Exception Handling

Team Id	NM2023TMID04410
Project Name	Project-Drug Traceability

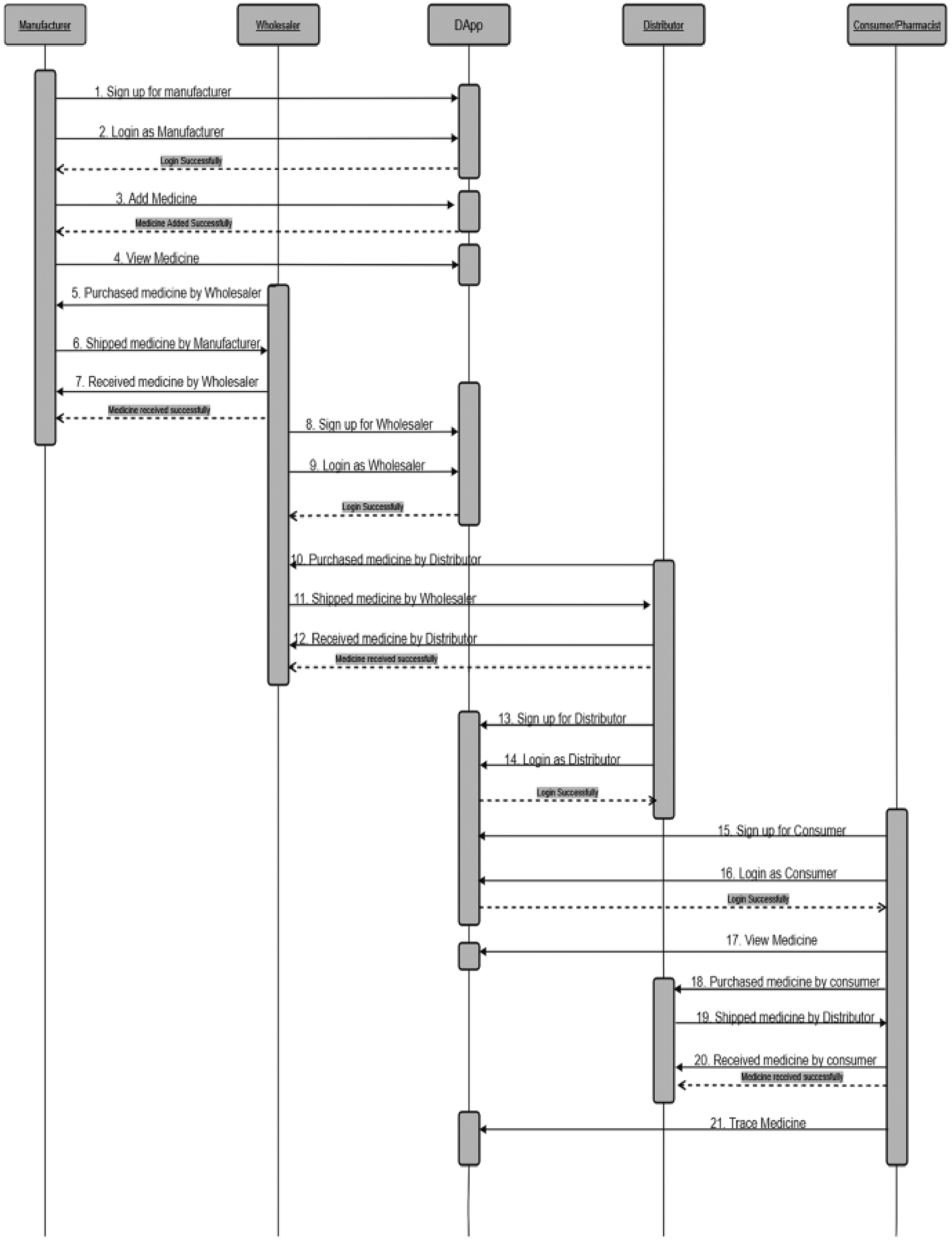
Exception handling in drug traceability:

- The Ethereum blockchain platform is being used to develop the proposed solution. Ethereum is a permissionless public blockchain, which means that anyone can access it.
- The truffle framework is used to compile and test the smart contract, which is written in Solidity.
- It provides a set of tools for creating smart contracts using the Solidity programming language.
- It also allows us to test and deploy smart contracts to the blockchain. Along with truffle we have used ganache which provides us with a local Ethereum blockchain for development and testing purposes.
- Ganache provides 10 default accounts with 100 ethers each. At last, metamask is used to make transactions with the help of decentralized applications onto the local blockchain network.
- At first, the stakeholders of the supply chain must register themselves onto the decentralized application with all the necessary details and once the stakeholder is added then an event will be triggered and announced to all participants in the supply chain.
- Secondly, it’s the manufacturer’s responsibility to add a pre-approved medicine to the system. Once the medicine is added an event is triggered that the medicine has been added and is up for the sale.
- Now the interested wholesalers can initiate the purchase process and once the required funds are transferred to the manufacturer account an event will notify the purchase of medicine and transfer of ownership to all the participants and then the medicine state will get changed to PurchasedByWholesalerAndForSale.
- Next the third level entity i.e., a distributor along with the same level stakeholder (other wholesalers in the system) can purchase medicine from the wholesaler and transfer the funds accordingly and trigger a purchase medicine event along with the current change in

the state of medicine either to PurchasedByWholesalerAndForSale or to PurchasedByDistributerAndForSale after recording the state of medicine in history to SoldByWholesaler, thus completing the purchase process by distributer.

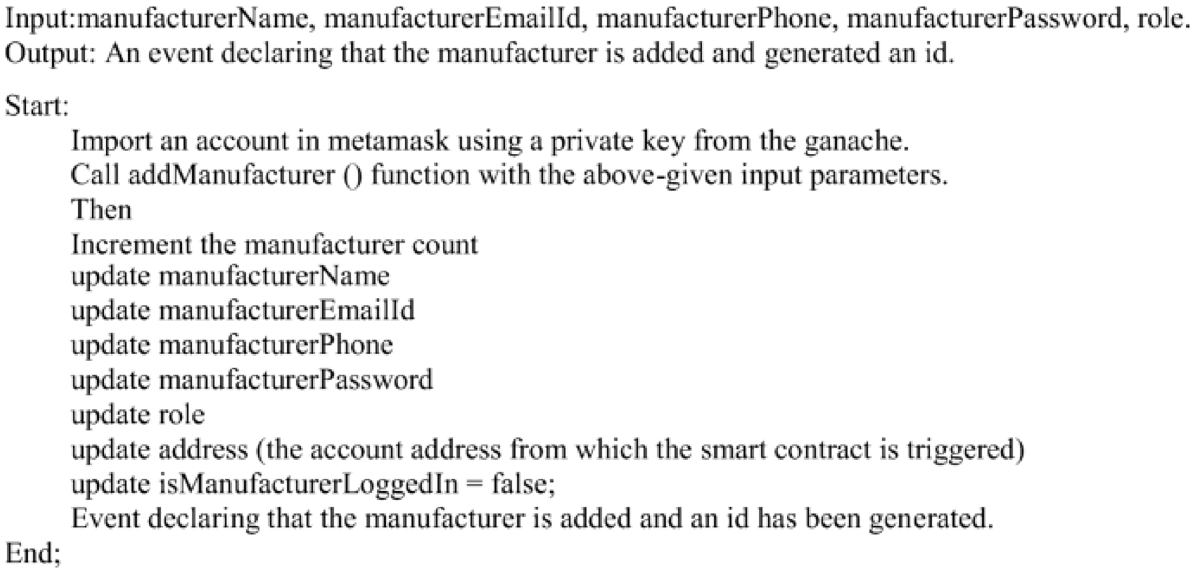
.

Fig. 2


Manufacturer
Wholesaler
DApp
Distributor
Consumer/Pharmacist
1. Sign up for manufacturer
2. Login as Manufacturer
Login Successfully
3. Add Medicine
Medicine Added Successfully
4. View Medicine
5. Purchased medicine by Wholesaler
6. Shipped medicine by Manufacturer
7. Received medicine by Wholesaler
Medicine received successfully
8. Sign up for Wholesaler
9. Login as Wholesaler
Login Successfully
10. Purchased medicine by Distributor
11. Shipped medicine by Wholesaler
12. Received medicine by Distributor
Medicine received successfully
13. Sign up for Distributor
14. Login as Distributor
Login Successfully
15. Sign up for Consumer
16. Login as Consumer
Login Successfully
17. View Medicine
18. Purchased medicine by consumer
19. Shipped medicine by Distributor
20. Received medicine by consumer
Medicine received successfully
21. Trace Medicine

Sequence diagram of the proposed system

Algorithm 1



```
Input:manufacturerName, manufacturerEmailId, manufacturerPhone, manufacturerPassword, role.
Output: An event declaring that the manufacturer is added and generated an id.

Start:
    Import an account in metamask using a private key from the ganache.
    Call addManufacturer () function with the above-given input parameters.
    Then
    Increment the manufacturer count
    update manufacturerName
    update manufacturerEmailId
    update manufacturerPhone
    update manufacturerPassword
    update role
    update address (the account address from which the smart contract is triggered)
    update isManufacturerLoggedIn = false;
    Event declaring that the manufacturer is added and an id has been generated.
End;
```

Creating an Account for Manufacturer

In algorithm 1 the **addManufacturer()**function is used to add the manufacturer's account to the decentralized application with the specific inputs mentioned abovein the blockchain. Similarly, addWholesaler(), addDistributor(), addPharmacy(), and addCustomer() functions are used in algorithm 1 to create an account for wholesaler, distributor, pharmacy, and customer accounts respectively by changing the variable name of the functions as per the requirement.

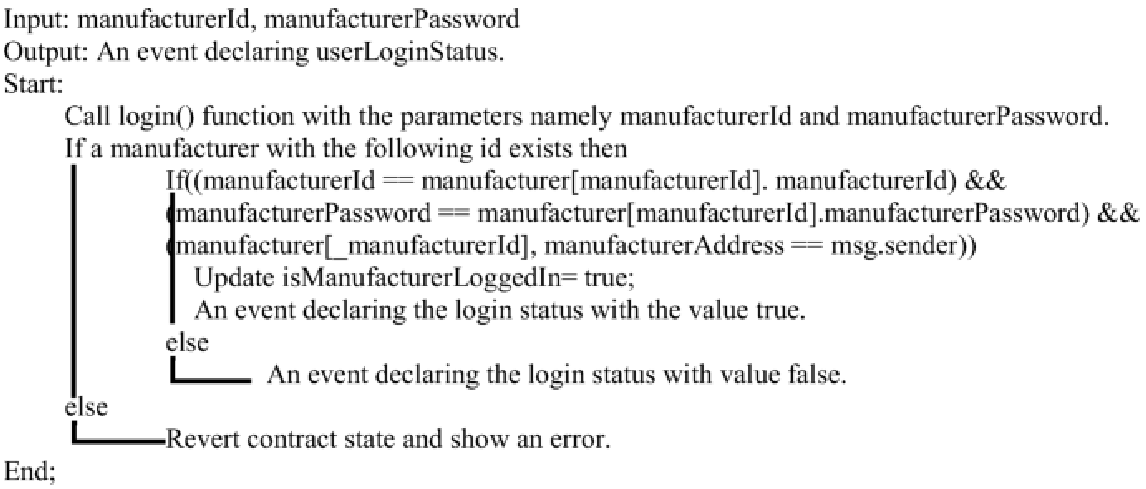
manufacturerName: is the name of the manufacturer.

manufacturerEmailId: is the email id of the manufacturer.

manufacturerPhone: is the phone number of the manufacturer.

manufacturerPassword: is the password of the manufacturer's account.

Algorithm 2



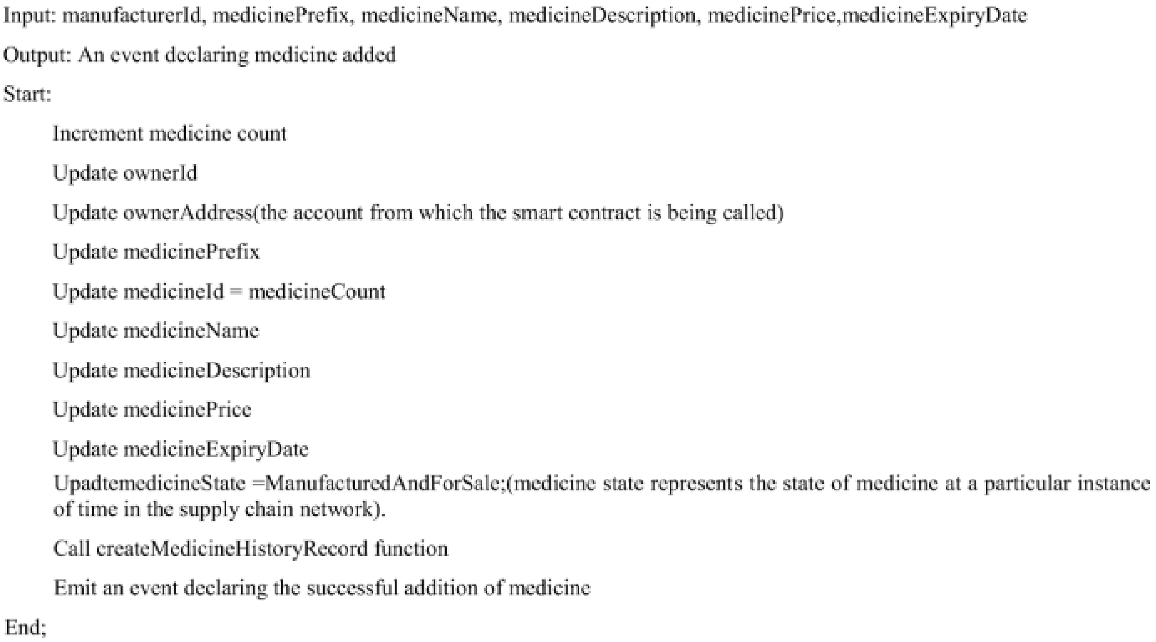
```
Input: manufacturerId, manufacturerPassword
Output: An event declaring userLoginStatus.
Start:
        Call login() function with the parameters namely manufacturerId and manufacturerPassword.
        If a manufacturer with the following id exists then
                If((manufacturerId == manufacturer[manufacturerId]. manufacturerId) &&
                (manufacturerPassword == manufacturer[manufacturerId].manufacturerPassword) &&
                (manufacturer[_manufacturerId], manufacturerAddress == msg.sender))
                  Update isManufacturerLoggedIn= true;
                  An event declaring the login status with the value true.
                else
                      An event declaring the login status with value false.
        else
                Revert contract state and show an error.
End;
```

Manufacturer Login into the Decentralized Application.

In algorithm 2, the login() function is used for the user credentials verification before login into the application.Similarly, the login() function for wholesaler, distributor, pharmacy, and customer also follows the above-given algorithm flow just with the change in the variable names and function names.
manufacturerId: is the unique id used for decentralized applications.

manufacturerPassword: is the password used for authentication purposes.

Algorithm 3



```
Input: manufacturerId, medicinePrefix, medicineName, medicineDescription, medicinePrice,medicineExpiryDate
Output: An event declaring medicine added
Start:
      Increment medicine count
      Update ownerId
      Update ownerAddress(the account from which the smart contract is being called)
      Update medicinePrefix
      Update medicineId = medicineCount
      Update medicineName
      Update medicineDescription
      Update medicinePrice
      Update medicineExpiryDate
      UpadtemedicineState =ManufacturedAndForSale;(medicine state represents the state of medicine at a particular instance
      of time in the supply chain network).
      Call createMedicineHistoryRecord function
      Emit an event declaring the successful addition of medicine
End;
```

Add Medicine.

In algorithm 3,addMedicine()function is used to add medicines to the supply chain management decentralized application. Which is further called the createMedicineHistoryRecord() function to add a medicine record and mark the beginning of the medicine life cycle into the chain.Only manufacturers can add medicines.

medicinePrefix: is the unique medicine id prefix that is generated by the javascript.

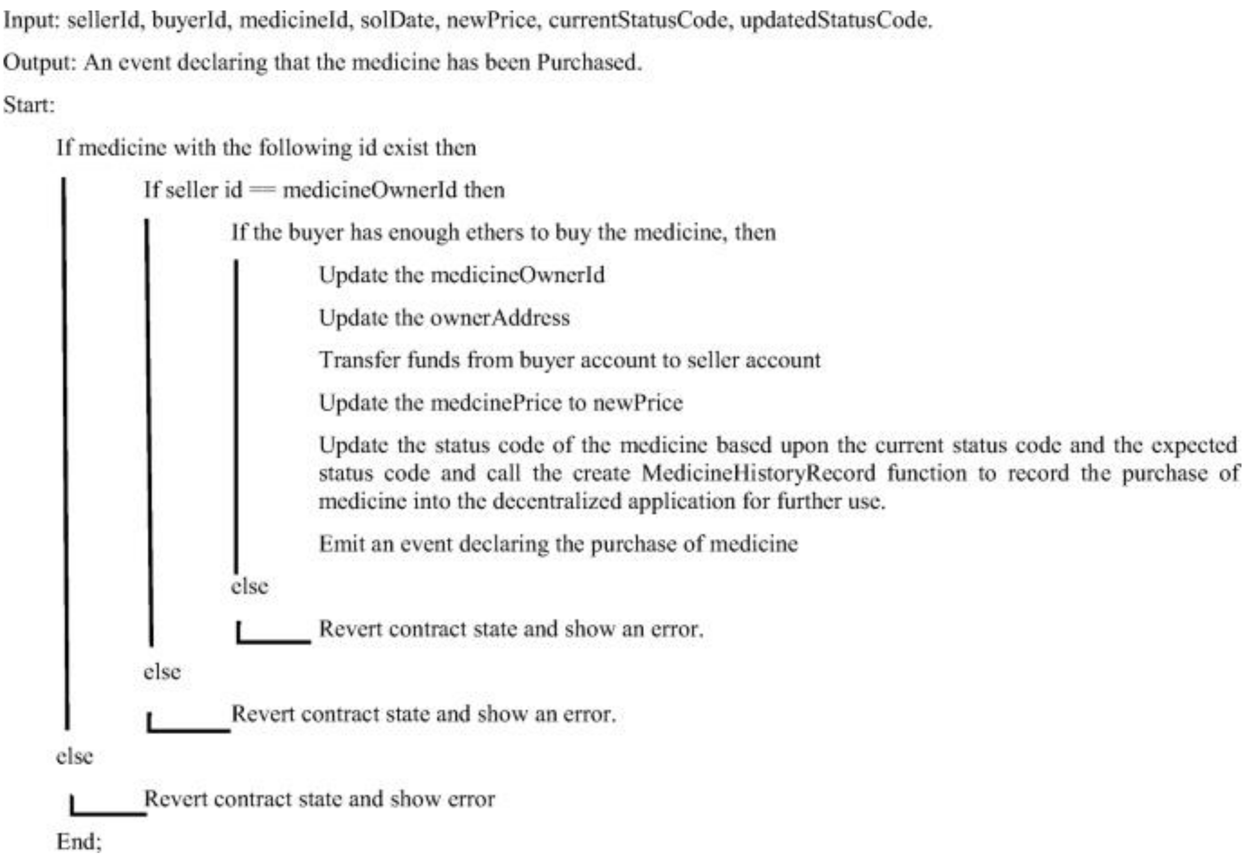
medicineName: is the name of the medicine to be added.

medicineDescription: this is a short description of the medicine.

medicinePrice: is the price associated with the medicine.

medicineExpiryDate: is the medicine expiry date.

Algorithm 4



```
Input: sellerId, buyerId, medicineId, solDate, newPrice, currentStatusCode, updatedStatusCode.
Output: An event declaring that the medicine has been Purchased.
Start:
    If medicine with the following id exist then
        If seller id == medicineOwnerId then
            If the buyer has enough ethers to buy the medicine, then
                Update the medicineOwnerId
                Update the ownerAddress
                Transfer funds from buyer account to seller account
                Update the medcinePrice to newPrice
                Update the status code of the medicine based upon the current status code and the expected
                status code and call the create MedicineHistoryRecord function to record the purchase of
                medicine into the decentralized application for further use.
                Emit an event declaring the purchase of medicine
            else
                Revert contract state and show an error.
        else
            Revert contract state and show an error.
    else
        Revert contract state and show error
    End;
```

Purchasing a Medicine

In algorithm 4,purchaseMedicine()function is used to make the purchase of medicine and record it into the decentralized application.Every stakeholder follows algorithm 4 to buy medicine in the application.

sellerId: is the unique id of the seller of the medicine.

buyerId: is the unique id of the buyer of the medicine.

medicineId: is the id associated with a particular medicine.

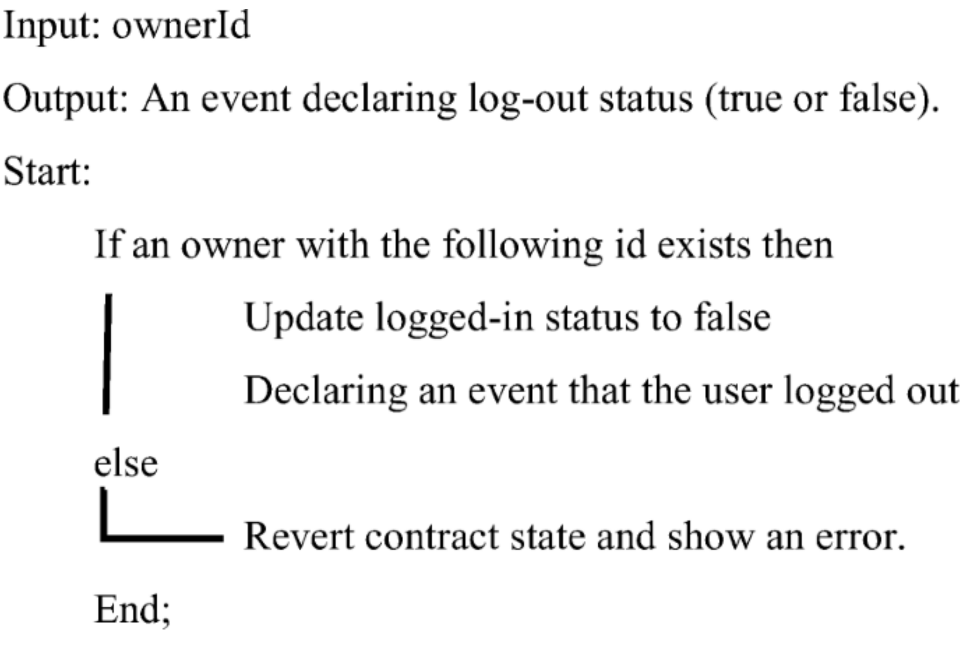
solDate: is the date on which the purchase is being made.

newPrice: is the updated price or the expected new price after the state change of medicine.

currentStatusCode: is the status code associated with the current state of medicine.

updatedStatusCode: is the expected change in the state of the medicine once the medicine is being bought by a particular stakeholder.

Algorithm 5



```
Input: ownerId
Output: An event declaring log-out status (true or false).
Start:
    If an owner with the following id exists then
    |       Update logged-in status to false
    |       Declaring an event that the user logged out
    else
    └────── Revert contract state and show an error.
    End;
```

Logout from the System

In algorithm 5, the logout () function is used to change the status of the owner of the account to log out and thus exit from the decentralized application. Logout for wholesaler, distributor, pharmacy, and customer also follow algorithm 5 just with the change in the variable name. ownerId: is the id of the stakeholder currently logged in into the system.

Testing and validation:

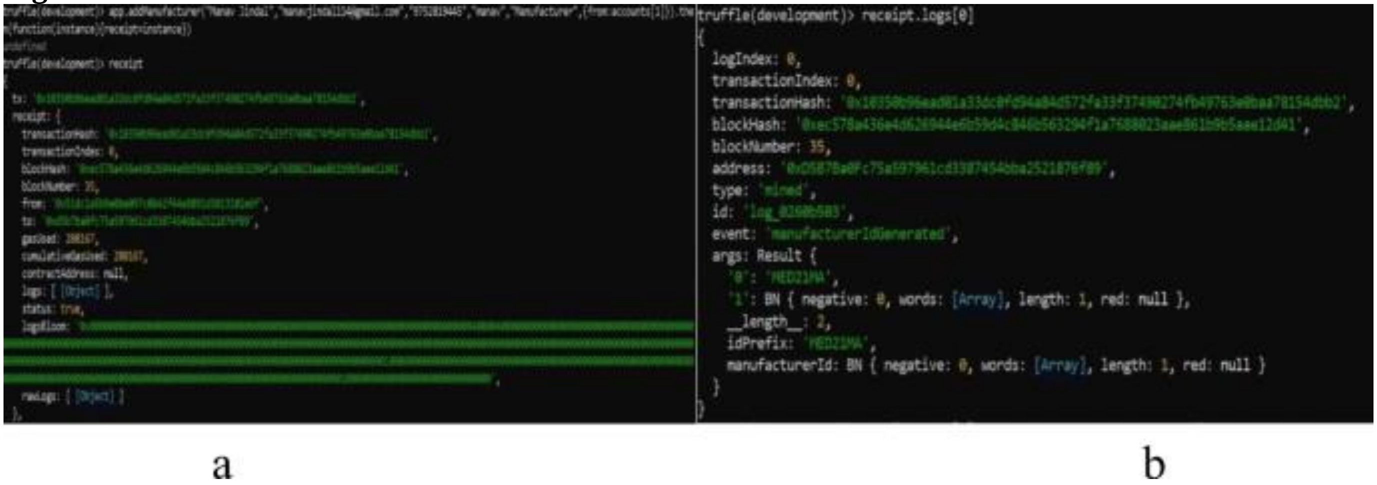
- To assess the smart contracts developed via Ethereum, Truffle console and the testing environment were used to test and validate different functions. The scenarios involved five accounts with different participants each representing one stakeholder of the chain and their corresponding Ethereum Addresses as presented in Table 3.

Table 3 Accounts address used to test the functions

Participants	Ethereum Address
Manufacture	0x51dC1A5B9e0be097c0b42F44E9891d3813102e6f
Wholesaler	0xF844C694d6a4ECAadcfF5B948e9389108DcA3Bd0
Distributer	0xDc7976B23EAa47E73D4CFAB4CA4943dAb395aB59
Pharmacy	0xBE47d7ae29c257fdb341c7E1A1aDFBCeFd5D90d3
Customer	0x652e7Ced6DdBda36819d6A206deB533DeC4c1dF2

addManufacturer(): In this function, it was tested whether a new stakeholder possessing the role of a manufacturer is able to create a new account on the system or not with the specific details. If the account is created then an event is triggered. Successful execution of the function and its corresponding logs and events are displayed in Fig. 3a and b resepectively.

Fig.



```
truffle(development)> app.addManufacturer("Manav Jindal","manavjindal1134@gmail.com","9752819445","manav","Manufacturer",{from:accounts[1]}).the
n(function(instance){receipt=instance})
undefined
truffle(development)> receipt
{
  tx: '[illegible]',
  receipt: {
    transactionHash: '[illegible]',
    transactionIndex: 0,
    blockHash: '[illegible]',
    blockNumber: 35,
    from: '[illegible]',
    to: '[illegible]',
    gasUsed: 208167,
    cumulativeGasUsed: 208167,
    contractAddress: null,
    logs: [ [Object] ],
    status: true,
    logsBloom: '[illegible]',
    rawLogs: [ [Object] ]
  },
```

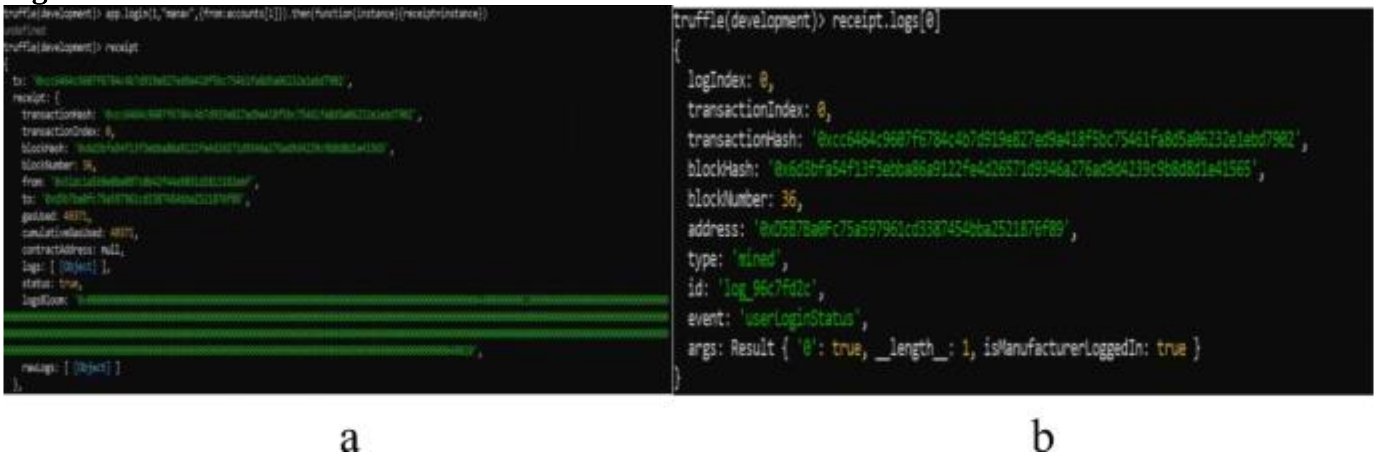
```
truffle(development)> receipt.logs[0]
{
  logIndex: 0,
  transactionIndex: 0,
  transactionHash: '0x10350b96ead01a33dc0fd94a84d572fa33f37490274fb49763e0baa78154dbb2',
  blockHash: '0xec578a436e4d628944e6b59d4c840b563294f1a7688023aae861b9b5aae12d41',
  blockNumber: 35,
  address: '0xD587Ba0Fc75a597961cd3387454bba2521876f89',
  type: 'mined',
  id: 'log_0260b583',
  event: 'manufacturerIdGenerated',
  args: Result {
    '0': 'MED21MA',
    '1': BN { negative: 0, words: [Array], length: 1, red: null },
    __length__: 2,
    idPrefix: 'MED21MA',
    manufacturerId: BN { negative: 0, words: [Array], length: 1, red: null }
  }
}
```

addManufacturer() execution (**a**) and log generation (**b**)

Similarly **addWholesaler(), addDistributer(), addPharmacy() and addCustomer()** are tested if a stakeholder is able to create an account with the respective role.

Login(): This function is used to login stakeholders into the decentralized system. Once the function gets executed without any error an event is triggered displaying the successful execution of the function. Here we have displayed the login activity of the manufacturer with the successful execution of the function and its corresponding logs and events in Figs. 4a and b respectively.

Fig. 4



```
truffle(development)> app.login(1,"manan",{from:accounts[1]}).then(function(instance){receipt=instance})
undefined
truffle(development)> receipt
{
  tx: '[illegible]',
  receipt: {
    transactionHash: '[illegible]',
    transactionIndex: 0,
    blockHash: '[illegible]',
    blockNumber: 36,
    from: '[illegible]',
    to: '[illegible]',
    gasUsed: [illegible],
    cumulativeGasUsed: [illegible],
    contractAddress: null,
    logs: [ [Object] ],
    status: true,
    logsBloom: '[illegible]',
    rawLogs: [ [Object] ]
  },
```

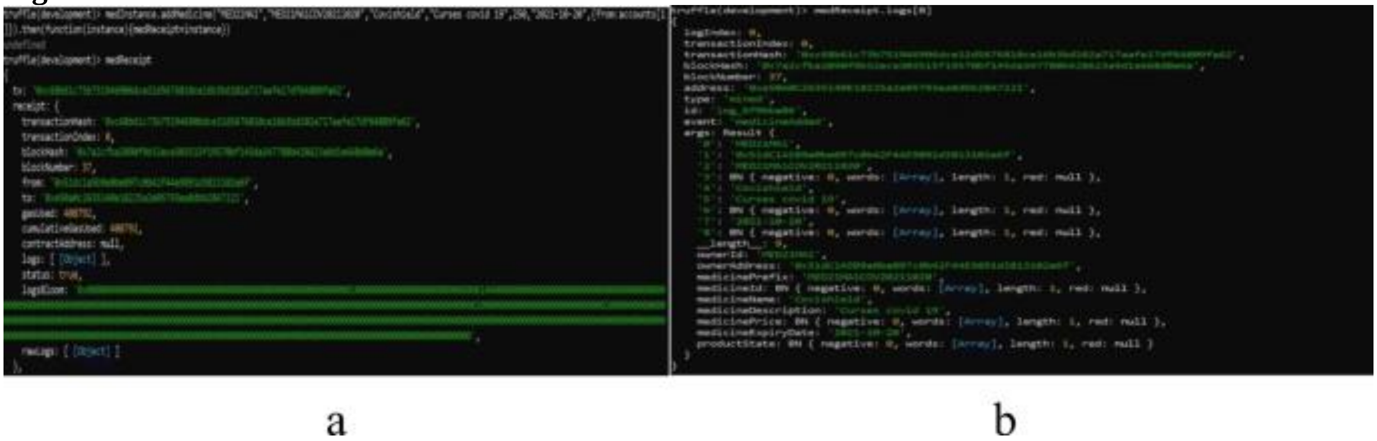
```
truffle(development)> receipt.logs[0]
{
  logIndex: 0,
  transactionIndex: 0,
  transactionHash: '0xcc6464c9607f6784c4b7d919e827ed9a418f5bc75461fa8d5a06232e1ebd7982',
  blockHash: '0x6d3bfa54f13f3ebba86a9122fe4d26571d9346a276ad9d4139c9b8d8d1e41565',
  blockNumber: 36,
  address: '0xD5878a0Fc75a597961cd3387454bba2521876F89',
  type: 'mined',
  id: 'log_96c7fd2c',
  event: 'userLoginStatus',
  args: Result { '0': true, __length__: 1, isManufacturerLoggedIn: true }
}
```

Login() execution (**a**) and log generation (**b**)

Similarly, the function will work for all the other stakeholders.

addMedicine(): This function is used by the manufacturer to add medicine to the decentralized application and to initiate the supply chain flow by marking the medicine for sale. If the medicine gets added without encountering any error,then an event is declared to all the participants of the chain the successful addition of the medicine else the state of the contract is reverted, and an error is thrown. Successful execution of the function and its corresponding logs and events are displayed in Figs. 5a and b respectively.

Fig. 5



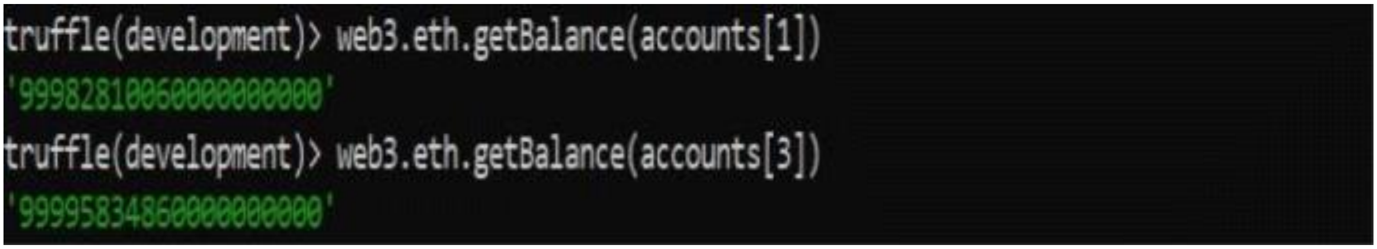
```
truffle(development)> medInstance.addMedicine("[illegible]","[illegible]","Covishield","Cures covid 19",250,"2021-10-20",{from:accounts[1
]}).then(function(instance){medReceipt=instance})
undefined
truffle(development)> medReceipt
{
  tx: '[illegible]',
  receipt: {
    transactionHash: '[illegible]',
    transactionIndex: 0,
    blockHash: '[illegible]',
    blockNumber: 37,
    from: '[illegible]',
    to: '[illegible]',
    gasUsed: [illegible],
    cumulativeGasUsed: [illegible],
    contractAddress: null,
    logs: [ [Object] ],
    status: true,
    logsBloom: '[illegible]',
    rawLogs: [ [Object] ]
  },
```

```
truffle(development)> medReceipt.logs[0]
{
  logIndex: 0,
  transactionIndex: 0,
  transactionHash: '[illegible]',
  blockHash: '[illegible]',
  blockNumber: 37,
  address: '[illegible]',
  type: 'mined',
  id: '[illegible]',
  event: 'medicineAdded',
  args: Result {
    '0': '[illegible]',
    '1': '[illegible]',
    '2': '[illegible]',
    '3': BN { negative: 0, words: [Array], length: 1, red: null },
    '4': 'Covishield',
    '5': 'Cures covid 19',
    '6': BN { negative: 0, words: [Array], length: 1, red: null },
    '7': '2021-10-20',
    '8': BN { negative: 0, words: [Array], length: 1, red: null },
    __length__: 9,
    ownerId: '[illegible]',
    ownerAddress: '[illegible]',
    medicinePrefix: '[illegible]',
    medicineId: BN { negative: 0, words: [Array], length: 1, red: null },
    medicineName: 'Covishield',
    medicineDescription: 'Cures covid 19',
    medicinePrice: BN { negative: 0, words: [Array], length: 1, red: null },
    medicineExpiryDate: '2021-10-20',
    productState: BN { negative: 0, words: [Array], length: 1, red: null }
  }
}
```

addMedicine() execution (**a**) and log generation (**b**)

purchaseMedicine(): This function is used by the following stakeholders for the purchase of medicine either from the same level stakeholder or from the one just above them in the supply chain network. When this function is called some specific checks are being made and the ownership of medicine is transferred to the new owner after the successful transfer of the fund. At last, a record is being recorded in the history with the createMedicineHistoryRecord function and an event is declared. If by chance any security check fails then the contract state is reverted and an error is shown. Figures 6 and 7 shows the manufacture and wholesaler balance before purchase and Fig. 8 shows the manufacture and wholesaler balance after purchase. The Successful execution of the function and its corresponding logs and events are displayed in Fig. 7a and b respectively.

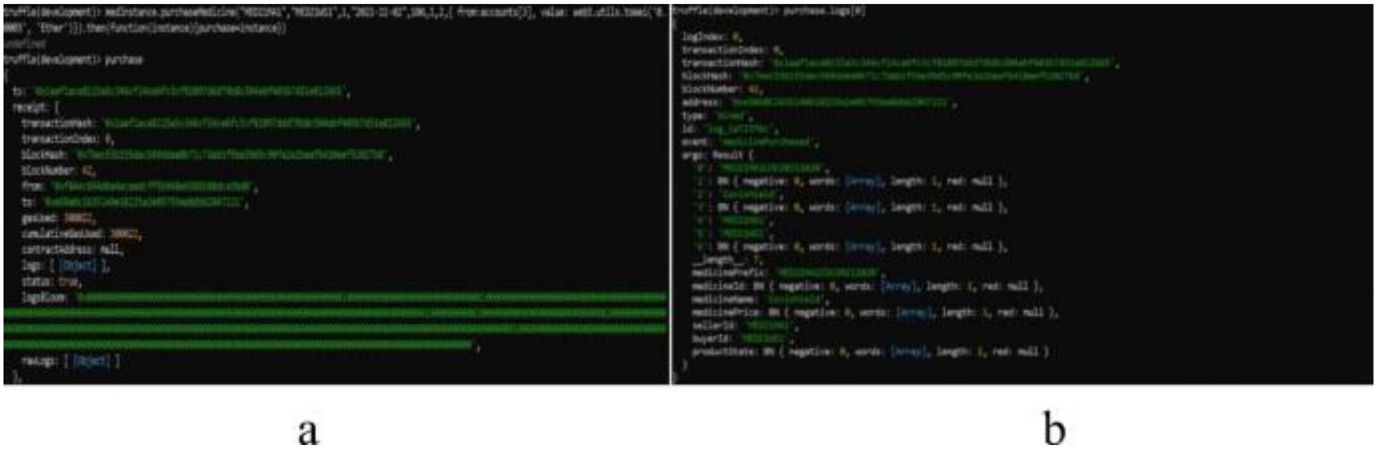
Fig. 6



```
truffle(development)> web3.eth.getBalance(accounts[1])
'99982810060000000000'
truffle(development)> web3.eth.getBalance(accounts[3])
'99995834860000000000'
```

Manufacturer (accounts [38]) and wholesaler (accounts [29]) balance before purchase

Fig. 7

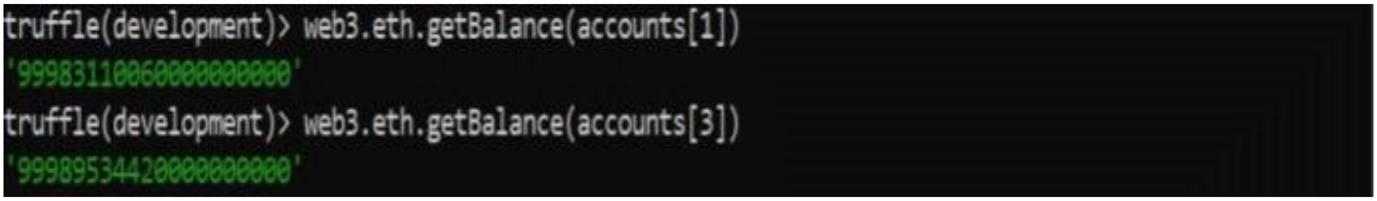


```
truffle(development)> medInstance.purchaseMedicine("MED1941","MED2341",1,"2021-11-02",100,1,2,{ from:accounts[3], value: web3.utils.toWei('0.
0001', 'Ether')}).then(function(instance){purchase=instance})
undefined
truffle(development)> purchase
{
  tx: '[illegible]',
  receipt: {
    transactionHash: '[illegible]',
    transactionIndex: 0,
    blockHash: '[illegible]',
    blockNumber: 42,
    from: '[illegible]',
    to: '[illegible]',
    gasUsed: [illegible],
    cumulativeGasUsed: [illegible],
    contractAddress: null,
    logs: [ [Object] ],
    status: true,
    logsBloom: '[illegible]',
    rawLogs: [ [Object] ]
  },
```

```
truffle(development)> purchase.logs[0]
{
  logIndex: 0,
  transactionIndex: 0,
  transactionHash: '[illegible]',
  blockHash: '[illegible]',
  blockNumber: 42,
  address: '[illegible]',
  type: 'mined',
  id: 'log_[illegible]',
  event: 'medicinePurchased',
  args: Result {
    '0': '[illegible]',
    '1': BN { negative: 0, words: [Array], length: 1, red: null },
    '2': '[illegible]',
    '3': BN { negative: 0, words: [Array], length: 1, red: null },
    '4': '[illegible]',
    '5': '[illegible]',
    '6': BN { negative: 0, words: [Array], length: 1, red: null },
    __length__: 7,
    medicinePrefix: '[illegible]',
    medicineId: BN { negative: 0, words: [Array], length: 1, red: null },
    medicineName: '[illegible]',
    medicinePrice: BN { negative: 0, words: [Array], length: 1, red: null },
    sellerId: '[illegible]',
    buyerId: '[illegible]',
    productState: BN { negative: 0, words: [Array], length: 1, red: null }
  }
}
```

purchaseMedicine() execution (**a**) and log generation (**b**)

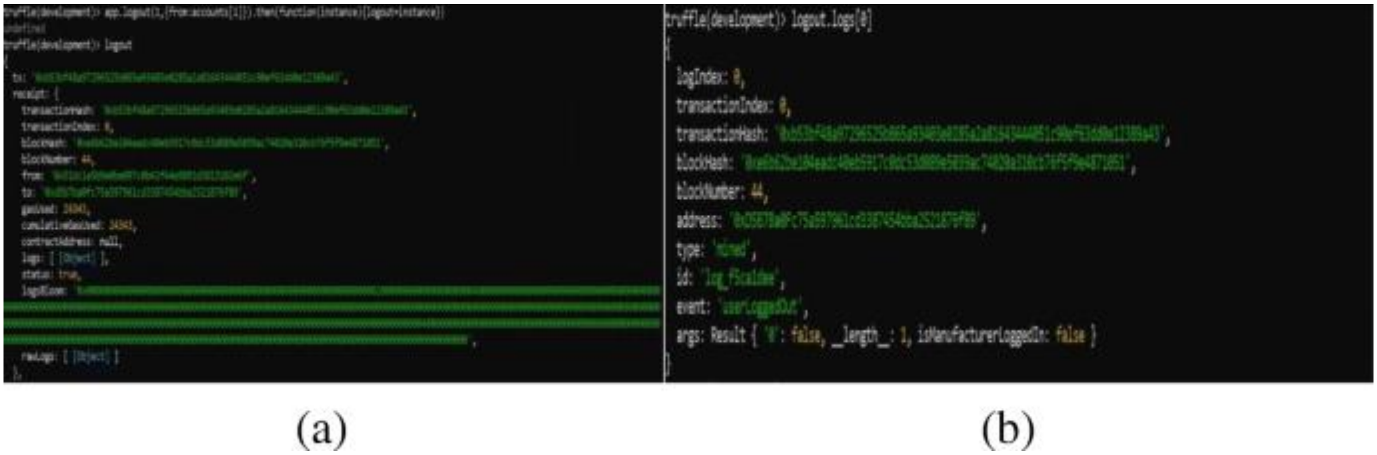
Fig. 8



```
truffle(development)> web3.eth.getBalance(accounts[1])
'99983110060000000000'
truffle(development)> web3.eth.getBalance(accounts[3])
'99989534420000000000'
```

Manufacturer (accounts [38]) and wholesaler (accounts [29]) balance after purchase

Fig. 9



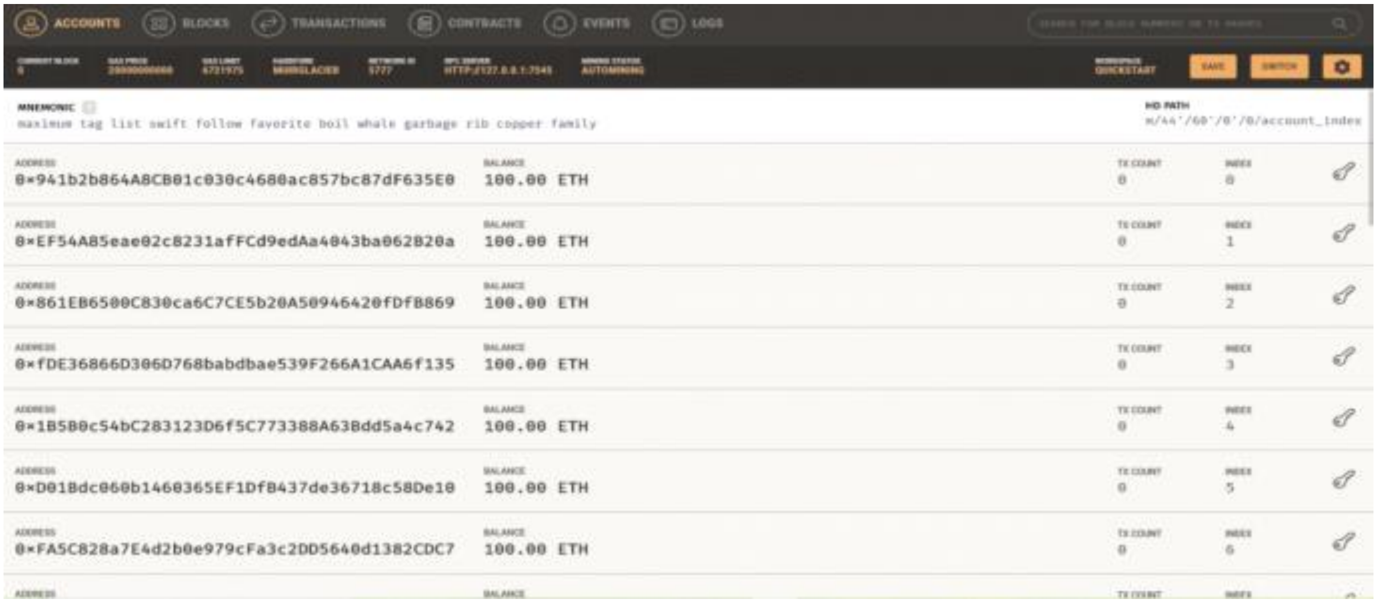
```
truffle(development)> app.logout(1,{from:accounts[1]}).then(function(instance){logout=instance})
undefined
truffle(development)> logout
{
  tx: '[illegible]',
  receipt: {
    transactionHash: '[illegible]',
    transactionIndex: 0,
    blockHash: '[illegible]',
    blockNumber: 44,
    from: '[illegible]',
    to: '[illegible]',
    gasUsed: 34343,
    cumulativeGasUsed: 34343,
    contractAddress: null,
    logs: [ [Object] ],
    status: true,
    logsBloom: '[illegible]',
    rawLogs: [ [Object] ]
  },
```

```
truffle(development)> logout.logs[0]
{
  logIndex: 0,
  transactionIndex: 0,
  transactionHash: '0xb53bf48a9729652b865a93465e8285a2a8194344851c90ef63dde11389a43',
  blockHash: '0xe0b62be104eadc40eb5917c0dc53d009e5859ac74820a310c676f5f9e487105l',
  blockNumber: 44,
  address: '0xD5878a0Fc75a89796lcd0387454bba252187bf89',
  type: 'mined',
  id: 'log_f5ca1dee',
  event: 'userLoggedOut',
  args: Result { '0': false, __length__: 1, isManufacturerLoggedIn: false }
}
```

(a) (b)

Logout execution (**a**) and log generation (**b**)

Fig. 10



CURRENT BLOCK 0 | GAS PRICE 20000000000 | GAS LIMIT 6721975 | HARDFORK MUIRGLACIER | NETWORK ID 5777 | RPC SERVER HTTP://127.0.0.1:7545 | MINING STATUS AUTOMINING | WORKSPACE QUICKSTART

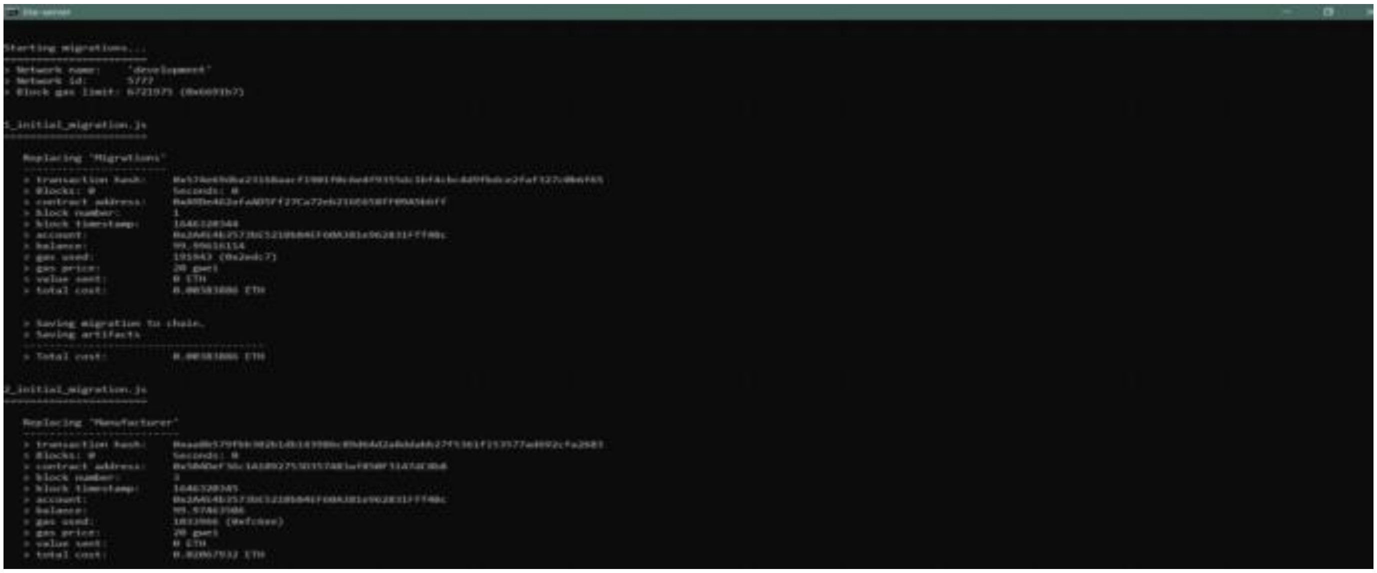
MNEMONIC
maximum tag list swift follow favorite boil whale garbage rib copper family

HD PATH
m/44'/60'/0'/0/account_index

ADDRESS	BALANCE	TX COUNT	INDEX
0×941b2b864A8CB01c030c4680ac857bc87dF635E0	100.00 ETH	0	0
0×EF54A85eae02c8231afFCd9edAa4043ba062820a	100.00 ETH	0	1
0×861EB6500C830ca6C7CE5b20A50946420fDfB869	100.00 ETH	0	2
0×fDE36866D306D768babdbae539F266A1CAA6f135	100.00 ETH	0	3
0×1B5B0c54bC283123D6f5C773388A63Bdd5a4c742	100.00 ETH	0	4
0×D018dc060b1460365EF1DfB437de36718c58De10	100.00 ETH	0	5
0×FA5C828a7E4d2b0e979cFa3c2DD5640d1382CDC7	100.00 ETH	0	6

Ganache accounts

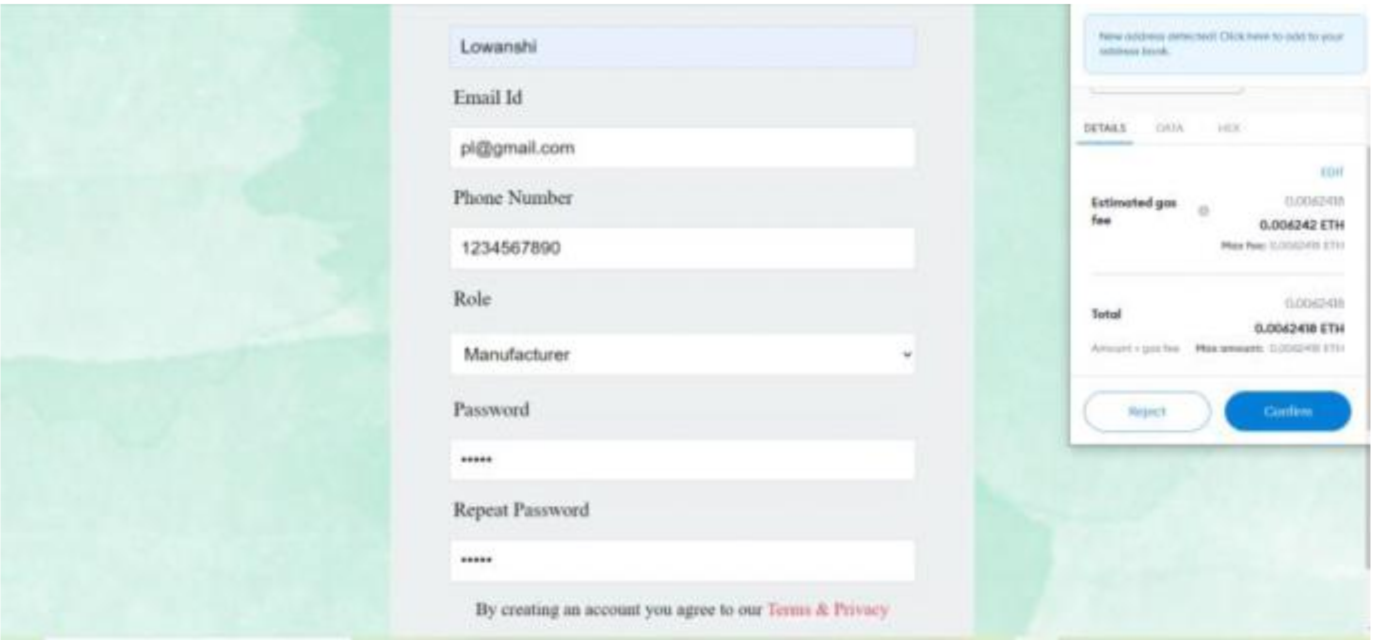
Fig. 11



```
Starting migrations...
======================
> Network name:    'development'
> Network id:      5777
> Block gas limit: 6721975 (0x6691b7)


1_initial_migration.js
======================

   Replacing 'Migrations'
   ----------------------
   > transaction hash:    [illegible]
   > Blocks: 0            Seconds: 0
   > contract address:    [illegible]
   > block number:        1
   > block timestamp:     1646320344
   > account:             [illegible]
   > balance:             99.99616114
   > gas used:            191943 (0x2edc7)
   > gas price:           20 gwei
   > value sent:          0 ETH
   > total cost:          0.00383886 ETH


   > Saving migration to chain.
   > Saving artifacts
   -------------------------------------
   > Total cost:          0.00383886 ETH


2_initial_migration.js
======================

   Replacing 'Manufacturer'
   ------------------------
   > transaction hash:    [illegible]
   > Blocks: 0            Seconds: 0
   > contract address:    [illegible]
   > block number:        3
   > block timestamp:     1646320345
   > account:             [illegible]
   > balance:             99.97463986
   > gas used:            1033966 (0xfc6ee)
   > gas price:           20 gwei
   > value sent:          0 ETH
   > total cost:          0.02067932 ETH
```

Contract deployment

Fig. 12



Lowanshi

Email Id

pl@gmail.com

Phone Number

1234567890

Role

Manufacturer

Password

*****

Repeat Password

*****

By creating an account you agree to our Terms & Privacy

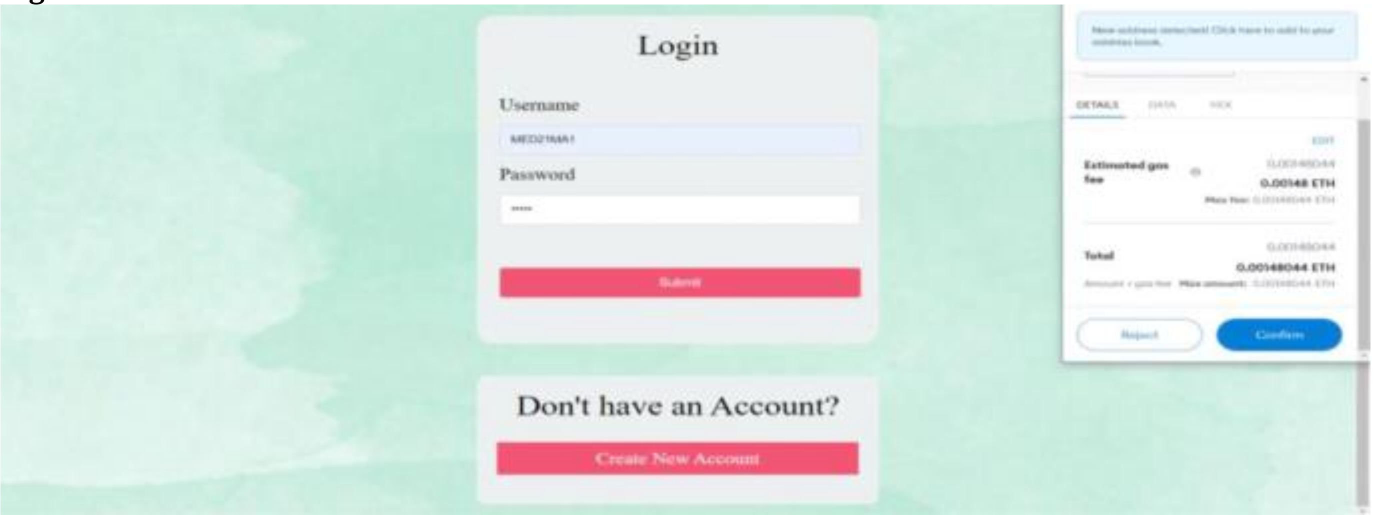
New address detected! Click here to add to your address book.

Estimated gas fee 0.0062418 0.006242 ETH
Max fee: 0.0062418 ETH

Total 0.0062418 0.0062418 ETH
Amount + gas fee Max amount: 0.0062418 ETH

Account creation

Fig. 13


Login

Username
MED21MA1

Password
*****

Don't have an Account?

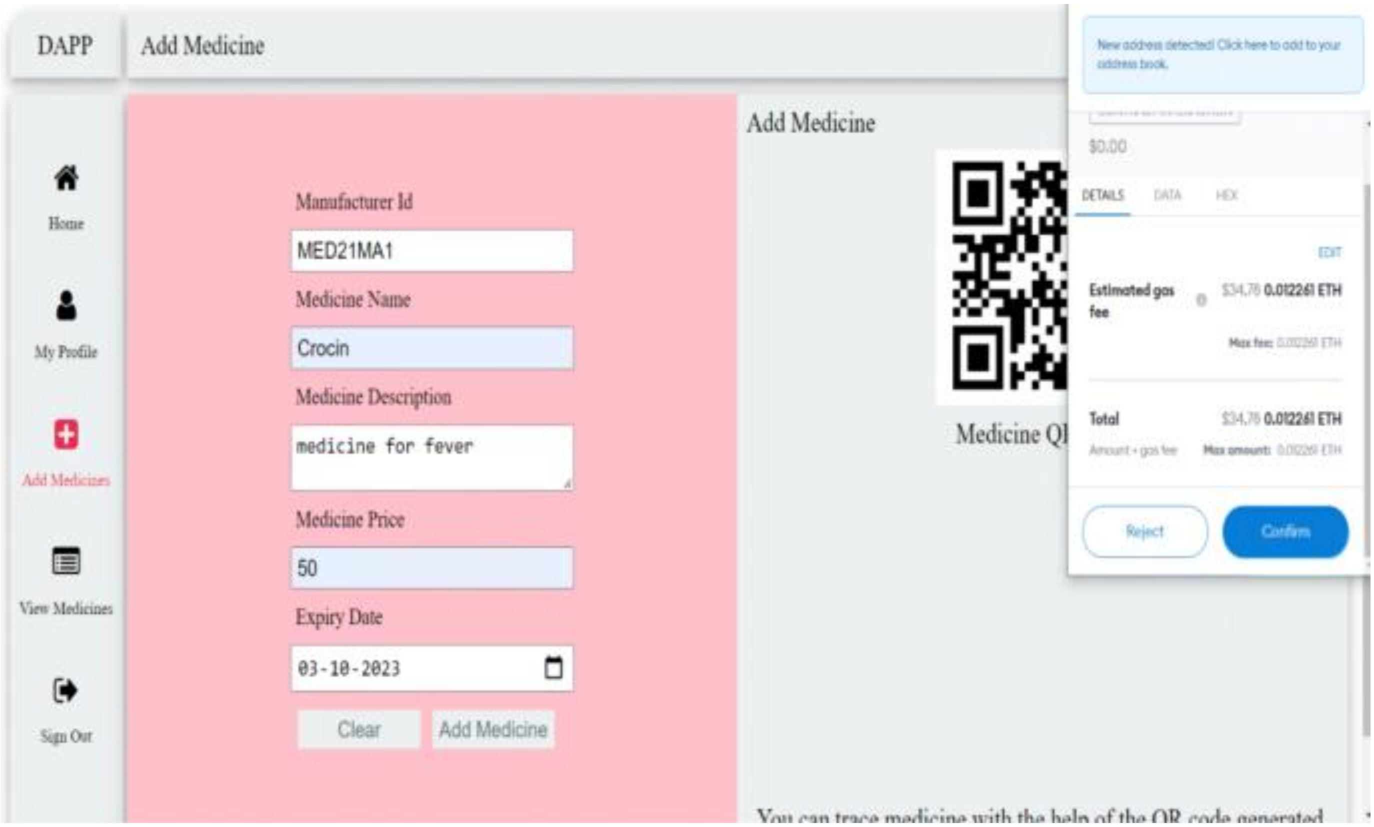
New address detected! Click here to add to your address book.

Estimated gas fee 0.00148044 0.00148 ETH
Max fee: 0.00148044 ETH

Total 0.00148044 0.00148044 ETH
Amount + gas fee Max amount: 0.00148044 ETH

Stake holder login

Fig. 14


DAPP
Add Medicine

Manufacturer Id
MED21MA1

Medicine Name
Crocin

Medicine Description
medicine for fever

Medicine Price
50

Expiry Date
03-10-2023

Add Medicine

Medicine QI

You can trace medicine with the help of the QR code generated

New address detected! Click here to add to your address book.

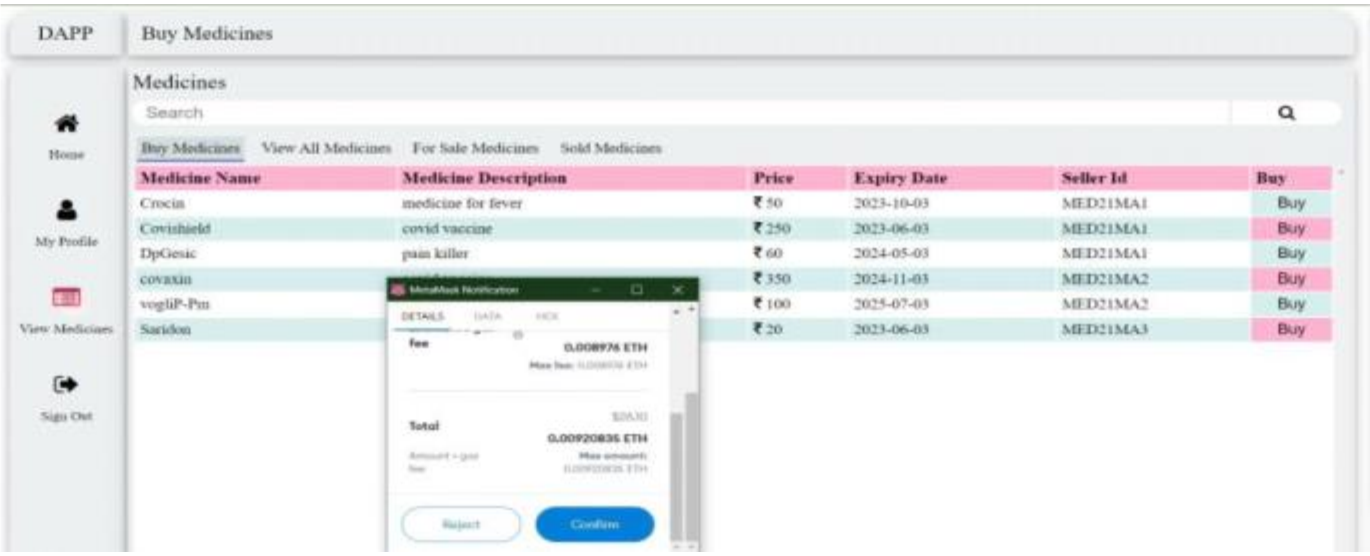
$0.00

Estimated gas fee $34.76 0.012261 ETH
Max fee: 0.012261 ETH

Total $34.76 0.012261 ETH
Amount + gas fee Max amount: 0.012261 ETH

Manufacturer add medicine

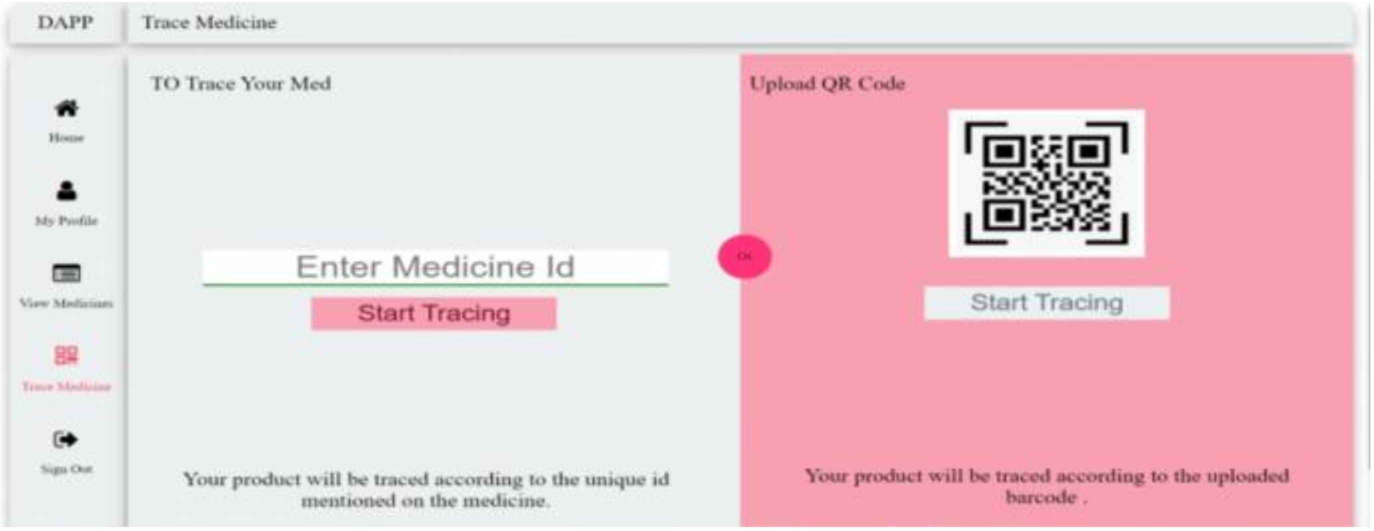
Fig. 15



Medicine Name	Medicine Description	Price	Expiry Date	Seller Id	Buy
Crocin	medicine for fever	₹ 50	2023-10-03	MED21MA1	Buy
Covishield	covid vaccine	₹ 250	2023-06-03	MED21MA1	Buy
DpGesic	pain killer	₹ 60	2024-05-03	MED21MA1	Buy
covaxin	[illegible]	₹ 350	2024-11-03	MED21MA2	Buy
vogliP-Pm		₹ 100	2025-07-03	MED21MA2	Buy
Saridon		₹ 20	2023-06-03	MED21MA3	Buy

Buy medicine window

Fig. 16

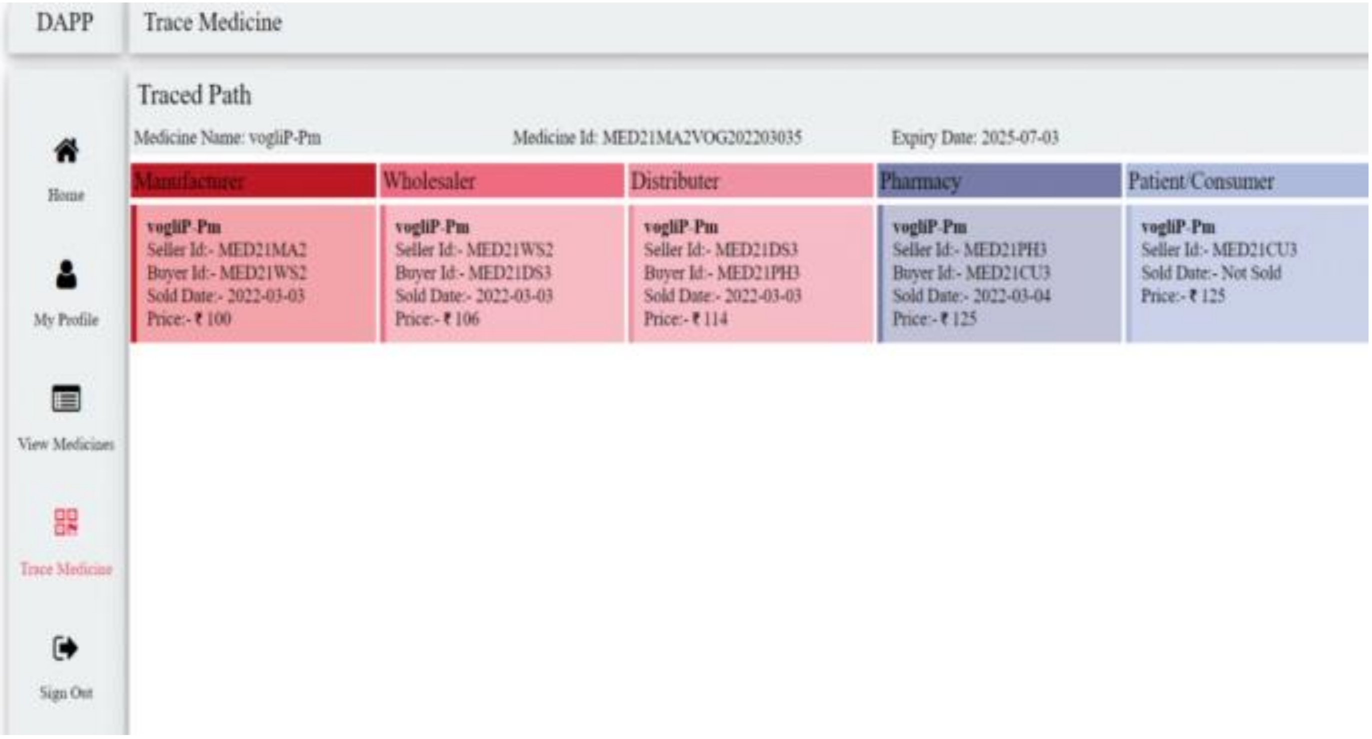


Your product will be traced according to the unique id mentioned on the medicine.

Your product will be traced according to the uploaded barcode .

Trace medicine window

Fig. 17



Trace Medicine

Traced Path

Medicine Name: vogliP-Pm Medicine Id: MED21MA2VOG202203035 Expiry Date: 2025-07-03

Manufacturer	Wholesaler	Distributer	Pharmacy	Patient/Consumer
vogliP-Pm Seller Id:- MED21MA2 Buyer Id:- MED21WS2 Sold Date:- 2022-03-03 Price:- ₹ 100	**vogliP-Pm** Seller Id:- MED21WS2 Buyer Id:- MED21DS3 Sold Date:- 2022-03-03 Price:- ₹ 106	**vogliP-Pm** Seller Id:- MED21DS3 Buyer Id:- MED21PH3 Sold Date:- 2022-03-03 Price:- ₹ 114	**vogliP-Pm** Seller Id:- MED21PH3 Buyer Id:- MED21CU3 Sold Date:- 2022-03-04 Price:- ₹ 125	**vogliP-Pm** Seller Id:- MED21CU3 Sold Date:- Not Sold Price:- ₹ 125

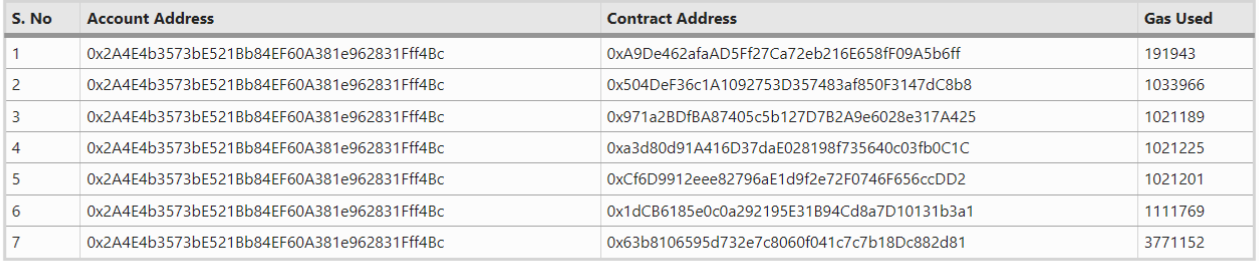
Full trace of medicine

logout: This function is used to log out users from the decentralized application and change status to logged out. Once the function gets executed without any error an event is triggered displaying the successful execution of the function. Successful execution of the function and its corresponding logs and events are displayed in Figs. 9a and b respectively.

Figure 11 shows the deployment of the smart contracts on the console system console

Table 4 shows the amount of gas used to deploy a smart contract on the blockchain network and the contract address generated. The account address for all the smart contracts is the same because the system has used only one account to deploy all the smart contracts.

Table 4 Smart contract deployment



S. No	Account Address	Contract Address	Gas Used
1	0x2A4E4b3573bE521Bb84EF60A381e962831Fff4Bc	0xA9De462afaAD5Ff27Ca72eb216E658fF09A5b6ff	191943
2	0x2A4E4b3573bE521Bb84EF60A381e962831Fff4Bc	0x504DeF36c1A1092753D357483af850F3147dC8b8	1033966
3	0x2A4E4b3573bE521Bb84EF60A381e962831Fff4Bc	0x971a2BDfBA87405c5b127D7B2A9e6028e317A425	1021189
4	0x2A4E4b3573bE521Bb84EF60A381e962831Fff4Bc	0xa3d80d91A416D37daE028198f735640c03fb0C1C	1021225
5	0x2A4E4b3573bE521Bb84EF60A381e962831Fff4Bc	0xCf6D9912eee82796aE1d9f2e72F0746F656ccDD2	1021201
6	0x2A4E4b3573bE521Bb84EF60A381e962831Fff4Bc	0x1dCB6185e0c0a292195E31B94Cd8a7D10131b3a1	1111769
7	0x2A4E4b3573bE521Bb84EF60A381e962831Fff4Bc	0x63b8106595d732e7c8060f041c7c7b18Dc882d81	3771152

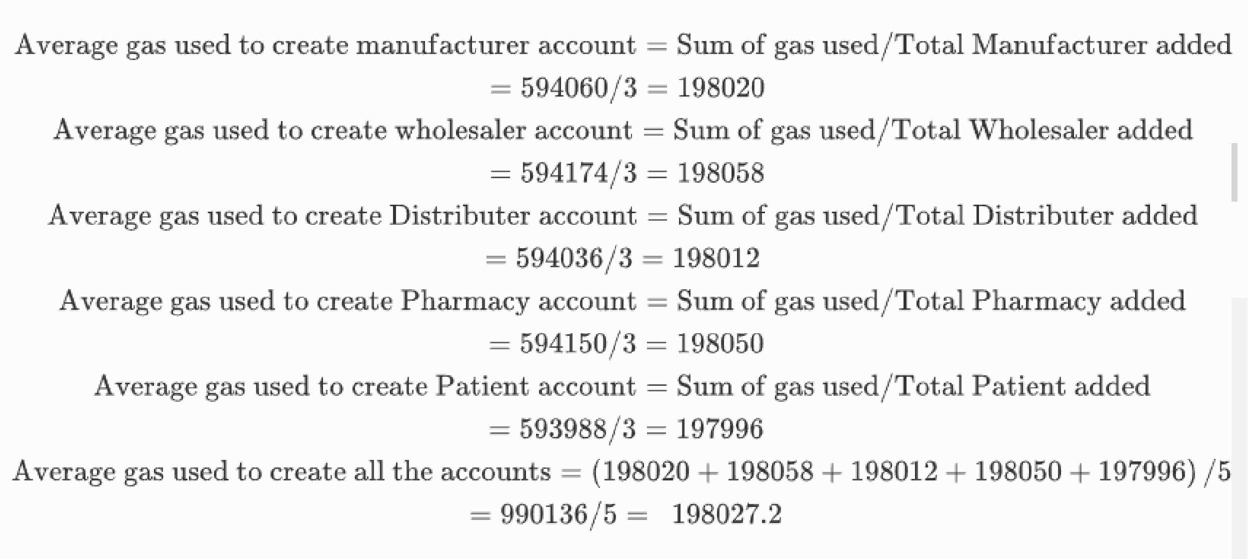
Average Gas Used for Contract Deployment = Sum of Gas Used for Contract

Deployment/Number of Contracts Deployed

=9172445/7=1310349.29

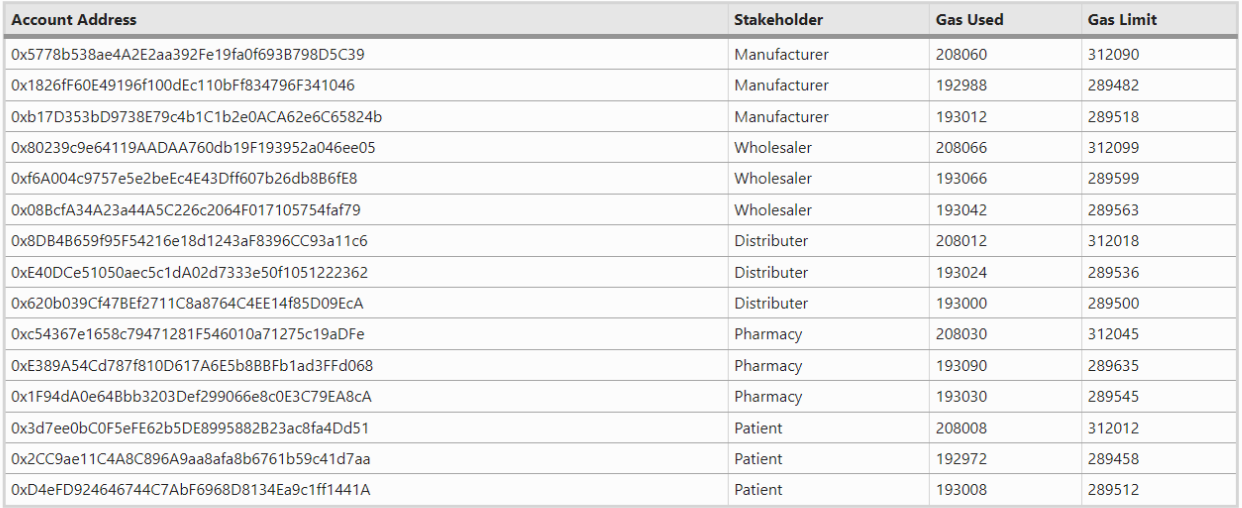
Figure 12 shows the account creation form after filling in all the information on clicking submit a connection request is generated through metamask on confirming which account is added to the chain.

Table 5 shows the amount of gas used to create an account for a particular stakeholder with a specific account address provided by the ganache. And the Gas limit is the maximum amount of gas allowed to be used for a transaction.



$$\text{Average gas used to create manufacturer account} = \text{Sum of gas used}/\text{Total Manufacturer added}$$
$$= 594060/3 = 198020$$
$$\text{Average gas used to create wholesaler account} = \text{Sum of gas used}/\text{Total Wholesaler added}$$
$$= 594174/3 = 198058$$
$$\text{Average gas used to create Distributer account} = \text{Sum of gas used}/\text{Total Distributer added}$$
$$= 594036/3 = 198012$$
$$\text{Average gas used to create Pharmacy account} = \text{Sum of gas used}/\text{Total Pharmacy added}$$
$$= 594150/3 = 198050$$
$$\text{Average gas used to create Patient account} = \text{Sum of gas used}/\text{Total Patient added}$$
$$= 593988/3 = 197996$$
$$\text{Average gas used to create all the accounts} = (198020 + 198058 + 198012 + 198050 + 197996)/5$$
$$= 990136/5 = \quad 198027.2$$

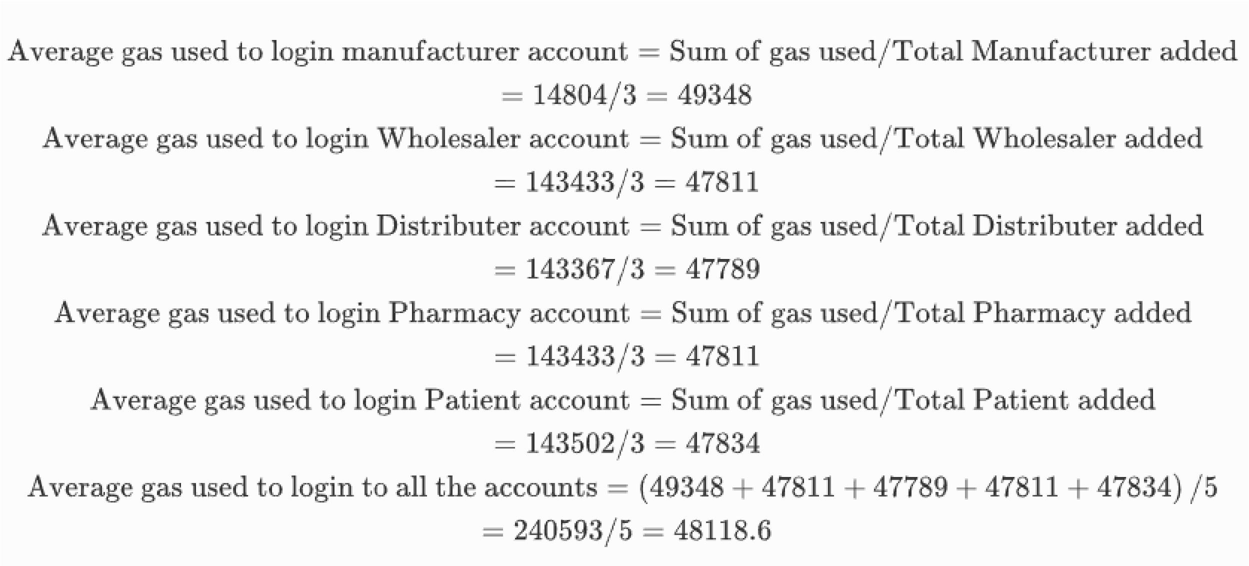
Table 5 Gas used to create stakeholder account



Account Address	Stakeholder	Gas Used	Gas Limit
0x5778b538ae4A2E2aa392Fe19fa0f693B798D5C39	Manufacturer	208060	312090
0x1826fF60E49196f100dEc110bFf834796F341046	Manufacturer	192988	289482
0xb17D353bD9738E79c4b1C1b2e0ACA62e6C65824b	Manufacturer	193012	289518
0x80239c9e64119AADAA760db19F193952a046ee05	Wholesaler	208066	312099
0xf6A004c9757e5e2beEc4E43Dff607b26db8B6fE8	Wholesaler	193066	289599
0x08BcfA34A23a44A5C226c2064F017105754faf79	Wholesaler	193042	289563
0x8DB4B659f95F54216e18d1243aF8396CC93a11c6	Distributer	208012	312018
0xE40DCe51050aec5c1dA02d7333e50f1051222362	Distributer	193024	289536
0x620b039Cf47BEf2711C8a8764C4EE14f85D09EcA	Distributer	193000	289500
0xc54367e1658c79471281F546010a71275c19aDFe	Pharmacy	208030	312045
0xE389A54Cd787f810D617A6E5b8BBFb1ad3FFd068	Pharmacy	193090	289635
0x1F94dA0e64Bbb3203Def299066e8c0E3C79EA8cA	Pharmacy	193030	289545
0x3d7ee0bC0F5eFE62b5DE8995882B23ac8fa4Dd51	Patient	208008	312012
0x2CC9ae11C4A8C896A9aa8afa8b6761b59c41d7aa	Patient	192972	289458
0xD4eFD924646744C7AbF6968D8134Ea9c1ff1441A	Patient	193008	289512

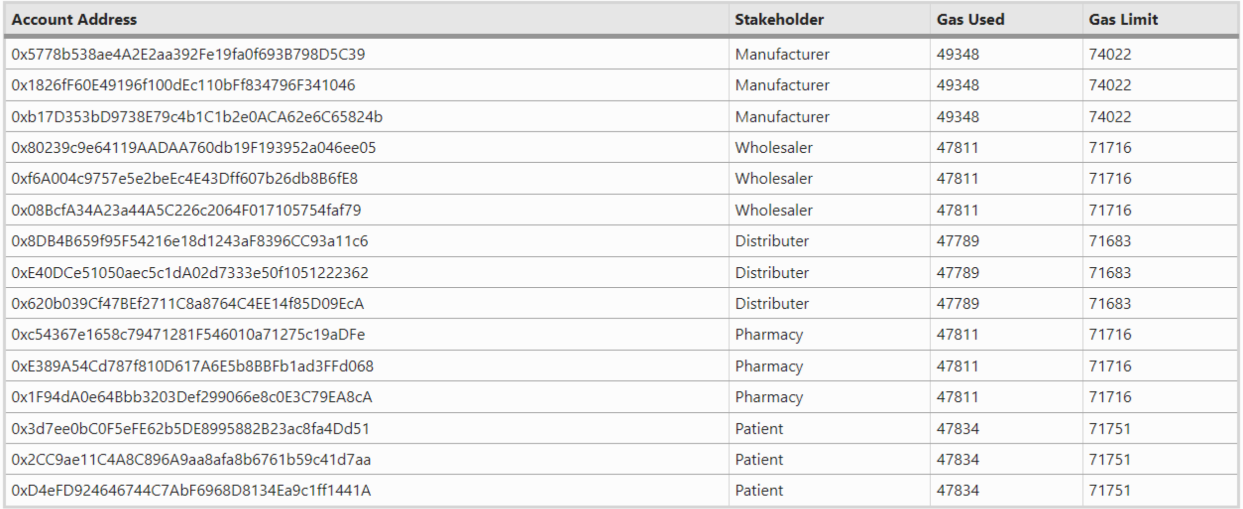
Figure 13 shows the login window. Where a stakeholder fill in the id and password and on clicking submit he has to confirm the request generated through the metamask to log into the system.

Table 6 shows the amount of gas used to log in to the system with a particular login id of a stakeholder.



$$\text{Average gas used to login manufacturer account} = \text{Sum of gas used}/\text{Total Manufacturer added}$$
$$= 14804/3 = 49348$$
$$\text{Average gas used to login Wholesaler account} = \text{Sum of gas used}/\text{Total Wholesaler added}$$
$$= 143433/3 = 47811$$
$$\text{Average gas used to login Distributer account} = \text{Sum of gas used}/\text{Total Distributer added}$$
$$= 143367/3 = 47789$$
$$\text{Average gas used to login Pharmacy account} = \text{Sum of gas used}/\text{Total Pharmacy added}$$
$$= 143433/3 = 47811$$
$$\text{Average gas used to login Patient account} = \text{Sum of gas used}/\text{Total Patient added}$$
$$= 143502/3 = 47834$$
$$\text{Average gas used to login to all the accounts} = (49348 + 47811 + 47789 + 47811 + 47834)\,/5$$
$$= 240593/5 = 48118.6$$

Table 6 Gas used to login in system

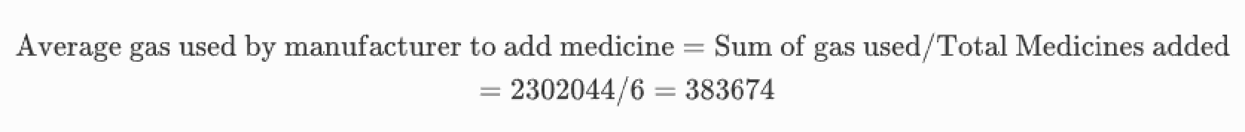


Account Address	Stakeholder	Gas Used	Gas Limit
0x5778b538ae4A2E2aa392Fe19fa0f693B798D5C39	Manufacturer	49348	74022
0x1826fF60E49196f100dEc110bFf834796F341046	Manufacturer	49348	74022
0xb17D353bD9738E79c4b1C1b2e0ACA62e6C65824b	Manufacturer	49348	74022
0x80239c9e64119AADAA760db19F193952a046ee05	Wholesaler	47811	71716
0xf6A004c9757e5e2beEc4E43Dff607b26db8B6fE8	Wholesaler	47811	71716
0x08BcfA34A23a44A5C226c2064F017105754faf79	Wholesaler	47811	71716
0x8DB4B659f95F54216e18d1243aF8396CC93a11c6	Distributer	47789	71683
0xE40DCe51050aec5c1dA02d7333e50f1051222362	Distributer	47789	71683
0x620b039Cf47BEf2711C8a8764C4EE14f85D09EcA	Distributer	47789	71683
0xc54367e1658c79471281F546010a71275c19aDFe	Pharmacy	47811	71716
0xE389A54Cd787f810D617A6E5b8BBFb1ad3FFd068	Pharmacy	47811	71716
0x1F94dA0e64Bbb3203Def299066e8c0E3C79EA8cA	Pharmacy	47811	71716
0x3d7ee0bC0F5eFE62b5DE8995882B23ac8fa4Dd51	Patient	47834	71751
0x2CC9ae11C4A8C896A9aa8afa8b6761b59c41d7aa	Patient	47834	71751
0xD4eFD924646744C7AbF6968D8134Ea9c1ff1441A	Patient	47834	71751

Figure 14 shows that the manufacturer added the medicine details and the gas amount is spent through metamask after a click on the confirmation button and then medicine is added to the blockchain.

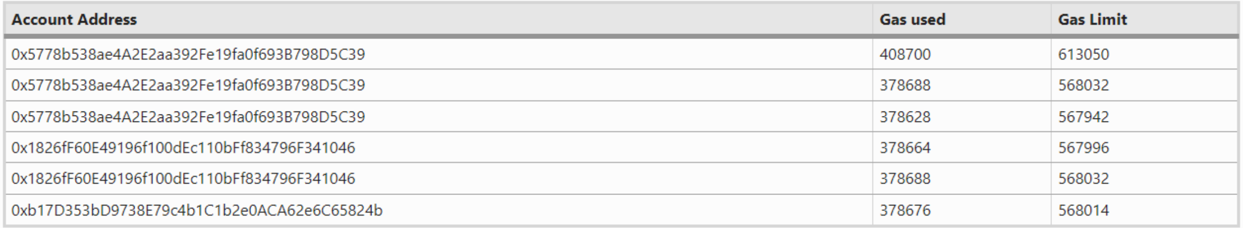
Table 7 shows the amount of gas used to add a medicine to the system. This functionality is allowed only to the manufacturer.

Average gas used by manufacturer to add medicine=Sum of gas used/Total Medicines added=2 302044/6=383674



$$\text{Average gas used by manufacturer to add medicine} = \text{Sum of gas used}/\text{Total Medicines added} = 2302044/6 = 383674$$

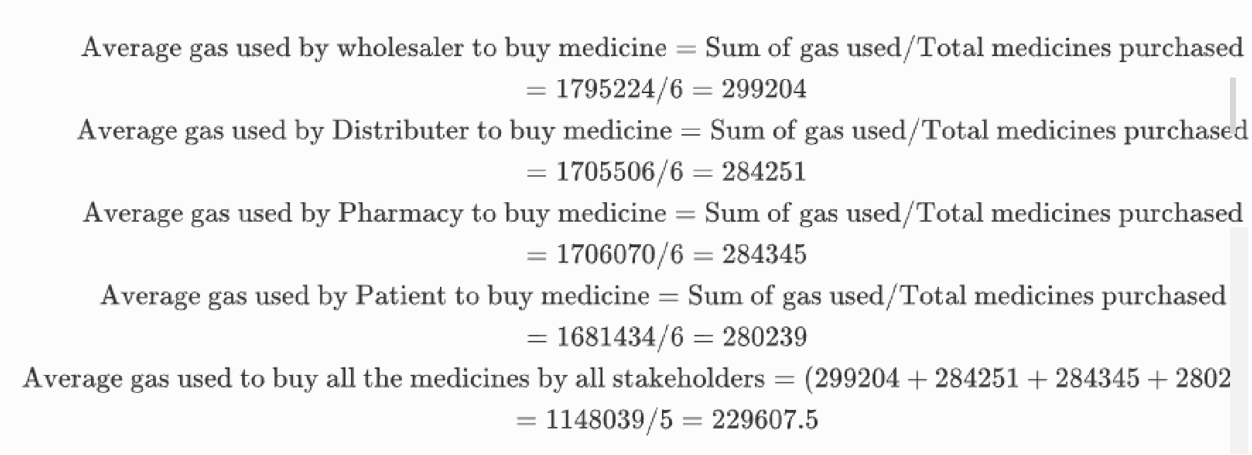
Table 7 Gas used to add the medicine



Account Address	Gas used	Gas Limit
0x5778b538ae4A2E2aa392Fe19fa0f693B798D5C39	408700	613050
0x5778b538ae4A2E2aa392Fe19fa0f693B798D5C39	378688	568032
0x5778b538ae4A2E2aa392Fe19fa0f693B798D5C39	378628	567942
0x1826fF60E49196f100dEc110bFf834796F341046	378664	567996
0x1826fF60E49196f100dEc110bFf834796F341046	378688	568032
0xb17D353bD9738E79c4b1C1b2e0ACA62e6C65824b	378676	568014

Figure 15 shows the amount to be transferred for a transaction made by a stakeholder to buy medicine on confirming the stated amount is transferred to the stakeholder from which the medicine is purchased.

Table 8 shows the amount of gas used to buy the medicine. This functionality is available for all stakeholders other than the manufacturer.

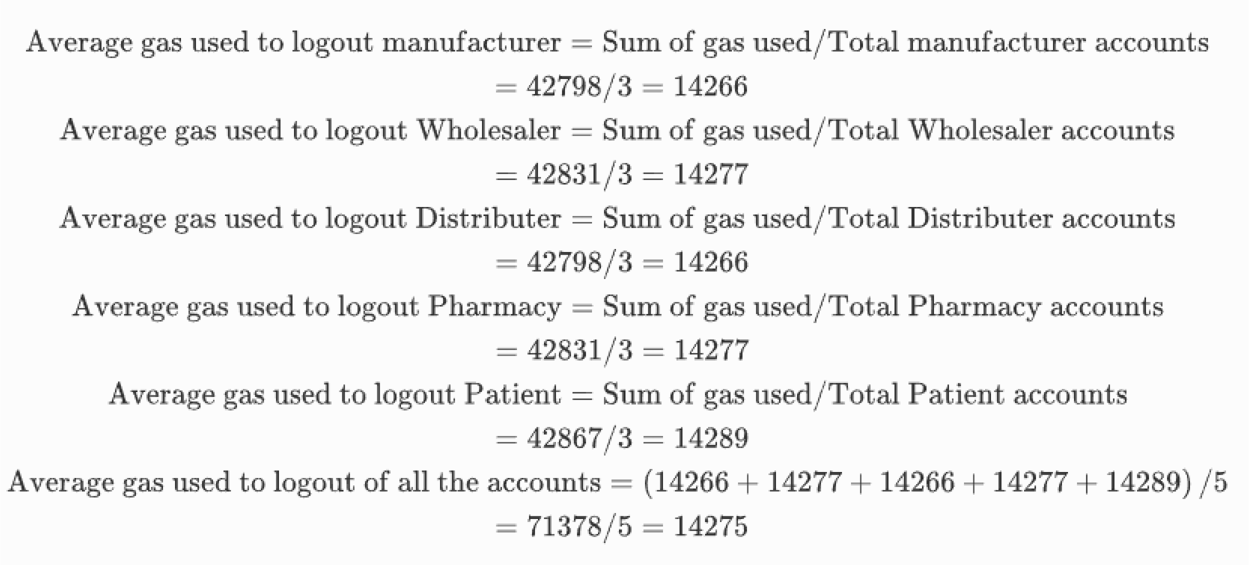


$$
\begin{aligned}
\text{Average gas used by wholesaler to buy medicine} &= \text{Sum of gas used/Total medicines purchased} \\
&= 1795224/6 = 299204 \\
\text{Average gas used by Distributer to buy medicine} &= \text{Sum of gas used/Total medicines purchased} \\
&= 1705506/6 = 284251 \\
\text{Average gas used by Pharmacy to buy medicine} &= \text{Sum of gas used/Total medicines purchased} \\
&= 1706070/6 = 284345 \\
\text{Average gas used by Patient to buy medicine} &= \text{Sum of gas used/Total medicines purchased} \\
&= 1681434/6 = 280239 \\
\text{Average gas used to buy all the medicines by all stakeholders} &= (299204 + 284251 + 284345 + 2802 \\
&= 1148039/5 = 229607.5
\end{aligned}
$$

Table 8 Gas Used to buy the medicine

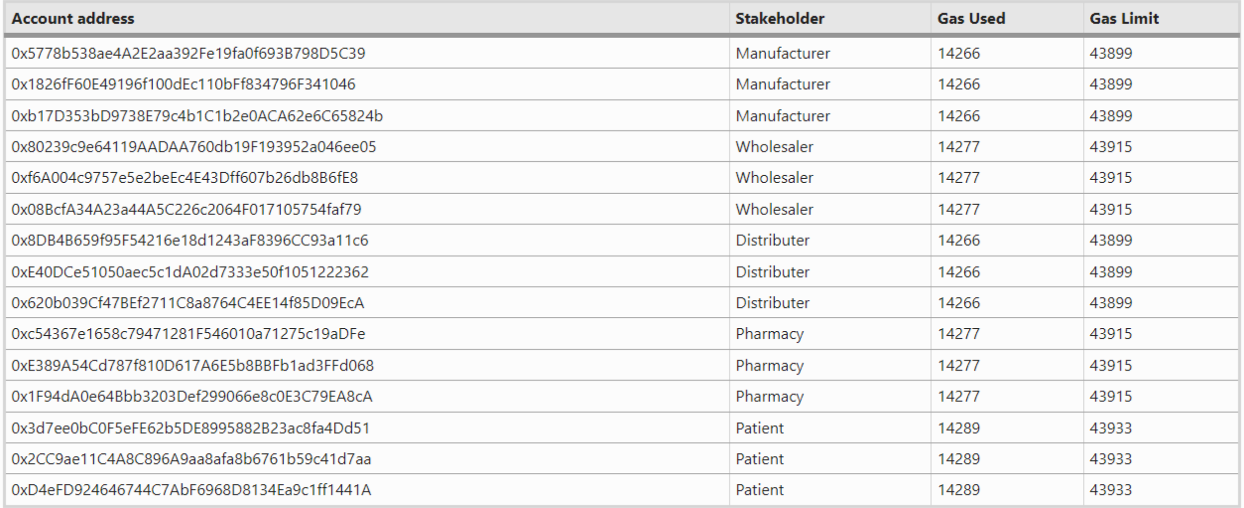
In Fig. 16 the option for entering a medicine id or uploading the QR code of medicine to trace its ownership and price history. In Fig. 17 the full trace of medicine is shown. History shows that the price of medicine is increased whenever its ownership is changed as there is a definite margin of profit that is to be earned by the stakeholder and also, they cannot sell it for a higher amount to earn more profit.

Table 9 shows the amount of gas used to log out from the system.



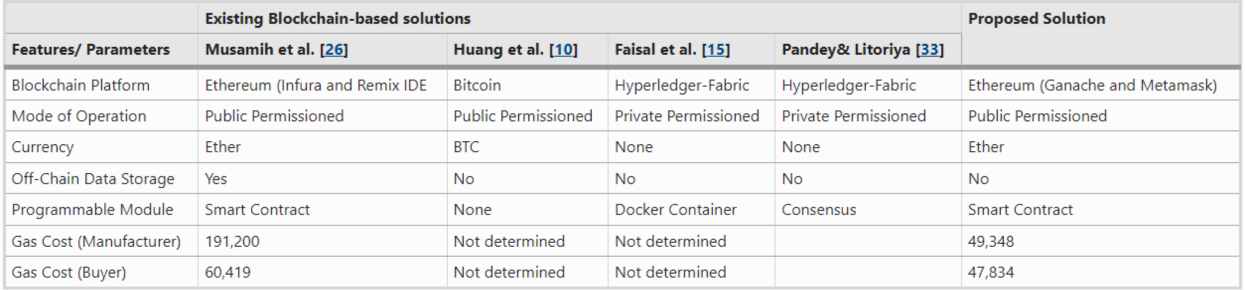
$$
\begin{aligned}
\text{Average gas used to logout manufacturer} &= \text{Sum of gas used/Total manufacturer accounts} \\
&= 42798/3 = 14266 \\
\text{Average gas used to logout Wholesaler} &= \text{Sum of gas used/Total Wholesaler accounts} \\
&= 42831/3 = 14277 \\
\text{Average gas used to logout Distributer} &= \text{Sum of gas used/Total Distributer accounts} \\
&= 42798/3 = 14266 \\
\text{Average gas used to logout Pharmacy} &= \text{Sum of gas used/Total Pharmacy accounts} \\
&= 42831/3 = 14277 \\
\text{Average gas used to logout Patient} &= \text{Sum of gas used/Total Patient accounts} \\
&= 42867/3 = 14289 \\
\text{Average gas used to logout of all the accounts} &= (14266 + 14277 + 14266 + 14277 + 14289)\,/5 \\
&= 71378/5 = 14275
\end{aligned}
$$

Table 9 Logout



Account address	Stakeholder	Gas Used	Gas Limit
0x5778b538ae4A2E2aa392Fe19fa0f693B798D5C39	Manufacturer	14266	43899
0x1826fF60E49196f100dEc110bFf834796F341046	Manufacturer	14266	43899
0xb17D353bD9738E79c4b1C1b2e0ACA62e6C65824b	Manufacturer	14266	43899
0x80239c9e64119AADAA760db19F193952a046ee05	Wholesaler	14277	43915
0xf6A004c9757e5e2beEc4E43Dff607b26db8B6fE8	Wholesaler	14277	43915
0x08BcfA34A23a44A5C226c2064F017105754faf79	Wholesaler	14277	43915
0x8DB4B659f95F54216e18d1243aF8396CC93a11c6	Distributer	14266	43899
0xE40DCe51050aec5c1dA02d7333e50f1051222362	Distributer	14266	43899
0x620b039Cf47BEf2711C8a8764C4EE14f85D09EcA	Distributer	14266	43899
0xc54367e1658c79471281F546010a71275c19aDFe	Pharmacy	14277	43915
0xE389A54Cd787f810D617A6E5b8BBFb1ad3FFd068	Pharmacy	14277	43915
0x1F94dA0e64Bbb3203Def299066e8c0E3C79EA8cA	Pharmacy	14277	43915
0x3d7ee0bC0F5eFE62b5DE8995882B23ac8fa4Dd51	Patient	14289	43933
0x2CC9ae11C4A8C896A9aa8afa8b6761b59c41d7aa	Patient	14289	43933
0xD4eFD924646744C7AbF6968D8134Ea9c1ff1441A	Patient	14289	43933

Table 10 Comparative analysis of different features of the proposed blockchain-based solution with the existing approaches



	Existing Blockchain-based solutions				Proposed Solution
Features/ Parameters	**Musamih et al. [26]**	**Huang et al. [10]**	**Faisal et al. [15]**	**Pandey& Litoriya [33]**	
Blockchain Platform	Ethereum (Infura and Remix IDE	Bitcoin	Hyperledger-Fabric	Hyperledger-Fabric	Ethereum (Ganache and Metamask)
Mode of Operation	Public Permissioned	Public Permissioned	Private Permissioned	Private Permissioned	Public Permissioned
Currency	Ether	BTC	None	None	Ether
Off-Chain Data Storage	Yes	No	No	No	No
Programmable Module	Smart Contract	None	Docker Container	Consensus	Smart Contract
Gas Cost (Manufacturer)	191,200	Not determined	Not determined		49,348
Gas Cost (Buyer)	60,419	Not determined	Not determined		47,834