Design Decisions

The core design decision of this project is to reduce gas cost as much as possible without compromising on the core development goals.

As part of the proof of concept, the provenance engine is placed in front-end. This is where the Social Value calculation is performed.

React JS was chosen as the front-end framework for the ease of integrating it with web3 js and ethereum components.

Redux is used for managing the global states which are required among components. With the use of Redux, passing global states form one component to other has been made easier.

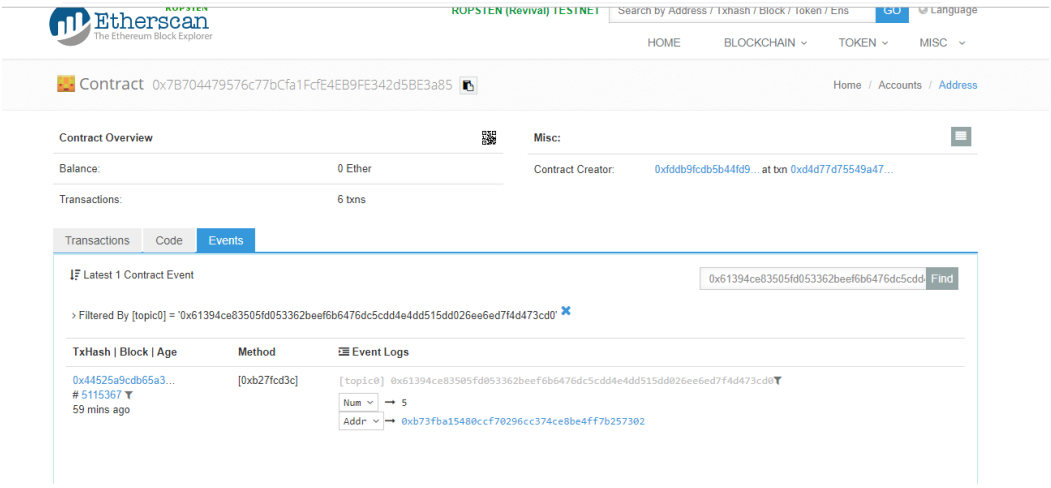
The DAPP is styled using a custom template based on material ui.

Add Product

The Product Add Component is designed to register a product. The smart contract is designed in a way that is only accessible for the admin user (makes use of onlyowner function). Admin user is the user that deploys the contract. This makes sure that the access to register the product only lies with the admin user. There is another function to view the existence of a product.

```
function product(uint productid, address[] memory supplieraddress, address _productowner) onlyowner public
```

Once the product has been registered, an event is set to fire through smart contract in etherscan. This returns product id and product owner.



Product Handover

The Product handover component is a key competent as most of the data inputs and their operations are done here. The core provenance engine is also lying here.

The first part of the component is to load the suppliers of a product. If there is no product existing, it will be alerted. If there is a product, which is specified, it will load the entire list of participants in the supply chain, so we can select the address of who is handing over the product. Once the supplier is selected, the index location is passed on to the smart 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===true) ? this.setState({productExist: true}) : this.setState({productExist: false});
    }
});
(this.state.productExist===true) ? this.getSuppliers() : alert("Product Not Found!");
```

The next part is to get the SE score of the selected address by the inputs of their records. The form for the SE calculation can be hidden or revealed as user wishes, by a click of a button.

The third part is to upload the annual records file of the participant in the distributed peer-to-peer Inter-Planetary File System (IPFS) and retrieves a hash of the file, which can be used to access the file later.

```
// 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({checkStatus: true}) : this.setState({checkStatus: false});
    });
}
```

The final section is the validation and submission part. The product validation (or product checking) part is only activated after the file upload. The product validation checks if the product is registered and if the current supplier already submitted their records. If the supplier already submitted, the submit part will be blocked,

otherwise, the submit will be activated, so that, the records can be submitted.

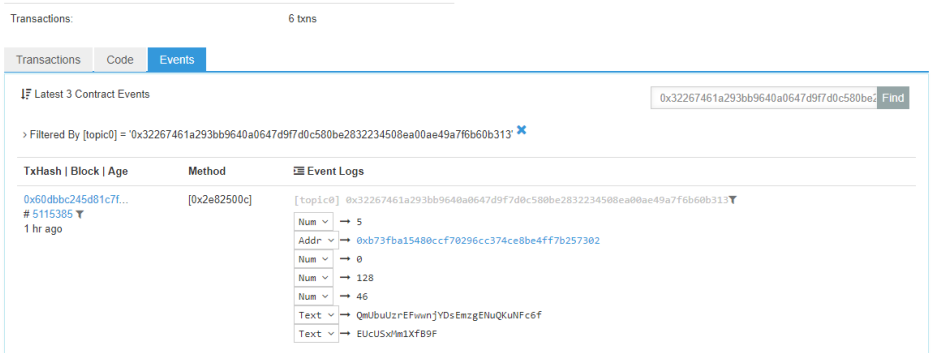
```
(this.state.supplier && this.state.supplierSE>=0 && this.state.ipfs) ?
    this.setState({gotAllInputs: true}) : this.setState({gotAllInputs: false});
console.log("All Inputs? ", this.state.gotAllInputs);
if(this.state.gotAllInputs===true) {
    this.setState({submitStatus: false});
    alert("Submit Blocked as Data already submitted! If you need a resubmit, please contact the Admin.");
} else {
    this.setState({submitStatus: true});
    alert("Warning: You can proceed to POST your records in PUBLIC if you are interested.");
}
```

As part of a verification, we made sure that the user that triggers this function is a registered supplier for the product. This verification is performed within the smart contract by making use of the below code

```
require(productsupplychain[productid][_index]==msg.sender,   "Not   a
supplier of this product supplychain");
```

Once the supplier enters their information for the handover, an event is fired through smart contract in etherscan. The returned values are product ID, Product owner, SE score and IPFS hash. IPFS is split in four parts; two parts in hash length and its corresponding hash values. The final ipfs hash will be QmUbuUzrEFwwnjYDsEmzgENuQKuNFc6fEUcUSxMm1XfB9F;

and url will be https://ipfs.io/ipfs/QmUbuUzrEFwwnjYDsEmzgENuQKuNFc6fEUcUSxMm1XfB9F



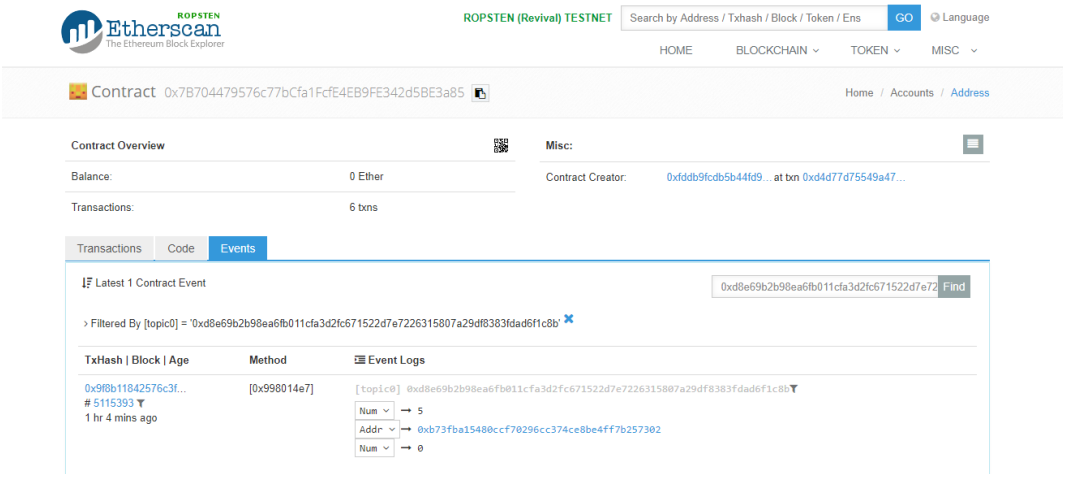
Publish Product Records

The Product Publish Component handles with agreement of the product owner to publish his product officially. It checks the product's existence at first. Then checks if all inputs from all the participants already submitted. If there is any participant (supplier) is pending to submit (handover), the address will be alerted. The publish button is activated only after this validation. Also, there will be a warning about publishing the records public. After the product publish, there will be a QR code generated about the basic details of the product.

Like the Product handover, the verification here is performed within the smart contract as well. This ensures that only the product owner can trigger this function.

```
require(productowner[productid] == msg.sender, "Sorry, you are not the product owner of this product");
```

Once the Product owner enters publishes the product information, an event is fired through smart contract in etherscan. The returned values are Product ID, Product Owner and Final SE score of the product.



Check for Incomplete Data

```
if(this.state.gotAllInputs===false) {
    this.setState({readyToPublish: false});
    this.setState({unsubmittedAddress: this.state.supplier});
    console.log("unsubmittedAddress ", this.state.unsubmittedAddress);
    alert("Warning: Records of supplier " + this.state.unsubmittedAddress + " has not been submitted yet. Please try again after!")
    break;
} else {
    this.setState({readyToPublish: false});
}
```

Check for product existence and cycle complete

```
// 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});
```

QR generator code

```
// Getting basic product details through QR-Code.
getQR = async () => {
    const { contract, accounts, productId } = this.state;
    // 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]});
        }
    });
    const qrtxt = "Product ID : " + this.state.productId + " | Product Owner : " + this.state.productOwner +
    // console.log("qrtxt: ", qrtxt);
    this.setState({qrtext: qrtxt});
    // console.log("qrtext: ", this.state.qrtext);
}
```

View Product Information

The Product View Component is a general component to view the posted data to a general customer. The product existence and incomplete data inputs will be alerted as the product is given for the view. If the product is registered and there is a complete set of data posted, the view component displays the existing data.

```
(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;
```

```
(   this.state.productExist===true &&
    this.state.cycleComplete===true &&
    this.state.productOwner &&
    this.state.supplierAddresses    )   ?  this.getInfo()   :   alert("Error: Please check if the product is registered
```