

# **IMPLEMENTING CORE FUNCTIONALITIES OF PDS USING HYPERLEDGER FABRIC**

**A PROJECT REPORT**

*Submitted by*

**GOKUL E [Register Number : 211418104066]  
HARI N [Register Number : 211418104074]  
HEMANTH S [Register Number : 211418104085]**

*in partial fulfillment for the award of the degree*

*of*

**BACHELOR OF ENGINEERING**

*in*

**COMPUTER SCIENCE AND ENGINEERING**



**PANIMALAR ENGINEERING COLLEGE**

**(An Autonomous Institution, Affiliated to Anna University, Chennai)**

**MAY 2022**



# **IMPLEMENTING CORE FUNCTIONALITIES OF PDS USING HYPERLEDGER FABRIC**

**A PROJECT REPORT**

*Submitted by*

**GOKUL E [Register Number : 211418104066]  
HARI N [Register Number : 211418104074]  
HEMANTH S [Register Number : 211418104085]**

*in partial fulfillment for the award of the degree*

*of*

**BACHELOR OF ENGINEERING**

*in*

**COMPUTER SCIENCE AND ENGINEERING**

**PANIMALAR ENGINEERING COLLEGE**

**(An Autonomous Institution, Affiliated to Anna University, Chennai)**

**MAY 2022**

**PANIMALAR ENGINEERING COLLEGE**  
(An Autonomous Institution, Affiliated to Anna University, Chennai)

**BONAFIDE CERTIFICATE**

Certified that this project report “**IMPLEMENTING CORE FUNCTIONALITIES OF PDS USING HYPERLEDGER FABRIC**” is the bonafide work of “GOKUL E (211418104066), HEMANTH S (211418104085), HARI N (211418104074)” who carried out the project work under my supervision.

**SIGNATURE**

**Dr. S. MURUGAVALLI, M.E., Ph.D.,  
HEAD OF THE DEPARTMENT**

DEPARTMENT OF CSE,  
PANIMALAR ENGINEERING COLLEGE,  
NASARATHPETTAI,  
POONAMALLEE,  
CHENNAI-600 123.

**SIGNATURE**

**Dr. G. SENTHIL KUMAR M.E., Ph.D.,  
SUPERVISOR  
PROFESSOR**

DEPARTMENT OF CSE,  
PANIMALAR ENGINEERING COLLEGE,  
NASARATHPETTAI,  
POONAMALLEE,  
CHENNAI-600 123.

Certified that the above candidate(s) was/ were examined in the Anna University Project

Viva-Voce Examination held on.....

**INTERNAL EXAMINER**

**EXTERNAL EXAMINER**

## **DECLARATION BY THE STUDENT**

We **GOKUL E (211418104066), HEMANTH S (211418104085), HARI N (211418104074)** hereby declare that this project report titled **“IMPLEMENTING CORE FUNCTIONALITIES OF PDS USING HYPERLEDGER FABRIC”**, under the guidance of **Project Guide Dr. G. SENTHIL KUMAR, M.E., Ph.D.**, is the original work done by us and we have not plagiarized or submitted to any other degree in any university by us.

## ACKNOWLEDGEMENT

We would like to express our deep gratitude to our respected Secretary and Correspondent **Dr. P. CHINNADURAI, M.A., Ph.D.**, for his kind words and enthusiastic motivation, which inspired us a lot in completing this project.

We express our sincere thanks to our Directors **Tmt. C. VIJAYARAJESWARI, Dr. C. SAKTHI KUMAR, M.E., Ph.D.**, and **Dr. SARANYASREE SAKTHI KUMAR B.E., M.B.A., Ph.D.**, for providing us with the necessary facilities to undertake this project.

We also express our gratitude to our Principal **Dr. K. MANI, M.E., Ph.D.**, who facilitated us in completing the project.

We thank the Head of the CSE Department, **Dr. S. MURUGAVALLI, M.E., Ph.D.**, for the support extended throughout the project.

We would like to thank my **Project Guide Dr. G. SENTHIL KUMAR, M.E., Ph.D.**, and all the faculty members of the Department of CSE for their advice and encouragement for the successful completion of the project.

**GOKUL E  
HEMANTH S  
HARIN**

## **ABSTRACT**

Public Distribution System (PDS) is an essential scheme for food security introduced by Indian Government under the Ministry of Consumer Affairs, Food and Public Distribution. Main purpose of the system is to make sure that edible and non-edible supplies reach to marginalized section of Indian society at subsidized cost. However, PDS is vulnerable to leakages and issues with procurement and storage. Indian Government spends more than Rs. 750 billion for PDS and there are more than 5 lakh Fair Price Shops in India. Computerization for PDS functionalities and AADHAR Biometrics verification is carried out in many states of India. It is found out that leakages have been mitigated as much as possible. But there are many challenges still unnoticed in PDS like opaque tender procurements, diversion of standard edible and non-edible supplies, negligent wastage accountability and corrupt book adjustments in warehouses. These transparency and accountability related challenges of PDS could be met with distributed immutable ledger framework like Hyperledger Fabric. Asset tracking aids in tamper detection, wastage avoidance and logistics optimization. Proof of Concept on functionalities of PDS using Fabric would imply that blockchain could be adopted in various food supply chains and logistics.

## LIST OF TABLES

SL. NO.	CONTENT	PG. NO.
2.1	LITERATURE SURVEY ON USE CASES OF BLOCKCHAIN	10
2.2	LITERATURE SURVEY ON IMPACT OF BLOCKCHAIN IN SUPPLY CHAIN	13
7.1	RFID SCANNING TESTCASE	65
7.2	AFTER STOPPING MODE TESTCASE	65

## LIST OF FIGURES

SL. NO.	CONTENT	PG. NO.
4.1	ER DIAGRAM FOR SUPPLY ASSET	23
4.2	DFD DIAGRAM FOR BLOCKCHAIN	24
4.3	DATA DICTIONARY DIAGRAM FOR SUPPLY ASSET	25
4.4	USE CASE DIAGRAM FOR PDS SUPPLY	26
4.5	SEQUENCE DIAGRAM FOR CREATE ASSET	27
4.6	ACTIVITY DIAGRAM FOR CREATE ASSET	28
4.7	PACKAGE DIAGRAM FOR PDS SUPPLY	29
5.1	OVERALL MODULES ILLUSTRATION ON SYSTEM	30
8.1	ESP8266 PROGRAMMING WITH ARDUINO IDE	66
8.2	CREATING BLOCKCHAIN SAMPLE NETWORK	66
8.3	DOCKERS CONTAINERS CREATED	67
8.4	CHAINCODE INSTALLATION ON PEERS	67
8.5	APPLICATION GATEWAY COMPILED	68
8.6	ESP8266 INTERFACED WITH RC522	68
A1.1	CREATE ASSET MODE FORM	70
A1.2	TRANSFER ASSET MODE FORM	71
A1.3	MODE STARTED PLACEHOLDER	72
A1.4	MODE STOP PLACEHOLDER	73
A1.5	MODE MENU SELECTION	74



## LIST OF SYMBOLS, ABBREVIATIONS

SL. NO.	ACRONYM	ABBREVIATION
1.	PDS	Public Distribution Scheme
2.	FCI	Food Corporation of India
3.	TNCSC	Tamil Nadu Civil Supplies Corporation
4.	UIDAI	Unique Identification Authority of India
5.	GUEST	Go Uniform Evaluate Solve Test
6.	POS	Point Of Sale
7.	MRM	Modern Rice Mills
8.	UPDS	Universal Public Distribution Scheme
9.	CAP	Cover And Plinth
10.	DPC	Direct Procurement Center
11.	FPS	Fair Price Shop
12.	API	Application Programming Interface

## TABLE OF CONTENTS

CHAPTER NO.	TITLE	PAGE NO.
	<b>ABSTRACT</b>	iv
	<b>LIST OF TABLES</b>	v
	<b>LIST OF FIGURES</b>	vi
	<b>LIST OF SYMBOLS, ABBREVIATIONS</b>	vii
<b>1.</b>	<b>INTRODUCTION</b>	
	1.1 Problem Definition	1
<b>2.</b>	<b>LITERATURE SURVEY</b>	7
<b>3.</b>	<b>SYSTEM ANALYSIS</b>	
	2.1 Existing System	16
	2.2 Proposed system	17
	2.3 Feasibility Study	18
	2.4 Hardware Environment	19
	2.5 Software Environment	21
<b>4.</b>	<b>SYSTEM DESIGN</b>	
	4.1. ER diagram	23
	4.2 Data dictionary	24
	4.3 Data Flow Diagram	25
	4.4 UML Diagrams	26

<b>5.</b>	<b>SYSTEM ARCHITECTURE</b>	
	5.1 Module Design Specification	30
	5.2 Algorithms	35
<b>6.</b>	<b>SYSTEM IMPLEMENTATION</b>	
	6.1 Client-side coding	37
	6.2 Server-side coding	44
<b>7.</b>	<b>SYSTEM TESTING</b>	
	7.1 Unit Testing	63
	7.2 Integration Testing	64
	7.3 Test Cases & Reports	65
<b>8.</b>	<b>CONCLUSION</b>	
	8.1 Results & Discussion	66
	8.2 Conclusion and Future Enhancements	69
	<b>APPENDICES</b>	
	A.1 Sample Screens	70
	<b>REFERENCES</b>	75

# **1. INTRODUCTION**

## **1.1. PROBLEM DEFINITION**

In India, it is estimated that 291.95 million tons of food grains were produced in 2019-2020. Out of the above-mentioned number, around 4.6-6% is wasted according to National Academy of Agricultural Sciences. This implies that the loss is around 12-18 million tons and experts anticipate that the loss could be higher. 10% of 1000 lakh MT wheat produced every year gets wasted due to improper storage methods. Food Corporation of India (FCI) is known to be the major organization which causes food wastage. Food Wastage happens in FCI due to various reasons. Food reserves are maintained above the buffer norms at FCI Warehouses. For Example, FCI rice stocks in January 2020 was at 21 MT whereas the actual buffer requirement is 7.6MT. It is also noted that inadequate storage facilities of both central and state government make them to resort to other alternatives like cover and plinth or taking warehouses based upon lease. These alternative measures are found to be the reason for food wastage as edible supplies may get spoiled due to rain, moisture and humidity.

State Governments collaborate with Central Government to implement PDS across the nation. State Governments of India handle PDS in diverging methods as they need to adapt to fulfil the needs of the respective state's population. This adaption is required due to population dynamics, economics, and food culture. In this paper, Tamil Nadu is taken as example to illustrate the process of PDS. While other states take measures in guise of Targeted Public Distribution System (TPDS) where only marginalized sections are catered, Tamil Nadu is succeeding

in eradicating food scarcity through Universal Public Distribution System (UPDS). More than 2 Crore family cards are getting benefitted through UPDS and there is a network of more than 33,000 Fair Price Shops. UPDS in Tamil Nadu is irrespective of Income Slabs.

In Tamil Nadu, Tamil Nadu Civil Supplies Corporation (TNCSC) is responsible for procurement, storage, and distribution of edible and non-edible supplies through network of Fair Price Shops and Warehouses. TNCSC gets the funds for operations and procurements from State and Central Government. TNCSC procures supplies through various methods. Direct Procurement Centers are designated places where rice and other cereals are procured directly from the farmers by TNCSC. Procurement price is decided by TNCSC at the start of Kharif season. In this process, Farmers register their cultivated paddy produce in the online portal with the help of Village Administrative Officer (VAO) along with land details, bank account details and verification documents like AADHAR. The required authorities will check whether the registered member is a farmer or not by verifying the land details. TNCSC will accept the request based upon their requirements, Farmer will be notified by SMS about procurement details. After the proper weighing and procurement of the cultivated produce, Bank transfer will be initiated by DPC. Staffs at DPC will transfer paddy manually from farmer's gunny sacks to standard 30kg TNCSC provided gunny sacks. TNCSC provided gunny sacks are stenciled by the staffs at DPC in which details like lot number, farmer token no., variety of paddy is mentioned. But, marking is not done at most cases due to negligence, and this is where accountability misses. Modern Rice Mills (MRM) are state owned, and they have more capacity comparatively whereas Hulling Agents are private agents who are appointed by state government for hulling through tender.

Procured supplies should be within specifications. For example, moisture of the procured paddy should be below 17%. If the paddy has moisture below 17%, then it would be sent to state buffer warehouses for storage otherwise it is rejected. In rainy season, moisture specification is exempted till 20%. In the latter case, paddy would be sent swiftly to Modern Rice Mill (or) Hulling Agent for drying and hulling paddy into rice. If space is not available in buffer warehouses, then Cover and Plinth (CAP) Storage will be chosen. TNCSC gets land for rent and paddy sacks are stored in those open spaces covered by Polypropylene ground (PP) covers (or) Tarpaulin covers. Sacks are stored in cone structure and crisscross manner. Ropes would be tied across the cover, and it is opened on every morning and closed by night for first 10 days and weekly twice for following days afterwards. This makes sure that paddy sack gets enough amount of air to get dried otherwise there are possibility for fungal infections. Pest Control and Fumigation is done in both warehouses and CAP storages. CAP storage is not a suitable method as there are high probability of paddy getting infected and wasted. Warehouses are also checked by quality inspection team of TNCSC to ensure that there is no rodent, pest infestations and no holes in the roof.

These paddy sacks procured are allotted and processed in First Come First Serve (FCFS) way. Hulling agents are chosen in the order by which they have applied and got approved. Security deposit and bank guarantee should be submitted according to the capacity of the mill. Hulling agent's facility is also inspected by government before the approval, and it is made sure that processing capacity is sufficient, and standards are effectively maintained. After Allotment and Transportation, hulling agent must do the whole process (drying and hulling) within a month. Hulling Agent should reuse the bags provided and if there are

excess bags then it should be returned with care to TNCSC. After processing, Rice is sent from hulling agent to state owned warehouses through TNCSC approved tendered transportation.

There are two types of warehouses owned by state government. They are Operational Warehouses and Buffer Warehouses. Buffer warehouse is used to store excess products and Operational Warehouses are used to store supplies which are to be sent to fair price shops. There are two types of rice based upon the treatment. They are boiled rice and raw rice. A Grade and Common are two subtypes within these types.

After Hulling, Rice sacks are sent from MRM (or) Hulling agent's mill to Operational warehouses. For every 200 bags in consignment from hulling agent, at most 50 bags must be sampled. If the rice sacks are from MRM, then the quality will not be checked at arrival as it is already inspected in MRM before departure. In Warehouse, Sacks are piled up in stack formation and the size of bottom of stack is 30ftx20ft. Poly Filler / Wooden Crates would be the base for stack.

There is another source of procurement for TNCSC (i.e.) FCI Procurement. FCI procures wheat and paddy from various states, and they store it in their central government owned warehouses. As the demand for State Government cannot be met up by supply from DPC, State procures from FCI warehouses. Rice (or) Wheat allotment from FCI procurement is released against payment and FCFS. It acts upon priority basis from divisions which means that the FCI divisional office is responsible for the allocation in a certain jurisdiction of state (i.e.) a part of district. As it is a divisional request, consignment is directly sent from FCI warehouse to Operational warehouse.

There are various movement (i.e.) transportation methods for transferring consignments. Mostly used movement is rail-head transportation. Railways is predominantly used medium for transporting the consignments. It is safer and can carry to a longer range. Another movement is roadway transportation through lorries and trucks. This is used to carry the consignment from one district to another. Lorry Transportation companies are approved and allotted through tenders.

Fair Price Shops are the endpoints through which citizens get essential supplies at subsidized cost. There are two types of Fair Price Shops (FPS) in Tamil Nadu. They are State owned shop and Cooperative shop. State owned shops are mainly found in cities and they are less in number compared to cooperative shops. In case of a cooperative shop, cooperative lead society will perform movement. Deputy Registrar and Joint Registrar are responsible in ensuring that respective cooperative society are functioning properly. Based on the population under an FPS, allotment of essential supplies from warehouse will be given to the FPS on monthly basis.

There are many malpractices and loopholes possible in the traditional PDS Supply Chain System. Businesspeople gather paddy and wheat from farmers for a lower rate and provide it to State Government through DPC in guise of farmers. This happens because the registration for farmer at DPC takes more time due to the verification and payment process. Bribe from farmers is asked by staffs at DPC and the staffs are more negligent and reluctant. For example, Load men ask for bribe to carry gunny bags. Due to negligence in procurement, Paddy and Wheat get wasted in rainy seasons. Tender for other essential supplies like



Kerosene and Dal varieties are done from other parties and merchants. Tenders are intentionally selected at higher prices than market prices as it paves way for corruption and government bureaucrats get exorbitant bribe. As said earlier, there are many leakages and diversions involved in the system. Hulling agents can manipulate allotted paddy for their personal benefit, and they indulge in diverting fine variety good quality rice. Accountability of consignment is not ensured, and essential supplies are sold in black market to other districts. Mentioned loopholes makes us to aim for a better accountable asset tracking system than the traditional system.

## 2. LITERATURE SURVEY

Bitcoin is a revolution which made the world to conceptualize distributed immutable ledger as Blockchain. In Bitcoin, Blockchain is used to store the transactions in various public peers by reaching consensus through proof of work where every peer competes each other to append block of transactions to the ledger (Satoshi Nakamoto, 2008) [1]. Blockchain acts as a trustful system between various parties which helps in various use cases for eliminating redundancy and unnecessary intermediaries. Decentralized Application, Decentralized Autonomous Organizations are built using blockchain as a trustful framework and Smart Contract on top of it as event-driven code for enabling transactions to happen which serves as backbone for Applications to run based upon it. Broad spectrum of use cases possible using blockchain are FinTech, Health Care, Governance, Intellectual Property Protection, Identity Verification and Supply Chain Management (Melanie Swan, 2015) [2]. GUEST (GO, UNIFORM, EVALUATE, SOLVE AND TEST) methodology can be applied to formulate blockchain designs for specific Supply Chain Management and Logistics use case. Comparing Blockchain Frameworks, it narrows down to Hyperledger Fabric for efficient use of Permissioned Blockchain Network (G. Perboli, 2018) [3].

Walmart is solving the challenges in logistics and supply network using IBM's blockchain solution. It successfully tried two blockchain test pilots on pork from China and mangoes from America. It lowered the time for tracking assets from 7 days to 2.2 seconds and increased transparency across the supply chain. RFID tags and camera are used to monitor pigs from pens and sensors are used to environment condition. Anomaly can be detected, and corrective action can be

done accordingly. These techniques help in increasing profit as risk exposure to customer is avoided and leads to increase in sales. Supermarkets connect their Point of Sale (POS) System and Enterprise Resource Planning System to the blockchain. End to End Traceability was made possible in these test pilots (Reshma Kamath, 2018) [4]. There were several case studies of Blockchain noticed in Agri-Food Domain. Tuna Tracking and Certification by Provenance, Olive Oil Tracking by Ambrosus, Celeia Dairy by OriginTrail, Pork Meat Traceability by TE-Food, FoodCoin and Wine Blockchain by EZLab. Ambrosus architecture is based on Amber. It is a token through which food products are getting tracked and sensor data is recorded and handled through Ethereum Smart Contract. The sensor detection happens by RFID detection and bio tracers inside the packaging (H Ziong, 2020) [5]. Research was conducted in Malaysia to design and test traceable supply chain management system for pepper. The blockchain system was designed to encompass various actors like Farmers, Processors, Distributors, Retailers, Customers, and it was code-named as Prochain. The research recommends that the permissioned blockchain is most effective for supply chain management use case compared to permissionless blockchain network. Prochain was designed in both Fabric and Sawtooth. (KY Chan, 2019) [6]. SmartAgriChain emphasizes that a greater number of IoT sensors could be connected to Hyperledger Fabric due to its volume capabilities. It also states that Fabric could handle transactions in a speed of more than 1000 Transactions per Second (TPS) (Rocha T, 2021) [7].

It is suggested that ecosystem comprising social network, marketplace and cryptocurrency can be created for farmers to bring up the interactivity towards blockchain systems. The ecosystem would ease governance and ensure for the reach of subsidy to farmers (M Kumarathunga, 2020) [8]. A paper on Supply

Chain Management throws light on Public Distribution System (PDS). It is conveyed that smooth functioning of PDS ensures supplies reach marginalized sector, but accountability is a major concern of PDS. Identity Verification, Logistics Optimization and Asset Tracking is made possible using Industry 4.0 Technologies like IoT, Big Data, Blockchain (V Pillai, 2020) [9]. Blockchain implementation of solution for PDS system is feasible using on-chain and off-chain trusted data (SK Singh, 2020) [10]. There are various blockages involved in PDS like leakages, diversions, bogus ration cards. Bogus ration cards and Identity Verification can be overcome by off-chain data solutions like UIDAI AADHAR. RFID tags are inexpensive and easy to detect so that it could be used for tracking essential supplies like rice gunny bags (H Mishra, 2021) [11].

Author	Title	Source	Findings
Satoshi Nakamoto (2008)	Bitcoin: A Peer-to-Peer Electronic Cash System	www.bitcoin.org	In Bitcoin, Blockchain is used to store the transactions in various public peers by reaching consensus through proof of work where every peer competes each other to append block of transactions to the ledger
Melanie Swan (2015)	Blockchain Blueprint for a New Economy	O'Reilly Media	Blockchain acts as a trustful system between various parties which helps in various use cases for eliminating redundancy and unnecessary intermediaries.
GUIDO PERBOLI, STEFANO MUSSO, MARIANGELA ROSANO (2018)	Blockchain in Logistics and Supply Chain: A Lean Approach for Designing Real-	IEEE	GUEST (GO, UNIFORM, EVALUATE, SOLVE AND TEST) methodology can be applied to formulate

	World Use Cases		blockchain designs for specific Supply Chain Management and Logistics use case
Reshma Kamath (2018)	Food Traceability on Blockchain: Walmart's Pork and Mango Pilots with IBM	The JBBA	Walmart is solving the challenges in logistics and supply network using IBM's blockchain solution. It successfully tried two blockchain test pilots on pork from China and mangoes from America.
Giorgio Alessandro Motta, Bedir Tekinerdogan and Ioannis N. Athanasiadis (2020)	Blockchain Applications in the Agri-Food Domain: The First Wave	<a href="http://www.frontiersin.org">www.frontiersin.org</a>	Tuna Tracking and Certification by Provenance, Olive Oil Tracking by Ambrosus, Celeia Dairy by OriginTrail, Pork Meat Traceability by TE-Food, FoodCoin and Wine Blockchain by EZLab. Ambrosus

			architecture is based on Amber.
Kok Yong Chan,Johari Abdullah,Adnan Shahid Khan (2019)	A Framework for Traceable and Transparent Supply Chain Management for Agri-food Sector in Malaysia using Blockchain Technology	IJACSA	Research was conducted in Malaysia to design and test traceable supply chain management system for pepper. The blockchain system was designed to encompass various actors like Farmers, Processors, Distributors, Retailers, Customers, and it was code- named as Prochain.

**Table 2.1 Literature Survey on Use cases of Blockchain**

Author	Title	Source	Findings
Rocha T, Costa P, Sousa V, Coelho P, Sousa F, Cardoso N (2021)	SmartAgriChain: A Blockchain Based Solution for Agri food Certification and Supply Chain Management	International Journal of Environment, Agriculture and Biotechnology	SmartAgriChain emphasizes that a greater number of IoT sensors could be connected to Hyperledger Fabric due to its volume capabilities. It also states that Fabric could handle transactions in a speed of more than 1000 Transactions per Second (TPS)
Malni Kumarathunga (2020)	Improving Farmers' Participation in Agri Supply Chains with Blockchain and Smart Contracts	IEEE	It is suggested that ecosystem comprising social network, marketplace and cryptocurrency can be created for farmers to bring up the interactivity towards blockchain systems.
Sini V. Pillai, Vipin H., Anand Mohan, Shruthi Dileep A (2020)	Infusing Blockchain in Supply Chain Operations of a Public Distribution System	Journal of Supply Chain Management Systems	A paper on Supply Chain Management throws light on Public Distribution System (PDS). It is conveyed that smooth



			functioning of PDS ensures supplies reach marginalized sector, but accountability is a major concern of PDS.
Sandeep Kumar Singh, Mamata Jenamani, Diptiman Dasgupta & Suman Das	A conceptual model for Indian public distribution system using consortium blockchain with on-chain and off-chain trusted data	Information Technology for Development	Blockchain implementation of solution for PDS system is feasible using on-chain and off-chain trusted data
Himani Mishra, Prateek Maheshwari	Blockchain in Indian Public Distribution System: a conceptual framework to prevent leakage of the supplies and its enablers and disablers	ResearchGate	There are various blockages involved in PDS like leakages, diversions, bogus ration cards. Bogus ration cards and Identity Verification can be overcome by off-chain data solutions like UIDAI AADHAR.

R Gayatri	SMS based monitoring system for Fair Price Shops in Tamilnadu	Informatics NIC	Monitoring System is developed by NIC TNSC that uses SMS communication to aggregate stock information of PDS supplies like Rice, Wheat, Sugar, Kerosene from Fair Price Shops (FPS) and warehouses within the state.
-----------	---	-----------------	--

**Table 2.1 Literature Survey on Impact of Blockchain in Supply Chain**

### **3. SYSTEM ANALYSIS**

#### **3.1. EXISTING SYSTEM**

Monitoring System is developed by NIC TNSC that uses SMS communication to aggregate stock information of PDS supplies like Rice, Wheat, Sugar, Kerosene from Fair Price Shops (FPS) and warehouses within the state. Lorries and movement vehicle are fitted with GPS. There are more than 30,000 functional fair price shops and 4000+ of those FPS are run by registered cooperative societies. Daily stock availability tracking facility for essential supplies at FPS are developed by NIC TNSC which can handle the information and SMS is sent daily by salesmen from every FPS in a specified form. There are several SMS formats for each use cases. Daily closing stock and stock positions of both FPS and warehouses, inspection records are sent in SMS through predefined format. Officials after periodic or surprise inspections send their investigation reports through SMS. This traditional system validates the SMS and updates it to database. Incorrect code, redundant commodities, incorrect syntax, messages from unauthorized mobile numbers are validated and displayed in a report. This traditional system is built using JSP and Struts framework. SMS gateway of NIC is interfaced with this system and the system is deployed at Tamil Nadu State Data Center. The daily stock reports are available for public transparency in a website [12] (R Gayatri, 2014). In 5G era, GPRS and SMS is still used in today's traditional tracking facility. GSM spoofing (or) misuse of registered mobile number can lead to malfunction of this traditional system. This is a manual process prone to human errors and inaccuracies. Accountability and Trust of the systems involved in the process can be breached.

### **3.2. PROPOSED SYSTEM**

Automation, reliability, and accountability are insisted in the proposed system compared to manual data entry and inconsistent records from traditional existing system. At every warehouse in Tamil Nadu, existing system could be replaced with a novel asset tracking IoT device interfaced with blockchain network. Asset Tracking IoT device is more trustful and reliable than manual entry because supply assets could be monitored carefully, and inconsistent (or) redundant records could be avoided. Hyperledger Fabric V2.4 is used to deploy trustful reliable permissioned blockchain network through which multiple stakeholder organizations involved can make transactions. Central Government's FCI, Ministry of Consumer Affairs, Food and Public Distribution can collaborate with several food distribution organizations of states across India. Blockchain's use case for Asset Tracking ensures that each supply asset's transaction history is stored safely. This would ensure the end-end traceability for PDS supply chain. Supply assets are tagged with RFID tags and their relative information would be stored on-chain. Above mentioned IoT device has a RFID reader to read the tags attached to the supply assets and the information retrieved by the microcontroller can be used to invoke APIs for creating (or) transferring ownership (or) reading the supply assets. These APIs in turn call the smart contracts deployed in blockchain network to modify the current state of asset. Even though, state of assets is changed but the changes are appended to the distributed immutable ledger. Blockchain implementations are found to be effective in Logistics and Supply Chain Network. IoT devices are secure because they are subjected to more validations compared to the existing SMS based system. Daily Stock records of warehouses and FPSs can be computed using trigger based smart contract which would aggregate assets information using conditional values.

### 3.3. FEASIBILITY STUDY

Factors concerned in the feasibility of this project are Electricity, Internet Connection, Cost for setup and maintenance, User experience and reliability of the proposed system in extreme conditions. For the proposed system mentioned above, intermittent electricity will not be a showstopper. IoT Microcontroller ESP8266 can function in battery mode as the required wattage is much less compared to other alternatives. ESP8266 have several power modes like Deep Sleep mode through which power consumption efficiency is attained. Renewable Energy Sources can be tapped in no (or) power regions. Internet access is getting ubiquitous in current scenario. Optical Fiber Cable plan for rural regions might be a catalyst and accelerator for our project. ESP8266 (or) another alternate Microcontroller can handle API calls in less amount of network bandwidth. It could also be coupled with GSM module in case of rural regions for accessing mobile network to communicate with Application Gateway. Cost for setup and maintenance is comparatively higher than traditional system. But these trustful systems could eradicate corruption in turn increase the profitability of the government organizations involved. This profit may easily engulf the increase of cost in setup and maintenance of proposed system. Proposed System is comparatively highly reliable and scalable than existing system. This ensures that the proposed system runs with very less down time and may handle various cases of validation. Offline storage for IoT devices can be a remedy during extreme environment conditions where internet connectivity may break. User experience is much easier due to the simple forms and learning curve for staffs and personnel involved in the supply chain process is low.

### **3.4. HARDWARE ENVIRONMENT**

IoT device comprises of Wi-Fi Module, RFID reader module and a microcontroller which gets the asset data from the RFID reader by serial communication and a PC/Mobile Phone can connect to it using IP Address of the node after it is connected to the Wi-Fi hotspot. PC/Mobile can connect to the microcontroller and set modes for creating (or) transferring using simple form which is hosted inside microcontroller. Wi-Fi module of IoT device connects to the nearest hotspot and IP address is provided to it using DHCP protocol. This IP address can be associated with domain name using multicast DNS. RFID reader reads the tag belonging to the asset and use it for transaction purposes. IoT device contacts with application gateway for invoking smart contracts and updating ledger in blockchain network.

ESP8266 is a low-cost micro controller with inbuilt Wi-Fi transceiver module. ESP8266 are better than Arduino because it is less power consuming and effective in performing specific tasks. There are various providers and models for ESP8266. We are distinctively using NodeMCU ESP8266 model in our project. For the purpose, ESP8266 is immensely useful because it can get the RFID information and call Chain code API to add the assets in the blockchain because ESP8266 is connected to Wi-Fi, and it can handle asynchronous requests. It has 17 GPIO pins but 11 only could be used for the project. As other pins are involved for internal activities of the microcontroller. Program is written to communicate with RC522 to get the RFID tag information. RC522 is interfaced with help of SPI communication. There are four pins involved in this Serial Peripheral Interfacing. They are Master Out Slave In (MOSI), Master In

Slave Out (MISO), Serial Clock (SCK), Signal Input. There are three other pins used for connecting to Power Supply (VCC, GROUND, RESET).

RC522 is a low-cost RFID reader which reads RFID Tags with the help of 13.56Mhz EM waves. There are two types of tags in use. They are Passive RFID and Active RFID. Passive RFID doesn't require power source and they are inexpensive. Due to those reasons, it can be used for mass scale real-time projects. Passive RFID Communication works by two steps. The two steps are energizing passive tag and reading information from those tags using back scattering. For the purpose above-mentioned, RFID tags are used for tracking edible and non-edible food supplies provided through PDS. RC522 is interfaced with ESP8266 using serial peripheral interface. There are 8 pins in RC522. 4 pins are provided specifically for Serial Communication.

### **3.5. SOFTWARE ENVIRONMENT**

Hyperledger Fabric is a permission blockchain framework through which multiple parties can collaborate and transact with the help of distributed immutable ledger. This framework is used to form a distributed network of peers across multiple organizations and these peers will be running smart contract and maintaining the blockchain. Fabric Blockchain is deployed in every peer, and it is composed of World State and Ledger. Ledger is the immutable history of transactions and World State is a key based database which will store the current state for every asset mentioned in the ledger. Trust and Consensus between several peers is reached using Crash Fault Tolerance Protocol (CFT). Ordering Service is an integral part of CFT protocol which will be validating the block of transactions before getting appended into the ledger.

Execute-Order-Validate architecture is used in Fabric framework instead of Order-Execute. Transaction proposal is sent from application to peers to get validated so that peers execute and sends endorsed transaction response. Endorsed transactions collected from the peers are send to ordering service by application. Ordering service node validates the blocks and sends to peers for validation and appending block to the ledger.

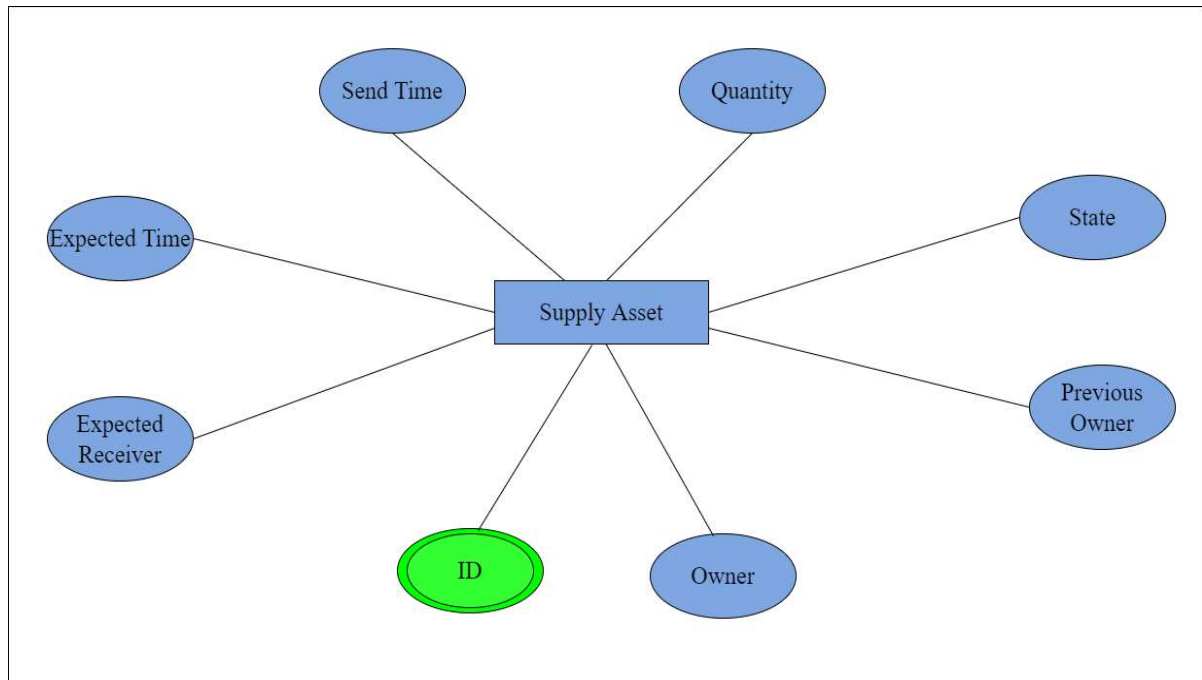
Hyperledger Fabric V2.4 is installed and deployed in Amazon EC2 Instance. Fabric Installation is done by following the documentation and prerequisites are installed properly. Hyperledger Fabric is installed in the peer as docker images, and those images run in docker containers. These containers are responsible for running Ordering Service, Certificate Authorities and Peer in EC2 instance.



Fabric-Samples repository is cloned which contains scripts and example programs to run sample blockchain network. 'network.sh' script in Test-Network folder is executed to run peers and create certificates through Certificate Authority (CA). Smart Contracts are compiled and packaged as a chain code, installed in peers of the blockchain network after they are approved by minimum number of endorsing peers from the multiple organizations. 'network.sh' is also executed to compile and deploy chain code into the peers. This makes blockchain network to get ready for testing and running transactions.

## 4. SYSTEM DESIGN

### 4.1. ER DIAGRAM



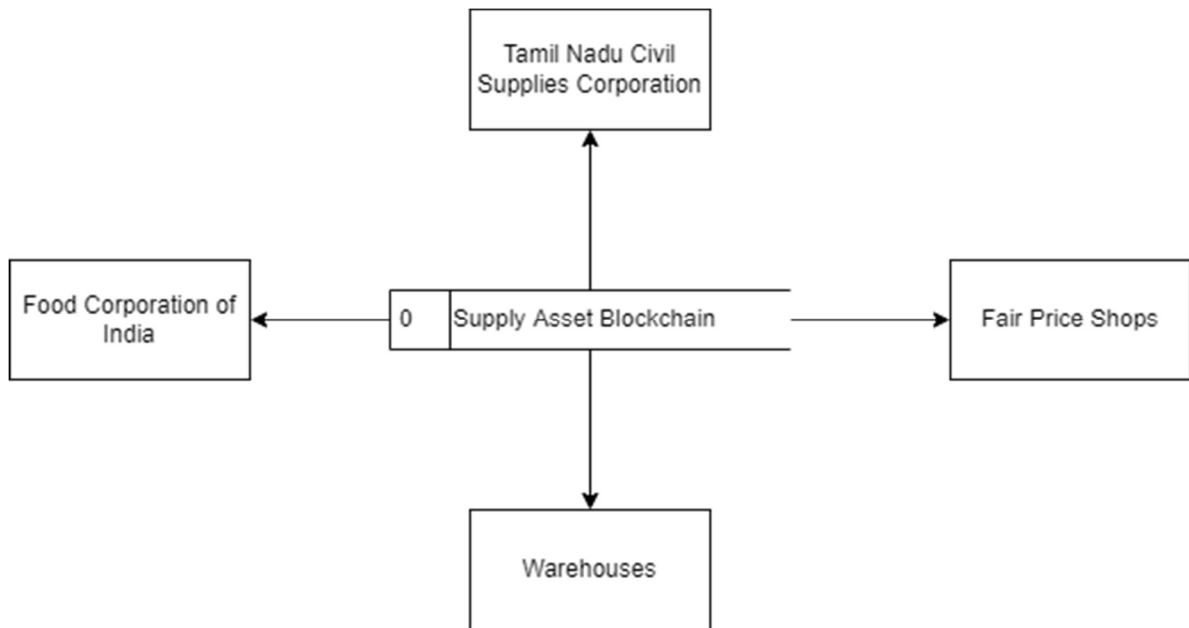
**Fig 4.1 ER Diagram for Supply Asset**

Supply Asset is the asset type to be stored in the block chain. Even though, Block chain is not like conventional database. Assets could be compared with the concept of Table.

Properties of the Supply Asset are considered as columns of the table. We could uniquely identify an asset using it's ID so it can be considered as Primary Key.

Columns of the table are ID (String), Owner (String), Previous Owners (Array of Strings), Expected Reciever (String), Expected Time (Timestamp), Send Time (Timestamp), State (String), Quantity (Number)

## 4.2. DFD DIAGRAM



**Fig. 4.2 DFD Diagram for Blockchain**

Above mentioned diagram is a Level 0 DFD Diagram. The various key players like TNCSC, FCI, FPS, Warehouses are mentioned as they are dependent to the data store system.

Supply Asset Blockchain is treated as a core datastore system in this diagram. Multiple Organization illustrated above collaborate with each other and they contribute to this blockchain.

Level 0 DFD diagram can be used for simple understanding of the organizations involved and how the datastore works.

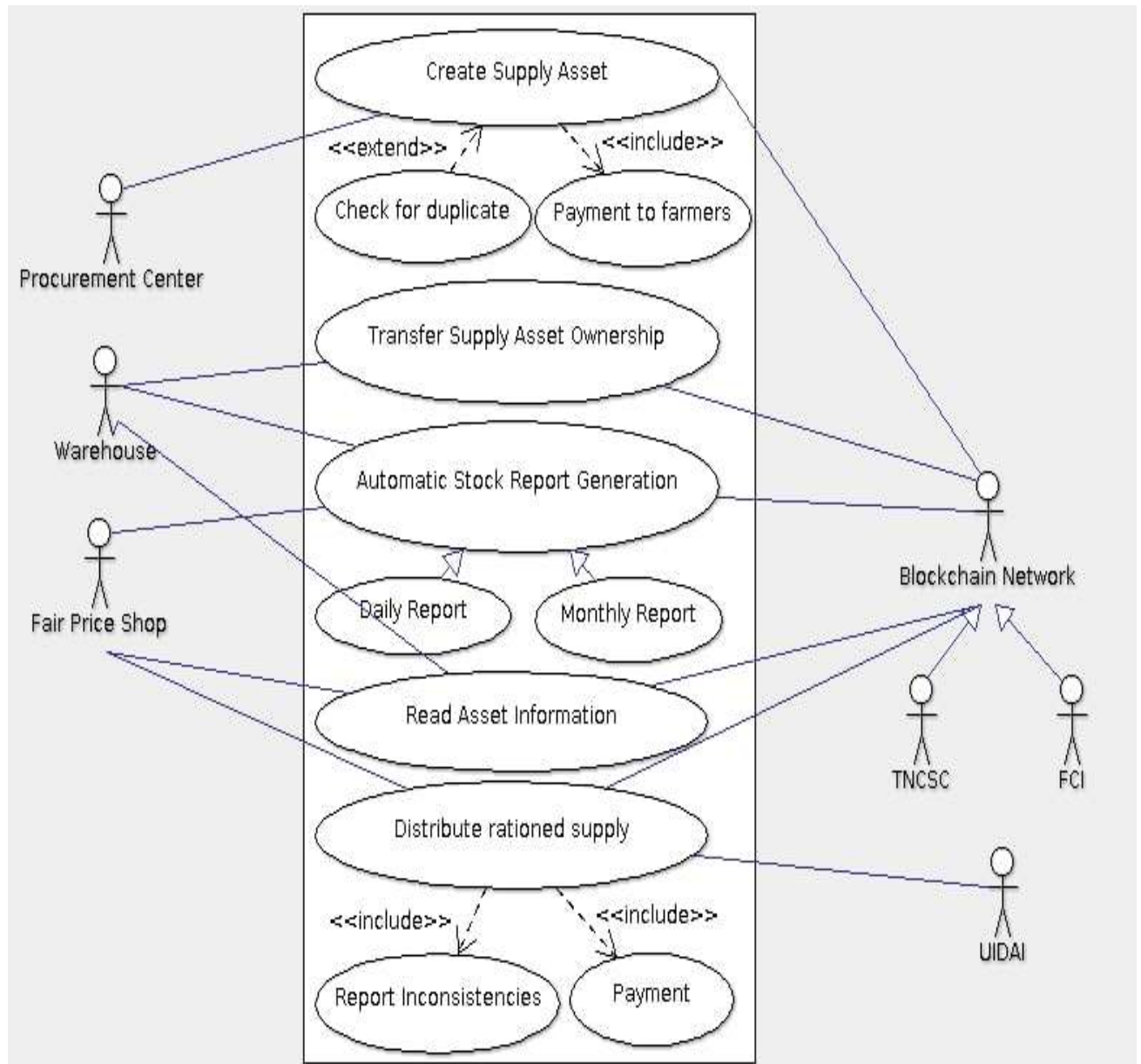
### 4.3. DATA DICTIONARY DIAGRAM

Field Name	Data Type	Field Length	Constraint	Description
ID	String	30	Primary Key	Autogenerated ID for Supply Asset
Owner	String	30	Not Null	Name of Owner
Previous Owners	Array of Strings		Not Null	Array of Previous Owner Names
Expected Receiver	String	30	Not Null	Name of transferred owner
Expected Time	Timestamp		Not Null	ETA
Send Time	Timestamp		Not Null	Time on which supply asset sent
State	String	30	Not Null	Detail of the state. Current Situation (Hulling, processed etc.)
Quantity	Number	10	Not Null	Quantity of the Supply Asset

**Fig 4.3 Data Dictionary Diagram for Supply Asset**

## 4.4. UML DIAGRAMS

### 4.4.1. USE CASE DIAGRAM



**Fig. 4.4 Use Case Diagram for PDS Supply**

4.4.2. SEQUENCE DIAGRAM

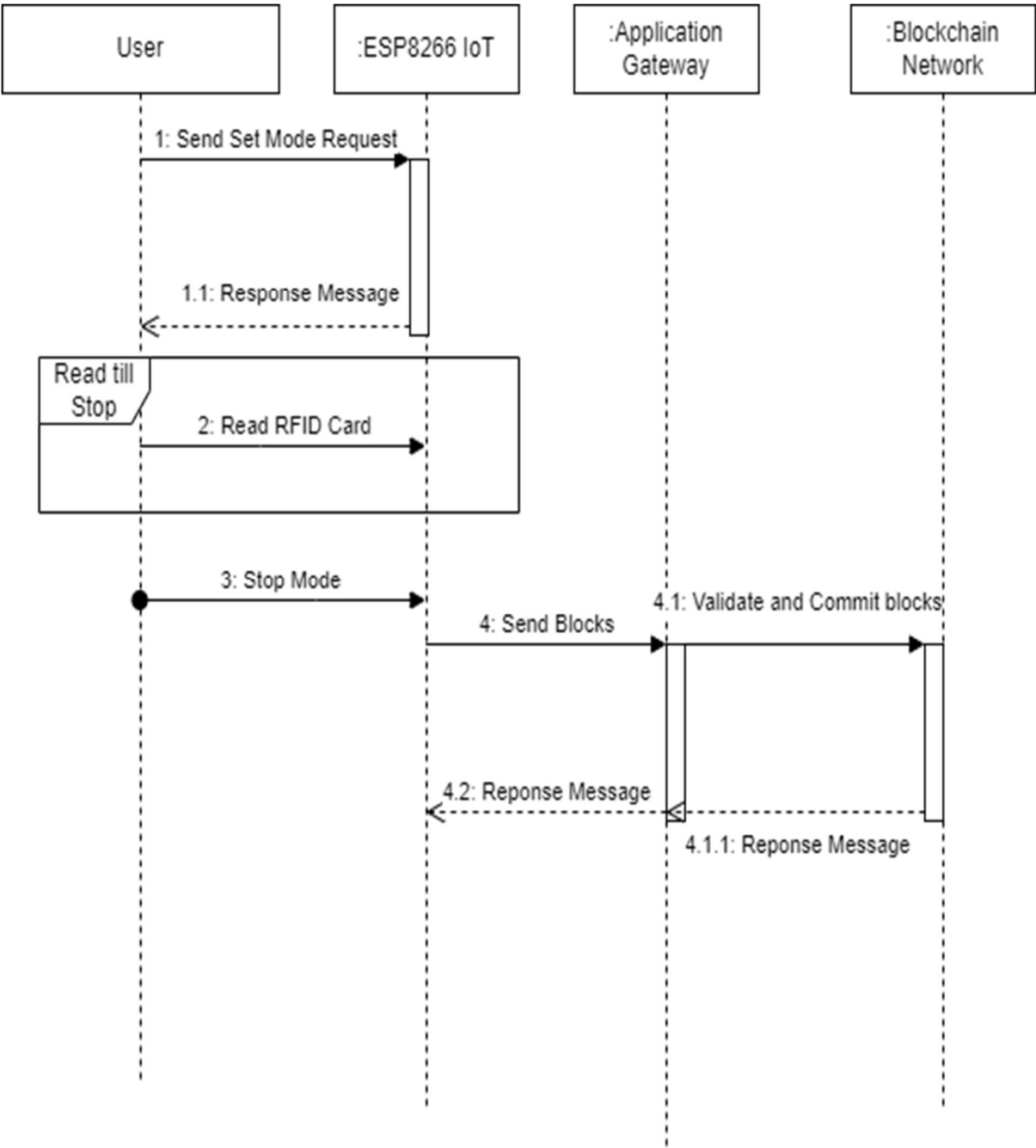


Fig. 4.5 Sequence Diagram for Create Asset

#### 4.4.3. ACTIVITY DIAGRAM

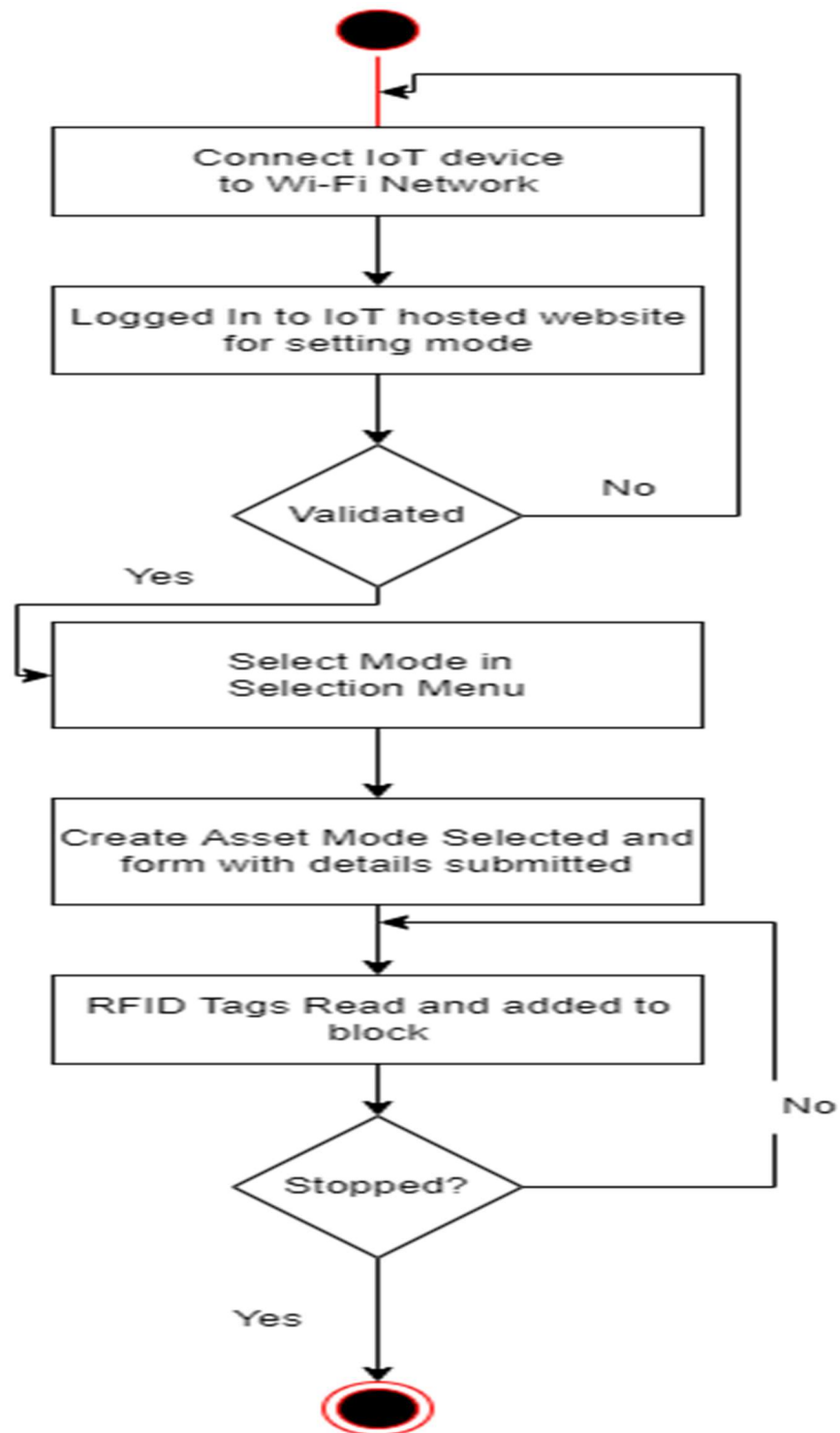


Fig. 4.6 Activity Diagram for Create Asset

4.4.4 PACKAGE DIAGRAM

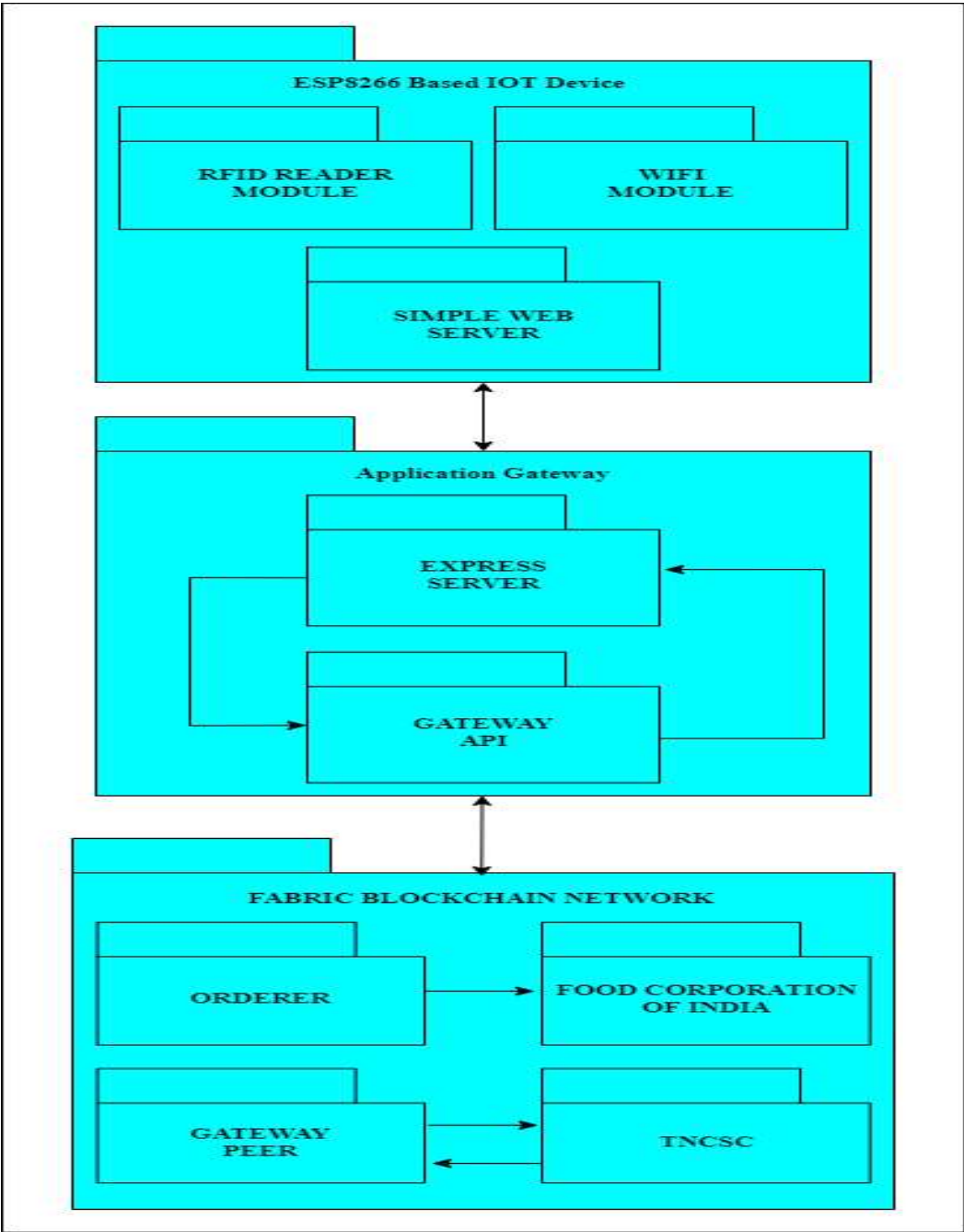
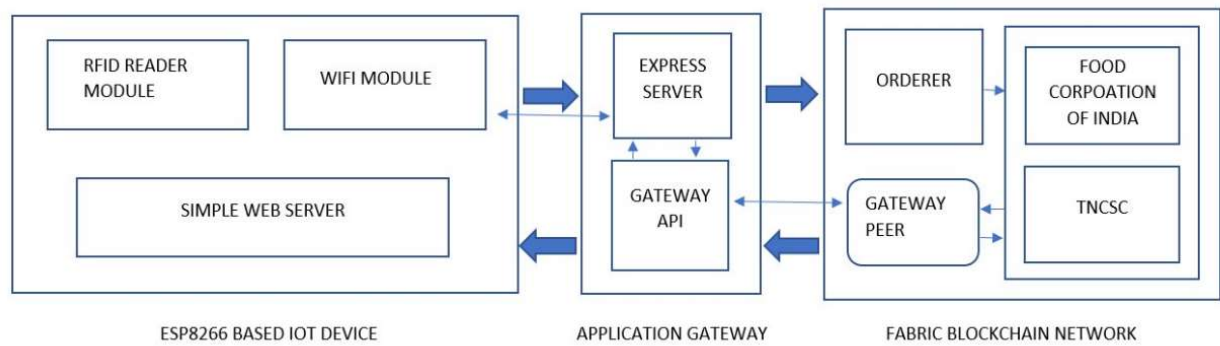


Fig 4.7 Package Diagram for PDS Supply



## 5. SYSTEM ARCHITECTURE

### 5.1. MODULE DESIGN SPECIFICATION



**Fig 5.1 Overall Modules Illustration on System**

Three Important Modules are discussed below with the explanation of sub modules confined within those main modules.

#### A. ESP8266 BASED IOT DEVICE

ESP8266 is a low-cost micro controller with inbuilt Wi-Fi transceiver module. ESP8266 are better than Arduino because it is less power consuming and effective in performing specific tasks. There are various providers and models for ESP8266. We are distinctively using NodeMCU ESP8266 model in our project. For the purpose, ESP8266 is immensely useful because it can get the RFID information and call Chain code API to add the assets in the blockchain because ESP8266 is connected to Wi-Fi, and it can handle asynchronous requests. It has 17 GPIO pins but 11 only could be used for the project. As other pins are involved for internal activities of the microcontroller. Program is written to communicate with RC522 to get the RFID tag information. RC522 is

interfaced with help of SPI communication. There are four pins involved in this Serial Peripheral Interfacing. They are Master Out Slave In (MOSI), Master In Slave Out (MISO), Serial Clock (SCK), Signal Input. There are three other pins used for connecting to Power Supply (VCC, GROUND, RESET).

RC522 is a low-cost RFID reader which reads RFID Tags with the help of 13.56Mhz EM waves. There are two types of tags in use. They are Passive RFID and Active RFID. Passive RFID doesn't require power source and they are inexpensive. Due to those reasons, it can be used for mass scale real-time projects. Passive RFID Communication works by two steps. The two steps are energizing passive tag and reading information from those tags using back scattering. For the purpose above-mentioned, RFID tags are used for tracking edible and non-edible food supplies provided through PDS. RC522 is interfaced with ESP8266 using serial peripheral interface. There are 8 pins in RC522. 4 pins are provided specifically for Serial Communication.

A simple web server should be developed and deployed on IoT device. This server renders a simple form, and the form is used to find the mode of operation and collect other details regarding the asset from user. This simple responsive form could be rendered in both PC and Mobile. There are three modes of operation: Create Asset, Transfer Ownership of Asset, Read Asset. Create Asset requires data like weight and type of asset. Transfer Ownership Mode is used to change ownership from sender to receiver. Read Asset at receiver's point to check whether the received supplies are as mentioned after transfer. Once a mode is started in a device, then it cannot be changed in between. Mode should be stopped by clicking stop button in the form.

## **B. APPLICATION GATEWAY**

Application Gateway is required to make a way for IoT devices to connect with Blockchain Framework SDK. It is designed in a form of REST API endpoints, and this makes sure that it is stateless and highly scalable as it is of microservice architecture. It is important for authorizing and validating the requests from IoT devices. It calls functions from Fabric SDK using Google Remote Procedure Calls (gRPC) to invoke smart contract as required for the given context. This acts like a barrier because IoT doesn't directly interact with Fabric SDK, and this ensures that malicious requests are avoided before reaching Fabric.

Application Gateway is written in TypeScript and built on Express Framework. Whenever the respective REST API endpoints are called by IoT device, then the corresponding service function will be called. Fabric SDKs are used to invoke smart contract, and which will in turn read (or) write ledger. Express Server and Routes are mentioned in 'index.ts' file and service functions are mentioned in 'app.ts'. These two files are under same folder. Dependencies are mentioned in 'package.js' file and Node dependency modules are installed using 'npm install'. Application gateway server runs on 8080 port so port forwarding is done to get requests from 80 port. 'npm run serve' command is executed to compile and run the server in EC2 instance. Gateway API contacts with the Fabric framework and used to invoke the smart contracts.

## **C. FABRIC BLOCKCHAIN NETWORK**

Hyperledger Fabric is a permission blockchain framework through which multiple parties can collaborate and transact with the help of distributed immutable ledger. This framework is used to form a distributed network of peers across multiple organizations and these peers will be running smart contract and maintaining the blockchain. Fabric Blockchain is deployed in every peer, and it is composed of World State and Ledger. Ledger is the immutable history of transactions and World State is a key based database which will store the current state for every asset mentioned in the ledger. Trust and Consensus between several peers is reached using Crash Fault Tolerance Protocol (CFT). Ordering Service is an integral part of CFT protocol which will be validating the block of transactions before getting appended into the ledger.

Execute-Order-Validate architecture is used in Fabric framework instead of Order-Execute. Transaction proposal is sent from application to peers to get validated so that peers execute and sends endorsed transaction response. Endorsed transactions collected from the peers are send to ordering service by application. Ordering service node validates the blocks and sends to peers for validation and appending block to the ledger.

Hyperledger Fabric V2.4 is installed and deployed in Amazon EC2 Instance. Fabric Installation is done by following the documentation and prerequisites are installed properly. Hyperledger Fabric is installed in the peer as docker images, and those images run in docker containers. These containers are responsible for running Ordering Service, Certificate Authorities and Peer in EC2 instance. Fabric-Samples repository is cloned which contains scripts and example

programs to run sample blockchain network. 'network.sh' script in Test-Network folder is executed to run peers and create certificates through Certificate Authority (CA). Smart Contracts are compiled and packaged as a chain code, installed in peers of the blockchain network after they are approved by minimum number of endorsing peers from the multiple organizations. 'network.sh' is also executed to compile and deploy chain code into the peers. This makes blockchain network to get ready for testing and running transactions.

Associated with every chaincode is an endorsement policy that applies to all of the smart contracts defined within it. An endorsement policy is very important; it indicates which organizations in a blockchain network must sign a transaction generated by a given smart contract for that transaction to be declared valid. Every smart contract has an endorsement policy associated with it. This endorsement policy identifies which organizations must approve transactions generated by the smart contract before those transactions can be identified as valid. All transactions, whether valid or invalid are added to a distributed ledger, but only valid transactions update the world state. If an endorsement policy specifies that more than one organization must sign a transaction, then the smart contract must be executed by a sufficient set of organizations for a valid transaction to be generated. In the example above, a smart contract transaction to transfer a car would need to be executed and signed by both ORG1 and ORG2 for it to be valid.

## **5.2. ALGORITHMS**

### **5.2.1. CREATE ASSET MODE**

#### **STEP 1:**

IoT device should be connected to Wi-Fi Network by providing the network credentials. Mobile/PC should also be connected to the same Wi-Fi network.

#### **STEP 2:**

Website hosted in the IoT device can be accessed by Mobile/PC. This website can be used to set and start/stop various modes provided.

#### **STEP 3:**

Create Asset Mode is selected in Mode Selection Menu.

#### **STEP 4:**

Details for the mode needs to be submitted in the redirected form.

#### **STEP 5:**

RFID tags would be read till STOP mode is pressed from the website.

#### **STEP 6:**

After STOP Mode, blocks of information are sent to Application gateway for those information to be validated and committed to the blockchain.

## **5.2.2 TRANSFER SUPPLY ASSET OWNER MODE**

### **STEP 1:**

IoT device should be connected to Wi-Fi Network by providing the network credentials. Mobile/PC should also be connected to the same Wi-Fi network.

### **STEP 2:**

Website hosted in the IoT device can be accessed by Mobile/PC. This website can be used to set and start/stop various modes provided.

### **STEP 3:**

Create Asset Mode is selected in Mode Selection Menu

### **STEP 4:**

Details for the mode needs to be submitted in the redirected form.

### **STEP 5:**

RFID tags would be read till STOP mode is pressed from the website.

### **STEP 6:**

After STOP Mode, blocks of information are sent to Application gateway for those information to be validated and committed to the blockchain.

## 6. SYSTEM IMPLEMENTATION

### 6.1. CLIENT-SIDE CODING

#### FILE: ARDUINO/HYPERLEDGERIOT.INO

```
#include <ESPAsyncTCP.h>
#include <ESPAsyncWebServer.h>
#include <ESP8266WiFi.h>
#include <ESP8266HTTPClient.h>
#include <WiFiClient.h>
#include <SPI.h>
#include <string.h>
#include "MFRC522.h"

/* wiring the MFRC522 to ESP8266 (ESP-12)
  RST    = GPIO5
  SDA(SS) = GPIO4
  MOSI   = GPIO13
  MISO   = GPIO12
  SCK    = GPIO14
  GND    = GND
  3.3V   = 3.3V
*/

#define RST_PIN 5
#define SS_PIN 4

//Wifi Credentials
```



```

const char *ssid = "AndroidAP_9346";
const char *pass = "helloworld";

//RFID Variables
MFRC522 rf(SS_PIN, RST_PIN); // Instance of MFRC522
byte nuidPICC[4];
String id = "";

//Server Variables
AsyncWebServer server(80);
char *state = "STOPPED"; // STARTED OR STOPPED
char *type = ""; // CREATE OR TRANSFER
String owner = "";
String sizes = "";
String types = "";

//
const char *createUrl = "http://13.233.57.179/create";
const char *transferURL = "http://13.233.57.179/transfer";

void notFound(AsyncWebServerRequest *request) {
    request->send(404, "text/plain", "Not found");
}

void idHex(byte *buffer, byte bufferSize) {
    for (byte i = 0; i < bufferSize; i++) {
        id += buffer[i] < 0x10 ? " 0" : " ";
    }
}

```

```

    id += String(buffer[i]);
}
}

void setup() {
  //Initial Logs-1
  Serial.begin(115200);
  delay(250);
  Serial.println(F("Booting...."));

  // Connecting with RFID Reader
  SPI.begin();
  rf.PCD_Init();

  // Connecting to WiFi Hotspot
  WiFi.mode(WIFI_STA);
  WiFi.begin(ssid, pass);
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
  if (WiFi.status() == WL_CONNECTED) {
    Serial.println(F("WiFi connected"));
    Serial.println(WiFi.localIP());
  }

  // Server Initialization with EndPoints

```

```

server.on("/create", HTTP_GET, [](AsyncWebServerRequest * request) {
    if (strcmp(state, "STOPPED") == 0) {
        if (request->hasParam("owner")) {
            owner = request->getParam("owner")->value();
        }
        if (request->hasParam("types")) {
            types = request->getParam("types")->value();
        }
        if (request->hasParam("size")) {
            sizes = request->getParam("size")->value();
        }
        type = "CREATE";
        state = "STARTED";
        Serial.print(state );
        Serial.print(type );
        Serial.print(types );
        Serial.print(sizes );
        Serial.print(owner );
        Serial.println();
        request->send(200, "text/plain", "CREATE CONFIG PREPARED");
    }
});

server.on("/transfer", HTTP_GET, [](AsyncWebServerRequest * request) {
    if (strcmp(state, "STOPPED") == 0) {
        if (request->hasParam("owner")) {
            owner = request->getParam("owner")->value();
        }
    }
}

```

```

    type = "TRANSFER";
    state = "STARTED";
    Serial.print(state );
    Serial.print(type );
    Serial.print(owner );
    Serial.println();
    request->send(200, "text/plain", "TRANSFER CONFIG PREPARED");
}
});
server.on("/stop", HTTP_GET, [](AsyncWebServerRequest * request) {
    if (strcmp(state, "STARTED") == 0) {
        state = "STOPPED";
    }
    request->send(200, "text/plain", "STOPPED");
});
server.onNotFound(notFound);
server.begin();

// Initial Logs-2
Serial.println(F("Ready!"));

Serial.println(F("=====
====="));

    Serial.println(F("Scan for Card and print UID:"));
}

void loop() {

```

```

//Check if Scanning is started from Form
if (strcmp(state, "STARTED") == 0) {
    //Check if card is in reciever
    if ( ! rf.PICC_IsNewCardPresent()) {
        delay(50);
        return;
    }
    if ( ! rf.PICC_ReadCardSerial()) {
        delay(50);
        return;
    }
    //Check whether card is same as previous or not
    if (rf.uid.uidByte[0] != nuidPICC[0] ||
        rf.uid.uidByte[1] != nuidPICC[1] ||
        rf.uid.uidByte[2] != nuidPICC[2] ||
        rf.uid.uidByte[3] != nuidPICC[3] ) {
        Serial.println(F("A new card has been detected."));
        // Store NUID into nuidPICC array
        for (byte i = 0; i < 4; i++) {
            if (i == 0) id = "";
            nuidPICC[i] = rf.uid.uidByte[i];
        }
        Serial.print(F("Card UID:"));
        idHex(rf.uid.uidByte,rf.uid.size);
        Serial.print(id);
        Serial.println();
        //Calling Application Gateway APIs

```

```

WiFiClient client;
HTTPClient http;
if (strcmp(type, "CREATE") == 0) {
    http.begin(client,createURL);
    http.addHeader("Content-Type", "application/json");
    String httpRequestData = "{\"id\":\"" + id + "\",\"type\":\"" + types +
    "\",\"owner\":\"" + owner + "\",\"size\":\"" + sizes + "\"}";
    int httpRequestCode = http.POST(httpRequestData);
    Serial.println(httpRequestCode);
    Serial.print(F("Created Asset"));
    http.end();
}
else if (strcmp(type, "TRANSFER") == 0) {
    http.begin(client,transferURL);
    http.addHeader("Content-Type", "application/json");
    String httpRequestData = "{\"id\":\"" + id + "\",\"owner\":\"" + owner +
    "\"}";
    int httpRequestCode = http.POST(httpRequestData);
    Serial.println(httpRequestCode);
    Serial.print(F("Asset Transferred"));
    http.end();
}
Serial.println();
}
}
}

```

## 6.2. SERVER-SIDE CODING

### FILE: APPLICATION\_GATEWAY/SRC/INDEX.TS

```
import { Contract } from '@hyperledger/fabric-gateway';
import { Request, Response } from 'express';
import { main, getAllAssets, initLedger, createAsset, transferAssetAsync,
readAssetByID } from './app';
const express = require('express');
const app = express();
const port = 80;
let contract:Contract;
app.listen(port, async () => {
    contract = await main();
    await initLedger(contract);
});
app.get('/get/:id?', async (req:Request,res:Response) => {
    const { id } = req.params;
    let result:JSON;
    if(id===undefined){
        result = await getAllAssets(contract);
    }
    else {
        result = await readAssetByID(contract, String(id));
    }
    res.json(result);
});
```

```
});  
app.post('/create', async (req:Request,res:Response) => {  
  const {id, type, size, owner} = req.body;  
  await createAsset(contract, id, type, size, owner);  
  res.send('Transaction is successfully committed');  
});  
app.post('/transfer', async (req:Request,res:Response) => {  
  const {id, owner} = req.body;  
  await transferAssetAsync(contract, id, owner);  
  res.send('Owner Transferred');  
});
```



## FILE: APPLICATION\_GATEWAY/SRC/APP.TS

```
import * as grpc from '@grpc/grpc-js';
import { connect, Contract, Identity, Signer, signers } from
'@hyperledger/fabric-gateway';
import * as crypto from 'crypto';
import { promises as fs } from 'fs';
import * as path from 'path';
import { TextDecoder } from 'util';

const channelName = envOrDefault('CHANNEL_NAME', 'mychannel');
const chaincodeName = envOrDefault('CHAINCODE_NAME', 'basic');
const mspId = envOrDefault('MSP_ID', 'Org1MSP');

// Path to crypto materials.
const cryptoPath = envOrDefault('CRYPTO_PATH',
path.resolve(__dirname, '..', '..', 'test-network', 'organizations',
'peerOrganizations', 'org1.example.com'));

// Path to user private key directory.
const keyDirectoryPath = envOrDefault('KEY_DIRECTORY_PATH',
path.resolve(cryptoPath, 'users', 'User1@org1.example.com', 'msp',
'keystore'));

// Path to user certificate.
const certPath = envOrDefault('CERT_PATH', path.resolve(cryptoPath,
'users', 'User1@org1.example.com', 'msp', 'signcerts', 'cert.pem'));
```

```

// Path to peer tls certificate.
const      tlsCertPath      =      envOrDefault('TLS_CERT_PATH',
path.resolve(cryptoPath, 'peers', 'peer0.org1.example.com', 'tls', 'ca.crt'));

// Gateway peer endpoint.
const peerEndpoint = envOrDefault('PEER_ENDPOINT', 'localhost:7051');

// Gateway peer SSL host name override.
const      peerHostAlias     =      envOrDefault('PEER_HOST_ALIAS',
'peer0.org1.example.com');

const utf8Decoder = new TextDecoder();

async function main(): Promise<Contract> {

    await displayInputParameters();

    // The gRPC client connection should be shared by all Gateway
connections to this endpoint.
    const client = await newGrpcConnection();

    const gateway = connect({
        client,
        identity: await newIdentity(),
        signer: await newSigner(),
        // Default timeouts for different gRPC calls

```

```

    evaluateOptions: () => {
        return { deadline: Date.now() + 5000 }; // 5 seconds
    },
    endorseOptions: () => {
        return { deadline: Date.now() + 15000 }; // 15 seconds
    },
    submitOptions: () => {
        return { deadline: Date.now() + 5000 }; // 5 seconds
    },
    commitStatusOptions: () => {
        return { deadline: Date.now() + 60000 }; // 1 minute
    },
});

try {
    // Get a network instance representing the channel where the smart
contract is deployed.
    const network = gateway.getNetwork(channelName);

    // Get the smart contract from the network.
    const contract = network.getContract(chaincodeName);

    // return contract object to express controllers
    return contract;

} finally {
    console.log('Contract returned successfully');
}

```

```

    }
}

```

```

main().catch(error => {
    console.error('***** FAILED to run the application:', error);
    process.exitCode = 1;
});

```

```

async function newGrpcConnection(): Promise<grpc.Client> {
    const tlsRootCert = await fs.readFile(tlsCertPath);
    const tlsCredentials = grpc.credentials.createSsl(tlsRootCert);
    return new grpc.Client(peerEndpoint, tlsCredentials, {
        'grpc.ssl_target_name_override': peerHostAlias,
    });
}

```

```

async function newIdentity(): Promise<Identity> {
    const credentials = await fs.readFile(certPath);
    return { mspId, credentials };
}

```

```

async function newSigner(): Promise<Signer> {
    const files = await fs.readdir(keyDirectoryPath);
    const keyPath = path.resolve(keyDirectoryPath, files[0]);
    const privateKeyPem = await fs.readFile(keyPath);
    const privateKey = crypto.createPrivateKey(privateKeyPem);
    return signers.newPrivateKeySigner(privateKey);
}

```

```
}
```

```
/**
```

\* This type of transaction would typically only be run once by an application the first time it was started after its

\* initial deployment. A new version of the chaincode deployed later would likely not need to run an "init" function.

```
*/
```

```
async function initLedger(contract: Contract): Promise<void> {
```

```
    console.log('\n--> Submit Transaction: InitLedger, function creates the initial set of assets on the ledger');
```

```
    await contract.submitTransaction('InitLedger');
```

```
    console.log('*** Transaction committed successfully');
```

```
}
```

```
/**
```

\* Evaluate a transaction to query ledger state.

```
*/
```

```
async function getAllAssets(contract: Contract): Promise<JSON> {
```

```
    console.log('\n--> Evaluate Transaction: GetAllAssets, function returns all the current assets on the ledger');
```

```
    const resultBytes = await contract.evaluateTransaction('GetAllAssets');
```

```
    const resultJson = utf8Decoder.decode(resultBytes);
```

```

    const result = JSON.parse(resultJson);
    console.log('*** Result:', result);
    return result;
}

/**
 * Submit a transaction synchronously, blocking until it has been committed
 * to the ledger.
 */
async function createAsset(contract: Contract, id: string, type: string, size:
string, owner: string) : Promise<void> {
    console.log('\n--> Submit Transaction: CreateAsset, creates new asset with
ID, Type, Size, Owner arguments');

    await contract.submitTransaction(
        'CreateAsset',
        id,
        type,
        size,
        owner,
    );

    console.log('*** Transaction committed successfully');
}

/**

```

\* Submit transaction asynchronously, allowing the application to process the smart contract response (e.g. update a UI)

\* while waiting for the commit notification.

\*/

```
async function transferAssetAsync(contract: Contract, id: string, owner: string): Promise<void> {
```

```
    console.log('\n--> Async Submit Transaction: TransferAsset, updates existing asset owner');
```

```
    const commit = await contract.submitAsync('TransferAsset', {
        arguments: [id, owner],
    });
```

```
    const oldOwner = utf8Decoder.decode(commit.getResult());
```

```
    console.log(`*** Successfully submitted transaction to transfer ownership from ${oldOwner} to Saptha`);
```

```
    console.log('*** Waiting for transaction commit');
```

```
    const status = await commit.getStatus();
```

```
    if (!status.successful) {
```

```
        throw new Error(`Transaction ${status.transactionId} failed to commit with status code ${status.code}`);
```

```
    }
```

```
    console.log('*** Transaction committed successfully');
```

```
}
```

```

async function readAssetByID(contract: Contract, id:string):
Promise<JSON> {
    console.log('\n--> Evaluate Transaction: ReadAsset, function returns asset
attributes');

    const resultBytes = await contract.evaluateTransaction('ReadAsset', id);

    const resultJson = utf8Decoder.decode(resultBytes);
    const result = JSON.parse(resultJson);
    console.log('*** Result:', result);
    return result;
}

/**
 * envOrDefault() will return the value of an environment variable, or a
default value if the variable is undefined.
 */
function envOrDefault(key: string, defaultValue: string): string {
    return process.env[key] || defaultValue;
}

/**
 * displayInputParameters() will print the global scope parameters used by
the main driver routine.
 */
async function displayInputParameters(): Promise<void> {
    console.log(`channelName:    ${channelName}`);

```



```
console.log(`chaincodeName:  ${chaincodeName}`);  
console.log(`mspId:          ${mspId}`);  
console.log(`cryptoPath:     ${cryptoPath}`);  
console.log(`keyDirectoryPath: ${keyDirectoryPath}`);  
console.log(`certPath:       ${certPath}`);  
console.log(`tlsCertPath:    ${tlsCertPath}`);  
console.log(`peerEndpoint:   ${peerEndpoint}`);  
console.log(`peerHostAlias:  ${peerHostAlias}`);  
}
```

```
export {  
  main,  
  initLedger,  
  getAllAssets,  
  createAsset,  
  transferAssetAsync,  
  readAssetByID,  
}
```

## **FILE: CHAINCODE/SRC/SMARTCONTRACT.GO**

```
package chaincode
```

```
import (
```

```
    "encoding/json"
```

```
    "fmt"
```

```
    "github.com/hyperledger/fabric-contract-api-go/contractapi"
```

```
)
```

```
// SmartContract provides functions for managing an Asset
```

```
type SmartContract struct {
```

```
    contractapi.Contract
```

```
}
```

```
// Asset describes basic details of what makes up a simple asset
```

```
//Insert struct field in alphabetic order => to achieve determinism accross  
languages
```

```
// go lang keeps the order when marshal to json but doesn't order automatically
```

```
type Asset struct {
```

```
    Type      string `json:"Type"`
```

```
    ID        string `json:"ID"`
```

```
    Owner     string `json:"Owner"`
```

```
    Size      int    `json:"Size"`
```

```
}
```

```

// InitLedger adds a base set of assets to the ledger
func (s *SmartContract) InitLedger(ctx
contractapi.TransactionContextInterface) error {
    assets := []Asset{
        {ID: "asset1", Type: "Rice", Size: 5, Owner: "Vellore"},
        {ID: "asset2", Type: "Sugar", Size: 5, Owner: "Ranipet"},
        {ID: "asset3", Type: "Rice", Size: 10, Owner: "Thiruvallur"},
        {ID: "asset4", Type: "Wheat", Size: 10, Owner: "Chennai"},
        {ID: "asset5", Type: "Kerosene", Size: 15, Owner:
"Kanchipuram"},
        {ID: "asset6", Type: "Rice", Size: 15, Owner: "Madurai"},
    }

    for _, asset := range assets {
        assetJSON, err := json.Marshal(asset)
        if err != nil {
            return err
        }

        err = ctx.GetStub().PutState(asset.ID, assetJSON)
        if err != nil {
            return fmt.Errorf("failed to put to world state. %v", err)
        }
    }

    return nil
}

```

```

// CreateAsset issues a new asset to the world state with given details.
func (s *SmartContract) CreateAsset(ctx
contractapi.TransactionContextInterface, id string, types string, size int,
owner string) error {
    exists, err := s.AssetExists(ctx, id)
    if err != nil {
        return err
    }
    if exists {
        return fmt.Errorf("the asset %s already exists", id)
    }

    asset := Asset{
        ID:      id,
        Type:     types,
        Size:     size,
        Owner:    owner,
    }
    assetJSON, err := json.Marshal(asset)
    if err != nil {
        return err
    }

    return ctx.GetStub().PutState(id, assetJSON)
}

```

```

// ReadAsset returns the asset stored in the world state with given id.
func (s *SmartContract) ReadAsset(ctx
contractapi.TransactionContextInterface, id string) (*Asset, error) {
    assetJSON, err := ctx.GetStub().GetState(id)
    if err != nil {
        return nil, fmt.Errorf("failed to read from world state: %v", err)
    }
    if assetJSON == nil {
        return nil, fmt.Errorf("the asset %s does not exist", id)
    }

    var asset Asset
    err = json.Unmarshal(assetJSON, &asset)
    if err != nil {
        return nil, err
    }

    return &asset, nil
}

```

// UpdateAsset updates an existing asset in the world state with provided parameters.

```

func (s *SmartContract) UpdateAsset(ctx
contractapi.TransactionContextInterface, id string, types string, size int,
owner string) error {
    exists, err := s.AssetExists(ctx, id)
    if err != nil {

```

```

        return err
    }
    if !exists {
        return fmt.Errorf("the asset %s does not exist", id)
    }

    // overwriting original asset with new asset
    asset := Asset{
        ID:      id,
        Type:     types,
        Size:     size,
        Owner:    owner,
    }
    assetJSON, err := json.Marshal(asset)
    if err != nil {
        return err
    }

    return ctx.GetStub().PutState(id, assetJSON)
}

// DeleteAsset deletes an given asset from the world state.
func (s *SmartContract) DeleteAsset(ctx
contractapi.TransactionContextInterface, id string) error {
    exists, err := s.AssetExists(ctx, id)
    if err != nil {
        return err
    }

```

```

    }
    if !exists {
        return fmt.Errorf("the asset %s does not exist", id)
    }

    return ctx.GetStub().DelState(id)
}

// AssetExists returns true when asset with given ID exists in world state
func (s *SmartContract) AssetExists(ctx
contractapi.TransactionContextInterface, id string) (bool, error) {
    assetJSON, err := ctx.GetStub().GetState(id)
    if err != nil {
        return false, fmt.Errorf("failed to read from world state: %v", err)
    }

    return assetJSON != nil, nil
}

// TransferAsset updates the owner field of asset with given id in world state,
and returns the old owner.
func (s *SmartContract) TransferAsset(ctx
contractapi.TransactionContextInterface, id string, newOwner string) (string,
error) {
    asset, err := s.ReadAsset(ctx, id)
    if err != nil {
        return "", err
    }

```

```

    }

    oldOwner := asset.Owner
    asset.Owner = newOwner

    assetJSON, err := json.Marshal(asset)
    if err != nil {
        return "", err
    }

    err = ctx.GetStub().PutState(id, assetJSON)
    if err != nil {
        return "", err
    }

    return oldOwner, nil
}

// GetAllAssets returns all assets found in world state
func (s *SmartContract) GetAllAssets(ctx contractapi.TransactionContextInterface) ([]*Asset, error) {
    // range query with empty string for startKey and endKey does an
    // open-ended query of all assets in the chaincode namespace.
    resultsIterator, err := ctx.GetStub().GetStateByRange("", "")
    if err != nil {
        return nil, err
    }
}

```



```

defer resultsIterator.Close()

var assets []*Asset
for resultsIterator.HasNext() {
    queryResponse, err := resultsIterator.Next()
    if err != nil {
        return nil, err
    }

    var asset Asset
    err = json.Unmarshal(queryResponse.Value, &asset)
    if err != nil {
        return nil, err
    }
    assets = append(assets, &asset)
}

return assets, nil
}

```

## **7. SYSTEM TESTING**

### **7.1. UNIT TESTING**

Several units of the system are tested, and the findings are listed below.

#### **Application Gateway Unit Tests Findings:**

1. It can handle duplicate IDs in the batch of scanned RFID tags.
2. It can handle information submitted through request which are not constrained to the types mentioned.
3. Logs and sends proper error messages when system error (or) unavailability of the system occurs.

#### **ESP8266 IoT Device Unit Tests Findings:**

1. It can handle information submitted through request which are not constrained to the types mentioned.
2. Logs and sends proper error messages when system error (or) unavailability of the system occurs.
3. Proper Error Messages are sent when the network is unavailable

#### **Fabric Blockchain Network Unit Tests Findings:**

1. Works as expected when peers of one (or) more organization are compromised.
2. Ordering Service and MSP are tested carefully and checked whether they are vulnerable to security attacks.

## **7.2. INTEGRATION TESTING**

End to End testing is done to find whether the modules are interfaced together properly, and error handling happens correctly.

In this type of testing, we test various integration of the project module by providing the input. The primary objective is to test the module interfaces to ensure that no errors are occurring when one module invokes the other module.

ESP8266 Interfacing with RC522 is checked and it's working as expected. The Information is getting passed through serial communication and there are no anomalies (or) interferences found when the RFID tags are scanned/ read.

Application Gateway is stateless and loosely coupled to IoT device. This makes a granular way connection and easier to scale the processes. Error Handling is done properly as network connection and unavailability issues are involved.

Application Gateway and Fabric Blockchain network is interfaced properly and the chain code behavior when invoked is as expected. Gateway API can handle communication with Peers and invoking smart contracts.

### 7.3. TEST CASES AND REPORTS

#### TEST CASE 1: CREATING ASSET

**DESCRIPTION:** Supply Assets information are appended to the blocks

**PRE-CONDITIONS FOR TEST CASE:**

- i. Should be logged into the IoT's Website
- ii. Create Asset Mode should be selected in Menu of Website

#### RFID Scanning

Steps	Step Description	Expected	Actual	Status
1.	RFID brought into proximity	RFID tags read properly	RFID tags are read properly	Success

**TABLE 7.1 RFID Scanning Testcase**

#### After Stopping Mode

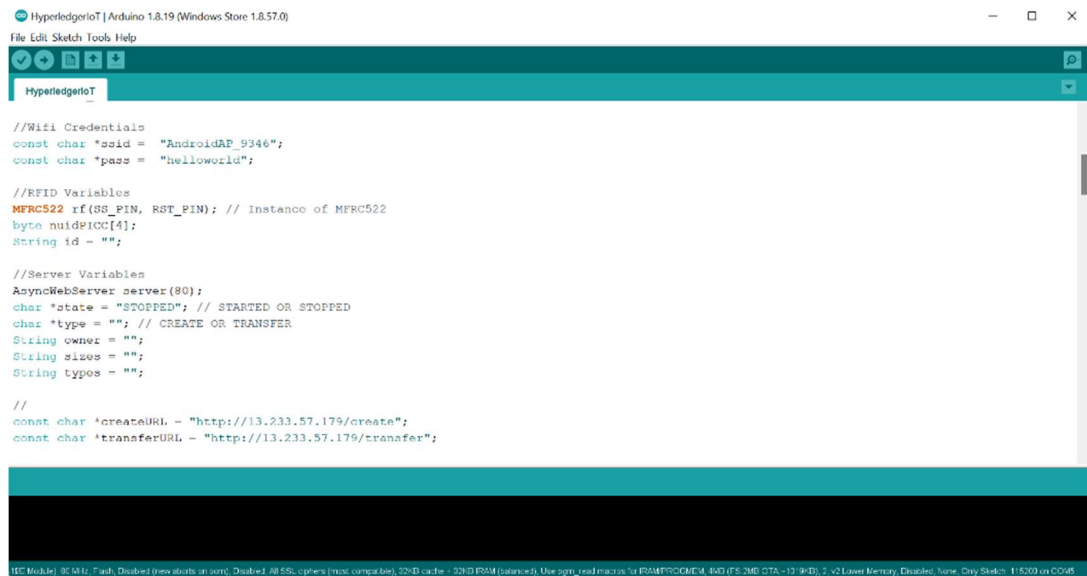
Steps	Step Description	Expected	Actual	Status
1.	IoT device connects with Application Gateway	Gateway sends proper response	Gateway calls Fabric SDK and sends proper response.	Success
2.	Application Gateway invokes smart contract to commit new blocks	New blocks should be appended	New blocks can be read	Success

**Table 7.2 After Stopping Mode Testcase**

## 8. CONCLUSION

### 8.1. RESULTS & DISCUSSION

From the obtained test cases and results above, we could infer that the proof of concept made possible using Hyperledger Fabric is working as expected.



```
HyperledgerIoT | Arduino 1.8.19 (Windows Store 1.8.57.0)
File Edit Sketch Tools Help

//Wifi Credentials
const char *ssid = "AndroidAP_9346";
const char *pass = "helloworld";

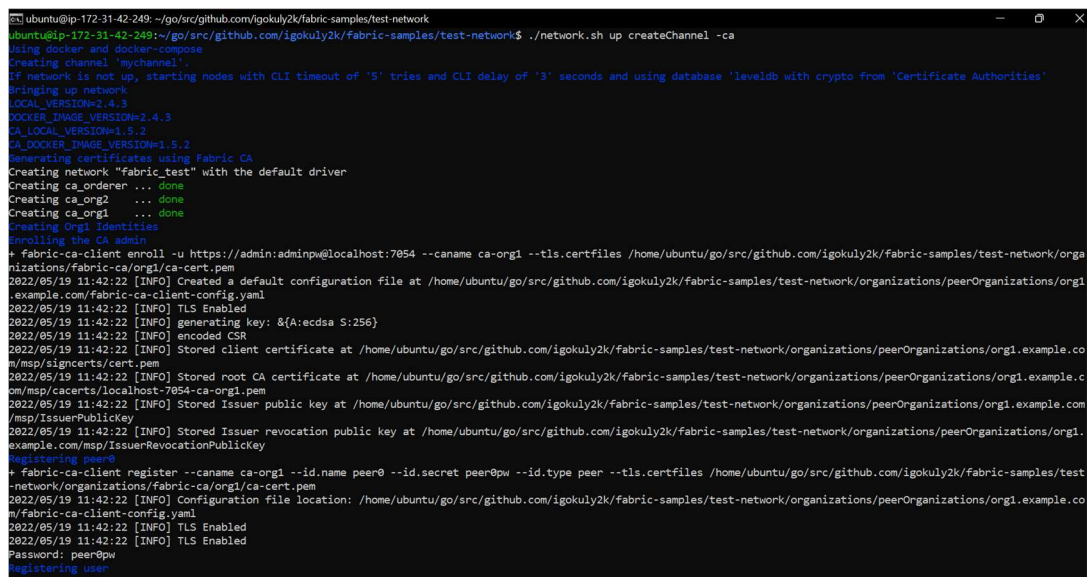
//RFID Variables
MFRC522 rf(SS_PIN, RST_PIN); // Instance of MFRC522
byte nuidPICC[4];
String id = "";

//Server Variables
AsyncWebServer server(80);
char *state = "STOPPED"; // STARTED OR STOPPED
char *type = ""; // CREATE OR TRANSFER
String owner = "";
String sizes = "";
String types = "";

//
const char *createUrl = "http://13.233.57.179/create";
const char *transferURL = "http://13.233.57.179/transfer";

//HC Module| 9C MHz; Flash, Disabled (new boards on sale); Disabled; All SPI; SPIs (most compatible); 32-D cache - 32K-D RAM (balanced); Use SPI; read macros for RAMPROGMEM; 4MB (FS: 2MB Ota - 1319-D); 2 v2 Lower Memory; Disabled; None; Only Sketch; 115200 on COM5
```

Fig. 8.1 ESP8266 Programming with Arduino IDE



```
ubuntu@ip-172-31-42-249: ~/go/src/github.com/igokuly2k/fabric-samples/test-network
ubuntu@ip-172-31-42-249:~/go/src/github.com/igokuly2k/fabric-samples/test-network$ ./network.sh createChannel -ca
Using docker and docker-compose
Creating channel 'mychannel'.
If network is not up, starting nodes with CLI timeout of '5' tries and CLI delay of '3' seconds and using database 'leveldb with crypto from 'Certificate Authorities'
Bringing up network
LOCAL_VERSION=2.4.3
DOCKER_IMAGE_VERSION=2.4.3
CA_LOCAL_VERSION=1.5.2
CA_DOCKER_IMAGE_VERSION=1.5.2
Generating certificates using Fabric CA
Creating network "fabric_test" with the default driver
Creating ca_orderer ... done
Creating ca_org2 ... done
Creating ca_org1 ... done
Creating Org1 identities
Enrolling the CA admin
+ fabric-ca-client enroll -u https://admin:adminpw@localhost:7054 --caname ca-org1 --tls.certfiles /home/ubuntu/go/src/github.com/igokuly2k/fabric-samples/test-network/organizations/fabric-ca/org1/ca-cert.pem
2022/05/19 11:42:22 [INFO] Created a default configuration file at /home/ubuntu/go/src/github.com/igokuly2k/fabric-samples/test-network/organizations/peerOrganizations/org1.example.com/fabric-ca-client-config.yaml
2022/05/19 11:42:22 [INFO] TLS Enabled
2022/05/19 11:42:22 [INFO] generating key: &{A:ecdsa S:256}
2022/05/19 11:42:22 [INFO] encoded CSR
2022/05/19 11:42:22 [INFO] Stored client certificate at /home/ubuntu/go/src/github.com/igokuly2k/fabric-samples/test-network/organizations/peerOrganizations/org1.example.com/msp/signcerts/cert.pem
2022/05/19 11:42:22 [INFO] Stored root CA certificate at /home/ubuntu/go/src/github.com/igokuly2k/fabric-samples/test-network/organizations/peerOrganizations/org1.example.com/msp/cacerts/localhost-7054-ca-org1.pem
2022/05/19 11:42:22 [INFO] Stored Issuer public key at /home/ubuntu/go/src/github.com/igokuly2k/fabric-samples/test-network/organizations/peerOrganizations/org1.example.com/msp/IssuerPublicKey
2022/05/19 11:42:22 [INFO] Stored Issuer revocation public key at /home/ubuntu/go/src/github.com/igokuly2k/fabric-samples/test-network/organizations/peerOrganizations/org1.example.com/msp/IssuerRevocationPublicKey
Registering peer0
+ fabric-ca-client register --caname ca-org1 --id.name peer0 --id.secret peer0pw --id.type peer --tls.certfiles /home/ubuntu/go/src/github.com/igokuly2k/fabric-samples/test-network/organizations/fabric-ca/org1/ca-cert.pem
2022/05/19 11:42:22 [INFO] Configuration file location: /home/ubuntu/go/src/github.com/igokuly2k/fabric-samples/test-network/organizations/peerOrganizations/org1.example.com/fabric-ca-client-config.yaml
2022/05/19 11:42:22 [INFO] TLS Enabled
2022/05/19 11:42:22 [INFO] TLS Enabled
Password: peer0pw
Registering user
```

Fig. 8.2 Creating Blockchain Sample Network

Fabric Blockchain network is created and instantiated using network.sh script file. Docker containers are created for peers and orderers.

```
ubuntu@ip-172-31-42-249: ~/go/src/github.com/igokuly2k/fabric-samples/test-network
Creating peer0.org1.example.com ... done
Creating cli ... done
CONTAINER ID        IMAGE                                COMMAND                  CREATED            STATUS              PORTS
0f6a1cfd17e        hyperledger/fabric-tools:latest    "/bin/bash"             1 second ago      Up Less than a second
742d1524887d        hyperledger/fabric-peer:latest     "peer node start"       2 seconds ago     Up 1 second        0.0.0.0:9444->9444/tcp
b9915c5f8e04        hyperledger/fabric-peer:latest     "peer node start"       2 seconds ago     Up Less than a second    0.0.0.0:9051->9051/tcp, 7851/tcp, 0.0
954895843f3c        hyperledger/fabric-orderer:latest  "orderer"               2 seconds ago     Up Less than a second    0.0.0.0:7050->7050/tcp, 0.0.0.0:7053-
4b385a3f3175        hyperledger/fabric-ca:latest       "sh -c 'fabric-ca-se..." 9 seconds ago      Up 7 seconds         0.0.0.0:7054->7054/tcp, 0.0.0.0:17054
b2e88d1be4d        hyperledger/fabric-ca:latest       "sh -c 'fabric-ca-se..." 9 seconds ago      Up 7 seconds         0.0.0.0:8054->8054/tcp, 7854/tcp, 0.0
95915c5f8e04        hyperledger/fabric-ca:latest       "sh -c 'fabric-ca-se..." 9 seconds ago      Up 7 seconds         0.0.0.0:9054->9054/tcp, 7854/tcp, 0.0
Using docker and docker-compose
Generating channel genesis block 'mychannel.block'
+ configtxgen -profile TwoOrgsApplicationGenesis -outputBlock ./channel-artifacts/mychannel.block -channelID mychannel
2022-05-19 11:42:28.864 UTC #0001 INFO [common.tools.configtxgen] main -> Loading configuration
2022-05-19 11:42:28.875 UTC #0002 INFO [common.tools.configtxgen.localconfig] completeInitialization -> orderer type: etcdraft
2022-05-19 11:42:28.875 UTC #0003 INFO [common.tools.configtxgen.localconfig] completeInitialization -> Orderer.Etcdraft.Options unset, setting to tick_interval: '500ms' elec
tion_tick:10 heartbeat_tick:1 max_inflight_blocks:5 snapshot_interval_size:16777216
2022-05-19 11:42:28.876 UTC #0004 INFO [common.tools.configtxgen.localconfig] Load -> Loaded configuration: /home/ubuntu/go/src/github.com/igokuly2k/fabric-samples/test-netw
ork/configtx/configtx.yaml
2022-05-19 11:42:28.882 UTC #0005 INFO [common.tools.configtxgen] doOutputBlock -> Generating genesis block
2022-05-19 11:42:28.882 UTC #0006 INFO [common.tools.configtxgen] doOutputBlock -> Creating application channel genesis block
2022-05-19 11:42:28.883 UTC #0007 INFO [common.tools.configtxgen] doOutputBlock -> Writing genesis block
+ res=0
creating channel mychannel
using organization 1
+ osnadmin channel join --channelID mychannel --config-block ./channel-artifacts/mychannel.block -o localhost:7053 --ca-file /home/ubuntu/go/src/github.com/igokuly2k/fabric
-samples/test-network/organizations/ordererOrganizations/example.com/tlsca/tlsca.example.com-cert.pem --client-cert /home/ubuntu/go/src/github.com/igokuly2k/fabric-samples/
test-network/organizations/ordererOrganizations/example.com/orderers/orderer.example.com/tls/server.crt --client-key /home/ubuntu/go/src/github.com/igokuly2k/fabric-samples
/test-network/organizations/ordererOrganizations/example.com/orderers/orderer.example.com/tls/server.key
+ res=0
Status: 201
```

Fig. 8.3 Docker Containers created

```
ubuntu@ip-172-31-42-249: ~/go/src/github.com/igokuly2k/fabric-samples/test-network
Using docker and docker-compose
Deploying chaincode on channel 'mychannel'
executing with the following
+ CHANNEL_NAME: mychannel
+ CC_NAME: basic
+ CC_SRC_PATH: ../asset-transfer-basic/chaincode-go/
+ CC_SRC_LANGUAGE: go
+ CC_VERSION: 1.0
+ CC_SEQUENCE: 1
+ CC_END_POLICY: NA
+ CC_COLL_CONFIG: NA
+ CC_INIT_FCN: NA
+ DELAY: 3
+ MAX_RETRY: 5
+ VERBOSE: false
Vendors Go dependencies at ../asset-transfer-basic/chaincode-go/
~/go/src/github.com/igokuly2k/fabric-samples/asset-transfer-basic/chaincode-go ~/go/src/github.com/igokuly2k/fabric-samples/test-network
~/go/src/github.com/igokuly2k/fabric-samples/test-network
Finished vendoring Go dependencies
+ peer lifecycle chaincode package basic.tar.gz --path ../asset-transfer-basic/chaincode-go/ --lang golang --label basic_1.0
+ res=0
Chaincode is packaged
Installing chaincode on peer0.org1...
using organization 1
+ peer lifecycle chaincode install basic.tar.gz
+ res=0
2022-05-19 11:46:04.594 UTC #0001 INFO [cli.lifecycle.chaincode] submitInstallProposal -> Installed remotely: response:(status:200 payload:"\n\"basic_1.0:4b4ef2874954eebf859
017f3c9928dc69bd2e25c8df8e773a5247414f613c\"@022/tbasic_1.0")
2022-05-19 11:46:04.601 UTC #0002 INFO [cli.lifecycle.chaincode] submitInstallProposal -> Chaincode code package identifier: basic_1.0:4b4ef2874954eebf859017f3c9928dc69bd2e
25c8df8e773a5247414f613c
Chaincode is installed on peer0.org1
Install chaincode on peer0.org2...
using organization 2
+ peer lifecycle chaincode install basic.tar.gz
+ res=0
2022-05-19 11:46:29.190 UTC #0001 INFO [cli.lifecycle.chaincode] submitInstallProposal -> Installed remotely: response:(status:200 payload:"\n\"basic_1.0:4b4ef2874954eebf859
017f3c9928dc69bd2e25c8df8e773a5247414f613c\"@022/tbasic_1.0")
2022-05-19 11:46:29.196 UTC #0002 INFO [cli.lifecycle.chaincode] submitInstallProposal -> Chaincode code package identifier: basic_1.0:4b4ef2874954eebf859017f3c9928dc69bd2e
25c8df8e773a5247414f613c
```

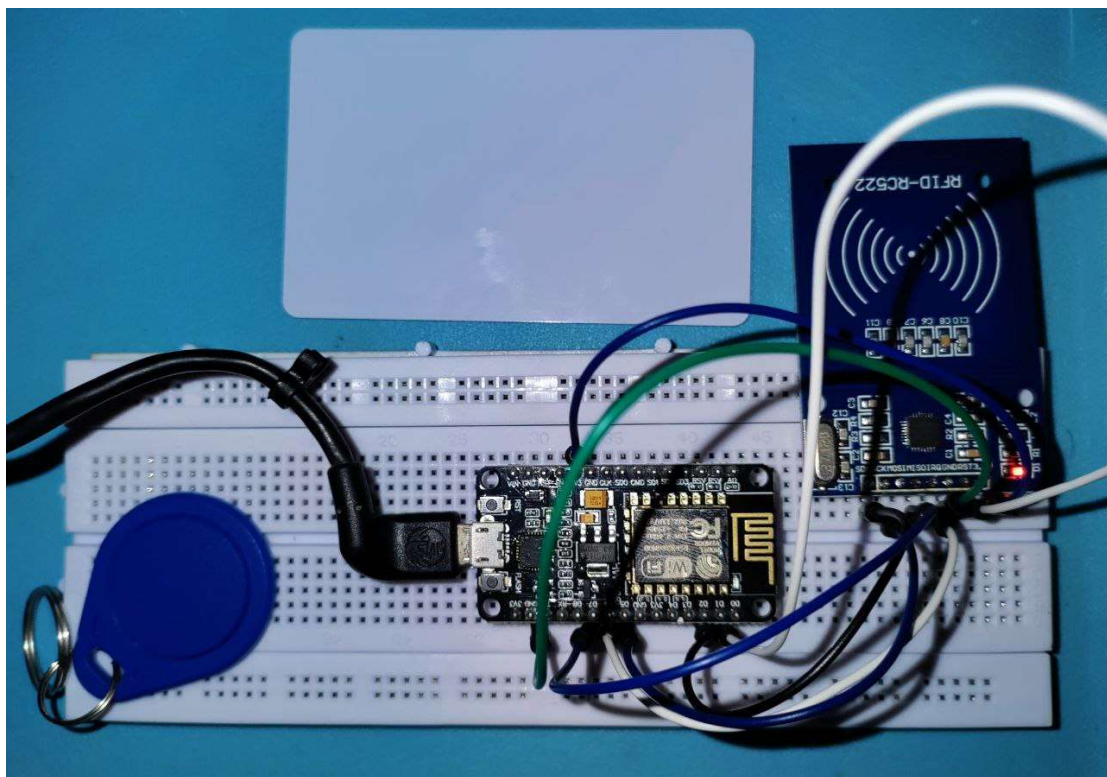
Fig. 8.4 Chaincode Installation on peers

```
ubuntu@ip-172-31-42-249: ~/go/src/github.com/igokuly2k/fabric-samples/asset-transfer-basic/application-gateway-typescript
ubuntu@ip-172-31-42-249:~/go/src/github.com/igokuly2k/fabric-samples/test-network$ cd ../asset-transfer-basic/application-gateway-typescript
ubuntu@ip-172-31-42-249:~/go/src/github.com/igokuly2k/fabric-samples/asset-transfer-basic/application-gateway-typescript$ sudo iptables -t nat -A PREROUTING -i eth0 -p tcp
--dport 80 -j REDIRECT --to-port 8080
ubuntu@ip-172-31-42-249:~/go/src/github.com/igokuly2k/fabric-samples/asset-transfer-basic/application-gateway-typescript$ npm run serve

> asset-transfer-basic@1.0.0 serve
> nodemon src/index.ts

[nodemon] 2.0.16
[nodemon] to restart at any time, enter 'rs'
[nodemon] watching path(s): *.*
[nodemon] watching extensions: ts,json
[nodemon] starting 'ts-node src/index.ts'
channelName:
chaincodeName:
basic
mspId:
Org1MSP
cryptoPath:
/home/ubuntu/go/src/github.com/igokuly2k/fabric-samples/test-network/organizations/peerOrganizations/org1.example.com
keyDirectoryPath:
/home/ubuntu/go/src/github.com/igokuly2k/fabric-samples/test-network/organizations/peerOrganizations/org1.example.com/users/User1@org1.example.com/msp/ke
ystore
certPath:
/home/ubuntu/go/src/github.com/igokuly2k/fabric-samples/test-network/organizations/peerOrganizations/org1.example.com/users/User1@org1.example.com/msp/si
gncerts/cert.pem
tlsCertPath:
/home/ubuntu/go/src/github.com/igokuly2k/fabric-samples/test-network/organizations/peerOrganizations/org1.example.com/peers/peer0.org1.example.com/tls/ca
.crt
peerEndpoint:
localhost:7051
peerHostAlias:
peer0.org1.example.com
channelName:
mychannel
chaincodeName:
basic
mspId:
Org1MSP
cryptoPath:
/home/ubuntu/go/src/github.com/igokuly2k/fabric-samples/test-network/organizations/peerOrganizations/org1.example.com
keyDirectoryPath:
/home/ubuntu/go/src/github.com/igokuly2k/fabric-samples/test-network/organizations/peerOrganizations/org1.example.com/users/User1@org1.example.com/msp/ke
ystore
certPath:
/home/ubuntu/go/src/github.com/igokuly2k/fabric-samples/test-network/organizations/peerOrganizations/org1.example.com/users/User1@org1.example.com/msp/si
gncerts/cert.pem
tlsCertPath:
/home/ubuntu/go/src/github.com/igokuly2k/fabric-samples/test-network/organizations/peerOrganizations/org1.example.com/peers/peer0.org1.example.com/tls/ca
.crt
peerEndpoint:
localhost:7051
peerHostAlias:
peer0.org1.example.com
Contract returned successfully
Contract returned successfully
```

**Fig. 8.5 Application gateway compiled**



**Fig. 8.6 ESP8266 Interfaced with RFC522**

## **8.2. CONCLUSION AND FUTURE ENHANCEMENTS**

Proof of Concept on functionalities of PDS using Hyperledger Fabric makes us to confirm that Permissioned Blockchain Infrastructures are helpful in inducing more transparency, traceability and accountability in food logistics and supply chain industry. Blockchain would fortify trust among organizations involved in the chain and consumers would be aware of the product and its lifecycle. Blockchain Infrastructures can be used as catalyst for Artificial Intelligence decisions, and this would give way for efficient, frugal, optimized logistics. The proof of concept can be scaled to production, and this would make sure that the intended right product reaches the consumer.

This prototype can be made possible into a real time project by using Far Range UHF Passive RFID technology. Tags are inexpensive and RFID reader's proximity can range up to 12 meters. Zigbee's performance could be tested against Wi-Fi performance which would lead us to choose the more power efficient communication protocol with sufficient bandwidth capability. Sawtooth and Fabric compared against each other, and this would make us to review the better blockchain framework for logistics and supply chain. Sophisticated Application-Specific Integrated circuit could be developed instead of general-purpose microcontrollers.

Much more logistics case studies could be analyzed and a better general tool matching the use cases required could be developed. Pilot tests could be conducted in several parts of Tamil Nadu to analyze the impact of blockchain implementation in PDS.



## APPENDICES

### A1. SAMPLE SCREENS



**PDS Supply Chain  
Tracking**

Create Asset Mode

Type

Size

**A1.1 Create Asset Mode Form**

The image shows a black smartphone with a white screen. The screen displays the following text and elements:

- PDS Supply Chain Tracking** (Title)
- Transfer Asset Mode** (Section Header)
- Owner** (Label) followed by a text input field.
- Submit** (Button)

**A1.2 Transfer Asset Mode Form**



**A1.3 Mode Started Placeholder**



**A1.4 Mode Stop Placeholder**



**A1.5 Mode Menu Selection**

## REFERENCES

- [1] Satoshi Nakamoto,"Bitcoin: A Peer-to-Peer Electronic Cash System",[www.bitcoin.org](http://www.bitcoin.org), 2008
- [2] Melanie Swan,"Blockchain Blueprint for a New Economy",Published by O'Reilly Media, Inc.,2015
- [3] GUIDO PERBOLI, STEFANO MUSSO, MARIANGELA ROSANO,"Blockchain in Logistics and Supply Chain: A Lean Approach for Designing Real-World Use Cases", IEEE,2018
- [4] Reshma Kamath,"Food Traceability on Blockchain: Walmart's Pork and Mango Pilots with IBM",  
The JBBA,2018
- [5] Giorgio Alessandro Motta, Bedir Tekinerdogan and Ioannis N. Athanasiadis,"Blockchain Applications in the Agri-Food Domain: The First Wave",Frontiers in Blockchain,2020
- [6] Kok Yong Chan,Johari Abdullah,Adnan Shahid Khan,"A Framework for Traceable and Transparent Supply Chain Management for Agri-food Sector in Malaysia using Blockchain Technology", IJACSA,2019
- [7] Rocha T, Costa P, Sousa V,Coelho P, Sousa F, Cardoso N,"SmartAgriChain: A Blockchain Based Solution for Agri food Certification and Supply Chain Management",International Journal of Environment, Agriculture and Biotechnology,2021
- [8] Malni Kumarathunga,"Improving Farmers' Participation in Agri Supply Chains with Blockchain and Smart Contracts", IEEE,2020
- [9] Sini V. Pillai, Vipin H., Anand Mohan, Shruthi Dileep A,"Infusing Blockchain in Supply Chain Operations of a Public Distribution System",Journal of Supply Chain Management Systems,2020

[10] Sandeep Kumar Singh, Mamata Jenamani , Diptiman Dasgupta & Suman Das,"A conceptual model for Indian public distribution system using consortium blockchain with on-chain and off-chain trusted data",Information Technology for Development,2020

[11] Himani Mishra,Prateek Maheshwari,"Blockchain in Indian Public Distribution System: a conceptual framework to prevent leakage of the supplies and its enablers and disablers",ResearchGate,2021

[12] R Gayatri, “SMS based monitoring system for Fair Price Shops in Tamilnadu”, Informatics NIC, July 2014