     end if
10: end function
```

6 Sending metrics

Any certified device can connect to the smart contract to send the maximum value of two sensors ultrasonic and potentiometer every 20 seconds, to facilitate the implementation of such capability two functions were designed which write down new values into arrays. The values are organized into two maps of arrays one for potentiometer and another for ultrasonic, where the keys are devices's addresses.

To avoid overcomplication of the system only the last 10 metrics are stored on blockchain, whenever the number of measurements reaches the limit, next metric is written to the end removing the first one from the array assuring the constant number of values.

Below (algorithm 6) there is a description of the functions logic:

Algorithm 6 Sending metrics

Require: deviceAddress

```
function removeDevice()
    if msg.sender = Manager then
        delete mapOfDevices[deviceAddress]
        delete potentiometerMetrics[deviceAddress]
        delete ultrasonicMetrics[deviceAddress]
    else
        Revert with error: "Only manager can call this function."
    end if
end function
```

7 Read only functions

The view functions play a crucial role in interface functionality and testing procedures. Due to their non-modifying nature of the contract's state, they are open for invocation by any entity.

There are four such functions outlined below:

- **getDevBalance() return(uint)**

  This function retrieves the balance attribute of the device structure stored in the map. It requires a device address as input and returns a uint value.

- **getDeviceCertification return (string)**

  This function retrieves the certificate attribute of the device structure stored in the map. It returns a string indicating the certification status of the device - "Device has been certified by the manager" for true state and "Device has not been certified by the manager" for false state. It also requires a device address as input.

- **getAllDevices return (Device[])**

  The function returns an array of the devices.

- **getMetrics return (uint[])**

  This function retrieves the metrics related to a device from the structure stored in the map. It requires a device address as input and the sensor id: 1 for potentiometer's and 2 for ultrasonic's measurements, and returns an array of uint values.

3.2.4 Smart contract testing

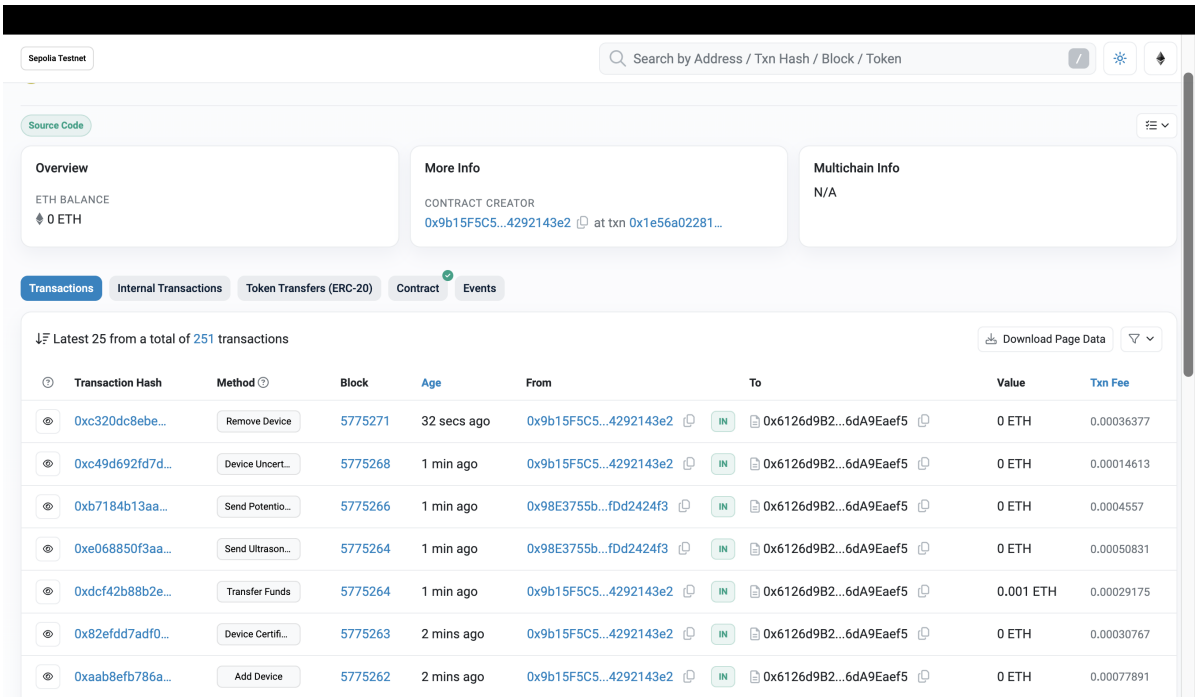

Figure 3.1: Workflow from initialization of device till removal.

Writing test cases is crucial for ensuring that a smart contract functions as expected. Test cases help verify the reliability, functionality, and security of the smart contract by checking if it satisfies the specified requirements. By conducting thorough testing, we can identify and resolve unexpected errors that may arise during the execution of the smart contract.The Hardhat testing framework was utilized to verify the smart contract's compliance with the specified requirements.

Hardhat provides a comprehensive set of tools for evaluating the system's performance under different scenarios. The framework enables developers to conduct tests that assess the functional completeness of the smart contract and identify potential issues. The network for testing was chosen to be Hardhat testnet as it provides the number of default accounts which can be used in the test cases to connect to the contract for access control testing. I defined six blocks of test cases: access control, device initialization, revoke and grant certificate, sending funds, sending metrics and remove device. They trigger each function of the smart contract and test it in different critical scenarios, they are listed below.

The following 20 tests were conducted. As a result, all tests were passed:

- **Access control:**

  ✓User access denied (addDevice)

✓User access denied (uncertify)

✓User access denied (transferFunds)

✓User access denied (removeDevice)

✓Device access denied (sendPotentiometerMetrics)

✓Device access denied (sendUltrasonicMetrics)

✓User access denied (certification)

- **Device initialization:**

  ✓Device already registered (addDevice)

  ✓AddDevice

- **Revoke and grant certificate:**

  ✓Has not been certified to revoke certificate

  ✓Certification of device

  ✓Has already been certified

  ✓Revoke certificate of device

- **Sending funds:**

  ✓Has no certificate, unable to receive funds

  ✓Transfer Funds

- **Sending metrics:**

  ✓Sending metrics of ultrasonic

  ✓Sending metrics of ultrasonic limit 10

  ✓Sending metrics of potentiometer

  ✓Sending metrics of potentiometer limit 10

- **Remove device:**

  ✓Remove device

The following testes indicate the proper functionality of the smart contract facilitating all needed functional and nonfunctional requirements. Each test case triggered the particular functions testing each component to assure continuous and comfortable usage experience. The

positive result of tests suggest that it can be passed to the further stages like deployment to Sepolia testnet.

Solc version: 0.8.19	Optimizer enabled: false	Runs: 200	Block limit: 30000000 gas
Methods			
Contract	**Method**	**Min**	**Max**	**Avg**	**# calls**	**usd (avg)**
DeviceManager	addDevice	–	–	142133	26	–
DeviceManager	deviceCertification	–	–	46606	16	–
DeviceManager	deviceUncertify	–	–	24728	2	–
DeviceManager	removeDevice	–	–	45712	2	–
DeviceManager	sendPotentiometerMetrics	56139	113767	64710	24	–
DeviceManager	sendUltrasonicMetrics	56095	113723	64666	24	–
DeviceManager	transferFunds	–	–	43625	2	–
Deployments				**% of limit**		
DeviceManager		–	–	2471898	8.2 %	–

Figure 3.2: Estimated gas usage based on test cases.

Apart from regular tests cases I used the Hardhat gas reporter, which gives a perspective on the gas consumption under different amounts of data processed, Figure 3.2. It gives several parameters for each function: min, max and average which allows to access the essential amount for gas required for execution of particular function. This information is incredibly important for building connection between IoT device, application and smart contract, as a web3 libraries used there demand the specific gas limit to be assigned.

As a result of the test cases the following constraints were uncovered: gas limit for sendMetrics functions should be raised to at least 114000, as in other cases transactions will fail with out of gas error.

3.2.5 Deployment of smart contract

The deployment was done to the Sepolia Testnet using Hardhat deploy. It was an significant step to unable remote communication of all components of the project with the smart contract.

The following steps were done before initial deployment:

1 I set up development environment: such as I acquired wallet account, as well as install a Hardhat development framework.

2 Wrote and compiled my smart contract: Using programming language Solidity to write smart contract, and then compiled it to generate the necessary bytecode and Application Binary Interface (ABI) files.

3 Funded wallet with Sepolia Ether, obtained some Ether, which can be used to pay for gas fees essential for smart contract deployment.

4 Deploying the smart contract to Sepolia testnet.

3.2.6 Verification of the smart contract on Etherium

Verifying the smart contract is an essential step to ensure the integrity and security of the deployed code. This process was done using the built-in Hardhat verify task, involves comparing the on-chain bytecode of the deployed smart contract with the source code. This verification step helps to eliminate any potential risks or vulnerabilities that may have been introduced during the deployment process, providing a high degree of confidence in the correctness and reliability of the smart contract.

After the successful verification of the smart contract, the system can be passed to the client usage development. This means that the deployed smart contract can be safely and securely interacted with by applications in our case written on swift, enabling the end-users to leverage the functionality and features provided by the smart contract. The verification process ensures that the smart contract's behavior matches the intended design, allowing for beautiful user experience.

Written by: Korotkevich Aleksandra

3.3 Frontend development

The decision to develop the mobile application for iOS users using the Swift language and UIKit framework was strategically made to facilitate administrator access to the Web3 service from their phones, enhancing user experience and expanding the reach to a broader audience, mobile smartphone users. iOS devices were chosen due to their substantial market share, user experience advantages, and robust security features, making them a preferred platform. While the audience size may be smaller compared to Android users, the focus on a more engaged and targeted user base aligns well with testing and refining the product effectively.

The closed ecosystem of iOS contributes to its stability and security, crucial factors when integrating technologies like blockchain. The user interface design, leveraging UIKit's intuitive and customizable components, plays a pivotal role in ensuring ease of use for administrators interacting with IoT devices. The decision to opt for UIKit over SwiftUI was driven by its flexibility, fewer constraints, and maturity, all of which are essential considerations in the development of a sophisticated mobile application like ours.

The project was implemented exclusively through programmatic means, eschewing the use of UIstoryboard, thereby ensuring a purely code-driven approach to user interface design and layout.

The mobile application is supporting versions of iOS devices no less than 16.0 and requires less than 50 megabytes.

The mobile application incorporates a black and white theme to enhance the user experience for device administrators, ensuring that all screens are compatible with both themes.

3.3.1 Architecture

The primary goal that is needed to be achieved in supply chain management is providing secure, transparent, and current data about the processes, therefore it is necessary to let users get access to real-time data and this concept is reflected in the architecture. The mobile application employs a decentralized architecture, leveraging the inherent properties of blockchain technology for data storage and management, thereby eliminating the need for a centralized database, and as a result, memory allocation is efficient as well. Furthermore, such an approach will prevent the whole system from failing compared to the traditional one.

The process can be broken down into several key parts: reading devices, reading sensors, adding new devices, transferring funds, certification, and removal. The application's functionality directly relies on data from the blockchain network and processes and stores results within it. The circular process of data exchange and updates between the mobile app and blockchain network forms a robust foundation for efficient and effective application functionality.

The reading devices function as a crucial gateway between the blockchain network and the application, facilitating the retrieval of essential data that underpins various functionalities within the system. Reading sensors' role can be denoted as providing a real-time representation of the recent sensor activity associated with a specific device. They display the last ten measurements recorded on the blockchain, which are continuously updated with information delivered to the blockchain network during the sensor's operational process, contingent upon the device's certification status. The application retrieves sensor data such as potentiometer and ultrasonic readings.

Information about the devices, including their balances, verification statuses, addresses, is interconnected with the blockchain data. Moreover, the last available measurement of the potentiometer and ultrasonic is passed there as well. This data is updated and maintained through interactions with the blockchain network.

Functions like adding devices, certification, funds transfer, and removal directly impact the blockchain data, leading to comprehensive updates across the system. This ensures that all devices read are refreshed and reflect the most recent information.

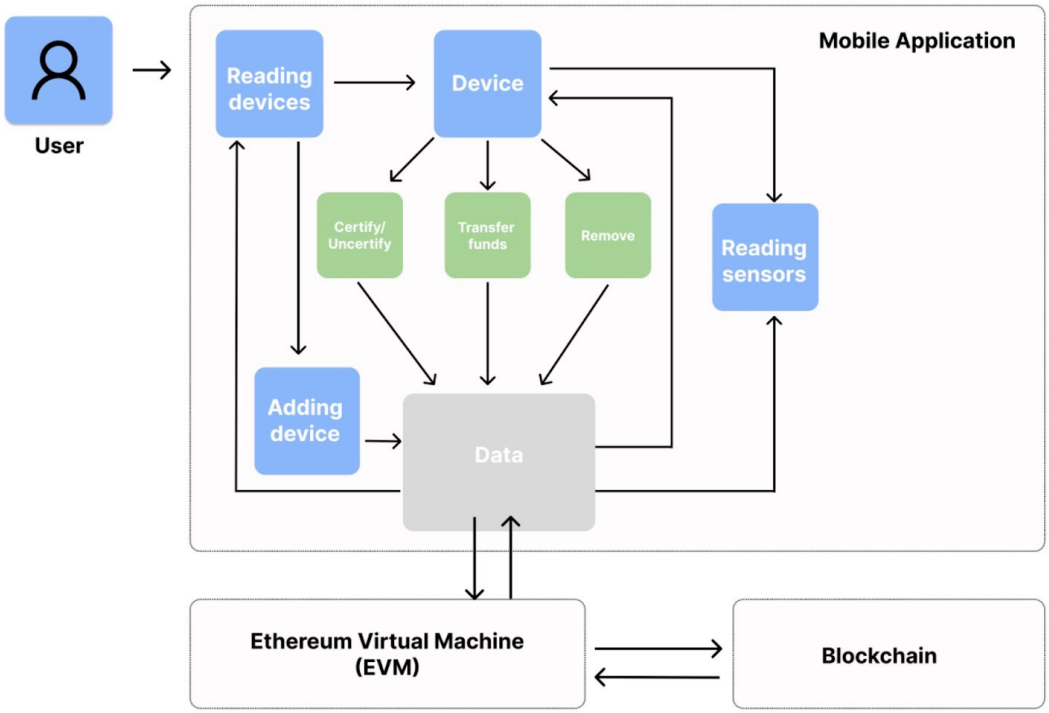

Figure 3.3: Workflow of the mobile application.

3.3.2 Implementation

Access to the system's administrative functions requires the user to log in using a specific Ethereum address. In our version the individual who deployed the smart contract is designated as the administrator and holds the sole authority to access the system. If an incorrect address is entered or if access is not granted to the specified address, entry into the system will be denied. This stringent access control mechanism ensures that only the authorized administrator can manage and oversee the system's operations. (Figure 3.4)

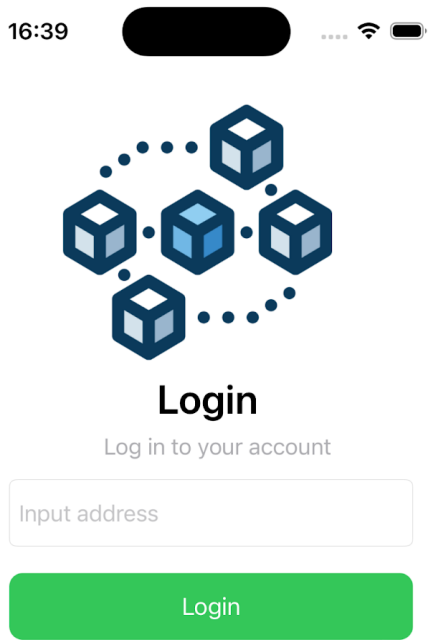

Figure 3.4: Login screen.

After logging in, the administrator gains access to track current device registrations within the system, view balances, certification statuses, and copy Ethereum addresses of the devices for personal use. Additionally, the administrator can add a new device by inputting its address and specifying the device type. The system rejects inappropriate addresses for addition, triggering an error message.

To initiate the operational workflow of the main page, the initial step involves establishing a connection with Web3 technology. It is implemented via a link to the personal Sepolia Testnet project of our team, which gets access to administrator wallets with tokens on this cryptocurrency network and allows to use Sepolia Ethereum for calling smart contracts' functions. Furthermore, for interaction with these functions it is needed to state the address of the deployed contract, JSON format of ABI data that is received from the contract, where ABI is Application Binary

Interface, which is needed to connect blockchain and out of blockchain interactions, private key which is used for transaction signing, address of the user wallet, nonce which is a unique value for hash generation that is used for transaction creation.

Using these parameters, the function from the smart contract can be called. After that the gas price is established, where the gas price is a small amount of cryptocurrency (Sepolia Ethereum in our project) that must be paid to the process validators of the network, then the transaction can be created, signed. In the end the transaction receipt can be obtained after processing to the blockchain for some time and in this receipt there is information if it was successful or not. However, in some cases gwei (gas price) is not paid due to the fact that a new transaction is not necessary to create every time, on the contrary, it is referring to the blocks that are already in the system.

The call to the smart contract's function getAllDevices results in the display of a table view on the main screen, where each cell represents a device along with its corresponding information. Additionally, an addition feature is presented through a separate window that emerges from the bottom of the main screen, containing two text fields for input. Upon clicking the button and verifying the correctness of the information, the addDevice function of the smart contract is triggered. Subsequently, after the main screen is updated, a new device is visible, and this update is facilitated by invoking getAllDevices once again. (Figure 3.5)

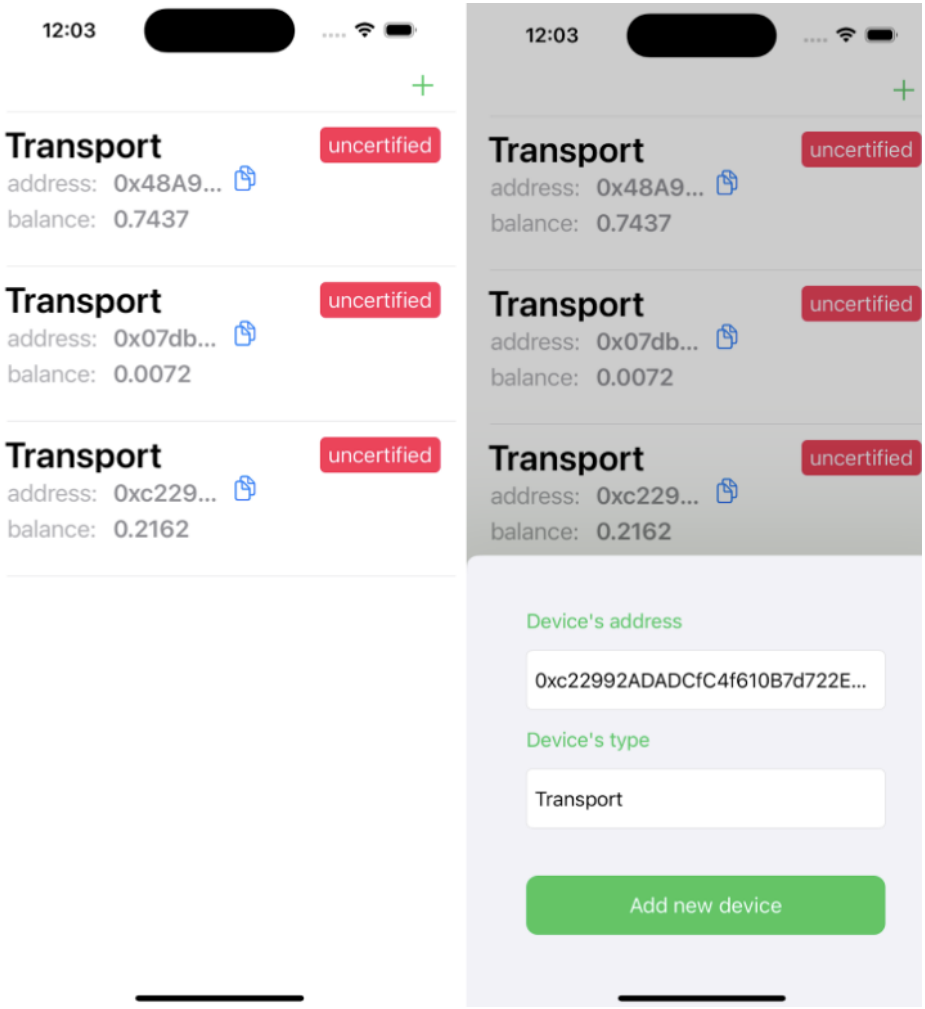

Figure 3.5: Main and add device screens.

Upon selecting a specific cell, a comprehensive overview of the device is presented, accompanied by additional options for interaction. For devices that have undergone certification, three buttons are available for engaging with the smart contract functions: transfer, uncertify, and remove. Conversely, if the device lacks certification, the transfer option is not accessible. The current balance of the device is displayed to inform the administrator of the necessity for fund transfers or the sufficiency of the existing balance. Insufficient funds for the device may hinder the execution of the smart contract's functions, making it problematic. Furthermore, there is a cell that contains the most recent data from the sensors. In the absence of information about them in the blockchain, no output is provided. (Figure 3.6 the first one)

Uncertifying or removing a device triggers an alert dialog that requires user confirmation before proceeding. The alert serves as a safeguard against unintentional actions by prominently displaying a warning message and providing buttons for the user to either confirm or cancel the uncertification or removal process. By requiring explicit confirmation, the alert reduces the likelihood of unintended uncertifications and removals and promotes a more secure and controlled

environment for managing device certifications. (Figure 3.6 the second one)

In contrast, certifying a device and transferring funds do not prompt an alert. The first triggers the deviceCertify function of the smart contract, while transfer displays a window that slides up from the bottom of the screen.

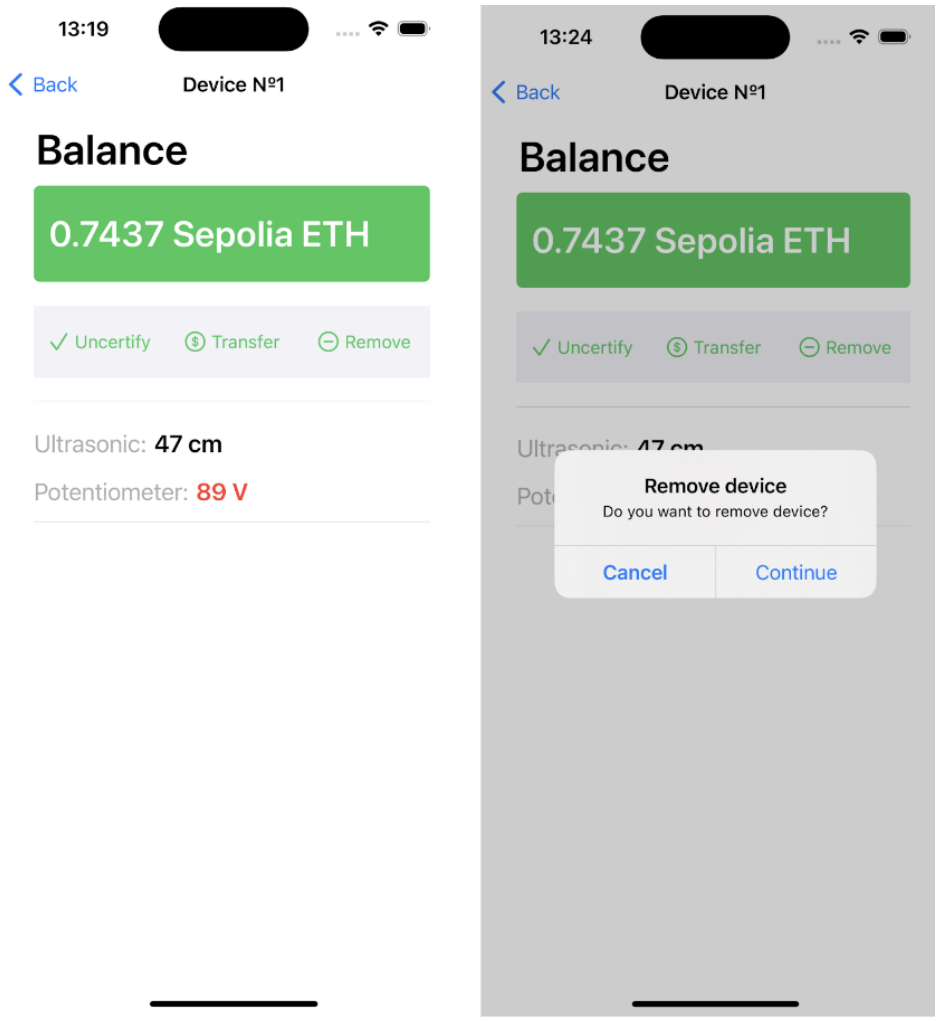

Figure 3.6: Device information screen and alert of removal.

To transfer Sepolia Ethereum, the user interface displays the balance of the administrator's address. This allows the user to verify the available amount of Sepolia Ethereum that can be transferred. The input field requires a valid amount of the currency, which must be entered without commas or other non-numeric characters. If the requested amount exceeds the available balance, the transfer will not be permitted. However, if the input is within the balance limits and in the correct format, the amount will be converted to gwei and the transferFunds function will be executed to complete the transfer. (Figure 3.7)

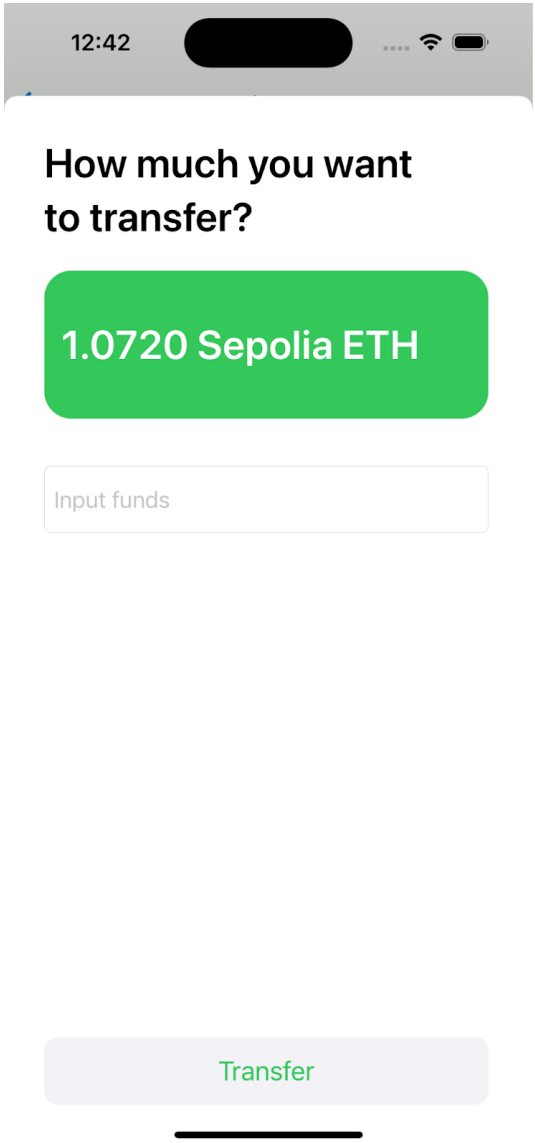

Figure 3.7: Transfer funds screen.

When accessing the sensor data cell, detailed information is displayed in a new screen in a table view containing all stored measurements of IoT sensors collecting real-time data. This data is automatically refreshed every 30 seconds to allow the blockchain sufficient time for processing. The values for the potentiometer are visually represented in the table according to their real-life color coding: green for values below 70 V, yellow for values below 85 V, and red for all other values. The identification of issues is crucial for administrators, therefore the mobile application shows values of the potentiometer colored differently in order to notify if there are any problems at the moment. Ultrasonic's data enables real-time monitoring and control of device movements. The administrator is informed about the conditions and distance in such a way.

The getMetrics function is triggered twice for potentiometer and ultrasonic data, and it runs continuously every 30 seconds to ensure data updates. This same methodology is applied to

the previous screen where sensor data is also available. When a device is uncertified, data transmission ceases, the measurement process halts, and updates are suspended until recertification is completed. (Figure 3.8)

Figure 3.8: Sensors data screen.

3.3.3 UI components

The user interface of the application primarily consists of the colors green, white, and gray (dark gray in black theme). The labels displaying certification statuses on the main screen are either red or green, depending on the value. If the value is true, the label is displayed in green.

The login, add new device, and transfer buttons are used to verify the provided input and are responsible for outputting errors on the screens. The color of the error messages depends on the outcome, with negative outcomes displayed in red and positive outcomes in green.

The uncertify and remove buttons contain alerts, and by tapping the continue button, the execution of the smart contract function is permitted.

The add new device, transfer, and certify/uncertify buttons also include animations to indicate that the button has been tapped.

The copy button contains an animation that transitions from a document icon to a checkmark icon and then back to the document icon after tapping. This animation ensures that the user knows the address of the required device has been copied to the clipboard.

3.3.4 Web3 library

In order to explain how functions using the Web3 library are implemented I would like to discuss some of them.

getAllDevices is responsible for retrieving a list of devices from the blockchain. It likely makes a call to a smart contract function that returns an array of device data. The function returns a structure called Devices with type, address, certification, balance

HomeViewController.contract function creates a dynamic contract instance using the contract address and ABI. By creating a dynamic contract, the application can interact with the smart contract.

callFun is used to call a specific function within the smart contract. It takes the function name as a parameter and returns a closure that represents the function call. The closure returns a SolidityInvocation object, which encapsulates the details of the function call, such as the function parameters and the contract address.

Since interacting with the blockchain involves making requests to the network, the function call needs to be waited upon to receive the response. This waiting period is necessary to ensure that the transaction is processed and the results are available.

Finally, the function extracts an array of Devices from the response, which is an array of dictionaries returned by the smart contract function. Each dictionary represents a single device and contains the device data as specified in the Devices structure. (Figure 3.9)

```
static func getAllDevices() -> [Devices]{
    var devices: [Devices] = []
    let contract = HomeViewController.contract()
    let call = try! callFun(input: "getAllDevices", contract: contract)!().call().wait()
    let array: [[String: Any]] = call[""]! as! [[String : Any]]
    for arr in array{
        devices.append(Devices(type: arr["deviceType"] as! String, balance:
            arr["deviceBalance"] as! BigUInt, certified: arr["isCertified"] as! Bool,
            address: arr["deviceAddress"]! as! EthereumAddress))
    }
    return devices
}
```

Figure 3.9: Implementation of getAllDevices.

The static function getPot by the address of a particular device gets the information about one of the sensors: potentiometer. By calling getMetrics smart contract function it executes and then returns an array of BigUInt values that is extracted from the call. (Figure 3.10)

```
static func getPot(_ deviceAddress: EthereumAddress)-> [BigUInt]{
    let contract = HomeViewController.contract()
    let call = try! HomeViewController.callFun(input: "getMetrics", contract:
        contract)!(deviceAddress, 1).call().wait()
    let array: [BigUInt] = call[""]! as! [BigUInt]
    return array
}
```

Figure 3.10: Implementation of getPot.

Due to the fact that functions return information from the blockchain, no transaction signing is needed.

```
static func certify(_ deviceAddress: EthereumAddress, _ deviceCert: Bool){
    let address = deviceAddress
    let cer = deviceCert
    let contract = HomeViewController.contract()
    let nonce = try! HomeViewController.web3.eth.getTransactionCount(address:
        HomeViewController.walletAddress, block: HomeViewController.tag).wait()
    var call: any SolidityInvocation
    if cer == false{
        call = HomeViewController.callFun(input: "deviceCertification", contract:
            contract)!(address)
    }
    else{
        call = HomeViewController.callFun(input: "deviceUncertify", contract:
            contract)!(address)
    }
    let gasPrice = try! HomeViewController.web3.eth.gasPrice().wait()
    guard let transaction = call.createTransaction(nonce: nonce, gasPrice: gasPrice,
        maxFeePerGas: gasPrice, maxPriorityFeePerGas: 500000, gasLimit: 500000, from:
        HomeViewController.walletAddress, value: 0, accessList: [:], transactionType:
        .eip1559)
    else {
        return
    }
    let signedTx = try! transaction.sign(with: HomeViewController.privateKey, chainId:
        11155111).guarantee.wait()
    firstly {
        HomeViewController.web3.eth.sendRawTransaction(transaction: signedTx)
    }.done {
        recipt in HomeViewController.web3.eth.getTransactionByHash(blockHash: recipt){
            txRecipt in print(txRecipt)
        }
    }.catch { error in
        print(error)}
}
```

Figure 3.11: Implementation of certify .

The static function certify necessitates the provision of the device's address and its certification status as input parameters. This information is crucial for invoking the appropriate function within the smart contract. Unlike previous functions, the certify function requires the creation of a transaction object, which entails the specification of parameters such as nonce, gas price, and others. The transaction is signed with the private key, accompanied by a chain ID, and transmitted to the Ethereum network. The function's outcome is reflected in the transaction receipt, which is retrieved by its hash and printed to indicate whether the transaction was successful or not. Additionally, any errors that may occur during the process are also printed. (Figure 3.11)

Written by: Parshina Daria

3.4 IoT devices

The astonishing capacity of physical items to connect and interact over the Internet in an easy-to-use manner is known as the Internet of Things (IoT). These items are made to competently collect important information from their environment, which eventually results in more cost-effective procedures, higher-quality products, and more efficient operations. An IoT system's physical devices, which are frequently furnished with sensors and actuators, constitute its central component.

The IoT microprocessors are able to connect, collect data, and make intelligent decisions on their own, leading to previously unheard-of levels of convenience and productivity.(You can see a example of IoT devices being used to help in a supply chain in the figure 3.12) [4] The gadgets' inbuilt sensors gather data so that, in response to commands, they may carry out particular tasks.

Figure 3.12: IoT devices in a supply chain.

Blockchain technology is now widely used in digital money transactions. There have also been encouraging recent advances in sensor technology (Internet of Things). Blockchain technology allows people to exchange content that they might not be able to do otherwise. Businesses run the danger of suffering major inefficiencies and facing a big increase in processing capacity if they do not optimize the distributed ledger using IoT.

3.4.1 Microprocessors

Microprocessors are essential to IoT systems since they carry out a multitude of functions. These responsibilities include operating system (OS) administration, programme execution, power management, actuator activation, data collecting, and device-to-device communication. When it comes to consumer microprocessors, the Raspberry Pi is a popular option. It is ideal for development because of its small size and economical nature.

Because of its ARM architecture, users can create apps for well-known operating systems like Linux. Prominent microprocessor manufacturers including AMD, Qualcomm, and Intel provide compelling solutions for use in enterprise settings. It is critical to understand that these essential elements are what make the Internet of Things (IoT) possible as a whole. They serve as the foundation for IoT device functionality and allow the sector to reach its enormous potential.

3.4.2 Microcontrollers

The Internet of Things (IoT) relies heavily on microcontrollers because they can monitor and control a wide range of systems and devices. These gadgets provide economical, portable, and energy-efficient solutions, which make them ideal for carrying out particular activities on a variety of devices.

In our study, we believe that the Arduino Uno microcontroller is the best option for prototyping.(The figure 3.13 describes the connections of our arduino circuit). One of the Arduino Uno's unique selling points is its ability to connect to computer systems through a virtual serial connection. This improves the overall effectiveness and functionality of Internet of Things applications by making it simple to control and coordinate a variety of input-output devices.

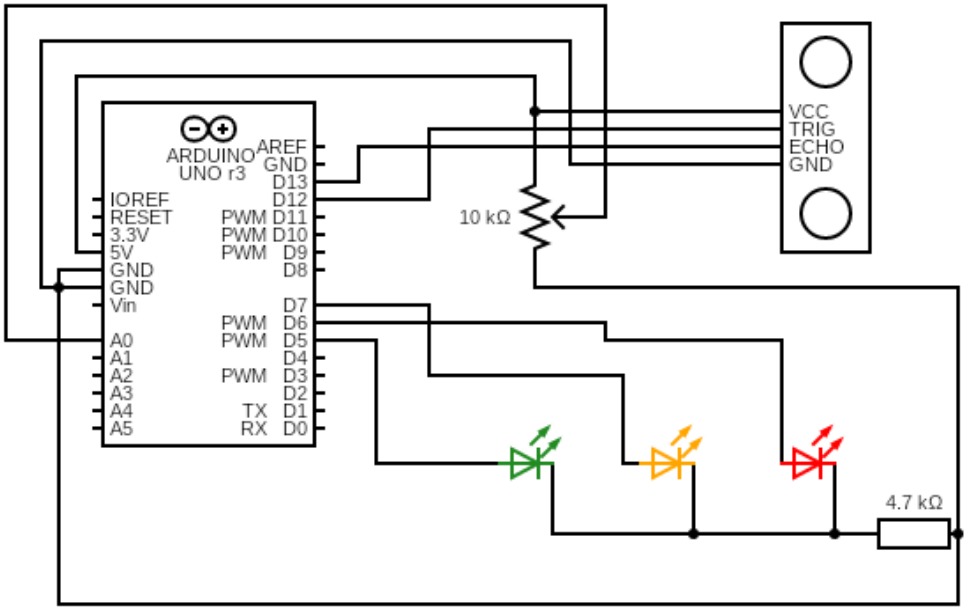

Figure 3.13: The Arduino circuit and connections sketch.

We are using the Potentiometer and Ultrasonic sensors as input sensors.

The Potentiometer is a variable resistor that allows users to turn a physical knob or dial to input analogue signals. Users can adjust and modify the desired output or response by accurately adjusting the knob's position, which increases the customization and flexibility of different electronic systems.

The Ultrasonic sensor is a non-contact type of sensor used to measure an object's distance and velocity.

This sensor operates on sound wave property to measure the velocity and distance of the object.

On the other hand, we are using led lights as output devices. We have three colors of leds, green, yellow, and red. These leds light up according to the value of the potentiometer, to give us a physical type of output.

3.4.3 IoT System Design

We link Raspberry Pi and Arduino so that values from the sensor in Arduino are sent to Raspberry Pi via a straightforward serial communication.

The Raspberry pi receives these sensor values, and calculates the maximum for each sensor every 30 seconds, and then sends it to the Blockchain to share it with our mobile application. We are using the Arduino uno, and raspberry pi 3 model B+ in our project.

The circuit consists of: An Arduino Uno will be used to load a programme that will accept input from 10 k potentiometers and, depending on the input range, emit a signal through various LEDs.

The LEDs that will light up according to the analogue input range inputted from the potentiometer.

An input potentiometer of 10 k is employed. It is a three-terminal variable resistor that may have its resistance changed with a knob or dial. The amount of resistance varies with the rotation of the potentiometer, producing a distinct analogue input. The analogue readout ranges from 0 to 1023 (but in our project we mapped its value to be between 0-100 for convenience) when the shaft's rotation fluctuates between 0 and 5V. The voltage applied to the pin determines this range.

We simulate the input manually (by turning the knob) and use the potentiometer as an example to show how we may change the input value to create the desired environment. Using a temperature sensor or pressure sensor, for instance, could not have been the best option because doing so would have required setting up the environment externally in order to manage it according to our input. But in real life, any type of sensor can be used to maximize efficiency according to the environment.

Ultrasonic sensor is also used to measure speed and distance. We have used an ultrasonic sensor because of its flexibility in changing values and demonstrating our work. Jump wires of d are used for communication among different entities.(Our Arduino circuit is shown in the figure 3.14).

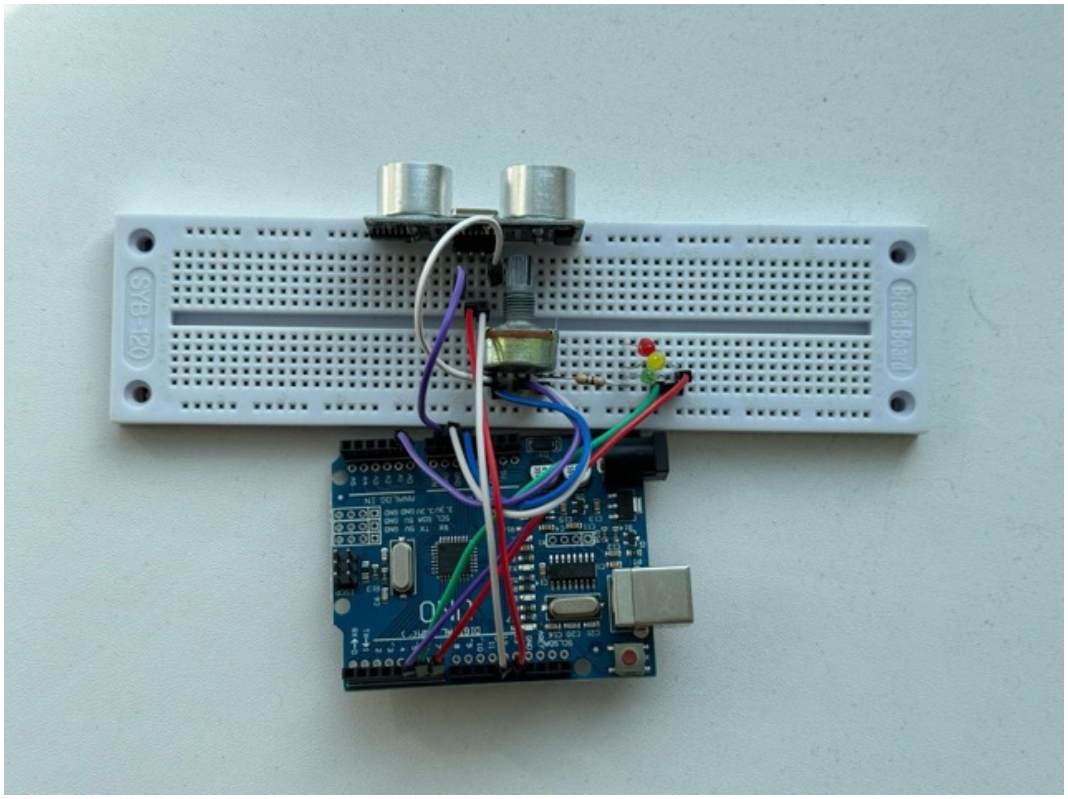

Figure 3.14: Our Arduino Circuit.

Algorithm 7 Arduino Circuit Code

1: $GREEN \leftarrow D5$
2: $YELLOW \leftarrow D7$
3: $RED \leftarrow D6$
4: $TRIG \leftarrow D12$
5: $ECHO \leftarrow D13$
6: $ANALOG\_IP \leftarrow A0$
7: **function** SETUP()
8: pinMode $GREEN \leftarrow Output$
9: pinMode $YELLOW \leftarrow Output$
10: pinMode $RED \leftarrow Output$
11: pinMode $TRIG \leftarrow Output$
12: pinMode $ECHO \leftarrow Input$
13: **end function**
14: **loop**
15: $value1 \leftarrow analogRead(ANALOG\_IP)$
16: $value \leftarrow map(value1, 0, 1023, 0, 100)$
17: **if** $value \leq 69$ **then**
18: digitalWrite $GREEN \leftarrow HIGH$
19: **end if**
20: **if** $value \leq 85$ **and** $value \geq 70$ **then**
21: digitalWrite $YELLOW \leftarrow HIGH$
22: **else**
23: digitalWrite $RED \leftarrow HIGH$
24: **end if**
25: digitalWrite $TRIG \leftarrow HIGH$
26: Delay(1000)
27: digitalWrite $TRIG \leftarrow LOW$
28: duration $\leftarrow$ pulseIn $(ECHO \leftarrow HIGH)$
29: distance $\leftarrow (duration/2)/28.5$
30: Print(duration)
31: Print(value)
32: **end loop**

3.4.4 Raspberry PI

Our prototype utilizes the Raspberry Pi 3, a low-cost computer that is about the size of a credit card. It can be connected to input devices such as a keyboard and mouse and can be displayed on external devices like monitors or televisions. The device is powered by a BCM2837B0 System-on-Chip (SoC) that includes a 1.4 GHz quad-core ARMv8-A 64-bit processor. (The raspberry pi sketch is shown in the figure 3.15) [11]

We used a USB flash driver to plug in the operating system of the raspberry pi.

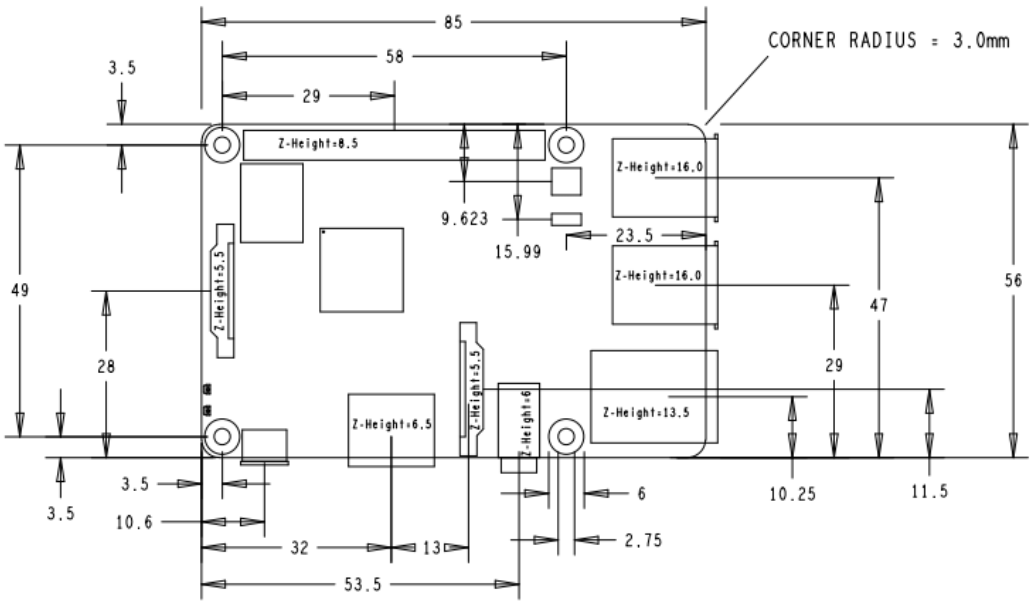

Figure 3.15: Raspberry pi sketch.

We execute the raspberry pi code by using a remote desktop using SSH connection.

SSH connection protocol is a method of securely sending commands to a computer over an unsecured network. We need to confirm that the Raspberry Pi and our device are linked to the local network. To continue, we will require the device's local address. To enhance security, SSH is turned off by default to stop hackers from gaining unauthorized access. To provide the highest level of security, a private key is kept inside the gadget. We must enter the password that has been added and configured on the Raspberry Pi in order to access the device. For increased security, it is advised that we modify the default password.

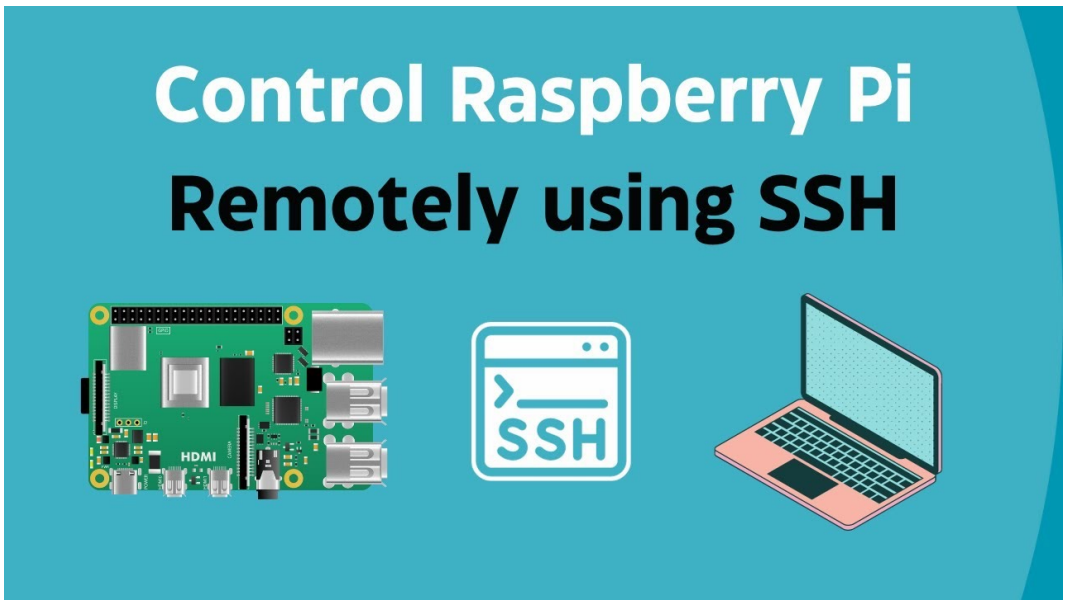

Figure 3.16: SSH connection protocol.

[13]

Algorithm 8 IoT Contract Communication

Required: Smart Contract Address, ABI

Required: Blockchain API URL

Required: Dotenv with privatekey

Required: timeGap ← From Smart Contract

Import required libraries
Establish Connection with Network
productID ← ID of Product
iotc ← contractInstance(Address,ABI)
function SendPotentiometerMetric(value)
    Broadcast transaction passing $productID$ and $value$
end function
function SendUltrasonicMetrics(value)
    Broadcast transaction passing $productID$ and $value$
end function
Import PrivateKey
Connect to Arduino Port
starting_time ← currentTime
tranaction_value ← 0
while True do
    sensorValue ← Read Sensor Value
    if sensorValue > 0 **then**
        sensorValue ← Value from Arduino
        if sensorValue > tx_value **then**
            tx_value ← sensorValue
        end if
        if currentTime ≥ starting_time + timeGap **then**
            INJECT_TRANSACTION(tx_value)
            starting_time ← currentTime
            tx_value ← 0
        end if
    end if
end while

3.4.5 Device Registration

In a supply chain scenario, the administrator controls the Internet of Things (IoT) devices. On the smart contract, the administrator registers the Internet of Things device. The authorization of the IoT device to engage in supply chain operations is explicitly acknowledged by this registration. The IoT devices are mounted on a transport vehicle so that it can actively interact with the blockchain network. Throughout transit, the Internet of Things devices stay in constant communication with the blockchain, allowing it to add transactions. These transactions usually relate to different phases of the product, like location updates, temperature tracking, or other relevant information.

Using a private key and the address that corresponds to its registration on the smart contract, the IoT device is authenticated. Only the approved IoT device will be able to communicate with the blockchain network and initiate transactions due to this authentication method.

Deregistering the Internet of Things device from the smart contract may also be part of the reset procedure. This procedure guards against any misuse or modification of the IoT device after its function in the supply chain has been completed.

This device is communicating with a smart contract within a designated time frame. Users can view information from the device through a user interface.

Figure 3.17: Device registration.

[10]

3.4.6 Security

In order to configure a secure Wi-Fi network and enable internet access and communication with other devices on the network, the device is linked to the local network. A secure connection keeps unauthorized users out, and the Raspberry Pi configuration has a number of security measures. Furthermore, SSH is on by default to stop hackers from accessing the device and its private key, which is encrypted with crucial data. Furthermore, the Raspberry pi needs to be logged in separately using the password linked to the Raspberry Pi since it is password protected. By implementing an additional security layer, you can be sure that only people with permission can use the device and its resources.

The figure 3.18 describes how security is always important in all situations:

Figure 3.18: Security visualization.

Written by: Seropian Nshteh

3.5 Experiment

To ensure the seamless integration and functionality of all three components - device, blockchain, and mobile application - a comprehensive experiment was conducted. The experiment aimed to validate the following key aspects:

The initial step involved activating the device to initiate data collection from sensors and transfer to the blockchain. This process is crucial to ensure the device functions as intended and successfully communicates with the blockchain network.

Also, the experiment focused on verifying the blockchain's ability to receive data from the device without errors, store it securely, and update it as required. This step is essential for maintaining data integrity and ensuring the accuracy of information stored on the blockchain.

The final phase of the experiment involved testing the mobile application's capability to display accurate data retrieved from the blockchain. The application was expected to continuously update data in real-time, reflecting the ongoing operations of the blockchain. Additionally, the application was designed to halt data updates if the blockchain ceased functioning, ensuring synchronization between the mobile interface and the blockchain network.

The experimental findings demonstrate that the coordinated operation of the three components proceeded seamlessly and error-free, aligning with the anticipated outcome, thereby confirming the success of the integration process. This outcome underscores the efficacy of the integrated system in achieving harmonious functionality across its constituent parts, validating the hypothesis of smooth and error-free operation when these components work in unison.

On the basement of accomplished work, there was created a paper "Empowering Autonomous IoT Devices in Blockchain through Gasless Transactions", which is applied for International Conference on Blockchain Research and Applications (BCRA 2024)

4 Conclusion

In conclusion, a major leap in supply chain management has been demonstrated by the creation of a blockchain-based management system for package tracking devices that integrates IoT technology. This study demonstrates how decentralised technology may effectively ensure security, and transparency in devices' administration.

The implementation included programming IoT devices, installing sensors to detect changes in the environment, developing a smart contract with a hierarchical role allocation structure, and a mobile application for UI. Through communication between the sensors and the blockchain, administrator is able to check the status and conditions of the devices. The technology guards against unauthorised access and data modification by utilising blockchain's decentralised structure.

Smart contract implementation streamlines the workflow, cuts down on delays, and enhances decision-making. The integration of IoT devices, specifically using Arduino and Raspberry Pi, further enhances the system by enabling real-time data collection and transmission to the blockchain. This ensures that environmental conditions, which potentially can be temperature and humidity, are continuously monitored and recorded, providing crucial information for maintaining product quality during transit.

Future development might concentrate on enlarging the programming software by addition other features besides management of devices and enhancing the system's functionality by adding machine learning algorithms to improve decision-making processes and integrating advanced analytics for predictive maintenance. Furthermore, expanding the system's compatibility with additional IoT sensor and device types would increase its application across a wider range of businesses.

This project, taken as a whole, demonstrates the revolutionary potential of using IoT, blockchain technology, and mobile development in supply chain management, providing a safe and scalable solution that improves product integrity, operational efficiency, and convenience.

5 Important links

GitHub Repository: https://github.com/SashaKoretkevich/2024-HSE-Project-Blockchain-in-Supply-Chain

References

[1] Amazon. *Amazon managed blockchain*. URL: https://aws.amazon.com/managed-blockchain/ (visited on May 27, 2024).

[2] Amazon. *What is Internet of Things, IoT?* URL: https://aws.amazon.com/ru/what-is/iot/ (visited on Feb. 6, 2024).

[3] Apple. *Swift*. URL: https://www.apple.com/in/swift/#:~:text=Swift%20is%20a%20robust%20and,Apple%20TV%20and%20Apple%20Watch (visited on Feb. 6, 2024).

[4] Atlasico. URL: https://images.app.goo.gl/tqep5mB7y2keC5Nf7 (visited on May 27, 2024).

[5] BitDegree. *What is Testnet?* URL: https://www.bitdegree.org/crypto/learn/crypto-terms/what-is-testnet (visited on Feb. 6, 2024).

[6] IBM. *What are smart contracts on blockchain?* URL: https://www.ibm.com/topics/smart-contracts (visited on Feb. 6, 2024).

[7] Investopedia. *The Supply Chain: From Raw Materials to Order Fulfillment*. URL: https://www.investopedia.com/terms/s/supplychain.asp (visited on Feb. 6, 2024).

[8] Angwei Law. "Smart Contracts and their Application in Supply Chain Management". In: 2017.

[9] Mangtas. *What is Solidity?* URL: https://www.mangtas.com/solidity#:~:text=Solidity%20is%20an%20object%2Doriented,smart%20contracts%20on%20Blockchain%20platforms (visited on Feb. 6, 2024).

[10] Pngtree. URL: https://images.app.goo.gl/XZGos6d87466QD7VA (visited on May 27, 2024).

[11] ResearchGate. URL: https://images.app.goo.gl/jmnn8e1mzMZFdnGd6 (visited on May 27, 2024).

[12] Shardeum. *16 Key Web3 Programming Languages for 2023 – A Complete Checklist*. URL: https://shardeum.org/blog/web3-programming-languages/ (visited on Feb. 6, 2024).

[13] Shilleh. URL: https://images.app.goo.gl/yzvSSLb4G3dS6DZz6 (visited on May 27, 2024).

[14] SimpliLearn. *What Is Raspberry Pi? Here's The Best Guide To Get Started*. URL: https://www.simplilearn.com/tutorials/programming-tutorial/what-is-raspberry-pi#:~:text=The%20Raspberry%20Pi%20is%20a,a%20modified%20version%20of%20Linux (visited on Feb. 6, 2024).

[15] Synopsys. *Blockchain*. URL: https://www.synopsys.com/glossary/what-is-blockchain.html (visited on Feb. 6, 2024).

[16] Wikipedia. *Arduino Uno*. URL: https://en.wikipedia.org/wiki/Arduino_Uno (visited on Feb. 6, 2024).