**Practical No: 09**

**Aim: Create Decentralized Applications in Ethereum.**

**Theory:**

* **Decentralized Applications**
* Decentralized Applications (or DApps) are applications that do not rely on a centralized backend running in AWS or Azure that power traditional web and mobile applications (outside of hosting the frontend code itself). Instead, the application interacts directly with a blockchain which can be thought of distributed cluster of nodes analogous to applications interacting directly with a “masterless” cluster of Cassandra nodes with full replication on every peer in an untrusted peer-to-peer network.
* These blockchain nodes do not require a leader which would defeat the purpose of being truly decentralized. Unlike leader election in various consensus protocols like Raft and Paxos, blockchain transactions are sent to and processed by “random” nodes via Proof of Work or Proof of Stake. These nodes are untrusted nodes running in an arbitrary sized network on various compute devices around the world. Such technology can enable true decentralized ledgers and systems of records.
* DApps are the frontend apps which interact with these blockchain over an API. For Ethereum, this API is a JSON-RPC layer called the Ethereum Web3 API which Moesif supports natively.
* **Advantages**
* Many of the advantages of dApps center around the program's ability to safeguard user privacy. With decentralized apps, users do not need to submit their personal information to use the function the app provides. DApps use smart contracts to complete the transaction between two anonymous parties without the need to rely on a central authority.
* Proponents interested in free speech point out that dApps can be developed as alternative social media platforms. A decentralized social media platform would be resistant to censorship because no single participant on the blockchain can delete messages or block messages from being posted.
* Ethereum is a flexible platform for creating new dApps, providing the infrastructure needed for developers to focus their efforts on finding innovative uses for digital applications. This could enable rapid deployment of dApps in a variety of industries including banking and finance, gaming, social media, and online shopping

1. **Dapps for Voting process.**

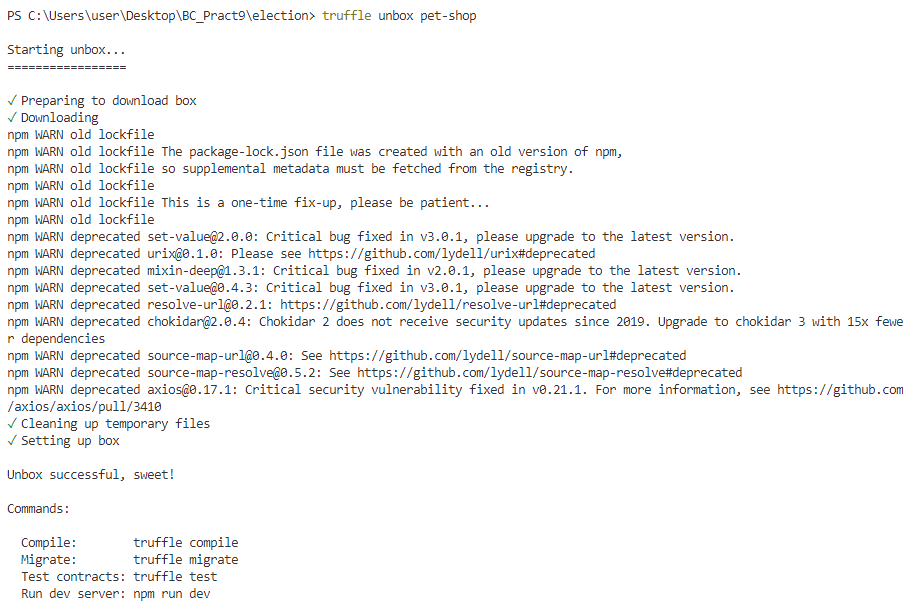
**Create a project directory for our dApp in the command line like this:**

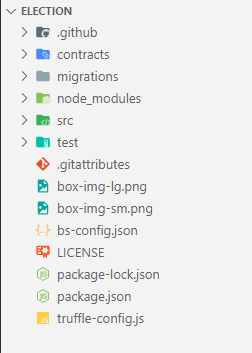
$ mkdir election

$ cd election

**From within your project directory, install the pet shop box from the command line like this:**

$ truffle unbox pet-shop





**Create a new contract file in the contracts directory like this:**

$ touch contracts/Election.sol

**Open the file and start with the following code:**

pragma solidity 0.4.2;

contract Election {

// Read/write candidate

string public candidate;

// Constructor

constructor() public {

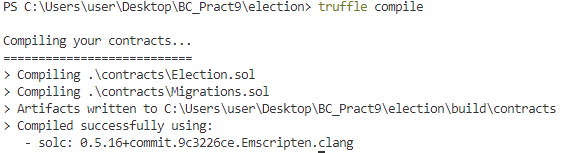
candidate = "Candidate 1";

}

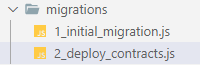
}

**Open Ganache**

$ truffle compile



$ touch migrations/2\_deploy\_contracts.js



**We number all of our files inside the migrations directory with numbers so that Truffle knows which order to execute them in.**

var Election = artifacts.require("./Election.sol");

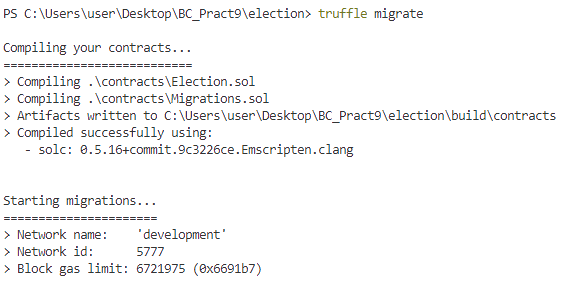
module.exports = function(deployer) {

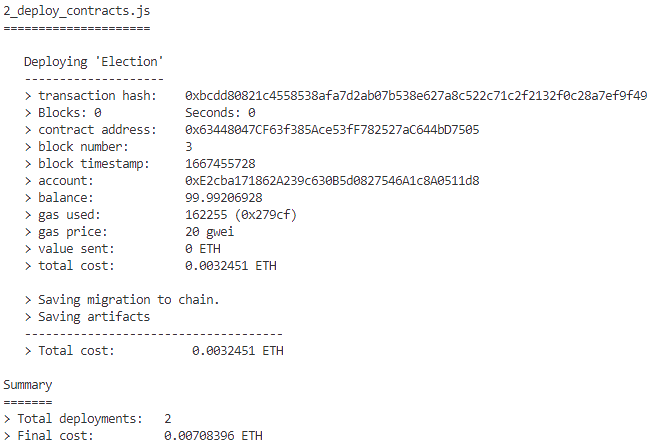
deployer.deploy(Election);

};

**run our migrations from the command line like this:**

$ truffle migrate



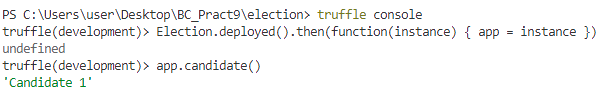


**open the console to interact with the smart contract.**

$ truffle console

**inside the console, let's get an instance of our deployed smart contract and see if we can read the candidate's name from the contract. From the console, run this code:**

Election.deployed().then(**function**(instance) { app = instance })



**Here Election is the name of the variable that we created in the migration file.**

**We retrieved a deployed instance of the contract with the deployed() function, and assigned it to an app variable inside the promise's callback function.**

Replace contracts/Election.sol

pragma solidity >=0.4.2 <=0.8.0;

contract Election {

// Model a Candidate

struct Candidate {

uint id;

string name;

uint voteCount;

}

// Read/write candidates

mapping(uint => Candidate) public candidates;

// Store accounts that have voted

mapping(address => bool) public voters;

// Store Candidates Count

uint public candidatesCount;

constructor() public {

addCandidate("Candidate 1");

addCandidate("Candidate 2");

}

event votedEvent(uint indexed \_candidateId);

function addCandidate (string memory \_name) private {

candidatesCount ++;

candidates[candidatesCount] = Candidate(candidatesCount, \_name, 0);

}

function vote (uint \_candidateId) public {

// require that they haven't voted before

require(!voters[msg.sender]);

// require a valid candidate

require(\_candidateId > 0 && \_candidateId <= candidatesCount);

// record that voter has voted

voters[msg.sender] = true;

// update candidate vote Count

candidates[\_candidateId].voteCount ++;

// trigger voted event

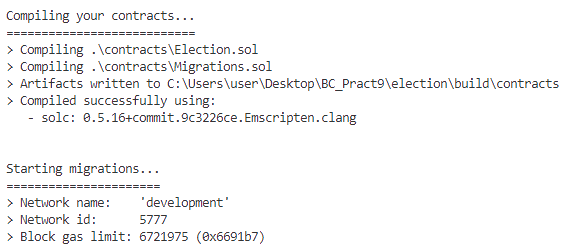
emit votedEvent(\_candidateId);

}

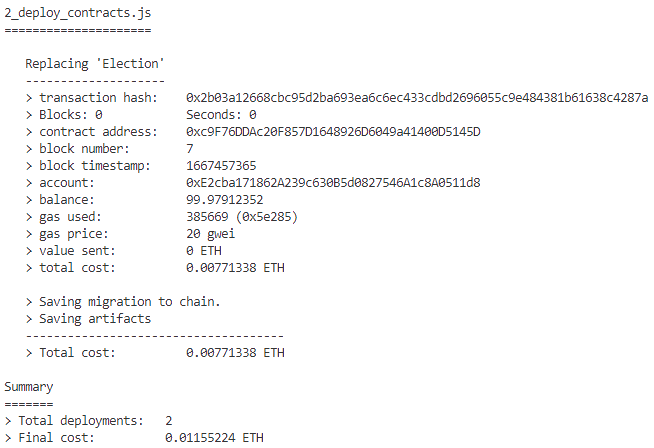
}

$ truffle compile

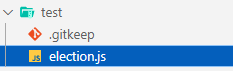
$ truffle migrate --reset







**Testing smart contract**



***test/election.js***

var Election = artifacts.require("./Election.sol");

contract("Election", function(accounts) {

var electionInstance;

it("initializes with two candidates", function() {

return Election.deployed().then(function(instance) {

return instance.candidatesCount();

}).then(function(count) {

assert.equal(count, 2);

});

});

it("it initializes the candidates with the correct values", function() {

return Election.deployed().then(function(instance) {

electionInstance = instance;

return electionInstance.candidates(1);

}).then(function(candidate) {

assert.equal(candidate[0], 1, "contains the correct id");

assert.equal(candidate[1], "Candidate 1", "contains the correct name");

assert.equal(candidate[2], 0, "contains the correct votes count");

return electionInstance.candidates(2);

}).then(function(candidate) {

assert.equal(candidate[0], 2, "contains the correct id");

assert.equal(candidate[1], "Candidate 2", "contains the correct name");

assert.equal(candidate[2], 0, "contains the correct votes count");

});

});

it("allows a voter to cast a vote", function() {

return Election.deployed().then(function(instance) {

electionInstance = instance;

candidateId = 1;

return electionInstance.vote(candidateId, { from: accounts[0] });

}).then(function(receipt) {

assert.equal(receipt.logs.length, 1, "an event was triggered");

assert.equal(receipt.logs[0].event, "votedEvent", "the event type is correct");

assert.equal(receipt.logs[0].args.\_candidateId.toNumber(), candidateId, "the candidate id is correct");

return electionInstance.voters(accounts[0]);

}).then(function(voted) {

assert(voted, "the voter was marked as voted");

return electionInstance.candidates(candidateId);

}).then(function(candidate) {

var voteCount = candidate[2];

assert.equal(voteCount, 1, "increments the candidate's vote count");

})

});

it("throws an exception for invalid candiates", function() {

return Election.deployed().then(function(instance) {

electionInstance = instance;

return electionInstance.vote(99, { from: accounts[1] })

}).then(assert.fail).catch(function(error) {

assert(error.message.indexOf('revert') >= 0, "error message must contain revert");

return electionInstance.candidates(1);

}).then(function(candidate1) {

var voteCount = candidate1[2];

assert.equal(voteCount, 1, "candidate 1 did not receive any votes");

return electionInstance.candidates(2);

}).then(function(candidate2) {

var voteCount = candidate2[2];

assert.equal(voteCount, 0, "candidate 2 did not receive any votes");

});

});

it("throws an exception for double voting", function() {

return Election.deployed().then(function(instance) {

electionInstance = instance;

candidateId = 2;

electionInstance.vote(candidateId, { from: accounts[1] });

return electionInstance.candidates(candidateId);

}).then(function(candidate) {

var voteCount = candidate[2];

assert.equal(voteCount, 1, "accepts first vote");

// Try to vote again

return electionInstance.vote(candidateId, { from: accounts[1] });

}).then(assert.fail).catch(function(error) {

assert(error.message.indexOf('revert') >= 0, "error message must contain revert");

return electionInstance.candidates(1);

}).then(function(candidate1) {

var voteCount = candidate1[2];

assert.equal(voteCount, 1, "candidate 1 did not receive any votes");

return electionInstance.candidates(2);

}).then(function(candidate2) {

var voteCount = candidate2[2];

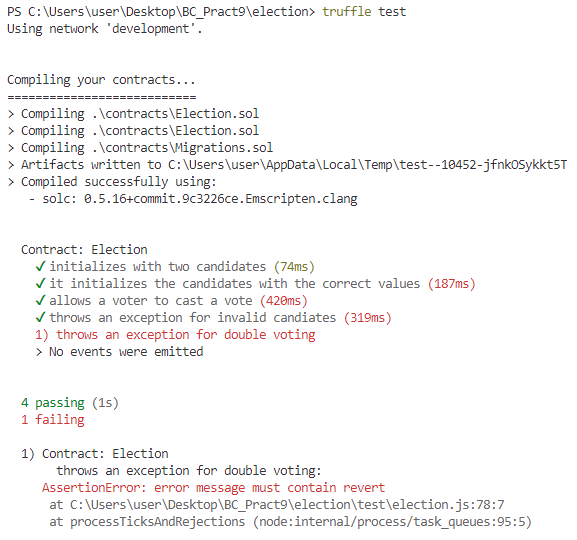
assert.equal(voteCount, 1, "candidate 2 did not receive any votes");

});

});

});

**Run the tests from the command line like this:**

$ truffle test

$ truffle migrate --reset

**Copy this to election/src/js/app.js**

App = {

web3Provider: null,

contracts: {},

account: '0x0',

hasVoted: false,

init: function() {

return App.initWeb3();

},

initWeb3: function() {

// TODO: refactor conditional

if (typeof web3 !== 'undefined') {

// If a web3 instance is already provided by Meta Mask.

App.web3Provider = web3.currentProvider;

web3 = new Web3(web3.currentProvider);

} else {

// Specify default instance if no web3 instance provided

App.web3Provider = new Web3.providers.HttpProvider('http://localhost:7545');

web3 = new Web3(App.web3Provider);

}

return App.initContract();

},

initContract: function() {

$.getJSON("Election.json", function(election) {

// Instantiate a new truffle contract from the artifact

App.contracts.Election = TruffleContract(election);

// Connect provider to interact with contract

App.contracts.Election.setProvider(App.web3Provider);

App.listenForEvents();

return App.render();

});

},

// Listen for events emitted from the contract

listenForEvents: function() {

App.contracts.Election.deployed().then(function(instance) {

// Restart Chrome if you are unable to receive this event

// This is a known issue with Metamask

// https://github.com/MetaMask/metamask-extension/issues/2393

instance.votedEvent({}, {

fromBlock: 0,

toBlock: 'latest'

}).watch(function(error, event) {

console.log("event triggered", event)

// Reload when a new vote is recorded

App.render();

});

});

},

render: function() {

var electionInstance;

var loader = $("#loader");

var content = $("#content");

loader.show();

content.hide();

// Load account data

web3.eth.getCoinbase(function(err, account) {

if (err === null) {

App.account = account;

$("#accountAddress").html("Your Account: " + account);

}

});

// Load contract data

App.contracts.Election.deployed().then(function(instance) {

electionInstance = instance;

return electionInstance.candidatesCount();

}).then(function(candidatesCount) {

var candidatesResults = $("#candidatesResults");

candidatesResults.empty();

var candidatesSelect = $('#candidatesSelect');

candidatesSelect.empty();

for (var i = 1; i <= candidatesCount; i++) {

electionInstance.candidates(i).then(function(candidate) {

var id = candidate[0];

var name = candidate[1];

var voteCount = candidate[2];

// Render candidate Result

var candidateTemplate = "<tr><th>" + id + "</th><td>" + name + "</td><td>" + voteCount + "</td></tr>"

candidatesResults.append(candidateTemplate);

// Render candidate ballot option

var candidateOption = "<option value='" + id + "' >" + name + "</ option>"

candidatesSelect.append(candidateOption);

});

}

return electionInstance.voters(App.account);

}).then(function(hasVoted) {

// Do not allow a user to vote

if(hasVoted) {

$('form').hide();

}

loader.hide();

content.show();

}).catch(function(error) {

console.warn(error);

});

},

castVote: function() {

var candidateId = $('#candidatesSelect').val();

App.contracts.Election.deployed().then(function(instance) {

return instance.vote(candidateId, { from: App.account });

}).then(function(result) {

// Wait for votes to update

$("#content").hide();

$("#loader").show();

}).catch(function(err) {

console.error(err);

});

}

};

$(function() {

$(window).load(function() {

App.init();

});

});

Copy this to election/src/index.html

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="utf-8">

<meta http-equiv="X-UA-Compatible" content="IE=edge">

<meta name="viewport" content="width=device-width, initial-scale=1">

<!-- The above 3 meta tags \*must\* come first in the head; any other head content must come \*after\* these tags -->

<title>Election Results</title>

<!-- Bootstrap -->

<link href="css/bootstrap.min.css" rel="stylesheet">

<!-- HTML5 shim and Respond.js for IE8 support of HTML5 elements and media queries -->

<!-- WARNING: Respond.js doesn't work if you view the page via file:// -->

<!--[if lt IE 9]>

<script src="https://oss.maxcdn.com/html5shiv/3.7.3/html5shiv.min.js"></script>

<script src="https://oss.maxcdn.com/respond/1.4.2/respond.min.js"></script>

<![endif]-->

</head>

<body>

<div class="container" style="width: 650px;">

<div class="row">

<div class="col-lg-12">

<h1 class="text-center">Election Results</h1>

<hr/>

<br/>

<div id="loader">

<p class="text-center">Loading...</p>

</div>

<div id="content" style="display: none;">

<table class="table">

<thead>

<tr>

<th scope="col">#</th>

<th scope="col">Name</th>

<th scope="col">Votes</th>

</tr>

</thead>

<tbody id="candidatesResults">

</tbody>

</table>

<hr/>

<form onSubmit="App.castVote(); return false;">

<div class="form-group">

<label for="candidatesSelect">Select Candidate</label>

<select class="form-control" id="candidatesSelect">

</select>

</div>

<button type="submit" class="btn btn-primary">Vote</button>

<hr />

</form>

<p id="accountAddress" class="text-center"></p>

</div>

</div>

</div>

</div>

<!-- jQuery (necessary for Bootstrap's JavaScript plugins) -->

<script src="https://ajax.googleapis.com/ajax/libs/jquery/1.12.4/jquery.min.js"></script>

<!-- Include all compiled plugins (below), or include individual files as needed -->

<script src="js/bootstrap.min.js"></script>

<script src="js/web3.min.js"></script>

<script src="js/truffle-contract.js"></script>

<script src="js/app.js"></script>

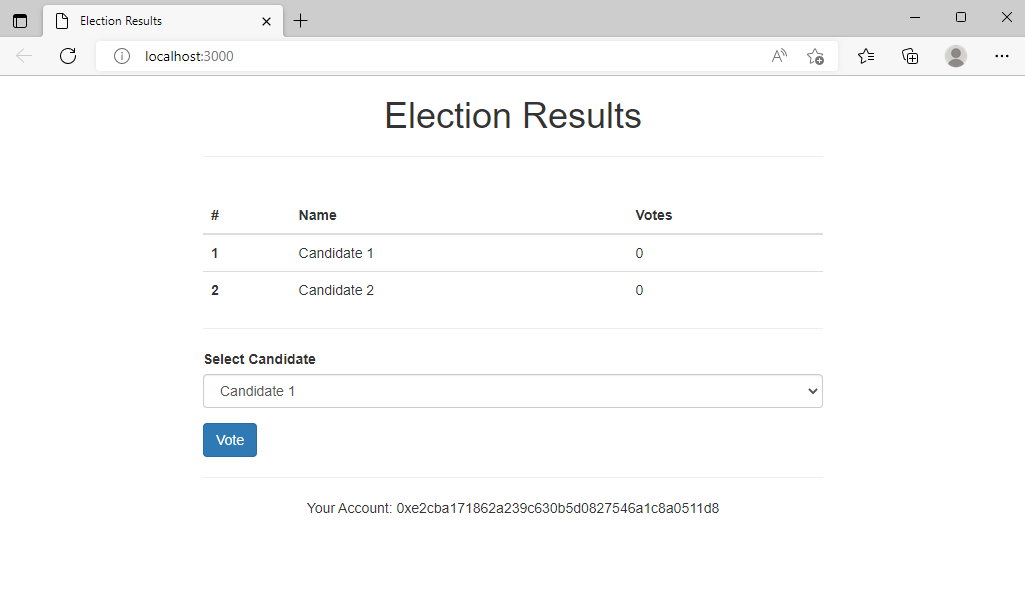
</body>

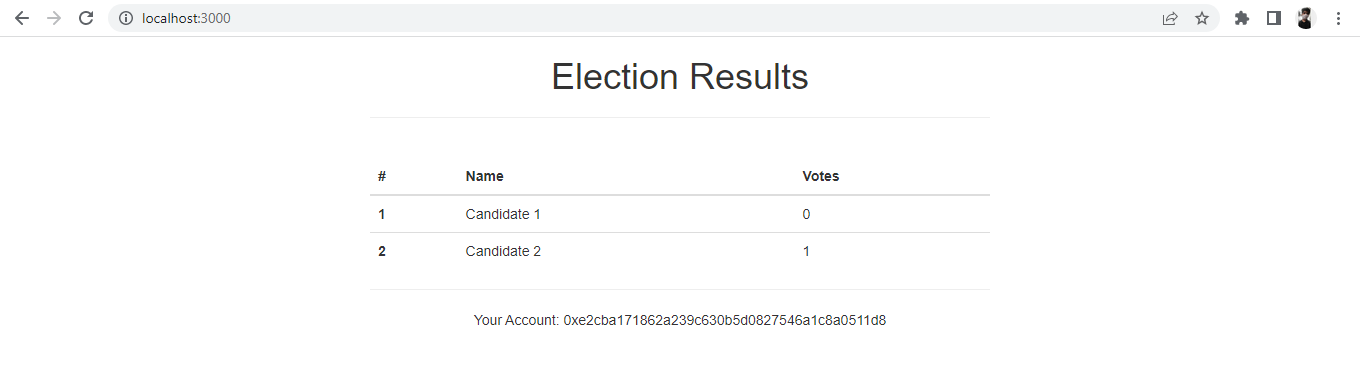
</html>

$ npm run dev



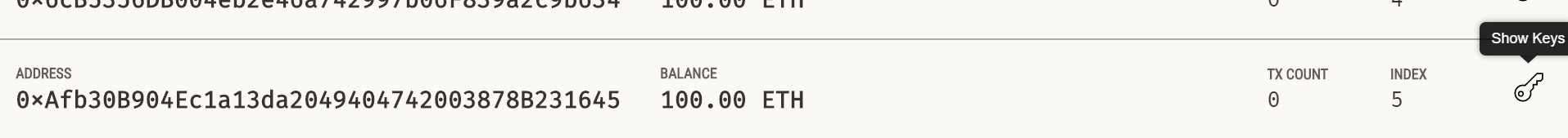
Select the candidate and click on **Vote**



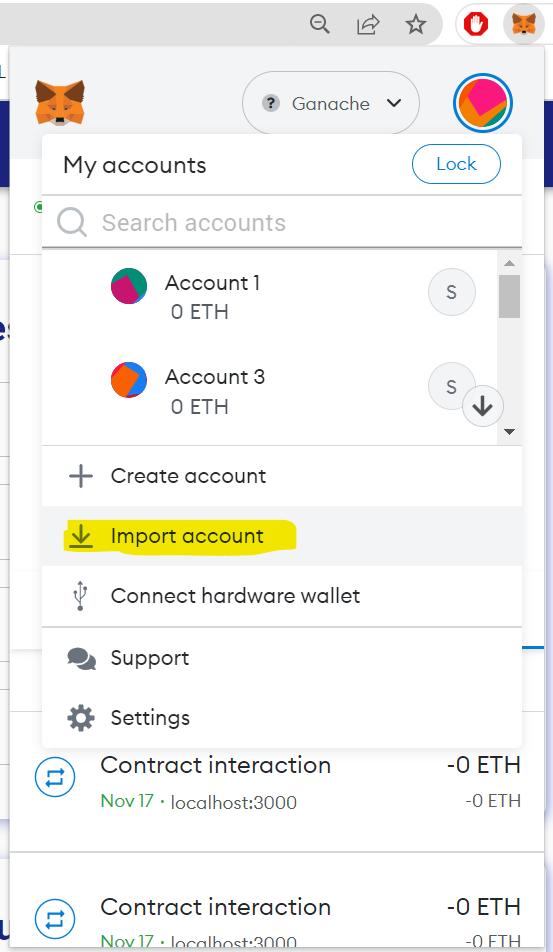
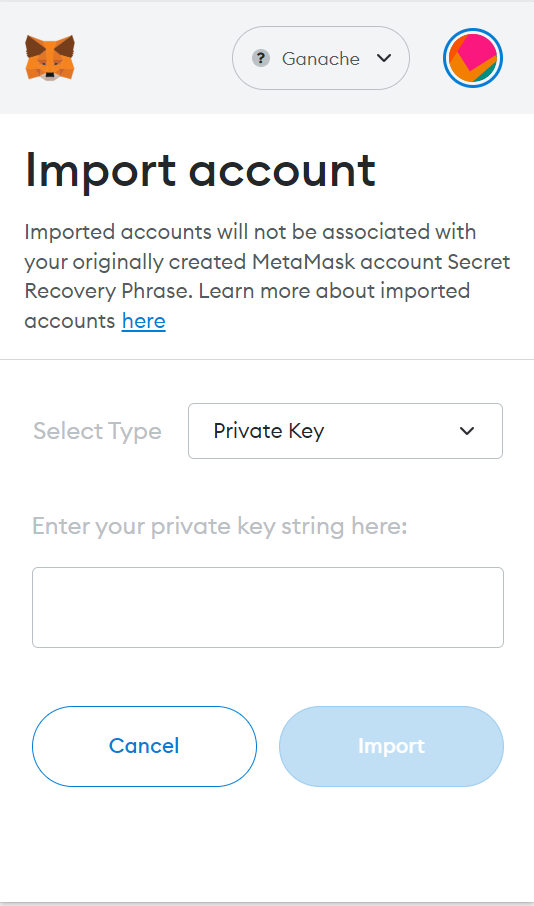


**2. Dapps for Supply chain**

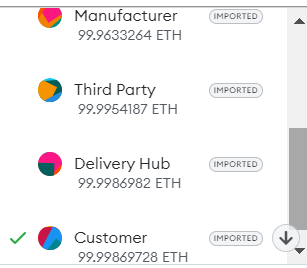
**Go to ganache-> Copy the private key from the addresses**

****

**Go to meta mask-> Import Account-> Paste the copied private key.**

** **

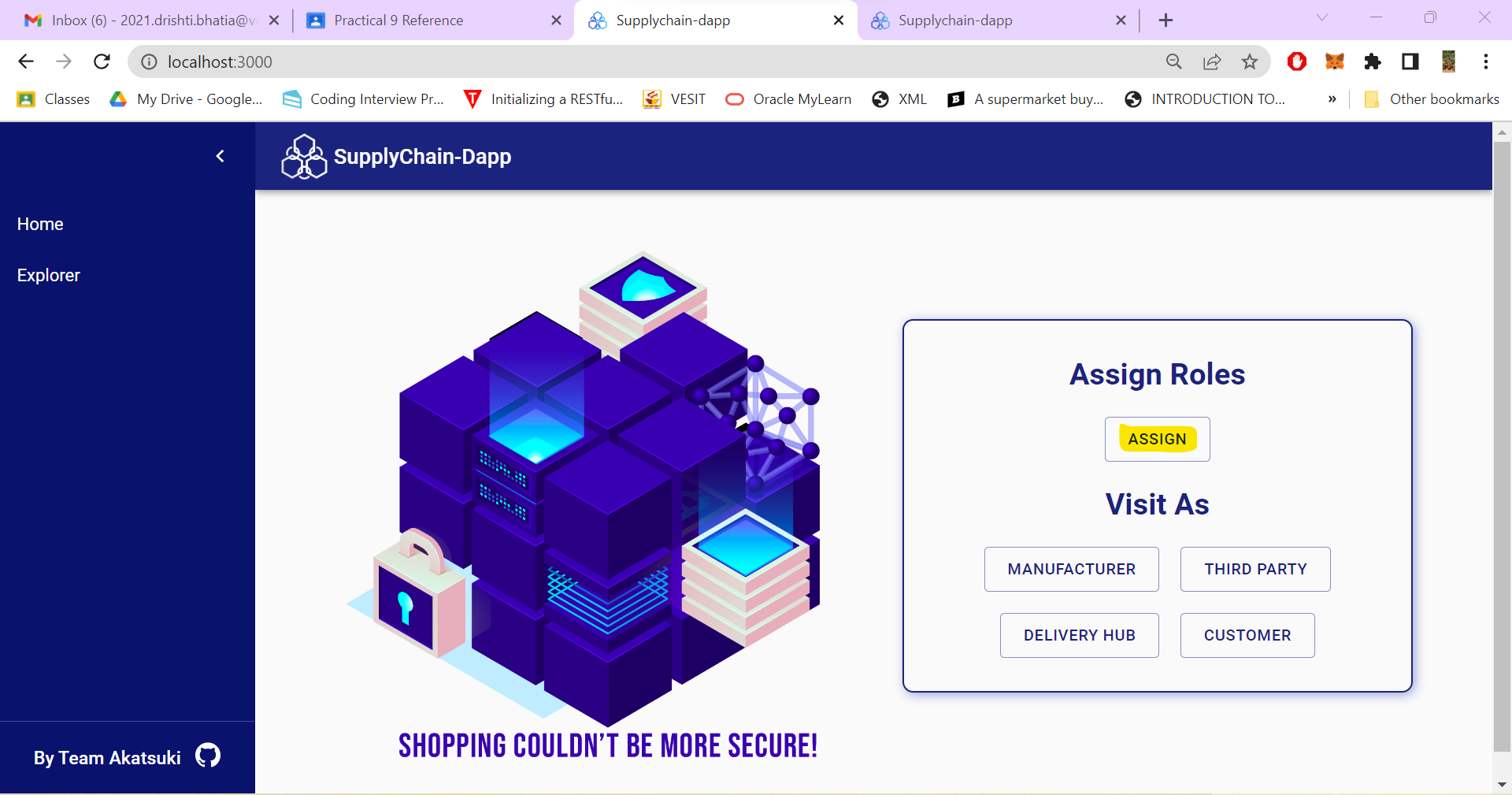
**Import 4 such unique accounts and name them respectively**

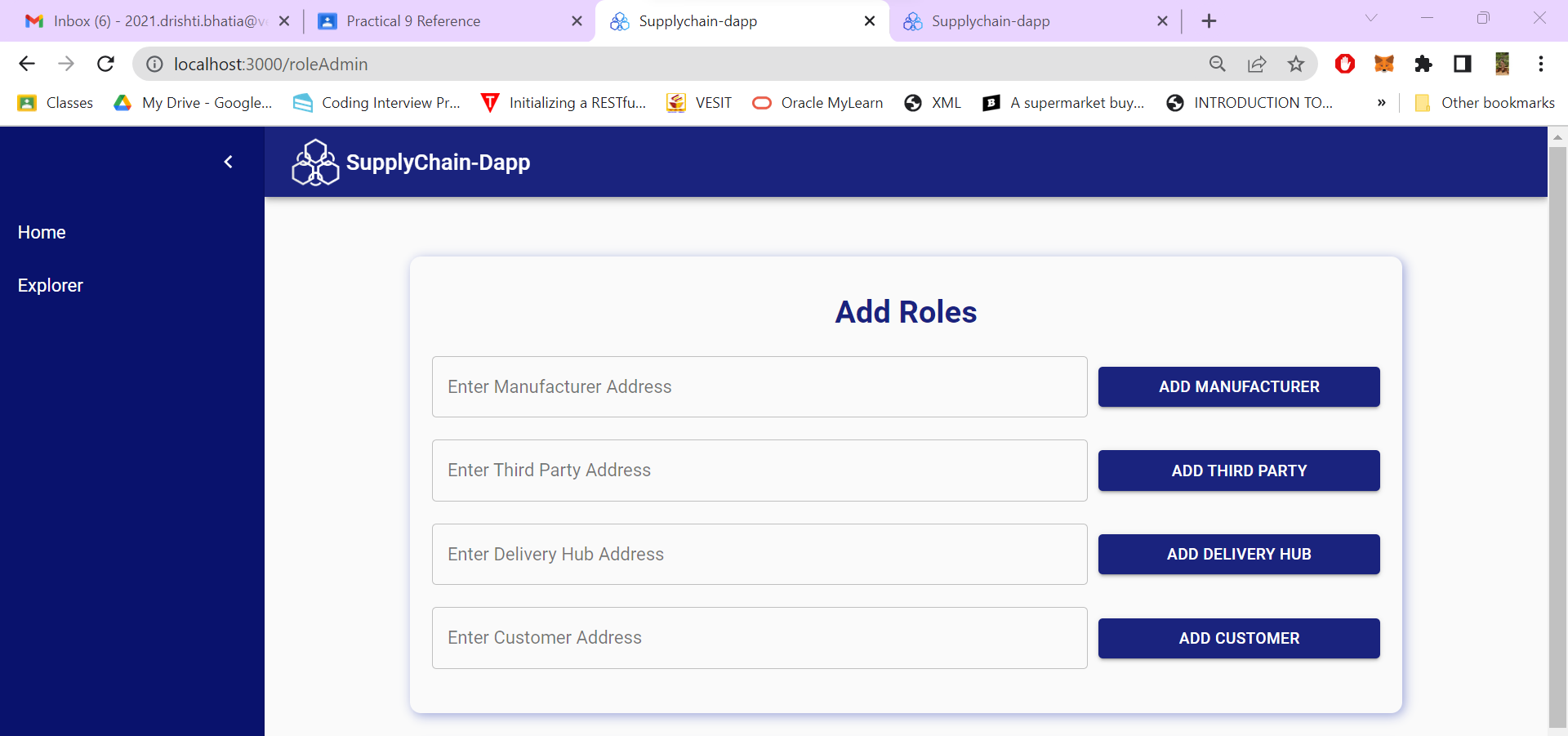
****

**In the terminal:**

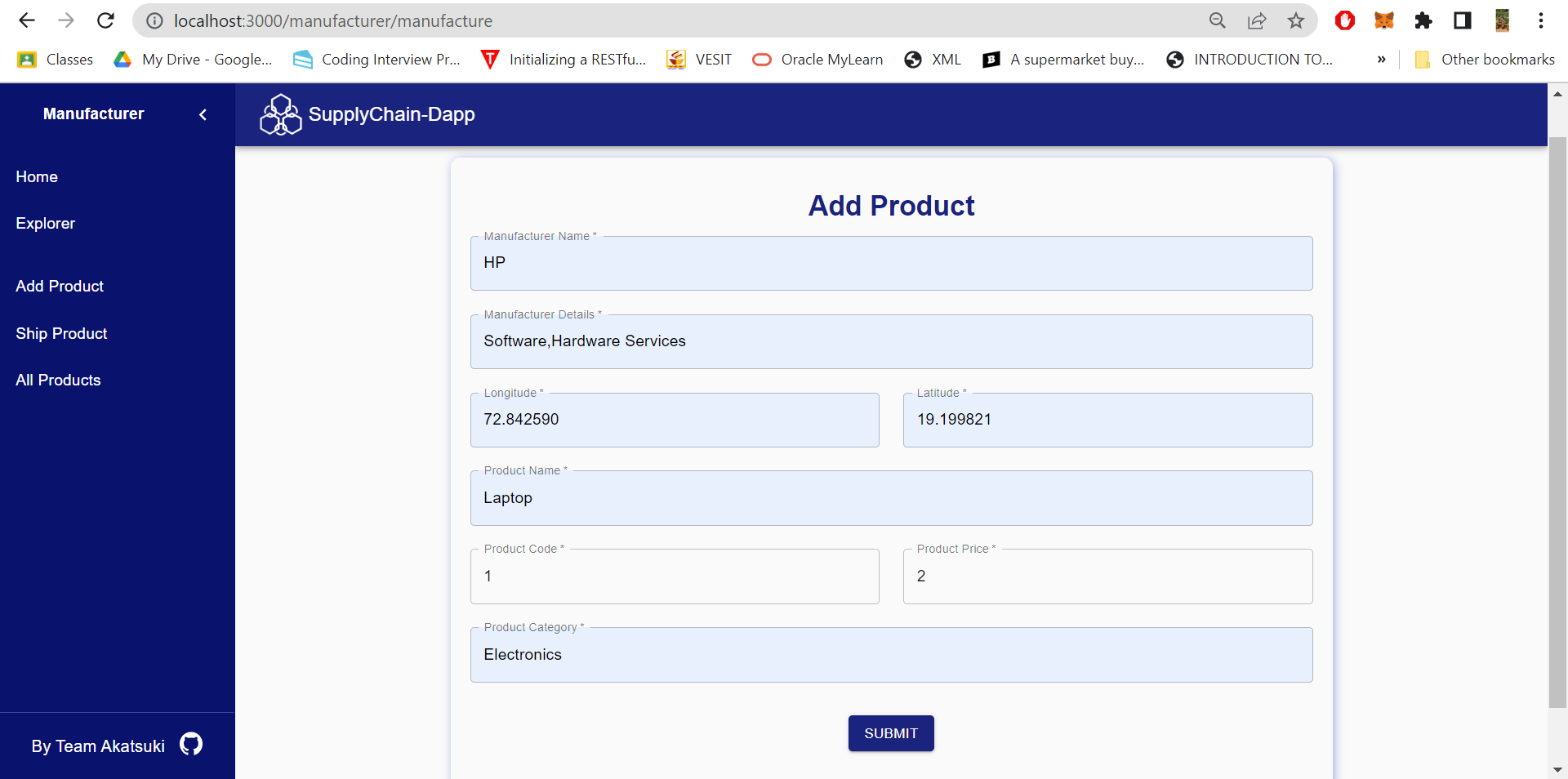
1. **cd client**
2. **npm start**

**Go to Assign and Assign the private keys to respective account names in the main page**

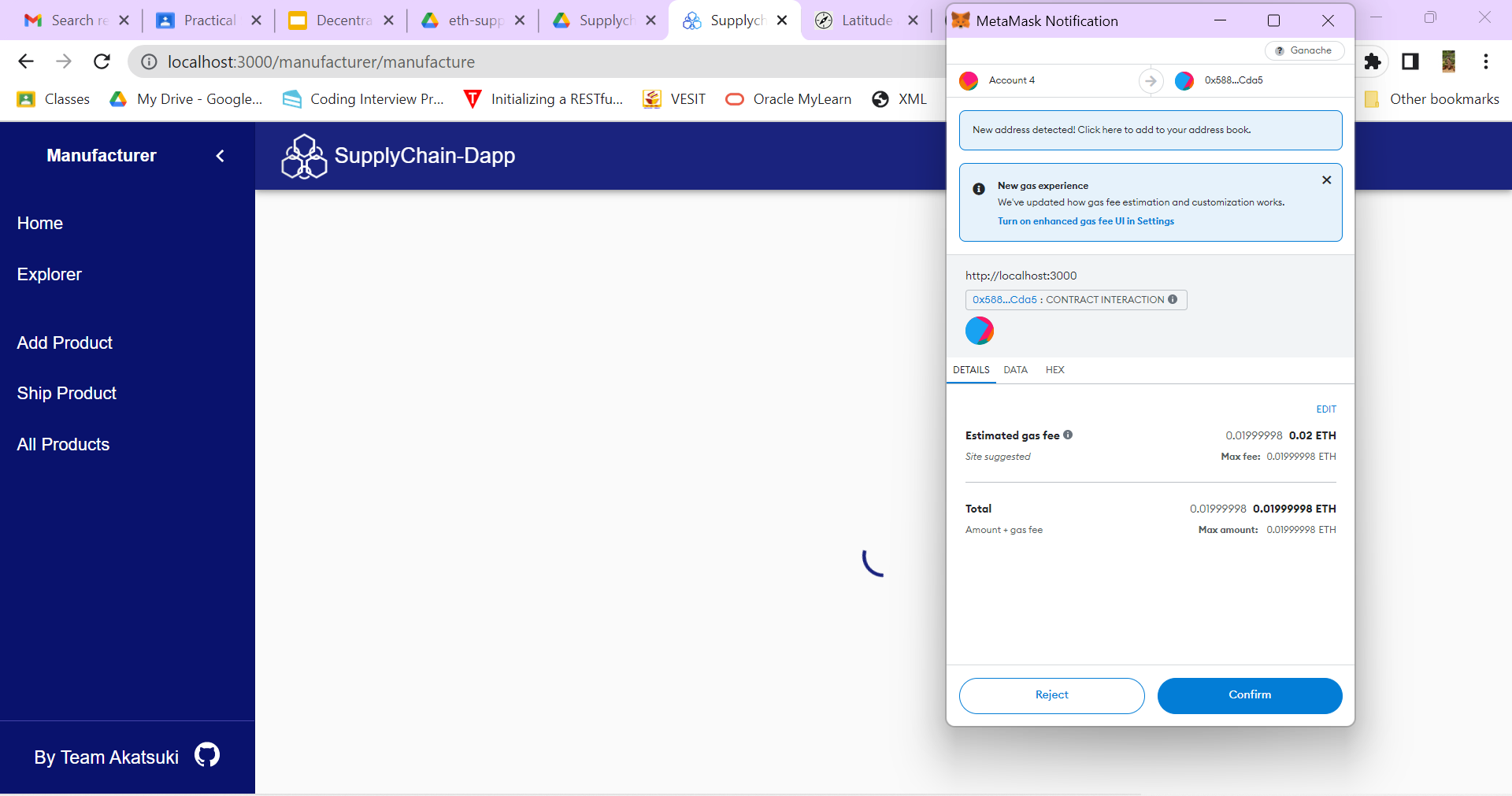
****

****

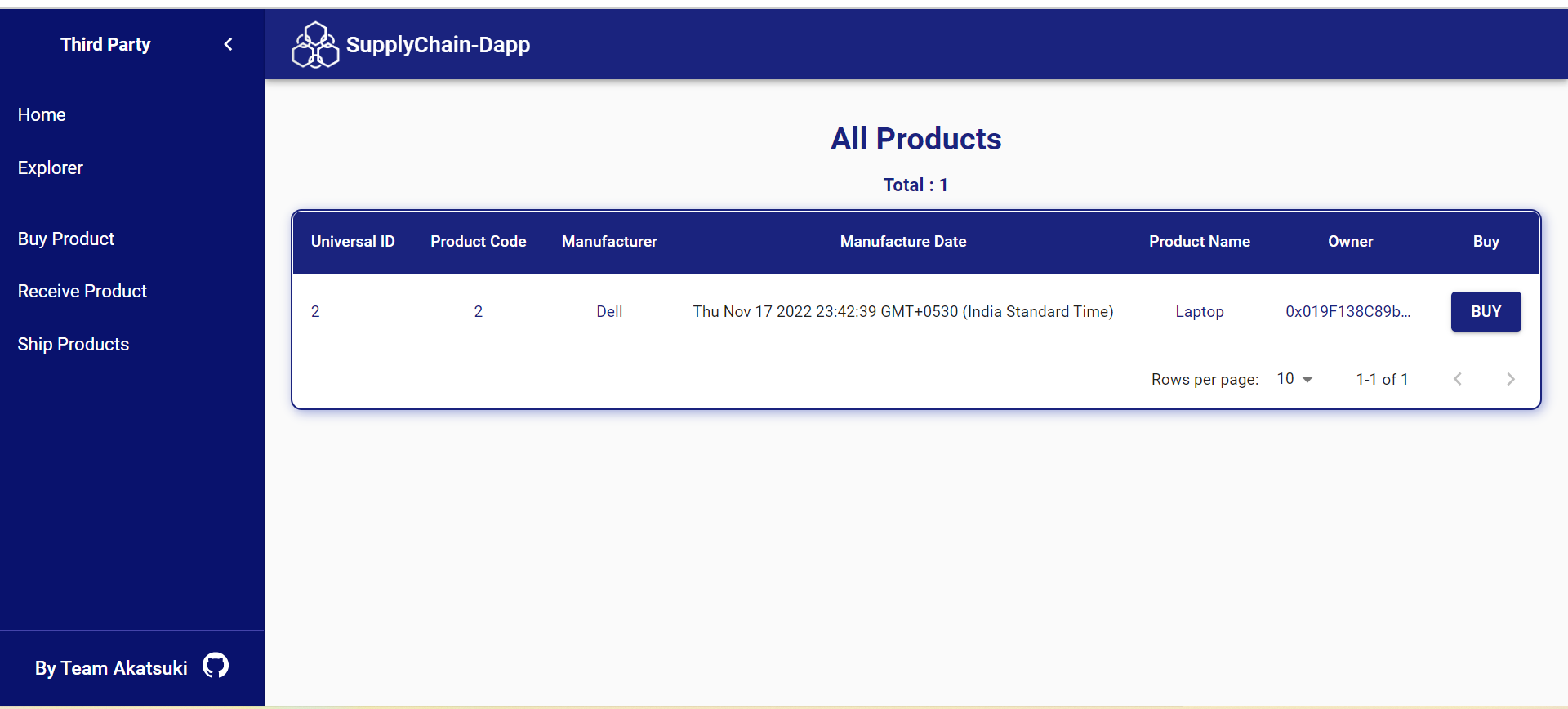
**Life cycle Step 1: Go to manufacturer and add the details *(Select the Manufacturer’s meta mask account while doing so)***

****

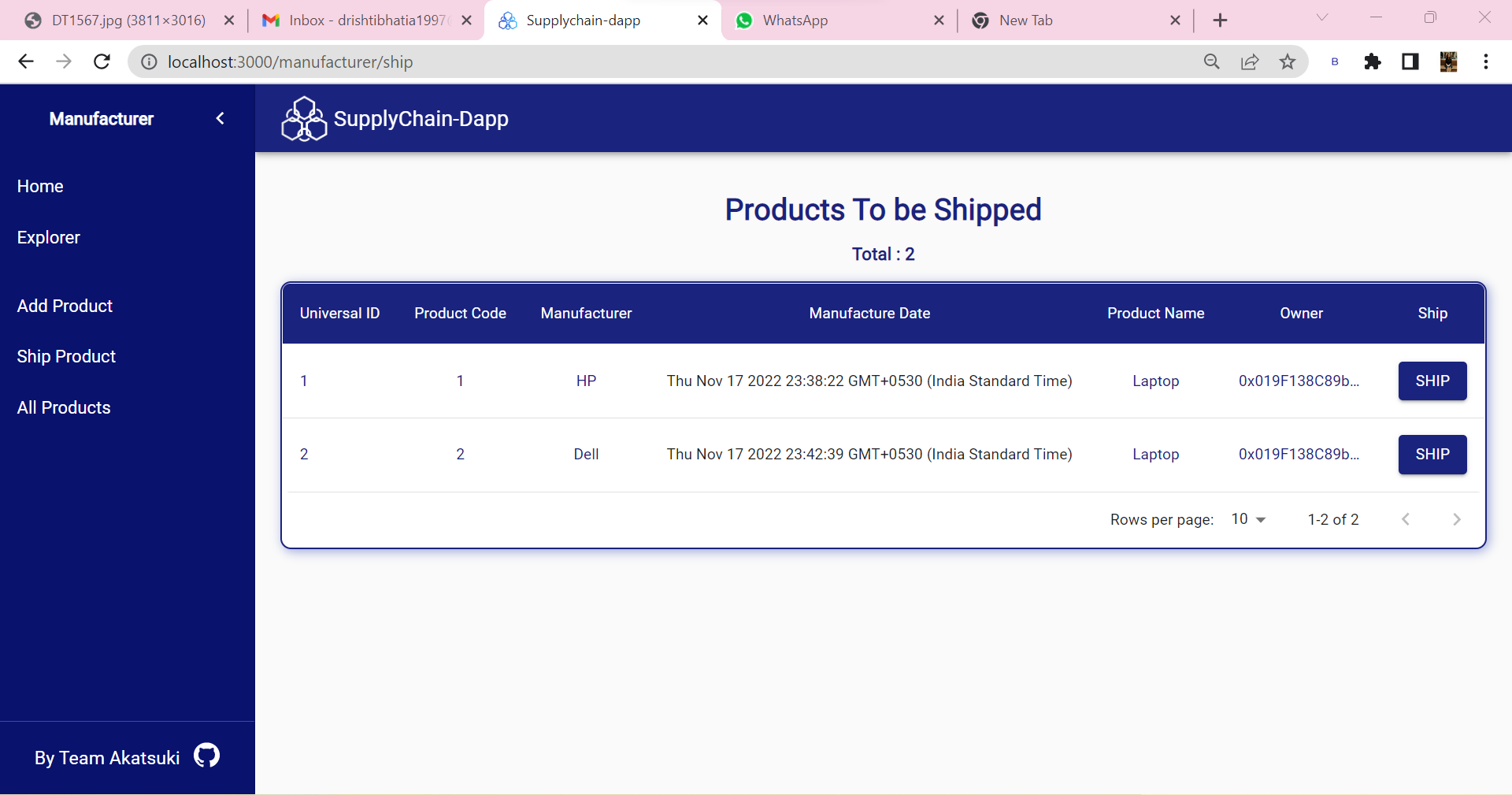
**Once you click on submit you will get a confirmation, please click on confirm**

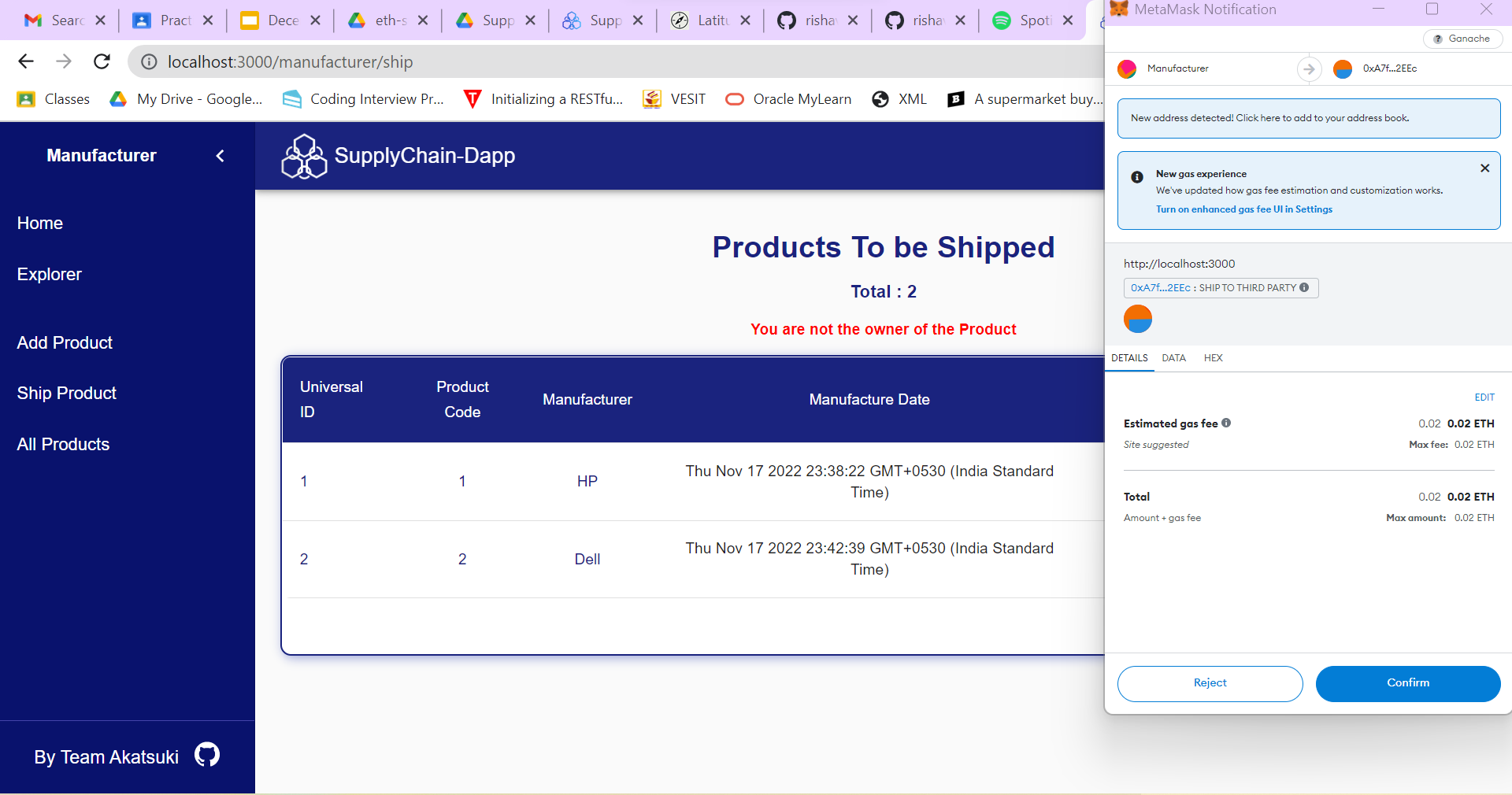
****

**Life Cycle Step 2: Buy (Change metamask account to Third Part)**

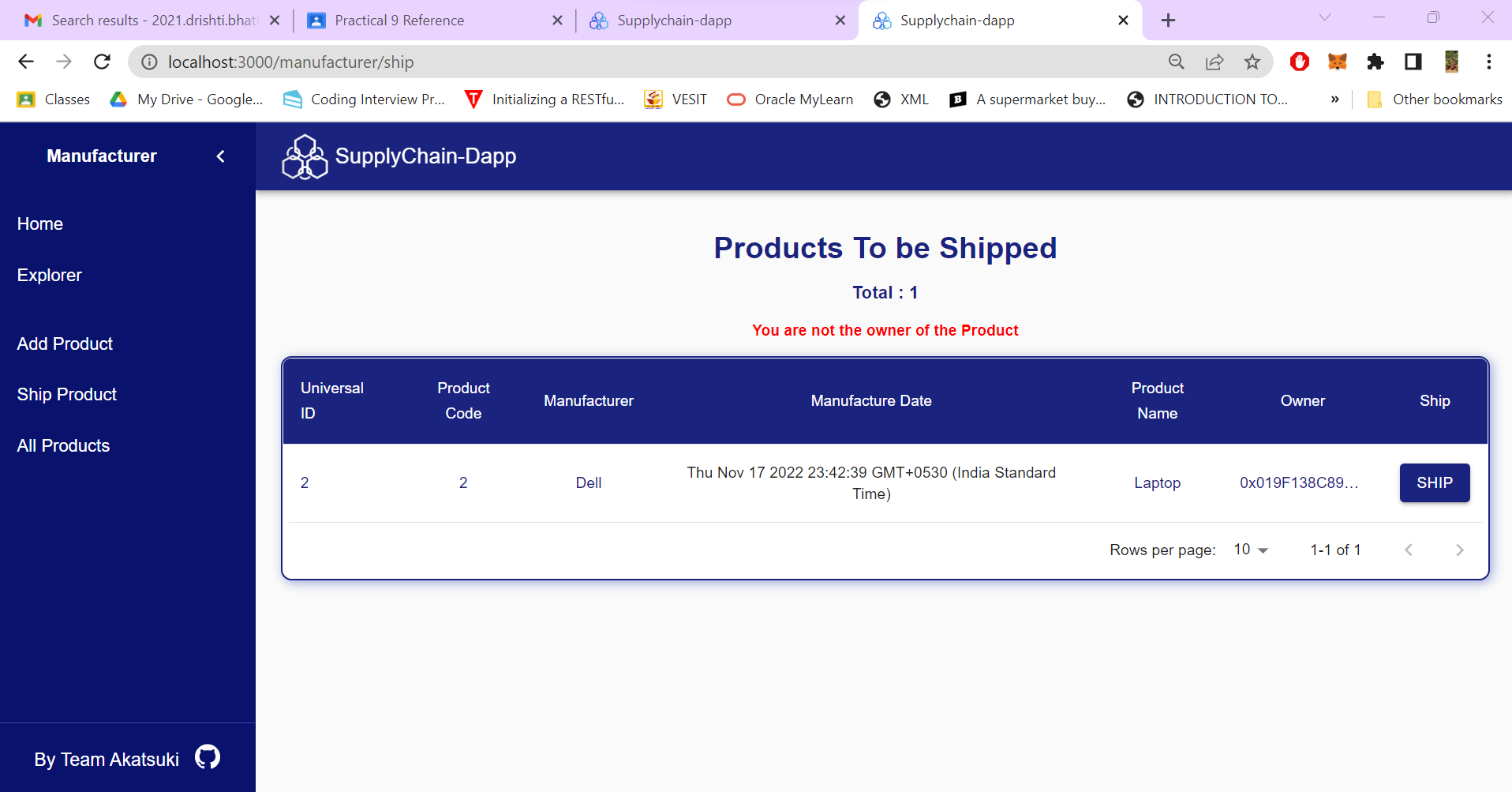
****

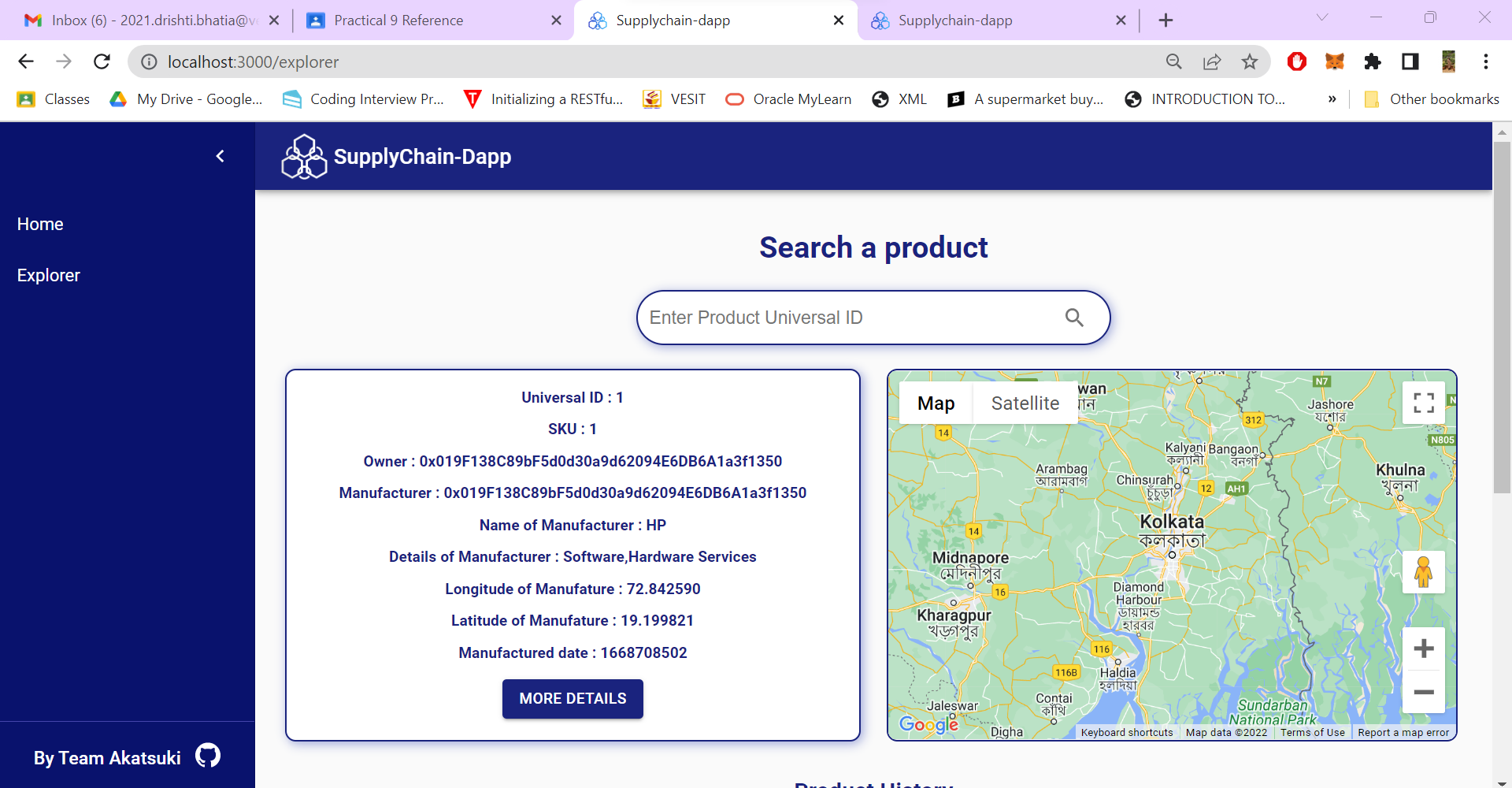
**Life Cycle Step 3: Shipping (Change Account to Manufacturer):**

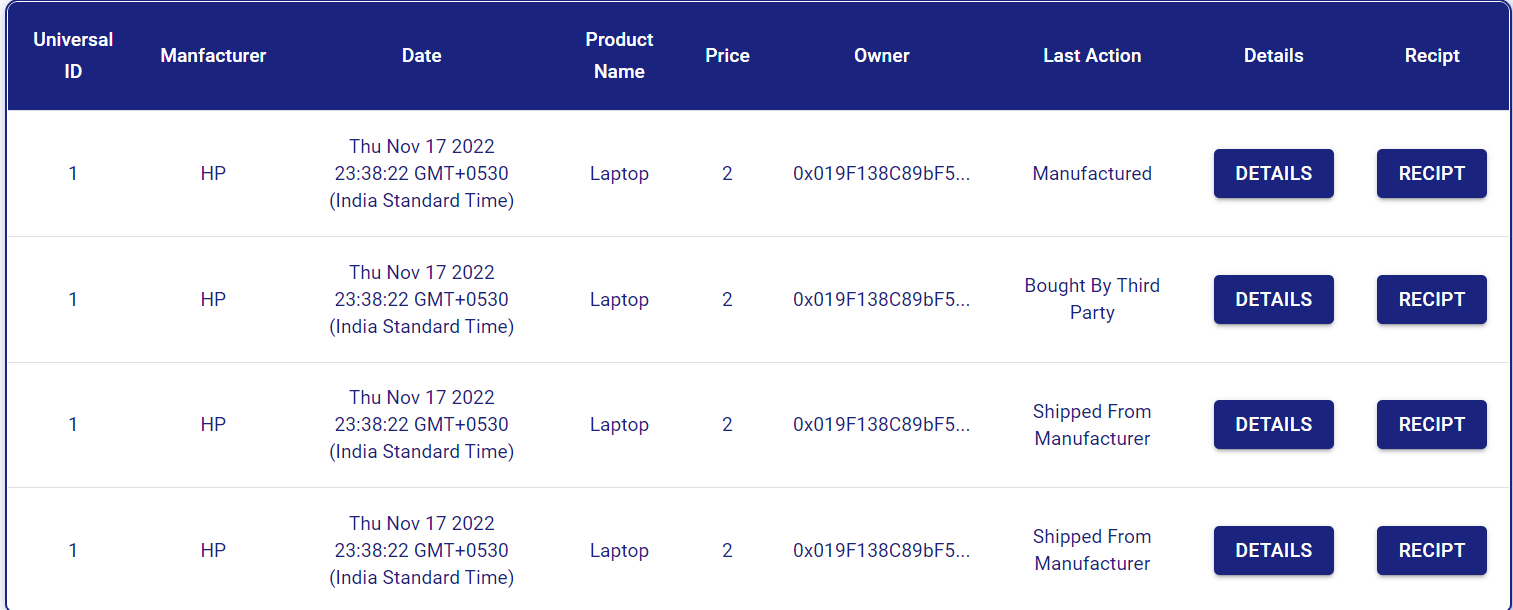
****

****

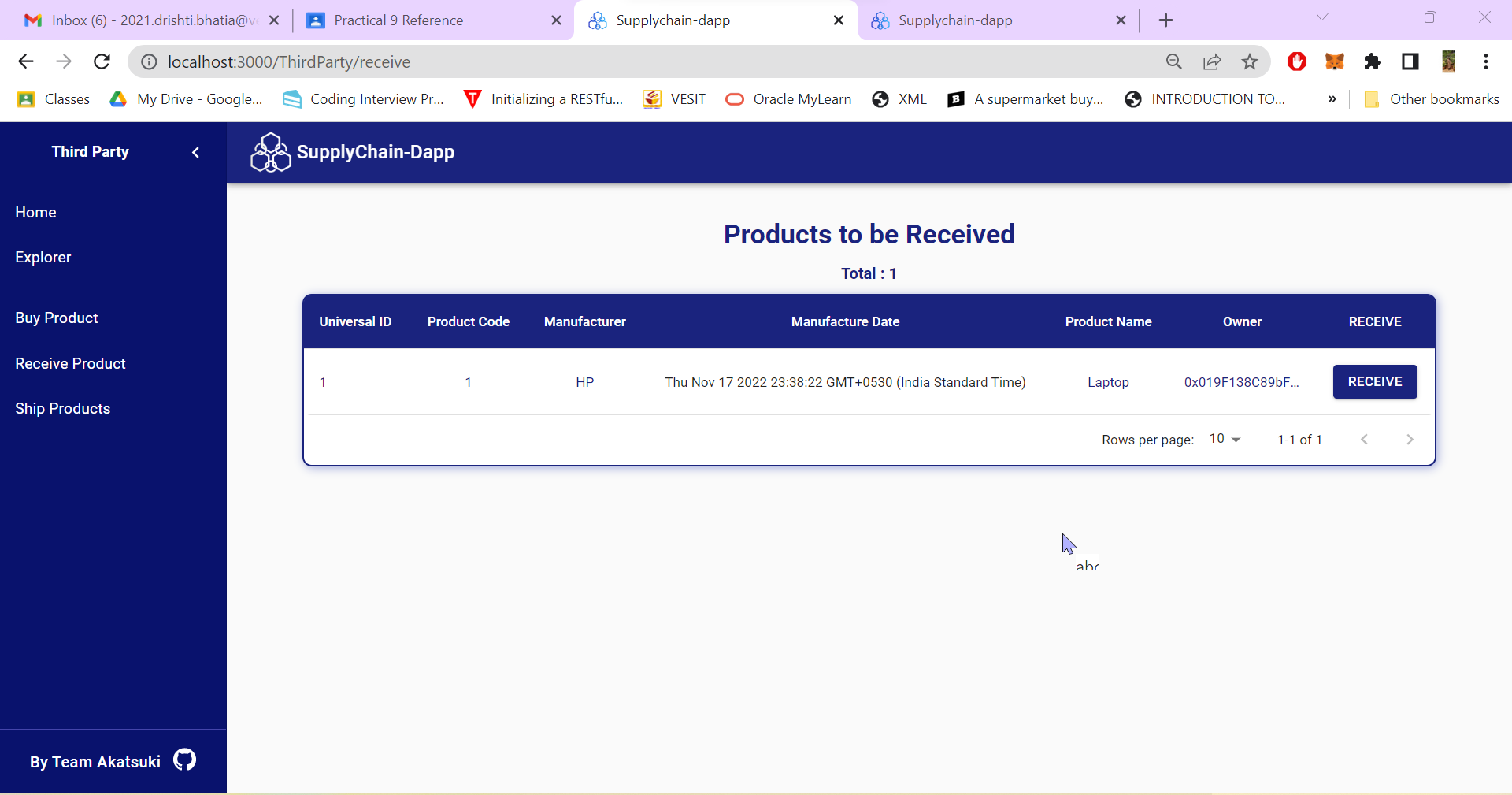
**After Confirming:**

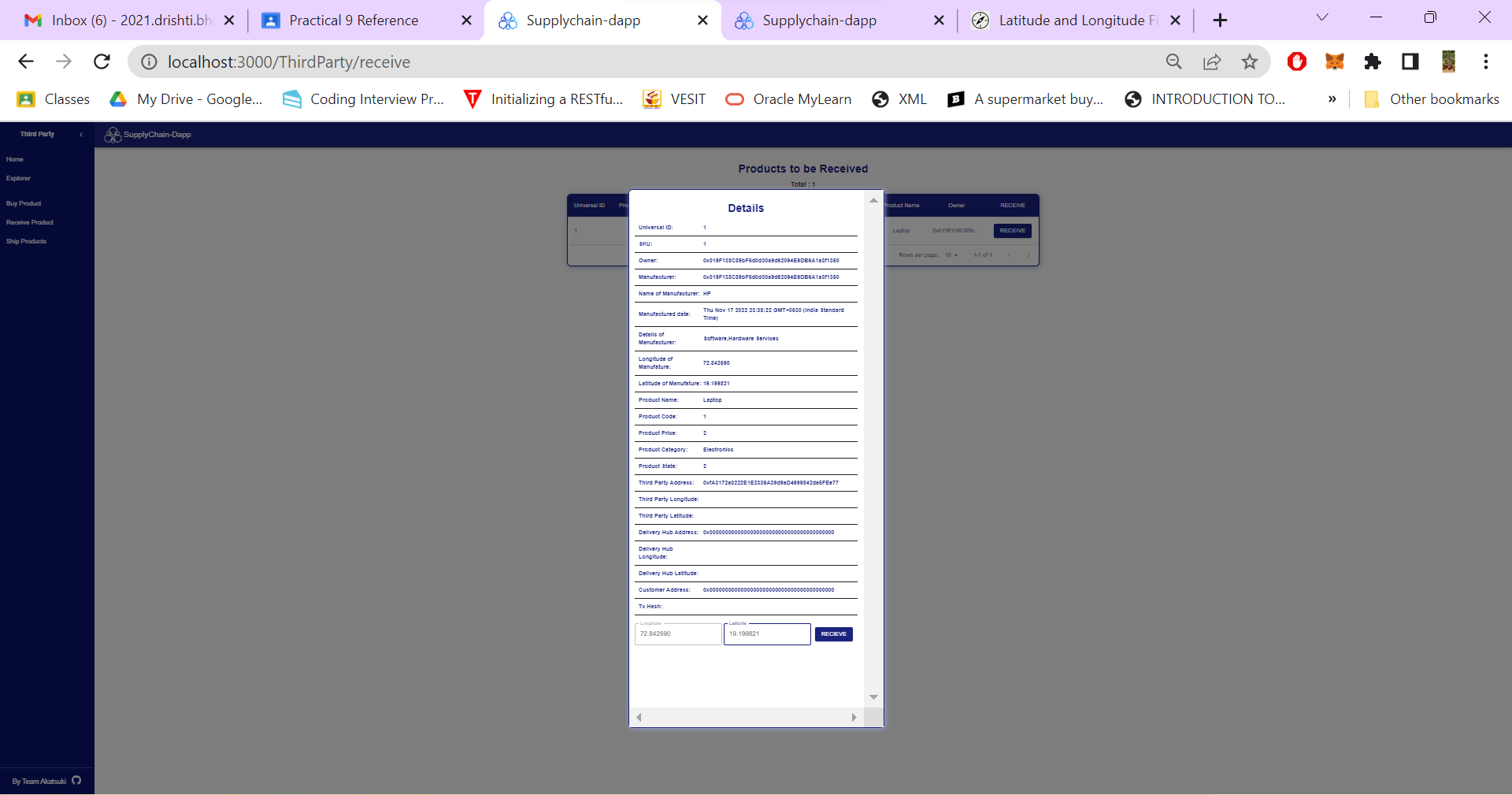
****

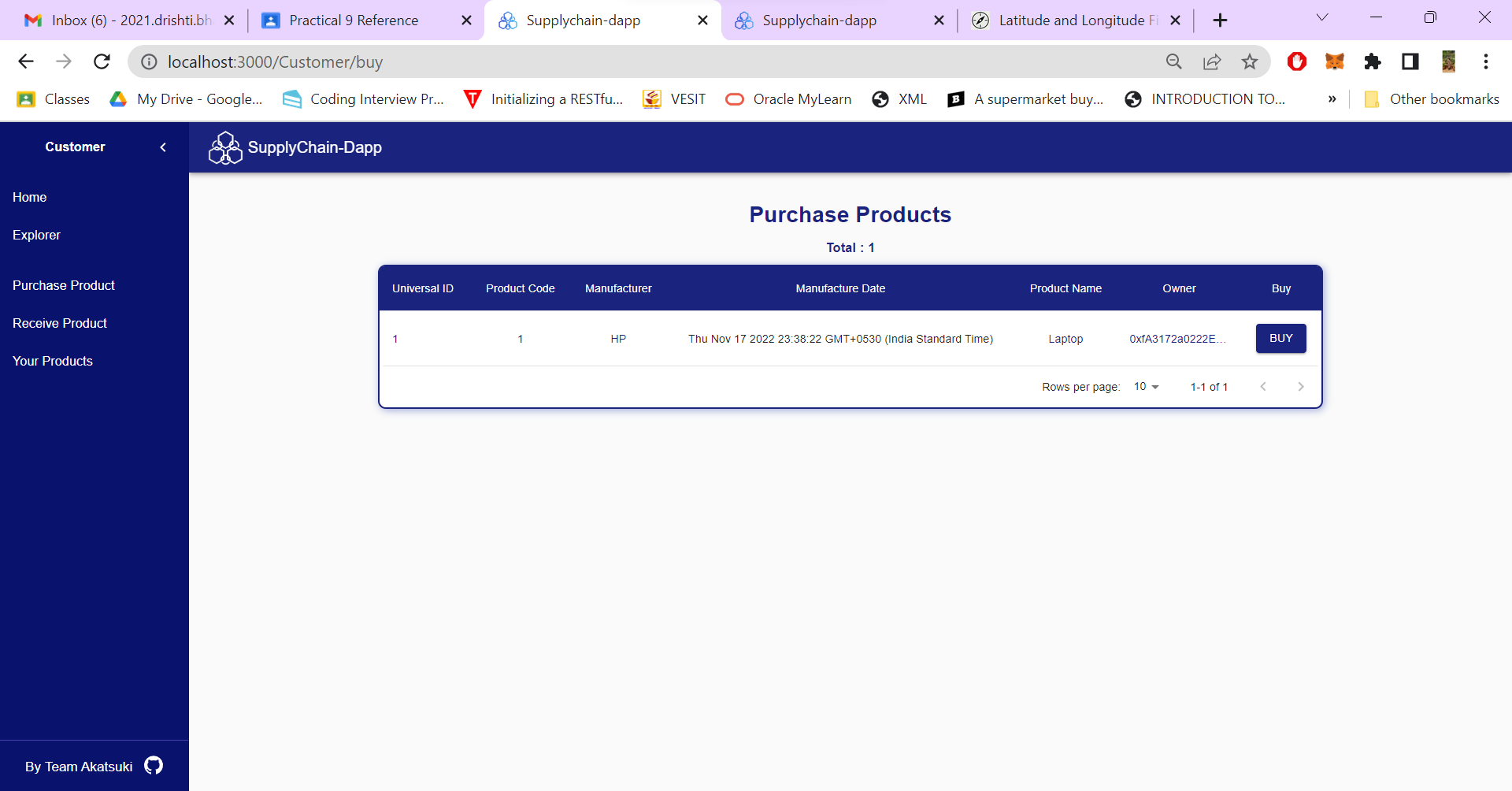
****

****

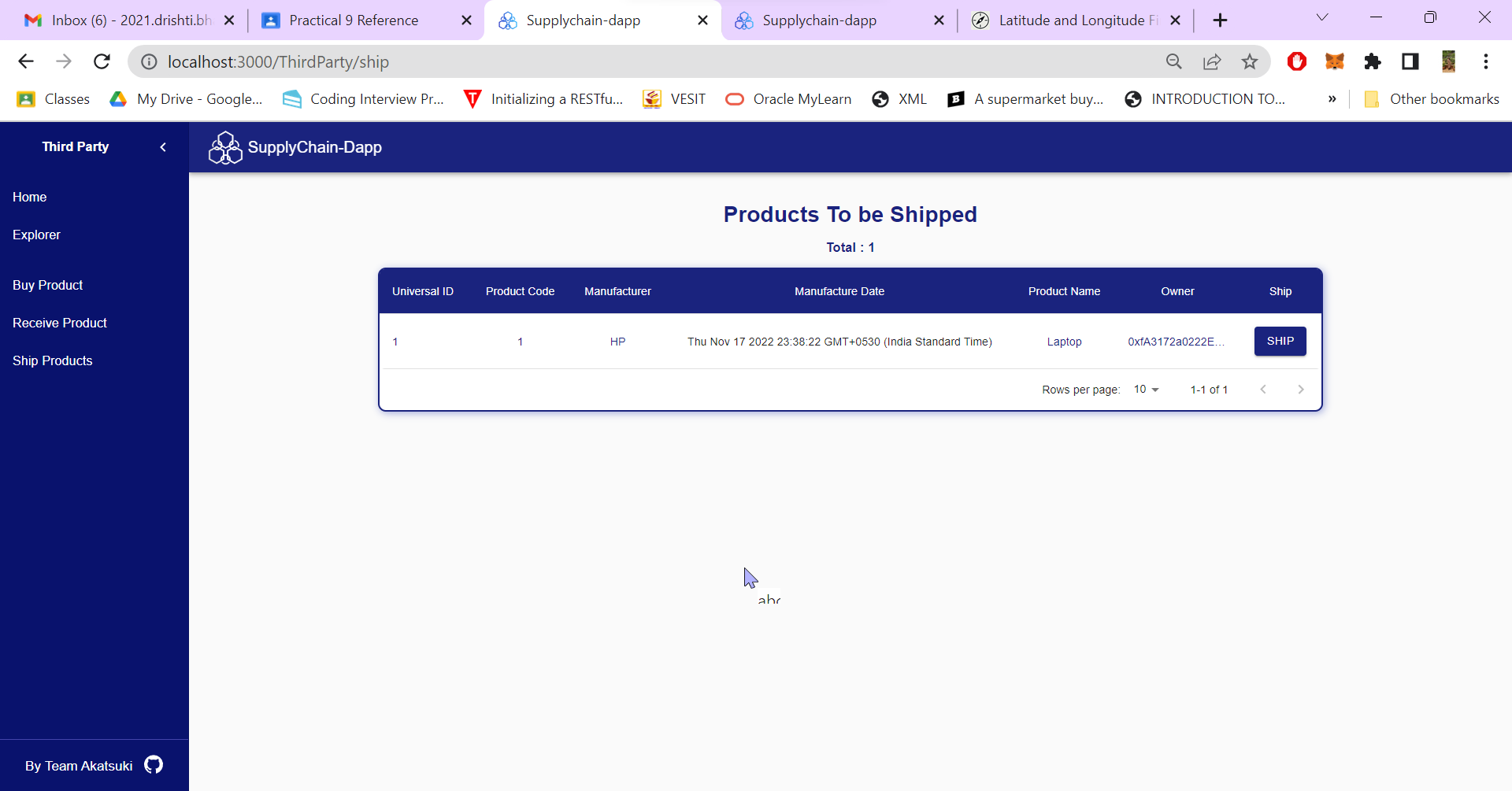
**Life Cycle Step 4: Customer:**

****

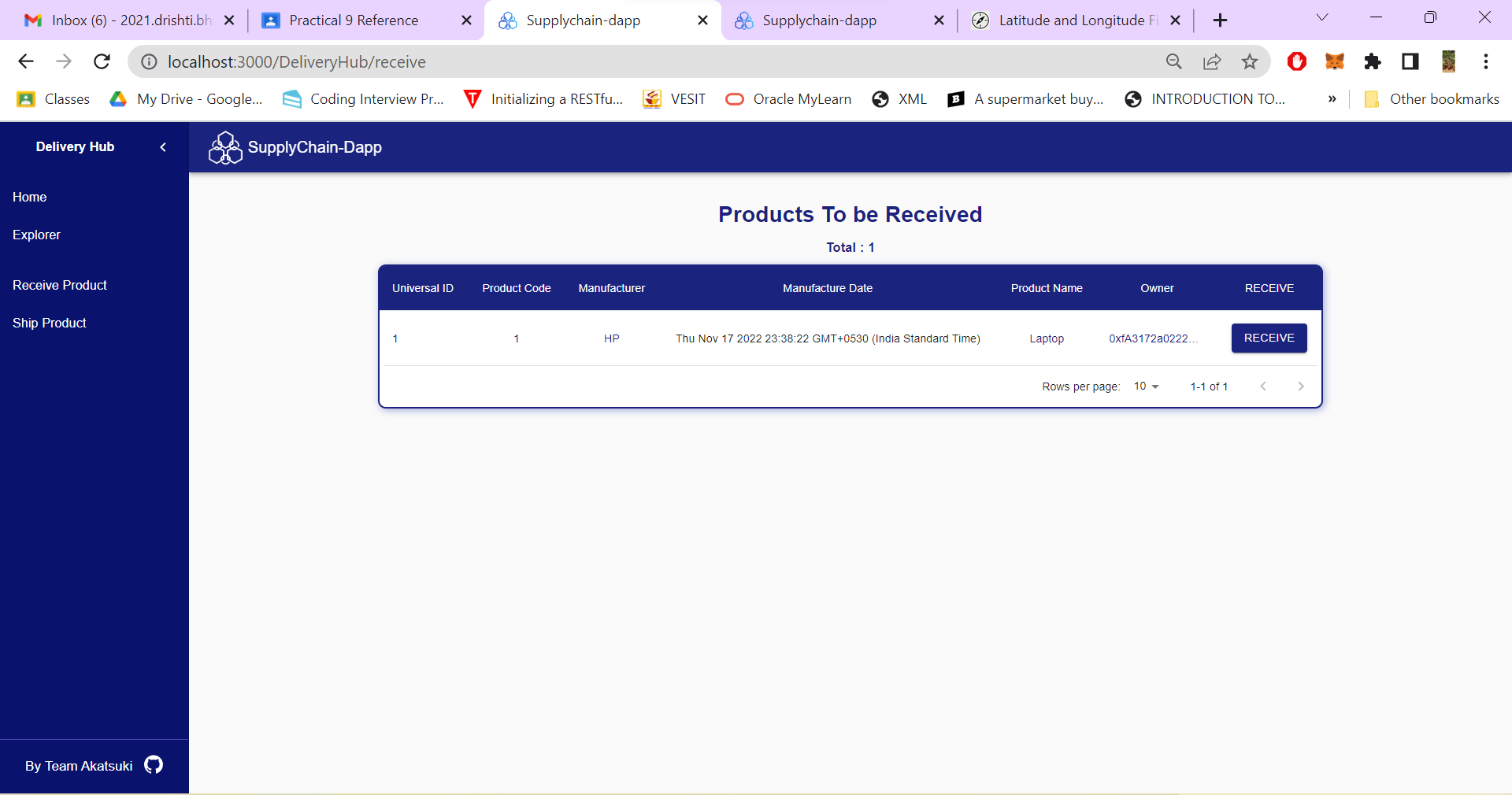
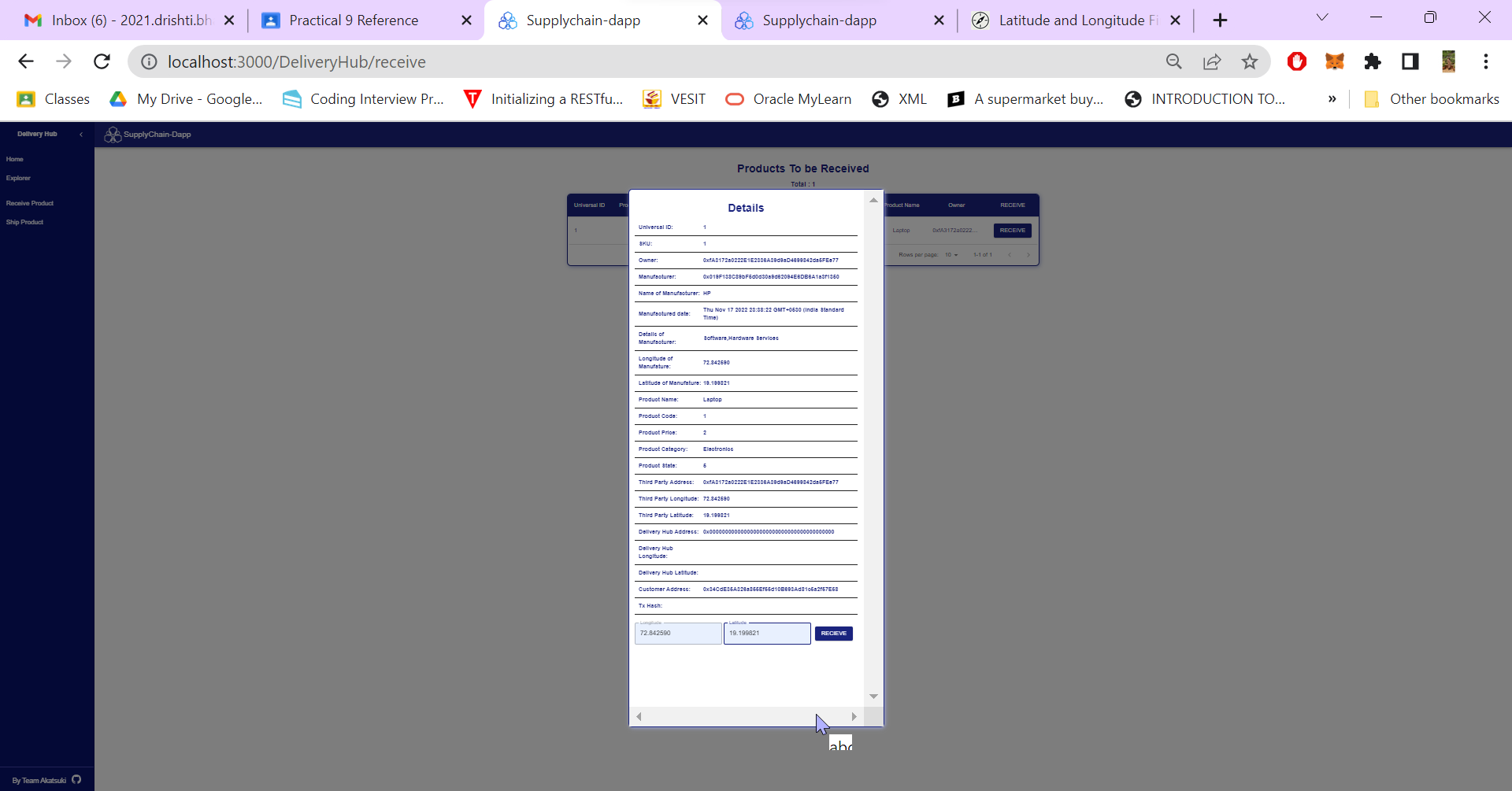


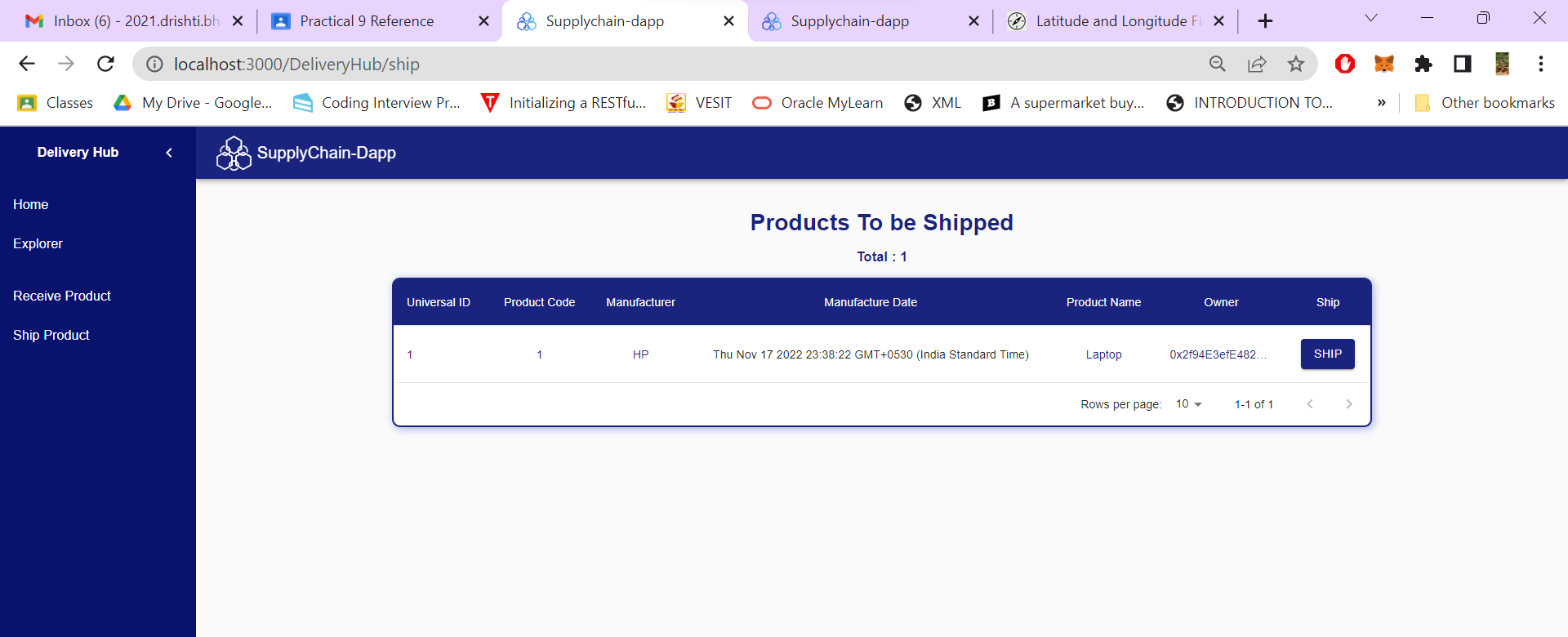


**Third Party (Move metamask to third party)**

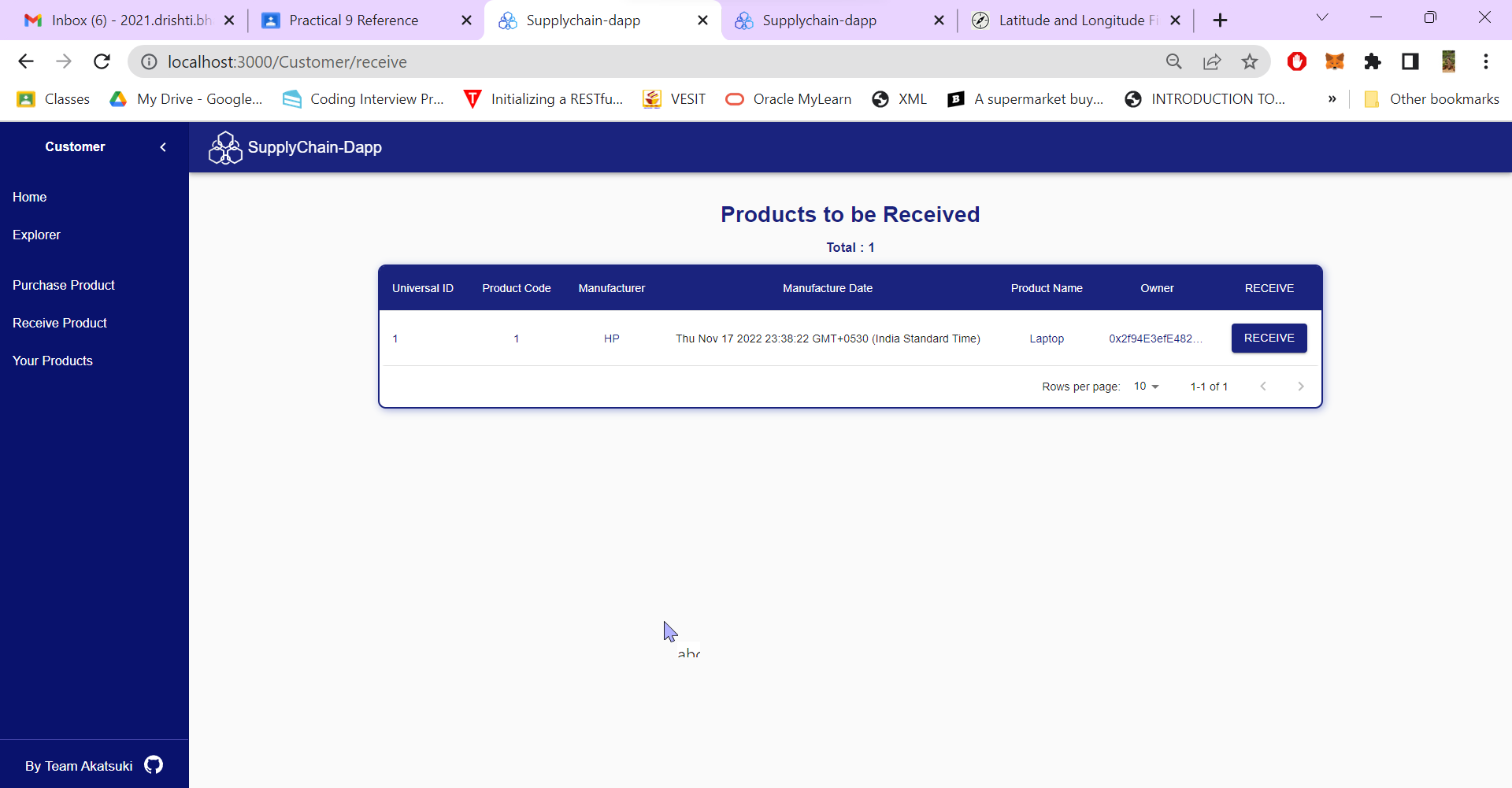


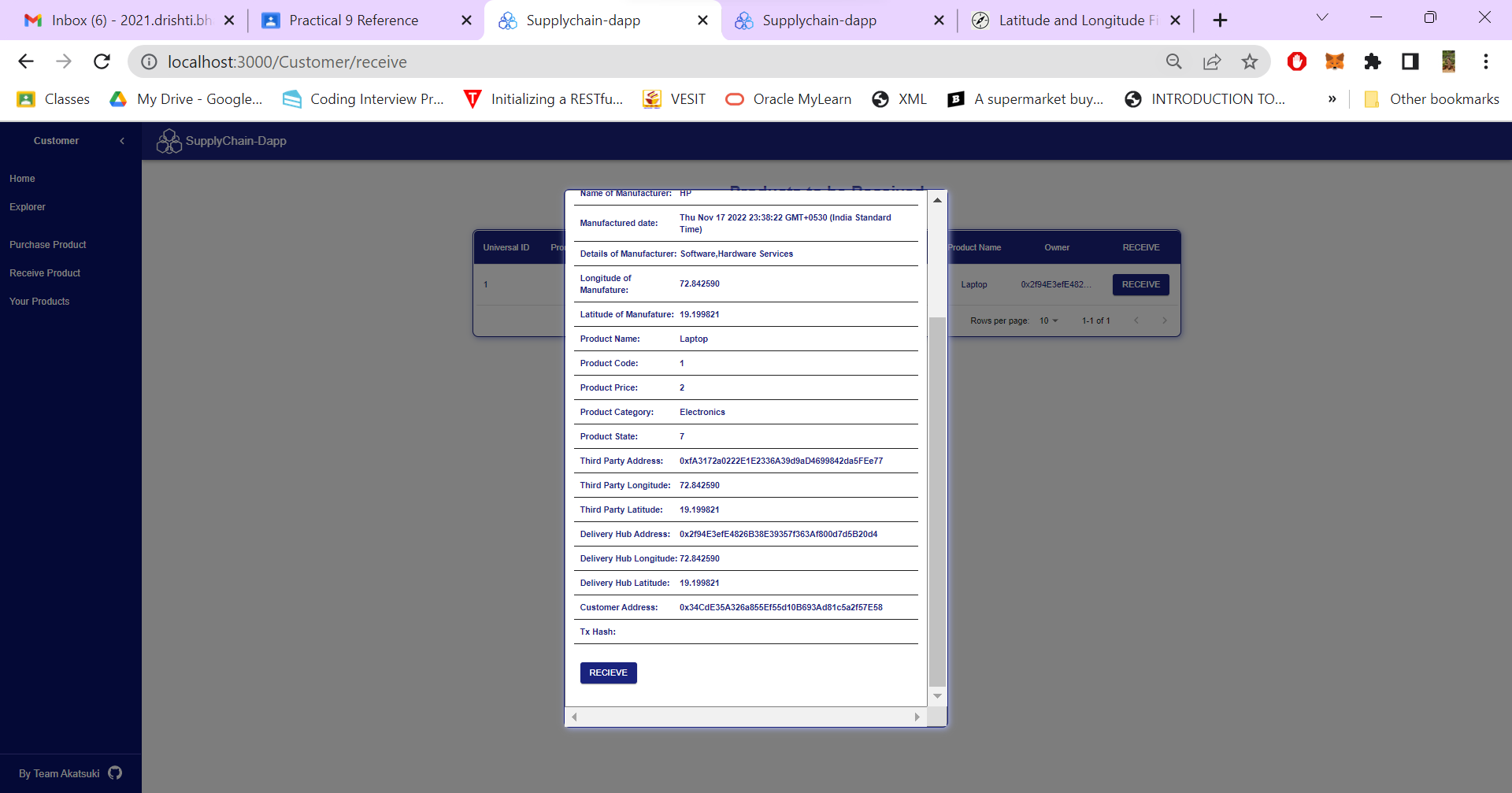
**Delivery Hub (Move to delivery hub):**

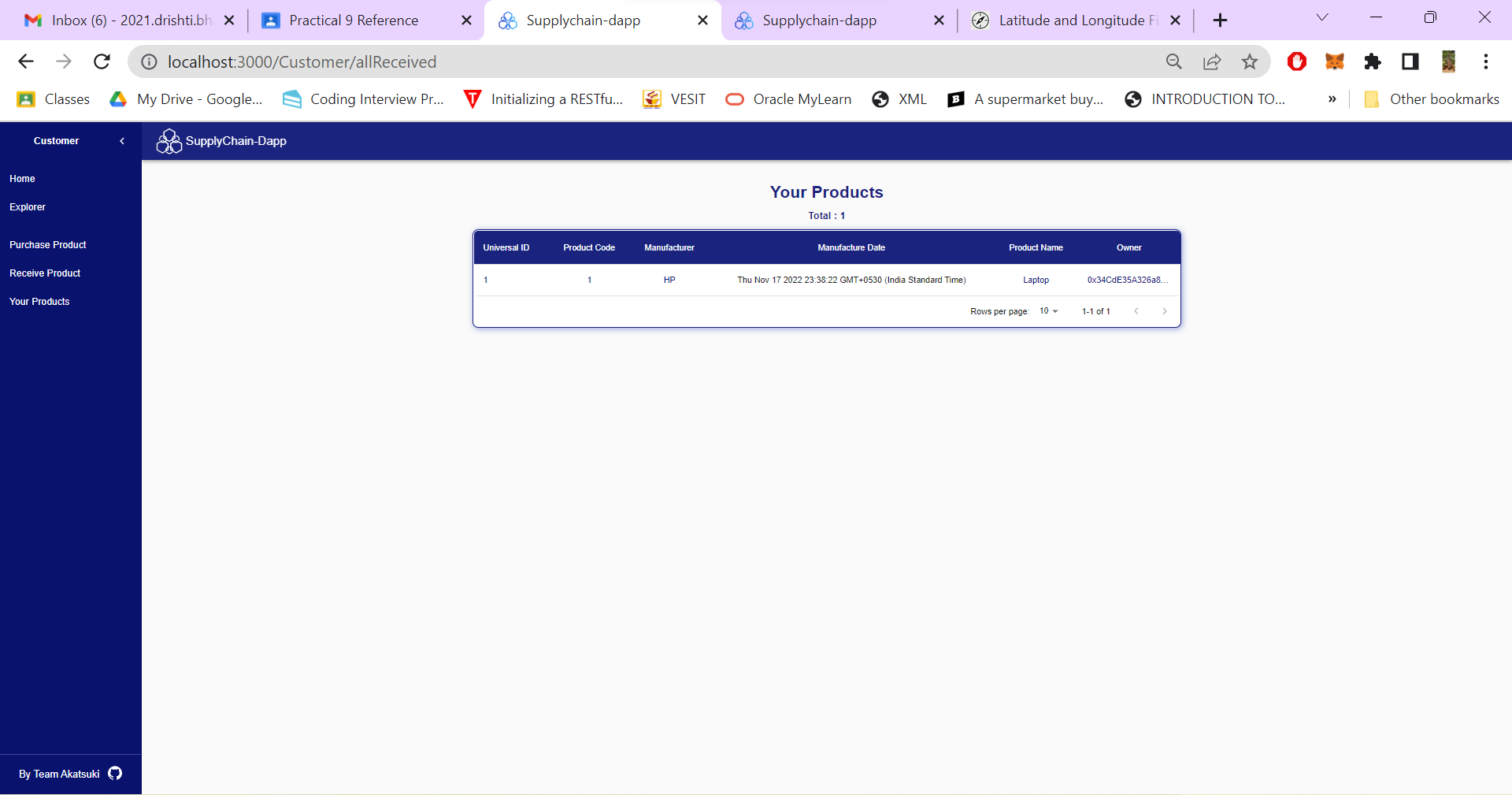
 

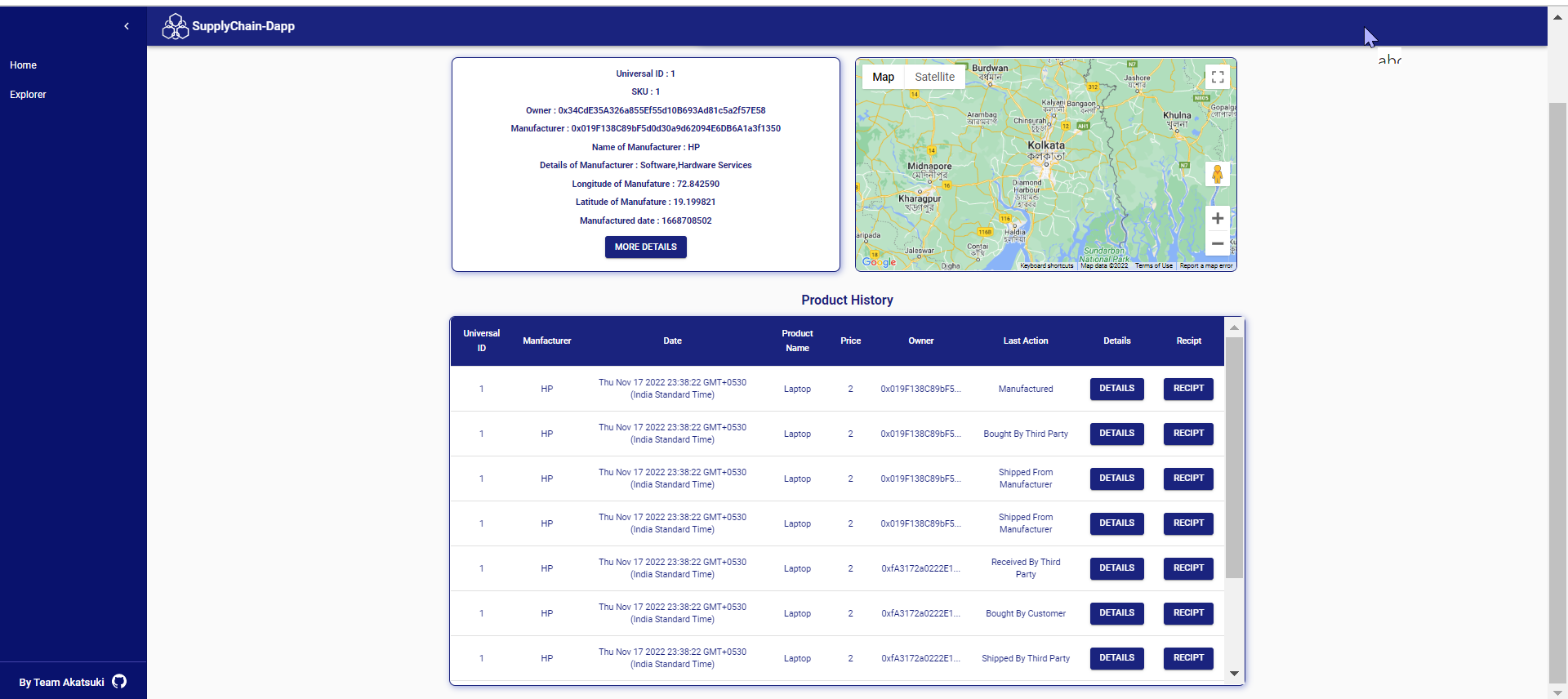


**Move to customer:**









**Conclusion: We have successfully implemented Decentralized apps in Ethereum.**