CHAPTER 1

INTRODUCTION

Chapter 1

Introduction

1.1 An overview

Supply chains tend to be distributed, complex and involve a growing number of parties. To deal with the intricacy of modern logistics, supply chain companies are upgrading their ability with technologies like IoT to accurately track assets.

Data collected from IoT devices attached to assets can bring greater real-time understanding and coherency in a complicated supply chain environment, but to do so, IoT networks need to have significant reach, be able to provide more detailed information than just the location of a given asset, and enable a sense of accountability amid an ecosystem of parties that might otherwise point fingers at one another when something goes wrong.

For example, a factory relies on raw material deliveries could learn through an IoT device on the shipment that it is headed in the wrong direction. Having that information in real time could allow the factory to cancel the shipment, help redirect it or buy it from another source and keep its supply chain schedule on track. Conversely, if the factory has no such data and the shipment fails to arrive on schedule, its ability to process the materials and deliver goods out to its own end customers could be severely hampered.

With the advancement of modern technologies, Appearance of Blockchain and Internet of Things (IoT), organizations are trying to adopt such new technologies to develop and implement autonomous strategic technology trends.

IOT for logistics and assets solutions is a new topic of demand. With its ever-expanding need for smarter supply chain management services, the logistics, and transportation industry companies have been a key strength of the Internet of Things Logistics Solutions. As obtaining the good product to the right client at the accurate time, right place and right condition in the right volume and at the right amount, is growingly a challenge, the logistics industry is spending money in logistics solutions, strongly represented by the Industrial IoT.

With the arrival of IoT, Internet connections now expand to physical devices that are not computers in the classic sense. An IoT connected vehicle can intelligently estimate its own maintenance requirements. A connected street light can sense the

presence of vehicles and send a signal to drivers. By 2020, it is estimated that there will be 5 billion connected Logistics [1].

Internet of things (IoT) solutions are being successfully adopted in many different industries, such as healthcare, warehousing, transportation, and logistics. Current centralized, cloud-based IoT solutions may not scale and meet the security challenges faced by large-scale enterprises. The use of blockchain as a distributed ledger of transactions and peer-to-peer communication among participating nodes can solve such problems. In this part we will define each technology that we use to serve this community of industries.

1.2 IOT

The Internet of Things (IoT) also called the Internet of Everything or the Industrial Internet, is a new technology paradigm envisioned as a global network of machines and devices capable of interacting with each other. The IoT is recognized as one of the most important areas of future technology and is gaining vast attention from a wide range of industries. The true value of the IoT for enterprises can be fully realized when connected devices are able to communicate with each other and integrate with vendor-managed inventory systems, customer sup- port systems, business intelligence applications, and business analytics [2].

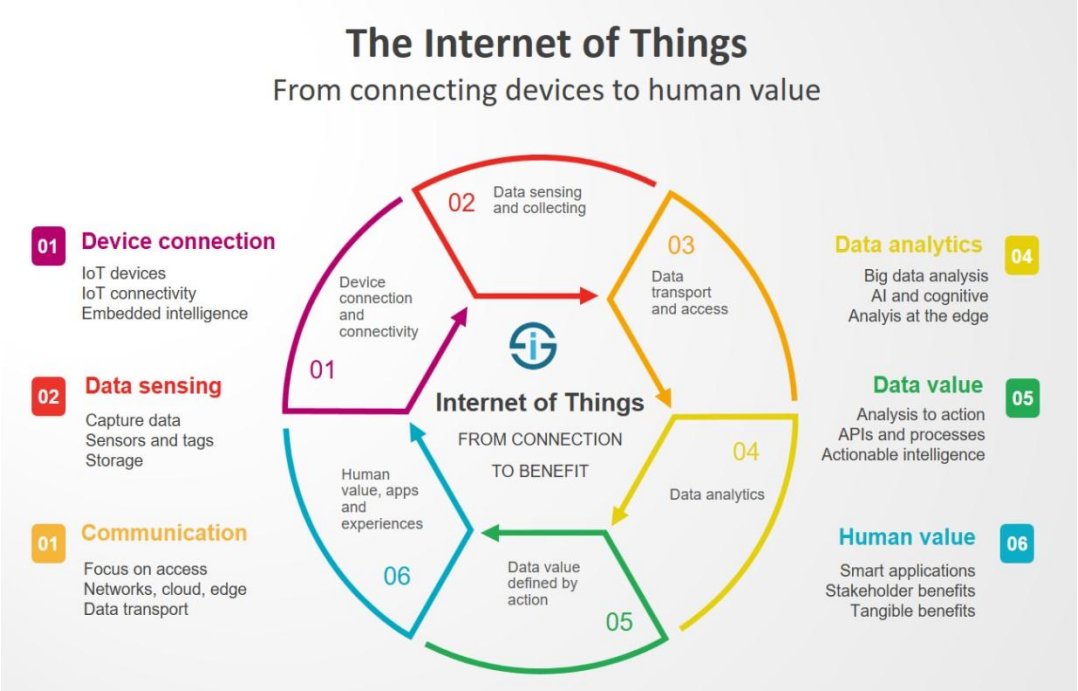


Figure 1.1 Internet of things applications fields

The IoT facilitates the development of myriad industry-oriented and user-specific IoT applications. Whereas devices and networks provide physical connectivity, IoT applications enable device-to- device and human-to-device interactions in a reliable and robust manner.

IoT applications on devices need to ensure that data/messages have been received and acted upon properly in a timely manner. For example, transportation and logistics applications monitor the status of transported goods such as fruits, fresh-cut produce, meat, and dairy products.

During transportation, the conservation status (e.g., temperature, humidity, shock) is monitored constantly and appropriate actions are taken automatically to avoid spoilage when the connection is out of range. For example, FedEx uses SenseAware to monitor the temperature, location, and other vital signs of a package, including when it is opened and whether it was tampered with along the way.

While device-to-device applications do not necessarily require data visualization, more and more human-centered IoT applications provide visualization to present information to end users in an intuitive and easy-to-understand way and to allow interaction with the environment.

It is important for IoT applications to be built with intelligence so devices can monitor the environment, identify problems, communicate with each other, and potentially resolve problems without the need for human intervention.

1.3 Blockchain

In general terms, a blockchain is an immutable transaction **ledger**, maintained within a distributed network of peer nodes. These nodes each maintain a copy of the ledger by applying transactions that have been validated by a consensus protocol, grouped into blocks that include a hash that binds each block to the preceding block.

1.3.1 blockchain ledger

A blockchain ledger is often described as **decentralized** because it is replicated across many network participants, each of whom **collaborate** in its maintenance. We'll see

that decentralization and collaboration are powerful attributes that mirror the way businesses exchange goods and services in the real world.

- There's a **world state** – a database that holds a cache of the **current values** of a set of ledger states. The world state makes it easy for a program to directly access the current value of a state rather than having to calculate it by traversing the entire transaction log. Ledger states are, by default, expressed as **key-value** pairs. The world state can change frequently, as states can be created, updated and deleted.
- There's a **blockchain** – a transaction log that records all the changes that have resulted in the current the world state. Transactions are collected inside blocks that are appended to the blockchain – enabling you to understand the history of changes that have resulted in the current world state. The blockchain data structure is very different to the world state because once written, it cannot be modified; it is **immutable**.

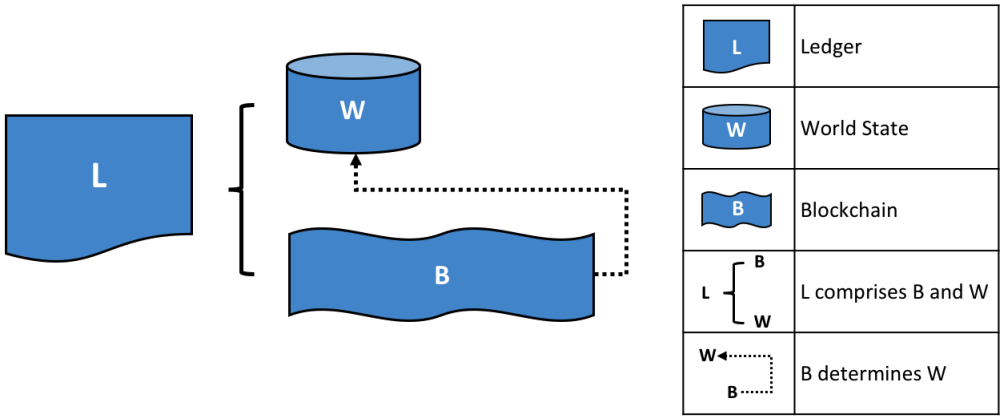


Figure 1.2 component of ledger

A Ledger L comprises blockchain B and world state W, where blockchain B determines world state W. We can also say that world state W is derived from blockchain B.

1.3.2 blockchain peers

A blockchain network is comprised primarily of a set of peer nodes (or, simply, peers). Peers are a fundamental element of the network because they host ledgers and smart contracts.

Peers host the ledger and smart contracts and make up the physical structure of a Blockchain network

smart contracts with a technology concept which is called a chaincode — simply a piece of code that accesses the ledger, written in one of the supported programming languages.

1.3.3 Ledgers and Chaincode

Let's look at a peer in a little more detail. We can see that it's the peer that hosts both the ledger and chaincode. More accurately, the peer actually hosts instances of the ledger, and instances of chaincode. Note that this provides a deliberate redundancy in a Fabric network — it avoids single points of failure.

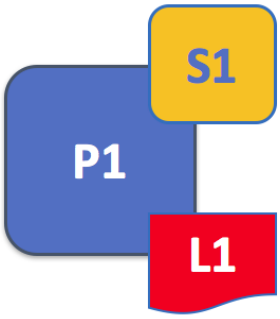


Figure 1.3 peer

Because a peer is a host for ledgers and chaincodes, applications and administrators must interact with a peer if they want to access these resources. That's why peers are considered the most fundamental building blocks of a Blockchain network.

1.3.4 Multiple Ledgers and Chaincodes

There isn't a fixed relationship between the number of ledgers a peer has and the number of chaincodes that can access that ledger. A peer might have many chaincodes and many ledgers available to it.

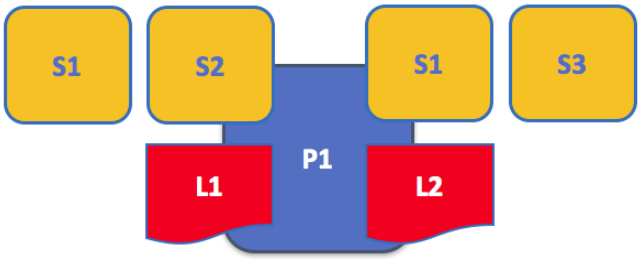


Figure 1.4 Multiple Ledgers And Chaincodes

1.3.5 Supply chain

Supply chain (SC) is a set of processes and entities (suppliers, customers, factories, distributors and retailers) which are interested to fulfill customer order. The plan, source, make, deliver, return and enable are the main processes of SC according to Supply Chain Operations Reference Model (SCOR) firms.
Supply chain management (SCM) means having the correct item in the correct volume at the correct time at the correct place for the correct price in the correct condition to the correct customer [3].
In traditional supply chain management systems, there exist several problems such as overstocking, delivery delays and stock out. These problems return to several factors such as complexity and uncertainty, which exist usually in real supply chains.

The cheaper, better and faster item is the desirable for SC managers. Also maximizing surplus which is the whole payments from end customers minus all costs, which incurred via SC. Traditional supply chains are becoming more costly, complex and vulnerable. To overcome these challenges, the supply chains must be smarter.

We can define smart supply chain as a modern and interconnected system, which expands from separated, regional and single firm applications to wide and systematic implementation of supply chains.

For effective management of supply chain, the information technology (IT) plays a very important role [4]. The IT has ability to integrate different processes, suppliers and customers internally and externally via enhancing communication, collection and transfer of data and information and then improve supply chain performance.
One of the most important development of information technology is the internet of things (IOT). The term IOT has coined by Kevin Ashton in 1999 [4]. We can define it as a set of physical and virtual objects, which are connected together via a network for communication and sensing or interaction with internal and external environment.

1.3.6 Smart Contract

A smart contract is a computer program that directly controls the transfer of digital currencies or assets between parties under certain conditions. A smart contract not only defines the rules and penalties related to an agreement in the same way that a traditional contract does, but it can also automatically enforce those obligations [6].

It does this by taking in information as input, assigning a value to that input through the rules set out in the contract and executing the actions required by those contractual clauses -- for example, determining whether an asset should go to one person or should be returned to the other person from whom the asset originated.

These contracts are stored on blockchain technology, a decentralized ledger that also underpins bitcoin and other cryptocurrencies.

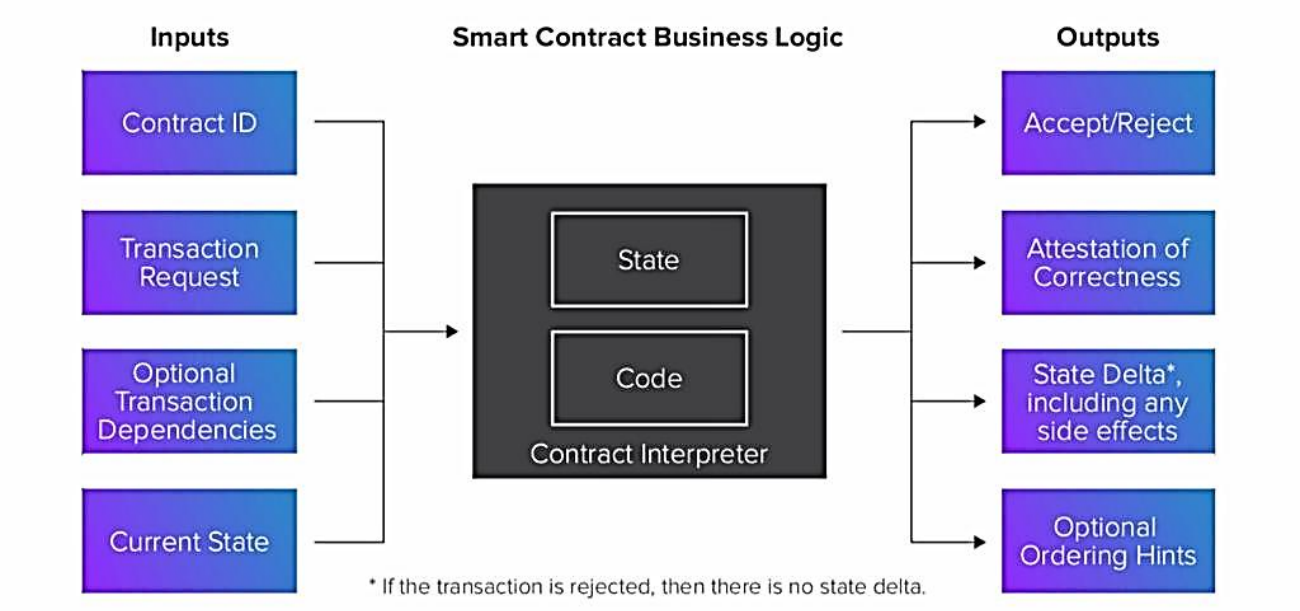


Figure 1.5 smart contract execution

Smart contracts are complex, and their potential goes beyond the simple transfer of assets -- they can execute transactions in a wide range of fields, from legal processes to insurance premiums to crowdfunding agreements to financial derivatives. Smart contracts have the potential to disintermediate the legal and financial fields; in particular, by simplifying and automating routine and repetitive processes for which people currently pay lawyers and banks sizable fees.

The role of lawyers could also shift in the future as smart contracts gain traction in areas from adjudicating traditional legal contracts to producing customizable smart contract templates. Additionally, smart contracts' ability not only to automate processes, but also to control behaviour, as well as their potential with real-time auditing and risk assessments, can be beneficial to compliance.

1.3.6.1 Smart contract applications and blockchain

Blockchain is ideal for storing smart contracts because of the technology's security and immutability. Smart contract data is encrypted on a shared ledger, making it impossible to lose the information stored in the blocks [5].

Another advantage of blockchain technology being incorporated into smart contracts is flexibility. Developers are able to store almost any type of data within a blockchain, and they have a wide variety of transaction options to choose from during smart contract deployment.

1.3.6.2 Smart contract advantages and disadvantages

There are several potential business advantages when using smart contracts.

Cost-efficiency: Smart contracts eliminate many operational expenses and save resources, including the personnel needed to monitor their progress.

Processing speed: Smart contracts run on automated processes and, in most cases, can eliminate human involvement, increasing the speed of business transactions stipulated in the contract.

Autonomy: Smart contracts are performed automatically by the network, eliminating the need and associated risk of a third party being involved in smart contract deployment.

Reliability: Data entered in the blockchain cannot be changed or deleted. If one party does not complete its obligations, the other will be protected by the conditions of the smart contract. The automated transactions also remove the potential for human error and ensure accuracy when executing the contracts.

There are numerous potential disadvantages to smart contracts, as well. A lack of international regulations focusing on blockchain , cryptocurrencies and smart contracts makes these technologies difficult to monitor in the global economy.

Smart contracts are also complicated to implement. They are also impossible to change, and while this is considered a security-related advantage, the parties cannot make any changes to the smart contract agreement or incorporate new details without developing a new contract.

1.3.6.3 Characteristics of a Smart Contract

Smart contracts are capable of tracking performance in real time and can bring tremendous cost savings. Compliance and controlling happen on the fly. In order to get external information, a smart contract needs information oracles, which feed the smart contract with external information.

- **Smart Contracts are**

- Self-verifying
- Self-executing
- Tamper resistant

- **Smart Contracts can**

- Turn legal obligations into automated processes.
- Guarantee a greater degree of security.
- Reduce reliance on trusted intermediaries.
- Lower transaction costs.

1.3.7 Hyperledger Composer

Hyperledger Composer is a framework or layer that builds on top of Hyperledger Fabric, that provides functionality that allows us to model blockchain business network, and easily generate REST full APIs and end user applications, and consists of:

A modeling language called CTO (an homage to the original project name, Concerto)

A user interface called Hyperledger Composer Playground for rapid configuration, deployment, and testing of a business network

Command-Line Interface (CLI) tools for integrating business networks modeled using Hyperledger Composer with a running instance of the Hyperledger Fabric Blockchain network

Hyperledger Composer modeling language (CTO)

We will use CTO to model the sample Perishable Goods network "This sample business network demonstrates how growers, shippers, and importers define contracts for the price of perishable goods, based on temperature readings received for shipping containers".

1.4 Problem Statement

Most products lose their market value (outdate) over time. Traditionally, perishables outdate due to their chemical structure. Examples of such perishable products are fresh produce, blood products, dairy products, meat, vitamins etc.

The very real problem of the safe delivery of perishable goods، that are sensitive, Every shipment of perishable goods has thresholds (refrigeration requirements,

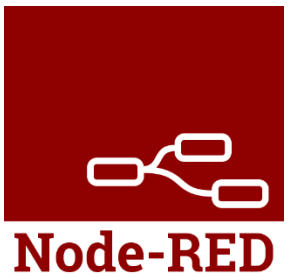
avoidance of shocks or vibration, etc. to protect the goods from contamination or damage by any number of causes, like natural disaster, theft, vandalism, fire.

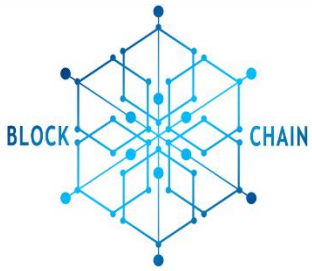
If the shipment exceeds these thresholds, the goods are damaged and might become a health hazard. In order to help mitigate the risk associated with goods in transit, one must be sure to choose the carrier carefully. All reputable carriers have substantial safeguards in place in order to reduce the likelihood that their clients will suffer severe losses. Meanwhile, the methodology of positive economics: If a cargo needs to be delivered within safe environ- mental parameters and a safe amount of time, it is extremely valuable to use an IOT Asset Tracking system that combines environmental sensors. Calculates its location via GPS, and then reports its location via Wi-Fi networks.

When multiple participants, such as farms, manufacturers, processing plants, trucks, ports, ships, distribution centers, consumer retail outlets, are involved in the safe shipment and payment of the cargo, a Hyperledger blockchain can be used to record immutable transactions as the shipment progresses through its delivery journey. By recording the details (Where, What, and When) of a shipment that experienced extreme conditions {thresholds specified in the smart contract) developers can verify that the goods were delivered successfully (or not). Then, payment is predicated on successful delivery. Tracking the conditions of the shipment across multiple participants using a blockchain provides verification and trust in these processes.

1.5 Proposed strategy

Tracking the conditions of the shipment across multiple participants using three important technologies combined together provides verification and trust.





IOT boards to provide reasonably accurate details of where the shipment has been, Sensors within the shipment can now record environmental conditions in real-time. By recording and transmitting the geolocation, environmental sensor data and the time to the Cloud that allows us to track the conditions and verify its safety, which in turn sends them to a safe environment provided by the blockchain.

IOT technology will be used to acquire data by using sensors and IOT microcontrollers without any human interaction that make it easier to get data, goods condition and degree of damaged part.

Cloud computing will save data to open a way for blockchain to get data and save it in blocks by the aid of node red.

Blockchain then will in turn take data, process it, verify it and send it to be displayed.

1.6 Project Tools

The three main components and tools of our projects are:

1) IOT
    - Sensors
        - Vibration sensor
        - Humidity sensor
        - Temperature sensor
        - GPS Module
    - Platforms
        - Arduino
    - Languages
        - C++
2) Blockchain
    - Languages
        - JavaScript
3) Dashboard
    - Platforms
        - Node-Red
    - Languages
        - Node.Js
        - html5
        - css3
        - javas cript
        - bootstrap
        - wow

        -json Api

1.7 book structure

This book is organized as follows:
Chapter 1 is introduction and theoretical background.
Chapter 2 is system analysis and defining system components with their relations.
Chapter 3 is implementation and integration for hardware and software components and their connections.
Chapter 4 is user manual to guide user to use our application.
Chapter 5 is conclusion and future work.
At the end of this book there are appendices of the main code.

CHAPTER 2
SYSTEM ANALYSIS

Chapter 2

System analysis

2.1 Proposed System Architecture

- Sender and Shipment Company receiver perform the same steps for register in blockchain, login, generate smart contracts, monitor trip, in respectively.
- Blockchain must accepted of the terms and rules from sender and Shipment Company.
- Blockchain generates ID, generate smart contracts, Decide payment.
- Transfer money.
- Microcontroller and GPS send collected data of sensors to blockchain.
- Blockchain receives collected data from microcontroller and this transfer to the cloud, the cloud transfers data to Dashboard, dashboard save all data in blockchain and display data to user, also to displays on the Map or the website.

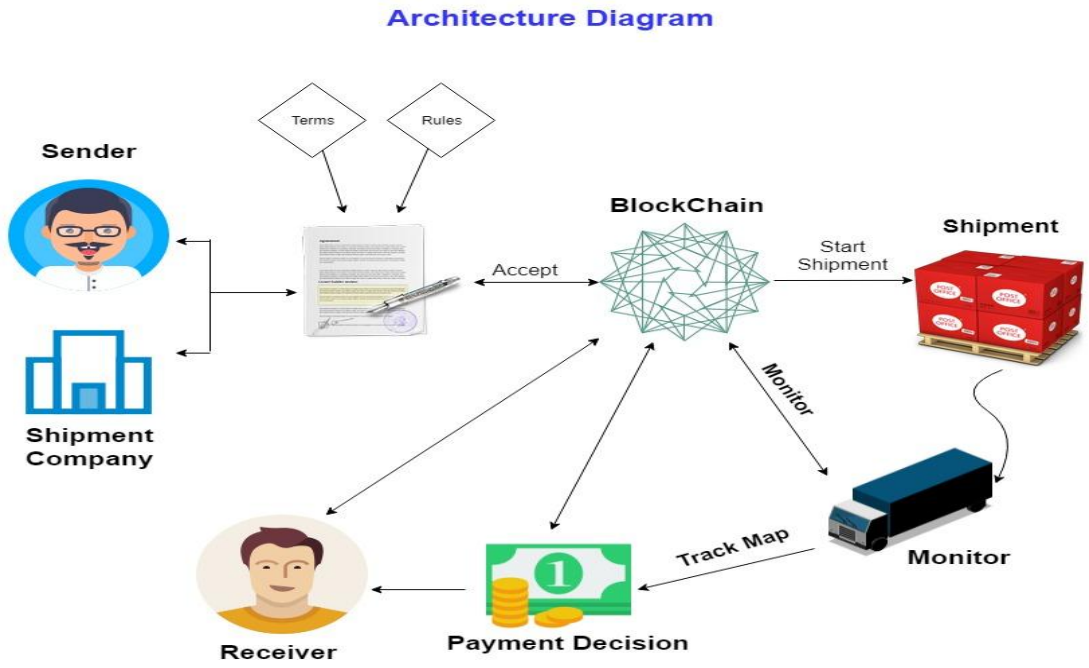


Figure 2.1 proposed system Architecture Diagram

2.2 System block diagram

- All of sensors send data to the microcontroller (GPS).
- The microcontroller takes this data & send this data to the cloud.
- The cloud sends information of data to the Dashboard (Node-Red).
- The dashboard sends and receives data to blockchain (buffer).
- The configuration of data review in the website (map).
- To use the conditions be easily of the user.

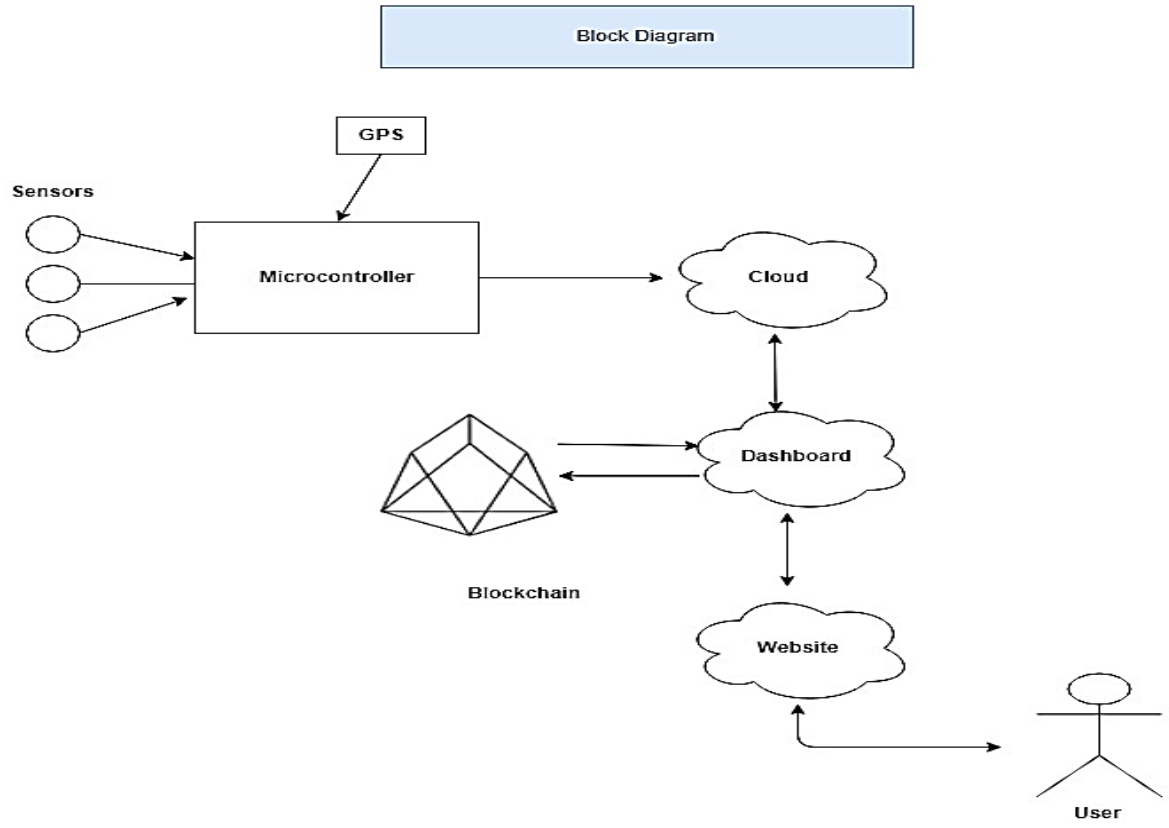


Figure 2.2 proposed system Block Diagram

2.3 IOT asset tracking using blockchain use case

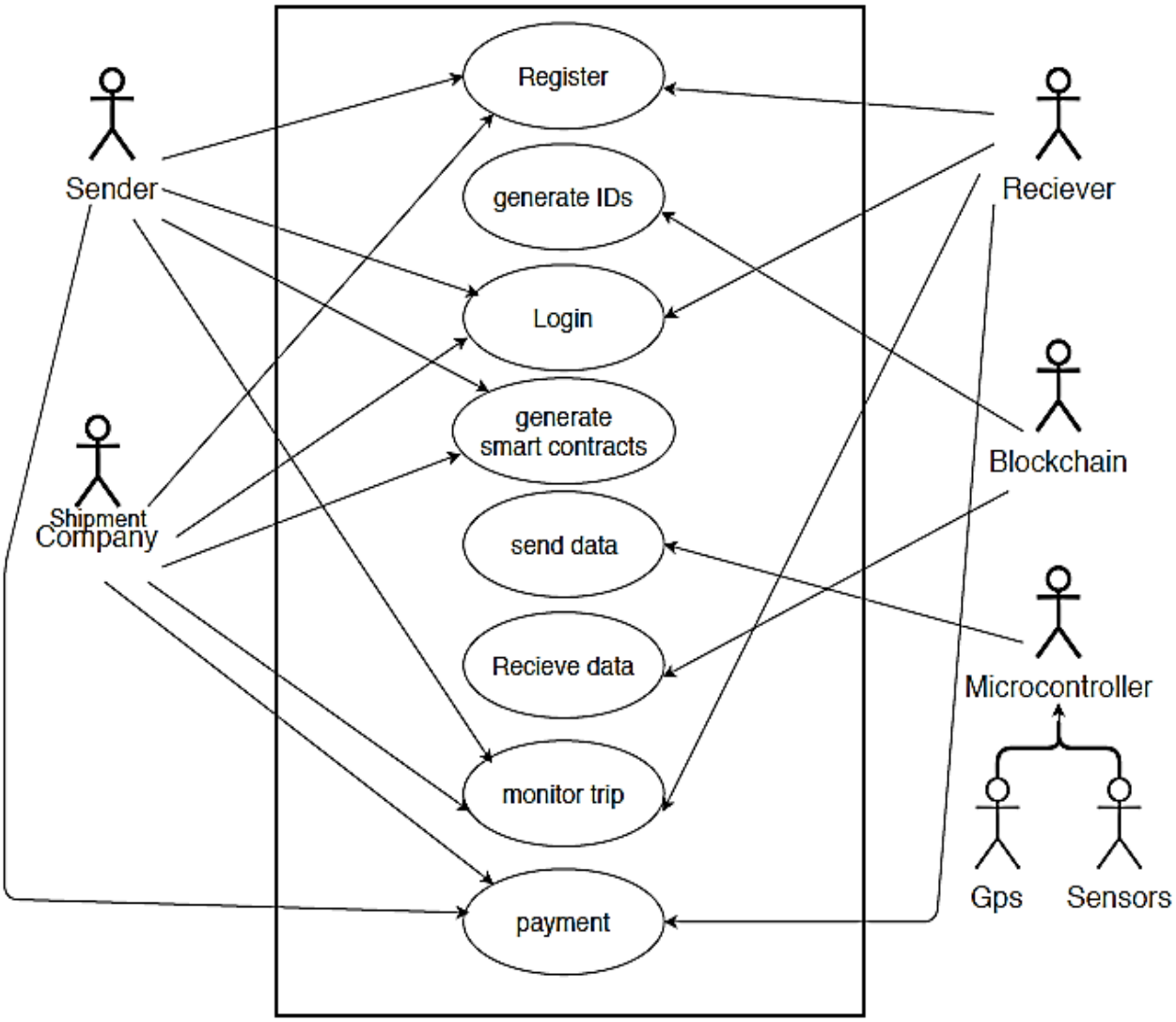


Figure 2.3 Use case diagram for IOT asset tracking using blockchain

Actors	Sender, Receiver, Shipment Company, blockchain, microcontroller, sensors, GPS, Dashboard.
Description	Sender and Shipment Company and receiver perform the same steps for register in blockchain, login, generate smart contracts, monitor trip in Respectively. Blockchain perform generate ID, generate smart contracts, Decide payment, Transfer money. Microcontroller and GPS send collected data of sensors to blockchain. Blockchain receive collected data from microcontroller and this transfer to the cloud, the cloud transfer data to Dashboard, dashboard save all data in blockchain and display data to user to displays on the Map or the website.
Data	Sender and receiver and Shipment Company uses in transfer personal data and Product data, microcontroller send sensors and GPS data, Blockchain receive data from dashboard besides microcontroller.
Response	Fast response, offering excellent quality, high-performance microcontroller.
Comments	Sensors types: **Dh11:** This sensor includes a resistive-type humidity measurement component and an NTC temperature measurement. **MPU:** is a device that implements the core elements of a computer system on a single integrated circuit (vibration). **Wi-Fi module:** that can give any microcontroller access to your Wi-Fi network. GPS module: is a navigation device that uses the Global Positioning System (GPS).

Table 2.1 description of use case diagram

2.4 use case for sending data to the blockchain

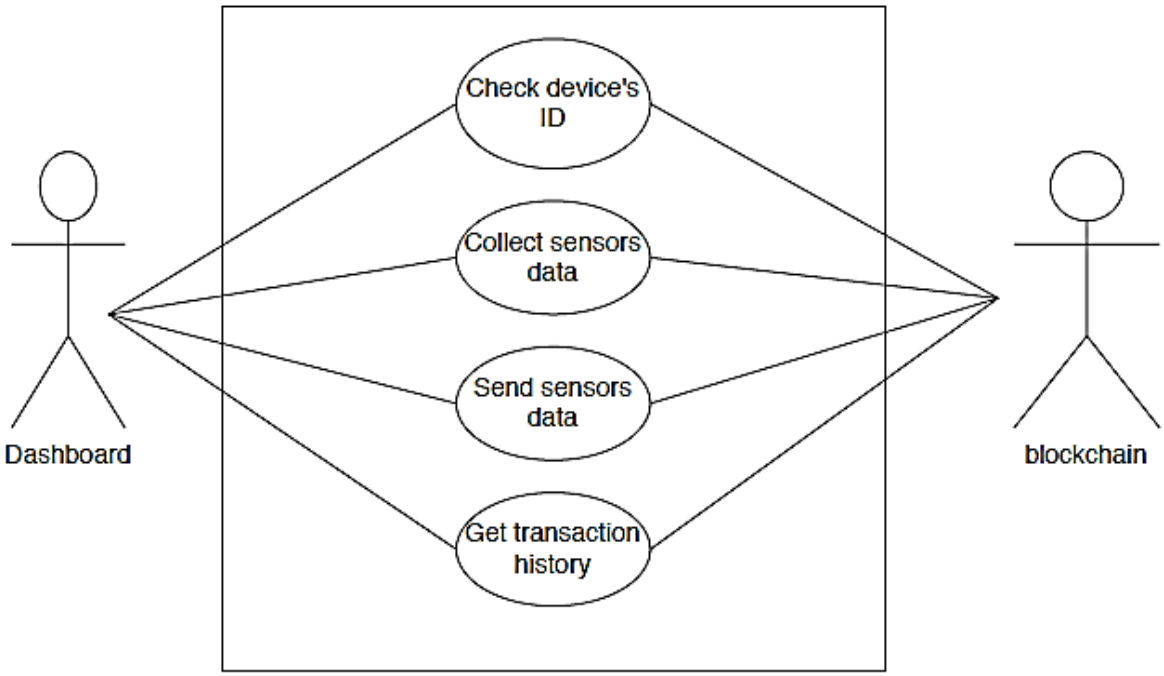


Figure 2.4 use case for sending data to blockchain

2.5 use case diagram for sending and reading data from blockchain

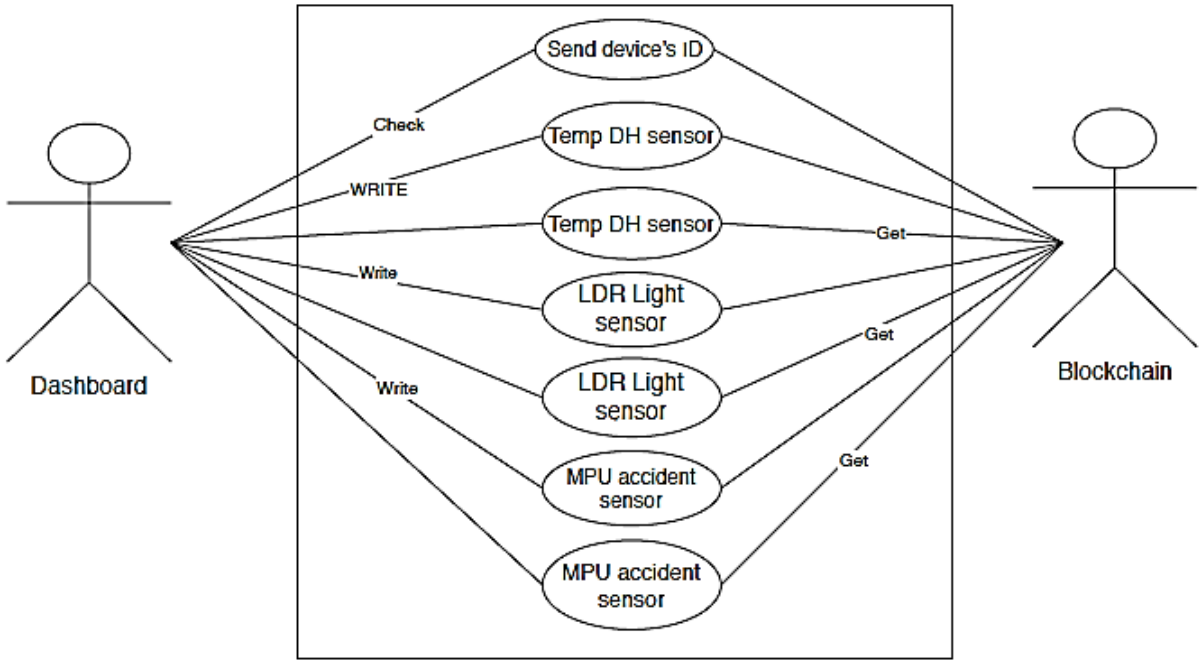


Figure 2.5 use case diagram for sending and reading data from blockchain

Description:

- Dashboard check send devices ID to Blockchain.
- Dashboard write temp DH sensor to Blockchain.
- Dashboard write temp DH sensor to get Blockchain.
- Dashboard write LDR Light sensor to get Blockchain.
- Dashboard write MPU accident sensor to get Blockchain.

2.6 Sequence diagram

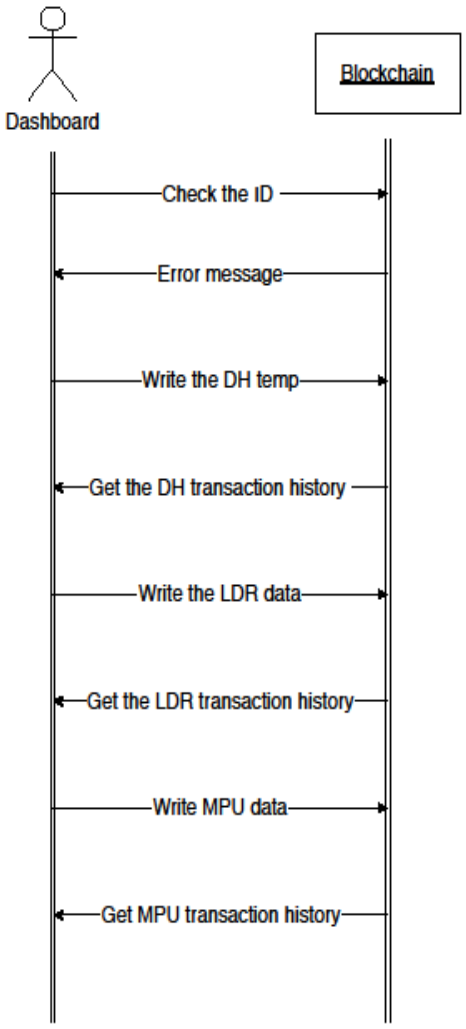


Figure 2.6 sequence Diagram

2.7 Activity diagram

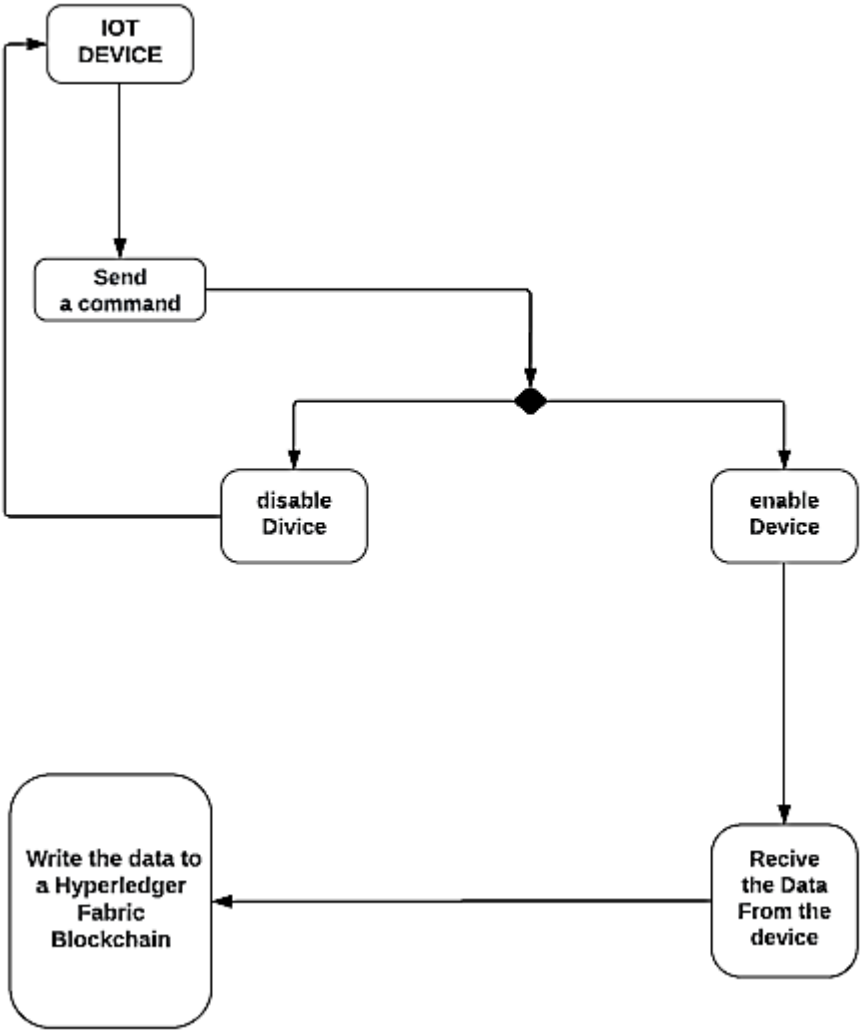


Figure 2.7 activity diagram

2.8 smart contract interaction

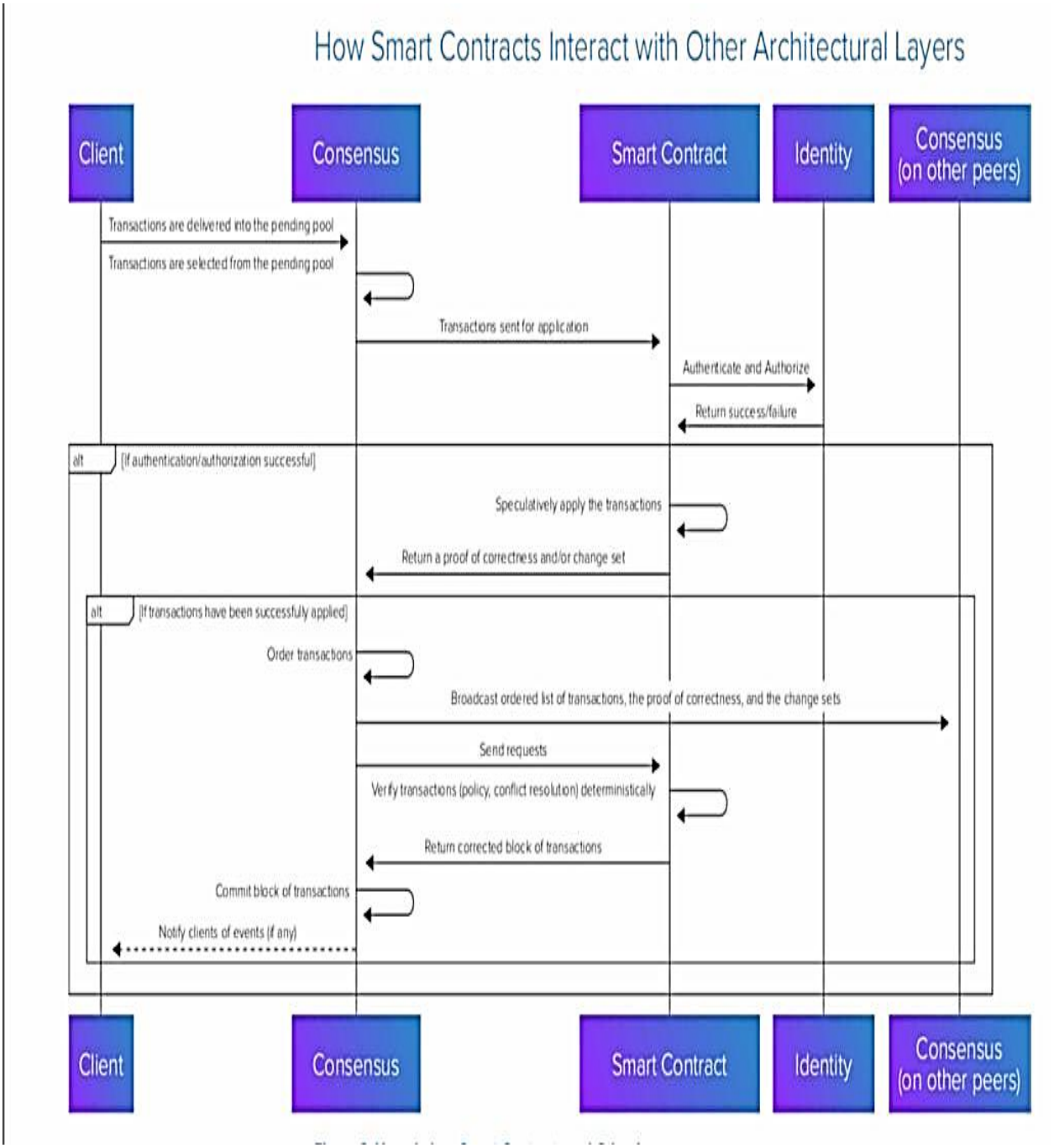


Figure 2.8 interaction of smart contract with other architectural layers

CHAPTER 3
SYSTEM INTEGRATION AND IMPLEMENTATION

Chapter 3

System integration and implementation

3.1 hardware block diagram

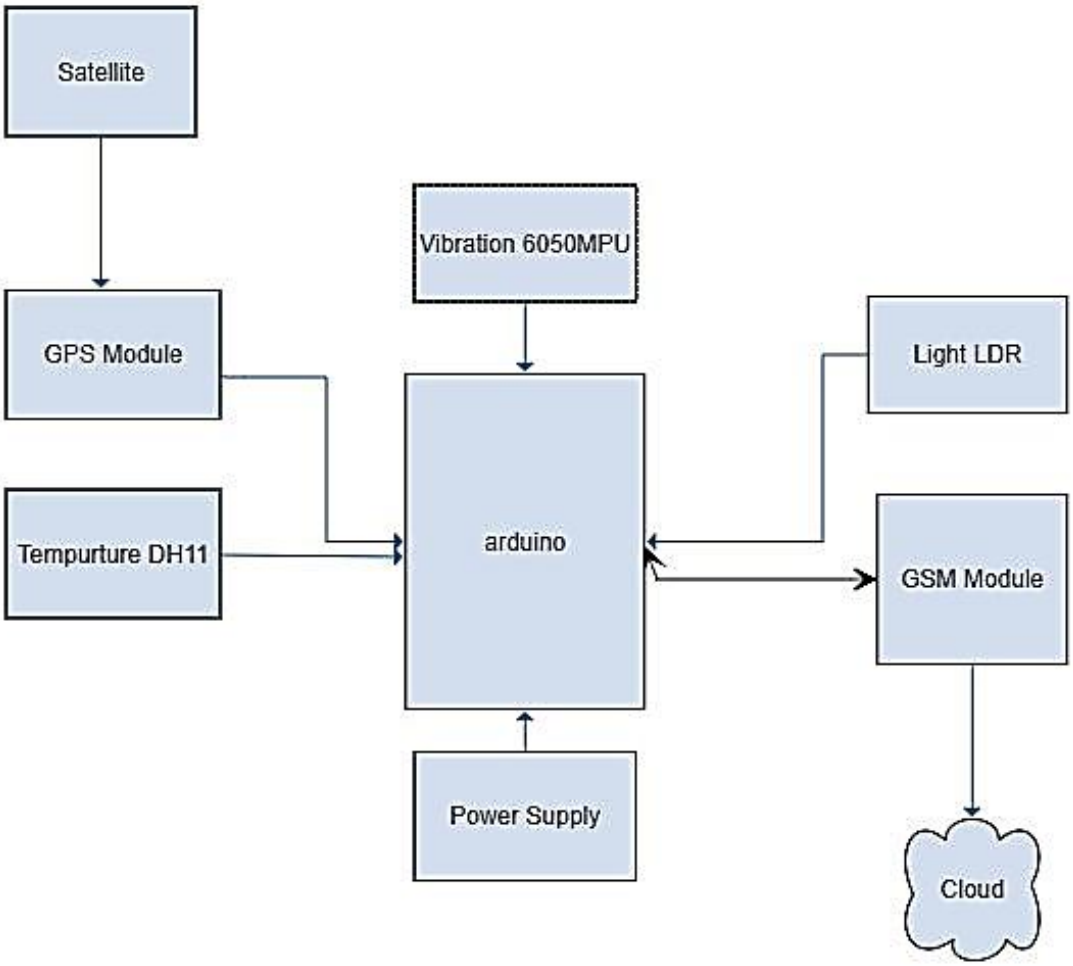


Figure 3.1 shows the block diagram of the hardware of our Project

3.2 Hardware component specification

By using Arduino Uno chip and Ethernet chip and many sensors like:
Vibration, Humidity, temperature sensors and translate output data to Json by Arduino code; then we will send data to cloud by Ethernet chip.

- Arduino Uno
- GPS sensor
- Ethernet chip
- Sensors
    - Vibration sensor
    - Humidity sensor
    - Temperature sensor

3.2.1 Arduino Uno

Figure 3.2 Arduino UNO

Arduino Uno is a microcontroller board based on the ATmega328P (datasheet). It has 14 digital input/output pins (of which 6 can be used as PWM outputs), 6 analog inputs, a 16 MHz quartz crystal, a USB connection, a power jack, an ICSP header and a reset button.

3.2.2 Temperature Sensor DH11

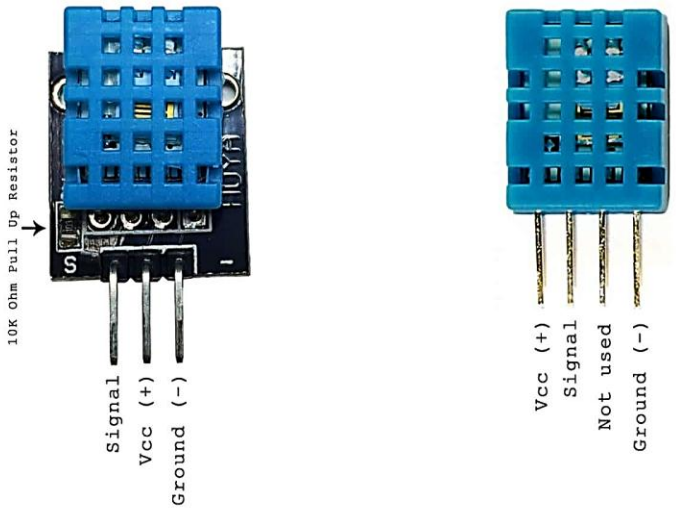


Figure 3.3 DH11 sensor

The DHT11 sensor can either be purchased as a sensor or as a module. Either way, the performance of the sensor is same. The sensor will come as a 4-pin package out of which only three pins will be used whereas the module will come with three pins as shown above.

The only difference between the sensor and module is that the module will have a filtering capacitor and pull-up resistor inbuilt, and for the sensor, you have to use them externally if required.

The DHT11 is a commonly used Temperature and humidity sensor. The sensor comes with a dedicated NTC to measure temperature and an 8-bit microcontroller to output the values of temperature and humidity as serial data. The sensor is also factory calibrated and hence easy to interface with other microcontrollers.

The sensor can measure temperature from 0°C to 50°C and humidity from 20% to 90% with an accuracy of ±1°C and ±1%. So if you are looking to measure in this range then this sensor might be the right choice for you.

Relative Humidity Sensor Features
1. Output Voltage is proportional to relative humidity & temperature.

2. Rugged design for long term use.
3. Compact size.
4. Very low power operation.
5. Precise measurement.
6. Wide supply voltage range (3.5 to 20VDC).
7. Weather resistant enclosure.
8. Compatible with Arduino.
9. No need to calibrate.

3.2.3 Vibration Sensor 6050MPU

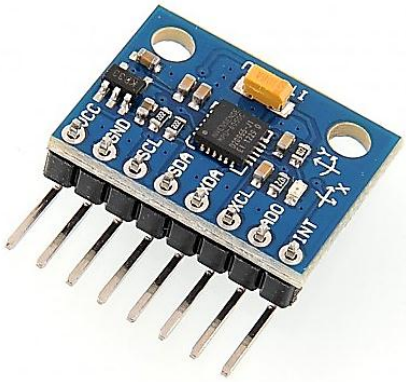


Figure 3.4 vibration sensor 6050MPU

The MPU-6050™ parts are the world's first Motion Tracking devices designed for the low power, low cost, and high-performance requirements of smartphones, tablets and wearable sensors.

The MPU-6050 devices combine a 3-axis gyroscope and a 3-axis accelerometer on the same silicon die, together with an onboard Digital Motion Processor™ (DMP™), which processes complex 6-axis MotionFusion algorithms. The device can access external magnetometers or other sensors through an auxiliary master I^2C bus, allowing the devices to gather a full set of sensor data without intervention from the system processor. The devices are offered in a 4 mm x 4 mm x 0.9 mm QFN package.

Features include:

- High Voltage Sensitivity (1 V/g)
- Over 5 V/g at Resonance
- Horizontal or Vertical Mounting
- Shielded Construction
- Solderable Pins, PCB Mounting
- Low Cost
- < 1% Linearity
- Up to 40 Hz (2,400 rpm) Operation Below Resonance

3.2.4 Light Sensor LDR

Figure 3.5 light sensor LDR

Light Sensors are photoelectric devices that convert light energy (photons) whether visible or infra-red light into an electrical (electrons) signal

A Light Sensor generates an output signal indicating the intensity of light by measuring the radiant energy that exists in a very narrow range of frequencies basically called “light”, and which ranges in frequency from “Infra-red” to “Visible” up to “Ultraviolet” light spectrum.

3.2.5 WIFI Module

Figure 3.6 WIFI module

The ESP8266 WiFi Module is a self-contained SOC with integrated TCP/IP protocol stack that can give any microcontroller access to your WiFi network. The ESP8266 is capable of either hosting an application or offloading all Wi-Fi networking functions from another application processor. Each ESP8266 module comes pre-programmed with an AT command set firmware, meaning, you can simply hook this up to your Arduino device and get about as much WiFi-ability as a WiFi Shield offers (and that's just out of the box)! The ESP8266 module is an extremely cost effective board with a huge, and ever growing, community.

3.2.6 GPS Module Board

Figure 3.7 GPS Module Board

Global Positioning System (GPS) makes use of signals sent by satellites in space and ground stations on Earth to accurately determine its position on Earth.

The NEO-6M GPS receiver module uses USART communication to communicate with microcontroller or PC terminal.

It receives information like latitude, longitude, altitude, UTC time, etc. from the satellites in the form of NMEA string. This string needs to be parsed to extract the information that we want to use.

There is many ways to connect Arduino data to cloud but the most common ways is by Arduino Ethernet Shield:

The Arduino Ethernet Shield 2 allows an Arduino board to connect to the internet using the Ethernet library and to read and write an SD card using the SD library. This shield is fully compatible with the former version, but relies on the newer W5500 chip.

3.3 Hardware integration

3.3.1 Interfacing ESP8266 WIFI module with Arduino

To communicate with the ESP8266 wifi module, microcontroller needs to use set of AT commands. Microcontroller communicates with ESP8266-01 wifi module using UART having specified Baud rate (Default 115200).

The ESP8266 module works with 3.3V only, anything more than 3.7V would kill the module hence be cautions with your circuits. The best way to program an **ESP-01** is by using the FTDI board that supports 3.3V programming. If you don't have one it is recommended to buy one or for time being you can also use an Arduino board. One commonly problem that everyone faces with ESP-01 is the powering up problem. The module is a bit power hungry while programming and hence you can power it with a 3.3V pin on Arduino or just use a potential divider. So it is important to make a small voltage regulator for 3.31v that could supply a minimum of 500mA. One recommended regulator is the LM317 which could handle the job easily. A simplified circuit diagram for using the ESP8266-01 module is given below:

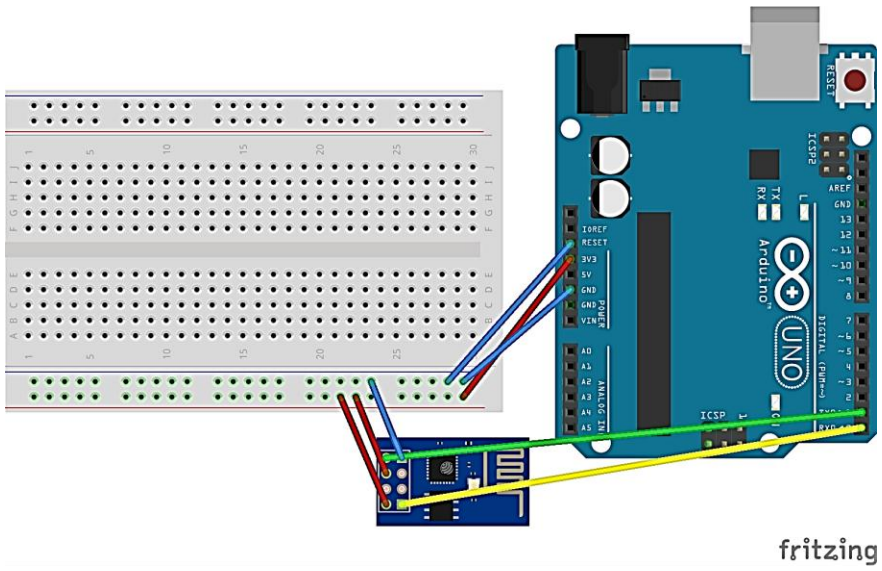


Figure 3.8 Interfacing ESP8266 WIFI module with Arduino

The switch SW2 (Programming Switch) should be held pressed to hold the GPIO-0 pin to ground. This way we can enter into the programming mode and upload the code. Once the code is released the switch can be released.

Connect your ESP8266 module as follows:

- Vcc = 3.3V (needs around 300-400mA peak)
- Gnd = -ve ground
- CH_PD = Chip enable so always +ve
- RST = Leave floating or ground to reset
- GPIO0 = Normally floating but this needs to be grounded when you start the update.
- GPIO2 = high level
- UTXD = Tx data connect to RX on FTDI/Serial interface
- URXD = Rx data connect to TX of FTDI/Serial interface

We can use any GPS for Arduino that supports the UART protocol. o we connect the TX of the GPS to the RX of the Arduino to get the serial data and display it.

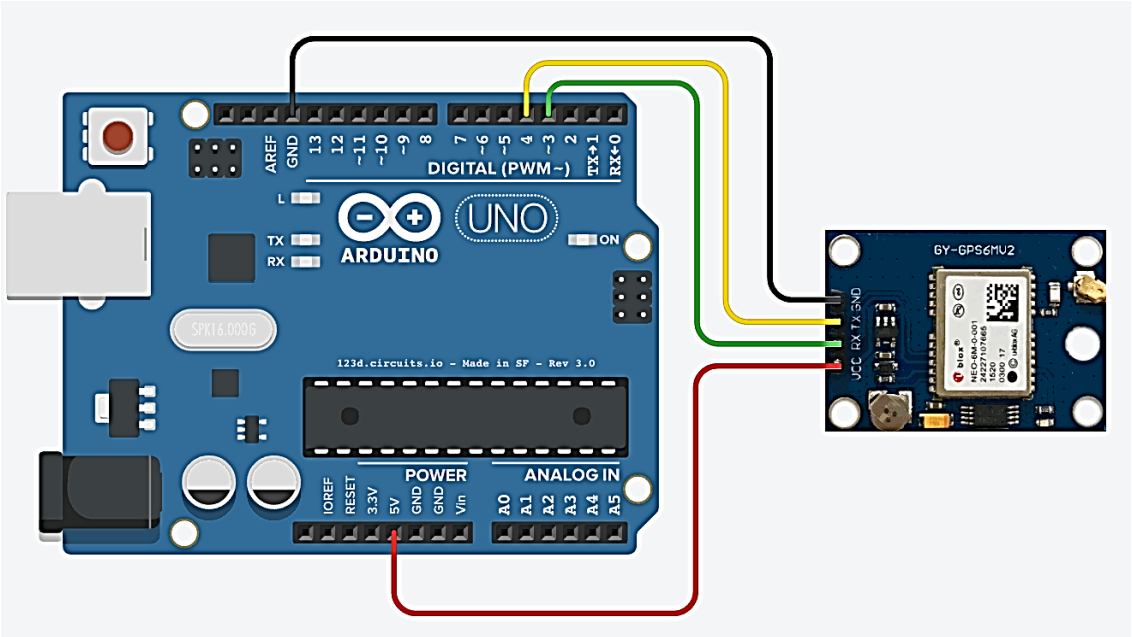


Figure 3.9 Interfacing GPS module with Arduino

The dedicated TX and RX pin are busy in communicating with the computer to download the code serially into the Arduino and to display the serial data on a serial monitor. So, to get the serial data from the GPS we used software serial of the Arduino. We connected the TX pin of GPS to the digital pin 4 as RX pin. And 3.3V supply to the GPS module. We have used the Device Example program from the Tiny GPS++ library that appears after adding the library to the Arduino ide. We run the Device Example Arduino program. But only this text appears on the serial monitor. No data like longitude, latitude, time or date.

Then we changed in the program the baud rate from the 4800 to 9600 and it worked. Because most of the GPS module working on default 9600 baud rate.

```
static const uint32_t GPSBaud = 4800;
```

```
static const uint32_t GPSBaud = 9600;
```

The 9600 baud rate for communication between GPS and Arduino. The 115200 baud rate Arduino uses to send the GPS data to the computer to display on the serial monitor.

```
Serial.begin(115200);
```

Now we get the expected result.

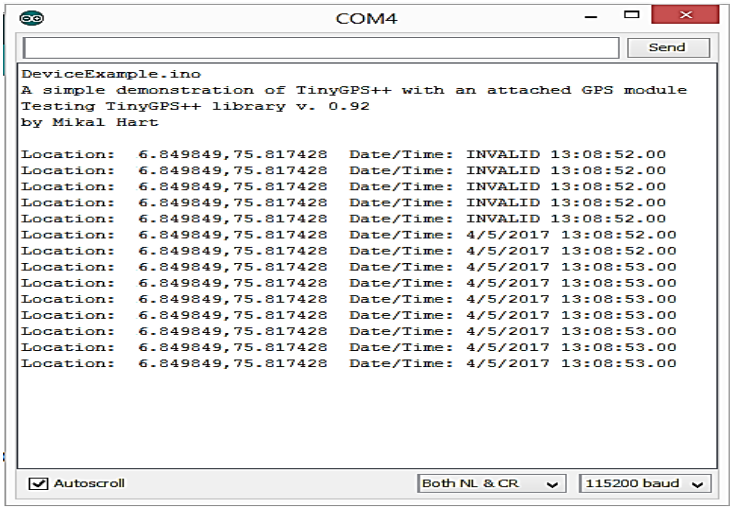


Figure 3.10 results from GPS

3.3.2 Interfacing the Arduino MPU 6050

The MPU 6050 communicates with the Arduino through the I2C protocol. The MPU 6050 is connected to Arduino as shown in the following diagram. If your MPU 6050

module has a 5V pin, then you can connect it to your Arduino's 5V pin. If not, you will have to connect it to the 3.3V pin. Next, the GND of the Arduino is connected to the GND of the MPU 6050.

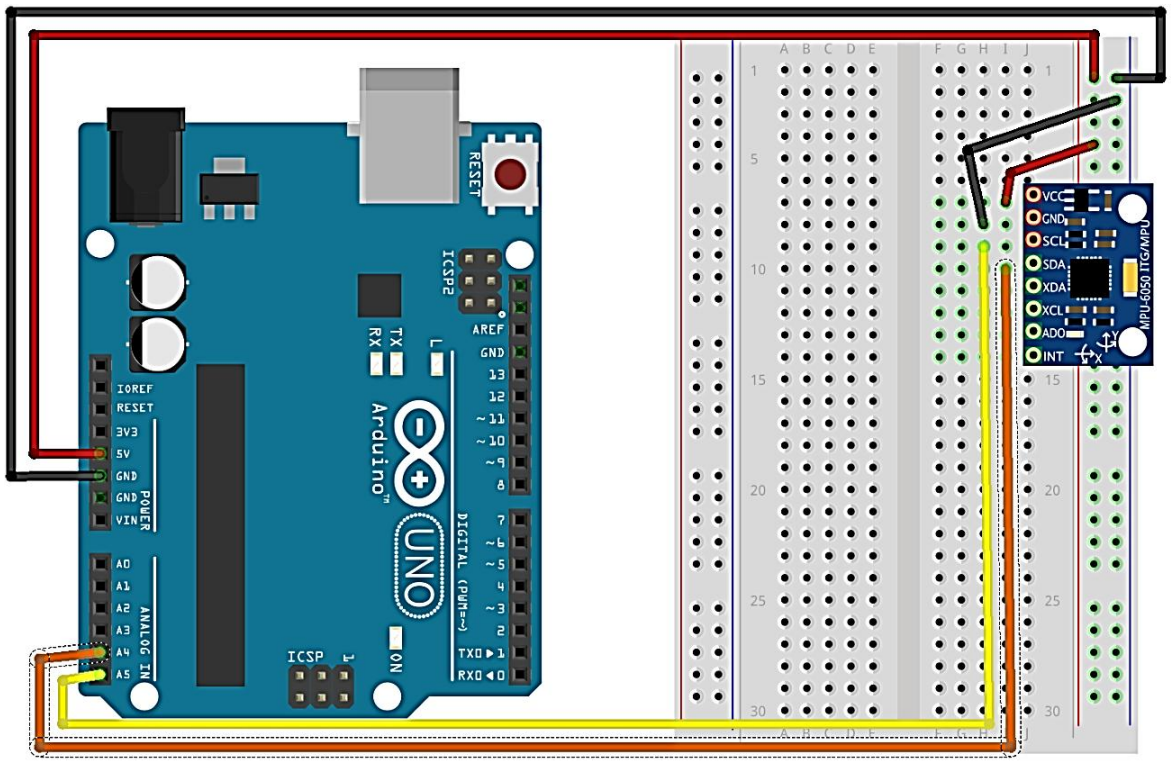


Figure 3.11 Arduino MPU 6050 connections

The program we will be running here also takes advantage of the Arduino's interrupt pin. Connect your Arduino's digital pin 2 (interrupt pin 0) to the pin labeled as INT on the MPU 6050.

Next, we need to set up the I2C lines. To do this, connect the pin labeled SDA on the MPU 6050 to the Arduino's analog pin 4 (SDA), and the pin labeled as SCL on the MPU 6050 to the Arduino's analog pin 5 (SCL). That's it! You have finished wiring up the Arduino MPU 6050.

3.3.3 Uploading the Code and Testing the Arduino MPU 6050

To test the Arduino MPU 6050, first download the Arduino library for MPU 6050, developed by Jeff Rowberg.

Next, you have to unzip/extract this library, take the folder named "MPU6050", and paste it inside the Arduino's "library" folder. To do this, go to the location where you have installed Arduino (Arduino --> libraries) and paste it inside the libraries folder. You might also have to do the same thing to install the I2Cdev library if you don't already have it for your Arduino.

If you have done this correctly, when you open the Arduino IDE, you can see "MPU6050" in File --> Examples. Next, open the example program from File --> Examples --> MPU6050 --> Examples --> MPU6050_DMP6 as shown in figure 3.12.

Next, you have to upload this code to your Arduino. After uploading the code, open up the serial monitor and set the baud rate as 115200. Next, check if you see something like "Initializing I2C devices ..." on the serial monitor. If you don't, just press the reset button. Now, you'll see a line saying, "Send any character to begin DMP programming and demo." Just type in any character on the serial monitor and send it, and you should start seeing the yaw, pitch, and roll values coming in from the MPU 6050.

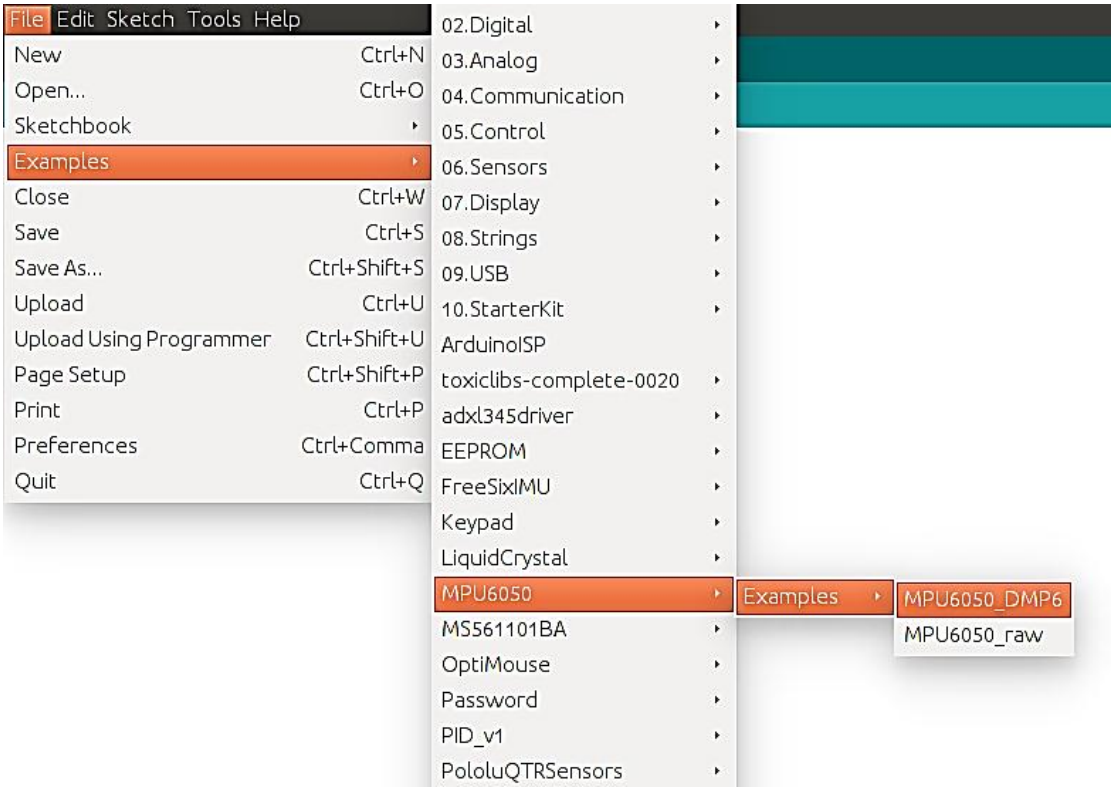


Figure 3.12 Arduino MPU 6050 DMP code

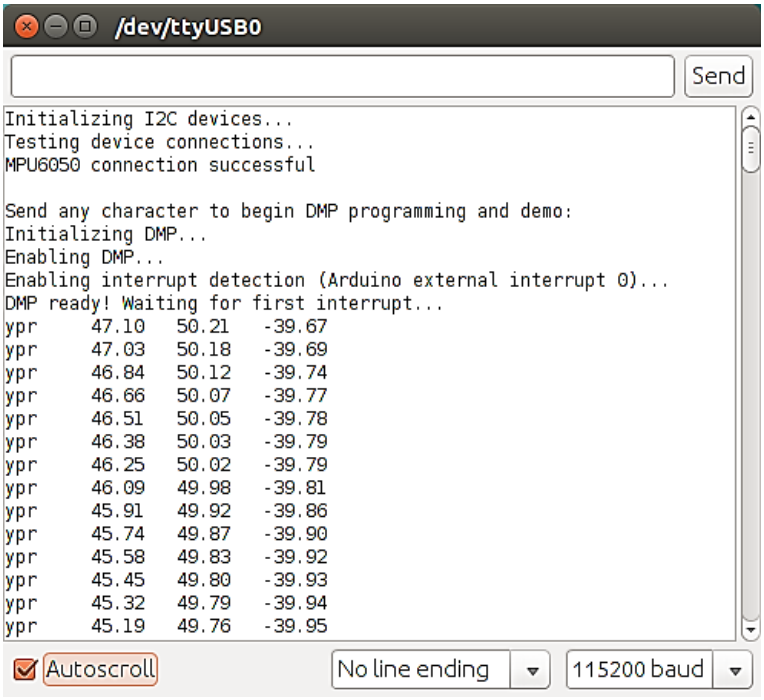


Figure 3.13 Arduino MPU 6050 Serial Monitor

DMP stands for Digital Motion Processing. The MPU 6050 has a built-in motion processor. It processes the values from the accelerometer and gyroscope to give us accurate 3D values.

3.3.4 Interfacing the DHT11 with Arduino

DHT11 is a humidity and temperature sensor. It can be used as humidity sensor as well as the temperature sensor. You can find the dht11 sensor of 2 types in the market. One is with 4 pins and another is with 3 pins. In 3 pin dht11 sensor already 10k Ohm resistor is added inside the module. The operating voltage of this module is **3.3V**

You need a 10K Ohm pull up resistor to get the proper data from 4 pin dht11 sensor. Pin configuration of this sensor, Left most pin is 3.3V (VCC), the second leftmost pin is output and the right most pin is ground. In the following code the I declared the pin to get reading from the sensor is Digital pin 4 . You need to add 10K Ohm pull up resistor between VCC and Output. You can find the connections in above circuit.

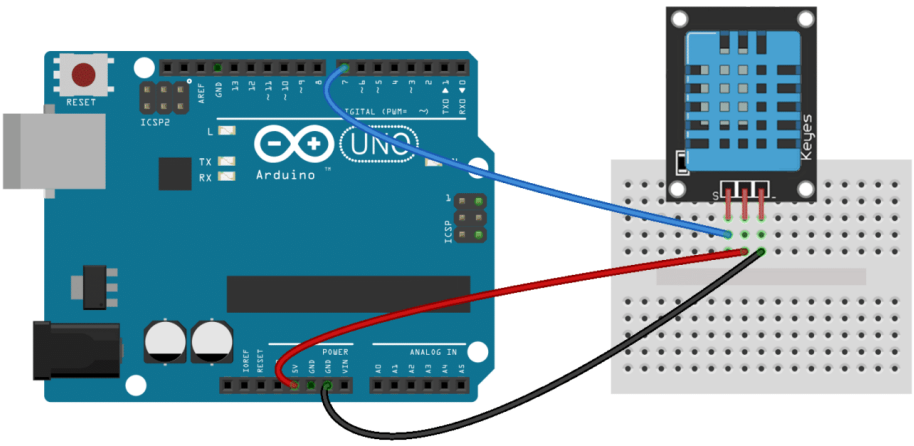


Figure 3.14 Interfacing the DHT11 with Arduino

Pin configuration of the dht11 sensor is, the leftmost pin is 3.3V(VCC), the middle pin is output, the rightmost pin is Ground. In the following code, I declared Digital pin 4 to get data from the sensor. So, we need to connect output pin of the dht11 sensor to the digital pin 4 of Arduino. If you have any confusion, you can find the connections in above circuit.

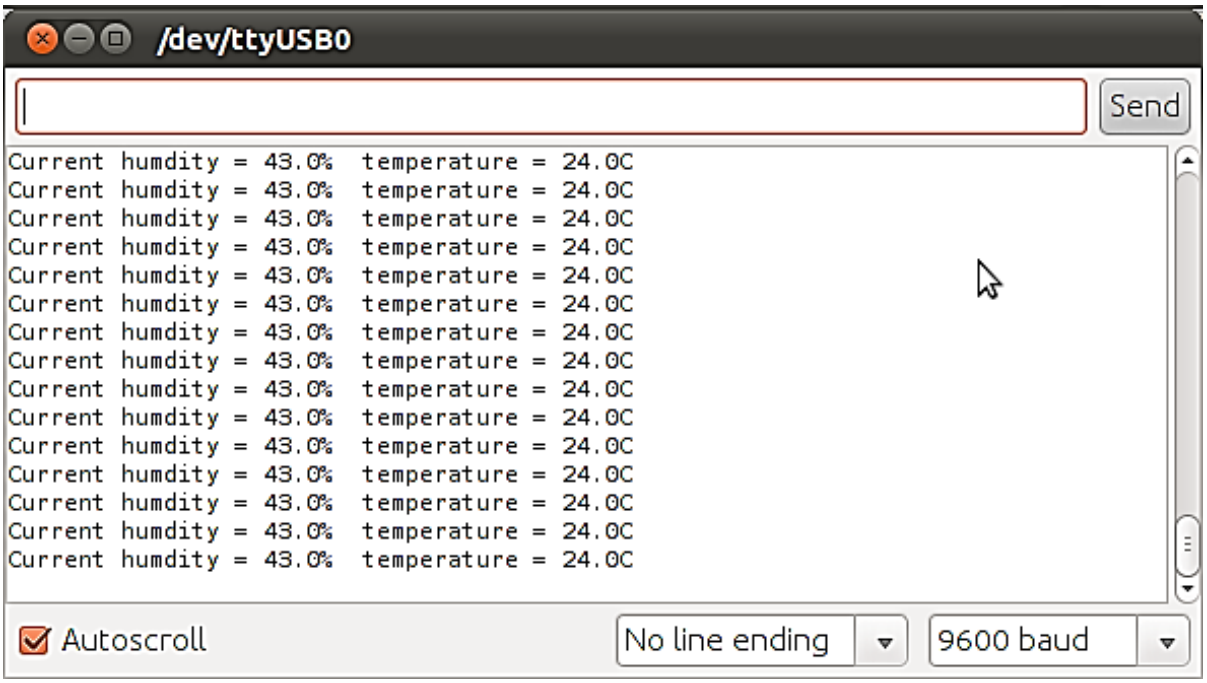


Figure 3.15 Temperature output on the serial monitor

You can see the output on the serial monitor. To open serial monitor go to Tools > Serial Monitor . Set the baud rate of serial monitor to 9600. The output is updated every second and can observe the changes in Temperature and Humidity.

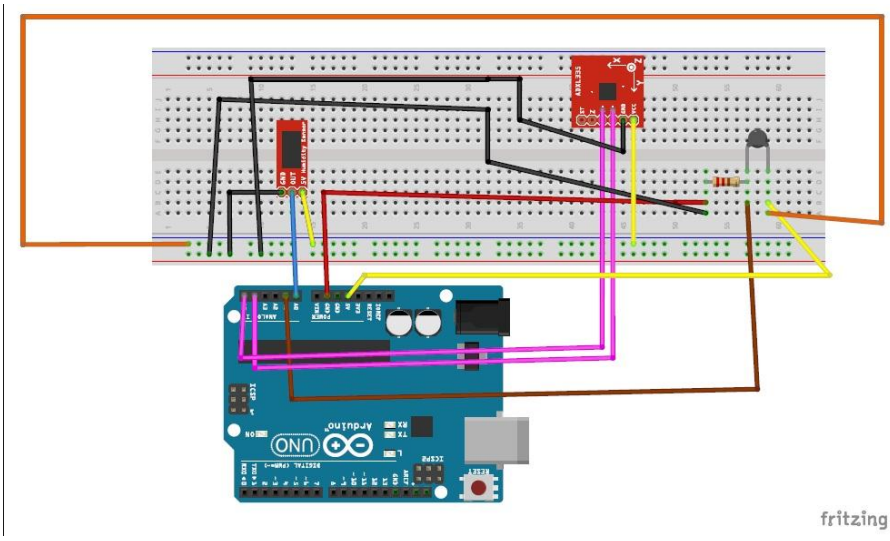


Figure 3.16 Final Arduino connection

3.4 Blockchain Build perishable Goods network

we will be using Hyperledger Composer to build our perishable goods network.

3.4.1 Hyperledger Composer Playground

Hyperledger Composer Playground is a browser-based interface that can use to model business network: what items of value (assets) are exchanged, who participates (participants) in their exchange, how access is secured (access control), what business logic (transactions) is involved in the process, and more.

Hyperledger Composer Playground (Playground from now on) uses browser's local storage to simulate the blockchain network's state storage, which means we don't need to run a real validating peer network to use Playground.

Playground runs in a Docker container, and can be installed to a computer in either of two modes:

- With a Hyperledger Fabric validating peer network
- In browser-only mode

Docker is a computer program that performs operating-system-level virtualization, also known as "containerization". Docker is used to running software packages called "containers"

Hyperledger Composer tutorial available here:
https://hyperledger.github.io/composer/latest/tutorials/tutorials.html

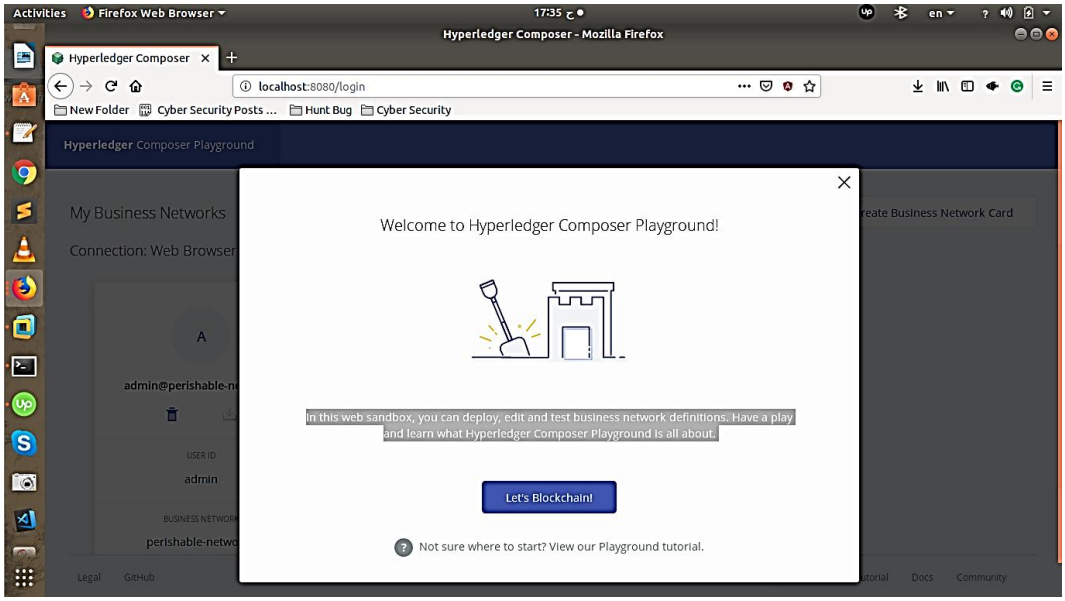


Figure 3.17 Show Hyperledger Composer Playground

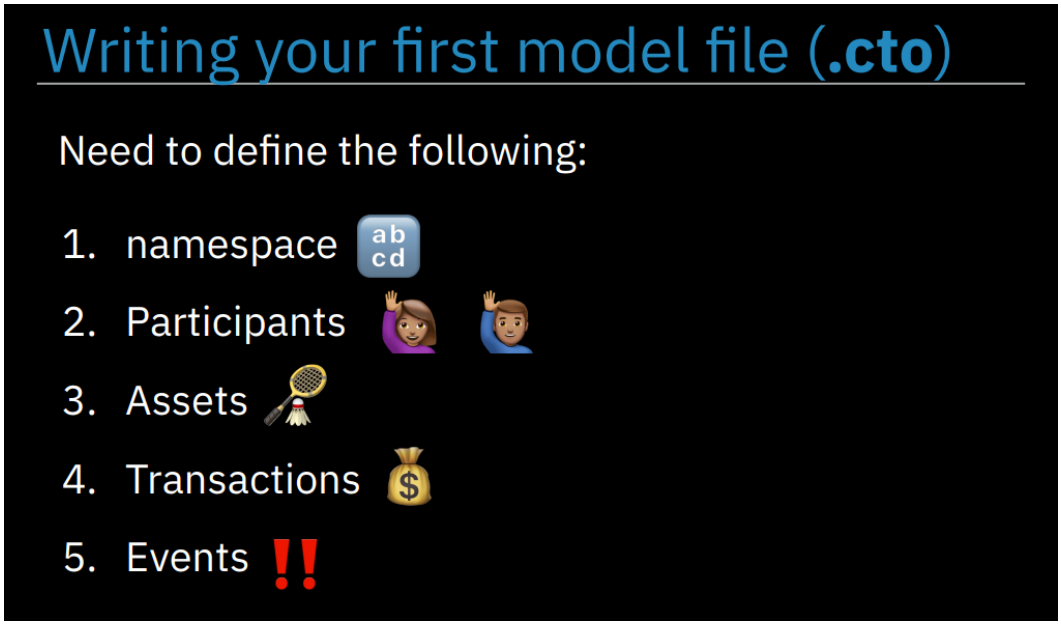


Figure 3.18 What to need to write model file .cto

3.4.2 Participants

A participant is a member of a business network. For the Perishable Goods network, this includes the growers who produce the perishable goods, the shippers who transport them from the growers to the ports, and the importers who take delivery of the goods at the ports.

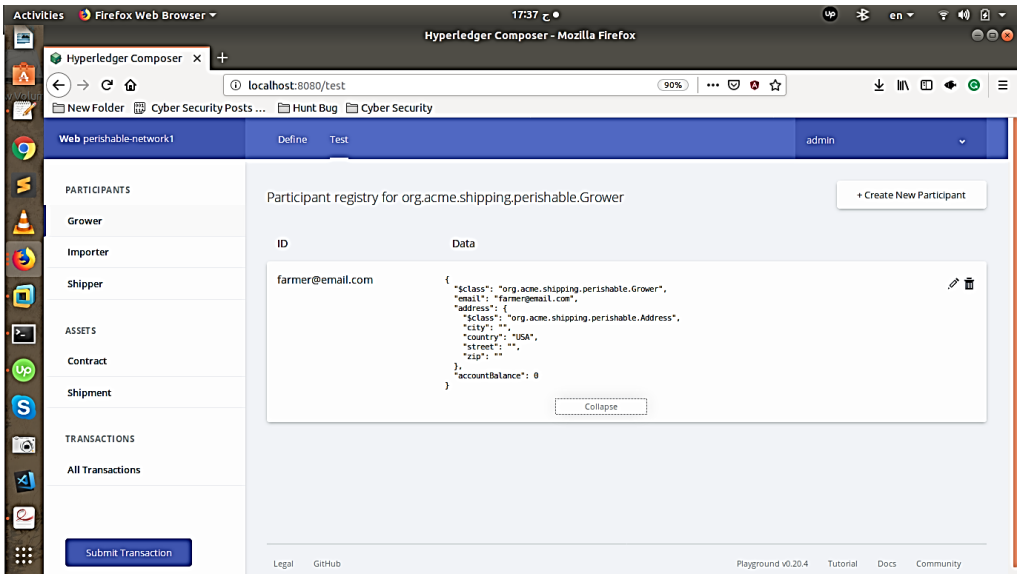


Figure 3.19 shows participate in composer playground

3.4.3 Assets

An asset is anything of value that can be exchanged between parties in a business agreement. That means an asset can be pretty much anything. Examples include:

A shipment of bananas

A contract for shipment of 1000 crates of bananas for X price based on conditions {X, Y, Z}

You name it. If it has perceived value and can be exchanged between parties, it's an asset. In the Perishable Goods network, assets include the perishable goods themselves, shipments of those goods, and the contracts that govern the activities performed during the exchange.

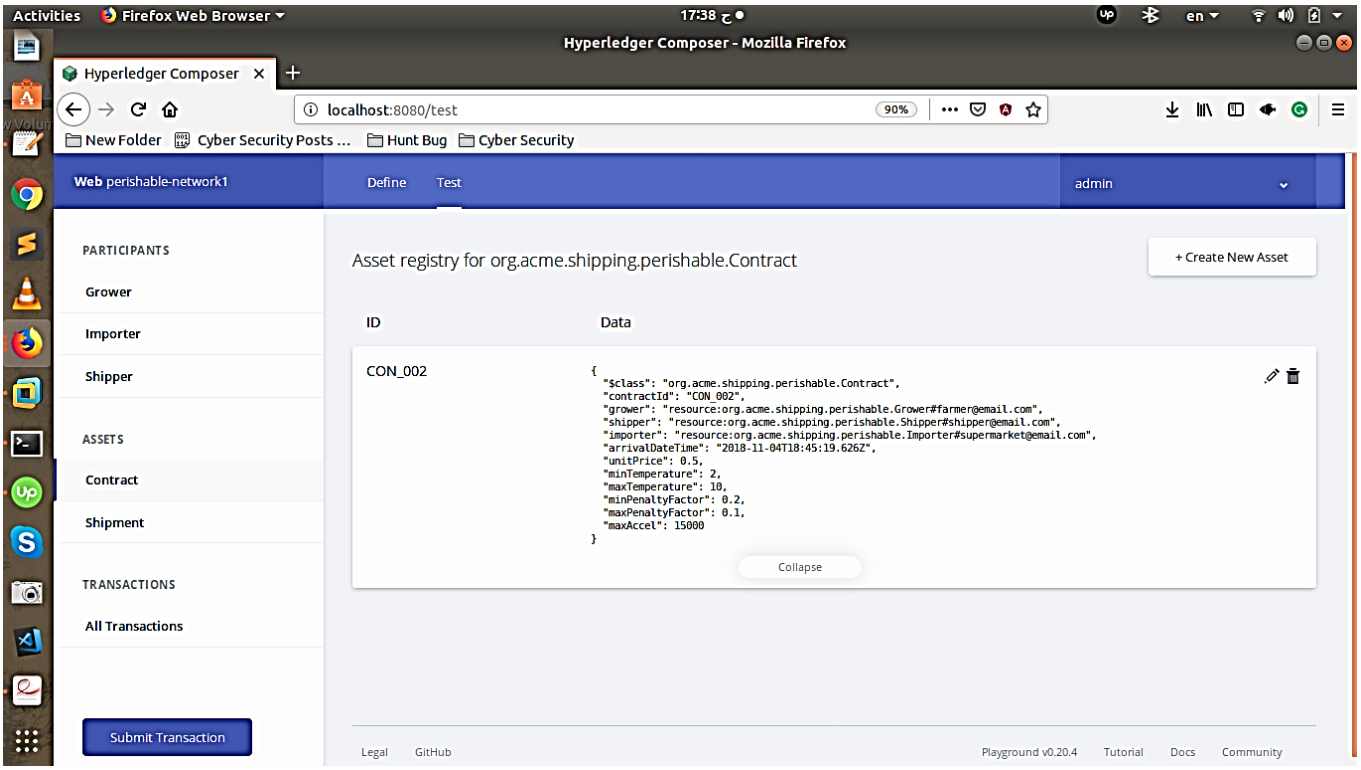


Figure 3.20 shows assets in composer playground

3.4.4 Transactions

When an asset is "touched," that interaction potentially affects the state of the blockchain ledger. The interaction is modeled in Hyperledger Composer as a transaction.

Examples of transactions in the Perishable Goods network include:

- An IoT sensor in the shipping container records a temperature reading
- An Importer receives a shipment of perishable goods
- An IoT GPS sensor records the shipping container's current location
- Transactions are the business logic (or smart contracts) of the system.

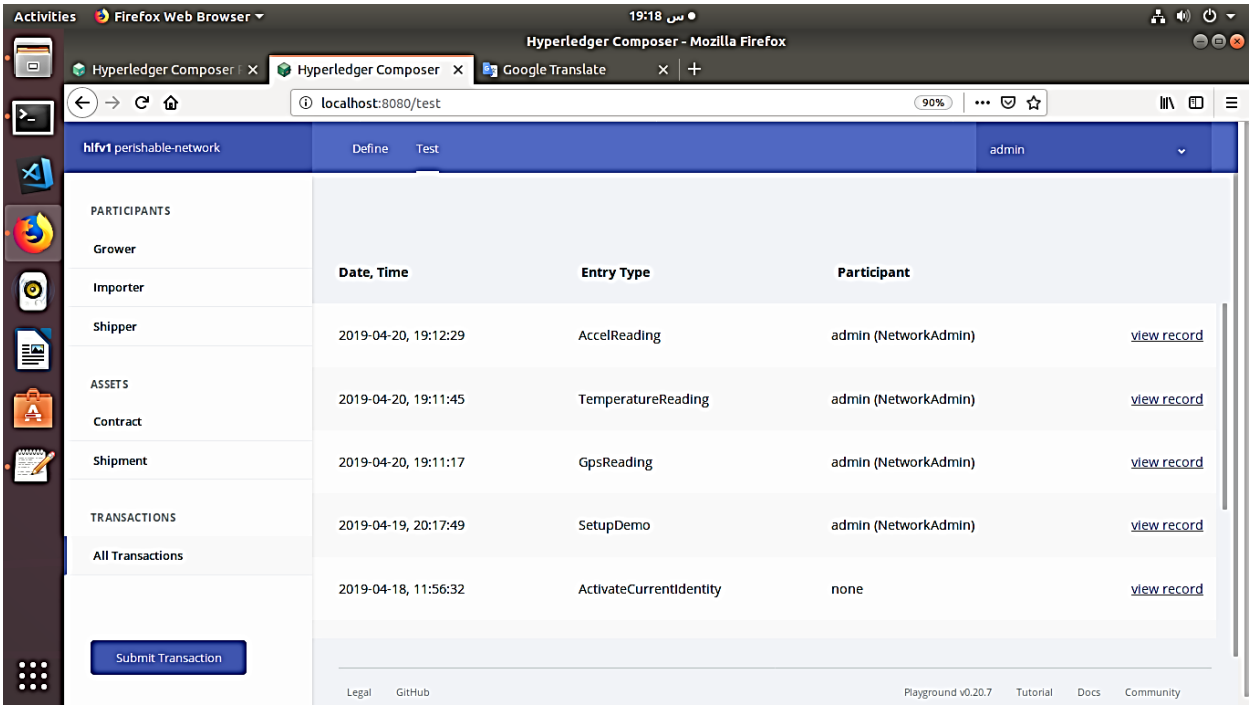


Figure 3.21 show all transaction in composer playground

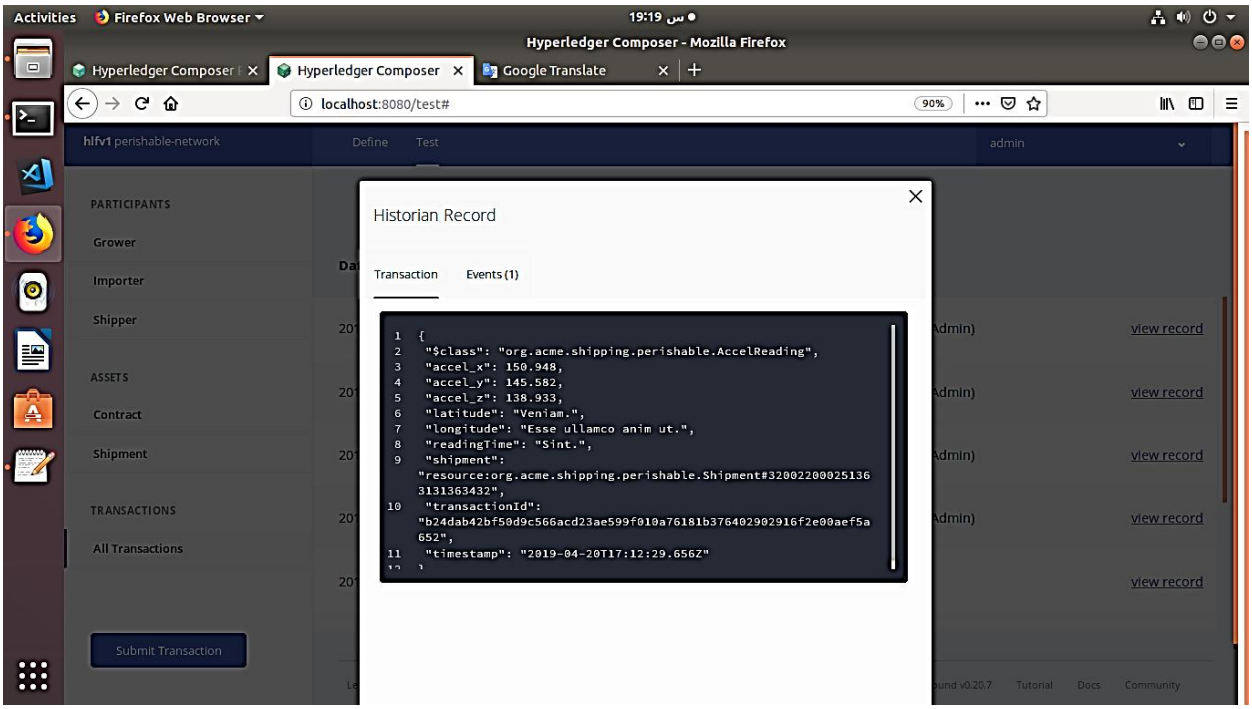


Figure 3.22 show transaction in composer playground

3.4.5 Events

An event is a notification produced by the blockchain application and consumed by an external entity (such as an application) in publish/subscribe fashion.

While the blockchain ledger is updated as assets are exchanged in the system, this is an internal (system) process. However, there are times when an external entity needs to be notified that the state of the ledger has changed, or that maybe something else of note has occurred (or not occurred but should have) in the system. blockchain application could use an event in this case.

Examples of events in the Perishable Goods network might include:

A temperature reading has exceeded an upper or lower boundary X number of times (this might indicate a problem with the shipping container itself, for example).

A shipment has been received.

A shipment has arrived at the port (an IoT GPS sensor could report this event, for example).

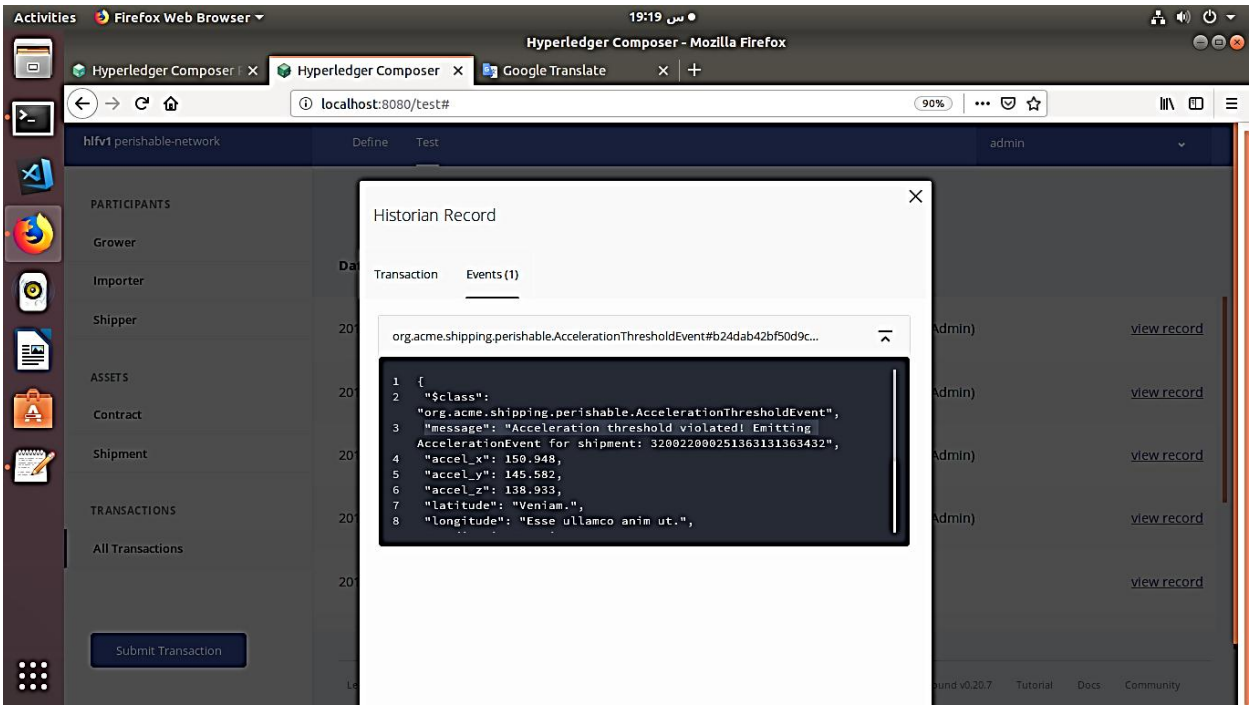


Figure 3.23 show event in composer playground

3.4.6 Business network concepts

At a high level, a business network is a group of entities who work together to accomplish certain goals. In order to achieve these goals, there must be agreement among the members of the business network regarding:

The goods and services that are exchanged.

How the exchange is to take place (including business rules governing payment and penalties). Which members of the group are allowed to participate, and when.

what is the business problem that blockchain business network will solve?

blockchain business network will solve: shipment of perishable goods.

3.4.7 The Perishable Goods network

The Internet of Things (IoT)-based Perishable Goods network is a business network involving:

- Perishable items such as bananas, pears, and coffee

- Business partners such as growers, shippers, and importers
- Shipments of perishable goods
- Agreements between business parties that stipulate conditions of the agreements
- Acknowledgment of receipt of goods and services

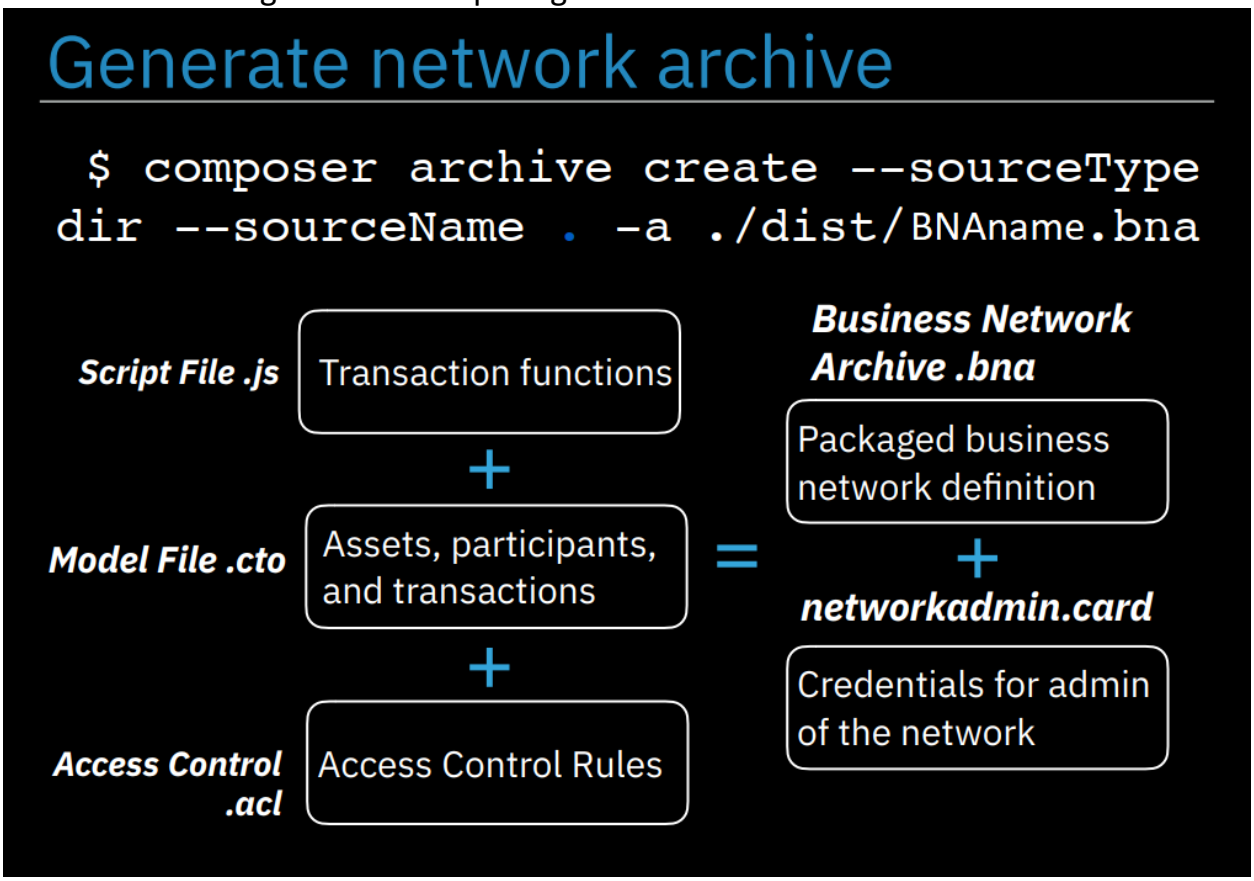


Figure 3. 24 What to need to create Business Network Archive (.bna)

3.4.8 Access control

Access control (.acl) In a business network, not every participant has access to everything. For example, a grower does not have access to the contract between the shipper and the importer. Access control is used to limit who has access to what (and under what conditions).

3.4.9 Script File

Script file (.js)) In a business network, a transaction was defined, specifying a relationship to an asset, and a participant. The transaction processor function file contains the JavaScript logic to execute the transactions defined In a business network

After we created a perishable goods network named perishable-network.bna to the local Hyperledger Fabric, but how see what's there? How interact with it? Hyperledger Composer provides a tool called composer-rest-server that generates a Loopback-based REST API interface to access network

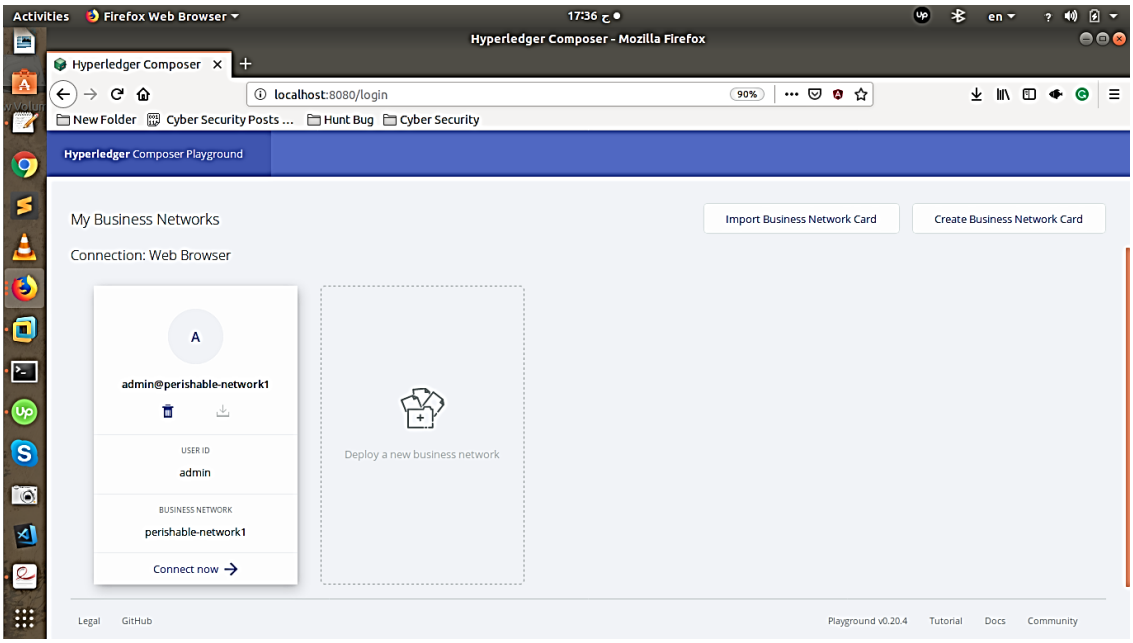


Figure 3.25 open and interact with business network from composer playground

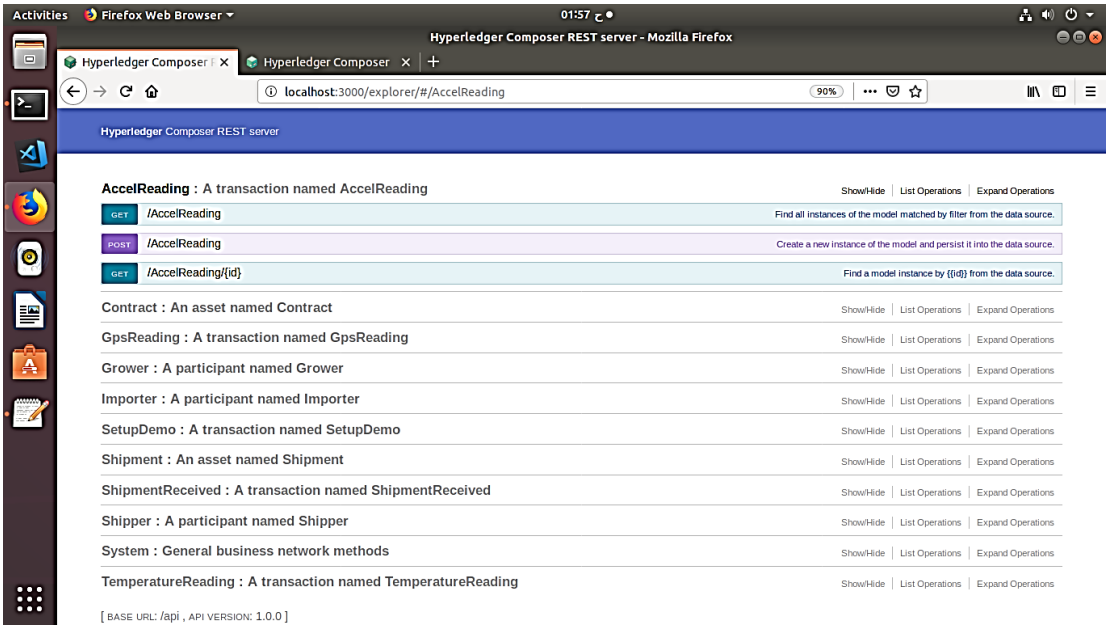


Figure 3.26 interact with business network from composer-rest-server API

After completing build the perishable Goods network section. Proceed to the Node-RED section which will leverage the REST API just enabled to write/read/visualize IoT Asset environmental sensor data to the transaction history.

3.5 Node-RED

Node-RED is a programming tool for wiring together hardware devices, APIs and online services in new and interesting ways, it provides a browser based editor that makes it easy to wire together flows using the wide range of nodes in the palette that can be deployed to its runtime in a single-click.
It can be found in [9].

3.5.1 Running Node-RED on a localhost:

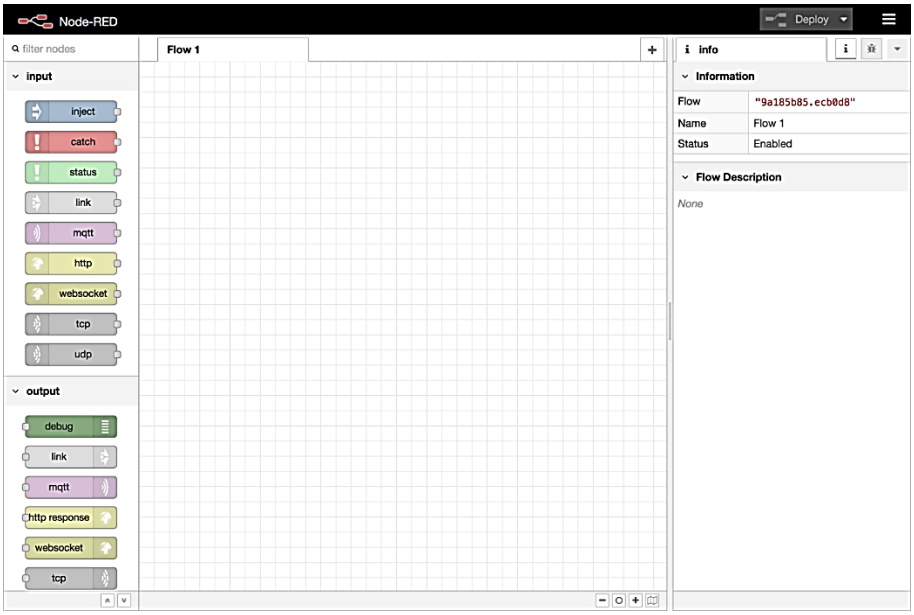


Figure 3.27 running node red

3.5.2 Usage of Node-Red in Hardware:

- Interacting with Arduino: Several ways to interact with Arduino is connected to host computer via USB serial connection and wifi.
- It can also work with (Raspberry Pi, BeagleBone Black, and Android).

3.5.3 Benefits of Node-Red:

- No need to start from scratch of coding, no need to worry about lines of code Node-Red does the codings and most importantly no need of deep knowledge background of coding.
- Saves time to get things done without wasting time to code.

3.5.4 Node-RED - IoT Asset Tracker:

These Node-RED program flows implement an IoT Asset Tracker that receives environmental sensor data from a hardware device and stores that information in a "Hyperledger Fabric blockchain".

3.5.5 The Transaction From The Node-Red to the Blockchain:

Node-Red-Contrib-blockchain: An easy to use mode for stamping data on the blockchain, which provides incontrovertible proof of existence.

This node uses Tiereion.com API to stamp incoming data onto the blockchain direct from Node-Red.

- **Key Features:**
- Drag and drop on node-red
- Fully configurable
- Produce irrefutable proof and a verifiable, immutable record.

3.5.6 Working with The Hyperledger From Node-Red

You can use the Hyperledger Composer Node-RED contribution to:

- Submit transactions
- Read and update assets and participants
- Subscribe to events
- Delete assets and participants

3.5.7 Hyperledger Composer Node-RED Nodes

Hyperledger-Composer-out: Output node that allows you to create, update or delete assets or participants and submit transactions. Ex: combining the Hyperledger-composer-out node with an inject nodes allows you to creating participants by submitting JSON.

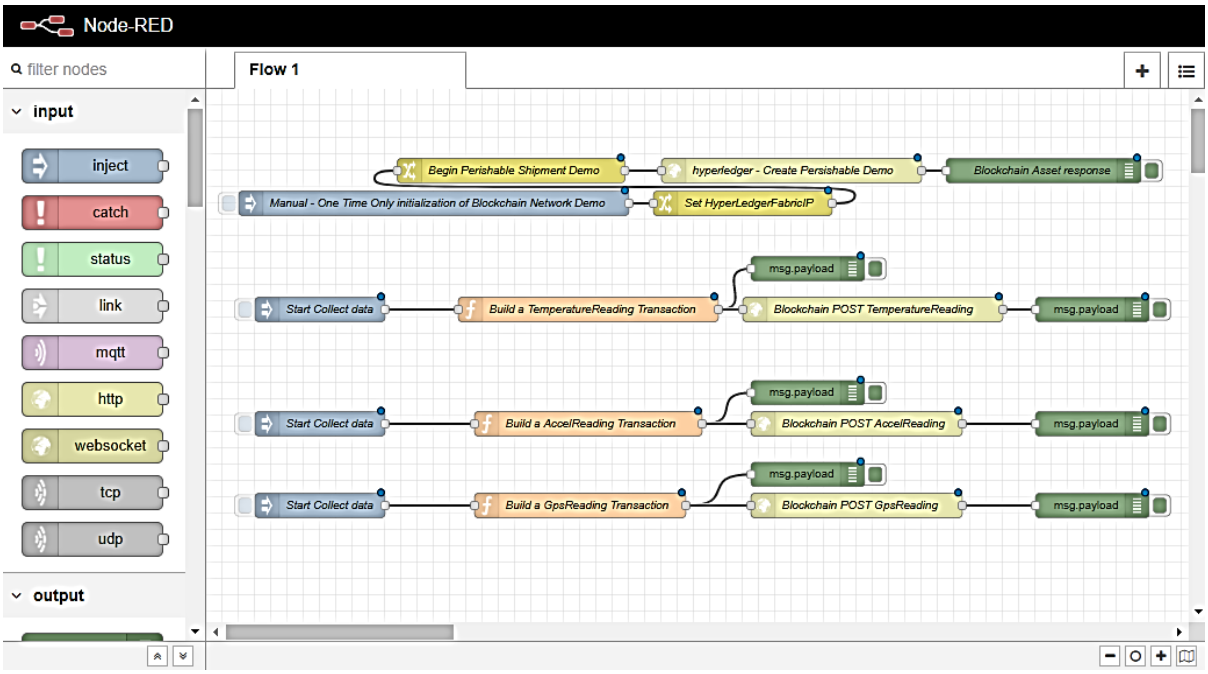


Figure 3.28 Node-RED program flows receives sensor data from an arduino, stores that information in a Hyperledger Fabric blockchain.

These flows perform the following functions:

- Initialize Blockchain and Node-RED variables - Set some globals that drive all of the flows.

- Receive Hardware events from Arduino.

  1- Build a TemperatureReading Transaction flow to display Temperature Sensor Data.

  2- Build a AccelReading Transaction flow to display the acceleration of the Data.

  3- Build a GpsReading Transaction flow to display GPS for details see [7,8].

- Write Arduino data to a Hyperledger Fabric Blockchain.

- Load Blockchain Transaction History.

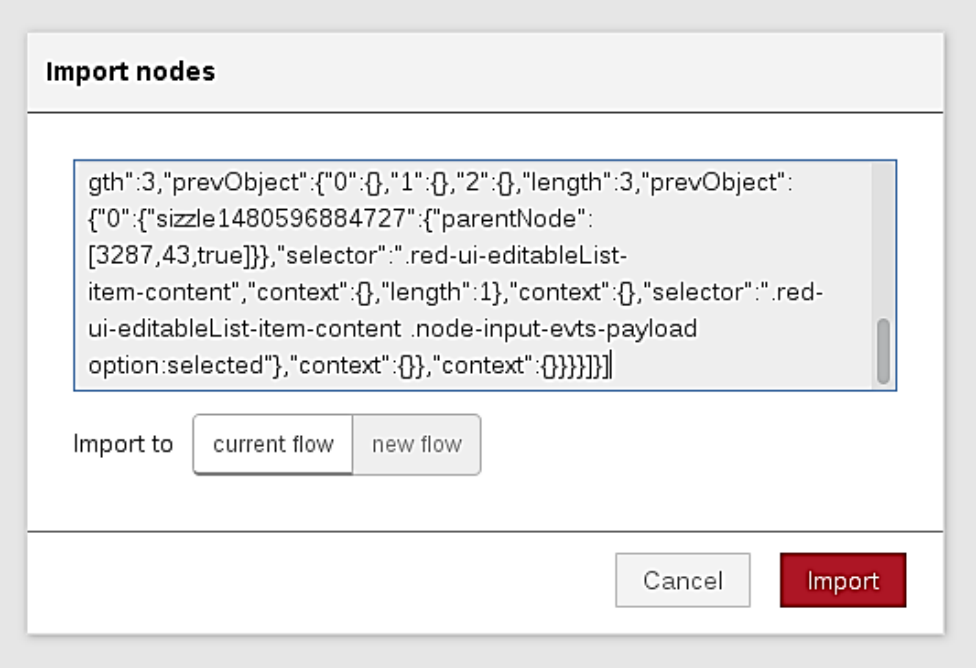


Figure 3.29 Writing Text of the flows to the Clipboard

3.5.8 Node-Red in our Application

- Operate Node-Red with our application write / read / visualize hardware devices by collecting data.
- Takes the arriving data, reformats it and calls the next flow to write the Events to the Hyperledger Perishable Network blockchain transaction

history.

CHAPTER 4

USER APPLICATION AND INTERFACE

4.1 registration page

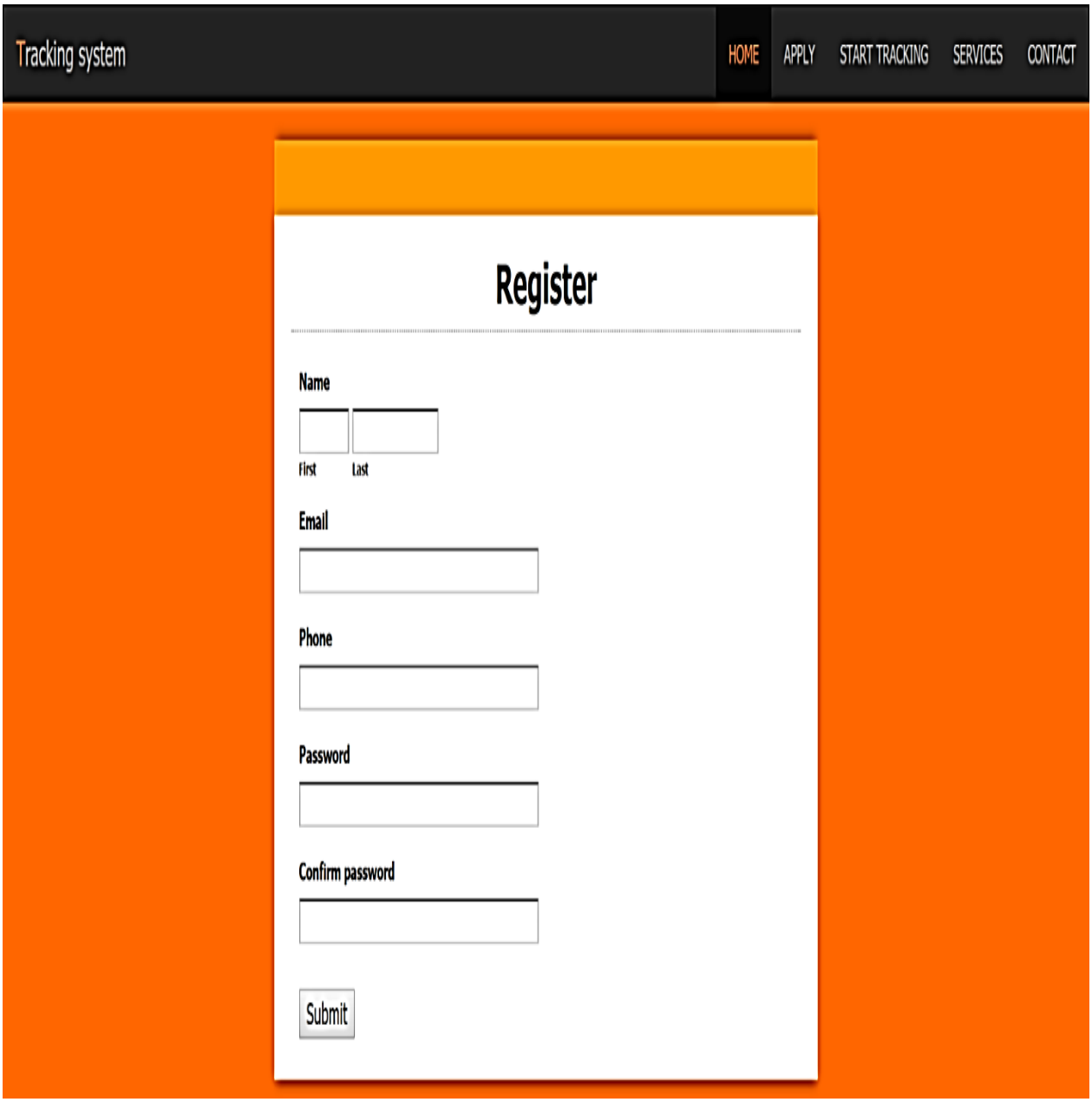


Figure 4.1 registration page

4.2 application home page

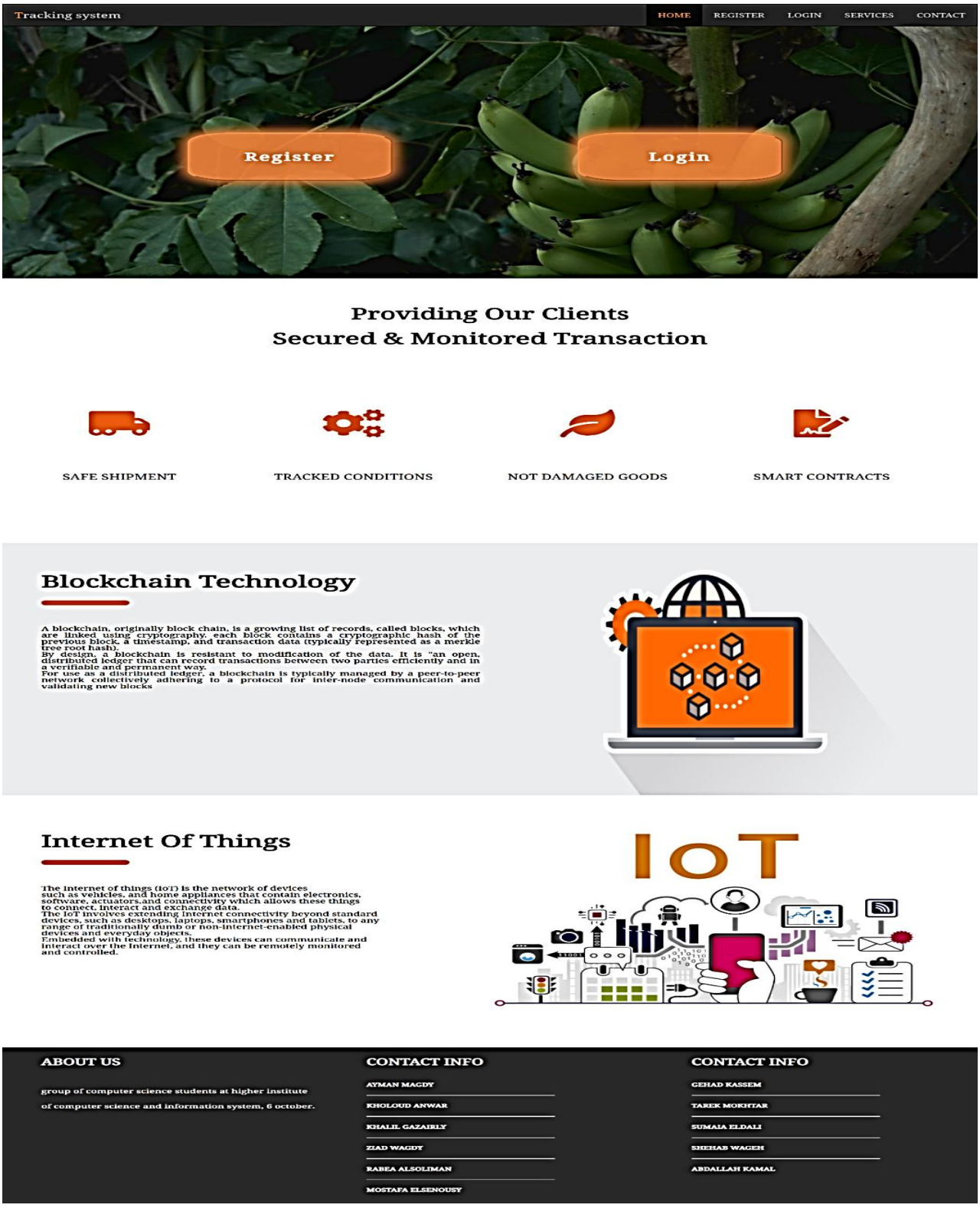


Figure 4.2 Application Home page

4.3 shipment details page

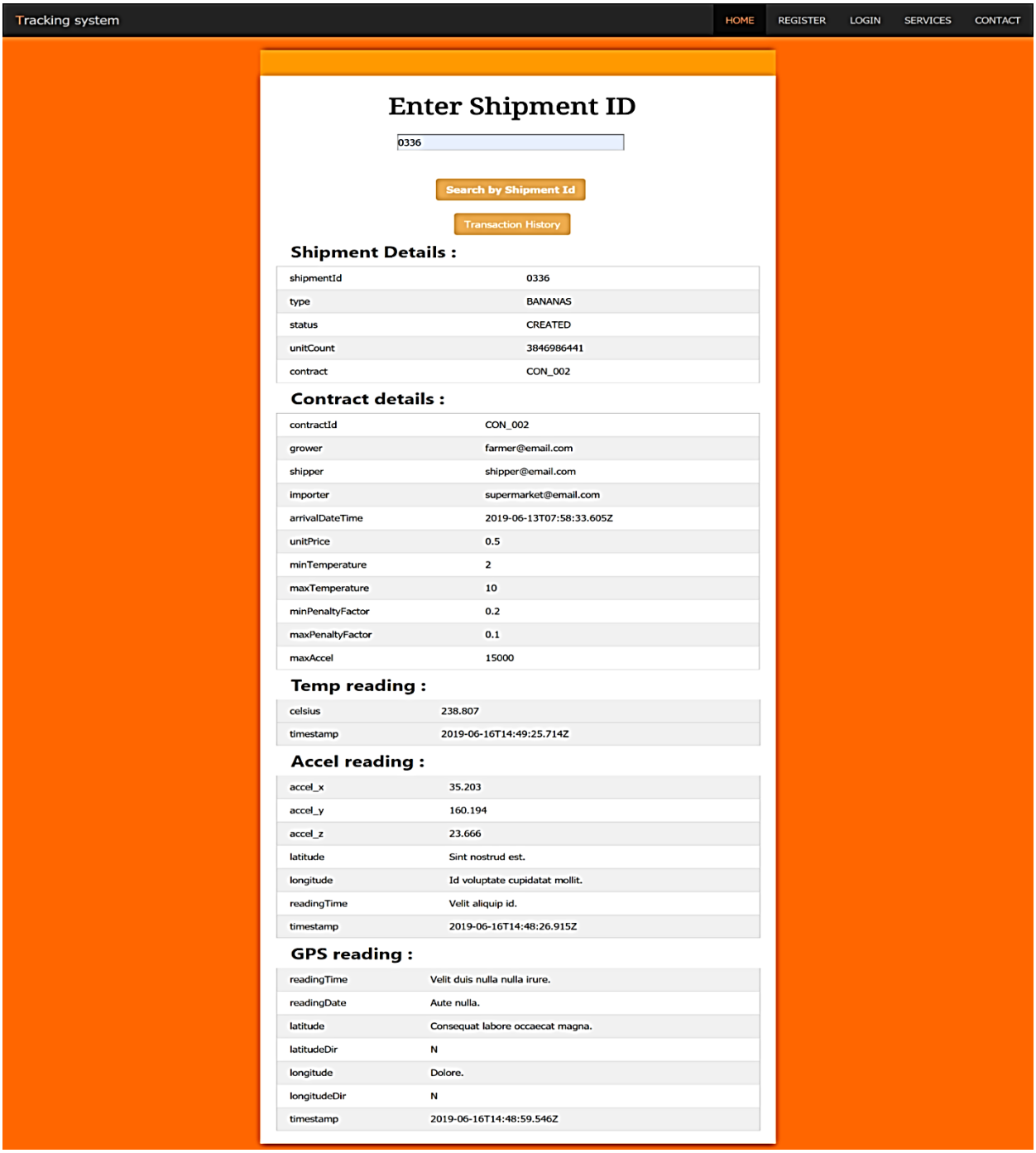


Figure 4.3 Shipment Details page

4.4 transaction history page

Figure 4.4 Transaction History page

4.5 simulated results in charts page

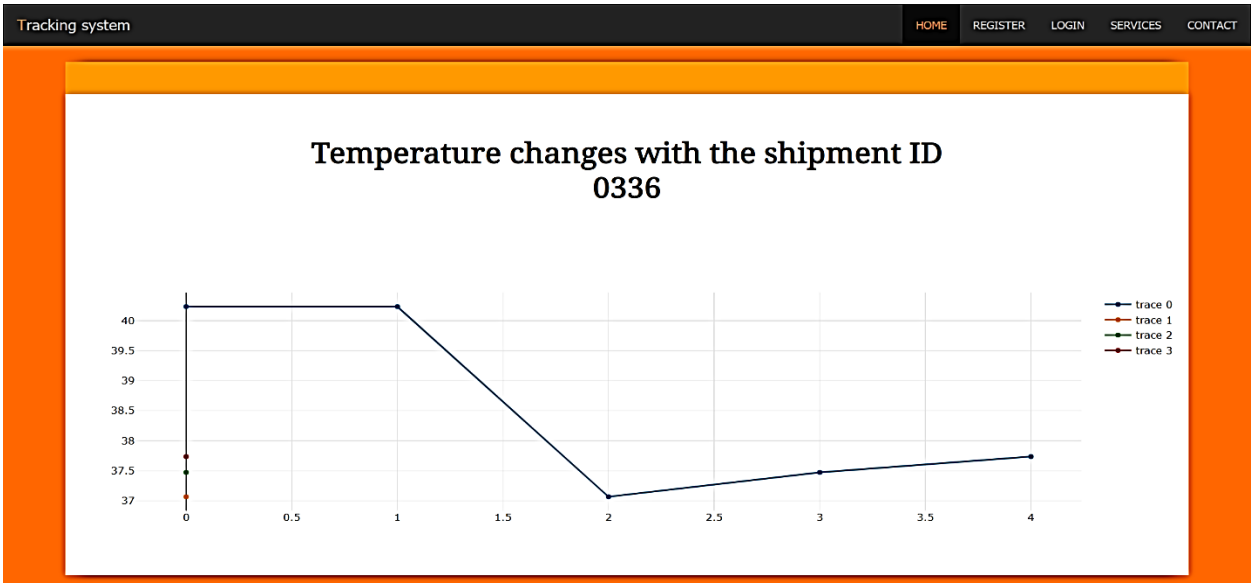


Figure 4.5 Simulated Results page

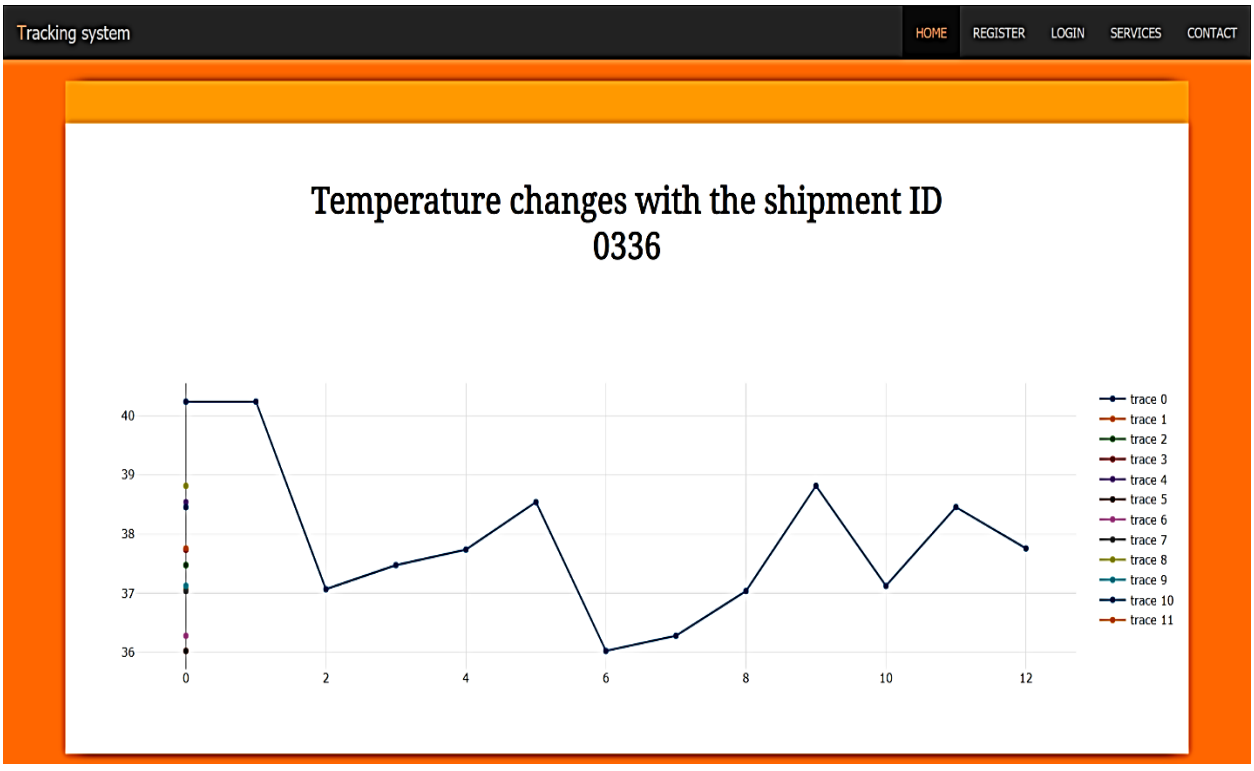


Figure 4.6 Simulated Results page

CHAPTER 5
CONCLUSION
AND FUTURE WORK

Conclusion

In our IOT assets tracking project we concerned with shipment and goods that have to be shipped with predefined constraints like temperature or acceleration. These goods may be food or medicine or any delicates that owner want to ship it without any damage.

In our project we succeeded to integrate between three new technologies which are IOT, Cloud computing and Blockchain.

As result from this integration we achieved a new way for tracking our shipment without human interaction by using IOT sensors and microcontrollers. Also, we have made our tracking process immutable and transparent and have real time readings by using Blockchain.

By using Smart contract which is one of blockchain tools that has terms and rules that once it has been verified the transaction will not be changed anymore.

We integrated the IOT with blockchain by using cloud computing and the node red tool.

Finally, we have presented our results on our website to give access to users to track his shipment and we displayed these readings and information in histograms.

Future work

In our project we faced some limitations and problems that we would in the future deal with.

One of these limitations was the GPS module that was very difficult to be obtained due to some security issues. To deal with this limitation we had to simulate location on map. So, we will work on that issue in the future.

Another limitation was testing our system on real world that has not been done yet. We have made a small model and tested it in small range. In the future this model will be implemented and tested on real shipment.

Also, our user interface which is the website that is globally opened to any user will have more features like building huge database for registration for users to be able to get data and roam in our web site freely.

Finally, we are looking to implement this project and broadcast it in labor market to get financial gain.

References

[1] https://www.amarinfotech.com/iot-logistics-assets-solutions.html.

[2] Shailendra Singh, Abdulsalam Yassine **"**IoT Big Data Analytics with Fog Computing for Household Energy Management in Smart Grids",2018.

[3] L. Wu, X. Yue, A. Jin, D.C. Yen, Smart supply chain management: A review and implications for future research, Int. J. Logist. Manage. 27 (2016) 395–417.

[4] D.F. Ross, Introduction to e-Supply Chain Management: Engaging Technology to Build Market-Winning Business Partnerships, CRC Press, 2016.

[5] ioannis Karamitsos, Maria papadaki, nedaa al barghuthi "design of the blockchain smart contract: A use case for real Estate, journal ofinformation security, January 2018.

[6] matthew N.O sadiku, Kelechi Eze, Sarhan M. Musa "Smart contract A primer", 2018.

[7] www.noderedguide.com.

[8] www.nodered.org.

[9] https://github.com/node-red.

[10] https://www.w3schools.com/htm

[11] https://www.w3schools.com/css/default.asp.

[12] https://www.w3schools.com/js/default.asp.

[13] https://www.w3schools.com/php/default.asp.

[14] https://www.codecademy.com/learn/learn-php.

APPENDICES

Appendix A

Node red connections

}

,"id": "811a05a4.4d5828"

,"type": "inject"

,"z": "ac3b7d6e.3a52"

,"name": "Manual - One Time Only initialization of Blockchain Network Demo"

,"" :"topic"

,"payload": "true"

,"payloadType": "bool"

,"" :"repeat"

,"" :"crontab"

,once": false"

,onceDelay": 0.1"

,x": 290"

,y": 140"

"34d4030.592747e"]] :"wires"

} ,{ [[

,"id": "34d4030.592747e"

,"type": "change"

,"z": "ac3b7d6e.3a52"

,"name": "Set HyperLedgerFabricIP"

] :"rules"

,"t": "set" }

,"p": "HyperLedgerFabricIP"

,"pt": "global"

,"<to": "<your Hyperledger Fabric Public IP"

,[{ "tot": "str"

```
,"" :"to"       ,"" :"from"       ,"" :"property"       ,"" :"action"

]          ] :"wires"       ,y": 140"       ,x": 670"       ,reg": false"

,{   [       [          "240a5098.d33d18"

}

,"id": "240a5098.d33d18"

,"type": "change"

,"z": "5e11e736.5672e"

,"name": "Begin Perishable Shipment Demo"

] :"rules"

}

,"pt": "msg"             ,"p": "payload"             ,"t": "set"

,"to": "{\"$class\":\"org.acme.shipping.perishable.SetupDemo\"}"

"tot": "json"

,{

}

,"to": "HyperLedgerFabricIP"             ,"pt": "msg"             ,"p": "FabricIP"             ,"t": "set"
,[       {          "tot": "global

,x": 385"       ,reg": false"       ,"" :"to"       ,"" :"from"       ,"" :"property"       ,"" :"action"

[          "5453caaf.c7c4fc"             ]          ] :"wires"       ,y": 100"

,"z": "5e11e736.5672e"       ,"type": "http request"       ,"id": "5453caaf.c7c4fc"       }   ,{   [
,"name": "hyperledger - Create Persishable Demo

,"url": "http://{{FabricIP}}:3000/api/SetupDemo"       ,"ret": "obj"       ,"method": "POST"

]          ] :"wires"       ,y": 100"       ,x": 730"       ,"" :"tls"

}   ,{   [       [          "c1424e9d.e9923"

,"type": "debug"       ,"id": "c1424e9d.e9923"

,"name": "Blockchain Asset response"       ,"z": "5e11e736.5672e"

,"complete": "true"       ,"console": "false"       ,active": true"

},{[] :"wires"       ,y": 100"       ,x": 1040"

,"id": "811a05a4.4d5827"

,"name": "Start Collect data"       ,"z": "ac3b7d6e.3a52"       ,"type": "inject"
```

,"" :"repeat" ,"payloadType": "bool" ,"payload": "true" ,"" :"topic"

,y": 270" ,x": 150" ,onceDelay": 0.1" ,once": false" ,"" :"crontab"

} ,{ [["73248645.0baa38"]] :"wires"

,"id": "73248645.0baa38"

,"type": "function"

,"z": "5e11e736.5672e"

,"name": "Build a TemperatureReading Transaction"

func": "// A TemperatureReading Blockchain transaction looks like this:\n// {\n// \"$class\": "
\"org.acme.shipping.perishable.TemperatureReading\",\n// \"celsius\": 0,\n// \"latitude\":
\"string\",\n// \"longitude\": \"string\",\n// \"readingTime\": \"string\",\n// \"shipment\":
\"resource:org.acme.shipping.perishable.Shipment#NNNN\"\n// }\nvar
FabricIP=global.get(\"HyperLedgerFabricIP\");\n\n// This function will be passed a msg.payload from the
Particle Electron\n// AssetTrackerTemperatureEvent which looks like this:\n// {\"AssetID\":\"<Particle
Device ID>\",\n// \"Temperature\":{\"Celsius\":22.625,\"Fahrenheit\":72.724998},\n//
\"gps\":{\"lat\":40.972904,\"lon\":-74.092216,\"accuracy\":0},\n// \"timestamp\":\"2018-02-
14T15:06:41.178Z\"}\n\nmsg.payload = {\n \"$class\" :
\"org.acme.shipping.perishable.TemperatureReading\",\n \"celsius\" : Math.random() * 4 +35,\n
\"readingTime\": Math.random() * 4 +35,\n \"latitude\" : Math.random() * 75 + 2000,\n \"longitude\"
: Math.random() * 60 + 2000,\n \"shipment\" :
\"resource:org.acme.shipping.perishable.Shipment#320022000251363131363432\"};\nTempReadingMs
g = JSON.stringify(msg.payload);\n\nmsg.url =
\"http://\"+FabricIP+\":3000/api/TemperatureReading?data=\"+TempReadingMsg;\nmsg.headers = {\n
,"; \"Content-Type\":\"application/json\",\n \"Accept\":\"application/json\"\n};\nreturn msg

,outputs": 1"

,noerr": 0"

,x": 480"

["fa963268.4a59d" ,"67e6c7c0.3b7be"]] :"wires" ,y": 270"

,"type": "http request" ,"id": "fa963268.4a59d" } ,{ [

,"z": "5e11e736.5672e"

,"name": "Blockchain POST TemperatureReading"

,"url": "http://FabricIP:3000/api/TemperatureReading" ,"ret": "obj" ,"method": "POST"

,y": 270" ,x": 830" ,"" :"tls"

["b946b867.8c2eb8"]] :"wires"

,"z": "5e11e736.5672e" ,"type": "debug" ,"id": "67e6c7c0.3b7be" } ,{ [

,"console": "false" ,active": true" ,"" :"name"

} ,{ [] :"wires" ,y": 220" ,x": 750" ,"complete": "false"

,"" :"name" ,"z": "5e11e736.5672e" ,"type": "debug" ,"id": "b946b867.8c2eb8"

,"complete": "false" ,"console": "false" ,active": true"

} ,{ [] :"wires" ,y": 270" ,x": 1100"

,"z": "ac3b7d6e.3a52" ,"type": "inject" ,"id": "811a05a4.4d5829"

,"payload": "true" ,"" :"topic" ,"name": "Start Collect data"

,"" :"crontab" ,"" :"repeat" ,"payloadType": "bool"

,onceDelay": 0.1" ,once": false"

]] :"wires" ,y": 410" ,x": 150"

} ,{ [["a2e6555e.5e7"

,"z": "5e11e736.5672e" ,"type": "function" ,"id": "a2e6555e.5e7"

,"name": "Build a AccelReading Transaction"

func": "// An AccelReading Blockchain transaction looks like this:\n// {\n// \"$class\": "
\"org.acme.shipping.perishable.AccelReading\",\n// \"accel_x\": 0,\n// \"accel_y\": 0,\n//
\"accel_z\": 0,\n// \"latitude\": \"string\",\n// \"longitude\": \"string\",\n// \"readingTime\":
\"string\",\n// \"shipment\": \"resource:org.acme.shipping.perishable.Shipment#NNNN\"\n// }\n\n//
This function will be passed a msg.payload from the Particle Electron\n// AssetTrackerAccelerationEvent
which looks like this:\n// Example:\n// {\"AssetID\":\"<Particle Device ID>\",\n//
\"Accel\":{\"x\":23264,\"y\":-20960,\"z\":-2448},\n// \"gps\":{\"lat\":40.972904,\"lon\":-
74.092216,\"accuracy\":0},\n// \"timestamp\":\"2018-02-14T15:16:44.284Z\"}\n\nvar
FabricIP=global.get(\"HyperLedgerFabricIP\");\n\nmsg.payload = {\n \"$class\" :
\"org.acme.shipping.perishable.AccelReading\",\n \"accel_x\" : Math.random() * 4 +50,\n
\"accel_y\" : Math.random() * 4 +90,\n \"accel_z\" : Math.random() * 4 +70,\n \"readingTime\":
Math.random() * 4 +35,\n \"latitude\" : Math.random() * 4 +35,\n \"longitude\" : Math.random() *
4 +35,\n \"shipment\" :
\"resource:org.acme.shipping.perishable.Shipment#320022000251363131363432\"};\nAccelReadingMs
g = JSON.stringify(msg.payload);\n\nmsg.url =
\"http://\"+FabricIP+\":3000/api/AccelReading?data=\"+AccelReadingMsg;\nmsg.headers = {\n
,";\"Content-Type\":\"application/json\",\n \"Accept\":\"application/json\"\n};\nreturn msg

,outputs": 1"

,noerr": 0"

,x": 480"

,y": 410"

} ,{ [["f743589c.73ae68" ,"a502b4b1.f9b898"]] :"wires"
,"z": "5e11e736.5672e" ,"type": "http request" ,"id": "f743589c.73ae68"

,"name": "Blockchain POST AccelReading"

,"url": "http://FabricIP:3000/api/AccelReading" ,"ret": "obj" ,"method": "POST"

,y": 410" ,x": 810" ,"" :"tls"

["ffb346ae.85fdf8"]] :"wires"

,"id": "a502b4b1.f9b898" } ,{ [

,"" :"name" ,"z": "5e11e736.5672e" ,"type": "debug"

,"complete": "false" ,"console": "false" ,active": true"

} ,{ [] :"wires" ,y": 372" ,x": 750"

,"z": "5e11e736.5672e" ,"type": "debug" ,"id": "ffb346ae.85fdf8"

,"console": "false" ,active": true" ,"" :"name"

,{ [] :"wires" ,y": 410" ,x": 1100" ,"complete": "false"

}

,"id": "811a05a4.4d5824"

,"name": "Start Collect data" ,"z": "ac3b7d6e.3a52" ,"type": "inject"

,"" :"repeat" ,"payloadType": "bool" ,"payload": "true" ,"" :"topic"

] :"wires" ,y": 510" ,x": 150" ,onceDelay": 0.1" ,once": false" ,"" :"crontab"

"41c2ad12.d9734c"]

} ,{ [[

,"id": "41c2ad12.d9734c"

,"type": "function"

,"z": "5e11e736.5672e"

,"name": "Build a GpsReading Transaction"

func": "// A GpsReading transaction looks like this:\n// {\n// \"$class\": "
\"org.acme.shipping.perishable.GpsReading\",\n// \"readingTime\": \"\",\n// \"readingDate\":
\"\",\n// \"latitude\": \"\",\n// \"latitudeDir\": \"N\",\n// \"longitude\": \"\",\n// \"longitudeDir\":
\"N\",\n// \"shipment\": \"resource:org.acme.shipping.perishable.Shipment#NNNN\"\n// }\n\n//
This function will be passed a msg.payload from the Particle Electron\n//
AssetTrackerTemperatureEvent or AssetTrackerAccelerationEvent\n// The msg.payload will look like
this:\n// {\"AssetID\":\"<Particle Device ID>\",\n//
\"Temperature\":{\"Celsius\":22.625,\"Fahrenheit\":72.724998},\n//

\"gps\":{\"lat\":40.972904,\"lon\":-74.092216,\"accuracy\":0},\n// \"timestamp\":\"2018-02-14T15:06:41.178Z\"}\n//\n// {\"AssetID\":\"<Particle Device ID>\",\n// \"Accel\":{\"x\":23264,\"y\":-20960,\"z\":-2448},\n// \"gps\":{\"lat\":40.972904,\"lon\":-74.092216,\"accuracy\":0},\n// \"timestamp\":\"2018-02-14T15:16:44.284Z\"}\n\n// Not certain why the Perishable Model splits the readingTime and readingDate. \n// That's useless, just store the timestamp (which is preserved in readingTime)\n// The model validation won't allow a blank readingDate so construct the properly formatted date here\nvar msec = Date.parse(msg.payload.timestamp);\nvar d = new Date(msec);\nvar ReadingDate = d.getFullYear() + \"-\" + ('0' + (d.getMonth()+1)).slice(-2) + '-' + ('0' + d.getDate()).slice(-2);\nvar FabricIP=global.get(\"HyperLedgerFabricIP\");\n\nmsg.payload = {\n \"$class\": \"org.acme.shipping.perishable.GpsReading\",\n \"readingTime\": Math.random() * 4 +35,\n \"readingDate\": Math.random() * 4 +3,\n \"latitude\": Math.random() * 4 +35,\n \"latitudeDir\": \"N\",\n \"longitude\": Math.random() * 4 +35,\n \"longitudeDir\": \"N\",\n \"shipment\": \"resource:org.acme.shipping.perishable.Shipment#320022000251363131363432\"};\nGpsReadingMsg = JSON.stringify(msg.payload);\n\nmsg.url = \"http://\"+FabricIP+\":3000/api/GpsReading?data=\"+GpsReadingMsg;\nmsg.headers = {\n ,\";\"Content-Type\":\"application/json\",\n \"Accept\":\"application/json\"\n};\nreturn msg

]] :"wires" ,y": 510" ,x": 470" ,noerr": 0" ,outputs": 1"

} ,{ [["6a46fe3d.ad24" ,"90e5c278.63f818"

,"z": "5e11e736.5672e" ,"type": "http request" ,"id": "6a46fe3d.ad24"

,"method": "POST" ,"name": "Blockchain POST GpsReading"

,"url": "http://FabricIP:3000/api/GpsReading" ,"ret": "obj"

} ,{ [["46ca2f88.f3fc4"]] :"wires" ,y": 510" ,x": 810" ,"" :"tls"

,"z": "5e11e736.5672e" ,"type": "debug" ,"id": "90e5c278.63f818"

,"complete": "false" ,"console": "false" ,active": true" ,"" :"name"

} ,{ [] :"wires" ,y": 472" ,x": 750"

,"z": "5e11e736.5672e" ,"type": "debug" ,"id": "46ca2f88.f3fc4"

,"complete": "false" ,"console": "false" ,active": true" ,"" :"name"

[{ [] :"wires" ,y": 510" ,x": 1100"

Appendix B

Create smart contract code

```
**/
A business network for shipping perishable goods *
The cargo is temperature controlled and contracts *
can be negociated based on the temperature *
readings received for the cargo *
/*
namespace org.acme.shipping.perishable
**/
The type of perishable product being shipped *
/*
} enum ProductType
o BANANAS
o APPLES
o PEARS
o PEACHES
o COFFEE
o MEDICINE
{
**/
The status of a shipment *
/*
} enum ShipmentStatus
o CREATED
o IN_TRANSIT
o ARRIVED
```

{

**/

Directions of the compass *

/*

} enum CompassDirection

o N

o S

o E

o W

{

**/

An abstract transaction that is related to a Shipment *

/*

} abstract transaction ShipmentTransaction

Shipment shipment <--

{

**/

An temperature reading for a shipment. E.g. received from a *

device within a temperature controlled shipping container *

/*

} transaction TemperatureReading extends ShipmentTransaction

o Double celsius

o String latitude

o String longitude

o String readingTime

{

**/

A notification that a shipment has been received by the *

importer and that funds should be transferred from the importer *

```
.to the grower to pay for the shipment *
/*
} transaction ShipmentReceived extends ShipmentTransaction
} transaction AccelReading extends ShipmentTransaction{
o Double accel_x
o Double accel_y
o Double accel_z
o String latitude
o String longitude
o String readingTime
} transaction GpsReading extends ShipmentTransaction{
o String readingTime
o String readingDate
o String latitude
o CompassDirection latitudeDir
o String longitude
o CompassDirection longitudeDir
{
**/
A shipment being tracked as an asset on the ledger *
/*
} asset Shipment identified by shipmentId
o String shipmentId
o ProductType type
o ShipmentStatus status
o Long unitCount
Contract contract <--
o TemperatureReading[] temperatureReadings optional
o AccelReading[] AccelReadings optional
```

o GpsReading[] gpsReadings optional

{

**/

Defines a contract between a Grower and an Importer to ship using *

a Shipper, paying a set unit price. The unit price is multiplied by *

a penality factor proportional to the deviation from the min and max *

.negociated temperatures for the shipment *

/*

} asset Contract identified by contractId

o String contractId

Grower grower <--

Shipper shipper <--

Importer importer <--

o DateTime arrivalDateTime

o Double unitPrice

o Double minTemperature

o Double maxTemperature

o Double minPenaltyFactor

o Double maxPenaltyFactor

o Double maxAccel

{

**/

A concept for a simple street address *

/*

} concept Address

o String city optional

o String country

o String street optional

o String zip optional

```
{
**/
An abstract participant type in this business network *
/*
} abstract participant Business identified by email
o String email
o Address address
o Double accountBalance
{
**/
A Grower is a type of participant in the network *
/*
} participant Grower extends Business
{
**/
A Shipper is a type of participant in the network *
/*
} participant Shipper extends Business
{
**/
An Importer is a type of participant in the network *
/*
} participant Importer extends Business
{
**/
JUST FOR INITIALIZING A DEMO *
/*
} transaction SetupDemo
{
```

} event TemperatureThresholdEvent

o String message

o Double temperature

o String latitude

o String longitude

o String readingTime

Shipment shipment <--

{

} event AccelerationThresholdEvent

o String message

o Double accel_x

o Double accel_y

o Double accel_z

o String latitude

o String longitude

o String readingTime

Shipment shipment <--

{

} event ShipmentInPortEvent

o String message

Shipment shipment <--

{

Appendix C

Design of simulated chart result

```
<!DOCTYPE html
   PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
   <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
   <title>Temperature</title>
   <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/3.4.0/css/bootstrap.min.css">
   <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.3.1/jquery.min.js"></script>
   <script src="https://maxcdn.bootstrapcdn.com/bootstrap/3.4.0/js/bootstrap.min.js"></script>
   <link rel="stylesheet" href="https://www.w3schools.com/w3css/4/w3.css">
   <link href="https://fonts.googleapis.com/css?family=Noto+Serif" rel="stylesheet">
   <link rel="stylesheet" type="text/css" href="css/form2.css" media="all">
   <script type="text/javascript" src="js/form2.js"></script>
   <script src="plotly-latest.min.js"></script>
   <style>
      h3 {
         font-size: 268%;
         font-weight: bold;
         line-height: 125%;
         font-family: 'Noto Serif', serif;
         text-align: center;
      }
      #li_2 {
         text-align: center;
      }
```

```
        #txtId {
            margin-top: 3%;
        }
        .buttons {
            text-align: center;
        }
        #searchByShiptId {
            font-weight: bold;
        }
    </style>
</head>
<body id="main_body">
    <!--nav-->
    <nav class="navbar navbar-inverse" style="margin-bottom: 0;border-radius: 0;">
        <div class="container-fluid">
            <div class="navbar-header">
                <a class="navbar-brand" href="graduation.html"><span style="color:#dd6215
">T</span>racking system</a>
            </div>
            <ul class="nav navbar-nav navbar-right">
                <li class="active"><a href="graduation.html"><span style="color:#dd6215
">HOME</span></a></li>
                <li><a href="form.html">REGISTER</a></li>
                <li><a href="track%20(2).html">LOGIN</a></li>
                <li><a href="#">SERVICES</a></li>
                <li><a href="#lastsection">CONTACT</a></li>
            </ul>
        </div>
    </nav>
    <img id="top" src="images/top1.png" alt="" style="width: 90%">
```

```
    <div id="form_container" style="width: 90%">
        <h1><a>Shipment</a></h1>
        <form id="form_52247" class="appnitro" method="post" action=""></form>
        <h3>Temperature changes with the shipment ID <p id="shipId"></p>
        </h3>
        <div id="chart" style="overflow-x: hidden"></div>
        <table class="w3-table-all">
            <tbody id="tbody"></tbody>
        </table>
    </div>
    <img id="bottom" src="images/bottom1.png" alt="" style="width:90%">
    <!--
        Hi
    Reading "JSON" data (Shipment, Contract) Using Shipment ID, and fetched Contract ID,
    from the Hyperledger composer REST API,
-->
    <script>
        // Retrieve data from the session Storage, (the shipment id)
        var shipID;
        // Get the id
        shipID = sessionStorage.getItem("shipmentIdValue");
        document.getElementById("shipId").innerText = shipID;
        // counter
        var i = 0;
        // to store temp data.
        var tempData;
        // to store celsius data globally
        var celsius;
        var getJSON = function (url, callback) {
```

```
    var xhr = new XMLHttpRequest();
    xhr.open('get', url, true);
    xhr.responseType = 'json';
    xhr.onload = function () {
      var status = xhr.status;
      if (status == 200) {
        callback(null, xhr.response);
      } else {      callback(status);        }      };
    xhr.send();       }
  setInterval(function () {
    getJSON('http://34.83.2.150:3000/api/Shipment/' + shipID,
      function (err, data) {
        if (err != null) {
          alert('Something went wrong: ' + err);
        } else {
          // Check if the Object is empty or not.
          // @obj Parameter will be any data sent by the api (Shipment, Contract, and readings)
          function isEmptyObject(obj) {
            for (var key in obj) {
              return false;
            }  return true;  }
          tempData = data["temperatureReadings"][i];
          if (isEmptyObject(tempData)) {
            console.log("DATASOURCE IS EMPTY!");
            // Stopping here,
            return;
          } else {   // Continue counting.
            i++;     }
          for (var key in tempData) {
```

```
            if (key === "celsius") {  if (celsius > 39) {  alert(celsius + " IS HIGHHHH!")   }
                console.log(key + tempData[key]);
                celsius = tempData[key];
                function getData() { return celsius; }
                    Plotly.plot('chart', [{  y: [getData()], type: 'graph'  }]);
                Plotly.extendTraces('chart', { y: [[getData()]] }, [0])                  }           }       }
        console.log(i);
        /*
        setInterval(function () {
           Plotly.extendTraces('chart', { y: [[getData()]] }, [0])
        }, 200);              */          });       }, 4000);
  </script>
</body>
</html>
```

Arabic summary of project

ملخص المشروع باللغة العربية

في حين أن جميع أنواع الممتلكات قد تتلف أو تتلف بسبب أي عدد من الأسباب ، بما في ذلك الكوارث الطبيعية والسرقة والتخريب والحرائق وما إلى ذلك ، فإن البضائع والمعدات التي تمر بمرورها تواجه مخاطر متزايدة ، سواء بسبب الأسباب التقليدية للخسارة ومن التعرضات المميزة للعبور. وعلى وجه الخصوص ، تتعرض هذه البضائع والمعدات لخطر متزايد من السرقة أو الاختفاء وتواجه حالات تعرض فريدة من نوعها على الطريق ، بما في ذلك تصادم أو اضطراب مركبة النقل وتغييرات الحضانة وتعارض نقل الملكية والتعبئة غير الصحيحة والإهمال نيابة عن الناقل.

يعالج هذا النمط المشكلة الحقيقية للغاية المتمثلة في التسليم الآمن للبضائع القابلة للتلف (الغذاء والدواء والماشية ، إلخ) الحساسة للظروف البيئية أثناء الشحن. كل شحنة من البضائع القابلة للتلف لديها عتبات (متطلبات التبريد ، وتجنب الصدمات أو الاهتزاز) لحماية البضائع من التلوث أو التلف. إذا تجاوزت الشحنة هذه العتبات ، فإن البضائع تالفة وقد تصبح خطراً على الصحة.

من خلال تسجيل تفاصيل (أين ، ماذا ، ومتى) شحنة عانت من ظروف قاسية (عتبات محددة في العقد الذكي) ، يمكن للمطورين التحقق من تسليم البضائع بنجاح (أو لا). ثم ، يعتمد الدفع على التسليم الناجح.

تتبع شروط الشحنة عبر عدة مشاركين باستخدام بلوكتشين يوفر التحقق والثقة في هذه العمليات. يوفر هذا المشروع حلاً مثاليًا لتتبع أصول بلوكتشين انترنت الاشياء عبر.