

INTERACTION WITH FRONTEND

Interact With The Frontend For All Functionalities

Team Id	NM2023TMID04410
Project Name	Drug Traceability

Interacting with the frontend in a smart contracts drug traceability system:

In a smart contract-based drug traceability system on the blockchain, interacting with the frontend involves a different set of functionalities and user actions compared to traditional systems. Here's how users interact with the frontend in such a system:

1.User Authentication:

- Users log in to the frontend using their credentials, which may be linked to their blockchain wallet addresses.
- Blockchain wallets and private keys are used for authentication.

2.Data Entry:

- Users can initiate the creation of new drug traceability records on the blockchain.
- They input data about drug products, such as serial numbers, batch information, and product details.

3.Data Retrieval and Verification:

- Users query the blockchain to retrieve information about drug products.
- They can verify the authenticity of products by checking the blockchain records.

4.Smart Contract Interactions:

- Users may interact directly with smart contracts through the frontend.
- This includes initiating actions like product transfers, ownership changes, or batch updates.

5.Alerts and Notification:

- The frontend can display alerts and notifications based on predefined smart contract conditions.
- Users receive alerts when specific events occur on the blockchain.

6.Inventory Management:

- Users can manage their inventory of drug products on the blockchain.
- They can add new products, update quantities, and track product movements.

7.Blockchain Explorer:

- The frontend may provide access to a blockchain explorer, allowing users to explore the blockchain's entire transaction history and traceability records.

8.Reporting and Analytics:

- Users can generate reports and analytics based on data recorded in smart contracts.
- They can visualize data on the frontend through charts and graphs.

9.User Training and Support:

- Training materials and documentation are available to guide users on how to interact with smart contracts on the blockchain.
- Support channels are accessible for assistance.

10.Security Measures:

- Users must manage their blockchain wallet security and private keys, which is critical for accessing and interacting with smart contracts.
- Encryption and secure key management are emphasized.

11.Mobile Optimization:

- The frontend may be optimized for mobile access, allowing users to perform blockchain interactions from their mobile devices.

12.User Acceptance Testing (UAT):

- Users participate in UAT to test the frontend's functionality and provide feedback.
- Testing includes smart contract interactions and blockchain-related actions.

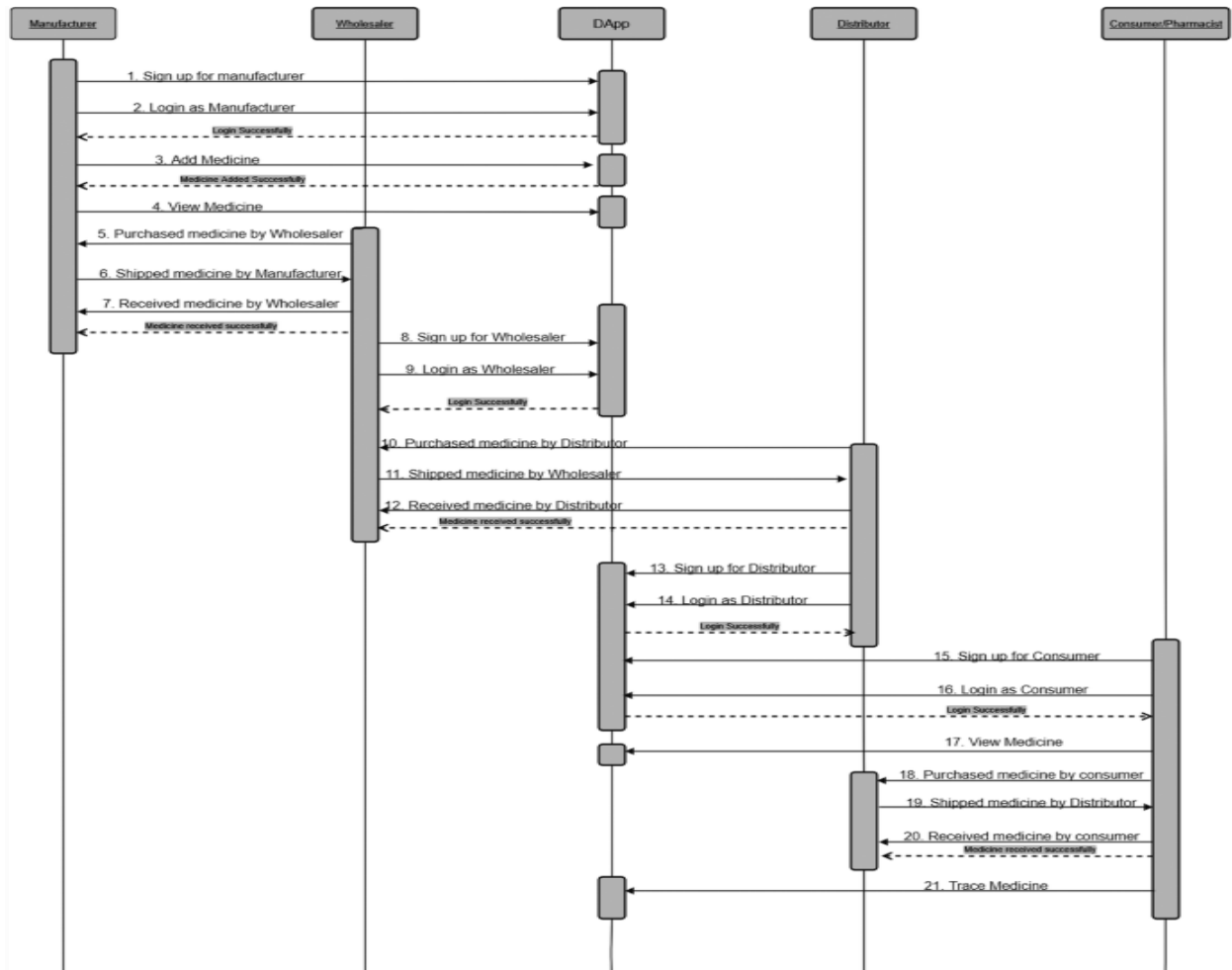
13.Deployment and Maintenance:

- After UAT, the frontend is deployed to production, ensuring that users can perform secure interactions with the blockchain.
- Ongoing maintenance and updates are essential for the blockchain system's integrity.

Implementation:

- 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.



Sequence diagram

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 above in 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 expire 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 medicinePrice 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:

addManufacturer():

In algorithm 1 the **addManufacturer()** function is used to add the manufacturer's account to the decentralized application with the specific inputs mentioned above in 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.

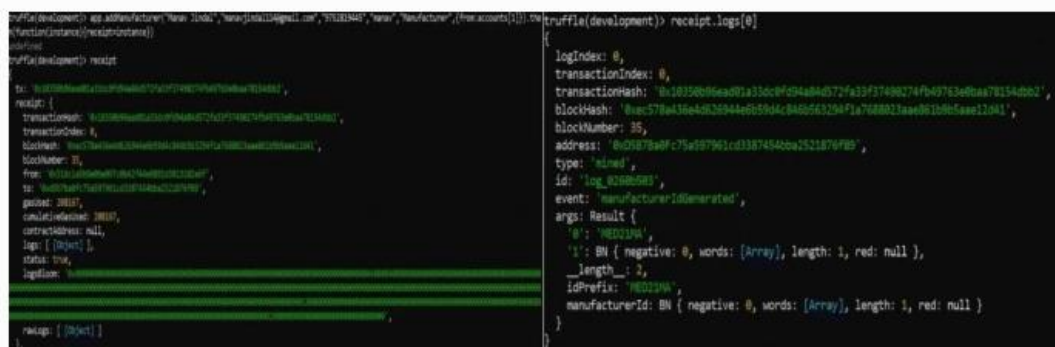
`manufacturerName`: is the name of the manufacturer.

`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.

Fig. 3



```
truffle(development)> addManufacturer('New 1234', 'new@abc.com', '9876543210', 'new', 'Manufacturer', 'Manufacturer1234')
[Function: Promise]
truffle(development)> receipt
{
  to: '0x0000000000000000000000000000000000000000',
  receipt: {
    transactionHash: '0x10350005ead01a33dc0f034a04d372fa33f374002747b497e3e0ba478154db02',
    transactionIndex: 0,
    blockHash: '0xacc578a13e40c26944e6c58d4c0456563294f1a7688021aae061b0e5aae12041',
    blockNumber: 35,
    address: '0x01870ba0c75a97961cd13874540ba252187670b9',
    type: ' mined',
    id: 'log_83080583',
    event: 'manufacturerIdGenerated',
    args: Result {
      0: 'Manufacturer1234',
      1: BN { negative: 0, words: [Array], length: 1, red: null },
      __length__: 2,
      idPrefix: 'Manufacturer1234',
      manufacturerId: BN { negative: 0, words: [Array], length: 1, red: null }
    }
  }
}
```

a

b

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

Similarly **addWholesaler()**, **addDistributor()**, **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

Figure 4 consists of two side-by-side terminal screenshots. The left screenshot, labeled 'a', shows the execution of the `login` function in a Truffle environment. It displays a transaction receipt with details such as `transactionHash`, `blockHash`, `blockNumber`, `address`, `type`, `id`, `event`, and `args`. The right screenshot, labeled 'b', shows the corresponding log generation, displaying the same transaction details and the event `UserLoginStatus` with its arguments.

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

Figure 5 consists of two side-by-side terminal screenshots. The left screenshot, labeled 'a', shows the execution of the `addMedicine` function in a Truffle environment. It displays a transaction receipt with details such as `transactionHash`, `blockHash`, `blockNumber`, `address`, `type`, `id`, `event`, and `args`. The right screenshot, labeled 'b', shows the corresponding log generation, displaying the same transaction details and the event `MedicineAdded` with its arguments.

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


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

[illegible]

(b)

Fig. 10

Fig. 10

ACCOUNTS					BLOCKS					TRANSACTIONS					CONTRACTS					EVENTS					LOGS				
CURRENT BLOCK					GAS PRICE					GAS LIMIT					NETWORK ID					RPC URL					MINING STATUS				
0					2000000000					8721975					5777					HTTP://127.0.0.1:7545					AUTOMINING				
MEMORIC					HD PATH																								
maximin tag list swift follow favorite boil whale garbage rib copper family					m/44'/60'/0'/0/account_index																								
ADDRESS					BALANCE					TX COUNT					INDEX														
0x941b2b854A8CB81c838c4680ac857bc87dF635E0					100.00 ETH					0					0														
ADDRESS					BALANCE					TX COUNT					INDEX														
0xEF54A85eae02c8231aFFCd9edAa4043ba062B20a					100.00 ETH					0					1														
ADDRESS					BALANCE					TX COUNT					INDEX														
0x861EB650C830ca6C7CE5b20A50946420FDf8869					100.00 ETH					0					2														
ADDRESS					BALANCE					TX COUNT					INDEX														
0xFDE3686D306D768babdbae539F266A1CAA5f135					100.00 ETH					0					3														
ADDRESS					BALANCE					TX COUNT					INDEX														
0x1B5B0c54bC283123D6f5C773388A638dd5a4c742					100.00 ETH					0					4														
ADDRESS					BALANCE					TX COUNT					INDEX														
0xD01Bdc060b1460365EF1DfB437de36718c58De10					100.00 ETH					0					5														
ADDRESS					BALANCE					TX COUNT					INDEX														
0xFA5C828a7E4d2b0e979cFa3c2DD5640d1382CDC7					100.00 ETH					0					6														
ADDRESS					BALANCE					TX COUNT					INDEX														

Ganache account

Fig. 11

```

Starting migrations...
> Network name: 'development'
> Network id: 5777
> Block gas limit: 8721975 (0x8451b7)

1_initial_migration.js
-----
Replating "Migrations"
> Transaction hash: 0x574ac5b0a2335baae71905f8e6d79355dc10f4b1e4d0f0ba22af1270464f05
> Block: 0
> Contract address: 0x0000420af0AD0Ff27C0720b23106500f700A360f
> Block number: 1
> Block timestamp: 1646320340
> Account: 0x2A014E25730C32205040F00A303a962031F7F00c
> Balance: 99.9999154
> Gas used: 103542 (0x20dc7)
> Gas price: 20 gwei
> Value sent: 0 ETH
> Total cost: 0.00351000 ETH

> Saving migration to chain.
> Saving artifacts
> Total cost: 0.00351000 ETH

2_initial_migration.js
-----
Replating "Migrations"
> Transaction hash: 0xadd0570fbb302b1db143000c0b04d2ab04b627f53c1f15377a009cfa2081
> Block: 0
> Contract address: 0x5b00a7501a10027530357a03c0f00f31474c00d
> Block number: 2
> Block timestamp: 1646320345
> Account: 0x2A014E25730C32205040F00A303a962031F7F00c
> Balance: 99.99991000
> Gas used: 103500 (0x20d00)
> Gas price: 20 gwei
> Value sent: 0 ETH
> Total cost: 0.00357032 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.

DETAILS DATA HISTORY

Estimated gas fee 0.0062418 ETH
Max Fee: 0.0062418 ETH

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

Reject Confirm

Account creation

Fig. 13

Login

Username
MED21MAR1

Password

Submit

Don't have an Account?
Create New Account

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

DETAILS DATA HISTORY

Estimated gas fee 0.00148044 ETH
Max Fee: 0.00148044 ETH

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

Reject Confirm

Stake holder login

Fig. 14



Manufacturer add medicine

Fig. 15

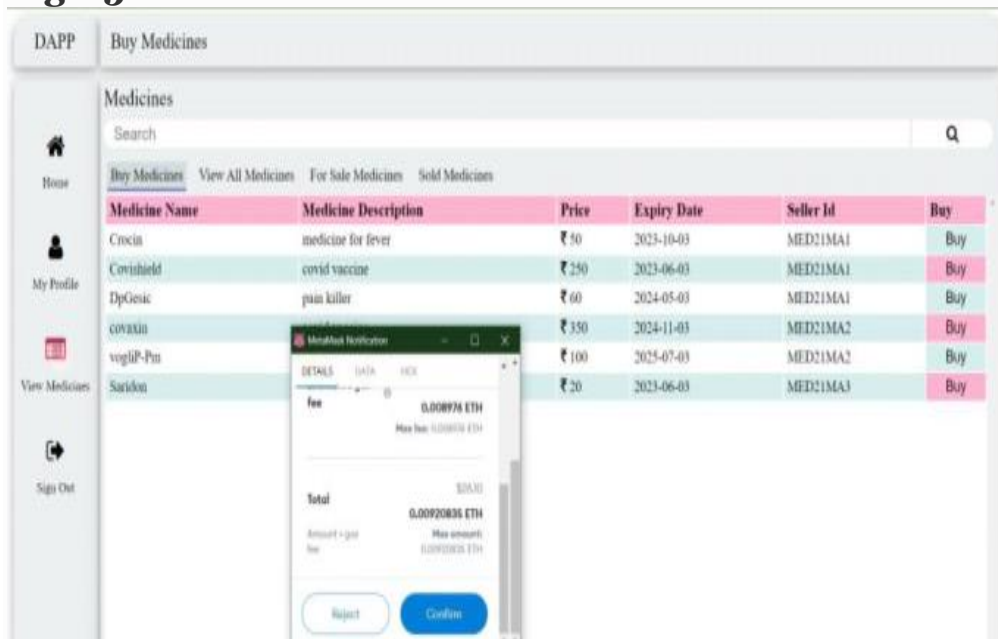


Fig. 16



Trace medicine window

Fig. 17

DAPP	Trace Medicine				
	Traced Path				
	Medicine Name: vogliP-Pm		Medicine Id: MED21MA2VOG202203035		Expiry Date: 2025-07-03
	Manufacturer	Wholesaler	Distributor	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.