BLOCKCHAIN BASED FOOD SUPPLY CHAIN TRACEABILITY IN SUPPLY CHAIN MANAGEMENT

Report submitted to the SASTRA Deemed to be University

as the requirement for the course

BCSCCS708: **MINI PROJECT**

Submitted by

Moldi Reddi Gari Teja Vardhan Reddy

(Reg. No.: 123003143, B. Tech CSE)

P J Sai Pavan

(Reg. No.: 123015080, B. Tech IT)

Kaperla Dharma Swaroop

(Reg. No.: 123003289, B. Tech CSE)

June 2022

SCHOOL OF COMPUTING

THANJAVUR, TAMIL NADU, INDIA – 613 401

SASTRA

ENGINEERING · MANAGEMENT · LAW · SCIENCES · HUMANITIES · EDUCATION

DEEMED TO BE UNIVERSITY

(U/S 3 OF THE UGC ACT, 1956)

THINK MERIT | THINK TRANSPARENCY | THINK SASTRA

SCHOOL OF COMPUTING

THANJAVUR – 613 401

<u>Bonafide Certificate</u>

This is to certify that the report titled “**Blockchain based food supply chain traceability in supply chain management**” submitted as a requirement for the course, BCSCCS708: **MINI PROJECT** for B.Tech. is a bonafide record of the work done by **Mr. Moldi Reddi Gari Teja Vardhan Reddy (Reg. No.123003143, B. Tech CSE), Mr. PJ Sai Pavan (Reg. No: 123015080, B. Tech IT), Mr. Kaperla Dharma Swaroop (Reg. No: 123003289, B. Tech CSE)** during the academic year 2021-22, in the School of Computing, under my supervision.

Signature of Project Supervisor	: [illegible]
Name with Affiliation	: **K. Geetha, Associate Professor, School of computing**
Date	: **28-06-22**

Mini Project *Viva voce* held on <u>29/06/2022</u>

Examiner 1 **Examiner 2**

ACKNOWLEDGEMENTS

First and foremost, I thank the almighty for helping me to gain support in all forms to finish our mini project successfully. I express my sincere thanks Dr. S. VAIDHYASUBRAMANIAM, Vice Chancellor and Dr. R. CHANDRA MOULI Registrar, SASTRA UNIVERSITY, for permitting me to do this Mini Project as a part of our curriculum.

I express our profound gratitude to Dr. A. UMA MAKESHWARI, Dean School of Computer Science Engineering, SASTRA UNIVERSITY and Dr. S. SHANKAR SRIRAM, Associate Dean, School of Computer Science Engineering, SASTRA UNIVERSITY for their complete support throughout the Mini Project.

I am fortunate to have Dr. GEETHA K as our project guide. Her valuable assistance and supervision guided me towards the successful completion of our Mini project. Lastly, I thank all the technical and non-technical staff of Computer science and Engineering Department and my parents and friends for their constant support throughout the completion of my Mini-Project Report.

Table of Contents

List of Figures

List of Tables

ABBREVATIONS

FSC	Food Supply Chain
IOT	Internet Of Things
IPFS	Inter Planetary File System
SC	Smart Contract
GUI	Graphical User Interface
ECC	Elliptic Curve Cryptography

ABSTRACT

Food traceability has become the most important factor in the Food Supply Chain (FSC). It is important to store and handle private case data for tracing in FSC. So, any FSC present out there collects information about the users to make the tracing possible. Also, the users are having high expectations on this FSC’s. Initially Traceability was made possible based on technologies like Internet of Things (IOT). Despite the benefits of these IoT based approach, there are still various drawbacks that exist in the FSC due to its complexity and fragmentation.

For multinational FSC networks with several suppliers, where traceability data needs to be in digital format and available by various stakeholders and its processing needs to be automated to comply with various requirements, the accuracy of the relevant information is also crucial. This work develops and implements a distributed, secure, and trustless architecture for FSC traceability. To assess the viability of the proposed strategy, a dairy company's food traceability case study is presented.

This is done using the concept of blockchain which is a latest booming technology and also the concept of smart contracts is used which is basically a set of instructions which are executed automatically and control the events. The creation of operational smart contracts and implementation of local private blockchain further demonstrate the model's applicability. In this, many connections between the proposed blockchain-based approach and its managerial ramifications are presented.

KEY WORDS: Blockchain, Smart Contracts, Supply Chain Management, Solidity, Truffle, Ganache.

CHAPTER 1

SUMMARY OF THE BASE PAPER

1.1 Introduction

Table 1.1 Base paper details

Title	Blockchain based food supply chain traceability in supply chain management
Journal Name	International Journal of Production Research
Publisher	Taylor and Francis
Year	2020
Indexed in	SCI

The concept of traceability is one of the biggest concerns when it comes to security sensitive areas like food and pharmacy. It is very important that traceability is maintained across the food supply systems. Food traceability Systems captures the information about a food, stores it and information is transferred. Also, the information about the feed, animals from which that food is produced or substances are captured, stored at every stage that is present in the Food Supply Chain (FSC). With the help of this process, the product can be verified for safety and quality is assured also, it can be tracked downward and upward at any time. The main characteristics that are essential for traceability systems are, identification all the units that consists of ingredients and also products, data regarding when those units are moved and also where they are moved and finally a system link the data that was acquired till now. With the advancement of technology and this traceability has now become the new quality index. As we have already mentioned, for tracing in FSC to be made possible, it is necessary to store and handle the sensitive information and this has become a mandatory case. Regulations are put in place to make it possible to trace and identify all of the ingredients used to make food items. All this effort we undergo is to ensure the safety of the food items and other items like medicines right from the manufacturer to consumer. Initially this was made possible with help of IoT technology. But this was difficult due to its complexity and its fragmentation. The food sector utilizes traceability technologies to enhance FSC and facilitate food safety and quality tracing. In this project, A dairy products related company's case study is considered for food traceability in order to help evaluate the viability of the suggested approach.

The approach used here is using blockchain to provide the required services. Blockchain has become one of the most booming technologies during recent years. The use of blockchain technology is a reality in many industries, and it is developing to be the most major technological revolution since the creation of the Internet. We are also using a concept related to blockchain that is smart contracts. The concept of smart contracts was first introduced to world by Szabo in the year 1997. It is a relatively new development in blockchain technology. Smart contracts give users the capacity to do computations on their own and these are done inside a blockchain, acting as a decentralized computer which is virtual. We can say that Smart Contracts are piece of codes that are executed automatically and self-controlled. A smart contract is generally understood to be a set of computer protocols that enable an agreement that is automatically performed while taking into consideration, a number of predetermined circumstances. The bulk of blockchain applications now in use incorporate smart contracts. We here create three smart contracts. Those are for Stakeholders, processes and products. We will explain more in detail regarding these smart contracts and the deployment of these smart contracts under Methodology section.

1.2 Literature Survey:

S.No	Paper Title, Journal Name	Methodology Used	Merit	Demerit
1.	The rise of blockchain technology in agriculture and food supply chains	Blockchain technology in agriculture and food supply chain, presents existing ongoing projects and initiatives, and discusses overall implications, challenges and potential, with a critical view over the maturity of these projects.	• Blockchain is a promising technology towards a transparent supply chain of food. • Discussion of the impact of blockchain in	Because supply chain information can be sensitive, a permissioned block chain is usually preferred. But in turn it makes the process slower.

			agriculture and food supply chains.	
2.	Food traceability: new trends and recent advances. A review	Implementation of RFID that can make to increase the sales of wheat flour, or allowing the consumer to know the full record of the IV range products through the smartphone.	Enhancing the efficiency and compatibility of the present traceability systems.	It takes extra time and effort to keep up with the tracking of requirements during the project.
3.	How blockchain improves the supply chain: case study alimentary	The novelty of this paper lies in a block cain to store all transaction information in the supply chain of the proposed case study.	The multi-agent system uses smart contracts to manage the entire supply chain proves more efficiently.	Multi-agent system can be improved by introducing new agents for the monitoring of the procedures.
4.	Boundary Conditions for Traceability in Food Supply Chains Using Blockchain Technology.	Blockchain (BCT) has been introduced as a technology for supporting the enhancement of product data traceability.	A different starting is used and the focus is put on the investigation of the boundary condition.	Although the supply chain processes differ significantly from each other, standardization of quality standards and requirements between all internal as well as external actors in the supply cain is essential.

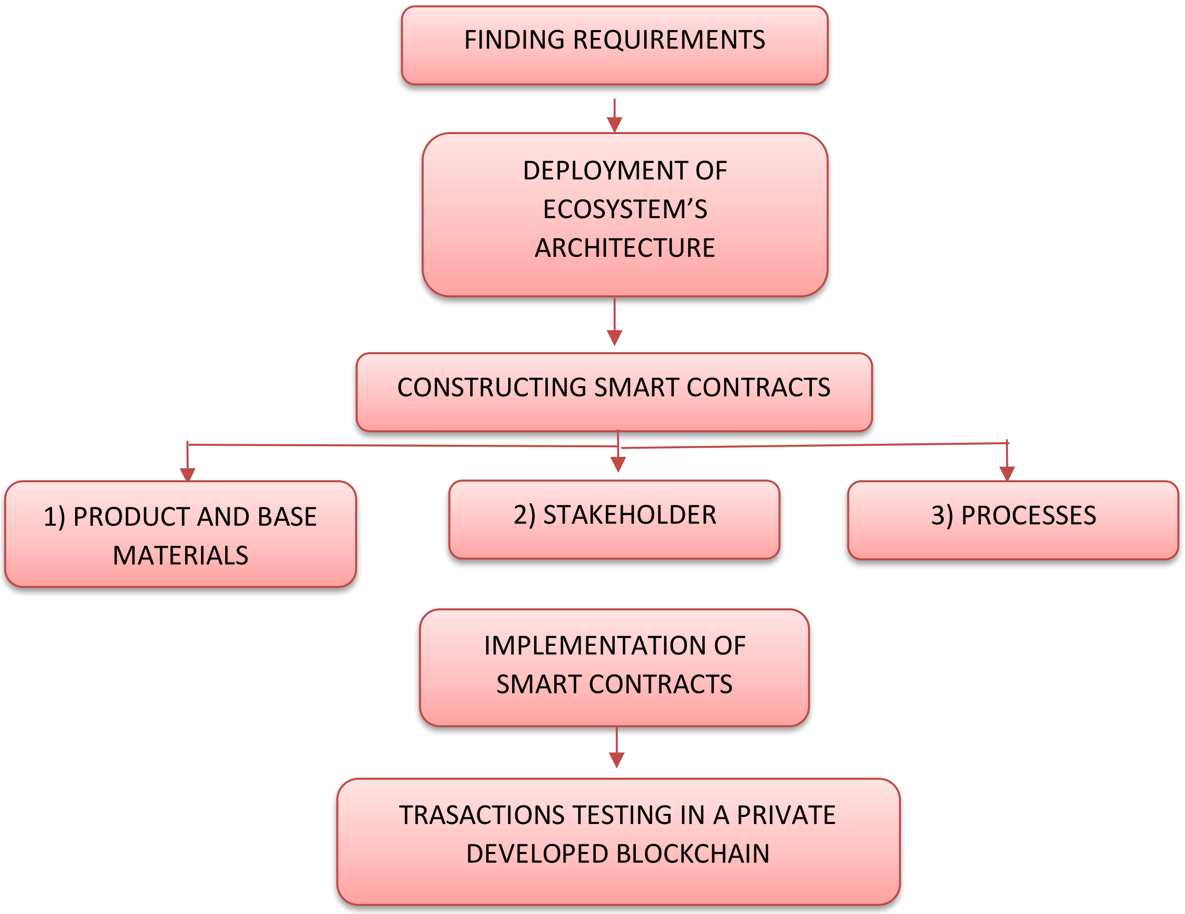
1.3 Methodology:

As we have mentioned before, the main concept used here is implementing blockchain based food supply chain. Let us look at the overall frame work of this idea and how it is implemented. The following steps make up the overall methodological framework:

i. Determining the needs of the ecosystem: The technical and management process requirements are set in accordance with the results that are theoretical from the existing literature and the present situation of practice. This was done based on the study conducted on food supply chain system known as Pagonis. This survey's information was used in order to continue the process.

ii. Development of ecosystem's architecture. This is the main step which involves development of smart contracts. We will describe about these smart contracts in more detail next.

iii. System implementation, testing and validation. This will be our final step to check the functionality of those smart contracts and to check if it's working as per the requirements.


FINDING REQUIREMENTS
DEPLOYMENT OF ECOSYSTEM'S ARCHITECTURE
CONSTRUCTING SMART CONTRACTS
1) PRODUCT AND BASE MATERIALS
2) STAKEHOLDER
3) PROCESSES
IMPLEMENTATION OF SMART CONTRACTS
TRASACTIONS TESTING IN A PRIVATE DEVELOPED BLOCKCHAIN

Let us look at how these steps are implemented in more detail

1. First, a real use case scenario is considered to find out the requirements and features. That is considering a dairy products supply system and knowing their requirements and problems faced by them.
2. Then, these requirements and corresponding features that are mapped into a functional blockchain that will be private one created using Ganache.
3. The use of "smart contracts" will be used to implement the necessary interactions between the products, stakeholders, and processes. “truffle” will be used in order to code and also deploying those set of functional smart contracts.
4. Three smart contracts that are created are:
   i) Products and base material
   ii) Stakeholders
   iii) Processes
5. These smart contracts contain all the necessary functions with permissions granted to only admin and user.

How are smart contracts implemented?

The language used to create smart contracts is solidity. The version we have used in this version 0.8.0 which have additional features compared to previous versions. We have different tools to implement these smart contracts. The tools that are used are Truffle and Ganache.

Truffle is a smart contract development IDE which makes implementing smart contracts easier.

Ganache is a tool that is used to create a local block chain which is Ethereum based and 10 fake accounts will be created that does not have any value. So, they can be used to check the working of our smart contracts.

i. Products smart contract:

Products smart contract contains all the necessary attributes and functions related to the dairy products. This preserves all connections between goods and raw materials as well as interactions with other two smart contracts.

The main functions that are implemented in this smart contract are:

- addProduct

- UpdateProductDescription
- addTestProduct
- getNumberOfProducts
- getProduct
- getProductTest
- addTrace
- addTemperature
- getNumberOfTraces
- getNumberOfTemperatures
- getTracesProduct
- checkStakeholder

ii. Stakeholders smart contract:

Stakeholders smart contract contains all the necessary attributes and functions related to Stakeholders. Similar to our previous smart contract, this also interact with other smart contracts as those other two smart contracts are also imported into this smart contracts.

The important functions that are implemented in this smart contract are:

- addStakeholder
- addStakeholderProduct
- changeStatus
- getStakeholdersProduct
- getStakeholder
- getNumberOfStakeholders
- updateNumberOfProducts
- updateNumberOfProcesses

iii. Processes smart contract:

Processes smart contract contains all the necessary attributes and the functions related to Processes. It also has other two smart contracts imported into this smart contract.

The important functions that are implemented in this smart contract are:

- addProcess
- addProcsessProduct
- changeStatus
- getProcessProduct
- getNumberOfPrroductsProcess
- getProcess
- checksSakeholder

so, these are the smart contracts that are being implemented in this work. Finally, to check the working of this smart contracts, they are being implemented in Remix IDE which is a online compiler for Smart contracts.

It has functionality to use any blockchain to implement our smart contracts and we will connect it to our local blockchain produced by Ganache command. That is “Ganache-cli” run on a terminal.

To do this we have to install Metamask extension to our chrome browser so that we can connect our local blockchain to our metamask and then finally we connect to out metamask accounts in from Remix IDE.

We finally verify the working of our smart contracts through this interactive web compiler.

CHAPTER 2

MERITS AND DEMERITS OF THE BASE PAPER

2.1 Methods that are already in use:

Initially, the problems that are raised in food supply management were solved using few novwl technologies as we have mentioned earlier. Here, let us look into few techniques that were used which had their own merits and demerits.

IOT based:

Foundation for FSC based management was formed by this traceability-driven technology known as Internet of Things (IoT).

Internet of Things plays a crucial role when it comes to providing all the real-time data regarding the food items or any delivered items for that matter. Also, the information regarding contamination throughout the production is made available. It also includes delivery data like real time location of the products. Application based on IoT have the capability to solve issues related to financial restrictions while redesigning and optimizing food supply networks.

By digitizing data so that it can be accessed and managed in real time, applications based on IoT and also pertinent technologies like RFID can together be used to revolutionize the sector.

2.2 Proposed method:

Blockchain based:

Despite the tremendous advantages of adopting IoT-related technologies, the FSC still faces a number of obstacles. For instance, FSC networks today are becoming more complicated and dispersed. Along globalized FSC networks, it is incredibly difficult to identify and trace items and processes due to its complexity. Also, during this time, the traceability has become a prerequisite for assuring safety in FSC and also for legal agreement, a deeper understanding on the life span of the products, and responsible consumption.

The blockchain is a technical development that has the potential to alter services by fostering trust in decentralised environments like FSC. It can work around established authorities, allow for quicker and more secure transactions, and offer backward control from the factory or farm to the final consumer.

For example, blockchain-enabled applications improve information sharing among various partners across FSC networks without sacrificing privacy and security.

2.3 Merits:

1. Data can be managed with ease when we use this technology.
2. The cost required to build the reliable traceability system will be reduced.
3. It proves complete transparency between the stakeholders, users and other people involved in the network.
4. It is easily auditable when someone wants to get the results which they want to verify.
5. Integrity is maintained as the blockchain technology is completely secure and also maintains authenticity.

2.4 Demerits:

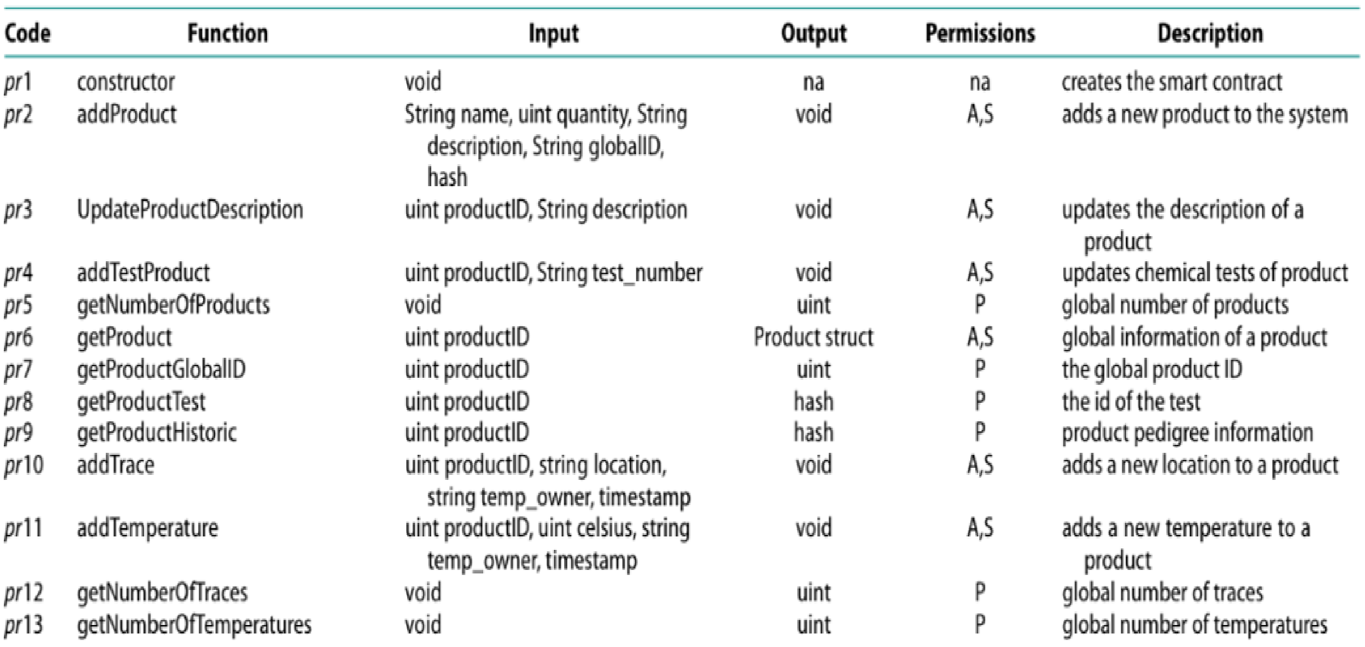
Though the blockchain implementation in FSC has many benefits and merits to it as we have seen before, it also has very few limitations such as,

- It can be difficult to store such large amount of data in a blockchain
- Whenever we want to do changes to the smart contracts, the whole deployment takes place again and takes up some gas payment for it to het deployed.
- It can be little complex to implement and create those smart contracts to interact with other smart contracts when there are many smart contracts involved.

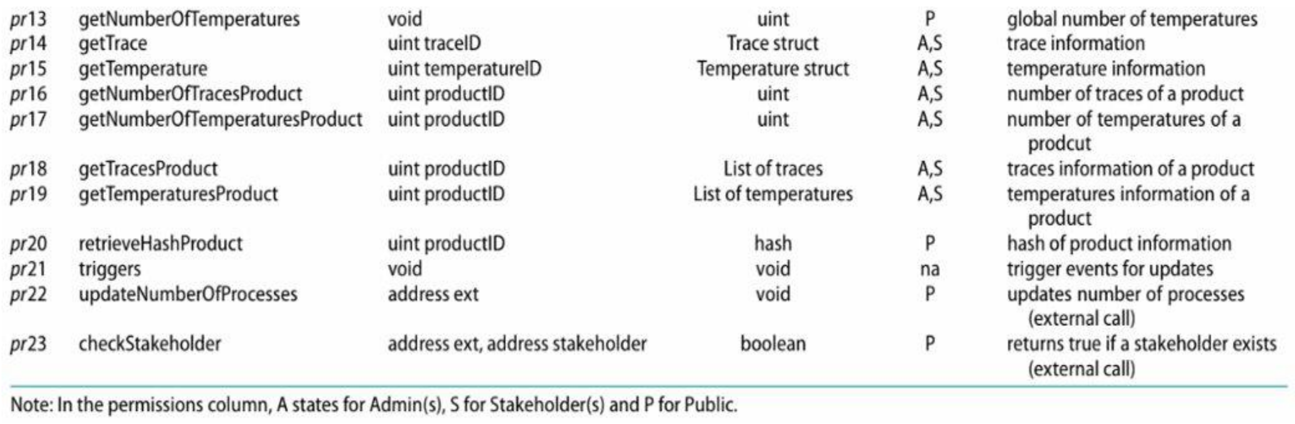
CHAPTER 3

SOURCE CODE

1) Food Pagonis. Sol



Code	Function	Input	Output	Permissions	Description
*pr*1	constructor	void	na	na	creates the smart contract
*pr*2	addProduct	String name, uint quantity, String description, String globalID, hash	void	A,S	adds a new product to the system
*pr*3	UpdateProductDescription	uint productID, String description	void	A,S	updates the description of a product
*pr*4	addTestProduct	uint productID, String test_number	void	A,S	updates chemical tests of product
*pr*5	getNumberOfProducts	void	uint	P	global number of products
*pr*6	getProduct	uint productID	Product struct	A,S	global information of a product
*pr*7	getProductGlobalID	uint productID	uint	P	the global product ID
*pr*8	getProductTest	uint productID	hash	P	the id of the test
*pr*9	getProductHistoric	uint productID	hash	P	product pedigree information
*pr*10	addTrace	uint productID, string location, string temp_owner, timestamp	void	A,S	adds a new location to a product
*pr*11	addTemperature	uint productID, uint celsius, string temp_owner, timestamp	void	A,S	adds a new temperature to a product
*pr*12	getNumberOfTraces	void	uint	P	global number of traces
*pr*13	getNumberOfTemperatures	void	uint	P	global number of temperatures

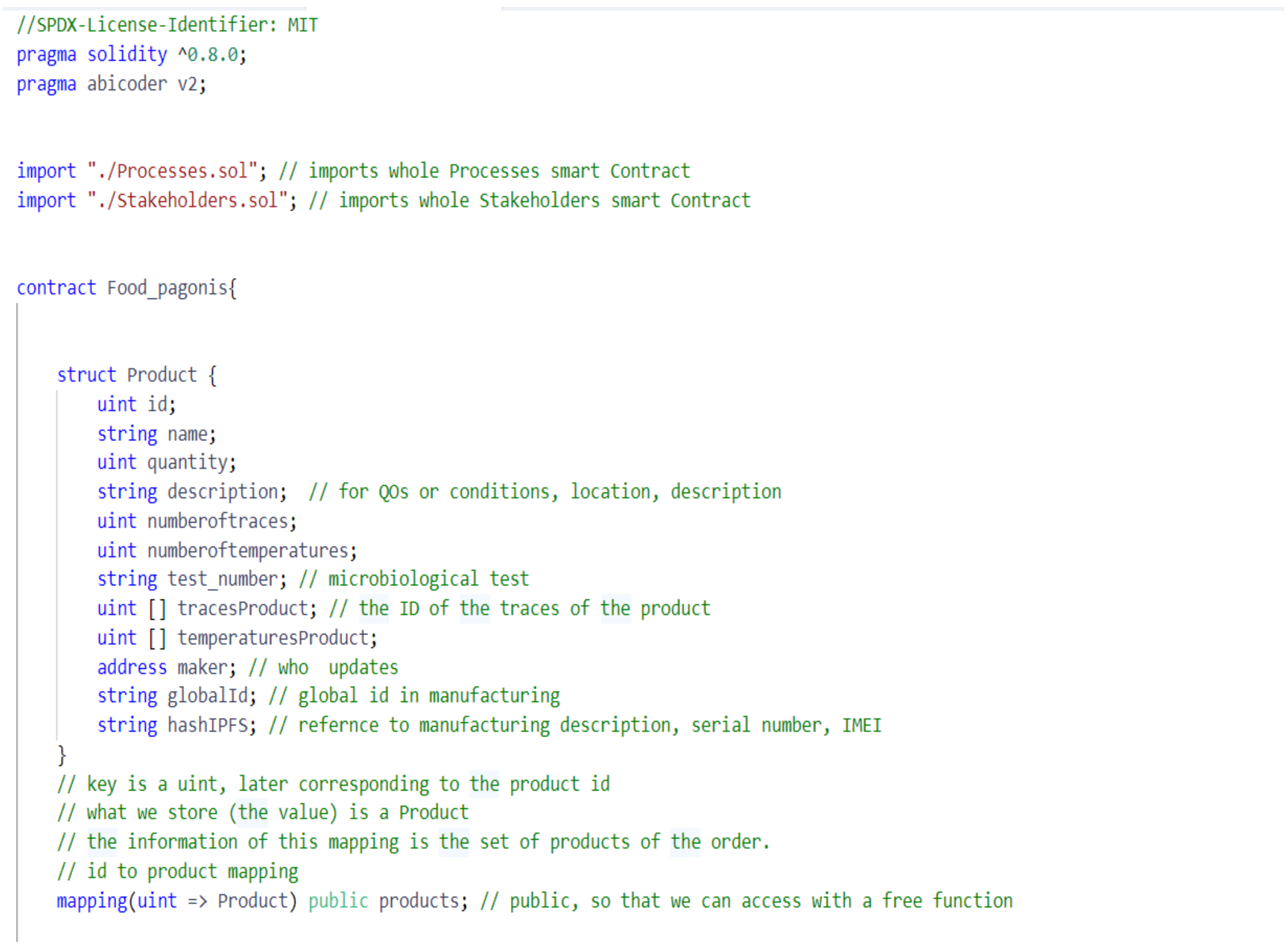


*pr*13	getNumberOfTemperatures	void	uint	P	global number of temperatures
*pr*14	getTrace	uint traceID	Trace struct	A,S	trace information
*pr*15	getTemperature	uint temperatureID	Temperature struct	A,S	temperature information
*pr*16	getNumberOfTracesProduct	uint productID	uint	A,S	number of traces of a product
*pr*17	getNumberOfTemperaturesProduct	uint productID	uint	A,S	number of temperatures of a prodcut
*pr*18	getTracesProduct	uint productID	List of traces	A,S	traces information of a product
*pr*19	getTemperaturesProduct	uint productID	List of temperatures	A,S	temperatures information of a product
*pr*20	retrieveHashProduct	uint productID	hash	P	hash of product information
*pr*21	triggers	void	void	na	trigger events for updates
*pr*22	updateNumberOfProcesses	address ext	void	P	updates number of processes (external call)
*pr*23	checkStakeholder	address ext, address stakeholder	boolean	P	returns true if a stakeholder exists (external call)

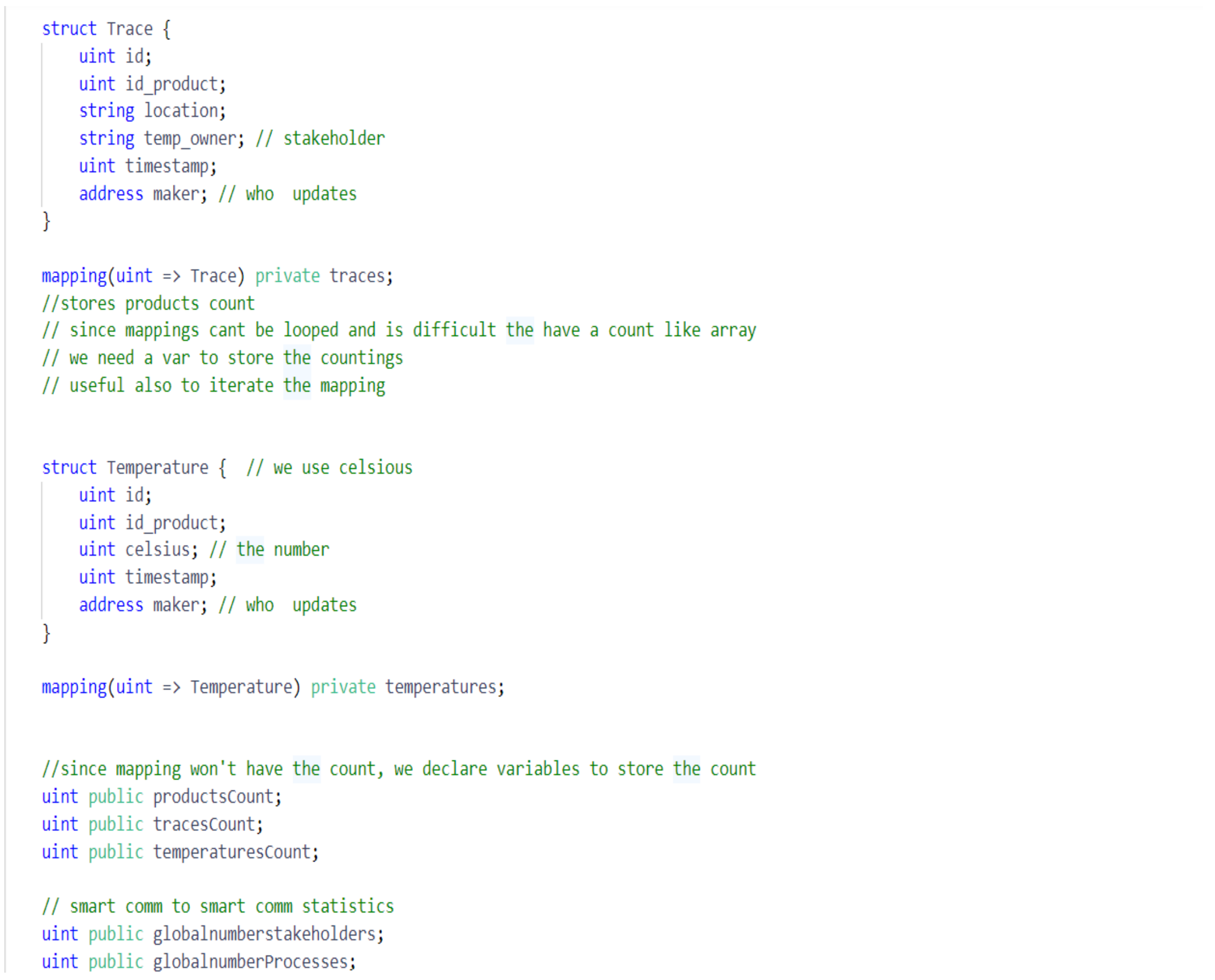
Note: In the permissions column, A states for Admin(s), S for Stakeholder(s) and P for Public.

Table 3.1 The above table shows the functions and permissions that are implemented in the smart contract

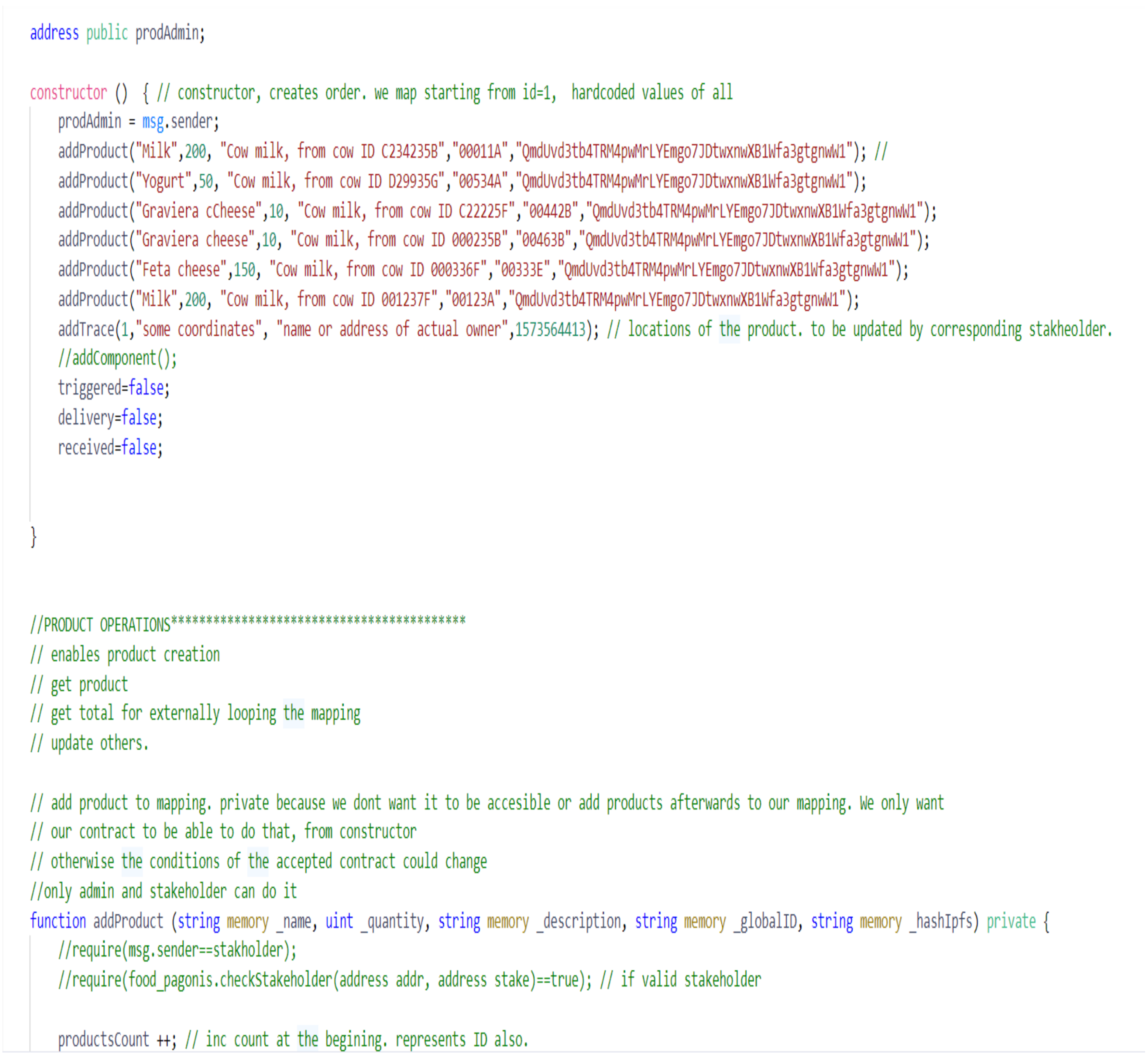
Code:



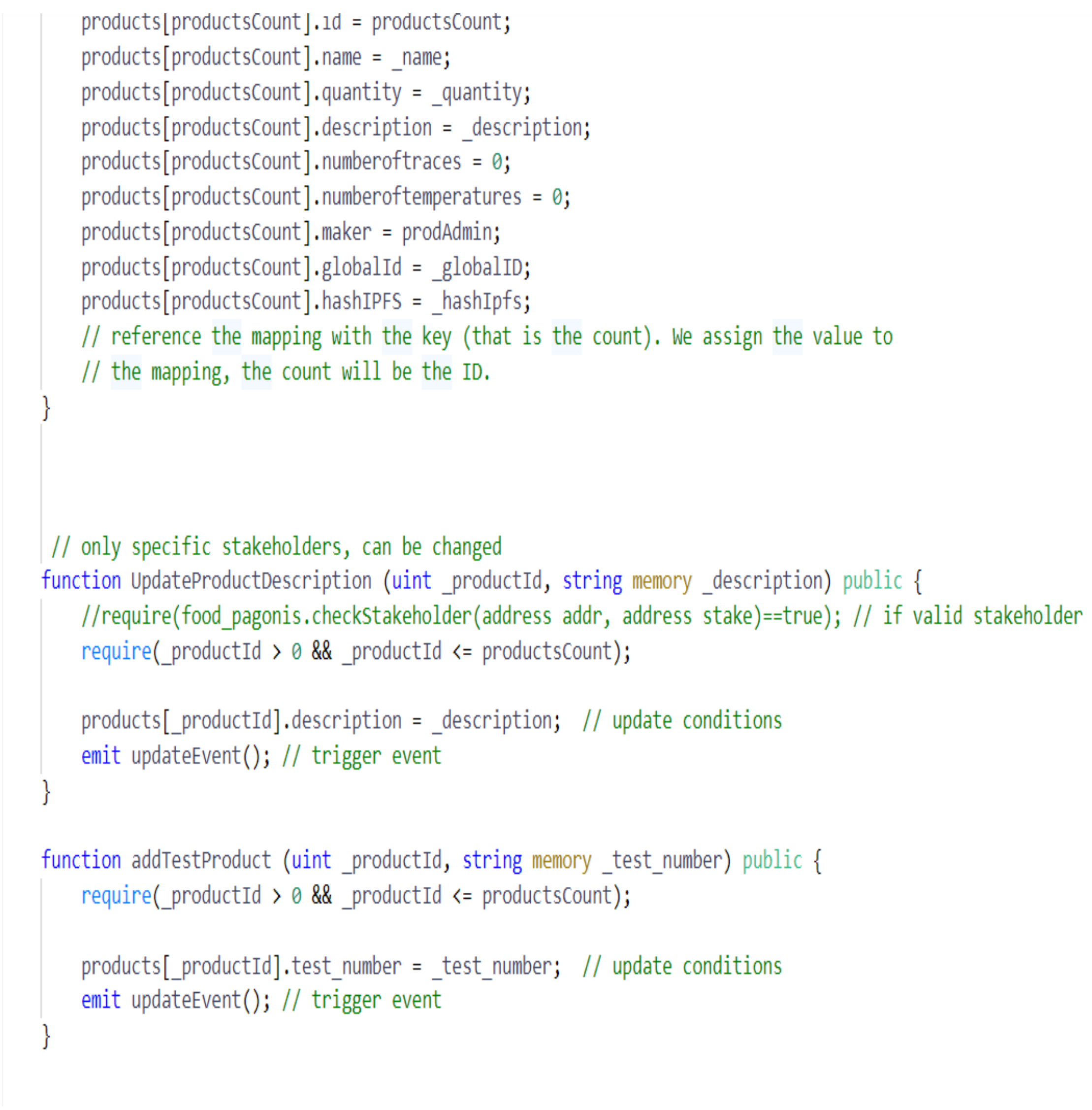
```
//SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;
pragma abicoder v2;


import "./Processes.sol"; // imports whole Processes smart Contract
import "./Stakeholders.sol"; // imports whole Stakeholders smart Contract


contract Food_pagonis{


    struct Product {
        uint id;
        string name;
        uint quantity;
        string description;  // for QOs or conditions, location, description
        uint numberoftraces;
        uint numberoftemperatures;
        string test_number; // microbiological test
        uint [] tracesProduct; // the ID of the traces of the product
        uint [] temperaturesProduct;
        address maker; // who  updates
        string globalId; // global id in manufacturing
        string hashIPFS; // refernce to manufacturing description, serial number, IMEI
    }
    // key is a uint, later corresponding to the product id
    // what we store (the value) is a Product
    // the information of this mapping is the set of products of the order.
    // id to product mapping
    mapping(uint => Product) public products; // public, so that we can access with a free function
```



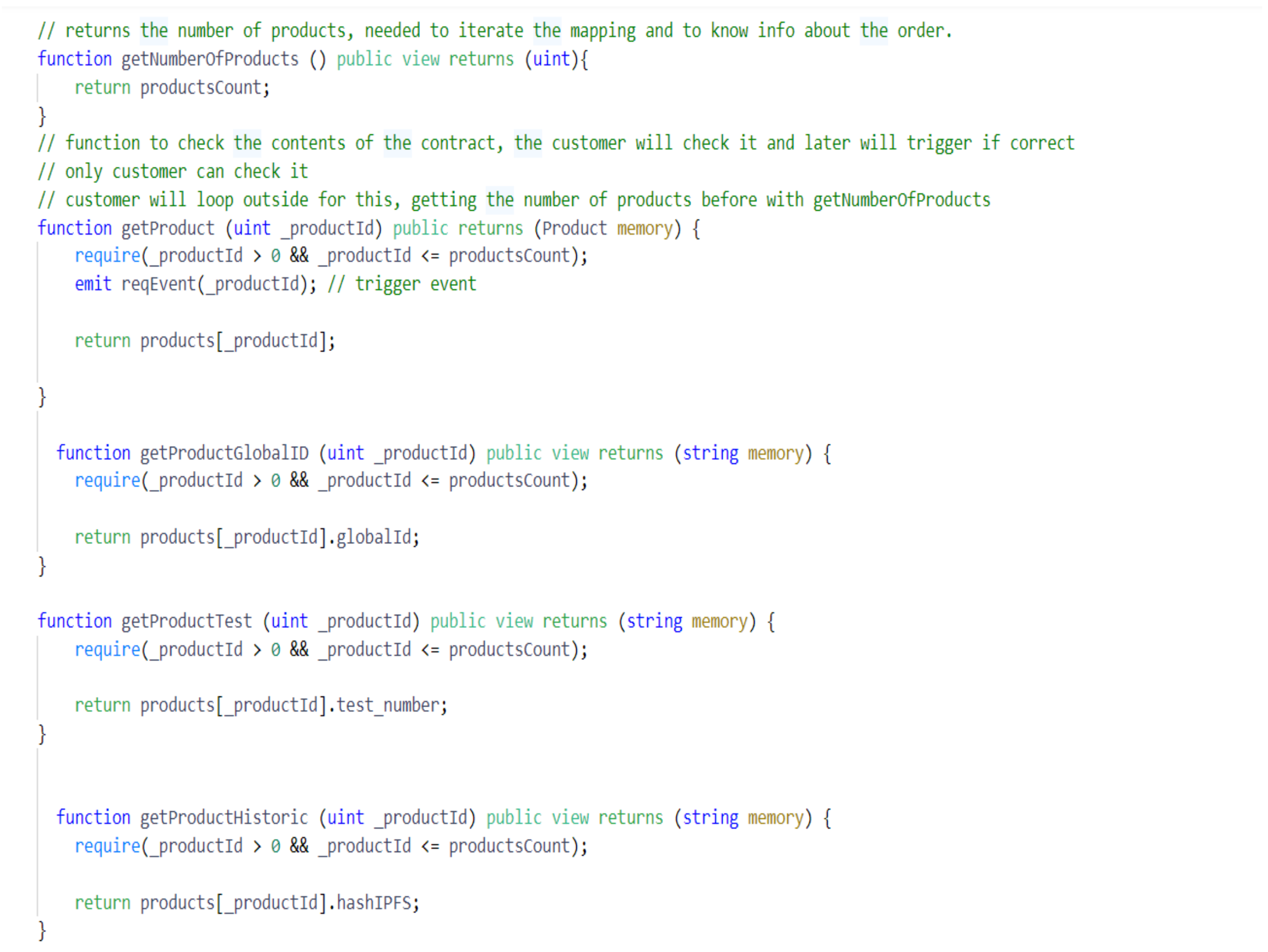
```
struct Trace {
    uint id;
    uint id_product;
    string location;
    string temp_owner; // stakeholder
    uint timestamp;
    address maker; // who  updates
}

mapping(uint => Trace) private traces;
//stores products count
// since mappings cant be looped and is difficult the have a count like array
// we need a var to store the countings
// useful also to iterate the mapping


struct Temperature {  // we use celsious
    uint id;
    uint id_product;
    uint celsius; // the number
    uint timestamp;
    address maker; // who  updates
}

mapping(uint => Temperature) private temperatures;


//since mapping won't have the count, we declare variables to store the count
uint public productsCount;
uint public tracesCount;
uint public temperaturesCount;

// smart comm to smart comm statistics
uint public globalnumberstakeholders;
uint public globalnumberProcesses;
```



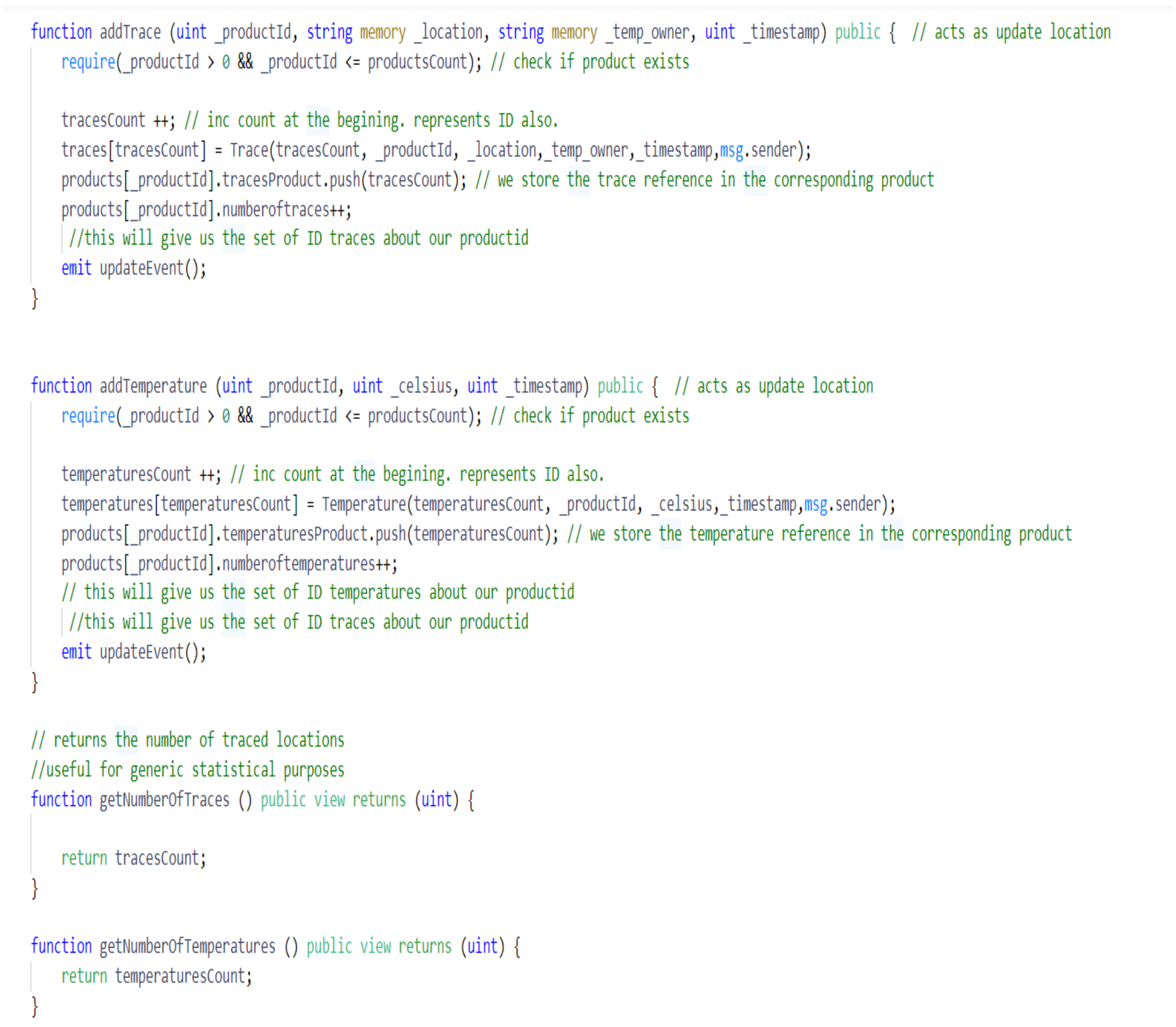
```
address public prodAdmin;

constructor ()  { // constructor, creates order. we map starting from id=1,  hardcoded values of all
    prodAdmin = msg.sender;
    addProduct("Milk",200, "Cow milk, from cow ID C234235B","00011A","QmdUvd3tb4TRM4pwMrLYEmgo7JDtwxnwXB1Wfa3gtgnwW1"); //
    addProduct("Yogurt",50, "Cow milk, from cow ID D29935G","00534A","QmdUvd3tb4TRM4pwMrLYEmgo7JDtwxnwXB1Wfa3gtgnwW1");
    addProduct("Graviera cCheese",10, "Cow milk, from cow ID C22225F","00442B","QmdUvd3tb4TRM4pwMrLYEmgo7JDtwxnwXB1Wfa3gtgnwW1");
    addProduct("Graviera cheese",10, "Cow milk, from cow ID 000235B","00463B","QmdUvd3tb4TRM4pwMrLYEmgo7JDtwxnwXB1Wfa3gtgnwW1");
    addProduct("Feta cheese",150, "Cow milk, from cow ID 000336F","00333E","QmdUvd3tb4TRM4pwMrLYEmgo7JDtwxnwXB1Wfa3gtgnwW1");
    addProduct("Milk",200, "Cow milk, from cow ID 001237F","00123A","QmdUvd3tb4TRM4pwMrLYEmgo7JDtwxnwXB1Wfa3gtgnwW1");
    addTrace(1,"some coordinates", "name or address of actual owner",1573564413); // locations of the product. to be updated by corresponding stakheolder.
    //addComponent();
    triggered=false;
    delivery=false;
    received=false;


}


//PRODUCT OPERATIONS*****************************************
// enables product creation
// get product
// get total for externally looping the mapping
// update others.

// add product to mapping. private because we dont want it to be accesible or add products afterwards to our mapping. We only want
// our contract to be able to do that, from constructor
// otherwise the conditions of the accepted contract could change
//only admin and stakeholder can do it
function addProduct (string memory _name, uint _quantity, string memory _description, string memory _globalID, string memory _hashIpfs) private {
    //require(msg.sender==stakholder);
    //require(food_pagonis.checkStakeholder(address addr, address stake)==true); // if valid stakeholder

    productsCount ++; // inc count at the begining. represents ID also.
```



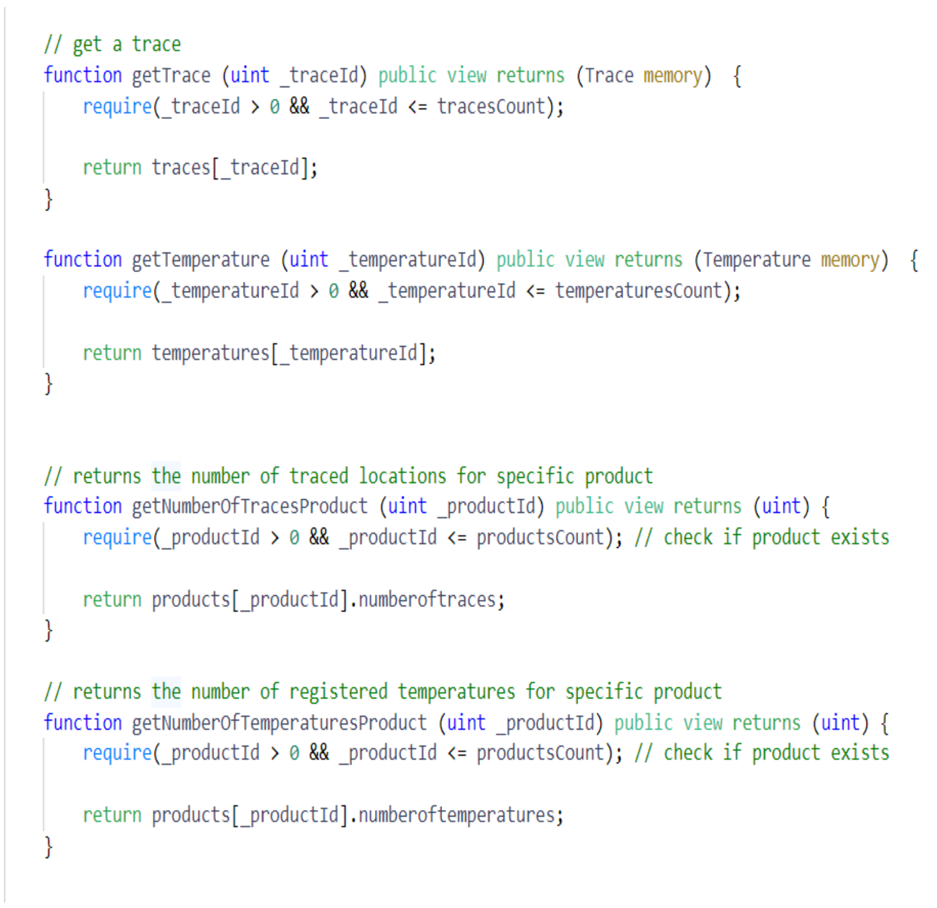
```
    products[productsCount].id = productsCount;
    products[productsCount].name = _name;
    products[productsCount].quantity = _quantity;
    products[productsCount].description = _description;
    products[productsCount].numberoftraces = 0;
    products[productsCount].numberoftemperatures = 0;
    products[productsCount].maker = prodAdmin;
    products[productsCount].globalId = _globalID;
    products[productsCount].hashIPFS = _hashIpfs;
    // reference the mapping with the key (that is the count). We assign the value to
    // the mapping, the count will be the ID.
}



 // only specific stakeholders, can be changed
function UpdateProductDescription (uint _productId, string memory _description) public {
    //require(food_pagonis.checkStakeholder(address addr, address stake)==true); // if valid stakeholder
    require(_productId > 0 && _productId <= productsCount);

    products[_productId].description = _description;  // update conditions
    emit updateEvent(); // trigger event
}

function addTestProduct (uint _productId, string memory _test_number) public {
    require(_productId > 0 && _productId <= productsCount);

    products[_productId].test_number = _test_number;  // update conditions
    emit updateEvent(); // trigger event
}
```



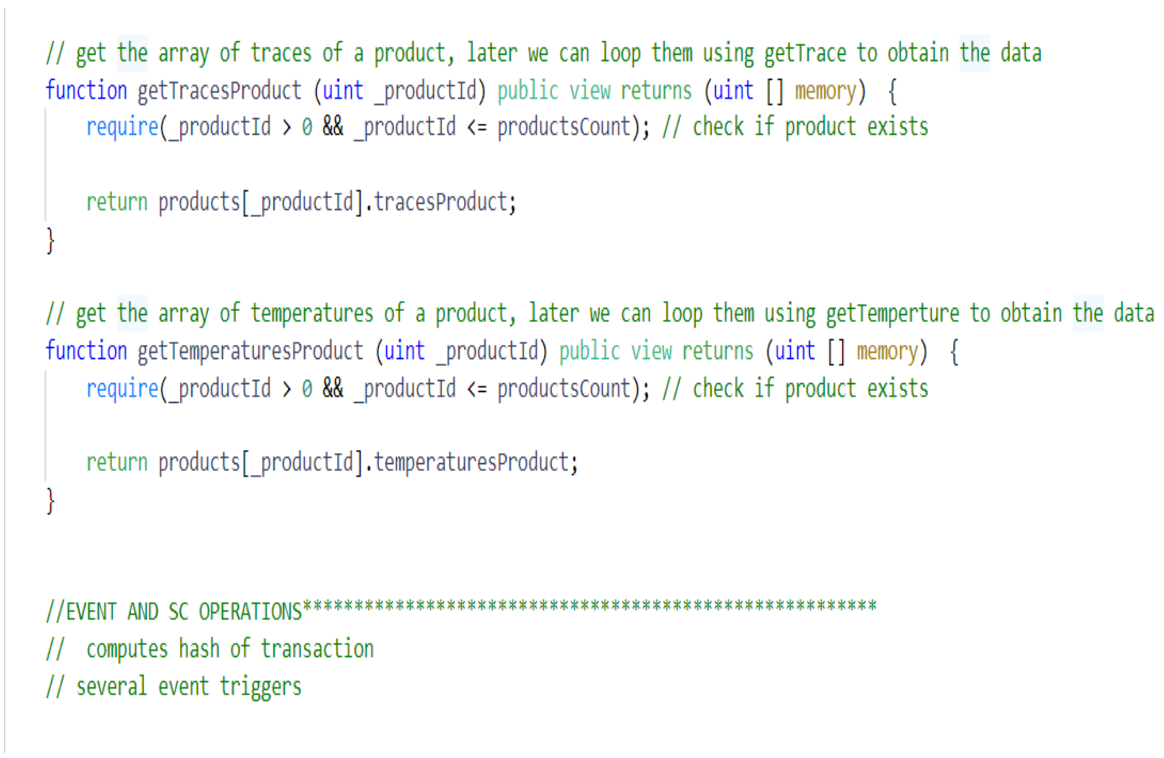
```
// returns the number of products, needed to iterate the mapping and to know info about the order.
function getNumberOfProducts () public view returns (uint){
    return productsCount;
}
// function to check the contents of the contract, the customer will check it and later will trigger if correct
// only customer can check it
// customer will loop outside for this, getting the number of products before with getNumberOfProducts
function getProduct (uint _productId) public returns (Product memory) {
    require(_productId > 0 && _productId <= productsCount);
    emit reqEvent(_productId); // trigger event

    return products[_productId];

}

  function getProductGlobalID (uint _productId) public view returns (string memory) {
    require(_productId > 0 && _productId <= productsCount);

    return products[_productId].globalId;
}

function getProductTest (uint _productId) public view returns (string memory) {
    require(_productId > 0 && _productId <= productsCount);

    return products[_productId].test_number;
}


  function getProductHistoric (uint _productId) public view returns (string memory) {
    require(_productId > 0 && _productId <= productsCount);

    return products[_productId].hashIPFS;
}
```



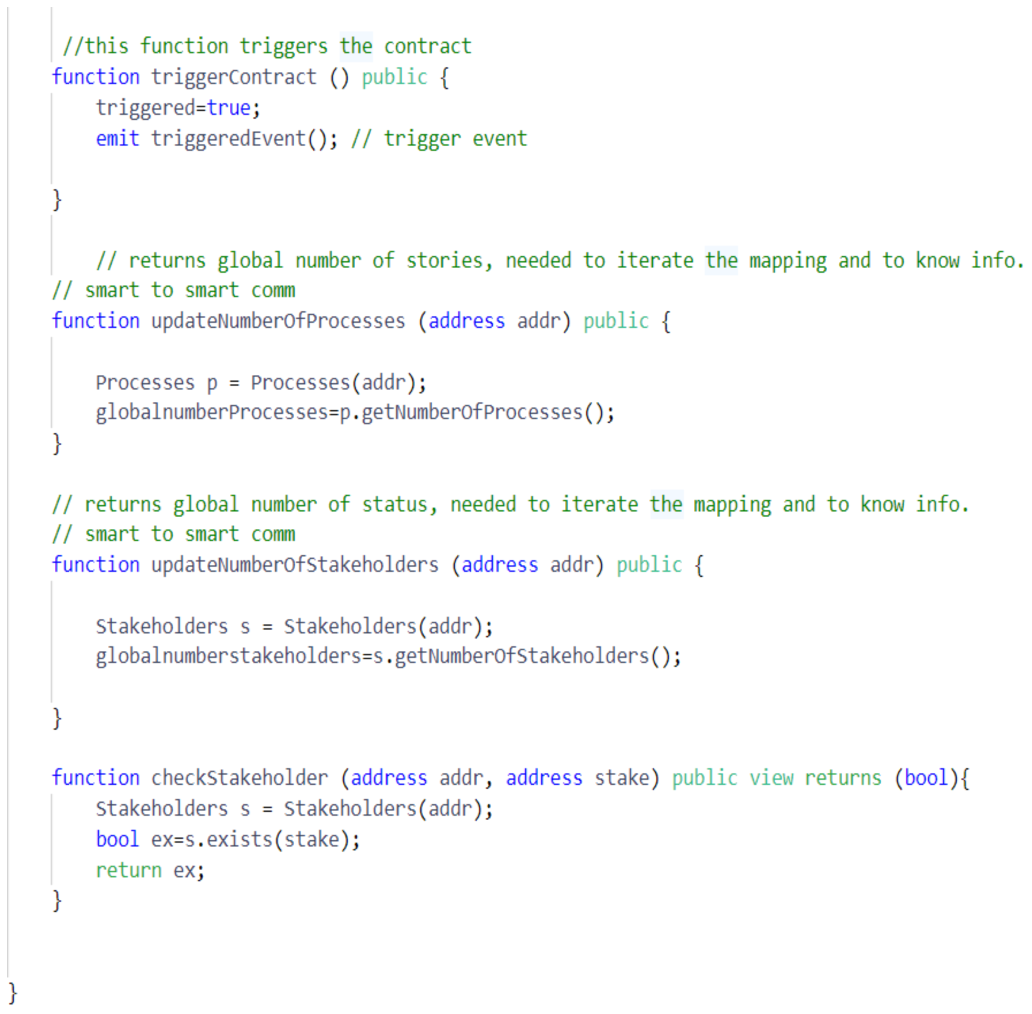
```
function addTrace (uint _productId, string memory _location, string memory _temp_owner, uint _timestamp) public {  // acts as update location
    require(_productId > 0 && _productId <= productsCount); // check if product exists

    tracesCount ++; // inc count at the begining. represents ID also.
    traces[tracesCount] = Trace(tracesCount, _productId, _location,_temp_owner,_timestamp,msg.sender);
    products[_productId].tracesProduct.push(tracesCount); // we store the trace reference in the corresponding product
    products[_productId].numberoftraces++;
     //this will give us the set of ID traces about our productid
    emit updateEvent();
}


function addTemperature (uint _productId, uint _celsius, uint _timestamp) public {  // acts as update location
    require(_productId > 0 && _productId <= productsCount); // check if product exists

    temperaturesCount ++; // inc count at the begining. represents ID also.
    temperatures[temperaturesCount] = Temperature(temperaturesCount, _productId, _celsius,_timestamp,msg.sender);
    products[_productId].temperaturesProduct.push(temperaturesCount); // we store the temperature reference in the corresponding product
    products[_productId].numberoftemperatures++;
    // this will give us the set of ID temperatures about our productid
     //this will give us the set of ID traces about our productid
    emit updateEvent();
}

// returns the number of traced locations
//useful for generic statistical purposes
function getNumberOfTraces () public view returns (uint) {

    return tracesCount;
}

function getNumberOfTemperatures () public view returns (uint) {
    return temperaturesCount;
}
```



```
// get a trace
function getTrace (uint _traceId) public view returns (Trace memory)  {
    require(_traceId > 0 && _traceId <= tracesCount);

    return traces[_traceId];
}

function getTemperature (uint _temperatureId) public view returns (Temperature memory)  {
    require(_temperatureId > 0 && _temperatureId <= temperaturesCount);

    return temperatures[_temperatureId];
}


// returns the number of traced locations for specific product
function getNumberOfTracesProduct (uint _productId) public view returns (uint) {
    require(_productId > 0 && _productId <= productsCount); // check if product exists

    return products[_productId].numberoftraces;
}

// returns the number of registered temperatures for specific product
function getNumberOfTemperaturesProduct (uint _productId) public view returns (uint) {
    require(_productId > 0 && _productId <= productsCount); // check if product exists

    return products[_productId].numberoftemperatures;
}
```

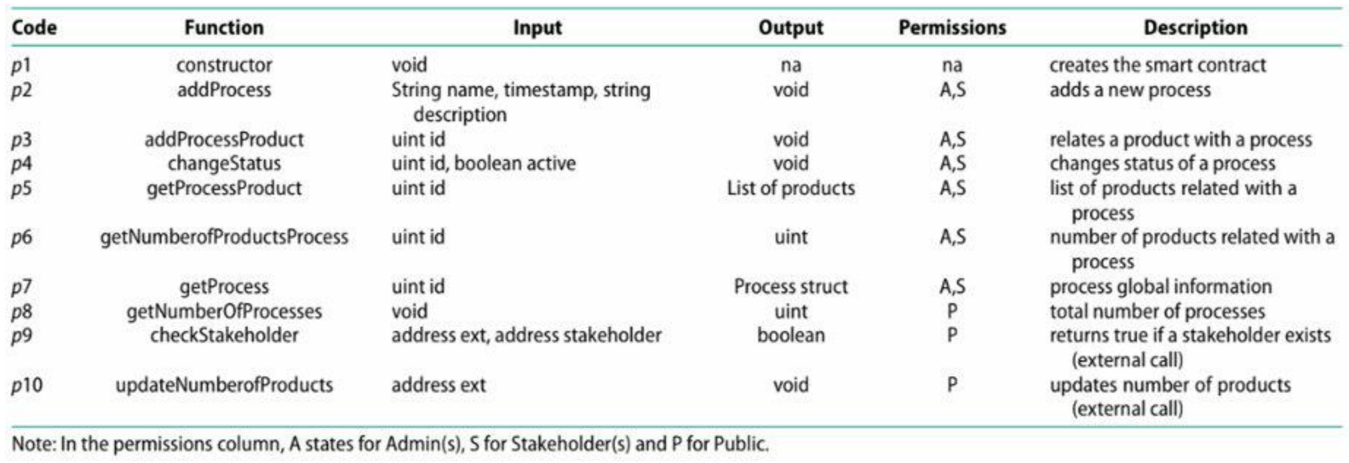


```
// get the array of traces of a product, later we can loop them using getTrace to obtain the data
function getTracesProduct (uint _productId) public view returns (uint [] memory)  {
    require(_productId > 0 && _productId <= productsCount); // check if product exists

    return products[_productId].tracesProduct;
}

// get the array of temperatures of a product, later we can loop them using getTemperture to obtain the data
function getTemperaturesProduct (uint _productId) public view returns (uint [] memory)  {
    require(_productId > 0 && _productId <= productsCount); // check if product exists

    return products[_productId].temperaturesProduct;
}


//EVENT AND SC OPERATIONS***********************************************************
//  computes hash of transaction
// several event triggers
```



```
     //this function triggers the contract
    function triggerContract () public {
        triggered=true;
        emit triggeredEvent(); // trigger event

    }

        // returns global number of stories, needed to iterate the mapping and to know info.
    // smart to smart comm
    function updateNumberOfProcesses (address addr) public {

        Processes p = Processes(addr);
        globalnumberProcesses=p.getNumberOfProcesses();
    }

    // returns global number of status, needed to iterate the mapping and to know info.
    // smart to smart comm
    function updateNumberOfStakeholders (address addr) public {

        Stakeholders s = Stakeholders(addr);
        globalnumberstakeholders=s.getNumberOfStakeholders();

    }

    function checkStakeholder (address addr, address stake) public view returns (bool){
        Stakeholders s = Stakeholders(addr);
        bool ex=s.exists(stake);
        return ex;
    }


}
```

2) Processes.sol:

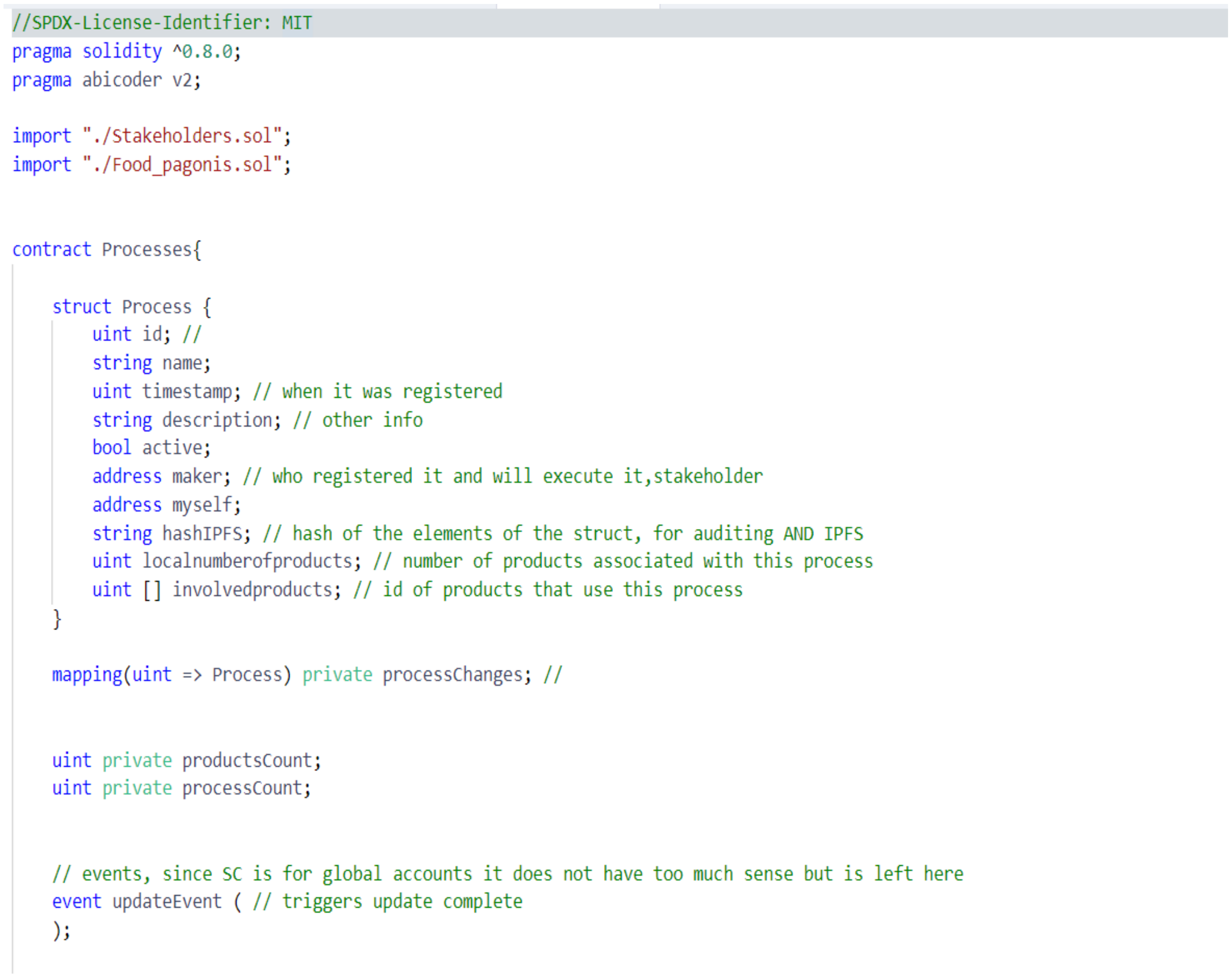


Code	Function	Input	Output	Permissions	Description
*p*1	constructor	void	na	na	creates the smart contract
*p*2	addProcess	String name, timestamp, string description	void	A,S	adds a new process
*p*3	addProcessProduct	uint id	void	A,S	relates a product with a process
*p*4	changeStatus	uint id, boolean active	void	A,S	changes status of a process
*p*5	getProcessProduct	uint id	List of products	A,S	list of products related with a process
*p*6	getNumberofProductsProcess	uint id	uint	A,S	number of products related with a process
*p*7	getProcess	uint id	Process struct	A,S	process global information
*p*8	getNumberOfProcesses	void	uint	P	total number of processes
*p*9	checkStakeholder	address ext, address stakeholder	boolean	P	returns true if a stakeholder exists (external call)
*p*10	updateNumberofProducts	address ext	void	P	updates number of products (external call)

Note: In the permissions column, A states for Admin(s), S for Stakeholder(s) and P for Public.

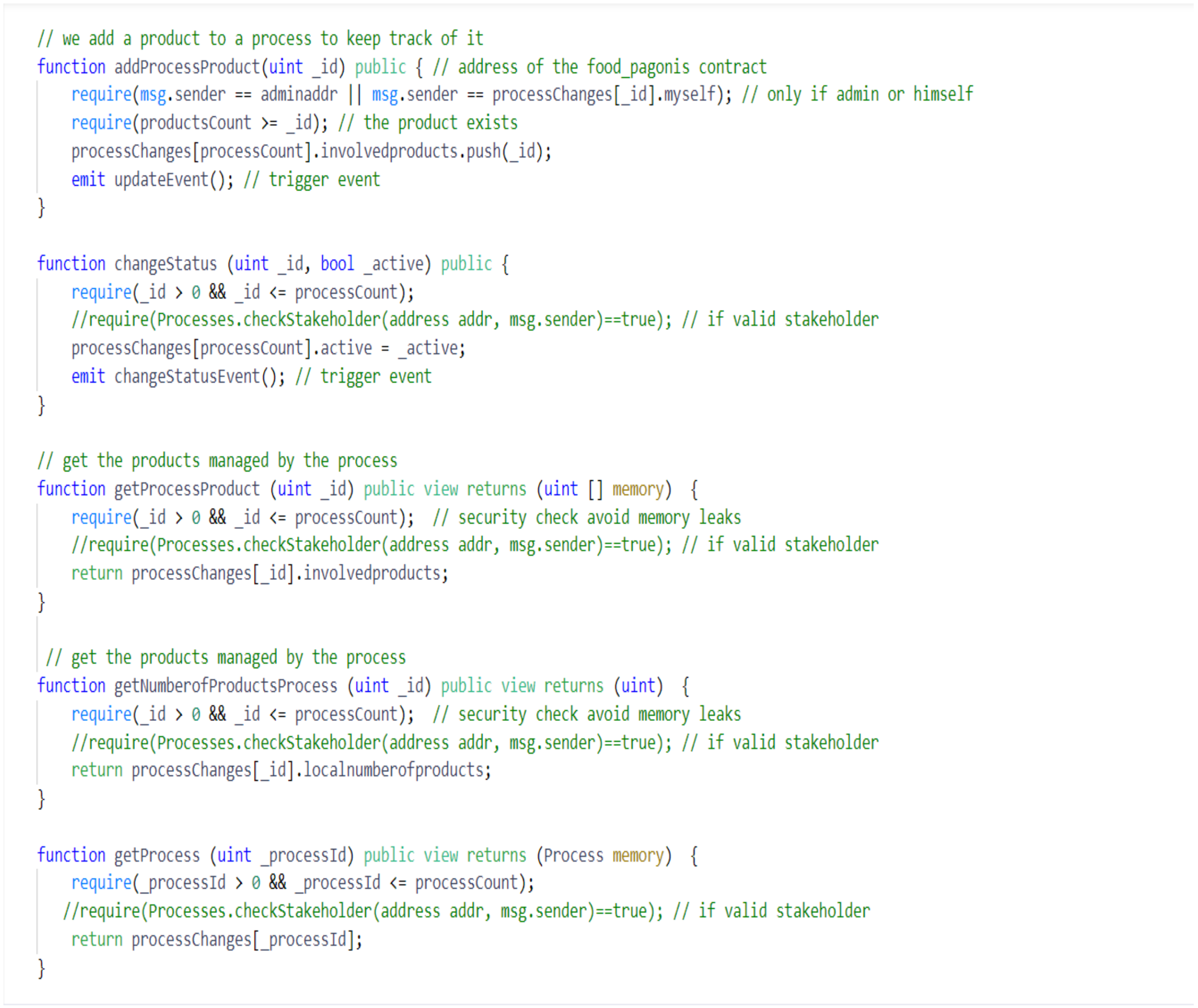
Table 3.2 This table shows all the functions that are implemented in "Processes" smart contract and also the permissions given to each function.

Code:

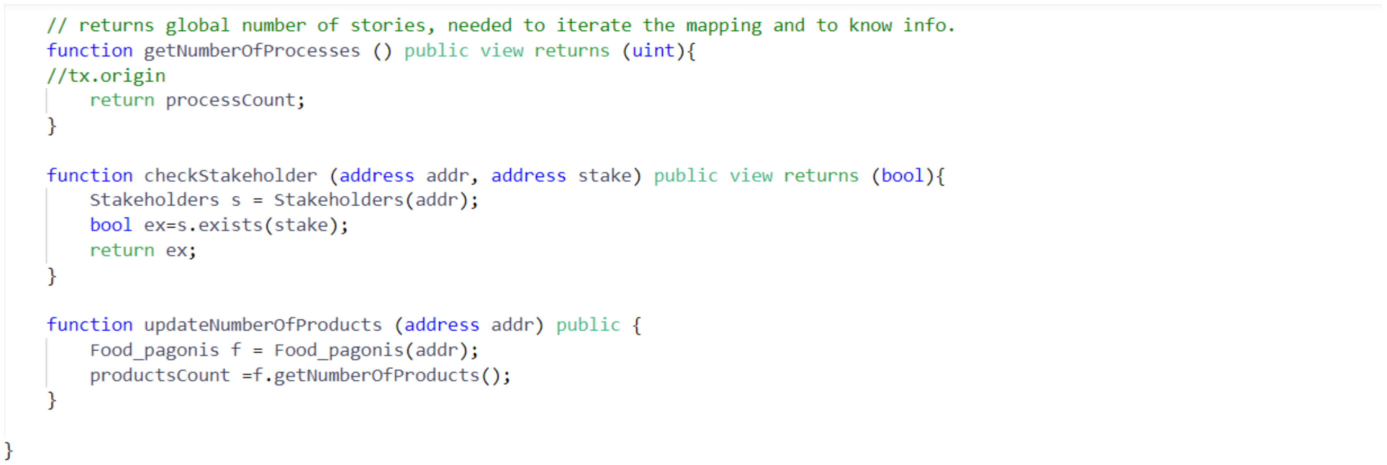


```solidity
//SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;
pragma abicoder v2;

import "./Stakeholders.sol";
import "./Food_pagonis.sol";


contract Processes{

    struct Process {
        uint id; //
        string name;
        uint timestamp; // when it was registered
        string description; // other info
        bool active;
        address maker; // who registered it and will execute it,stakeholder
        address myself;
        string hashIPFS; // hash of the elements of the struct, for auditing AND IPFS
        uint localnumberofproducts; // number of products associated with this process
        uint [] involvedproducts; // id of products that use this process
    }

    mapping(uint => Process) private processChanges; //


    uint private productsCount;
    uint private processCount;


    // events, since SC is for global accounts it does not have too much sense but is left here
    event updateEvent ( // triggers update complete
    );
```

```
event changeStatusEvent ( // triggers status change
);

address constant public stakeholder = 0xE0f5206BBD039e7b0592d8918820024e2a7437b9; // who registers the token into system.
//address constant public adminaddr = 0xE0F5206bbd039e7b0592d8918820024E2A743222; // admin, current

address public adminaddr;

constructor ()  { // constructor, inserts new process in system. we map starting from id=1, hardcoded values of all
    adminaddr = msg.sender;
    addProcess("Mixing",1573564413,"Mix milk with L.bacteria"); // simple example.

}

// add process
function addProcess (string memory _name, uint _timestamp, string memory _description) public {
    //require(Processes.checkStakeholder(address addr, msg.sender)==true); // if valid stakeholder
    processCount++;

    processChanges[processCount].id = processCount;
    processChanges[processCount].name = _name;
    processChanges[processCount].timestamp = _timestamp;
    processChanges[processCount].description = _description;
    processChanges[processCount].active = true;
    processChanges[processCount].localnumberofproducts=0;
    processChanges[processCount].maker = adminaddr;
    processChanges[processCount].myself = msg.sender;
    emit updateEvent(); // trigger event

}
```

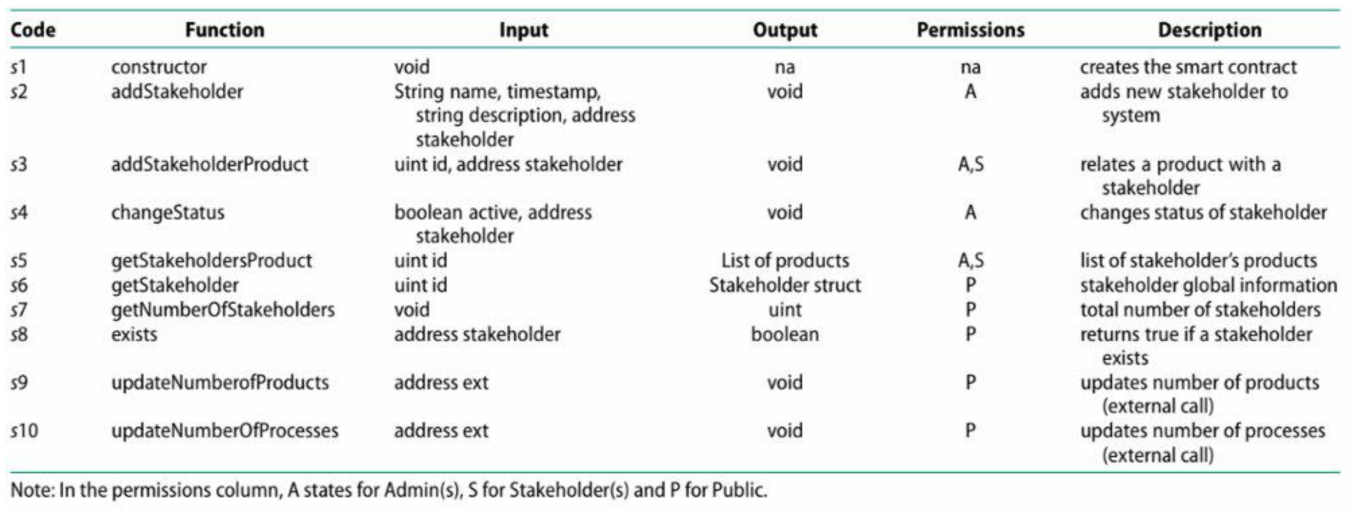


```
// we add a product to a process to keep track of it
function addProcessProduct(uint _id) public { // address of the food_pagonis contract
    require(msg.sender == adminaddr || msg.sender == processChanges[_id].myself); // only if admin or himself
    require(productsCount >= _id); // the product exists
    processChanges[processCount].involvedproducts.push(_id);
    emit updateEvent(); // trigger event
}

function changeStatus (uint _id, bool _active) public {
    require(_id > 0 && _id <= processCount);
    //require(Processes.checkStakeholder(address addr, msg.sender)==true); // if valid stakeholder
    processChanges[processCount].active = _active;
    emit changeStatusEvent(); // trigger event
}

// get the products managed by the process
function getProcessProduct (uint _id) public view returns (uint [] memory)  {
    require(_id > 0 && _id <= processCount);  // security check avoid memory leaks
    //require(Processes.checkStakeholder(address addr, msg.sender)==true); // if valid stakeholder
    return processChanges[_id].involvedproducts;
}

 // get the products managed by the process
function getNumberofProductsProcess (uint _id) public view returns (uint)  {
    require(_id > 0 && _id <= processCount);  // security check avoid memory leaks
    //require(Processes.checkStakeholder(address addr, msg.sender)==true); // if valid stakeholder
    return processChanges[_id].localnumberofproducts;
}

function getProcess (uint _processId) public view returns (Process memory)  {
    require(_processId > 0 && _processId <= processCount);
   //require(Processes.checkStakeholder(address addr, msg.sender)==true); // if valid stakeholder
    return processChanges[_processId];
}
```



```
    // returns global number of stories, needed to iterate the mapping and to know info.
    function getNumberOfProcesses () public view returns (uint){
    //tx.origin
        return processCount;
    }

    function checkStakeholder (address addr, address stake) public view returns (bool){
        Stakeholders s = Stakeholders(addr);
        bool ex=s.exists(stake);
        return ex;
    }

    function updateNumberOfProducts (address addr) public {
        Food_pagonis f = Food_pagonis(addr);
        productsCount =f.getNumberOfProducts();
    }

}
```

3) Stakeholders.sol

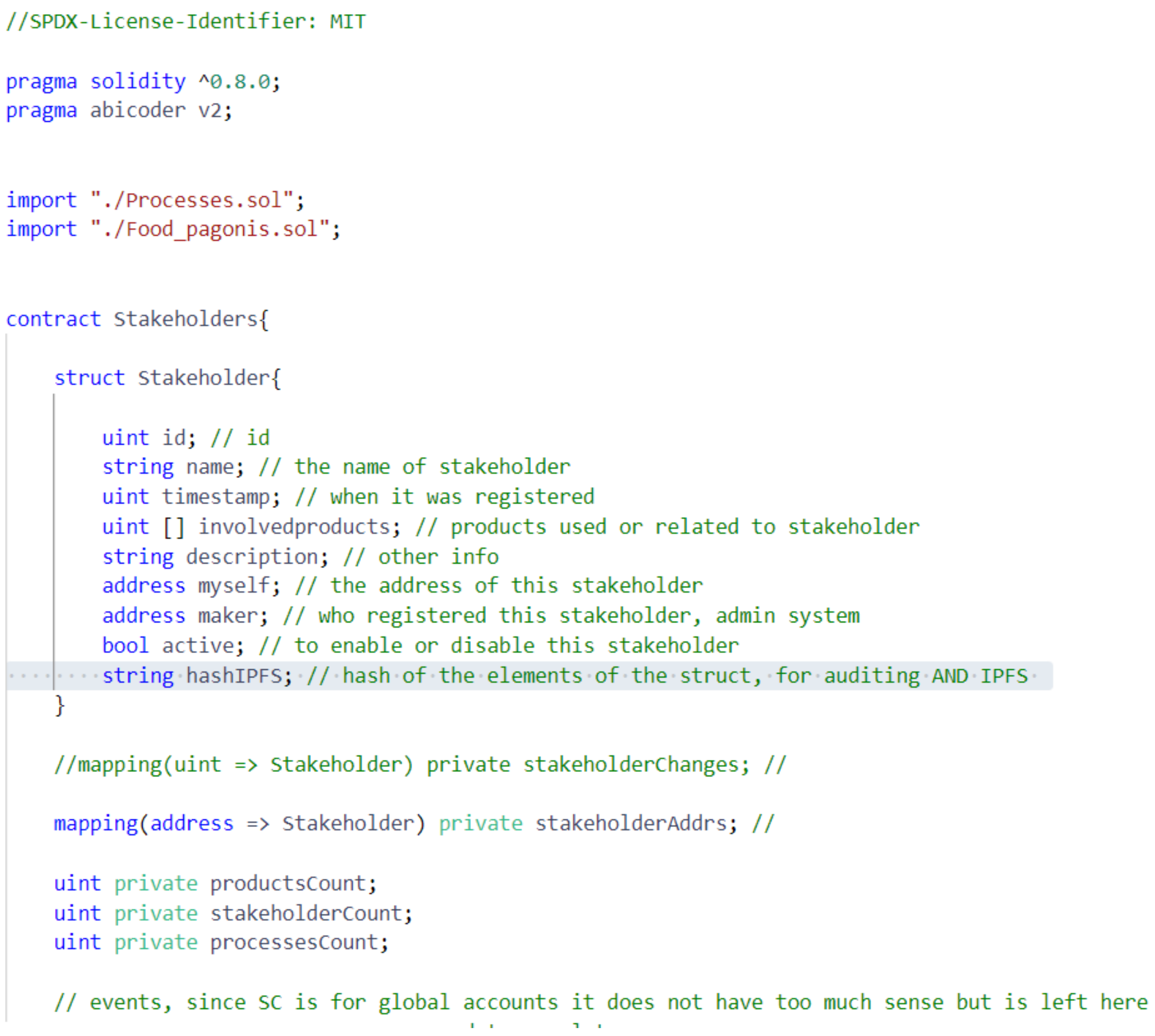


Code	Function	Input	Output	Permissions	Description
s1	constructor	void	na	na	creates the smart contract
s2	addStakeholder	String name, timestamp, string description, address stakeholder	void	A	adds new stakeholder to system
s3	addStakeholderProduct	uint id, address stakeholder	void	A,S	relates a product with a stakeholder
s4	changeStatus	boolean active, address stakeholder	void	A	changes status of stakeholder
s5	getStakeholdersProduct	uint id	List of products	A,S	list of stakeholder's products
s6	getStakeholder	uint id	Stakeholder struct	P	stakeholder global information
s7	getNumberOfStakeholders	void	uint	P	total number of stakeholders
s8	exists	address stakeholder	boolean	P	returns true if a stakeholder exists
s9	updateNumberofProducts	address ext	void	P	updates number of products (external call)
s10	updateNumberOfProcesses	address ext	void	P	updates number of processes (external call)

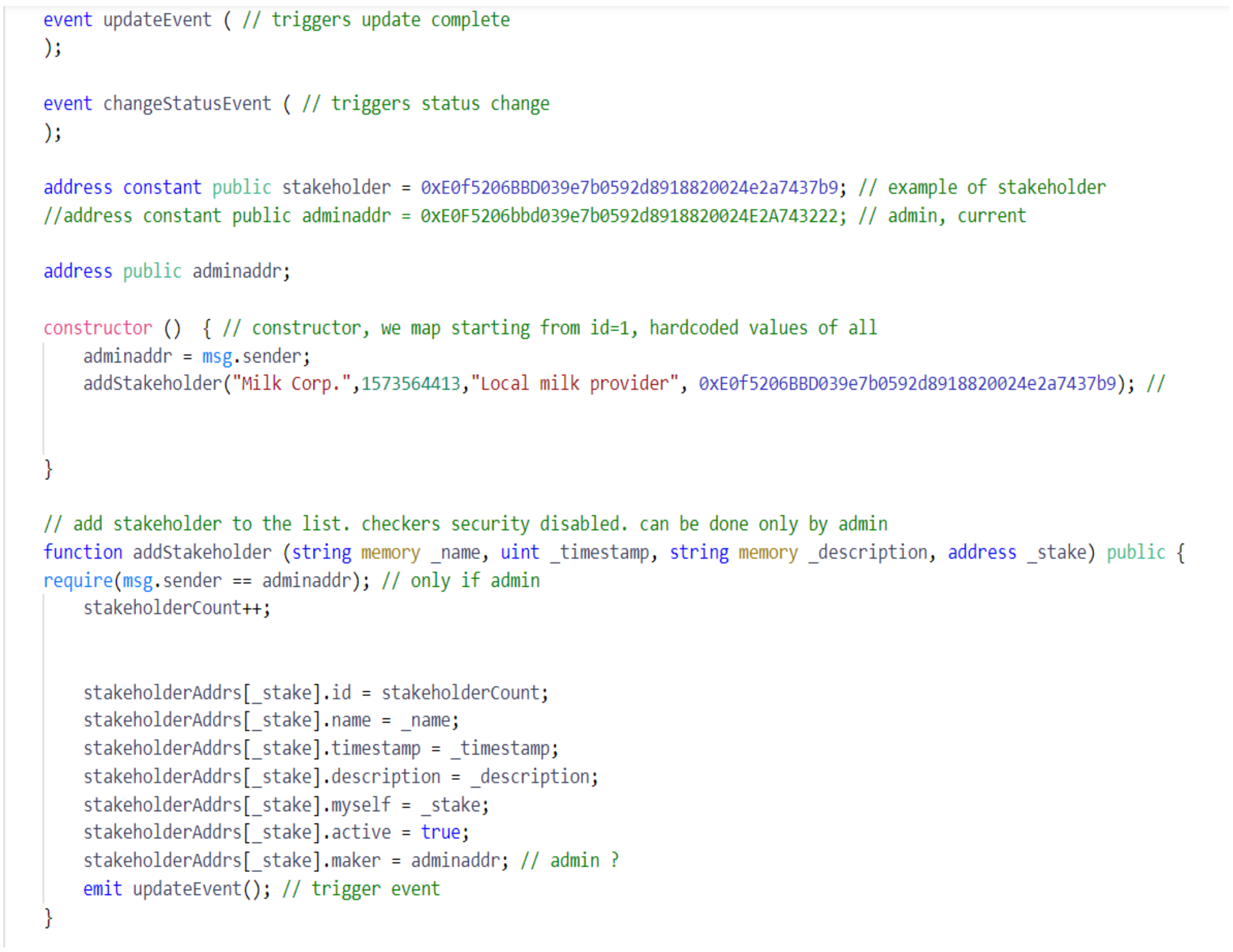
Note: In the permissions column, A states for Admin(s), S for Stakeholder(s) and P for Public.

3.3 The above table shows all the functions that are implemented in "Stakeholders" smart contract and also the permission given to each function.

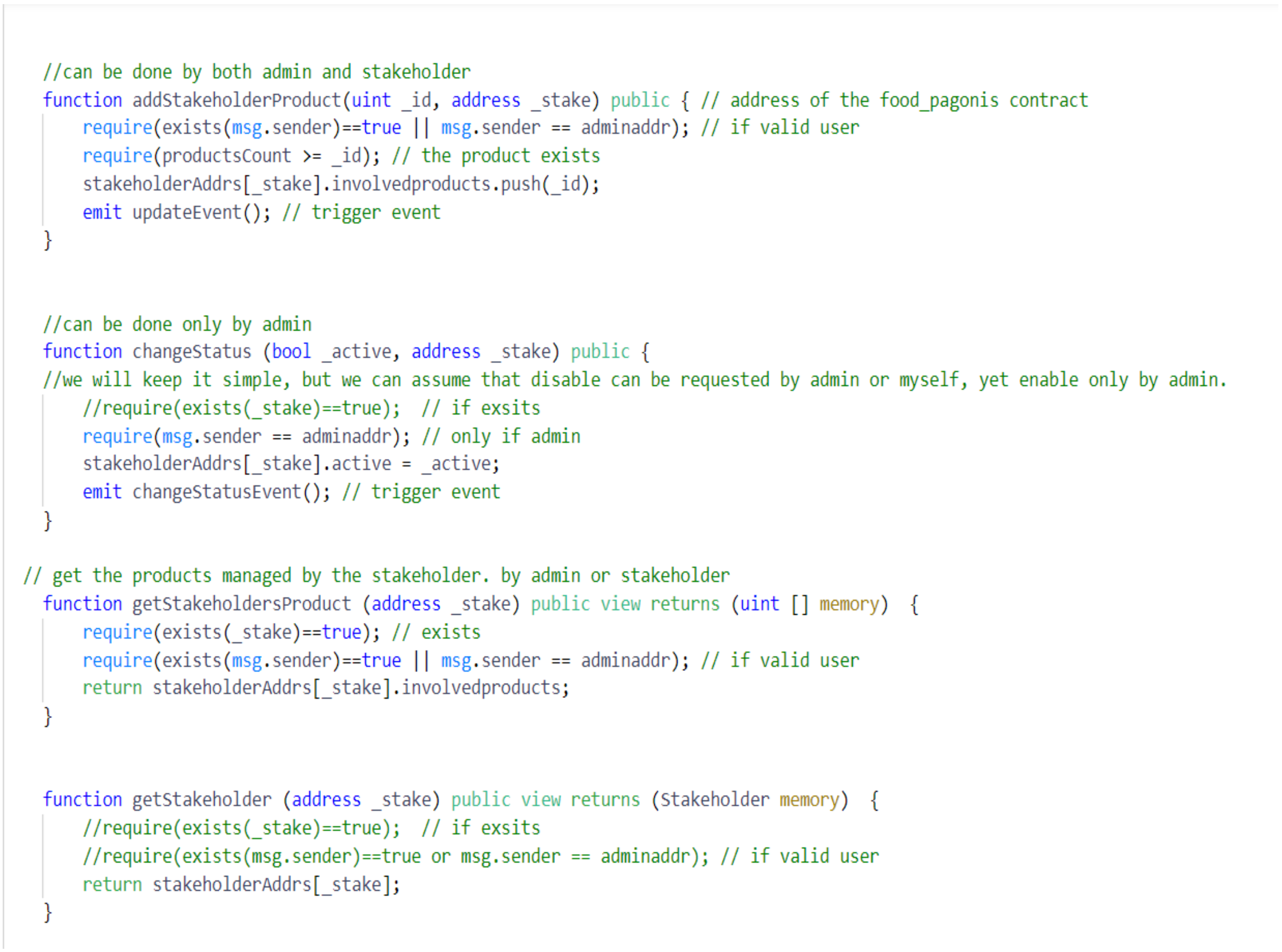
Code:



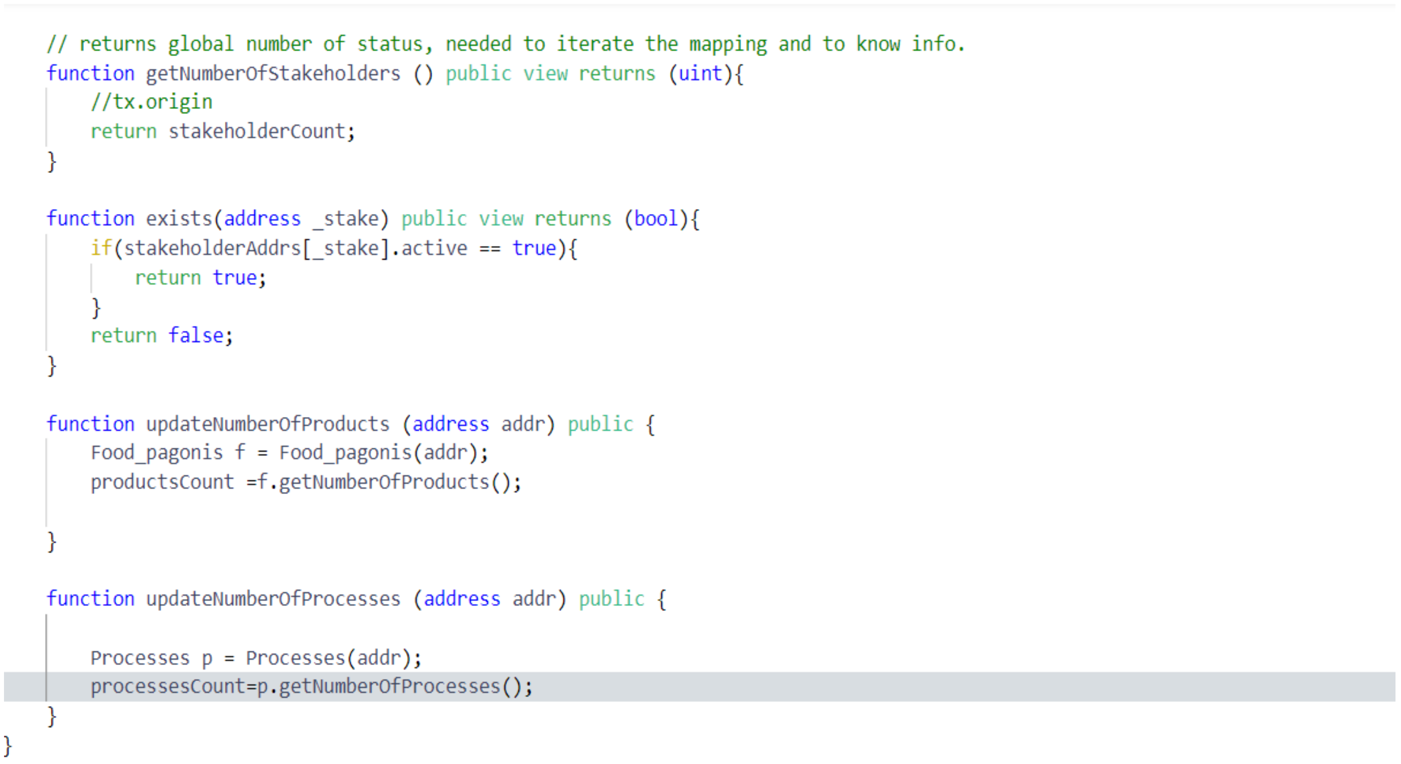
```
//SPDX-License-Identifier: MIT

pragma solidity ^0.8.0;
pragma abicoder v2;


import "./Processes.sol";
import "./Food_pagonis.sol";


contract Stakeholders{

    struct Stakeholder{

        uint id; // id
        string name; // the name of stakeholder
        uint timestamp; // when it was registered
        uint [] involvedproducts; // products used or related to stakeholder
        string description; // other info
        address myself; // the address of this stakeholder
        address maker; // who registered this stakeholder, admin system
        bool active; // to enable or disable this stakeholder
        string hashIPFS; // hash of the elements of the struct, for auditing AND IPFS
    }

    //mapping(uint => Stakeholder) private stakeholderChanges; //

    mapping(address => Stakeholder) private stakeholderAddrs; //

    uint private productsCount;
    uint private stakeholderCount;
    uint private processesCount;

    // events, since SC is for global accounts it does not have too much sense but is left here
```



```
event updateEvent ( // triggers update complete
);

event changeStatusEvent ( // triggers status change
);

address constant public stakeholder = 0xE0f5206BBD039e7b0592d8918820024e2a7437b9; // example of stakeholder
//address constant public adminaddr = 0xE0F5206bbd039e7b0592d8918820024E2A743222; // admin, current

address public adminaddr;

constructor ()  { // constructor, we map starting from id=1, hardcoded values of all
    adminaddr = msg.sender;
    addStakeholder("Milk Corp.",1573564413,"Local milk provider", 0xE0f5206BBD039e7b0592d8918820024e2a7437b9); //


}

// add stakeholder to the list. checkers security disabled. can be done only by admin
function addStakeholder (string memory _name, uint _timestamp, string memory _description, address _stake) public {
require(msg.sender == adminaddr); // only if admin
    stakeholderCount++;


    stakeholderAddrs[_stake].id = stakeholderCount;
    stakeholderAddrs[_stake].name = _name;
    stakeholderAddrs[_stake].timestamp = _timestamp;
    stakeholderAddrs[_stake].description = _description;
    stakeholderAddrs[_stake].myself = _stake;
    stakeholderAddrs[_stake].active = true;
    stakeholderAddrs[_stake].maker = adminaddr; // admin ?
    emit updateEvent(); // trigger event
}
```



```
 //can be done by both admin and stakeholder
 function addStakeholderProduct(uint _id, address _stake) public { // address of the food_pagonis contract
     require(exists(msg.sender)==true || msg.sender == adminaddr); // if valid user
     require(productsCount >= _id); // the product exists
     stakeholderAddrs[_stake].involvedproducts.push(_id);
     emit updateEvent(); // trigger event
 }


 //can be done only by admin
 function changeStatus (bool _active, address _stake) public {
 //we will keep it simple, but we can assume that disable can be requested by admin or myself, yet enable only by admin.
     //require(exists(_stake)==true);  // if exsits
     require(msg.sender == adminaddr); // only if admin
     stakeholderAddrs[_stake].active = _active;
     emit changeStatusEvent(); // trigger event
 }

// get the products managed by the stakeholder. by admin or stakeholder
 function getStakeholdersProduct (address _stake) public view returns (uint [] memory)  {
     require(exists(_stake)==true); // exists
     require(exists(msg.sender)==true || msg.sender == adminaddr); // if valid user
     return stakeholderAddrs[_stake].involvedproducts;
 }


 function getStakeholder (address _stake) public view returns (Stakeholder memory)  {
     //require(exists(_stake)==true);  // if exsits
     //require(exists(msg.sender)==true or msg.sender == adminaddr); // if valid user
     return stakeholderAddrs[_stake];
 }
```



```
    // returns global number of status, needed to iterate the mapping and to know info.
    function getNumberOfStakeholders () public view returns (uint){
        //tx.origin
        return stakeholderCount;
    }

    function exists(address _stake) public view returns (bool){
        if(stakeholderAddrs[_stake].active == true){
            return true;
        }
        return false;
    }

    function updateNumberOfProducts (address addr) public {
        Food_pagonis f = Food_pagonis(addr);
        productsCount =f.getNumberOfProducts();

    }

    function updateNumberOfProcesses (address addr) public {

        Processes p = Processes(addr);
        processesCount=p.getNumberOfProcesses();
    }
}
```

CHAPTER 4

SNAPSHOTS

Table 4.1 Implementation specifications

Operating System	Windows 10
Programming Language	solidity
IDE	Visual Studio code and Remix
External Libraries	experimental ABIEncoderV2

deployed contracts:

Below Fig 4.1 shows the deployment of food Pagonis smart contract where it implements all the public functions attributes are displayed and can be interacted with them,

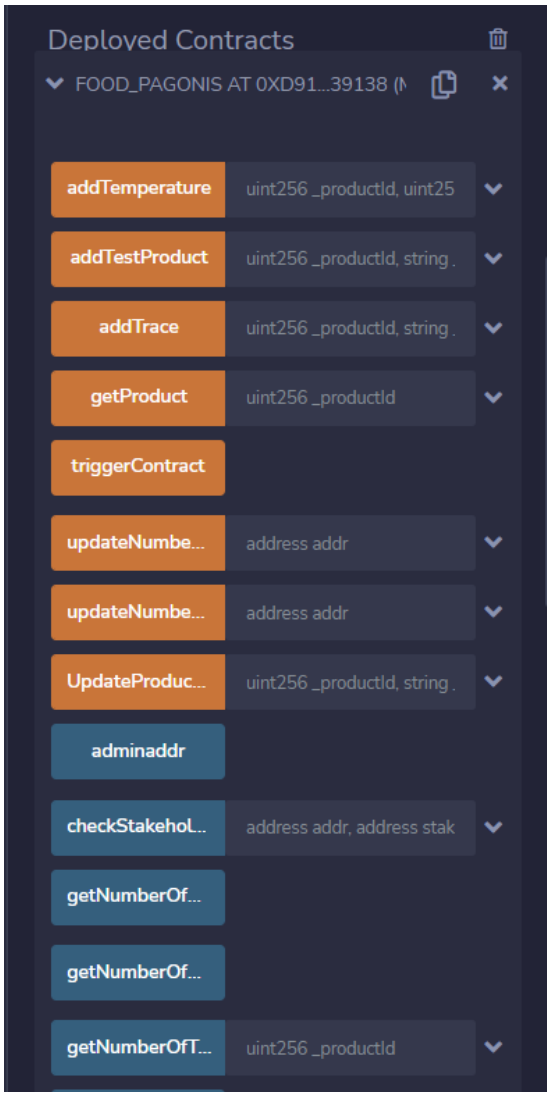
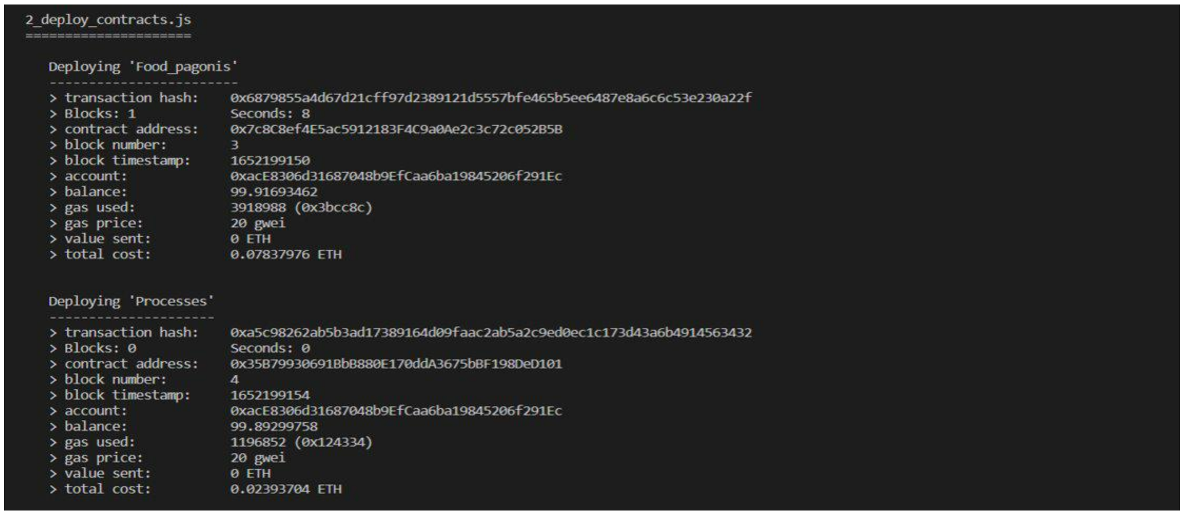


Fig 4.1 Deployment of food_pagonis smart contract

Next Fig 4.2 shows the successful deployment of Processes smart contract and all the public functions and attributes are displayed and can be interacted with them.



```
2_deploy_contracts.js
=====================

   Deploying 'Food_pagonis'
   ------------------------
   > transaction hash:    0x6879855a4d67d21cff97d2389121d5557bfe465b5ee6487e8a6c6c53e230a22f
   > Blocks: 1            Seconds: 8
   > contract address:    0x7c8C8ef4E5ac5912183F4C9a0Ae2c3c72c052B5B
   > block number:        3
   > block timestamp:     1652199150
   > account:             0xacE8306d31687048b9EfCaa6ba19845206f291Ec
   > balance:             99.91693462
   > gas used:            3918988 (0x3bcc8c)
   > gas price:           20 gwei
   > value sent:          0 ETH
   > total cost:          0.07837976 ETH


   Deploying 'Processes'
   ---------------------
   > transaction hash:    0xa5c98262ab5b3ad17389164d09faac2ab5a2c9ed0ec1c173d43a6b4914563432
   > Blocks: 0            Seconds: 0
   > contract address:    0x35B79930691BbB880E170ddA3675bBF198DeD101
   > block number:        4
   > block timestamp:     1652199154
   > account:             0xacE8306d31687048b9EfCaa6ba19845206f291Ec
   > balance:             99.89299758
   > gas used:            1196852 (0x124334)
   > gas price:           20 gwei
   > value sent:          0 ETH
   > total cost:          0.02393704 ETH
```

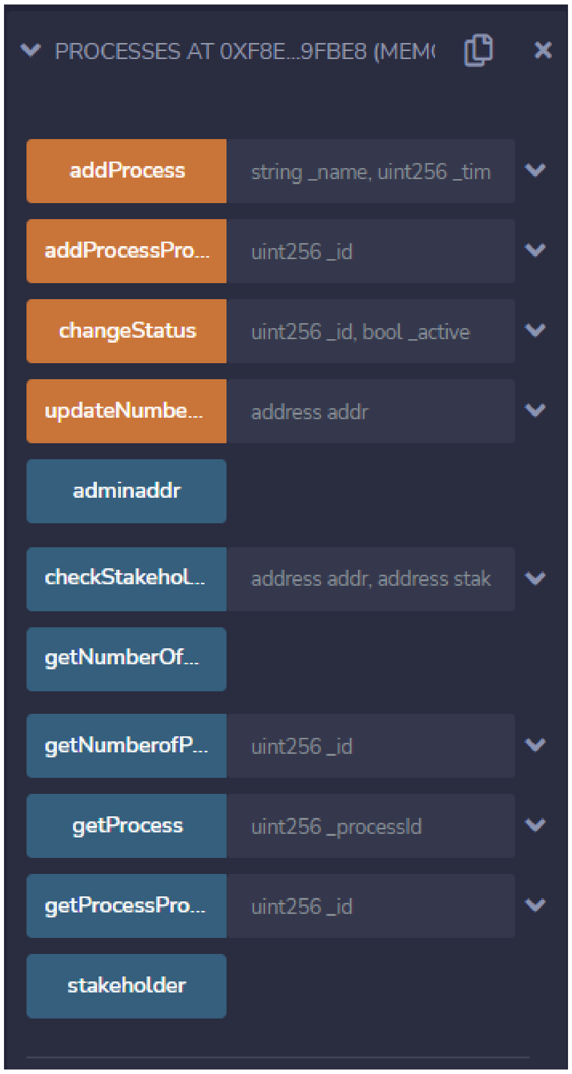
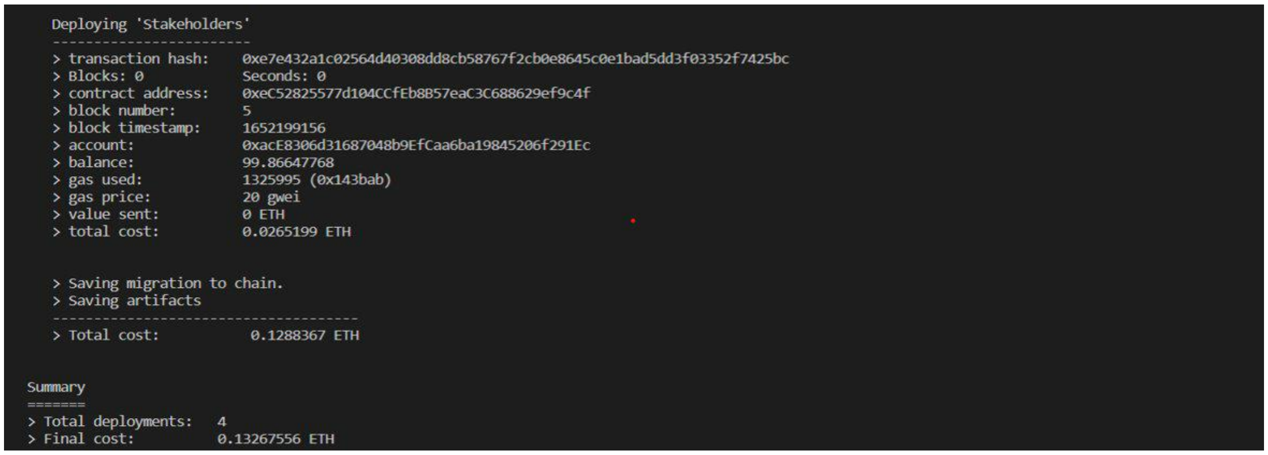


Fig 4.2 Deployment of processes smart contract

Finally, Fig 4.3 shows the deployment of Stakeholders smart contract and all the public functions and attributes are displayed and can be interacted with them.



```
   Deploying 'Stakeholders'
   ------------------------
   > transaction hash:    0xe7e432a1c02564d40308dd8cb58767f2cb0e8645c0e1bad5dd3f03352f7425bc
   > Blocks: 0            Seconds: 0
   > contract address:    0xeC52825577d104CCfEb8B57eaC3C688629ef9c4f
   > block number:        5
   > block timestamp:     1652199156
   > account:             0xacE8306d31687048b9EfCaa6ba19845206f291Ec
   > balance:             99.86647768
   > gas used:            1325995 (0x143bab)
   > gas price:           20 gwei
   > value sent:          0 ETH
   > total cost:          0.0265199 ETH


   > Saving migration to chain.
   > Saving artifacts
   -------------------------------------
   > Total cost:           0.1288367 ETH


Summary
=======
> Total deployments:   4
> Final cost:          0.13267556 ETH
```

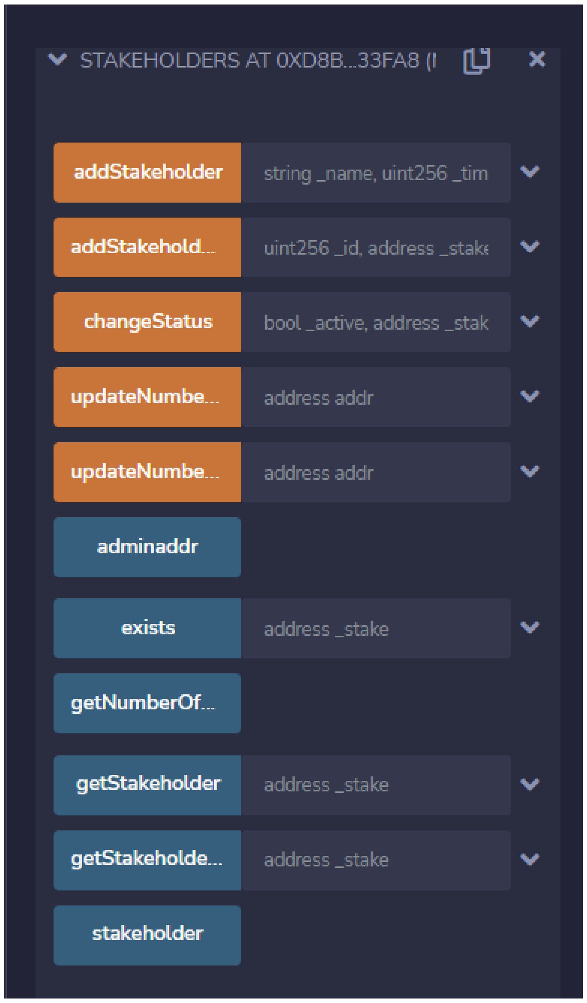


Fig 4.3 Deployment of Stakeholders smart contract

CHAPTER 5

CONCLUSIONS AND FUTURE PLANS

5.1 Conclusion

In this project an architecture for FSC traceability based on blockchain is described. All the traceability functions that are offered using this architecture, the different processes related to stakeholders and functions related to the dairy products that are involved, and also how they interact are all described. So, here we have used Smart Contracts and a private blockchain in order to automate the technique for managing the smart contracts like stakeholders, processes as well as for smart contract that gives product definition and creation. A distributed chain-of-custody system that is tamper-proof is provided by blockchain. The context and managerial requirements of a local dairy products supplier are defined using a real-world scenario. Three smart contracts namely Food Pagonis, Processes and Stakeholders that provide a set of services and communication enabled between them to facilitate an end-to-end traceability flow, from purchase of raw materials to the delivery of finished goods to end customers. Finally, these smart contracts are implemented in truffle with the use of a local private blockchain for the implementation.

5.2 Future plan

We have implemented this block chain technology and smart contracts in dairy products industries, as a future plan we can extend this work for other applications that that are more sensitive with security needs like health care sector, defense sector, etc., We can develop smart contracts for those industries and check the viability of proposed idea. More user-friendly environment to interact with the smart contracts will be developed by connecting it with the more interactive user interface and running it on networks like web3 and checking with private block chain.

CHAPTER 6

REFERENCES

1 Badia-Melis, R., P. Mishra, and L. Ruiz-García. 2015. “Food Traceability: New Trends and Recent Advances. A Review.” Food Control 57: 393–401.

2 Behnke, K., and M. F. W. H. A. Janssen. 2019. “Boundary Conditions for Traceability in Food Supply Chains Using Blockchain Technology.” International Journal of Information Management 52: 101969.

3 Casado-Vara, R, J Prieto, F. D. La Prieta, and J. M. Corchado. 2018. “How Blockchain Improves the Supply Chain: Case Study Alimentary Supply Chain.” Procedia Computer Science 134: 393–398.

4 Andreas Kamilaris, Agusti Fonts, Francesc X. Prenafeta-Boldúa. 2019. “The rise of blockchain technology in agriculture and food supply chains”.