**Drug Logistics Supply Chain System**

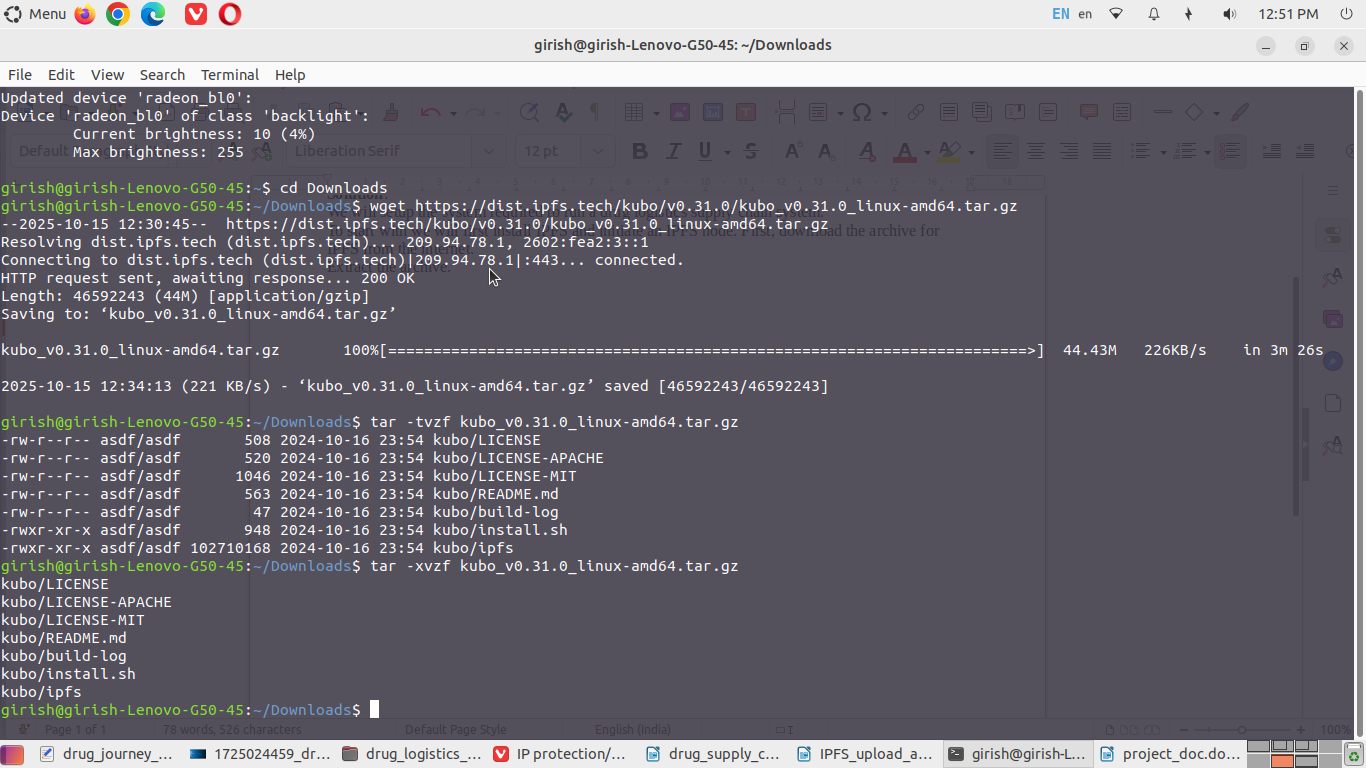
**Problem statement**:

The pharmaceutical supply chain is plagued by drug counterfeiting, lack of transparency, and operational inefficiencies. These issues compromise drug safety and supply chain reliability, impacting patient care and increasing healthcare costs.

**Solution**:

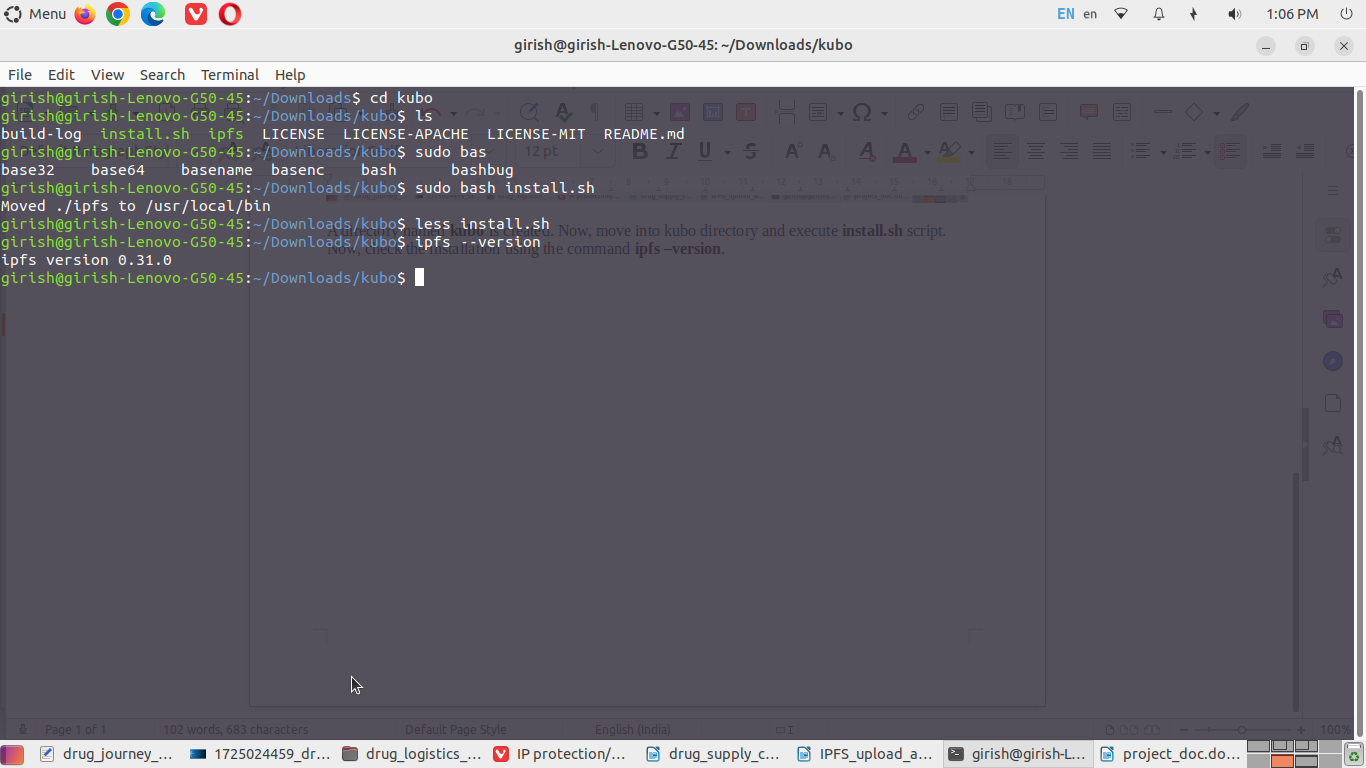
We will setup the system required to run a drug logistics supply chain system.

To start with we will first install IPFS and initiate an IPFS node. First, **download** the archive for IPFS from the internet. **Extract** the archive.

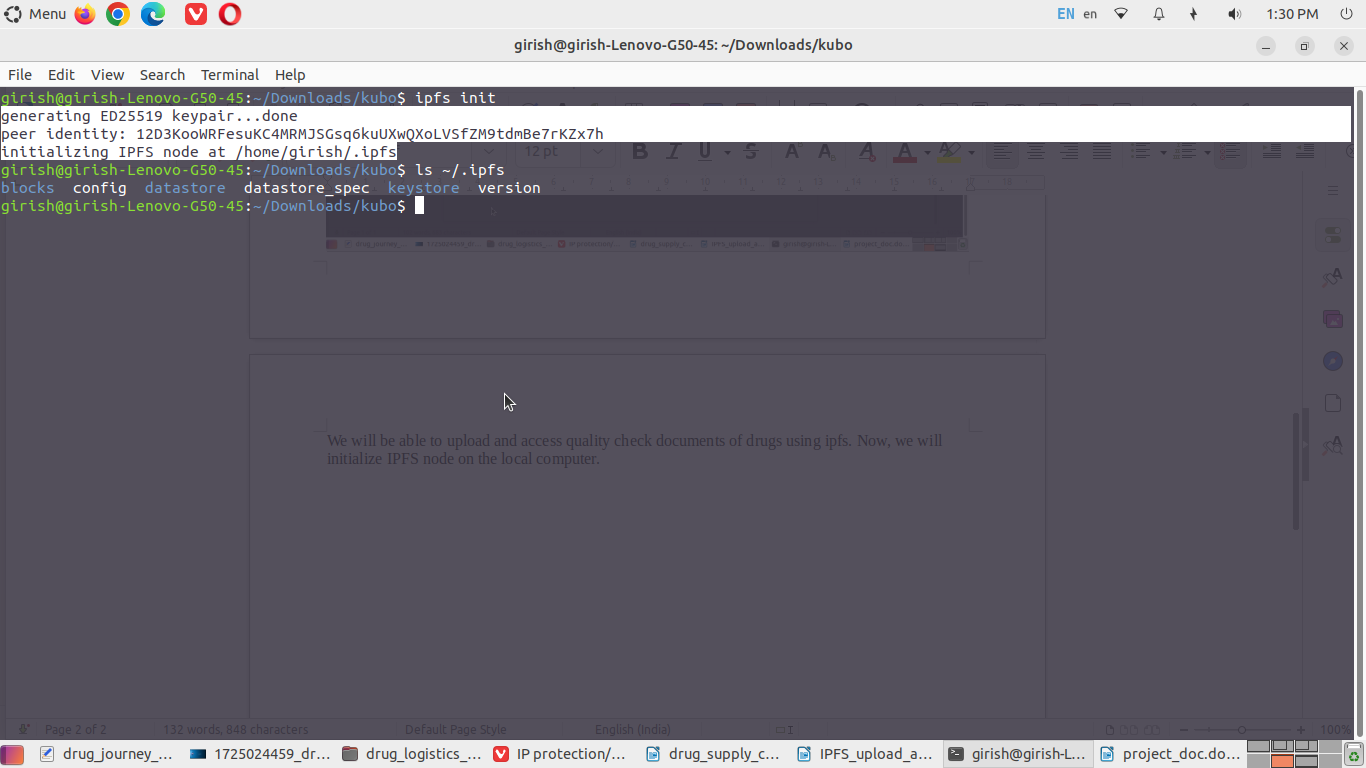


A directory named **kubo** is created. Now, move into kubo directory and execute **install.sh** script.

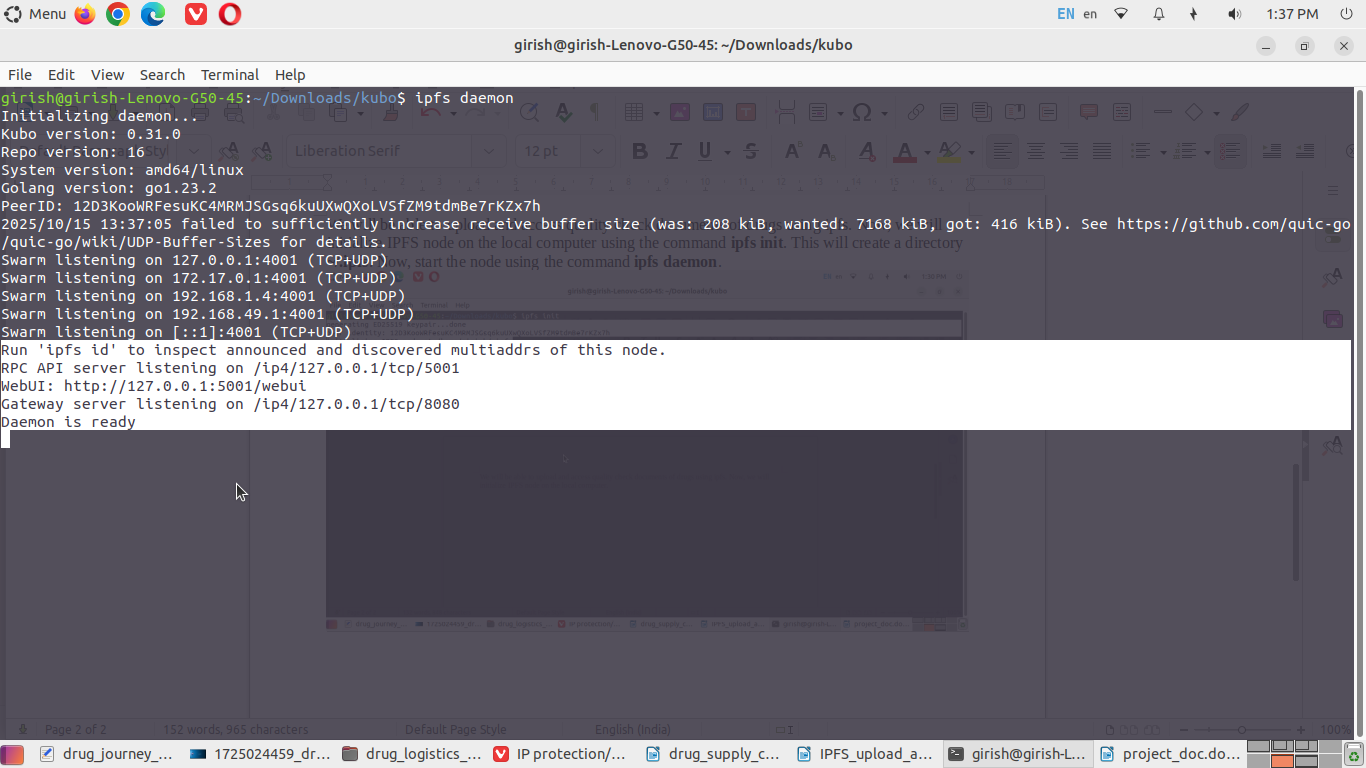
Now, check the installation using the command **ipfs –version**. This way ipfs is installed.

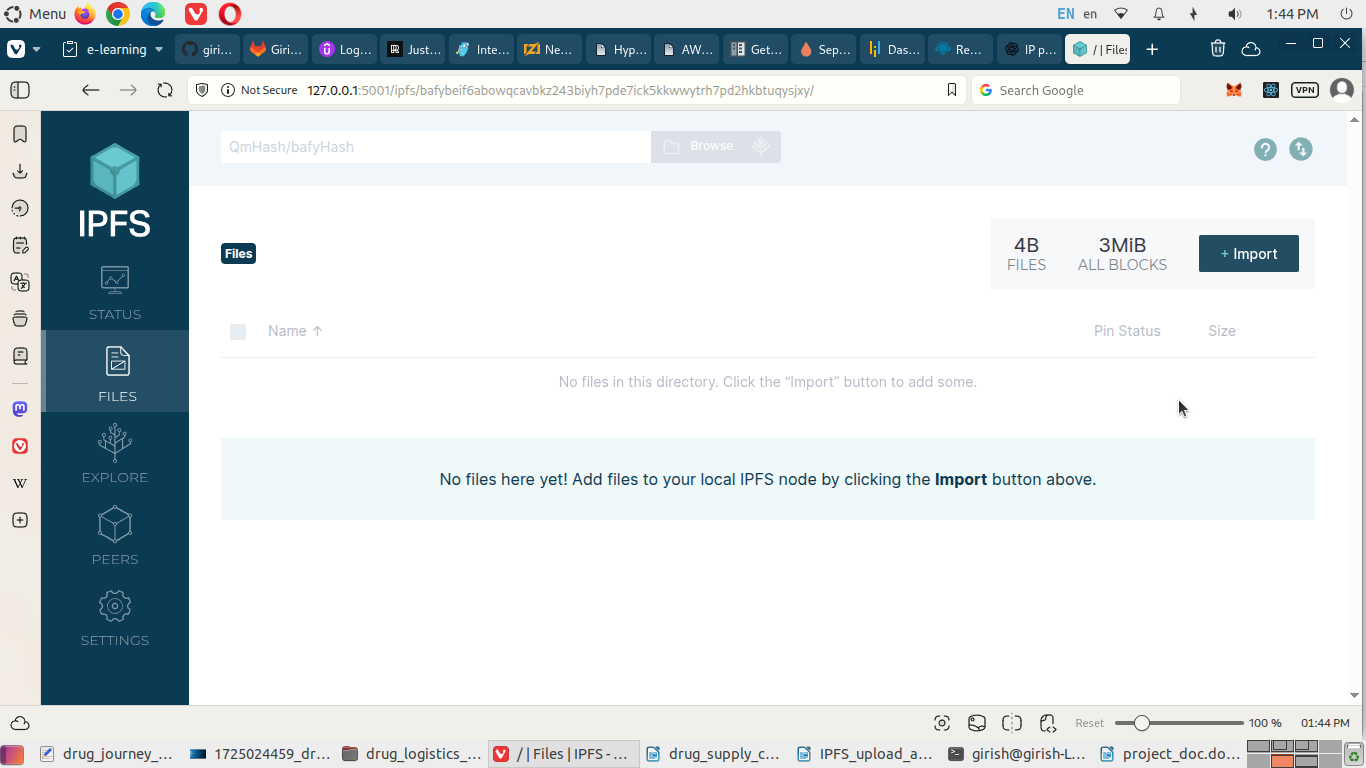


We will be able to upload and access quality check documents of drugs using ipfs. Now, we will initialize IPFS node on the local computer using the command **ipfs init**. This will create a directory **~/.ipfs**. Now, start the node using the command **ipfs daemon**.

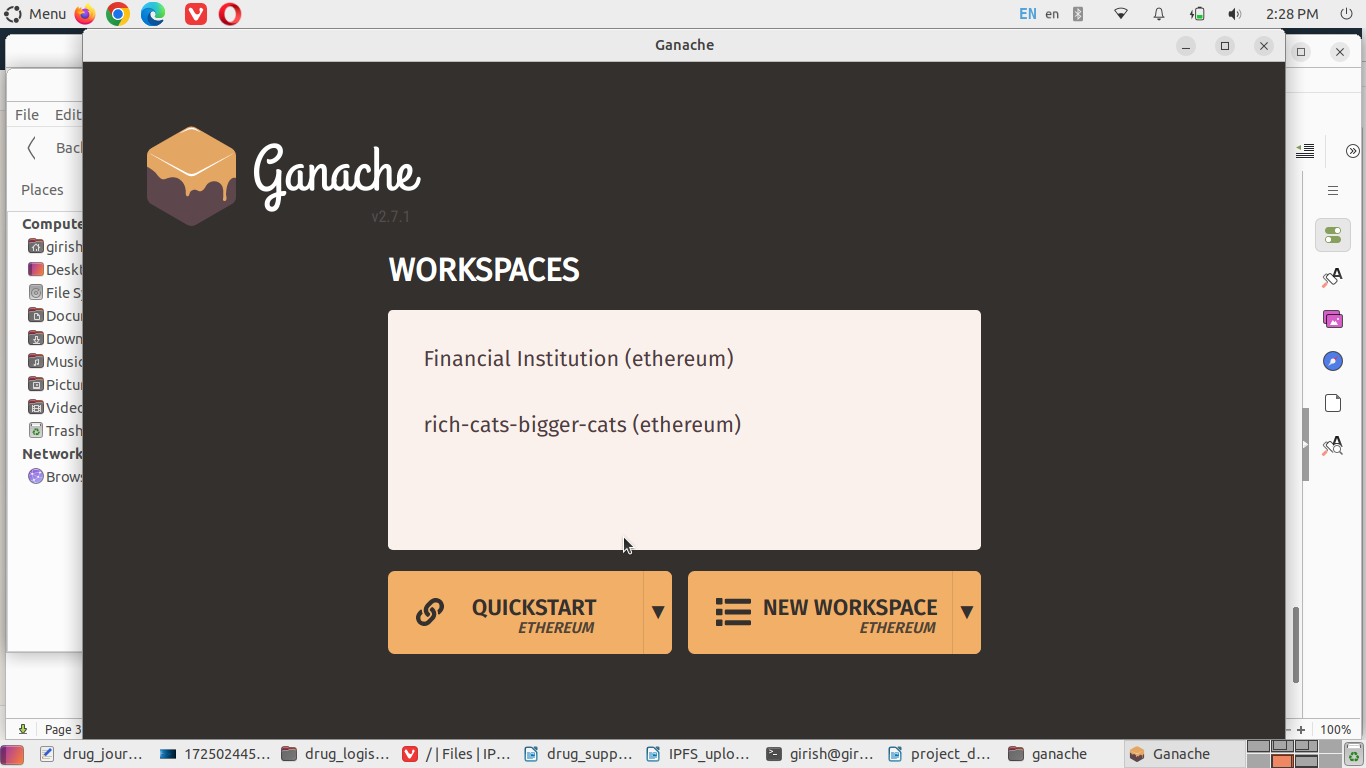


When you run ipfs daemon command, you will see the below output.

You will get a WebUI at [**http://127.0.0.1:5001/webui**](http://127.0.0.1:5001/webui) as per the output above. Below is your WebUI for IPFS. Click on **Files** and you will see a screen with **import** button. So, IPFS is now working in our system. We will be using IPFS for uploading the drug quality report documents.



We shall now create a **Ganache Blockchain**. I have downloaded Ganache application from internet. The file name is “ganache-2.7.1-linux-x86\_64.AppImage”. **Double-click** on Ganache application.



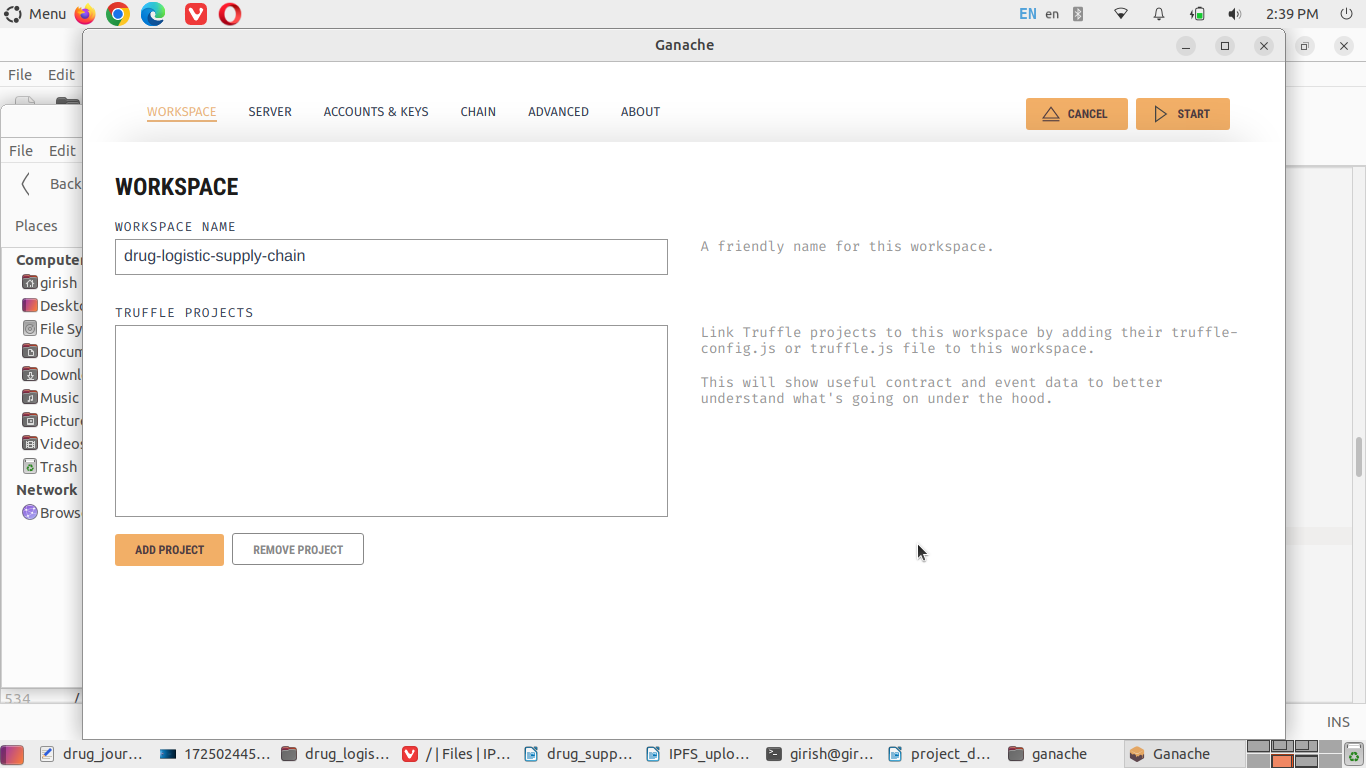
Select NEW WORKSPACE ETHEREUM to launch a new blockchain. Name the workspace as “drug-logistic-supply-chain”.

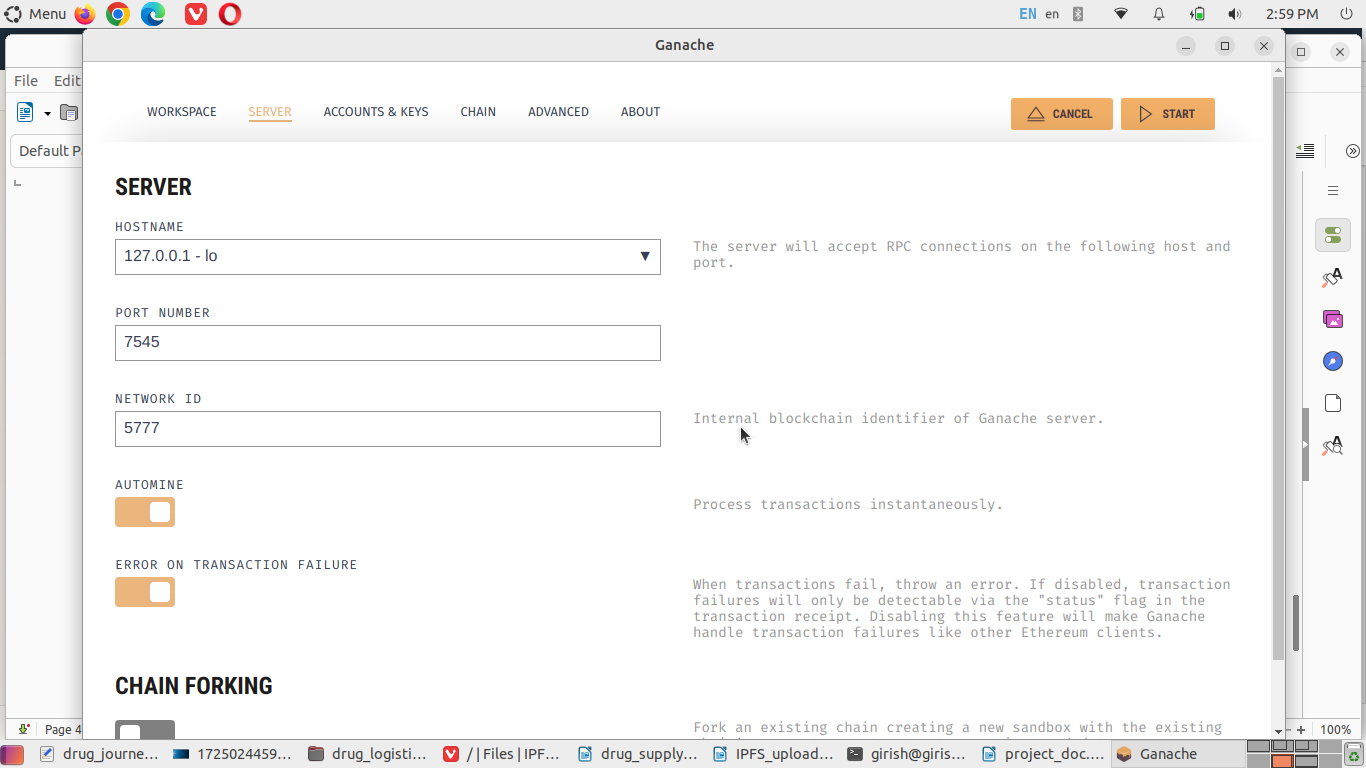
Click SERVER tab. Note the host name and port number.

Click on ACCOUNTS & KEYS tab. Specify default balance as 400 and total accounts as 10.

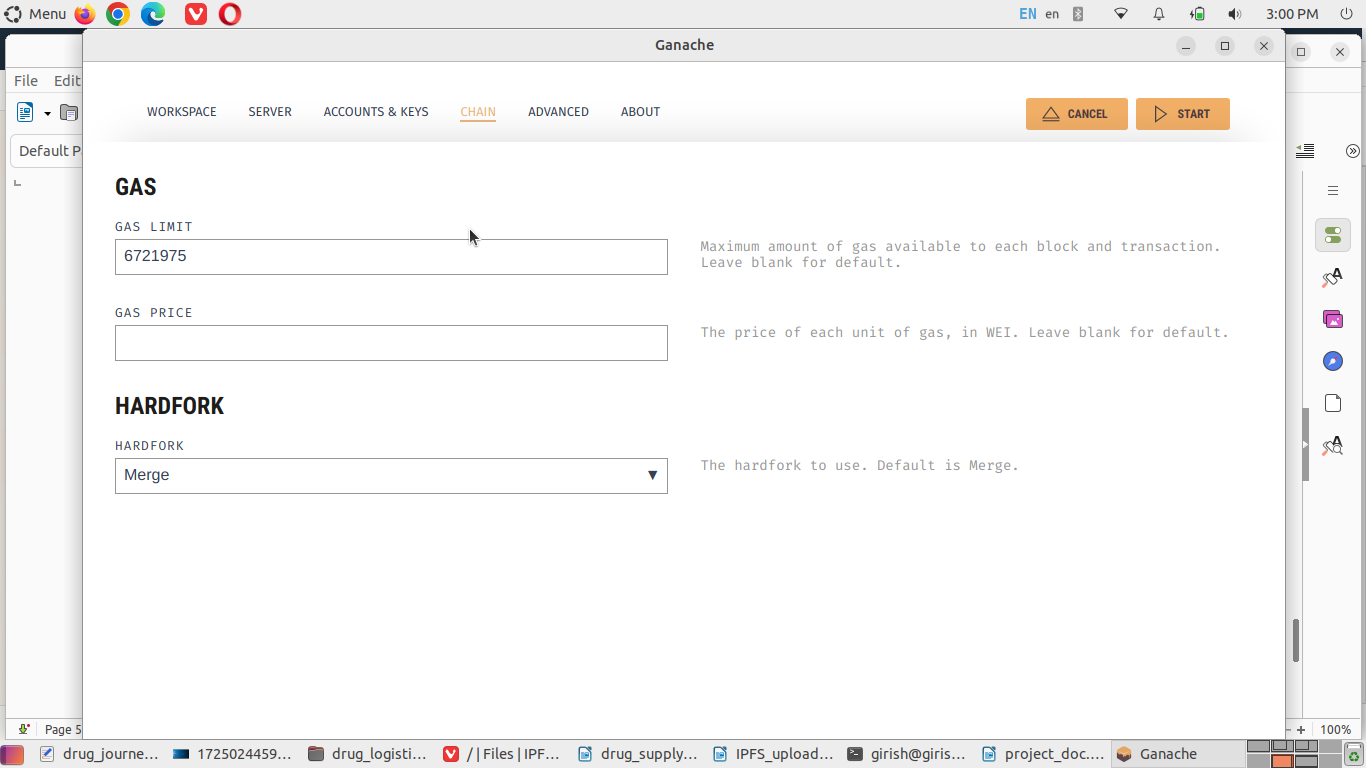
Click on CHAIN tab. Keep GAS PRICE as blank. Select HARDFORK as Merge (default).

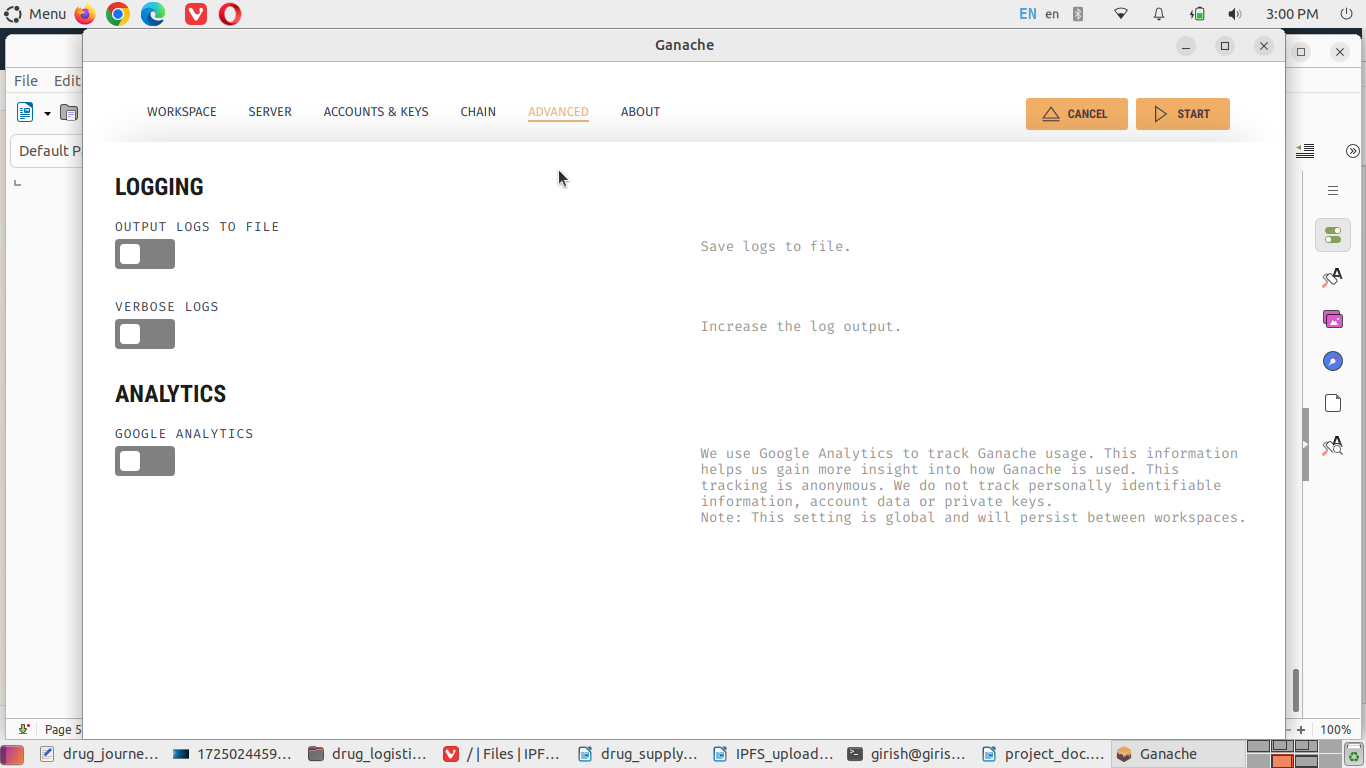
Click on ADVANCE tab. Keep the settings to default.











Click on START button on the top-right of the application window.

The blockchain with 10 accounts with each having 400 ETH is created.



In our smart contract, we require minimum 7 roles. So, we will assign one Ganache account to each role. These Roles are as follows.

Unknown, Manufacturer, Distributor, Pharmacy, Regulator, QualityInspector, Consumer.

We use Unknown as zero value for user (participant) role.

For this project, we will have 5 users/ participant. We will add them once the Solidity smart contract is deployed. But, before that, we need to connect Ganache blockchain with Remix IDE.

We will have to create the smart contract for the drug logistics supply chain. Below is the code for the same.

// SPDX-License-Identifier: MIT

pragma solidity ^0.8.17;

contract DrugSupplyChainWithQuality {

// Roles

enum Role { Unknown, Manufacturer, Distributor, Pharmacy, Regulator, QualityInspector, Consumer }

// Drug lifecycle states

enum DrugState {

Manufactured,

ShippedToDistributor,

ReceivedByDistributor,

ShippedToPharmacy,

ReceivedByPharmacy,

SoldToCustomer,

Recalled

}

// Participant

struct Participant {

address addr;

string name;

string location;

Role role;

bool active; // true for active, false for inactive

uint256 registeredAt;

}

// Transfer record

struct TransferRecord {

address from;

address to;

uint256 timestamp;

string notes; // mention shipment ID, condition, transport info etc.

}

// Quality Check record

struct QualityCheck {

address inspector; // inspector account address

uint256 timestamp; // time of quality check (block time)

string location; // whether lab or warehouse

int16 temperature; // temperature or drugs in °C, can be negative

uint8 humidity; // 0% to 100%

bool passed; // QC true for pass, false for fail

string remarks; // test remarks

string documentHash; // IPFS CID

string ipfsGatewayURL; // optional gateway URL

// url example: https://ipfs.io/ipfs/

}

// Drug batch

struct Drug {

uint256 drugId; // unique ID

string name; // name of the drug

string batchNumber; // batch or lot number (unique)

string documentHash; // optional manufacturer doc CID in IPFS

address manufacturer; // manufacturer address

uint256 manufactureDate; // timestamp of manufacture

uint256 expiryDate; // timestamp of expiry of the medicine

DrugState state; // current status of the drug

address currentOwner; // current holder, who is in possesstion

address[] ownershipHistory; // maintain owners in chronological order

TransferRecord[] transfers; // details of drug transfer

QualityCheck[] qualityHistory; // QC records (with IPFS links)

}

// Storage

address public contractOwner; // Regulator is the administrator

uint256 private \_drugCounter; // incremental counter for drugId

mapping(address => Participant) public participants; // participant registry

mapping(uint256 => Drug) private drugs; // drugId => Drug

mapping(string => uint256) public batchToDrugId; // batchNumber => drugId (0 = unset)

mapping(address => bool) public isRegistered; // quick lookup

// Events

event ParticipantRegistered(address indexed addr, Role role, string name);

event ParticipantActivated(address indexed addr);

event DrugManufactured(uint256 indexed drugId, string batchNumber, address indexed manufacturer);

event DrugStateChanged(uint256 indexed drugId, DrugState newState);

event DrugTransferred(uint256 indexed drugId, address indexed from, address indexed to, string notes);

event DrugRecalled(uint256 indexed drugId, string notes);

event QualityCheckAdded(uint256 indexed drugId, address indexed inspector, string documentHash, string ipfsGatewayURL);

// Modifiers

modifier onlyContractOwner() {

require(msg.sender == contractOwner, "Only contract owner");

\_;

}

modifier onlyRegisteredActive() {

require(isRegistered[msg.sender], "Not registered");

require(participants[msg.sender].active, "Participant not active");

\_;

}

modifier onlyRole(Role required) {

require(isRegistered[msg.sender], "Not registered");

require(participants[msg.sender].role == required, "Incorrect role");

require(participants[msg.sender].active, "Participant not active");

\_;

}

modifier drugExists(uint256 drugId) {

require(drugId > 0 && drugId <= \_drugCounter, "Drug does not exist");

\_;

}

modifier batchUnique(string memory batchNumber) {

require(batchToDrugId[batchNumber] == 0, "Batch number already exists");

\_;

}

// Constructor

constructor() {

contractOwner = msg.sender;

\_drugCounter = 0;

// Register owner as Regulator by default

participants[contractOwner] = Participant(contractOwner, "Admin", "OnChain", Role.Regulator, true, block.timestamp);

isRegistered[contractOwner] = true;

emit ParticipantRegistered(contractOwner, Role.Regulator, "Admin");

emit ParticipantActivated(contractOwner);

}

// Participant management

/// @notice Register a participant (only contract owner / admin)

/// @param addr address of participant

/// @param name human-readable name

/// @param location physical address / location string

/// @param roleNum numeric role value per Role enum (1..6)

function registerParticipant(address addr, string calldata name, string calldata location, uint8 roleNum)

external onlyContractOwner

{

require(addr != address(0), "Invalid address");

require(!isRegistered[addr], "Already registered");

Role role = Role(roleNum);

require(role != Role.Unknown, "Invalid role");

participants[addr] = Participant({

addr: addr,

name: name,

location: location,

role: role,

active: false, // owner must activate after off-chain KYC. Meaning offline KYC.

registeredAt: block.timestamp

});

isRegistered[addr] = true;

emit ParticipantRegistered(addr, role, name);

}

/// @notice Activate a registered participant after verification (only owner)

function activateParticipant(address addr) external onlyContractOwner {

require(isRegistered[addr], "Not registered");

participants[addr].active = true;

emit ParticipantActivated(addr);

}

/// @notice Deactivate a participant (only owner)

function deactivateParticipant(address addr) external onlyContractOwner {

require(isRegistered[addr], "Not registered");

participants[addr].active = false;

}

/// @notice Update participant metadata (callable by the participant)

function updateParticipantInfo(string calldata name, string calldata location) external {

require(isRegistered[msg.sender], "Not registered");

Participant storage p = participants[msg.sender];

p.name = name;

p.location = location;

}

// Drug lifecycle & logistics

/// @notice Manufacturer creates a new drug batch (registers it on-chain)

/// @param name drug name

/// @param batchNumber unique batch/lot number

/// @param documentHash optional: CID of manufacturer certificate or spec (IPFS CID)

/// @param expiryDate UNIX timestamp of expiry

function manufactureDrug(

string calldata name,

string calldata batchNumber,

string calldata documentHash,

uint256 expiryDate

)

external onlyRole(Role.Manufacturer)

batchUnique(batchNumber)

{

require(expiryDate > block.timestamp, "Expiry must be in future");

\_drugCounter += 1;

uint256 newId = \_drugCounter;

// create dynamic arrays

address[] memory ownersInitial;

ownersInitial[0] = msg.sender;

Drug storage d = drugs[newId];

d.drugId = newId;

d.name = name;

d.batchNumber = batchNumber;

d.documentHash = documentHash;

d.manufacturer = msg.sender;

d.manufactureDate = block.timestamp;

d.expiryDate = expiryDate;

d.state = DrugState.Manufactured;

d.currentOwner = msg.sender;

d.ownershipHistory = ownersInitial;

batchToDrugId[batchNumber] = newId;

emit DrugManufactured(newId, batchNumber, msg.sender);

}

/// @notice Transfer a drug batch from current owner to another participant

/// @param drugId internal drug id

/// @param to address of recipient (must be registered & active)

/// @param notes optional transfer notes

function transferDrug(uint256 drugId, address to, string calldata notes)

external drugExists(drugId)

{

Drug storage d = drugs[drugId];

require(d.currentOwner == msg.sender, "Only current owner can transfer");

require(isRegistered[to] && participants[to].active, "Recipient not active registered participant");

require(to != address(0), "Cannot transfer to zero address");

// record transfer

d.transfers.push(TransferRecord({

from: msg.sender,

to: to,

timestamp: block.timestamp,

notes: notes

}));

// update ownership

d.ownershipHistory.push(to);

address prev = d.currentOwner;

d.currentOwner = to;

emit DrugTransferred(drugId, prev, to, notes);

}

/// @notice Update the state of a drug (for supply-chain stage tracking)

/// @param drugId internal drug id

/// @param newState new state enum value

function updateDrugState(uint256 drugId, DrugState newState)

external drugExists(drugId)

{

Drug storage d = drugs[drugId];

// Only current owner or contract owner (admin/regulator) can update state

require(msg.sender == d.currentOwner || msg.sender == contractOwner, "Not authorized to change state");

d.state = newState;

emit DrugStateChanged(drugId, newState);

}

/// @notice Mark a batch as recalled (only contract owner / regulator)

function recallDrug(uint256 drugId, string calldata notes) external onlyContractOwner drugExists(drugId) {

Drug storage d = drugs[drugId];

d.state = DrugState.Recalled;

// Append a transfer-like record with same from/to to show recall event context

d.transfers.push(TransferRecord({

from: d.currentOwner,

to: d.currentOwner,

timestamp: block.timestamp,

notes: notes

}));

emit DrugRecalled(drugId, notes);

emit DrugStateChanged(drugId, DrugState.Recalled);

}

// Quality checks (IPFS linked)

/// @notice Add a quality check record for a drug (only QualityInspector or Regulator)

/// @param drugId internal drug id

/// @param location inspection location (lab, warehouse)

/// @param temperature °C (int16)

/// @param humidity % (uint8)

/// @param passed pass/fail boolean

/// @param remarks textual notes

/// @param documentHash IPFS CID

/// @param ipfsGatewayURL optional gateway link string. For example: https://ipfs.io/ipfs/

function addQualityCheck(

uint256 drugId,

string calldata location,

int16 temperature,

uint8 humidity,

bool passed,

string calldata remarks,

string calldata documentHash,

string calldata ipfsGatewayURL

)

external drugExists(drugId)

{

require(isRegistered[msg.sender], "Inspector not registered");

require(participants[msg.sender].active, "Inspector not active");

//Role r = participants[msg.sender].role;

require(participants[msg.sender].role == Role.QualityInspector || participants[msg.sender].role == Role.Regulator, "Only inspector or regulator can add QC");

require(bytes(documentHash).length > 0, "documentHash (IPFS CID) required");

Drug storage d = drugs[drugId];

d.qualityHistory.push(QualityCheck({

inspector: msg.sender,

timestamp: block.timestamp,

location: location,

temperature: temperature,

humidity: humidity,

passed: passed,

remarks: remarks,

documentHash: documentHash,

ipfsGatewayURL: ipfsGatewayURL

}));

emit QualityCheckAdded(drugId, msg.sender, documentHash, ipfsGatewayURL);

}

// Queries & convenience getters

/// @notice Verify a batch by batchNumber

/// @param batchNumber unique batch number

/// @return exists true if registered

/// @return drugId internal id;If not found then, return 0

function verifyByBatch(string calldata batchNumber) external view returns (bool exists, uint256 drugId) {

uint256 id = batchToDrugId[batchNumber];

if (id == 0) return (false, 0);

return (true, id);

}

/// @notice Get basic details of a drug

function getDrugBasic(uint256 drugId) external view drugExists(drugId)

returns (

uint256 id,

string memory name,

string memory batchNumber,

string memory documentHash,

address manufacturer,

uint256 manufactureDate,

uint256 expiryDate,

DrugState state,

address currentOwner

)

{

Drug storage d = drugs[drugId];

return (

d.drugId,

d.name,

d.batchNumber,

d.documentHash,

d.manufacturer,

d.manufactureDate,

d.expiryDate,

d.state,

d.currentOwner

);

}

/// @notice Get ownership history (addresses) for a drug

function getOwnershipHistory(uint256 drugId) external view drugExists(drugId) returns (address[] memory) {

return drugs[drugId].ownershipHistory;

}

/// @notice Number of transfers recorded for a drug

function getTransfersCount(uint256 drugId) external view drugExists(drugId) returns (uint256) {

return drugs[drugId].transfers.length;

}

/// @notice Get a single transfer record by index

function getTransferByIndex(uint256 drugId, uint256 index) external view drugExists(drugId)

returns (address from, address to, uint256 timestamp, string memory notes)

{

TransferRecord storage t = drugs[drugId].transfers[index];

return (t.from, t.to, t.timestamp, t.notes);

}

/// @notice Number of QC records for a drug

function getQualityChecksCount(uint256 drugId) external view drugExists(drugId) returns (uint256) {

return drugs[drugId].qualityHistory.length;

}

/// @notice Get a single QC record by index (safer than returning array)

function getQualityCheckByIndex(uint256 drugId, uint256 index) external view drugExists(drugId)

returns (

address inspector,

uint256 timestamp,

string memory location,

int16 temperature,

uint8 humidity,

bool passed,

string memory remarks,

string memory documentHash,

string memory ipfsGatewayURL

)

{

QualityCheck storage q = drugs[drugId].qualityHistory[index];

return (

q.inspector,

q.timestamp,

q.location,

q.temperature,

q.humidity,

q.passed,

q.remarks,

q.documentHash,

q.ipfsGatewayURL

);

}

/// @notice Return participant details for an address

function getParticipant(address addr) external view returns (Participant memory) {

require(isRegistered[addr], "Not registered");

return participants[addr];

}

/// @notice Total registered drug batches

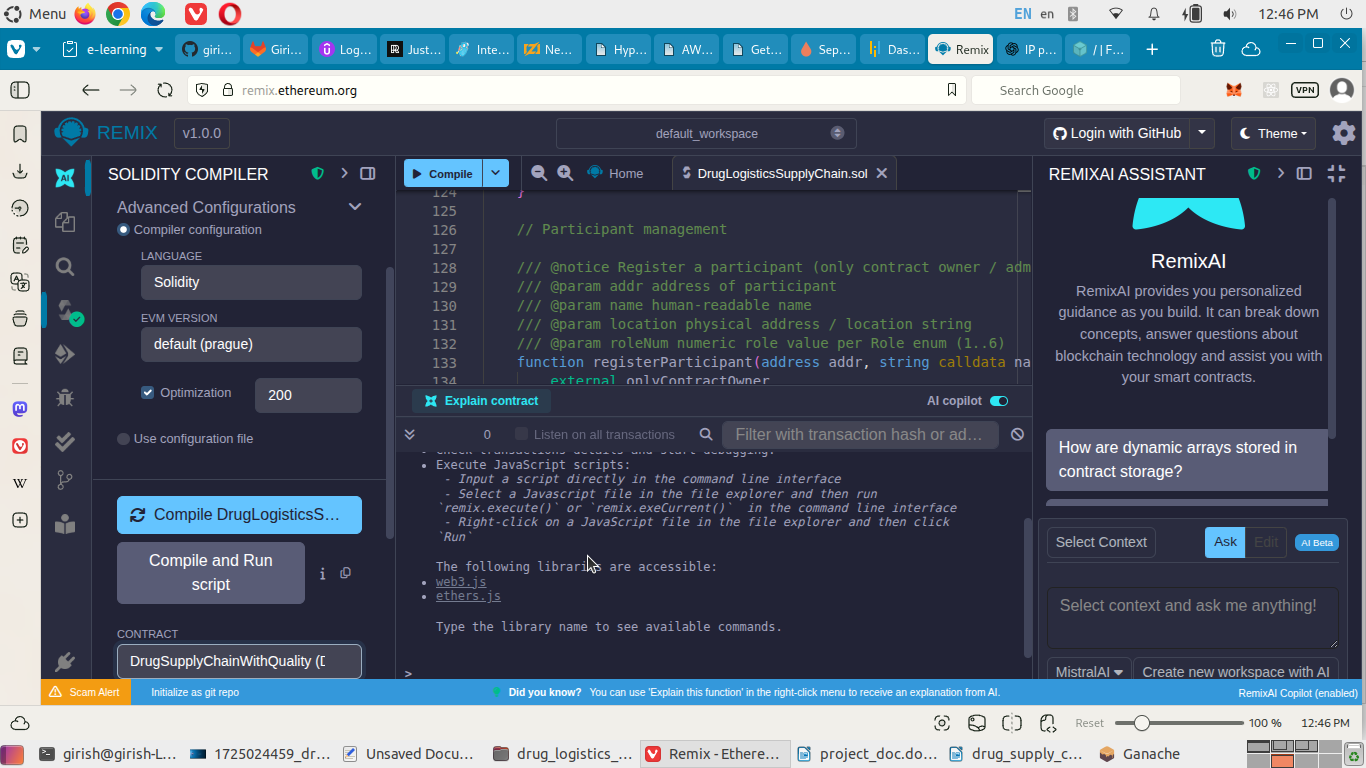
function totalDrugs() external view returns (uint256) {

return \_drugCounter;

}

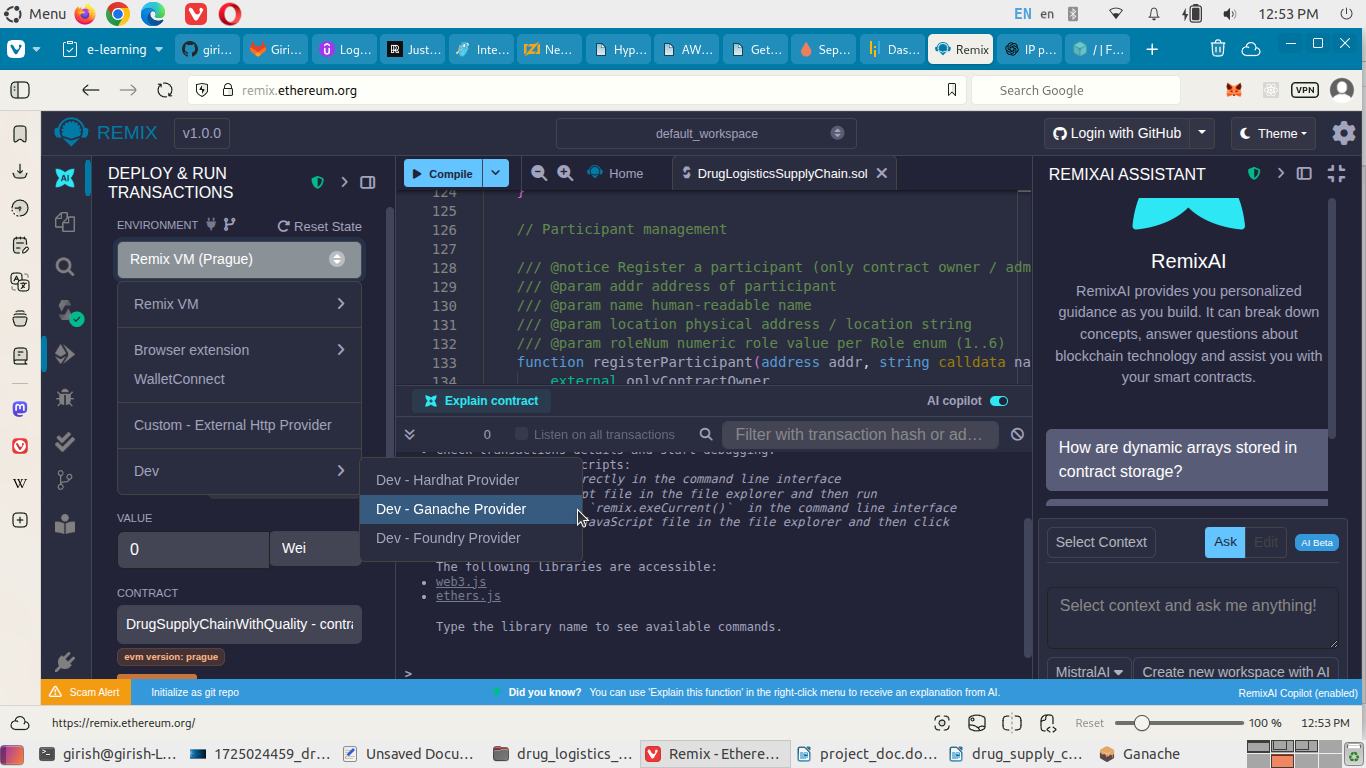
}

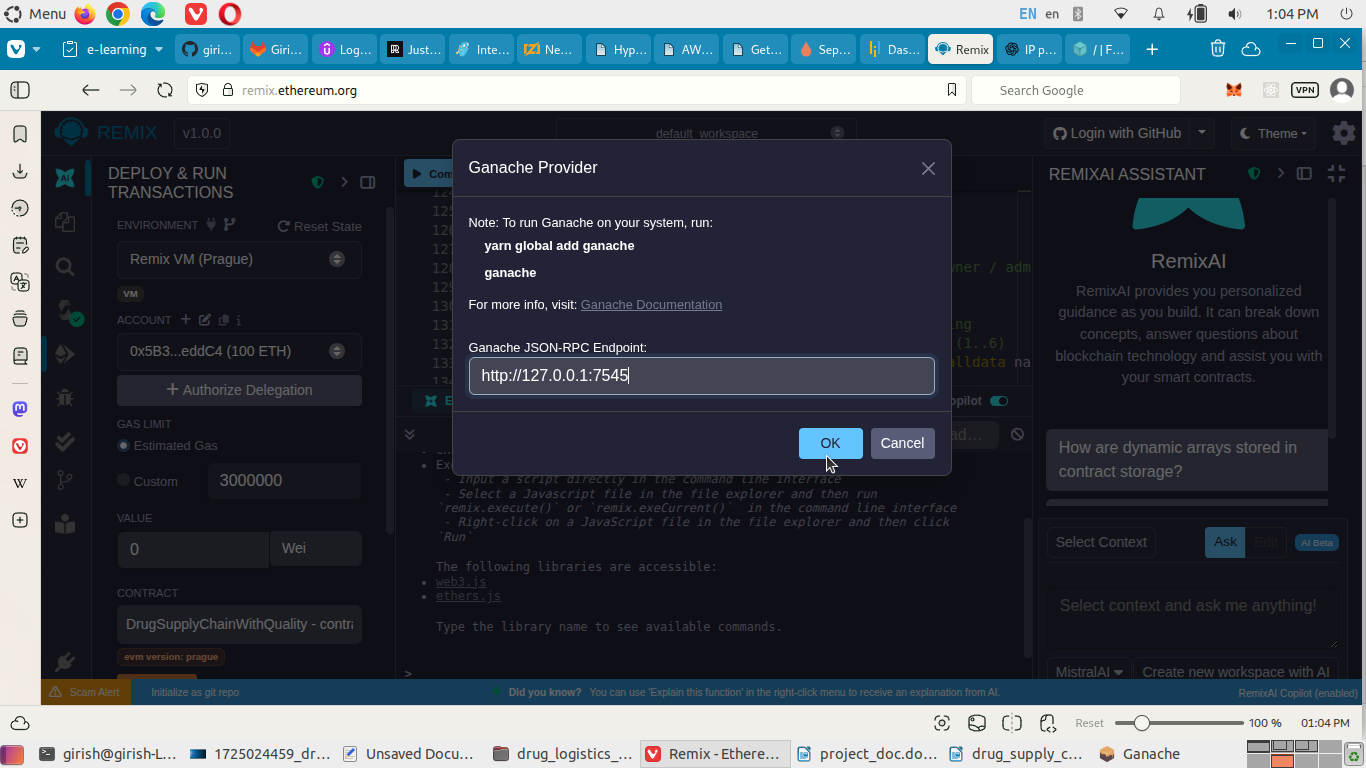
Now we need to compile the smart contract using Remix IDE. Click on the **Solidity Compiler** icon on the left bar in Remix IDE. Click on **Advanced Configurations** tab to open the section. Click on **optimization** checkbox so that, the smart contract could be executed successfully. Click on **Compile DrugLogisticsSupplyChain** button in the left pane of the Remix IDE.



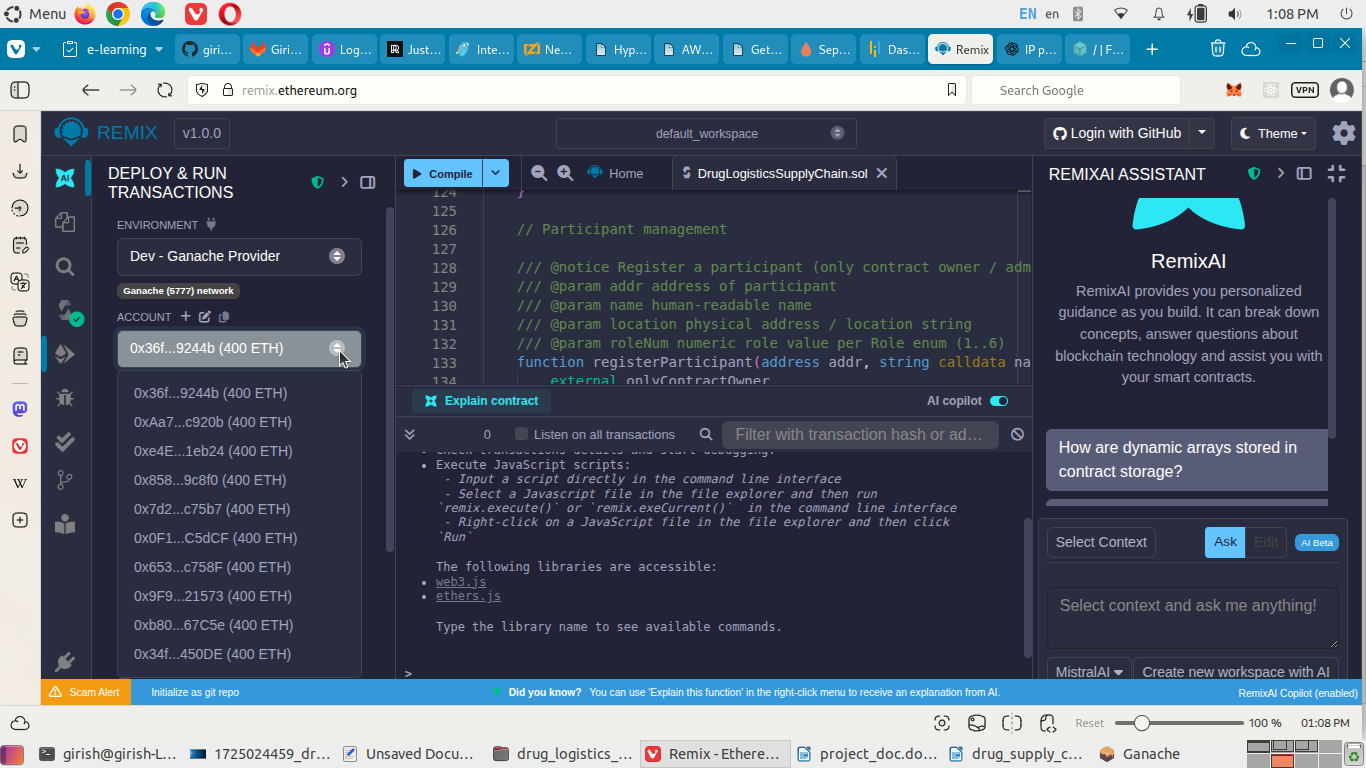
A smart contract is created with name **DrugSupplyChainWithQuality**.

Now, click on **Deploy & run transactions** icon on the left tab. A section to deploy the smart contract will appear. Click on the ENVIRONMENT option; then, navigate to **Dev -> Dev - Ganache Provider**. A dialog box will appear. In this dialog box, we have to give the Ganache RPC server url, in this case, [**http://127.0.0.1:7545**](http://127.0.0.1:7545/). This is the **Ganache JSON-RPC Endpoint**.





Click on **OK** button. Ganache blockchain is now connected with the 10 accounts listed in the interface.Ganache network ID 5777 can be seen below the ENVIRONMENT option. We will assign six of these account addresses to our participants when we run the tasks of the deployed smart contract. The smart contract will be run by an account with role as Regulator. This account will be made the admin account which will run the smart contract.



**Select** the account to deploy the smart contract. This account will be added as a participant with Regulator role, by default.

Due to issues with Ganache, we will proceed with Remix (Prague) environment accounts from here.

**Select** the contract to be deployed, under CONTRACT (There is only one contract in this case).

==================

0x5B38Da6a701c568545dCfcB03FcB875f56beddC4 = Regulator/ Admin, default

0xAb8483F64d9C6d1EcF9b849Ae677dD3315835cb2 = Manufacturer

0x4B20993Bc481177ec7E8f571ceCaE8A9e22C02db = Distributor

0x78731D3Ca6b7E34aC0F824c42a7cC18A495cabaB = Pharmacy

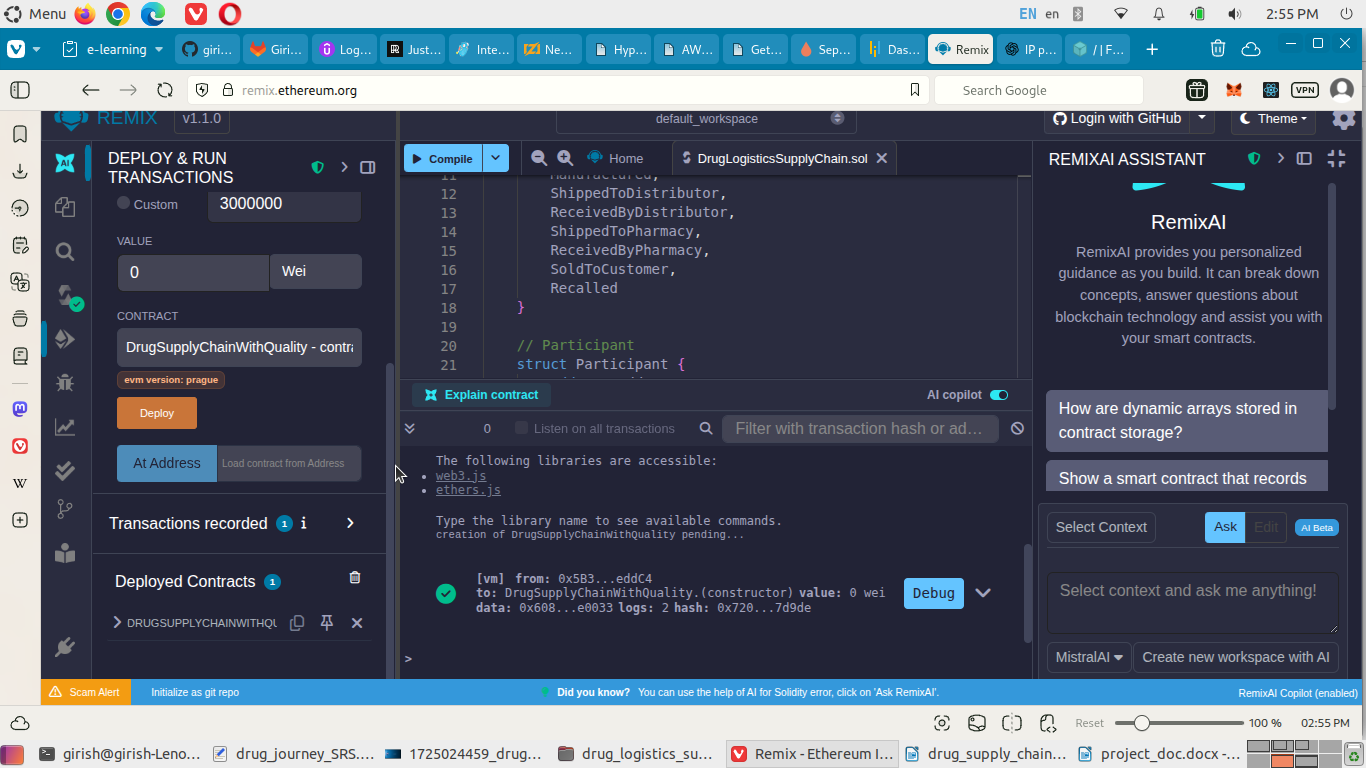
0x617F2E2fD72FD9D5503197092aC168c91465E7f2 = Quality Inpector

0x17F6AD8Ef982297579C203069C1DbfFE4348c372 = Consumer

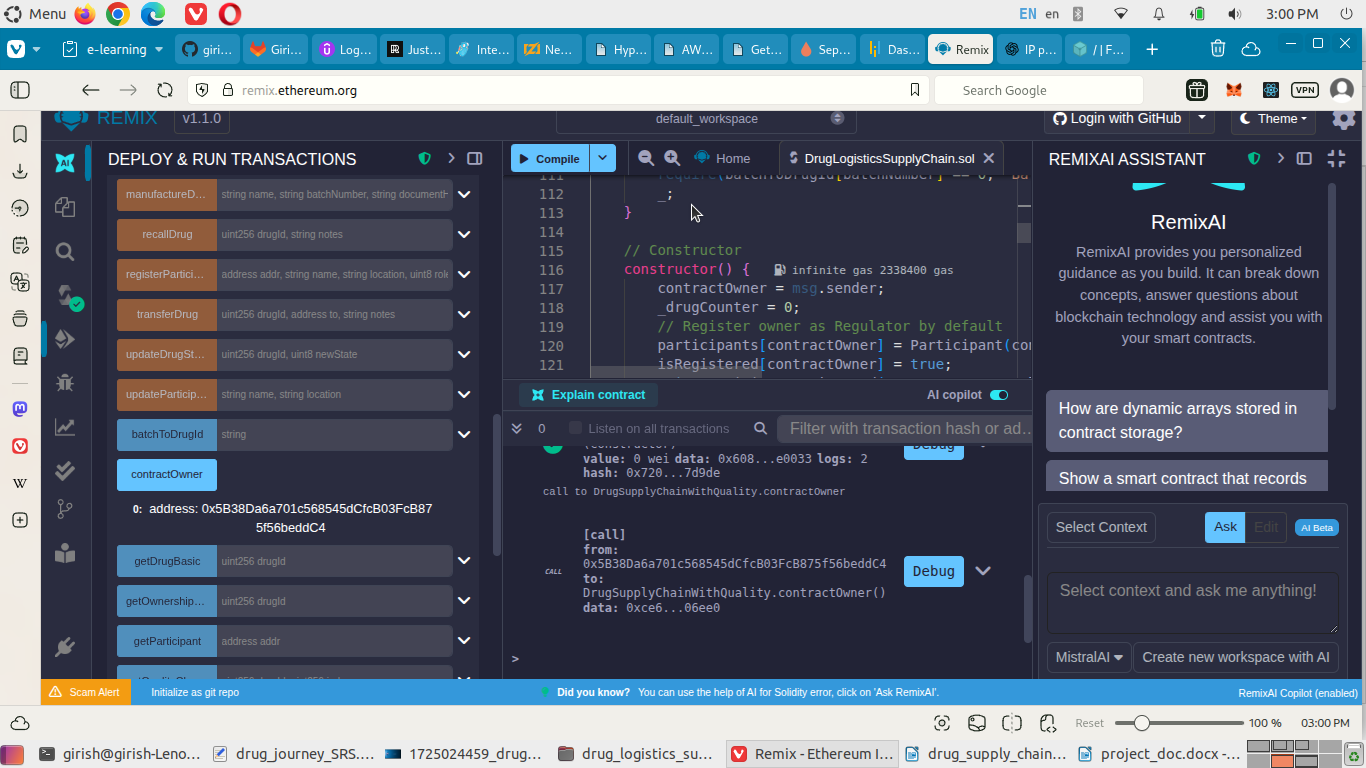
==================

Using Regulator/ Admin account, we create the smart contract by clicking on **Deploy** button.

The smart contract is created in the Deployed Contracts section.



Expand the contract to see the actions/tasks of the smart contracts. **Click** on the **contractOwner** button to see the address of the contract owner i.e. participant with Regulator role.



Use **registerParticipant** task to add entity to the Drug logistics supply chain. One for each role. Click on **transact** button. The data being as follows.

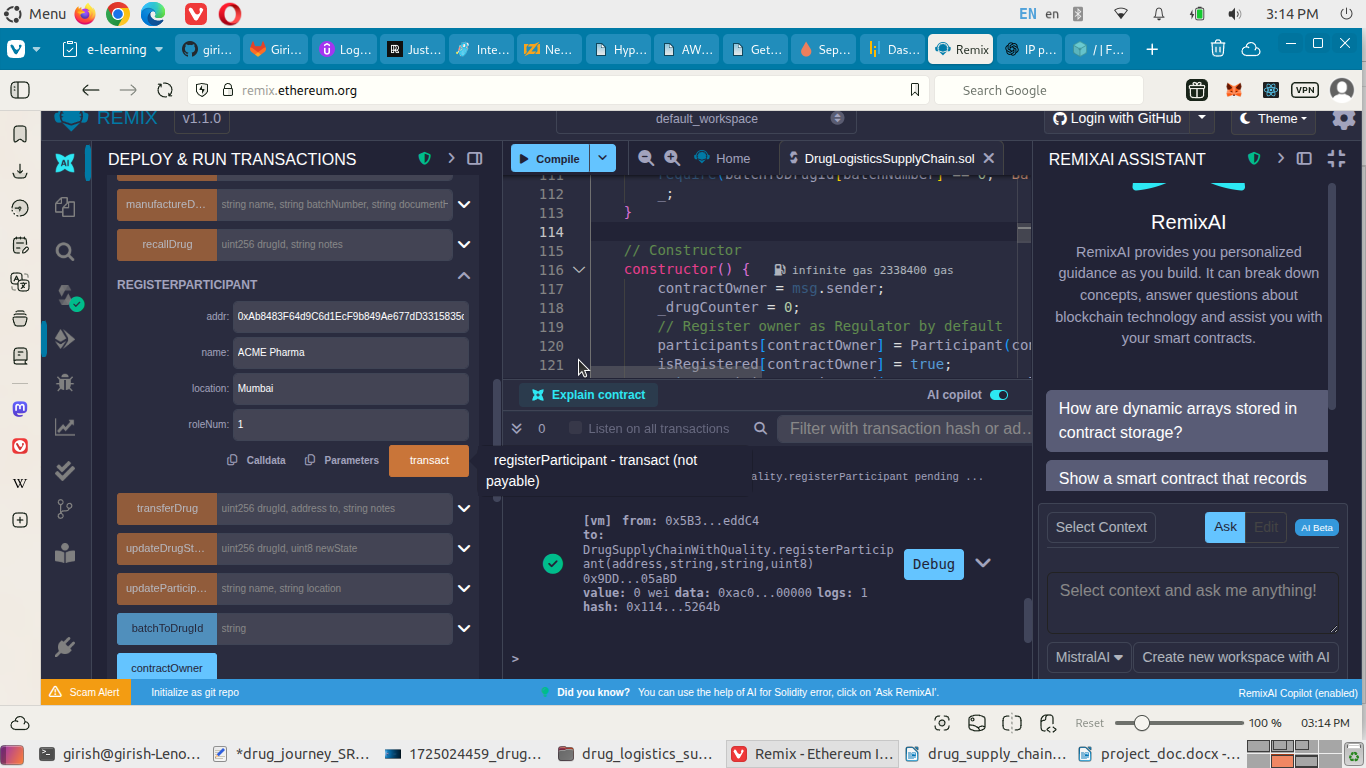
0xAb8483F64d9C6d1EcF9b849Ae677dD3315835cb2, ACME Pharma, Mumbai, 1-> Manufacturer

0x4B20993Bc481177ec7E8f571ceCaE8A9e22C02db, MediLink Distributors, Delhi, 2-> Distribut.r

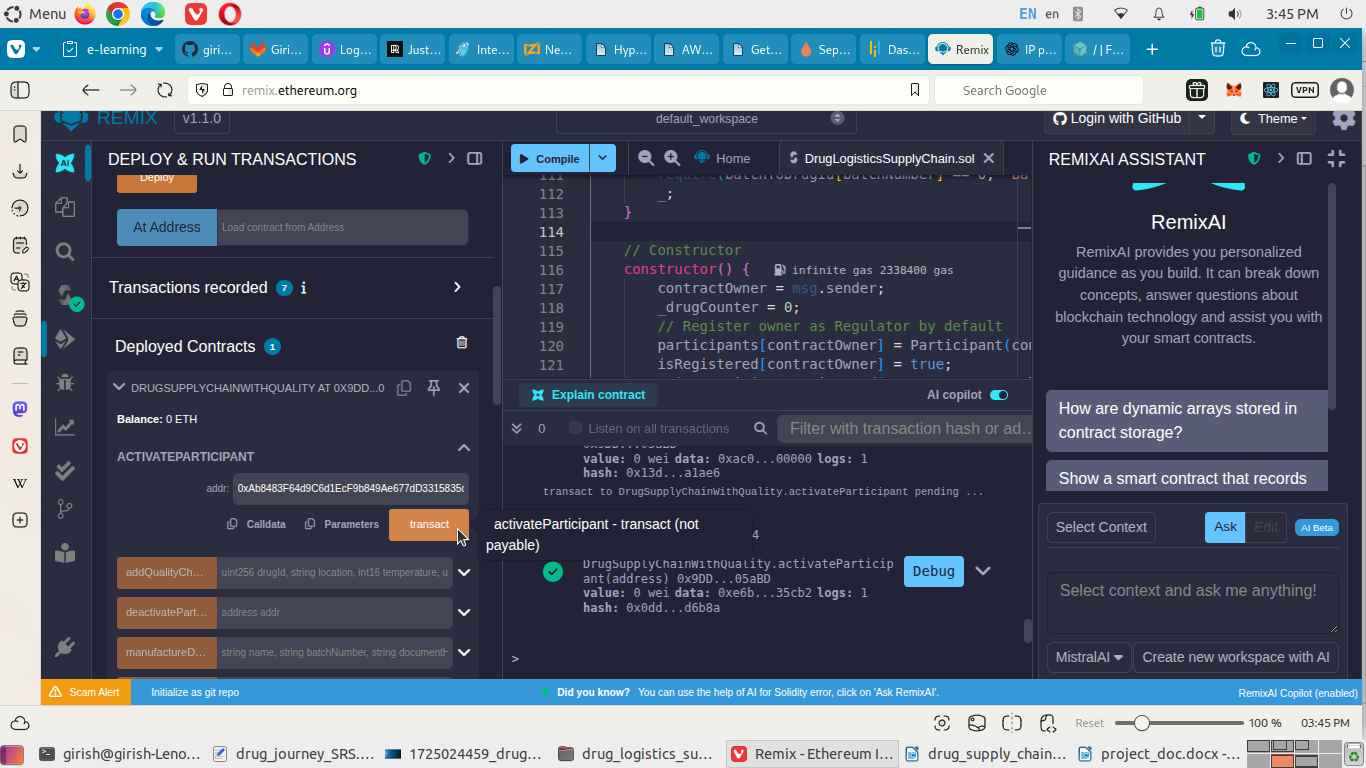
0x78731D3Ca6b7E34aC0F824c42a7cC18A495cabaB, HealthPlus, Pune,3-> Pharmacy

0x617F2E2fD72FD9D5503197092aC168c91465E7f2, NABL Lab, Hyderabad, 5-> Quality Insp.r

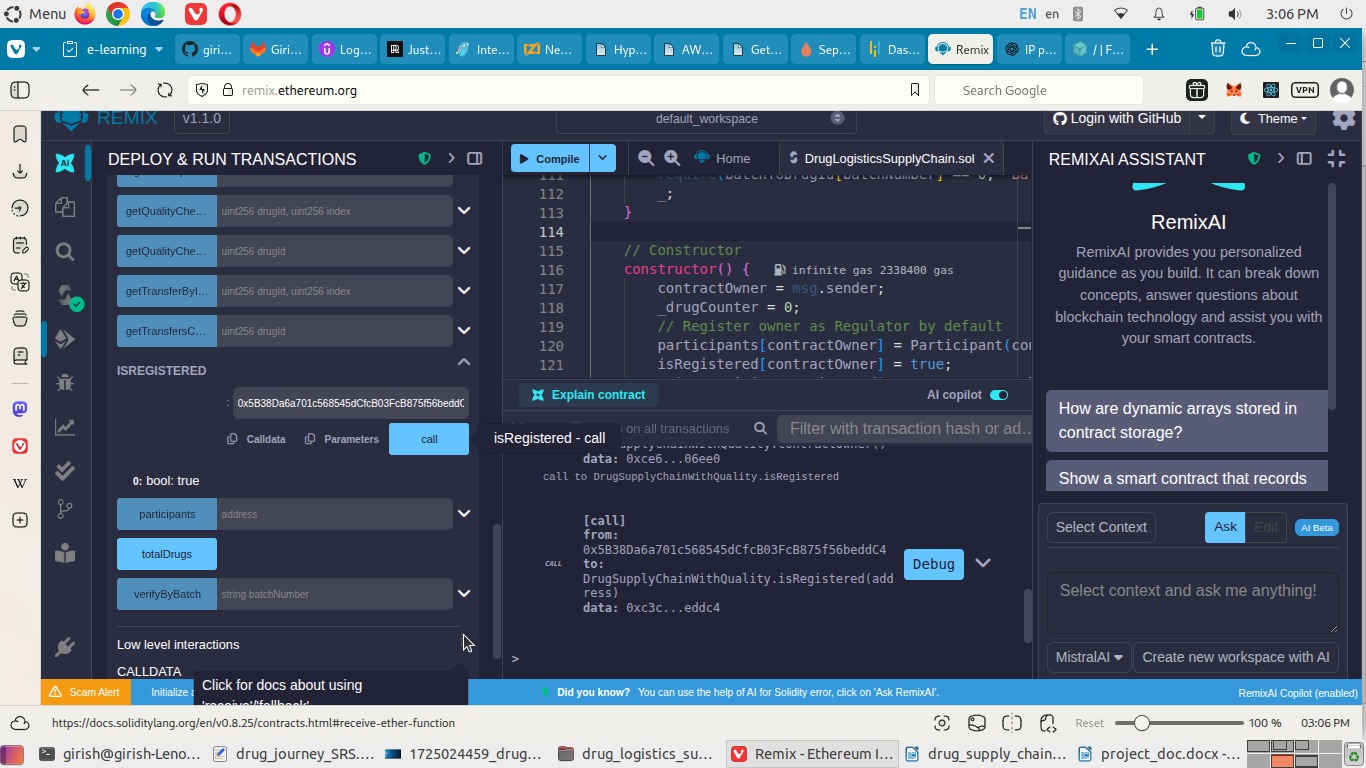
0x17F6AD8Ef982297579C203069C1DbfFE4348c372, Girish, Chennai, 6-> Consumer



Now, having the entities onboard, we need to activate each of these participants. **Click** on **activateParticipant** task. Execute this task for each entity so as to enable them to carry out operations.



To check whether a Participant is registered with the system, we click on **isRegistered** button. You will need to supply the address of the Participant. The result is boolean.



Now, the manufacturer creates a new batch of drugs. For this, first the manufacturer uploads Product Specification to **IPFS**. This will return a CID for the file. The file is uploaded on IPFS using the command: **ipfs add <path\_to\_file>**

Here, the CID = QmSMLqsio3rsRZAgWNH1pBw2hYzyxYqzc4a2T63znVptrn

The IPFS Gateway URL = [**https://ipfs.io/ipfs/**](https://ipfs.io/ipfs/)**<CID>** or

[http://127.0.0.1:5001/ipfs/bafybeif6abowqcavbkz243biyh7pde7ick5kkwwytrh7pd2hkbtuqysjxy/#/ipfs/QmSMLqsio3rsRZAgWNH1pBw2hYzyxYqzc4a2T63znVptrn](http://127.0.0.1:5001/ipfs/bafybeif6abowqcavbkz243biyh7pde7ick5kkwwytrh7pd2hkbtuqysjxy/" \l "/ipfs/QmSMLqsio3rsRZAgWNH1pBw2hYzyxYqzc4a2T63znVptrn)

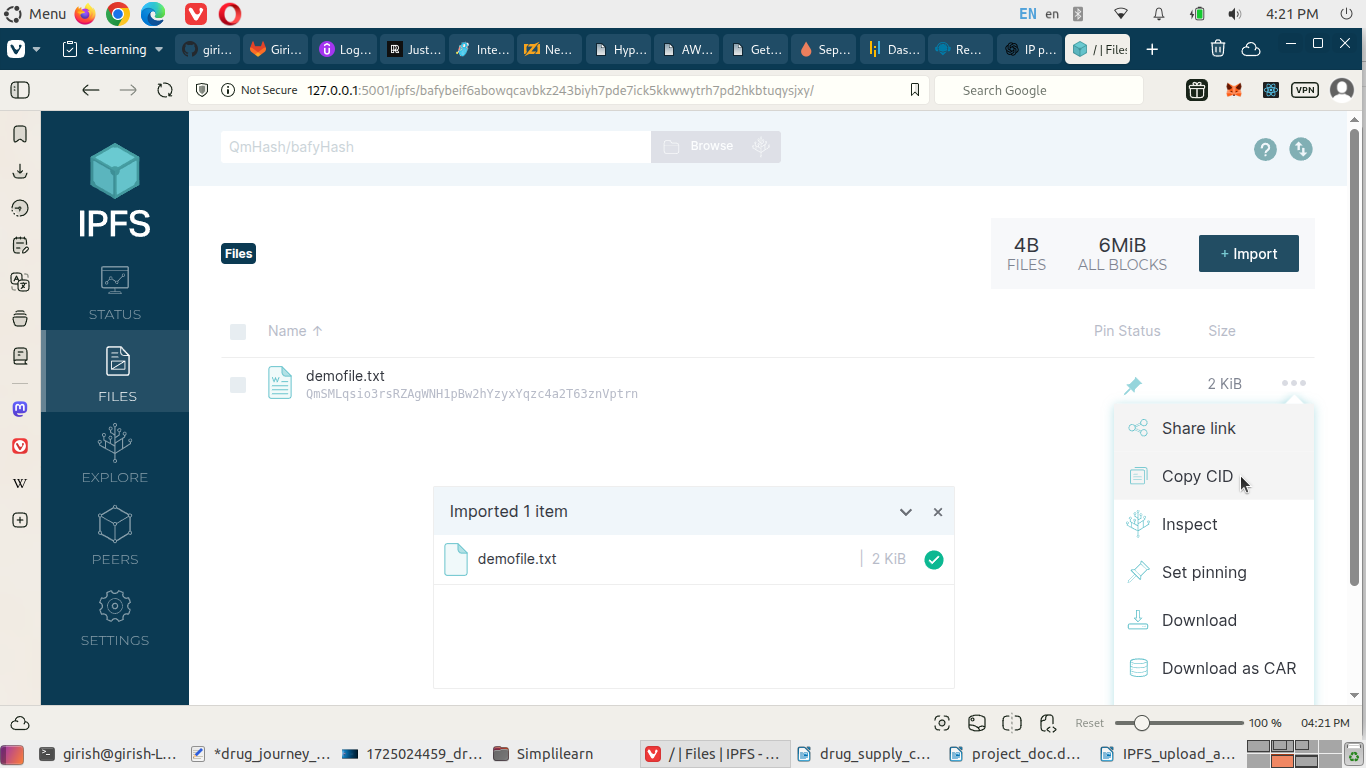
(IPFS setup on my local)

**Otherwise,** you can use the **GUI** as well.



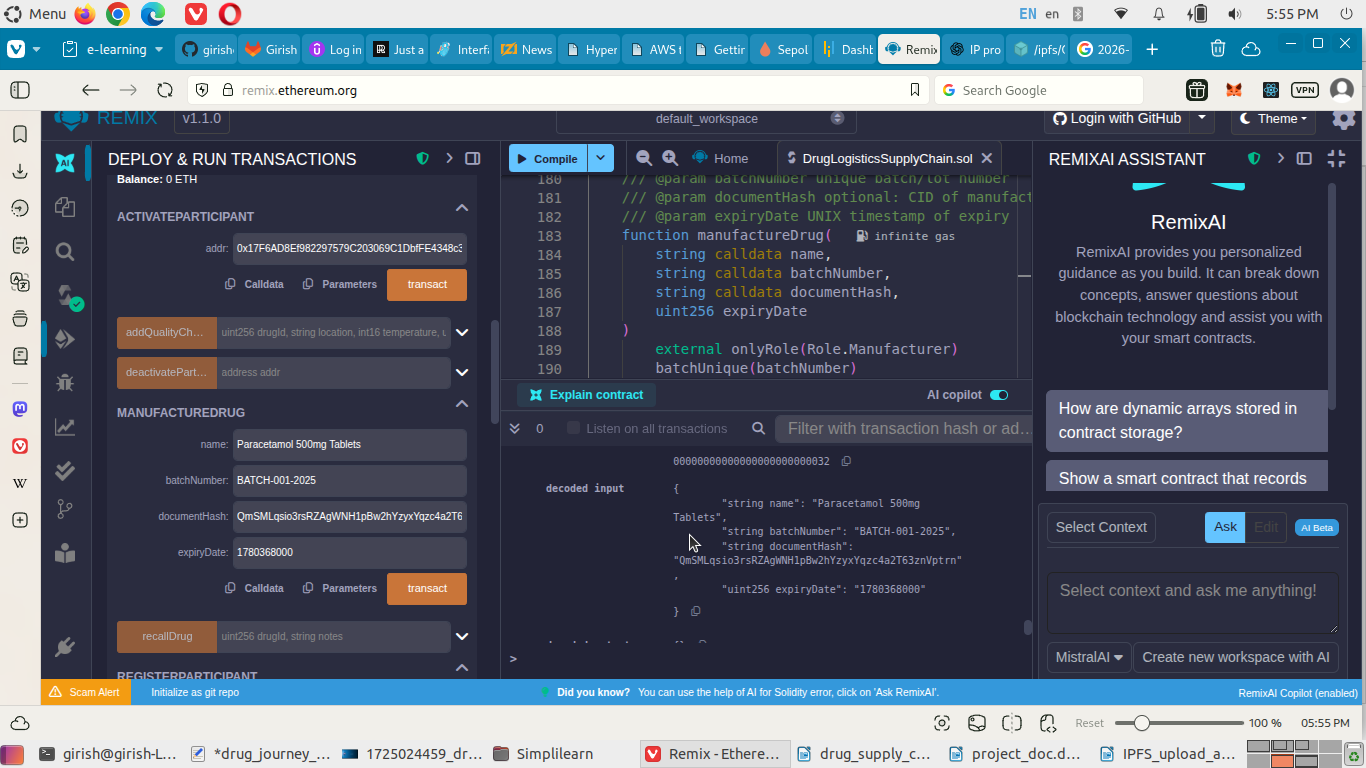
Note the CID of the file.

We need to add this data into our system that a drug has been manufactured.



You need to switch to the account for manufacturer before executing the task. So, for the task **manufactureDrug**, following is the data to input.

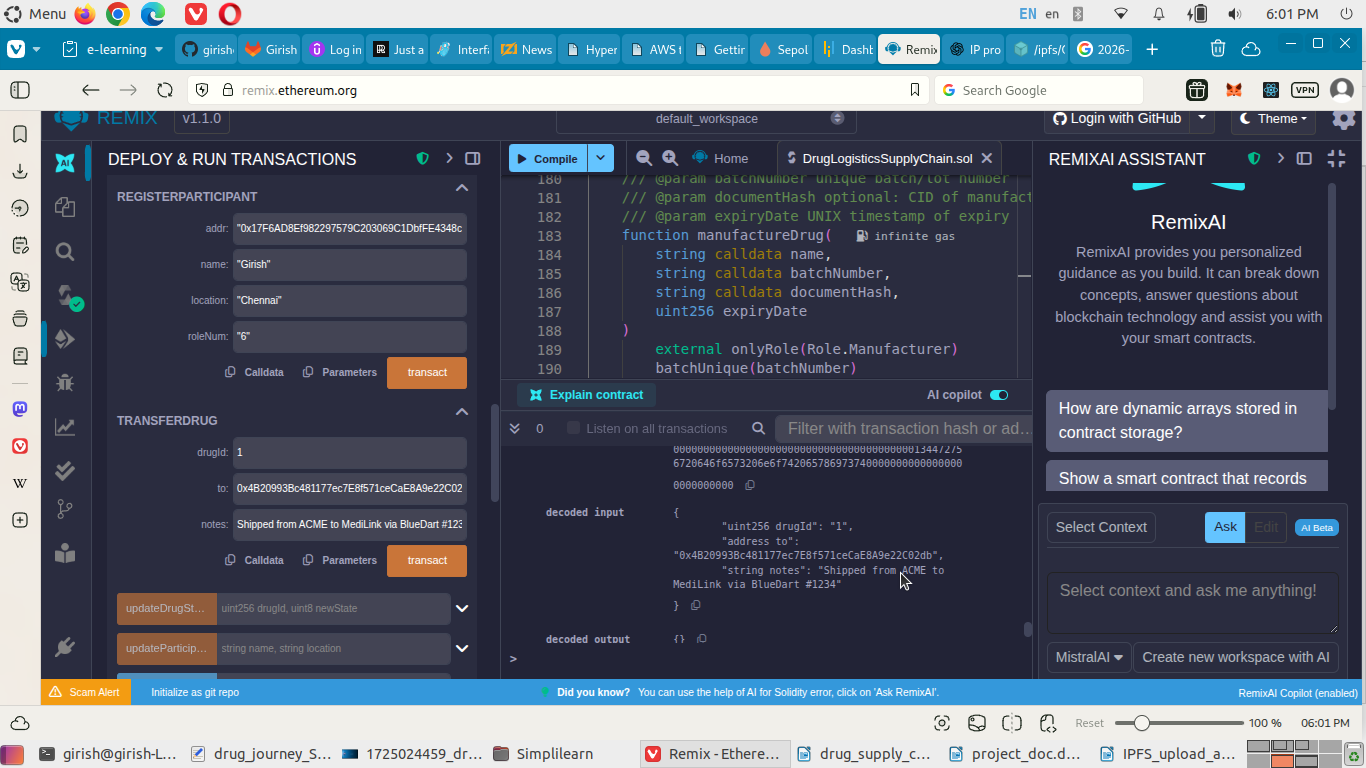
Paracetamol 500mg Tablets, BATCH-001-2025, QmSMLqsio3rsRZAgWNH1pBw2hYzyxYqzc4a2T63znVptrn, 1780368000 (2026-06-01)



This will emit an event DrugManufactured. The manufacturer transfers the drug to the distributor.

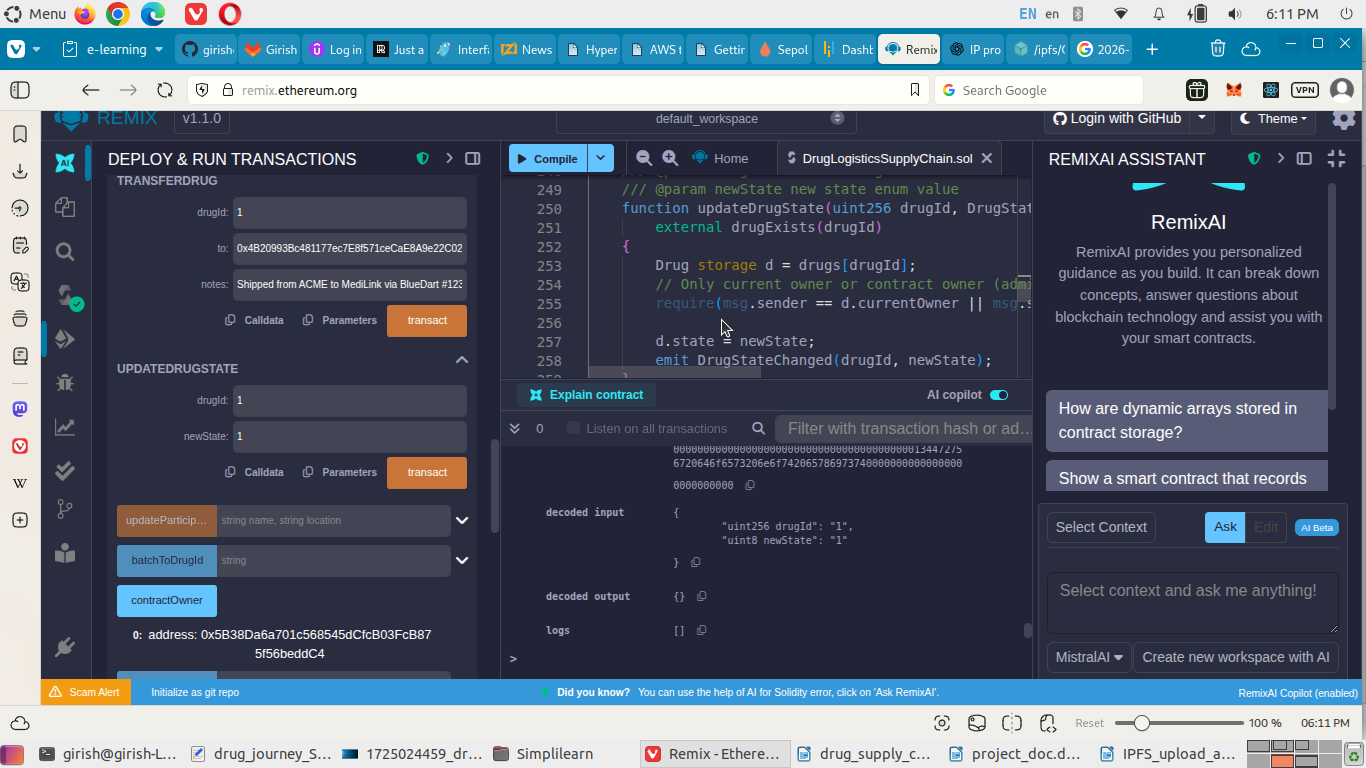
Click on **transferDrug** and provide detail as below.

1, 0x4B20993Bc481177ec7E8f571ceCaE8A9e22C02db,Shipped from ACME to MediLink via BlueDart #1234

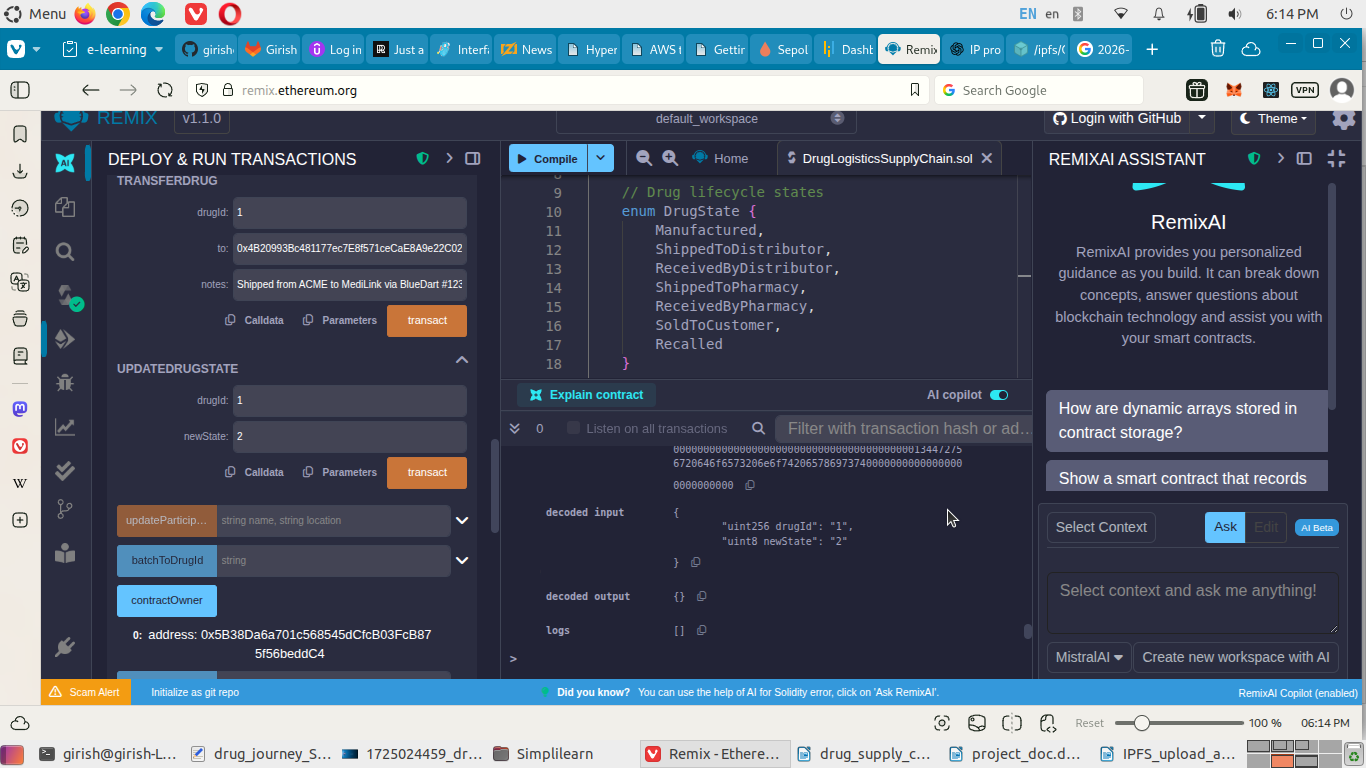


Now, the drug state need to be updated. Click on **updateDrugState** and update the state of the drug.

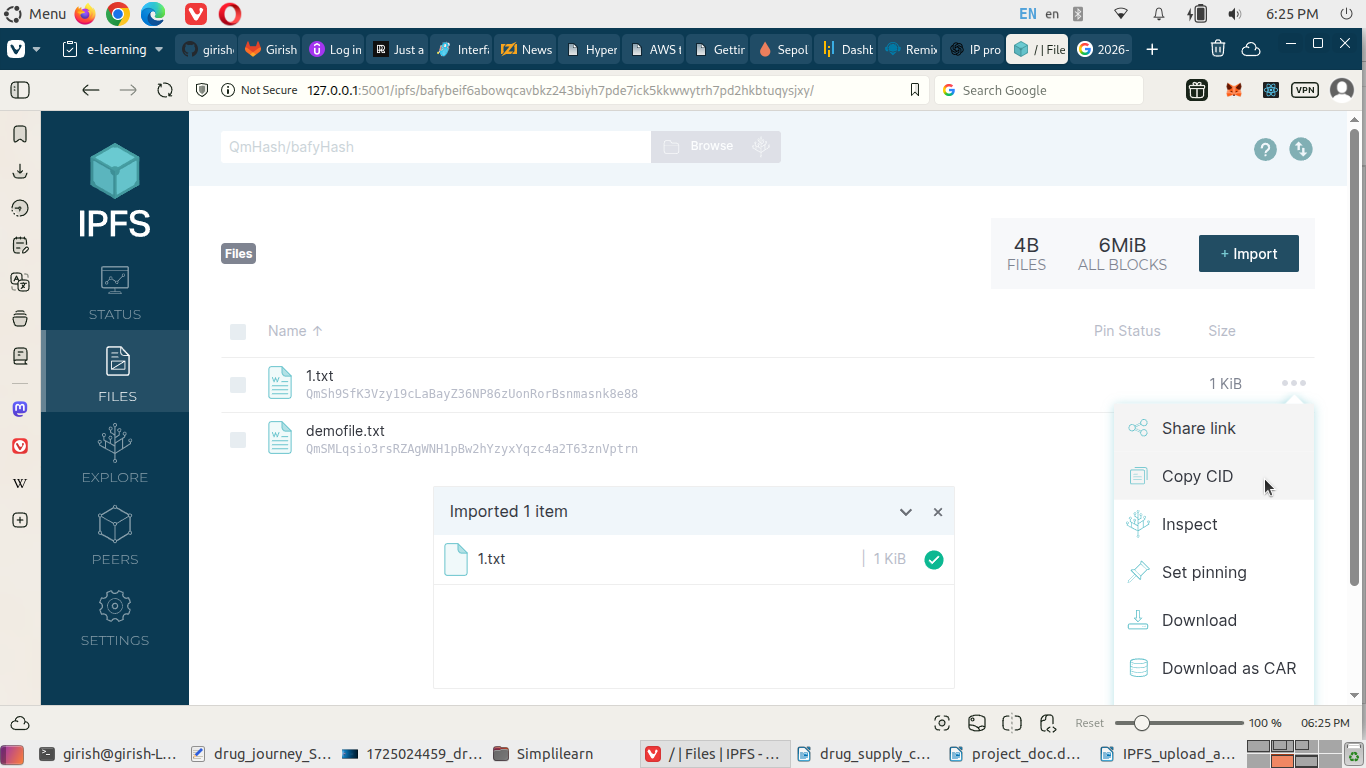
Provide input as: 1,1.



Now, the distributer will confirm the receipt of the drug.

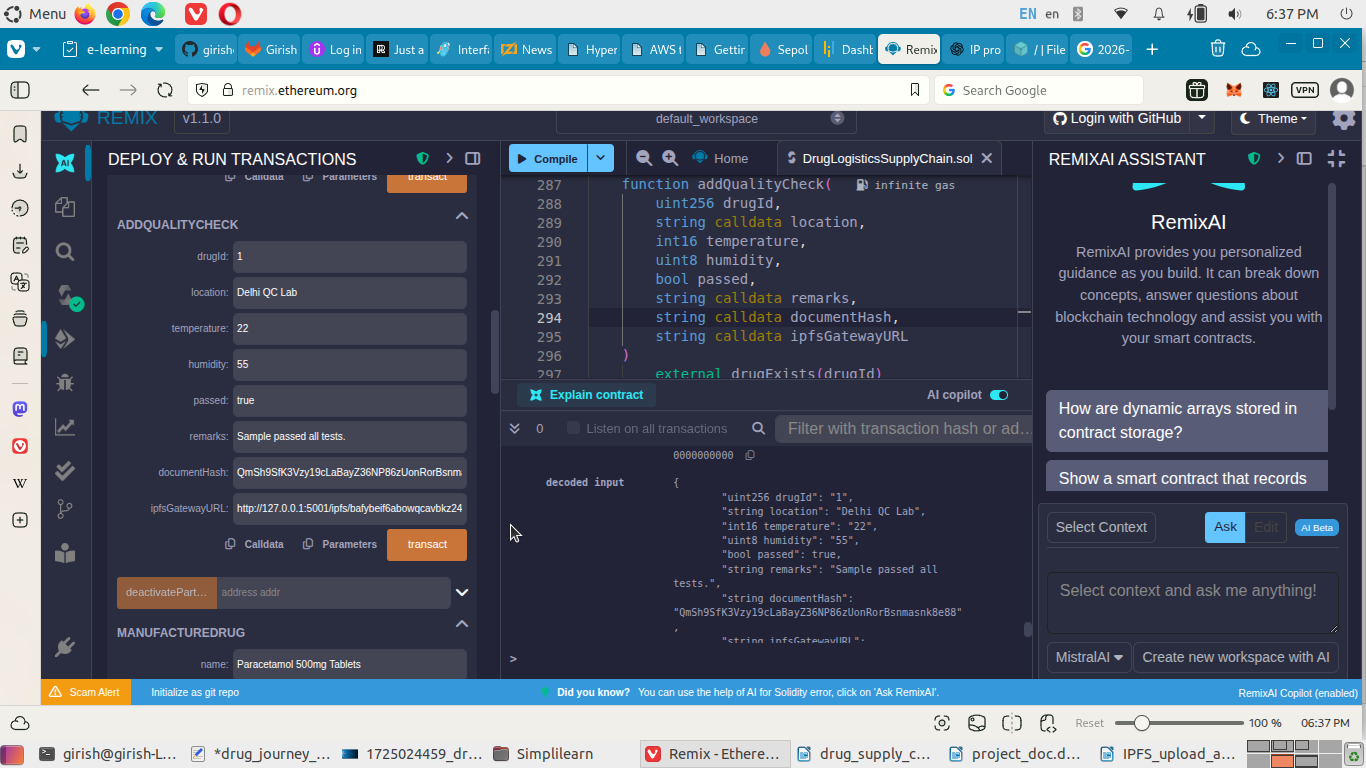


The distributor is the current owner. The Quality Inspector will perform a quality check. The quality inspector will upload a report on the IPFS.



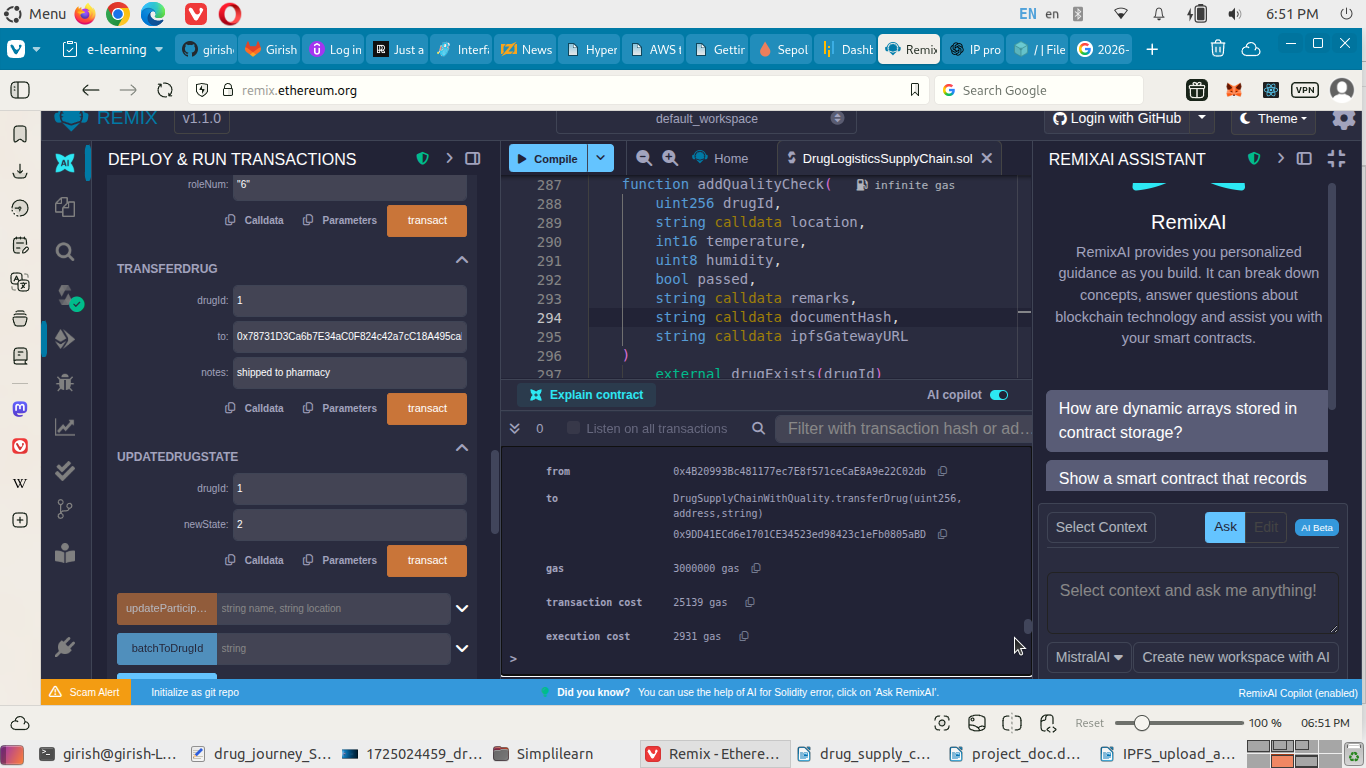
Copy the CID of the uploaded file. Now, change the account to Quality Inspector’s account. Click on **addQualityCheck** task and provide details as below.

1, Delhi QC Lab, 22, 55, true, "Sample passed all tests.", "QmSh9SfK3Vzy19cLaBayZ36NP86zUonRorBsnmasnk8e88", "http://127.0.0.1:5001/ipfs/bafybeif6abowqcavbkz243biyh7pde7ick5kkwwytrh7pd2hkbtuqysjxy/#/ipfs/QmSh9SfK3Vzy19cLaBayZ36NP86zUonRorBsnmasnk8e88"

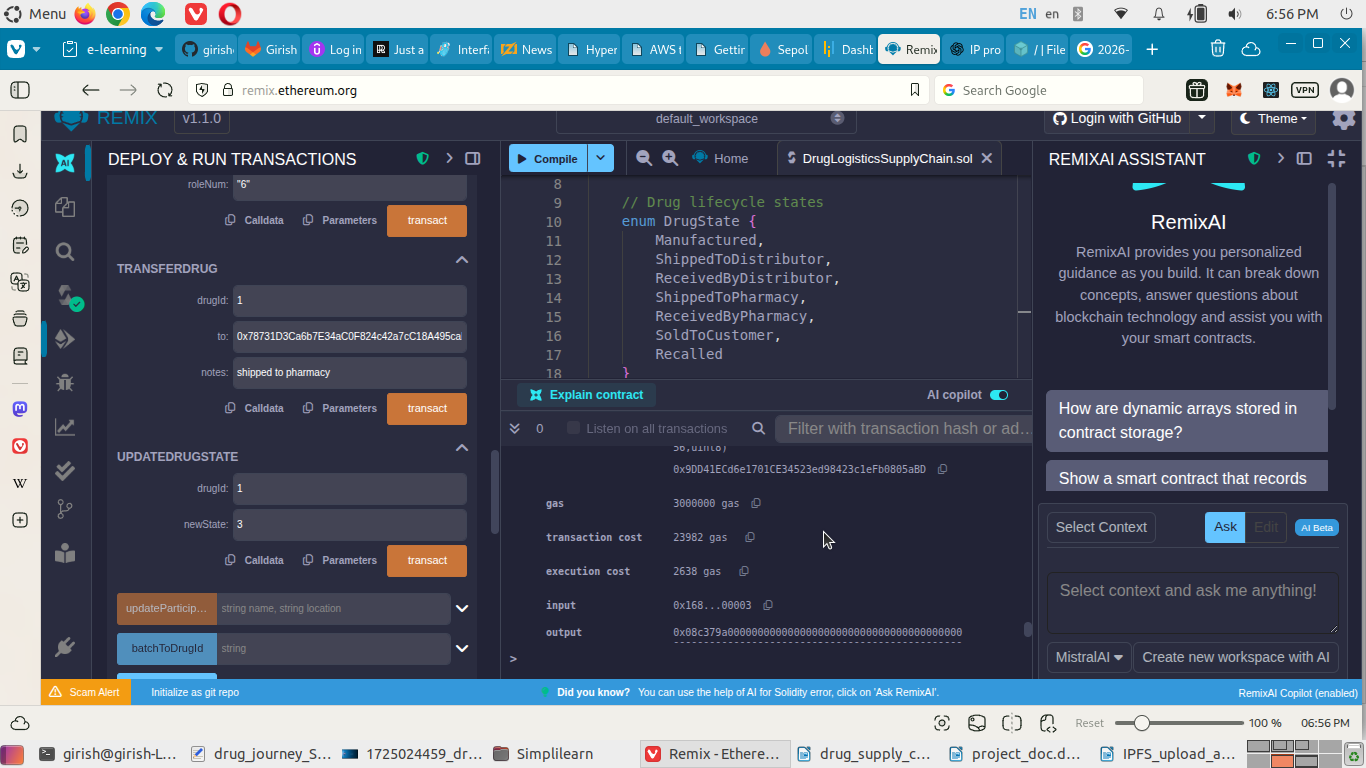


The distributor transfers the drug to the pharmacy. To do this, first change the account to distributor’s account. Then click on the **transferDrug** task. Now, provide the input as below.

1, 0x78731D3Ca6b7E34aC0F824c42a7cC18A495cabaB, shipped to pharmacy

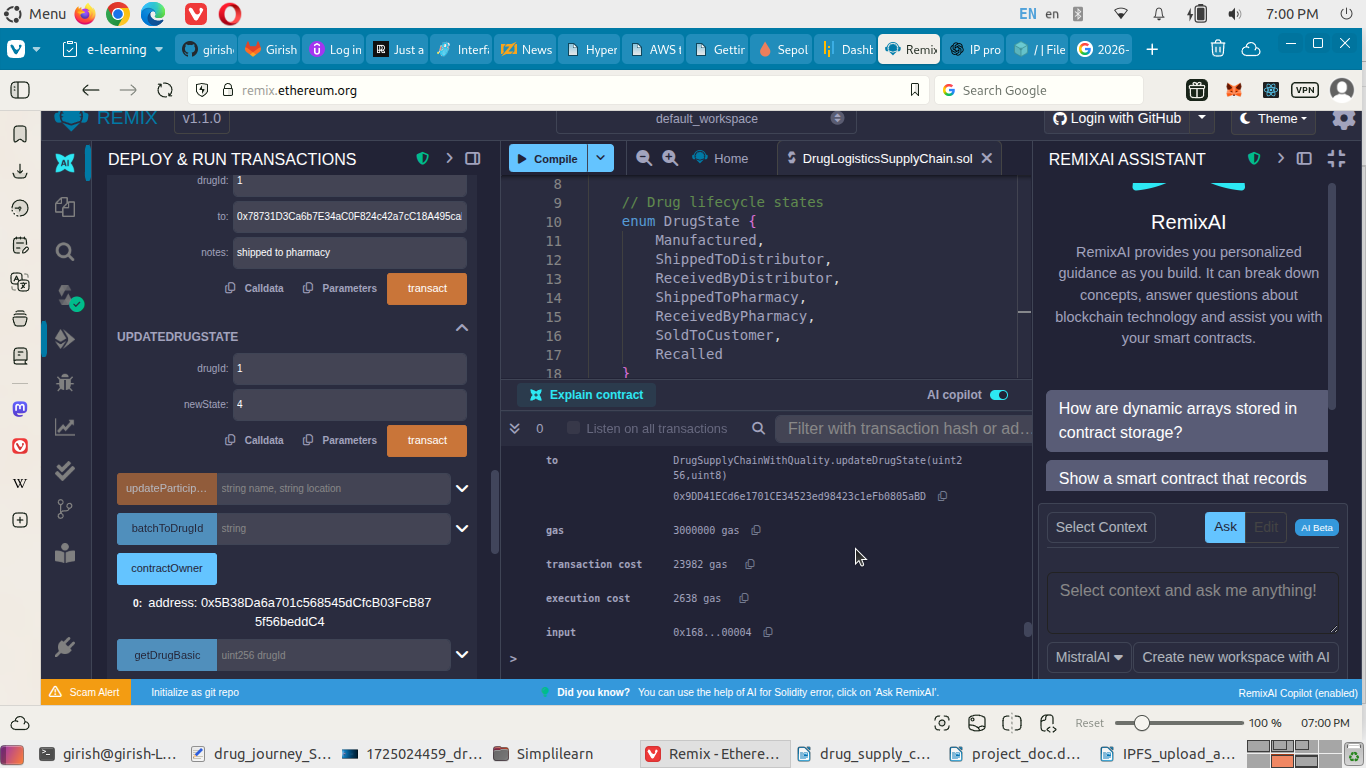


Update the drug transfer record. Give input as 1,3.



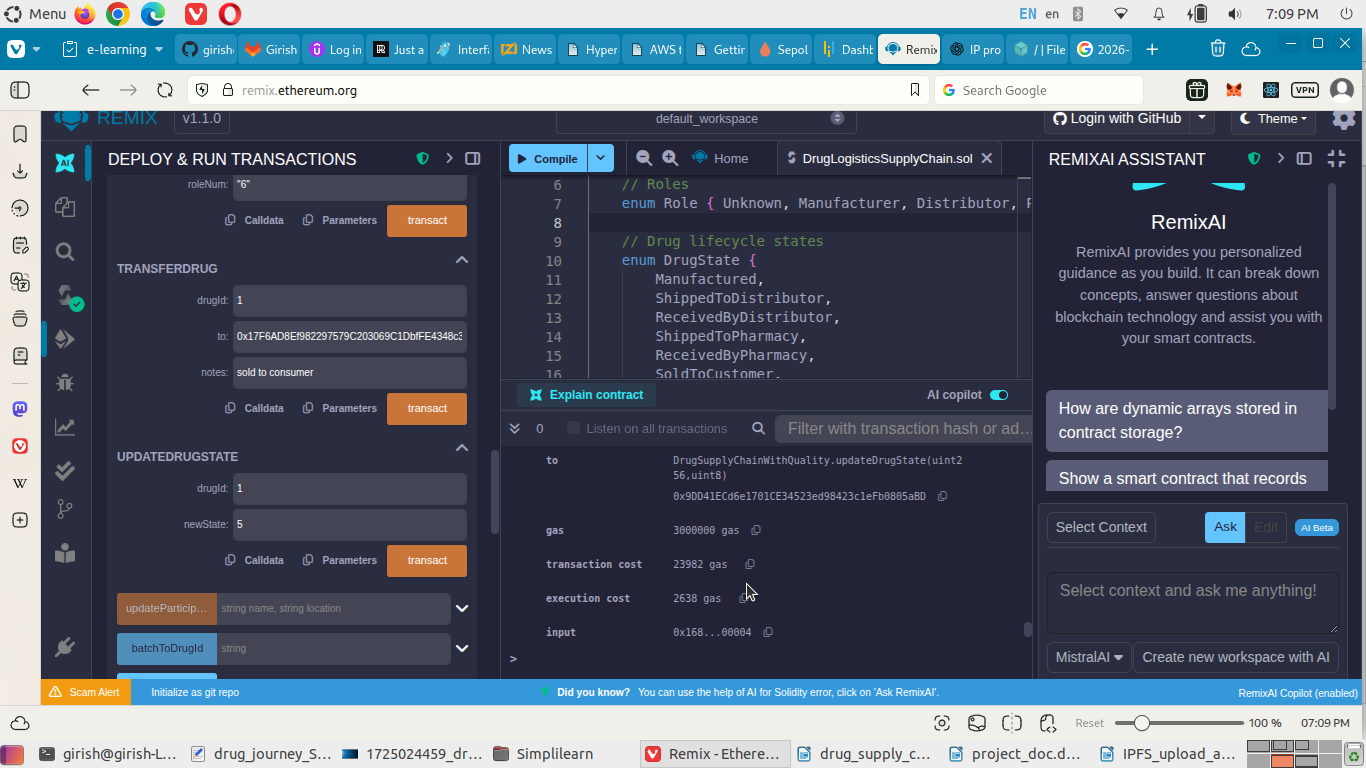
Now, the pharmacy has to confirm the transfer. First change the account to Pharmacy’s account.

Return to the **updateDrugState** and give input as: 1,4.

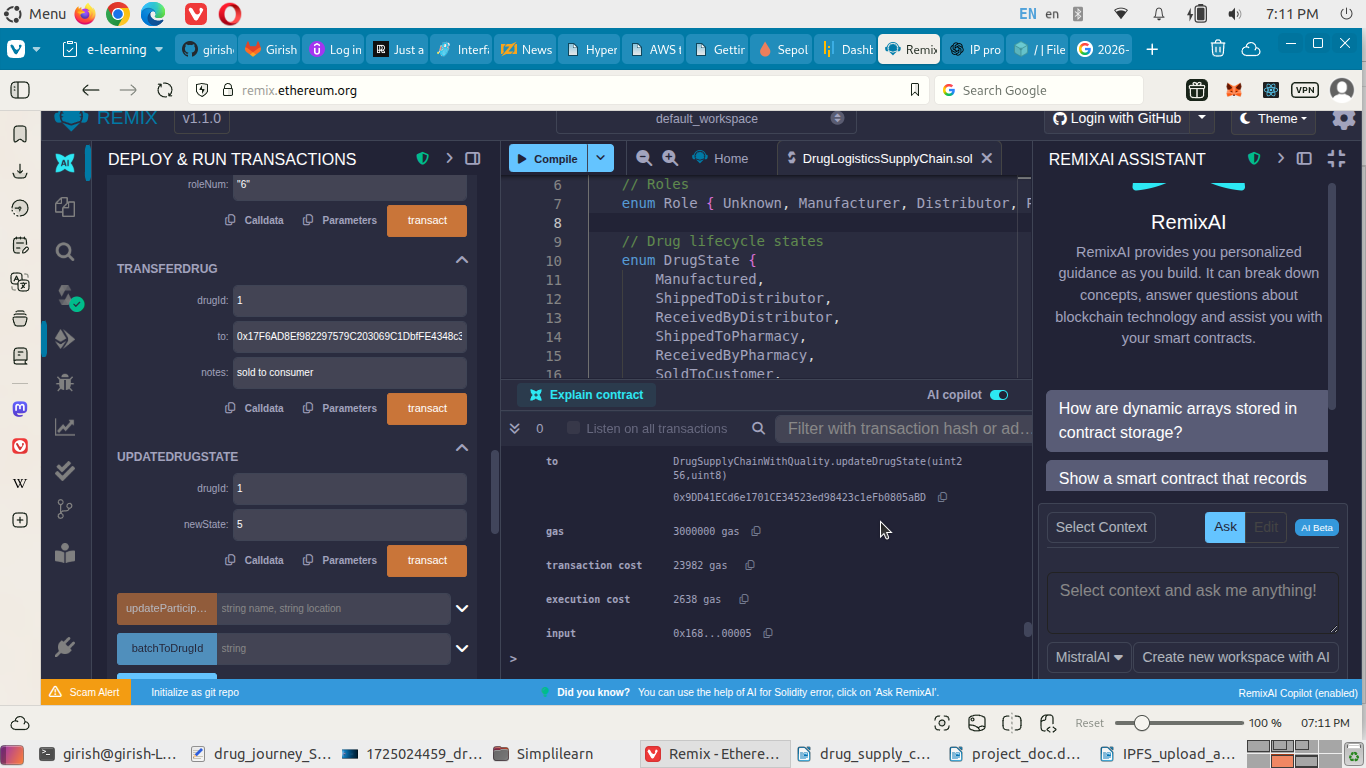


After this, the pharmacy will sell the drug to the consumer. So, the current selected account should be of the pharmacy. Click on the transferDrug and input values as:

1, 0x17F6AD8Ef982297579C203069C1DbfFE4348c372, sold to consumer.



Now update the drug state with values as 1,5 by going to the **updateDrugState** task.



Now the Consumer got the drug for consumption.