# Provenance Engine

## Challenges and Solutions

By:

Gokul Narayan

Mohammed Sinphil

Sajin Abdu

## Contents

*In a nice day, you come to the supermarket, hold in your hand a pack of tomatoes. The succulent tomatoes look tasty with green leaves promising a good meal for your dinner. But, have you ever tried to get the address of the farm grew up the fresh and tasty tomatoes in your basket? Does it adapt the requirements of food standards? Have ever you questioned about what it says in the food labels is true? Although we are in the Information Age when your mind is flooded with the news from all around the globe, sometimes you face a lot of challenges to answer the simple given questions. But what is the root cause?*

## 1.0 Main challenges of tracking provenance.

Before reaching the end consumer, goods come across a long journey through a lot of factors. In each end-point of this journey, each retailer, each manufacturer has their own system with the different way of storing, tracking and circulating data. For example, the storage facility is running a system made by company A, the transporter is controlling the data management system by the product of company B. The compatibility in those centralized systems is not valued in its level of importance. Otherwise, the difference of specific progress makes the synchronization between two systems is impossible. If someone would dedicate their time to draw the picture of this product life-cycle from birth-to-death, they must dedicate an enormous amount of time with the huge workload to overcome a lot of obstacles in gathering, classifying and accumulating essential data. Consequently, surprisingly how little we know about the info of all goods we purchased. The unseen dimension of our possessions has still existed if the suppliers are insisted with the current information system meanwhile customers keep concentrating only the info in the wrap-page. Obviously, we need some changes.

Currently, some companies dedicated their resource to build their information repository that opens for customers to observe their vast network of sub-suppliers. However, all current systems are just focusing on the hard asset attributes with the boring scientific index or rough numbers about the industry standard what does not bring much significance to the understanding of normal customers. We must acknowledge that with a little knowledge about the food industry, we are hard to differentiate the advantages of Global G.A.P certification against the organic certification from USDA. All the terminologies or even financial figures are not more useful than the brand of producers. More than that, the customers demand simple metrics that directly relate to their value preferences.

The last challenge I would like to share on this blog may be illustrated after this short story. Most customers may know about the top favourite product of Apple – iPhone which is normally seen in the bright, clean and modern iStore where is full of happiness and well-mannered people. But only a few ones have ever heard about the investigation conducted by The Guardian (https://www.theguardian.com/technology/2017/jun/18/foxconn-life-death-forbidden-city-longhua-suicide-apple-iphone-brian-merchant-one-device-extract) that tells you about the sad story behind the lovely and cutting-edge products made from the assembly-line workers who have begun killing themselves since 2010. The given article reports that 18 reported suicide attempts with 14 confirmed deaths, depression and suicide have become normalised in the sprawling factory where more than 1.3 million workers are working daily. The life condition of these pitiable workers is controversial but the numbers never lie. Obviously, they are not happy or at least have the positive attitude with their work. Once the figure about the working hours or average salary gets the failure to reveal the secret of these poor workers, people are eager to find the alternative approach which incorporates not only the hard asset criteria but also soft asset information.

## 2.0 Social Earnings Ratio (SER)

Within the financial world, the Price/Earnings Ratio (P/E Ratio or PER) is the main single index evaluation technique of economic performance and health check. Used throughout the world, and reported daily on stock exchanges, it has become the universally adopted tool. In rough terms, the P/E measures the capitalization of a company (what it's worth) against the amount of profit it generates. We have developed the corollary to the P/E, the S/E Ratio (SER) which measures the Social Impact of an organisation against the monies invested to achieve it (the earnings diverted which otherwise would be profit).

A share price is a relatively weak artefact representing degrees of alignment between shareholders and the organisations. It is subject to numerous reported and unmeasured biases which mean it cannot in isolation reflect the true "value" of the corporate entity. Generally, only the shareholders and management benefit, and all others are a slave to that goal. Using human dynamic and descriptive analytics we have devised academically rigorous and peer reviewed methodologies which extend beyond these simple financial metrics.

Think of yourself as an operator within an ever-changing environment, the impacts you make upon the world around you will change constantly as the environment evolves. Your impact as a human being is not simply a constituent of your productivity or ability to make money.The social impact interpretation is that of citizenship, that the total value of an organisation is a total of the financial and social values. Taking social value, one must measure the stakeholder value of customers, suppliers, staff, statutory bodies, community and the environment.

Independently commissioned, the methodology has been developed by the Centre for Citizenship, Enterprise and Governance (cceg.org.uk) and its affiliated network of 30+ International Universities and Research Institutes to ensure honesty and integrity in the reporting process.

## 3.0 Our Solution:

One essentially need to prove to customers that they are buying a ‘good’ product, rather than one with questionable provenance. With the development of international trade and global value chains, making any product is complex intertwining many suppliers with multi-step processes. Each step involves the creation of data, storage and centralized access. Technically, having detailed information of products from birth to death is impossible. These complexities make supply chain provenance a significant non-trivial exercise.

Blockchain technology can potentially improve the transparency and trace-ability issues within the manufacturing supply chain using immutable record of data, distributed storage, and controlled user access. All data in each step of the supply chain will update directly and securely in block-chain. All stakeholders could trace and access information of the product with every detail of the animal husbandry, labour conditions, chemical processes and other intangible KPI’s that directly affect the tangible price of the product at each stage.

For example, the leather industry, Provenance Engine will represent a decentralized distributed system that collect, store and manage key product information of each leather product throughout its life cycle. Product information includes both tangible (financial hard) attributes of all the stages of production & distribution as well intangible (non-financial soft) ones. This creates a secure, shared record of exchange for each product along with specific product information, such as animal husbandry at the farm, modern slavery conditions in which the product was processed, environmental issues surrounding the use of chemicals, retail interventions and postconsumer stages through to the land-fill.

## 4.0 Proposed system for general purposes

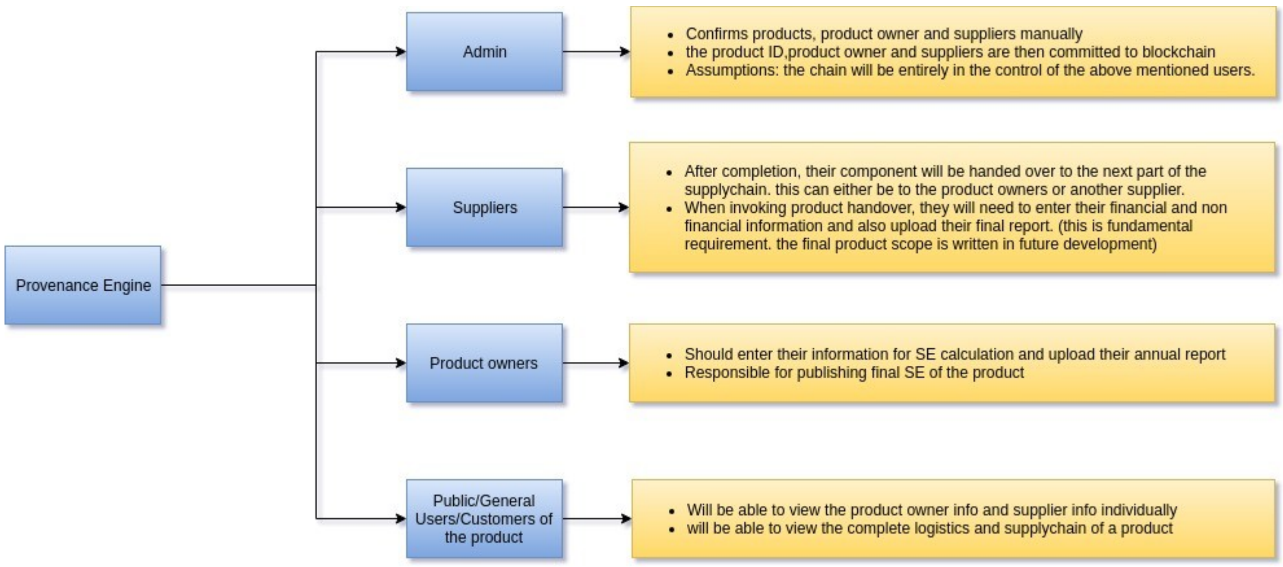
The proposed approach comprises of a decentralized distributed system that uses Ethereum blockchain to collect, store and manage key product information throughout its life cycle. This creates a secure, shared record of exchange for each product along with specific product information.

The provenance engine (Beta) is designed for general use in any supply chain for any industry. However, in the long run, the provenance engine is expected to deliver industry specific supply chain management system. This will help the companies monitor their current supply chain and help look for other suppliers or logistics that aligns with their values. The Provenance engine is completely decentralized even including the registration process.

The system proposes to publish the records to every stakeholder of the supply chain and add an addition layer of SER rating system to analyze the total value of each supplier or the product as such. The provenance engine uses IPFS to store the financial and no financial records for each of the supplier and for the product owner. From a customer point of view, they could search the product from the provenance engine and would be able to track every information relating to the product supply chain. For the product owner, they could search for other suppliers (or vice versa) that aligns better with their values and improve their final SE score, thereby improving the product value.

The Provenance Engine will have four types of users:

- Admin
- Suppliers
- product owners
- General users/Customers of the product



- Confirms products, product owner and suppliers manually
- the product ID,product owner and suppliers are then committed to blockchain
- Assumptions: the chain will be entirely in the control of the above mentioned users.

- After completion, their component will be handed over to the next part of the supplychain. this can either be to the product owners or another supplier.
- When invoking product handover, they will need to enter their financial and non financial information and also upload their final report. (this is fundamental requirement. the final product scope is written in future development)

- Should enter their information for SE calculation and upload their annual report
- Responsible for publishing final SE of the product

- Will be able to view the product owner info and supplier info individually
- will be able to view the complete logistics and supplychain of a product

Fig 1: Roles and responsibilities

Fig 1 depicts the roles and responsivities of each of the users within the provenance engine. However, in order to trigger these functionalities, there is also a proposed timeline of events. For example, a supplier cannot enter their details for the product if the product was not listed by the admin.

## 5.0 Solidity smart contract

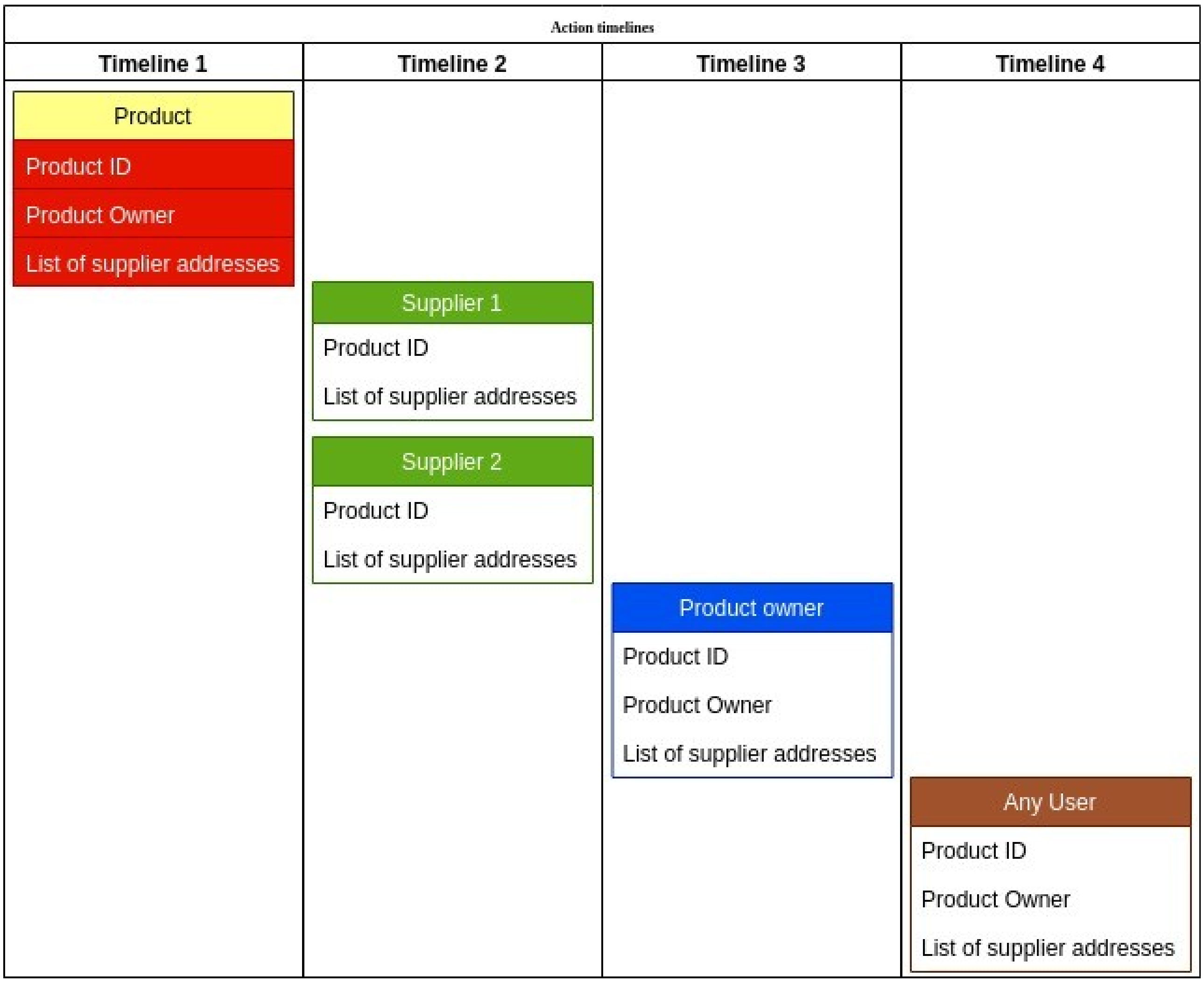
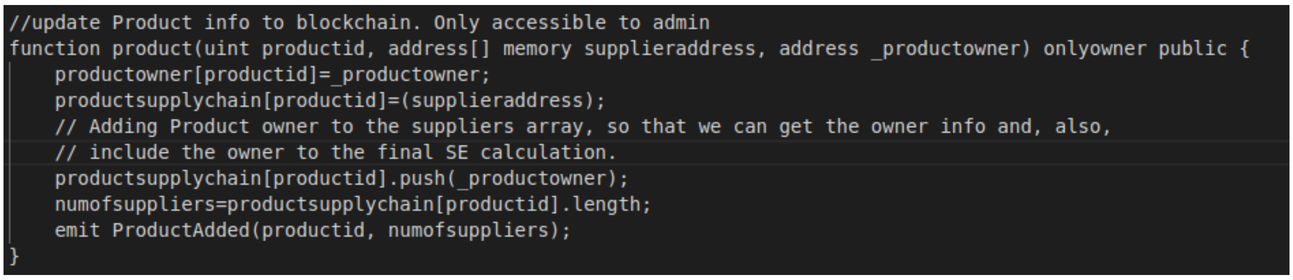


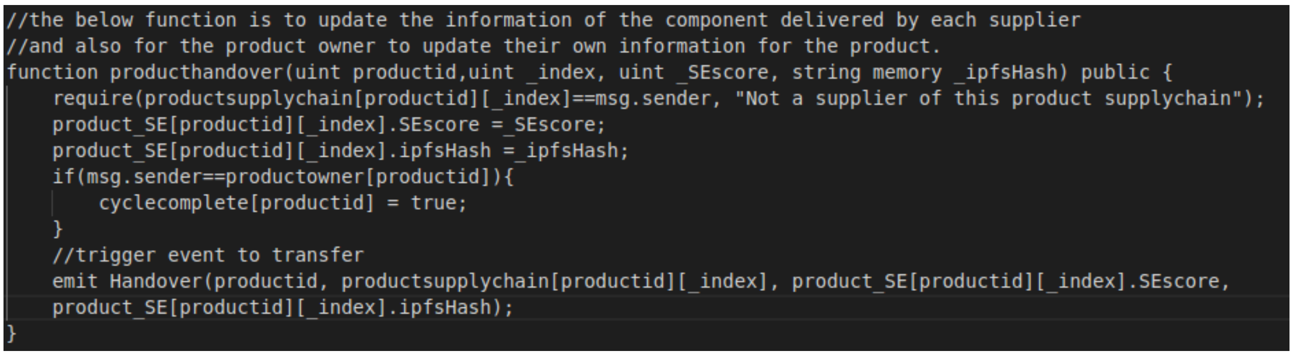
Fig 2: Action Timelines

Figure 2 shows the timeline of events that helps the end user to retrieve the complete supply chain information of the product. It starts with Admin registering the product with its product dependencies such as product owner and list of suppliers including the logistics information. This information is provided by the company which would like to use the service offered by provenance engine. Once after the admin verifies the information provided the function to update the product will be used. This registers the product in blockchain.



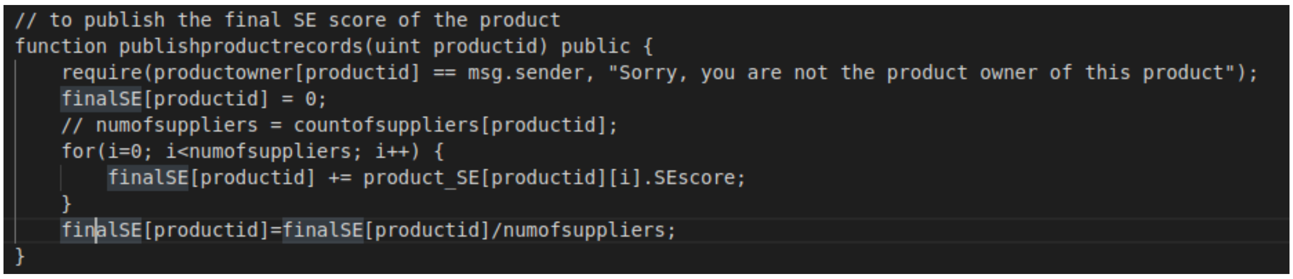
```
//update Product info to blockchain. Only accessible to admin
function product(uint productid, address[] memory supplieraddress, address _productowner) onlyowner public {
    productowner[productid]=_productowner;
    productsupplychain[productid]=(supplieraddress);
    // Adding Product owner to the suppliers array, so that we can get the owner info and, also,
    // include the owner to the final SE calculation.
    productsupplychain[productid].push(_productowner);
    numofsuppliers=productsupplychain[productid].length;
    emit ProductAdded(productid, numofsuppliers);
}
```

Once this has been established the product owner and the suppliers will be allowed to enter their information for the product. The smart contract has been created to mark the cycle as complete once the product owner enters their information. This is used as a verification condition before publishing the final SER of the product. The front end will also check if all the supplier listed for the product has entered their information for the same. This ensures that the final product information entails the complete product supply chain with their individual SE scores and Annual reports in IPFS.



```
//the below function is to update the information of the component delivered by each supplier
//and also for the product owner to update their own information for the product.
function producthandover(uint productid,uint _index, uint _SEscore, string memory _ipfsHash) public {
    require(productsupplychain[productid][_index]==msg.sender, "Not a supplier of this product supplychain");
    product_SE[productid][_index].SEscore =_SEscore;
    product_SE[productid][_index].ipfsHash =_ipfsHash;
    if(msg.sender==productowner[productid]){
        cyclecomplete[productid] = true;
    }
    //trigger event to transfer
    emit Handover(productid, productsupplychain[productid][_index], product_SE[productid][_index].SEscore,
    product_SE[productid][_index].ipfsHash);
}
```

In order to create the final SE score for the product the product owner should invoke the publishproductrecord function in the smart contract. When the product owner triggers the publishproductrecord function, the smart contract will retrieve the SE score for all the product suppliers and calculate the final SE score for the product. When customers view the product information, they will be able to see the final SE of the product followed by the information of the suppliers and product owner. They could also verify this information from the IPFS link for every stakeholders in the supply chain.



```
// to publish the final SE score of the product
function publishproductrecords(uint productid) public {
    require(productowner[productid] == msg.sender, "Sorry, you are not the product owner of this product");
    finalSE[productid] = 0;
    // numofsuppliers = countofsuppliers[productid];
    for(i=0; i<numofsuppliers; i++) {
        finalSE[productid] += product_SE[productid][i].SEscore;
    }
    finalSE[productid]=finalSE[productid]/numofsuppliers;
}
```

As discussed above, in the smart contract each of the functions are reserved for specific users. The below figure shows the data flow and how the data is used for each of their functions

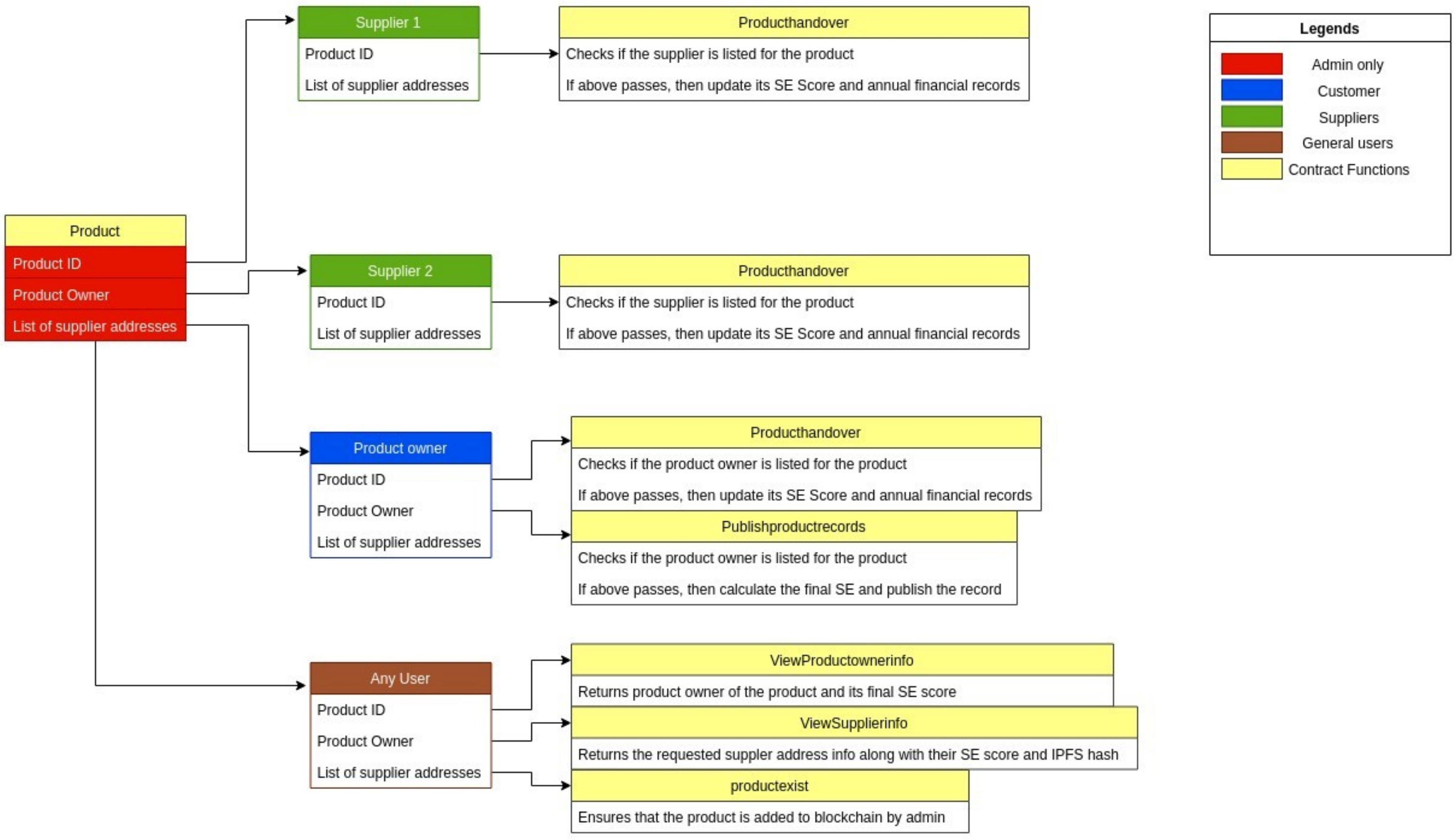

Legends
Admin only
Customer
Suppliers
General users
Contract Functions
Product
Product ID
Product Owner
List of supplier addresses
Supplier 1
Product ID
List of supplier addresses
Producthandover
Checks if the supplier is listed for the product
If above passes, then update its SE Score and annual financial records
Supplier 2
Product ID
List of supplier addresses
Producthandover
Checks if the supplier is listed for the product
If above passes, then update its SE Score and annual financial records
Product owner
Product ID
Product Owner
List of supplier addresses
Producthandover
Checks if the product owner is listed for the product
If above passes, then update its SE Score and annual financial records
Publishproductrecords
Checks if the product owner is listed for the product
If above passes, then calculate the final SE and publish the record
Any User
Product ID
Product Owner
List of supplier addresses
ViewProductownerinfo
Returns product owner of the product and its final SE score
ViewSupplierinfo
Returns the requested suppler address info along with their SE score and IPFS hash
productexist
Ensures that the product is added to blockchain by admin

Fig 3: User Types, accessible functions and flow of data

# 6.0 Front-End Documentation:

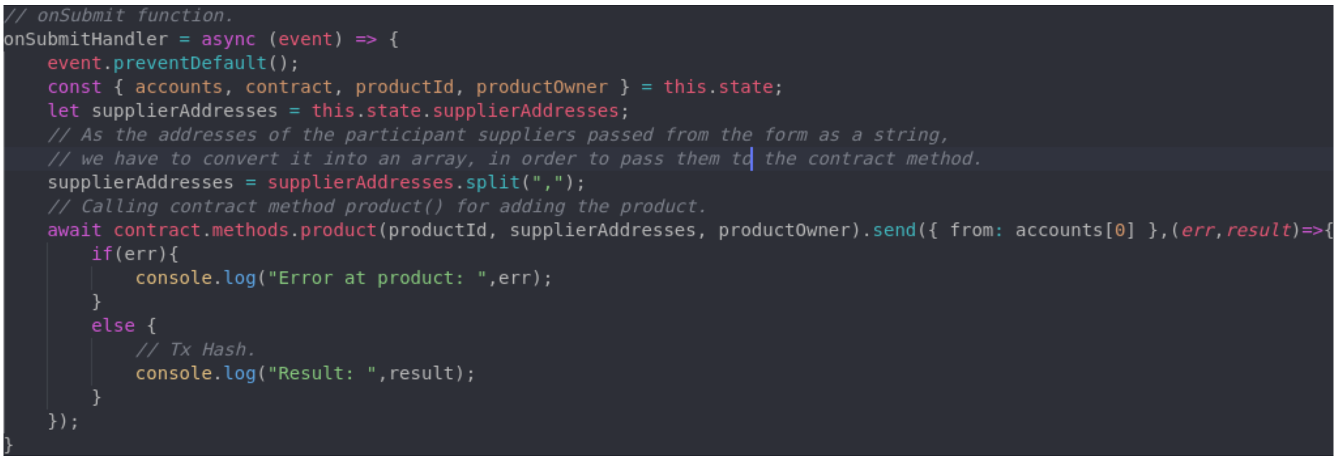
## 6.1 PRODUCT ADD COMPONENT

The addition (or registration) of a product can only be done by the Admin. The product owner (Brand owner) will reach out the admin to register the product for publishing their SER if they are interested to make their reports public. The admin is the main verification agent after the reports are submitted to publish. Either the product owner can collect and submit the reports of each participant in the supply chain or he can request them to submit. After verification, admin can register the product with primary details and tell the supply chain participants (including product owner) to input their detailed reports, so that, the admin can cross verify the details with the details submitted with initial request.

In ProductAdd component, there are two main functions:

- To register product.
- To check the existence of the product (The product registration will be blocked if it is already registered).

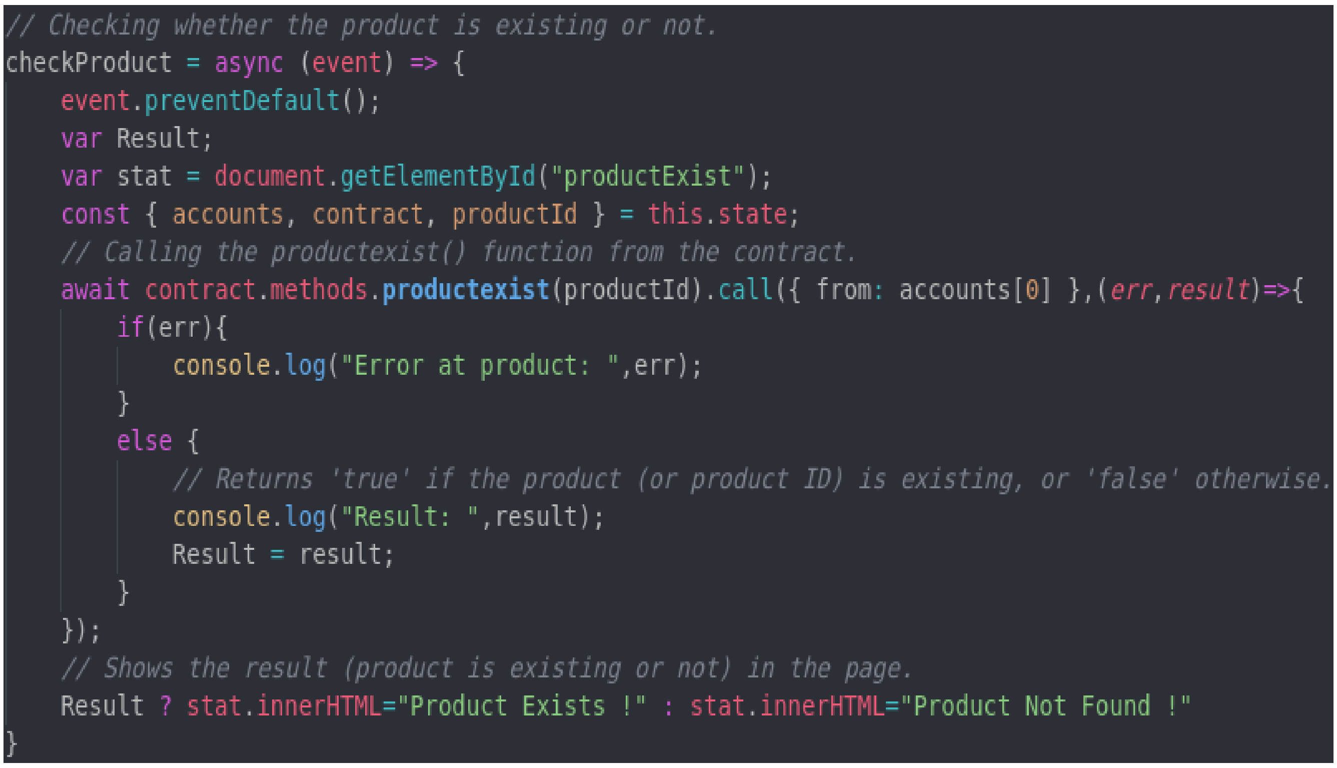
### 6.1.1 To register product



```
// onSubmit function.
onSubmitHandler = async (event) => {
    event.preventDefault();
    const { accounts, contract, productId, productOwner } = this.state;
    let supplierAddresses = this.state.supplierAddresses;
    // As the addresses of the participant suppliers passed from the form as a string,
    // we have to convert it into an array, in order to pass them to the contract method.
    supplierAddresses = supplierAddresses.split(",");
    // Calling contract method product() for adding the product.
    await contract.methods.product(productId, supplierAddresses, productOwner).send({ from: accounts[0] },(err,result)=>{
        if(err){
            console.log("Error at product: ",err);
        }
        else {
            // Tx Hash.
            console.log("Result: ",result);
        }
    });
}
```

The above function is an asynchronous function to call the “product()” function from the smart contract to register a product.

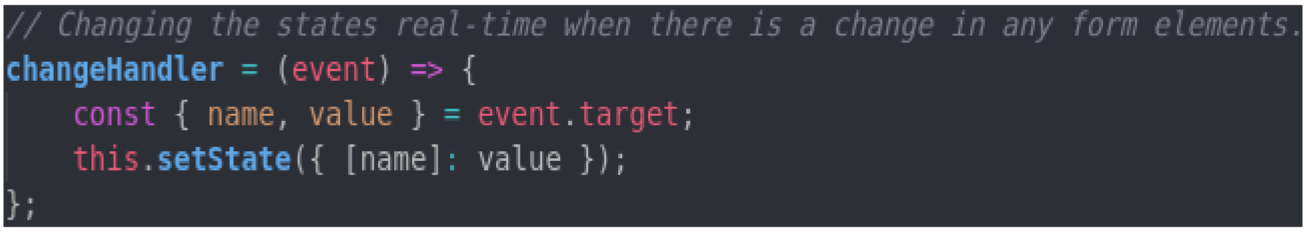
## 6.1.2 To check the existence of the product



```
// Checking whether the product is existing or not.
checkProduct = async (event) => {
    event.preventDefault();
    var Result;
    var stat = document.getElementById("productExist");
    const { accounts, contract, productId } = this.state;
    // Calling the productexist() function from the contract.
    await contract.methods.productexist(productId).call({ from: accounts[0] },(err,result)=>{
        if(err){
            console.log("Error at product: ",err);
        }
        else {
            // Returns 'true' if the product (or product ID) is existing, or 'false' otherwise.
            console.log("Result: ",result);
            Result = result;
        }
    });
    // Shows the result (product is existing or not) in the page.
    Result ? stat.innerHTML="Product Exists !" : stat.innerHTML="Product Not Found !"
}
```

The above function calls the “productexist()” function from the smart contract which is a view function, and returns “true” if the product exists, returns “false” otherwise.

Also, we are using another function “changeHandler” in common, in all the components, to reflect the changes from the input fields to the states in real-time.



```
// Changing the states real-time when there is a change in any form elements.
changeHandler = (event) => {
    const { name, value } = event.target;
    this.setState({ [name]: value });
};
```

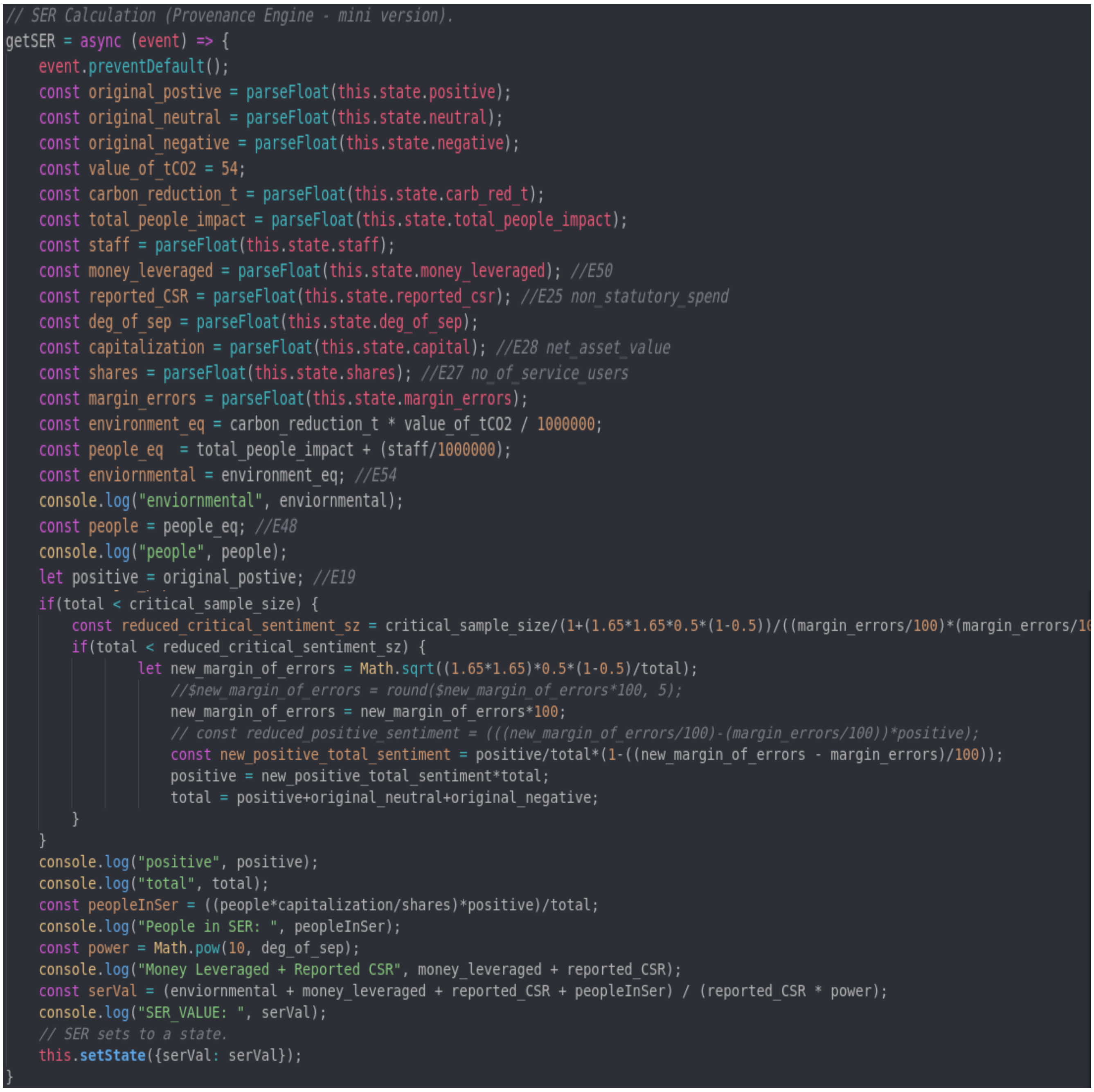
## 6.2 PRODUCT HANDOVER COMPONENT

The Product Handover component handles the individual supplychain participant inputs. In this component, the participants (each suppliers and product owner) must input their individual reports and handover to the next as the supply chain forwards. This is the component which includes the most important element in this project, SER. It includes SER form to generate individual SER, File upload segment to upload report file to IPFS (Inter-Planetary File System), which is a decentralized peer-to-peer hypermedia storage.

The main functionalities of this component are:

- SER Calculation.
- IPFS file upload.
- Individual participants report input.

### 6.2.1 SER Calculation

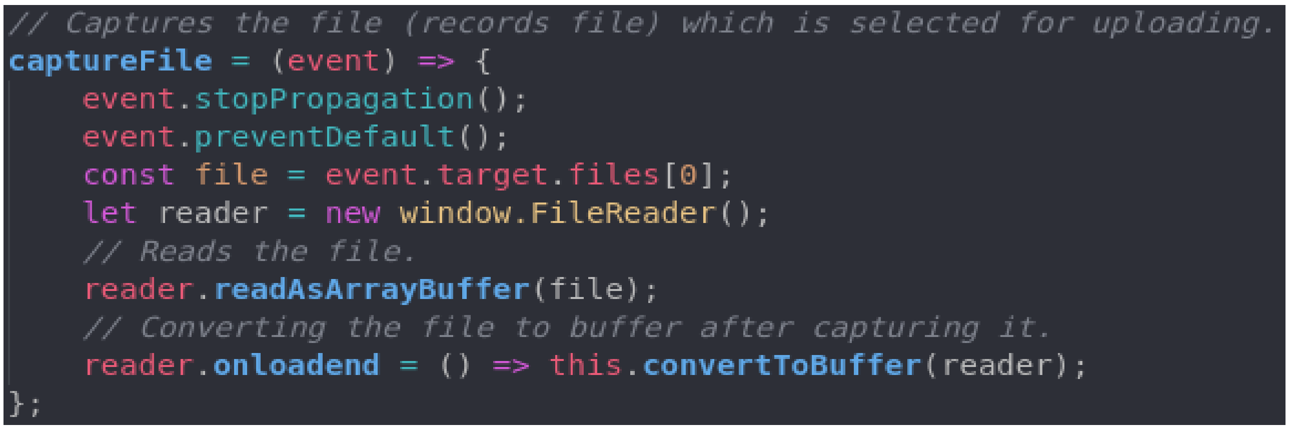


```
// SER Calculation (Provenance Engine - mini version).
getSER = async (event) => {
    event.preventDefault();
    const original_postive = parseFloat(this.state.positive);
    const original_neutral = parseFloat(this.state.neutral);
    const original_negative = parseFloat(this.state.negative);
    const value_of_tCO2 = 54;
    const carbon_reduction_t = parseFloat(this.state.carb_red_t);
    const total_people_impact = parseFloat(this.state.total_people_impact);
    const staff = parseFloat(this.state.staff);
    const money_leveraged = parseFloat(this.state.money_leveraged); //E50
    const reported_CSR = parseFloat(this.state.reported_csr); //E25 non_statutory_spend
    const deg_of_sep = parseFloat(this.state.deg_of_sep);
    const capitalization = parseFloat(this.state.capital); //E28 net_asset_value
    const shares = parseFloat(this.state.shares); //E27 no_of_service_users
    const margin_errors = parseFloat(this.state.margin_errors);
    const environment_eq = carbon_reduction_t * value_of_tCO2 / 1000000;
    const people_eq  = total_people_impact + (staff/1000000);
    const enviornmental = environment_eq; //E54
    console.log("enviornmental", enviornmental);
    const people = people_eq; //E48
    console.log("people", people);
    let positive = original_postive; //E19
    if(total < critical_sample_size) {
        const reduced_critical_sentiment_sz = critical_sample_size/(1+(1.65*1.65*0.5*(1-0.5))/((margin_errors/100)*(margin_errors/1
        if(total < reduced_critical_sentiment_sz) {
                let new_margin_of_errors = Math.sqrt((1.65*1.65)*0.5*(1-0.5)/total);
                    //$new_margin_of_errors = round($new_margin_of_errors*100, 5);
                    new_margin_of_errors = new_margin_of_errors*100;
                    // const reduced_positive_sentiment = (((new_margin_of_errors/100)-(margin_errors/100))*positive);
                    const new_positive_total_sentiment = positive/total*(1-((new_margin_of_errors - margin_errors)/100));
                    positive = new_positive_total_sentiment*total;
                    total = positive+original_neutral+original_negative;
        }
    }
    console.log("positive", positive);
    console.log("total", total);
    const peopleInSer = ((people*capitalization/shares)*positive)/total;
    console.log("People in SER: ", peopleInSer);
    const power = Math.pow(10, deg_of_sep);
    console.log("Money Leveraged + Reported CSR", money_leveraged + reported_CSR);
    const serVal = (enviornmental + money_leveraged + reported_CSR + peopleInSer) / (reported_CSR * power);
    console.log("SER_VALUE: ", serVal);
    // SER sets to a state.
    this.setState({serVal: serVal});
}
```

## 6.2.2 IPFS File upload

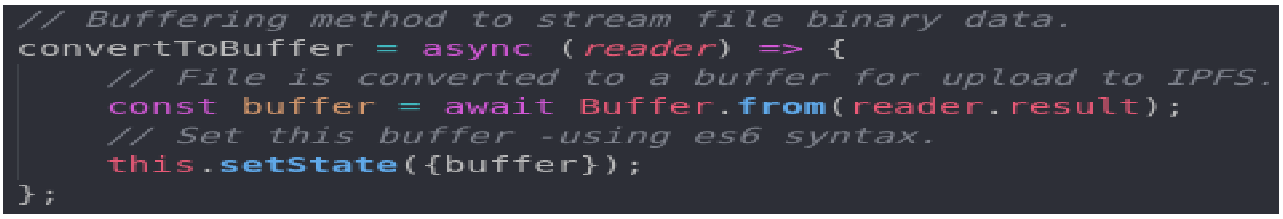
There are three main features implemented in this function. They are

- Get file streams (File Read)



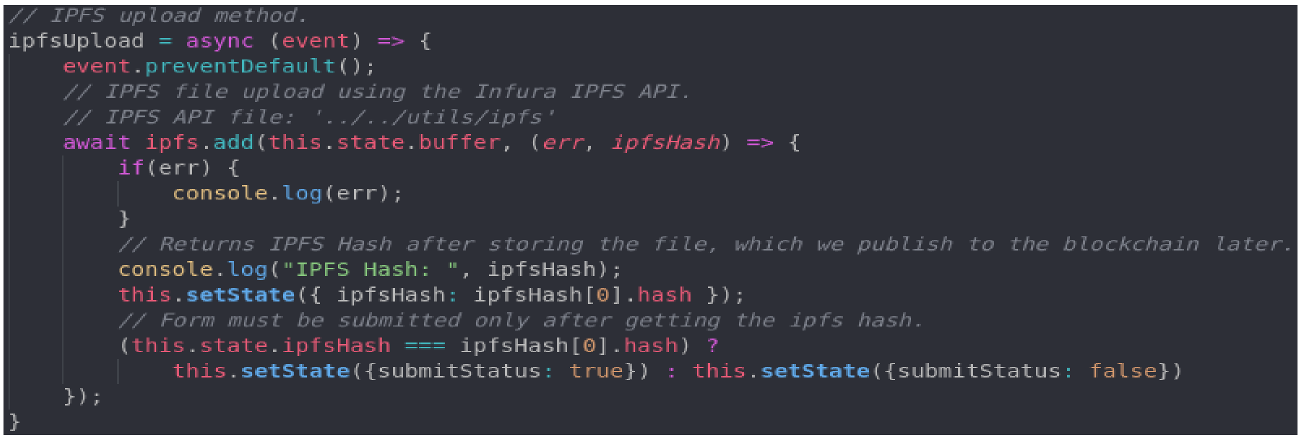
```
// Captures the file (records file) which is selected for uploading.
captureFile = (event) => {
    event.stopPropagation();
    event.preventDefault();
    const file = event.target.files[0];
    let reader = new window.FileReader();
    // Reads the file.
    reader.readAsArrayBuffer(file);
    // Converting the file to buffer after capturing it.
    reader.onloadend = () => this.convertToBuffer(reader);
};
```

- Buffer the file streams



```
// Buffering method to stream file binary data.
convertToBuffer = async (reader) => {
    // File is converted to a buffer for upload to IPFS.
    const buffer = await Buffer.from(reader.result);
    // Set this buffer -using es6 syntax.
    this.setState({buffer});
};
```

- IPFS file upload (using Infura API)



```
// IPFS upload method.
ipfsUpload = async (event) => {
    event.preventDefault();
    // IPFS file upload using the Infura IPFS API.
    // IPFS API file: '../../utils/ipfs'
    await ipfs.add(this.state.buffer, (err, ipfsHash) => {
        if(err) {
            console.log(err);
        }
        // Returns IPFS Hash after storing the file, which we publish to the blockchain later.
        console.log("IPFS Hash: ", ipfsHash);
        this.setState({ ipfsHash: ipfsHash[0].hash });
        // Form must be submitted only after getting the ipfs hash.
        (this.state.ipfsHash === ipfsHash[0].hash) ?
            this.setState({submitStatus: true}) : this.setState({submitStatus: false})
    });
}
```

## 6.3 PRODUCT PUBLISH COMPONENT (PUBLISH RECORDS)

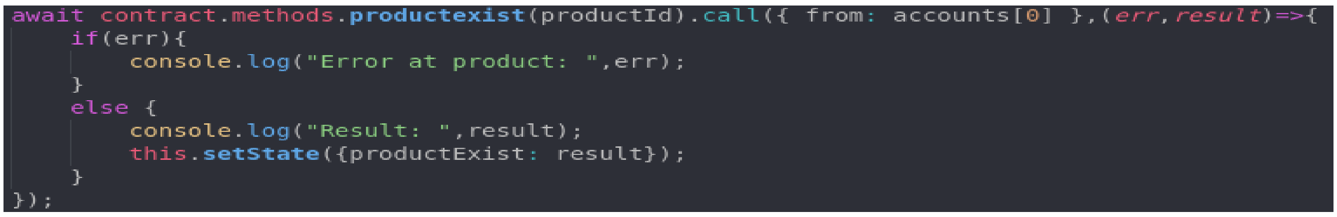
The product data is officially published in public here. It is done only by the Product owner. After this step, there will be a complete set of data of the product on blockchain, that is publicly available.

The main functionalities in this component are:

1. Double check and verify the data.
2. Officially publish the product.

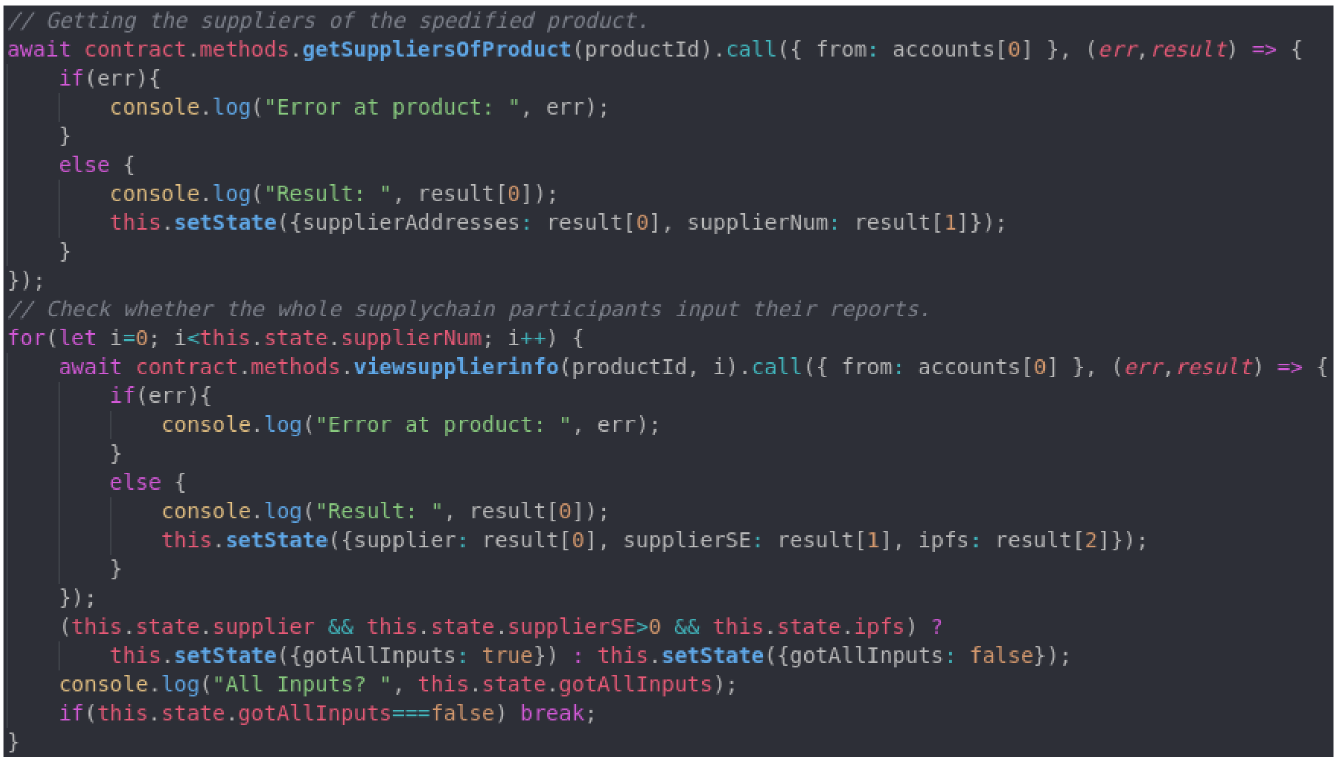
**Data Verification for publish**

1. Check product existence



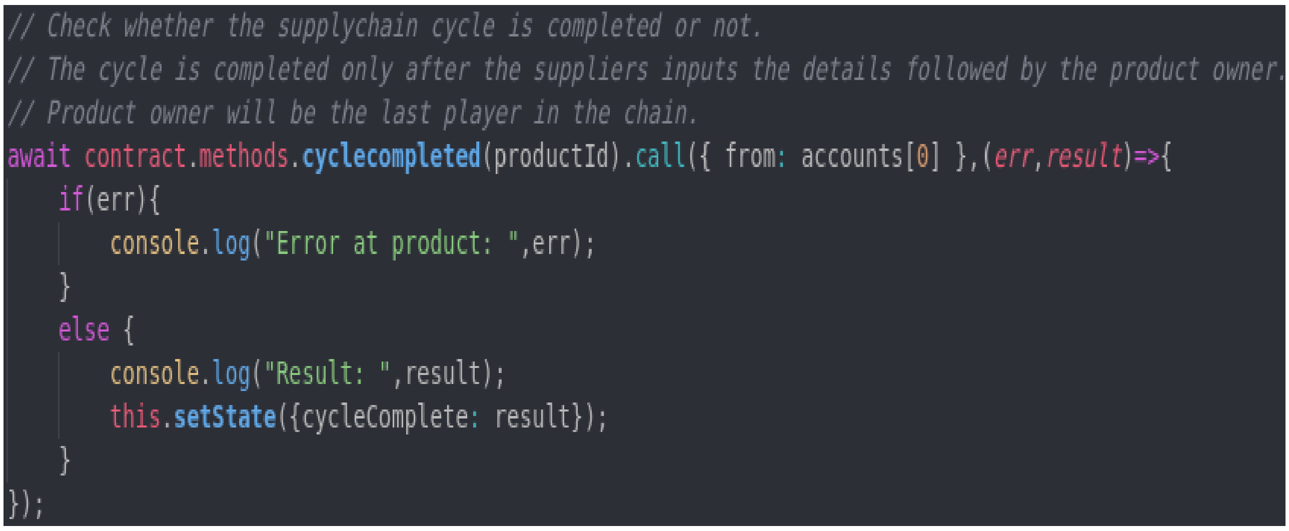
```
await contract.methods.productexist(productId).call({ from: accounts[0] },(err,result)=>{
    if(err){
        console.log("Error at product: ",err);
    }
    else {
        console.log("Result: ",result);
        this.setState({productExist: result});
    }
});
```

2. Checking for incomplete data



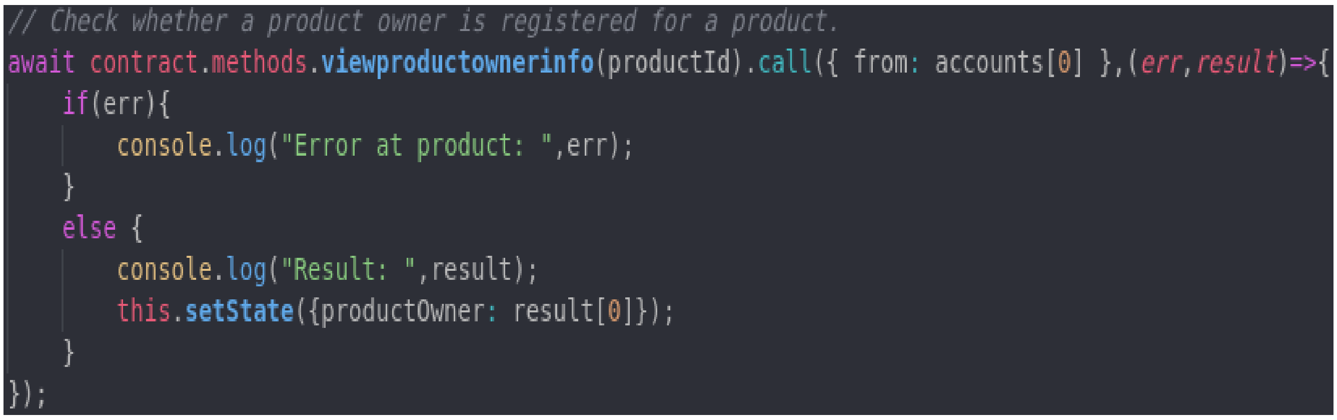
```
// Getting the suppliers of the spedified product.
await contract.methods.getSuppliersOfProduct(productId).call({ from: accounts[0] }, (err,result) => {
    if(err){
        console.log("Error at product: ", err);
    }
    else {
        console.log("Result: ", result[0]);
        this.setState({supplierAddresses: result[0], supplierNum: result[1]});
    }
});
// Check whether the whole supplychain participants input their reports.
for(let i=0; i<this.state.supplierNum; i++) {
    await contract.methods.viewsupplierinfo(productId, i).call({ from: accounts[0] }, (err,result) => {
        if(err){
            console.log("Error at product: ", err);
        }
        else {
            console.log("Result: ", result[0]);
            this.setState({supplier: result[0], supplierSE: result[1], ipfs: result[2]});
        }
    });
    (this.state.supplier && this.state.supplierSE>0 && this.state.ipfs) ?
        this.setState({gotAllInputs: true}) : this.setState({gotAllInputs: false});
    console.log("All Inputs? ", this.state.gotAllInputs);
    if(this.state.gotAllInputs===false) break;
}
```

3. Check whether the data submission cycle is completed by the product owner



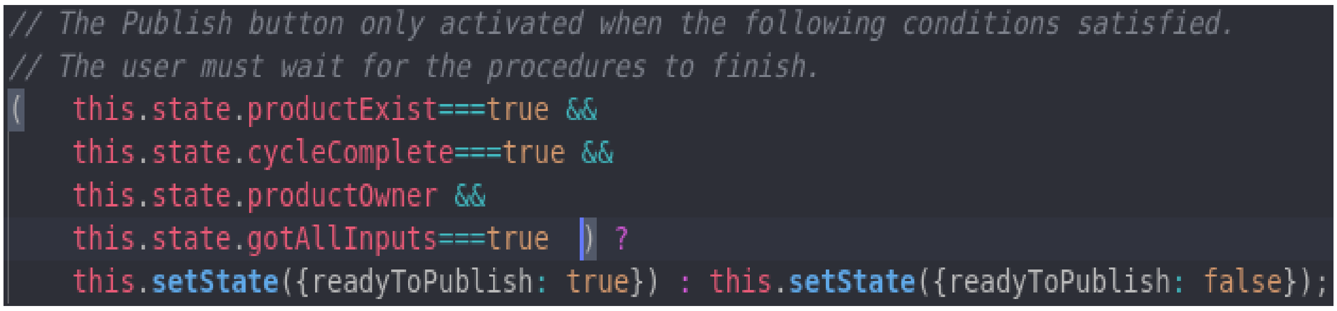
```
// Check whether the supplychain cycle is completed or not.
// The cycle is completed only after the suppliers inputs the details followed by the product owner.
// Product owner will be the last player in the chain.
await contract.methods.cyclecompleted(productId).call({ from: accounts[0] },(err,result)=>{
    if(err){
        console.log("Error at product: ",err);
    }
    else {
        console.log("Result: ",result);
        this.setState({cycleComplete: result});
    }
});
```

4. Check whether the product is registered to the product owner



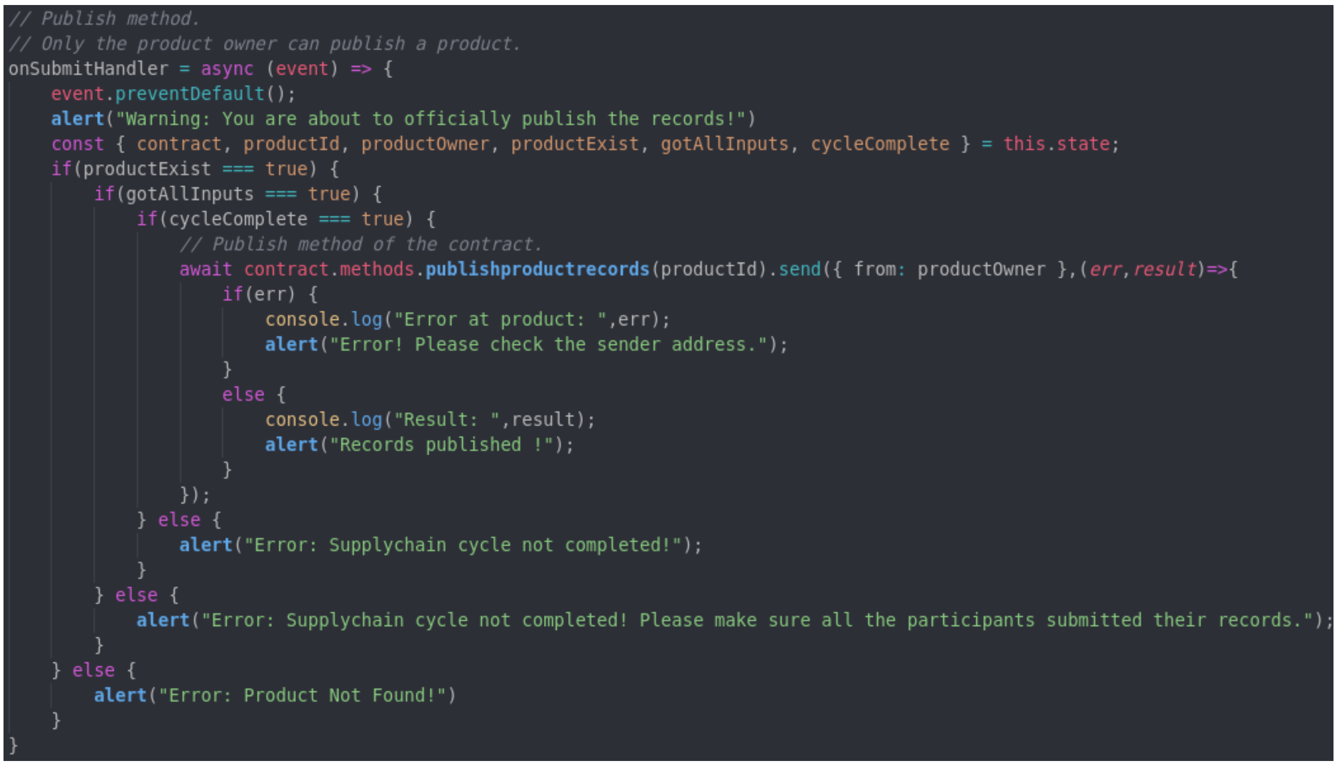
```
// Check whether a product owner is registered for a product.
await contract.methods.viewproductownerinfo(productId).call({ from: accounts[0] },(err,result)=>{
    if(err){
        console.log("Error at product: ",err);
    }
    else {
        console.log("Result: ",result);
        this.setState({productOwner: result[0]});
    }
});
```

The above conditions are checked at the end and go for the publish method.



```
// The Publish button only activated when the following conditions satisfied.
// The user must wait for the procedures to finish.
(   this.state.productExist===true &&
    this.state.cycleComplete===true &&
    this.state.productOwner &&
    this.state.gotAllInputs===true  ) ?
    this.setState({readyToPublish: true}) : this.setState({readyToPublish: false});
```

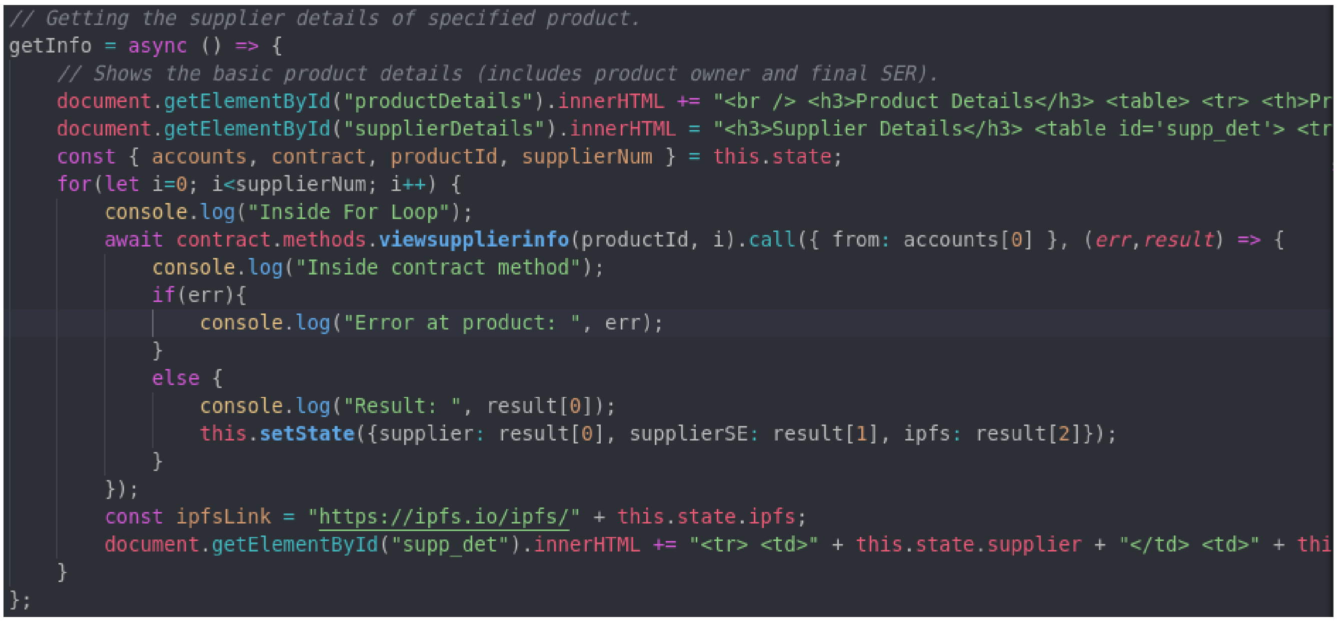
**Publish records**



```
// Publish method.
// Only the product owner can publish a product.
onSubmitHandler = async (event) => {
    event.preventDefault();
    alert("Warning: You are about to officially publish the records!")
    const { contract, productId, productOwner, productExist, gotAllInputs, cycleComplete } = this.state;
    if(productExist === true) {
        if(gotAllInputs === true) {
            if(cycleComplete === true) {
                // Publish method of the contract.
                await contract.methods.publishproductrecords(productId).send({ from: productOwner },(err,result)=>{
                    if(err) {
                        console.log("Error at product: ",err);
                        alert("Error! Please check the sender address.");
                    }
                    else {
                        console.log("Result: ",result);
                        alert("Records published !");
                    }
                });
            } else {
                alert("Error: Supplychain cycle not completed!");
            }
        } else {
            alert("Error: Supplychain cycle not completed! Please make sure all the participants submitted their records.");
        }
    } else {
        alert("Error: Product Not Found!")
    }
}
```

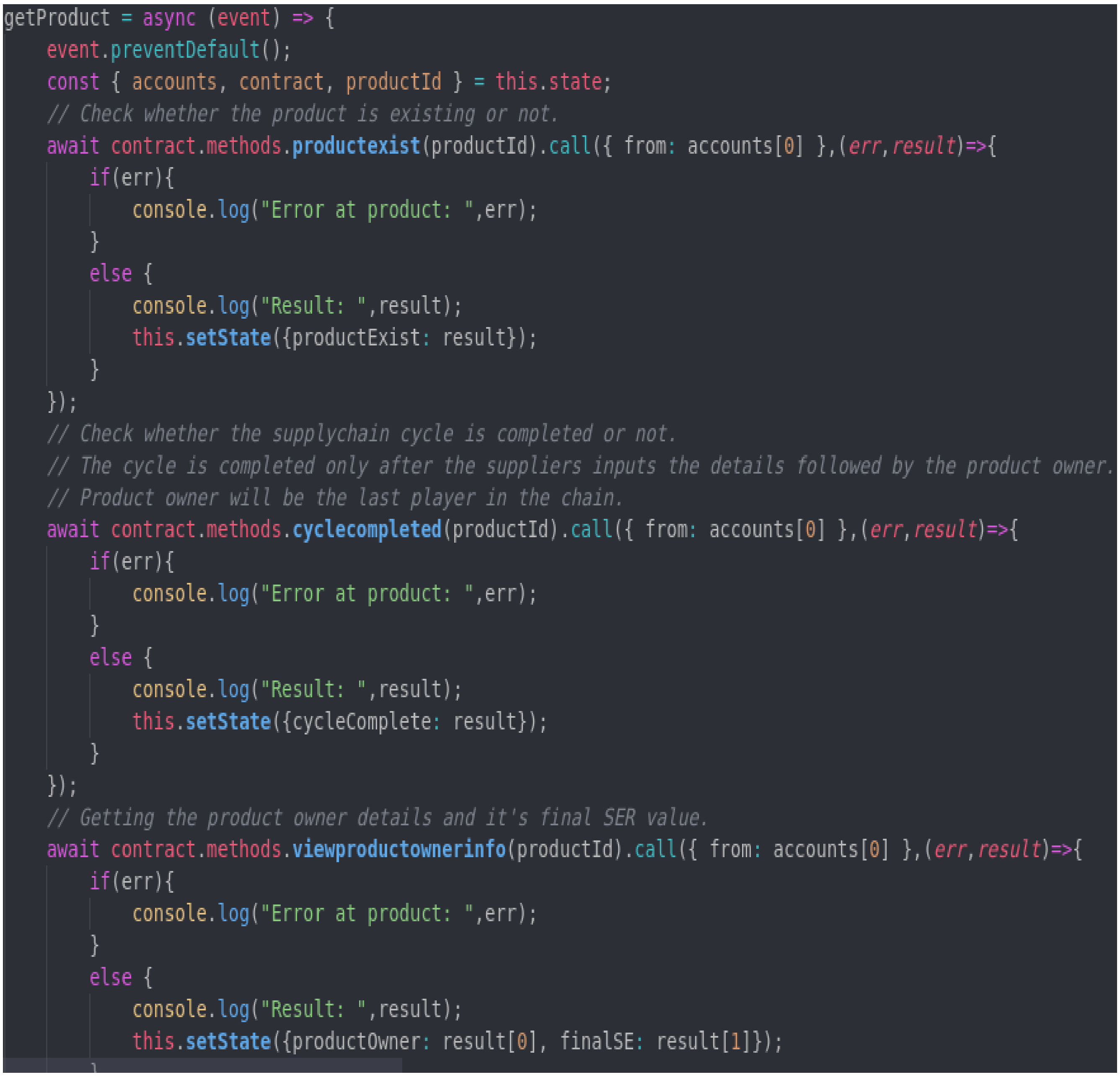
## 6.4 PRODUCT VIEW COMPONENT

The general component which is public, to view the report information based on the Product ID. This component can be accessed by anyone to view the public data.



```
// Getting the supplier details of specified product.
getInfo = async () => {
    // Shows the basic product details (includes product owner and final SER).
    document.getElementById("productDetails").innerHTML += "<br /> <h3>Product Details</h3> <table> <tr> <th>Pr
    document.getElementById("supplierDetails").innerHTML = "<h3>Supplier Details</h3> <table id='supp_det'> <tr
    const { accounts, contract, productId, supplierNum } = this.state;
    for(let i=0; i<supplierNum; i++) {
        console.log("Inside For Loop");
        await contract.methods.viewsupplierinfo(productId, i).call({ from: accounts[0] }, (err,result) => {
            console.log("Inside contract method");
            if(err){
                console.log("Error at product: ", err);
            }
            else {
                console.log("Result: ", result[0]);
                this.setState({supplier: result[0], supplierSE: result[1], ipfs: result[2]});
            }
        });
        const ipfsLink = "https://ipfs.io/ipfs/" + this.state.ipfs;
        document.getElementById("supp_det").innerHTML += "<tr> <td>" + this.state.supplier + "</td> <td>" + thi
    }
};
```

There is a verification stage before the “getInfo()” function, called “getProduct()”, to double check whether there is any incomplete data.



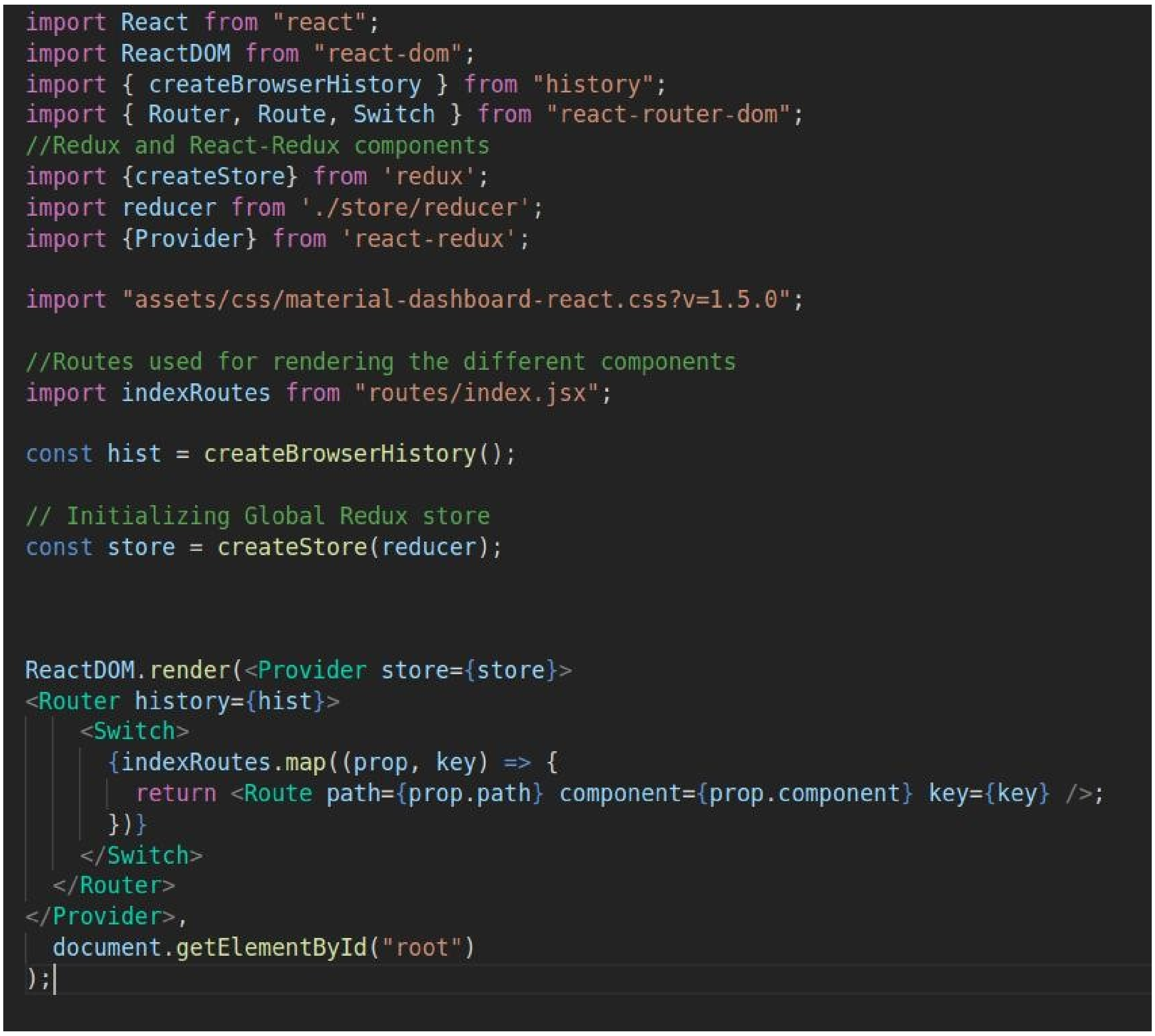
```
getProduct = async (event) => {
    event.preventDefault();
    const { accounts, contract, productId } = this.state;
    // Check whether the product is existing or not.
    await contract.methods.productexist(productId).call({ from: accounts[0] },(err,result)=>{
        if(err){
            console.log("Error at product: ",err);
        }
        else {
            console.log("Result: ",result);
            this.setState({productExist: result});
        }
    });
    // Check whether the supplychain cycle is completed or not.
    // The cycle is completed only after the suppliers inputs the details followed by the product owner.
    // Product owner will be the last player in the chain.
    await contract.methods.cyclecompleted(productId).call({ from: accounts[0] },(err,result)=>{
        if(err){
            console.log("Error at product: ",err);
        }
        else {
            console.log("Result: ",result);
            this.setState({cycleComplete: result});
        }
    });
    // Getting the product owner details and it's final SER value.
    await contract.methods.viewproductownerinfo(productId).call({ from: accounts[0] },(err,result)=>{
        if(err){
            console.log("Error at product: ",err);
        }
        else {
            console.log("Result: ",result);
            this.setState({productOwner: result[0], finalSE: result[1]});
```

## 7.0 Design and Layout

The core components used in designing the front-end of this DAPP are:

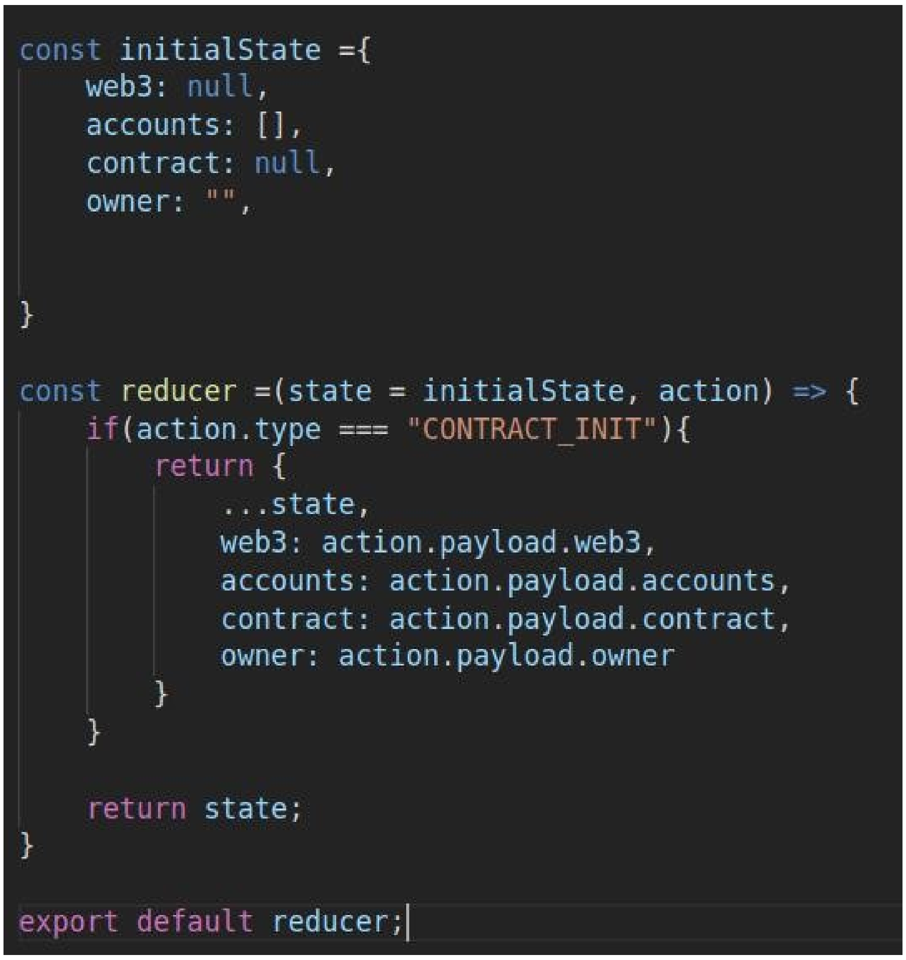
- React JS (Front-End Framework)
- Redux (State-Management Tool)
- Material-UI free template (Styling the components)

The application start point is at the index.js (product _ provenance _ 2.0 / client / src / index.js)



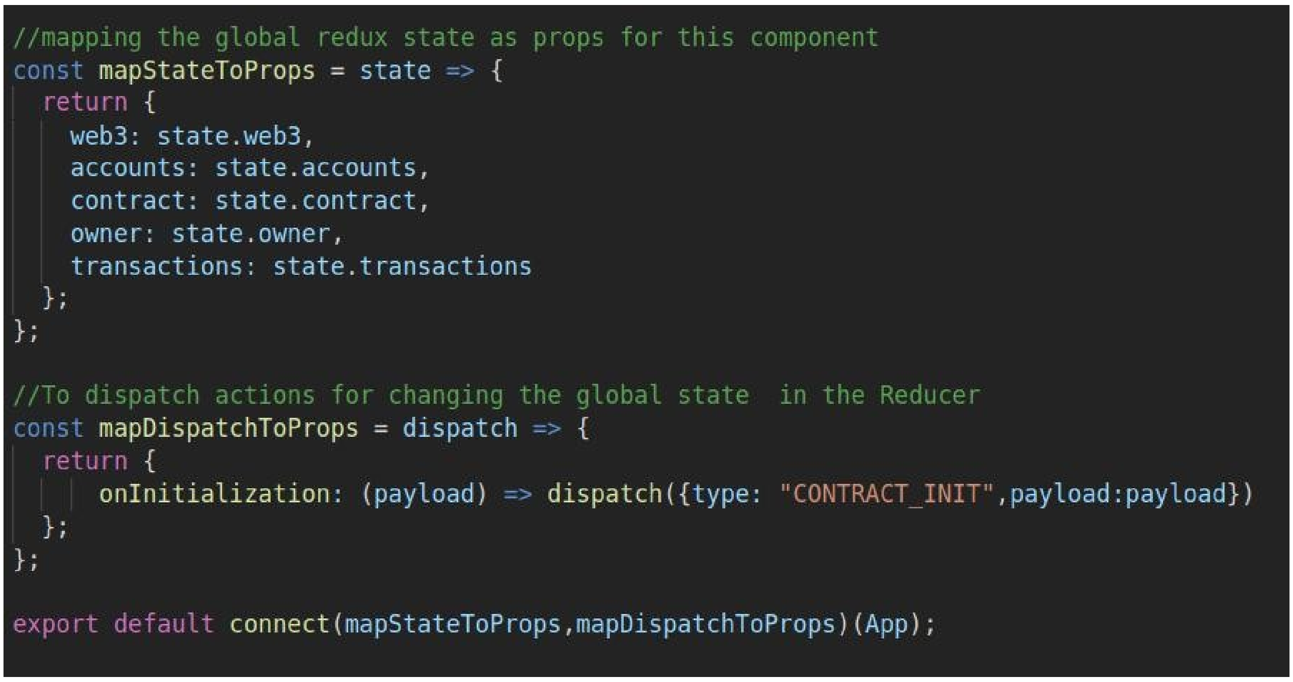
```
import React from "react";
import ReactDOM from "react-dom";
import { createBrowserHistory } from "history";
import { Router, Route, Switch } from "react-router-dom";
//Redux and React-Redux components
import {createStore} from 'redux';
import reducer from './store/reducer';
import {Provider} from 'react-redux';

import "assets/css/material-dashboard-react.css?v=1.5.0";

//Routes used for rendering the different components
import indexRoutes from "routes/index.jsx";

const hist = createBrowserHistory();

// Initializing Global Redux store
const store = createStore(reducer);



ReactDOM.render(<Provider store={store}>
<Router history={hist}>
    <Switch>
      {indexRoutes.map((prop, key) => {
        return <Route path={prop.path} component={prop.component} key={key} />;
      })}
    </Switch>
  </Router>
</Provider>,
  document.getElementById("root")
);
```

Here the Redux store is created and relates to the whole app using the Provider. Redux store is initialized using the createStore command which receives a Reducer as an argument. The routes for the application come from client/src/routes where each component routing parameters are defined. The reducer which manages the global state variables is initialized in the reducer(client/src/store/reducer.js)



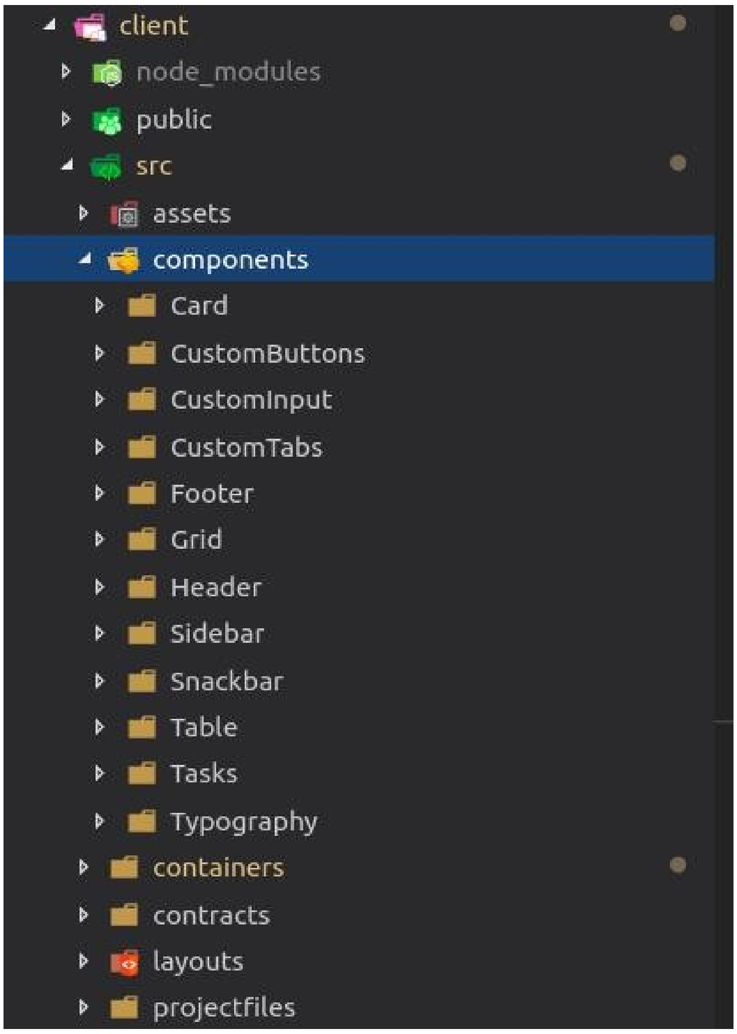
```
const initialState ={
    web3: null,
    accounts: [],
    contract: null,
    owner: "",


}

const reducer =(state = initialState, action) => {
    if(action.type === "CONTRACT_INIT"){
        return {
            ...state,
            web3: action.payload.web3,
            accounts: action.payload.accounts,
            contract: action.payload.contract,
            owner: action.payload.owner
        }
    }

    return state;
}

export default reducer;
```

The reducer initializes the state variables when the contract is deployed, and every component connected to the reducer gets the state variables as props. The code below is used to connect the component with the global states and the dispatch action for the reducer.



```
//mapping the global redux state as props for this component
const mapStateToProps = state => {
  return {
    web3: state.web3,
    accounts: state.accounts,
    contract: state.contract,
    owner: state.owner,
    transactions: state.transactions
  };
};

//To dispatch actions for changing the global state  in the Reducer
const mapDispatchToProps = dispatch => {
  return {
      onInitialization: (payload) => dispatch({type: "CONTRACT_INIT",payload:payload})
  };
};

export default connect(mapStateToProps,mapDispatchToProps)(App);
```

The components used for styling are custom Material Dashboard React components which is freely available in the web.



```
client
  node_modules
  public
  src
    assets
    components
      Card
      CustomButtons
      CustomInput
      CustomTabs
      Footer
      Grid
      Header
      Sidebar
      Snackbar
      Table
      Tasks
      Typography
    containers
    contracts
    layouts
    projectfiles
```

## 8.0 Truffle Testing

To understand the reliability and efficiency of the smart contract, testing is a must. The tests were carried out based on the below test conditions.

<u>Test1 (success)</u>: Check whether the product is registered in blockchain

<u>Test2 (success)</u>: Retrieve the list of suppliers and product owner for the product id. Check if the results match with expectations

<u>Test3 (success)</u>: Check the producthandover function and retrieve the information of one of the suppliers to verify the data.

<u>Test4 (success)</u>: Run the publishproductrecords function then check if the record was published, the owner address matches and also the Final SE matches the expected outcome.

<u>Test5 (Must fail)</u>: Check if product exists for an unregistered product id

Test6 (Must fail): Check if the total number of suppliers match with the expected outcome

Test7 (Must fail): Check if a specific supplier address matches the expected outcome

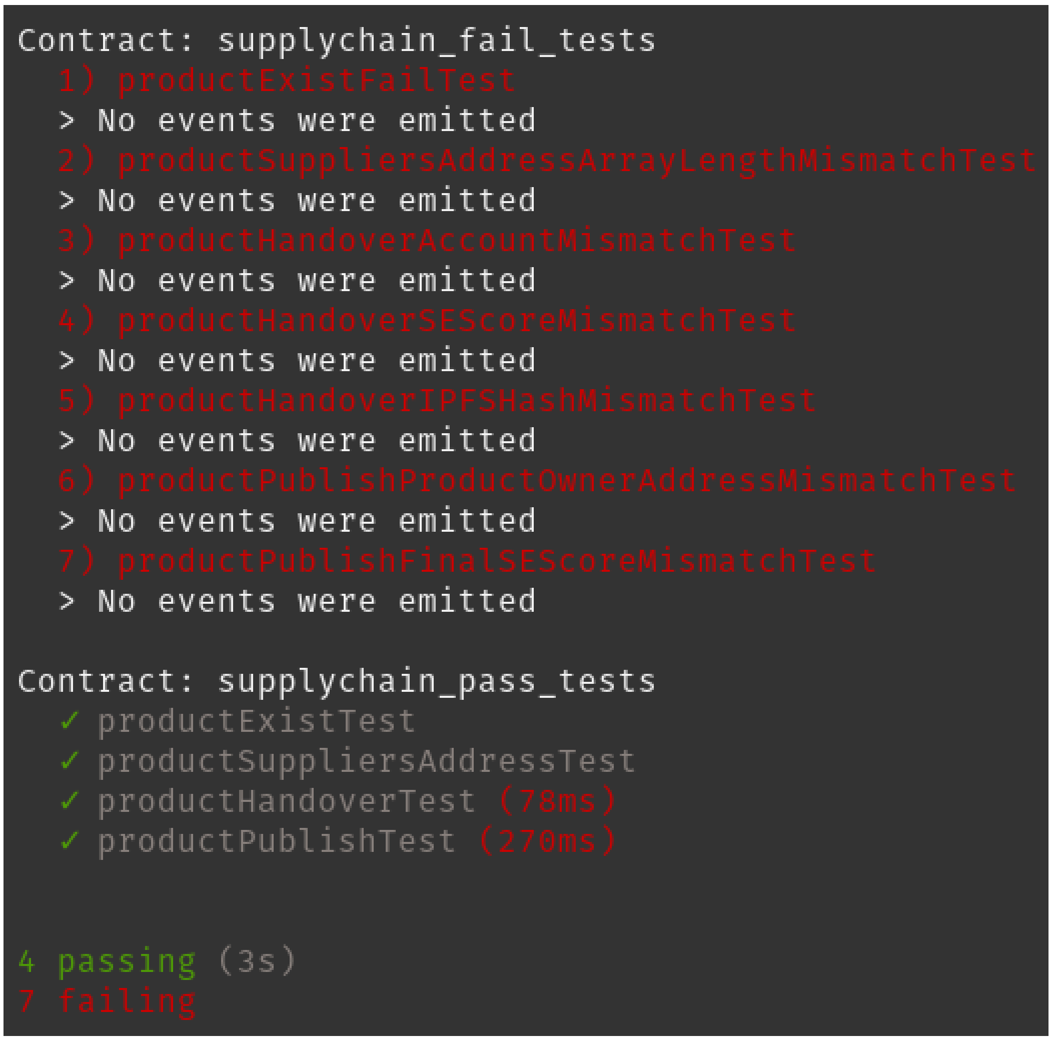
Test8 (Must fail): Check if the SE score for the above supplier matches expectations.

Test9 (Must fail): Check for mismatch in IPFS hash.

Test10 (Must fail): Check for mismatch in product owner address

Test11 (Must fail): Check for mismatch in Final SE

Further tests were carried out to check if any user could trigger the product, producthandover and publishproductrecords functions. These resulted in EVM revert error due to the require feature within the respective functions. The result for testing is as below.



```
Contract: supplychain_fail_tests
  1) productExistFailTest
  > No events were emitted
  2) productSuppliersAddressArrayLengthMismatchTest
  > No events were emitted
  3) productHandoverAccountMismatchTest
  > No events were emitted
  4) productHandoverSEScoreMismatchTest
  > No events were emitted
  5) productHandoverIPFSHashMismatchTest
  > No events were emitted
  6) productPublishProductOwnerAddressMismatchTest
  > No events were emitted
  7) productPublishFinalSEScoreMismatchTest
  > No events were emitted

Contract: supplychain_pass_tests
  ✓ productExistTest
  ✓ productSuppliersAddressTest
  ✓ productHandoverTest (78ms)
  ✓ productPublishTest (270ms)


4 passing (3s)
7 failing
```

## 9.0 Future Scope:

Admin must verify the product and its supply chain manually. This requires a front-end CRM that enables every user of the supply chain to enter and upload their annual financial report before committing to blockchain. Once the data is verified the admin will trigger the product function to update the details. For the product handover feature every field will be auto populated from a database based on the inputs provided for verification. If the supplier or the product owner wishes to change the information, they should contact the admin.

Company portfolio should show all their products and its respective Provenance information.

A QR code for the product is generated after the product is published by the product owner/ company. This QR code once scanned will show the final SE score along with every data published on blockchain by the suppliers and product owner.

## 10.0 Conclusion

To combat growing consumer awareness of the aforementioned issues, the transparency of product provenance, trace-ability and effective control of suppliers are key to the growth of the sector. This system has been created to achieve maximum impact and with least cost. When you can measure it, you can influence the sustainable development of the cycle.  It allows for both upstream and downstream controls to be enabled and embedded, from farms to land-fill. In effect a product can carry a digital passport that contains all the information you need to make and direct decision making at each stage.