International Institute of Information Technology, Bangalore

CS 816- Software Production Engineering – Major Project

Blockchain based Supply Chain Management Solution

Developed By

Jass Sadana
(MT2023011)

Priyanshu Vaish
(MT2023089)

Table of Contents

1 Abstract

1.1 Problem Defnition

In modern supply chains, ensuring end-to-end product verification and maintaining data integrity across various stages of production, transportation, and delivery is a significant challenge. Traditional supply chain systems often suffer from issues such as lack of transparency, susceptibility to fraud, and difficulties in tracing the origin and journey of products. These challenges can lead to inefficiencies, increased costs, and a loss of consumer trust.
To address these issues, there is a need for a robust, transparent, and secure system that can provide real-time verification and tracking of products throughout the supply chain. This system should facilitate seamless transaction processing, ensure data integrity, and be scalable and portable for deployment across various blockchain environments.

1.2 Solution Description

Since the core of the problem lies in the fact that it demands transparency, data integrity and security, we need to think about a solution which can cater all of these requirements and hence we came across blockchain as the solution. Using blockchain as the core of the application, we are able to do :

- **End-to-End Product Verification**: The application ensures comprehensive tracking and verification of products throughout the entire supply chain, enhancing transparency and trust.
- **Decentralized Transaction Processing**: Utilizes blockchain technology to process transactions securely and transparently, reducing the risk of fraud and errors.
- **Data Integrity**: Smart contracts ensure that data related to the supply chain is immutable and reliable, preventing tampering and ensuring accuracy.
- **Scalability**: Designed to handle growing amounts of data and transactions efficiently, making it suitable for large-scale supply chains.

Real-time Monitoring: Provides real-time tracking and updates of product status, enabling stakeholders to monitor the supply chain continuously.**1.2.1**

1.3 Features of the website

There are 3 actors available in our environment:

- Manufacturer
- Seller
- Consumer

- **Manufacturer:** He is the one who will actually creates the product and he will be the entity who will be authorised to create the product and assign seller and consumer for that product. He will be able to view where his product goes, when it is getting verified

- **Seller :** These are the registered people with a particular manufacturer. The manufacturer can actually select from a list of sellers to assign a product to them to be able to sell it to the consumers.

- **Consumers:** After receiving the products, the consumer is able to scan the QR code on the product and verify if it was received from a verified seller and manufacturer or not. Using this metrics, he can easily verify the authenticity of that product.

1.4 Important Links

1. **Backend Source Code**: https://github.com/Priyansuvaish/SPE.git

2. **Frontend Source Code:** https://github.com/Priyansuvaish/SPE.git

3. **Docker Hub Registry:**

Ganache: https://hub.docker.com/r/secy2520/ganache

Frontend: https://hub.docker.com/r/secy2520/gan-frontend

Backend: https://hub.docker.com/r/secy2520/eth-backs

2 Introduction

2.1 What is DevOps?

DevOps is a group of concepts emerging from the collision of two trends. It combines software development with IT ops. It aims to decrease SDLC time and provide continuous delivery and integration functionality to produce high quality software.

2.2 Why to use DevOps?

It has now become important to include DevOps in software development as it enables faster development of product, faster bug fxes and easier maintenance as compared to traditional methods.

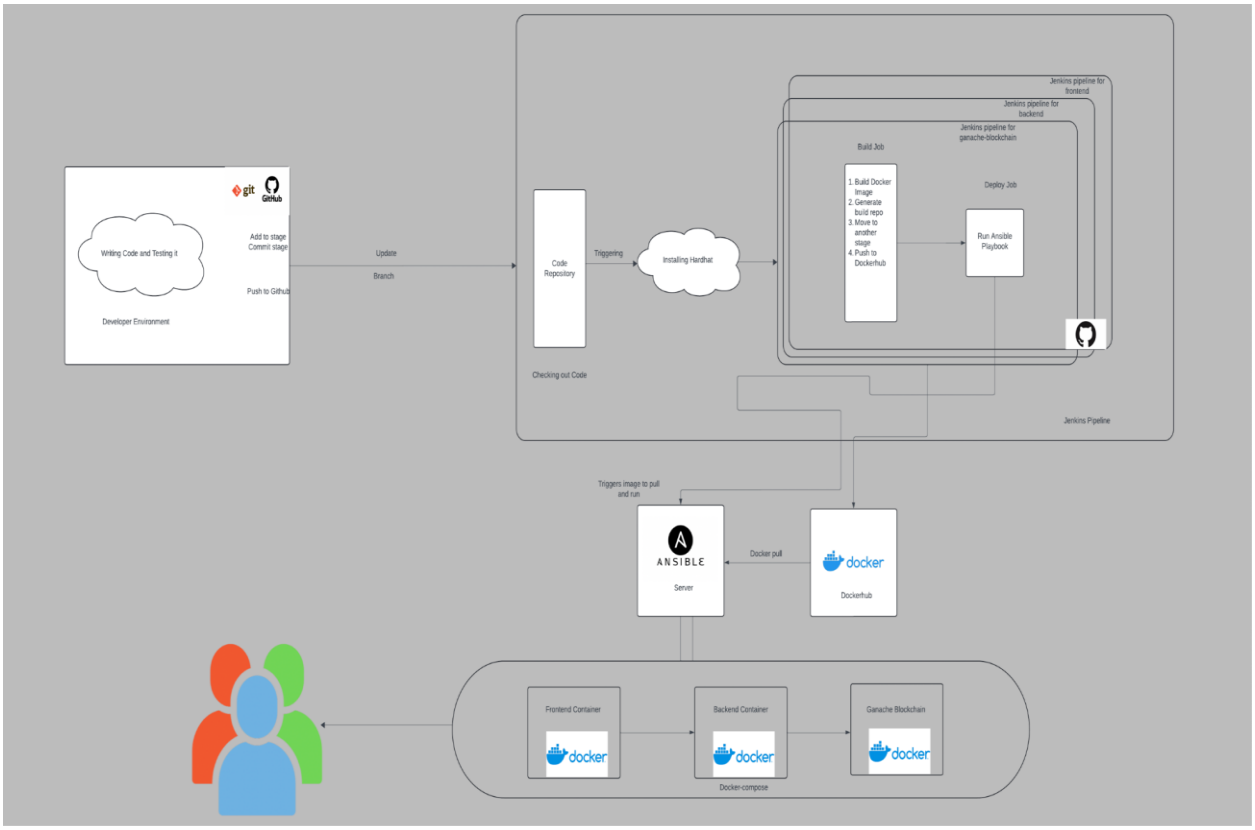
2.3 Technologies Used:

Frontend: ReactJs
Backend: NodeJs,Solidity
Databas: Ethereum Blockchain

2.5 Operation Tools Used

- **SCM** : Git and GitHub
- **CI/CD** : Jenkins
- **Containerization :** Docker
- **Build Tool** :NPM (Node Packet Manager), Hardhat(IDE for smart contracts)
- **Blockchain:** Ganache
- **Deployment:** Ansible

3 Architecture Diagram



4 System Configuration

- We used Kali Linux for testing purpose.
- Make sure that you have Jenkins and docker installed in it.
- Make sure NodeJs is installed in the system.
- Ansible should be installed and configured properly in the system.
- Make sure you have proper connectivity of internet.
- Make sure that repo is accessible from Jenkins node.

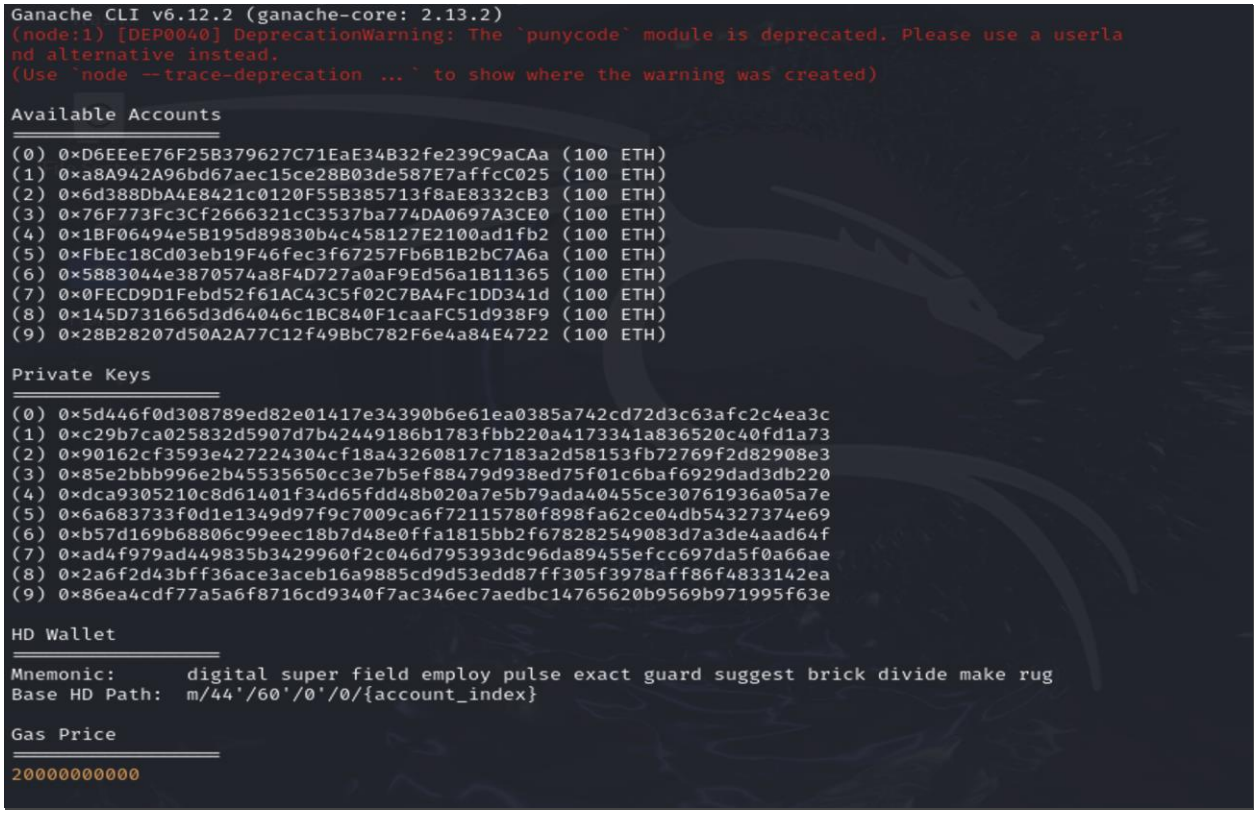
5 Setting Up the Ganache Blockchain

5.1 Installing the Ganache

1. In order to install ganache into linux, we used the npm.
2. Make sure your system has Nodejs installed in it.
3. Just go and enter the command

   $ npm install -g ganache-cli
4. After installing it, just run:

   $ ganache-cli -h <hostname>



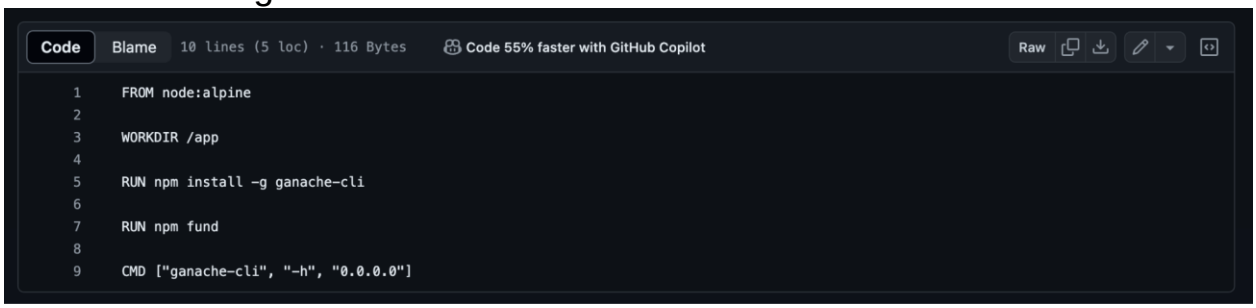
This is the screenshot of our private blockchain that got created after I ran the command ganache-cli -h at localhost. As we can see in the diagram, there are testing accounts along with their private keys in it. We will use these account address with private keys when we will be testing our application.

5.2 Containerizing the Blockchain

In order to containerise the blockchain, we need to first make ensure about the 2 things:

- proper base image is used
- Ganache can be installed through NPM inside the container.

Hence keeping these 2 things in our mind, we designed the Dockerfile looks something like this:

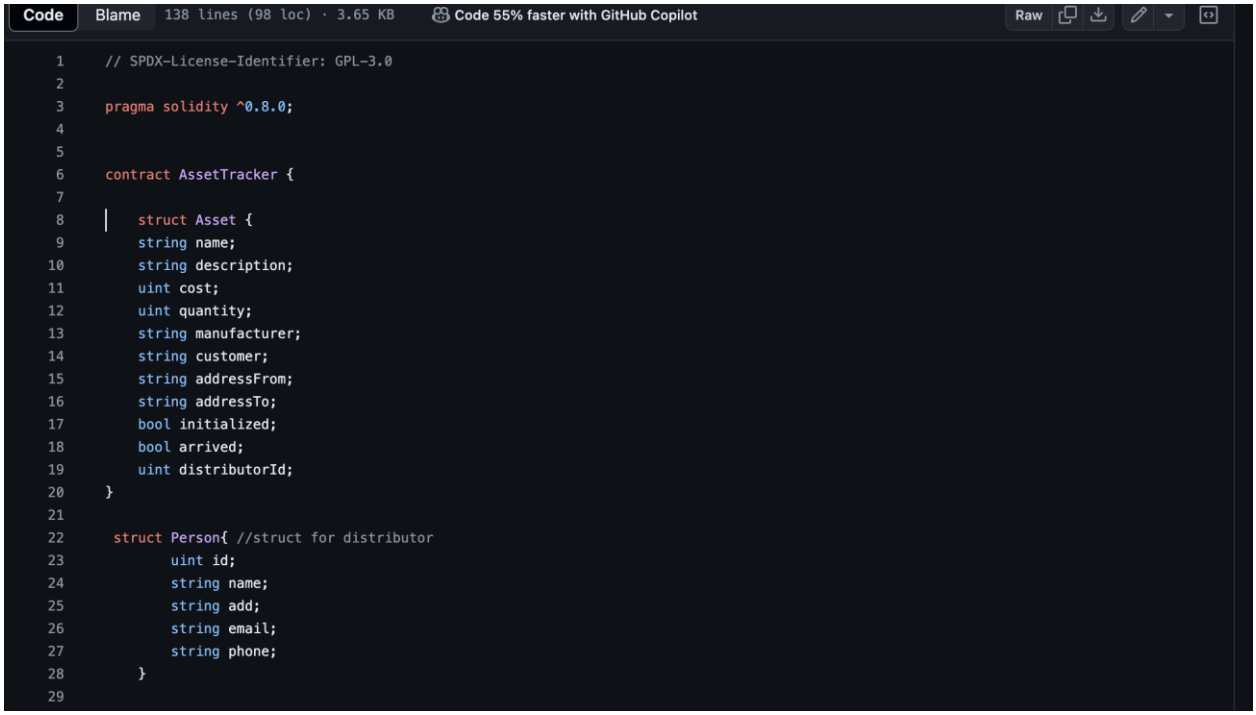


These 9 lines of code is doing nothing but just pulling the base image of the node from the docker hub and installing the ganache-cli in it. After that, it is starting the ganach-cli on the host 0.0.0.0, because we want to start ganache-cli on 0.0.0.0 which means that ganache will be started on all the available interfaces on that host.

6 Writing Smart Contract code with Hardhat

6.1 Writing the smart contracts

Smart contracts are one of the most important core logic layer of the blockchain. They will actually decide how will be transactions actually work. How to manage the Ethereum accounts, on what rules will the transaction takes place, all of that.




```
// SPDX-License-Identifier: GPL-3.0

pragma solidity ^0.8.0;


contract AssetTracker {

    struct Asset {
        string name;
        string description;
        uint cost;
        uint quantity;
        string manufacturer;
        string customer;
        string addressFrom;
        string addressTo;
        bool initialized;
        bool arrived;
        uint distributorId;
    }

     struct Person{ //struct for distributor
            uint id;
            string name;
            string add;
            string email;
            string phone;
        }

```

```
30      mapping(uint  => Person) private distributorStruct;
31      uint public distributorCount=0;
32      uint public assetCount=0;
33
34      event RejectDistributor(string  name,string  add,string  email,string  phone);
35
36       function insertDistributor(string memory name,string memory add,string memory email,string memory phone) public returns(bool)
37      {
38          for(uint i=0;i<distributorCount;i++){
39                  require(keccak256(abi.encodePacked((distributorStruct[i].email))) != keccak256(abi.encodePacked(("123@gmail.com"))),"Distr
40          }
41          distributorStruct[distributorCount]=Person(distributorCount,name,add,email,phone);
42          distributorCount++;
43          return true;
44      }
45
46      function getDistributorbyId(uint id)public view returns(string memory,string memory,string memory,string memory){
47          return (distributorStruct[id].name,distributorStruct[id].add,distributorStruct[id].email,distributorStruct[id].phone);
48      }
49
50      function getAlldistributors()public view returns (Person[] memory){
51        Person[]   memory id = new Person[](distributorCount);
52        for (uint i = 0; i < distributorCount; i++) {
53            Person storage member = distributorStruct[i];
54            id[i] = member;
55        }
56        return id;
57      }
58
59      function balance(uint _amount) public pure returns(bool){
60          require(_amount<20,"Balance need to be greater than 20");
61          return true;
62      }
```

As you can see this is some overview of our smart contract code written in Solidity, a languages used in blockchain and very similar to Javascript. Here the first line is defining the version of solidity we are using. Then we have some custom made structures that we defined in our application and then simply is the example of our functions.

6.2 Writing the Web3.js script

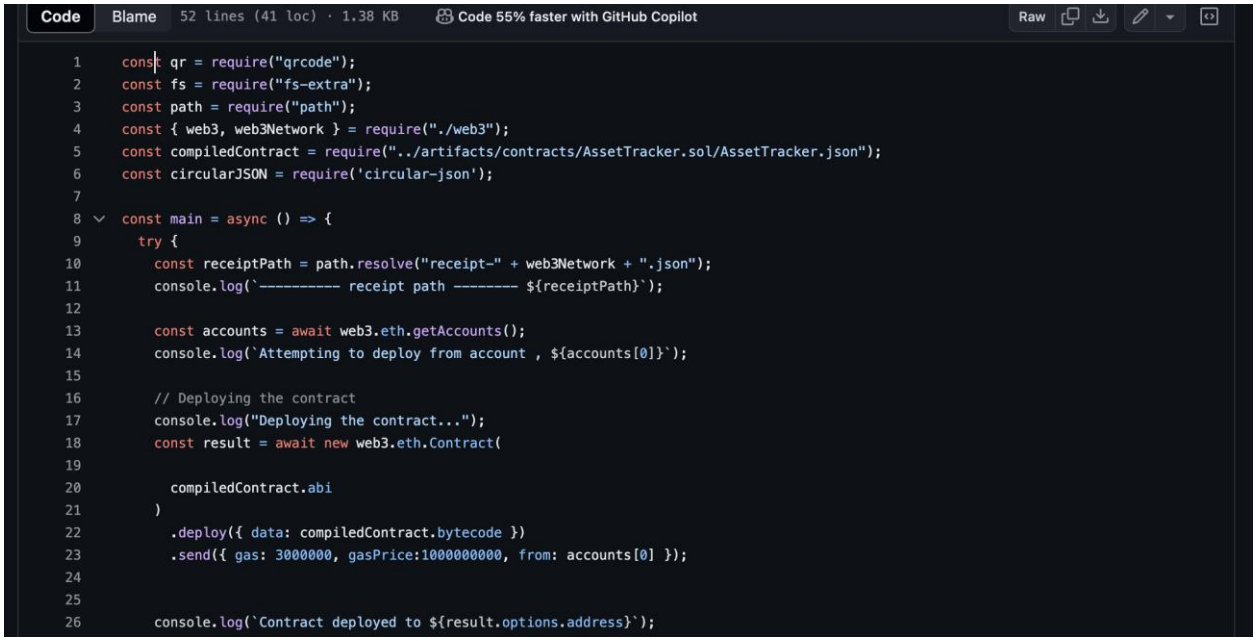
We3.js is a script that actually acts a interfacing script to the blockchain network. We will use the script to actually configure Blockchain Network for our application.

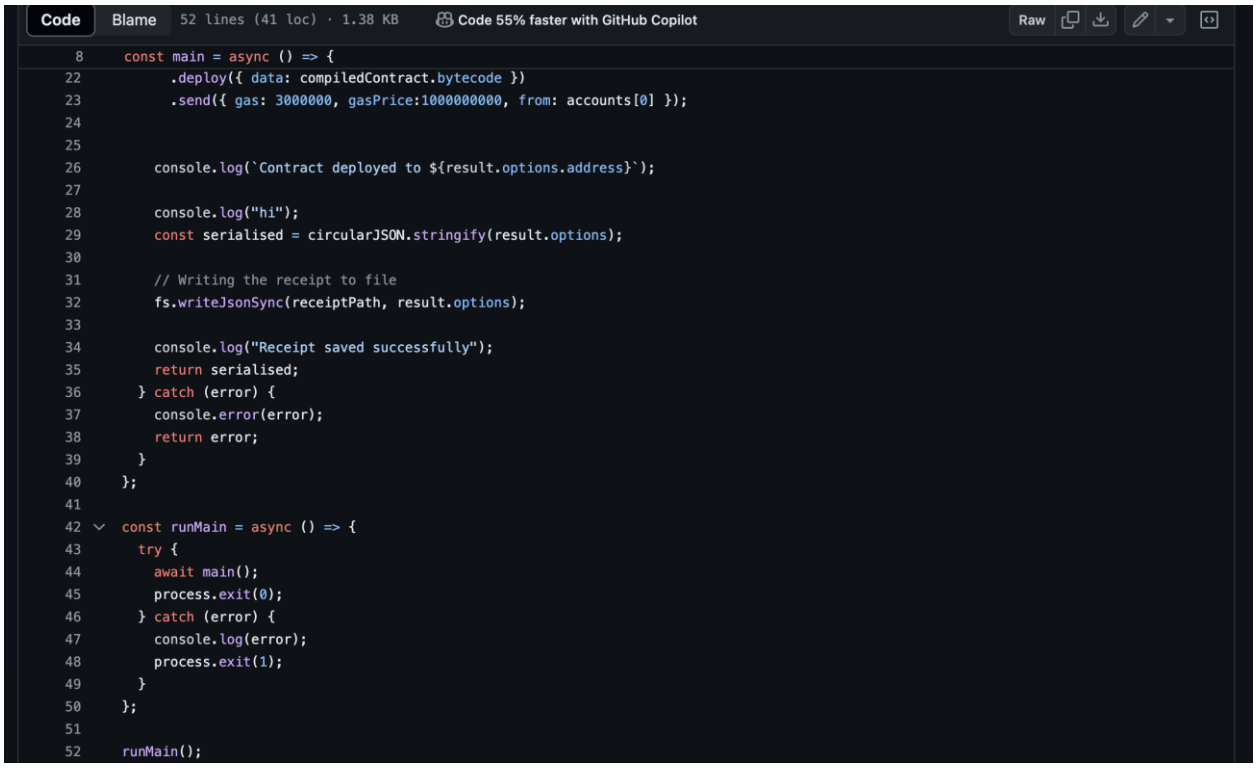
```
1
2   const {Web3} = require("web3");
3
4
5   const hardhatNetworkURL = "http://ganache:8545";
6
7
8   const web3Network = "gtr";
9   const web3 = new Web3(hardhatNetworkURL);
10
11
12
13  module.exports = {
14    web3,
15    web3Network
16  };
```

As per our given configuration details, the hardhatNetworkURL is the URL on which the our Ganache Blockchain is running.
Web3Network name is defined as ***gtr.***

6.3 Deploying the Smart Contract on the blockchain using deploy.js

In order to deploy smart contract on the blockchain that can actually take this built smart contract code on the blockchain, we need to write NodeJS script for this task.
The smart contract code will look like this:





As you can see the above code, it is importing two scripts web3.js and the artifact from the respective directories.

We3.js code is the code which will be act as an interfacing script to tell the hardhat on which application to deploy the smart contract code.

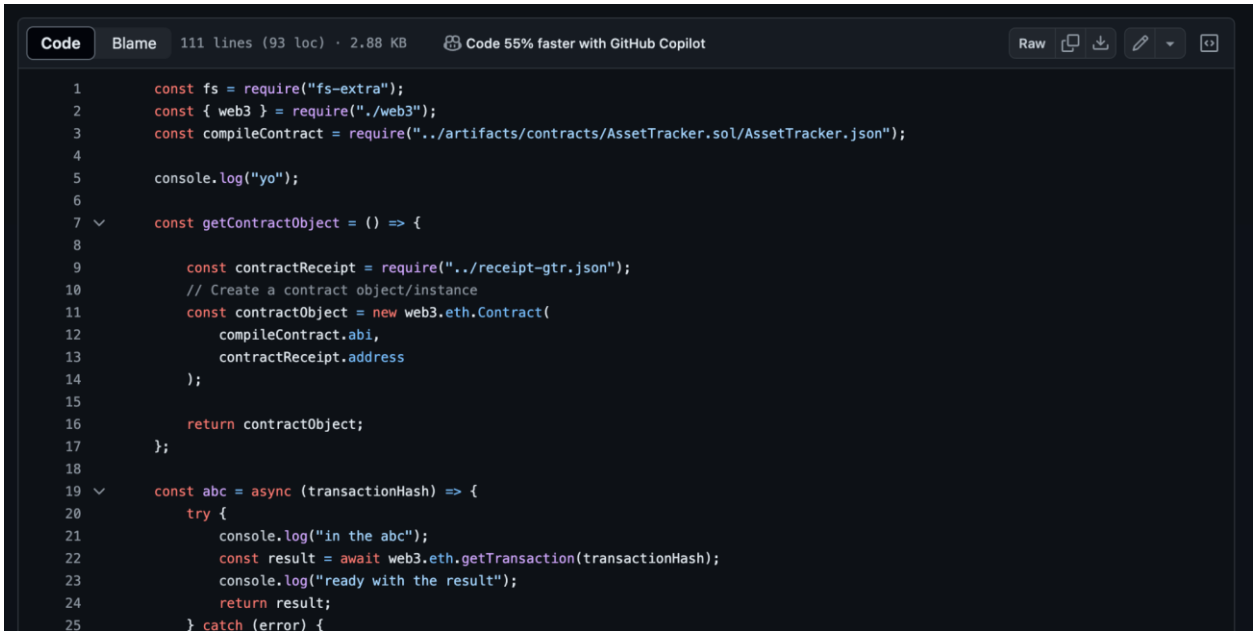
AssetTracker.json is the compiled binary produced during the compilation process.

Now as per the code it is first fetching Ethereum accounts from the defined web3Network and it uses first Ethereum account address as the main address to deploy the solidity code on the network.

After successful execution of this script, a json file will be created in the present directory which will have the contractAddress that is returned after successfully deploying it on the blockchain network and other useful interfacing details required to interact with the frontend code.

6.4 Writing the logic with Logic.js

Logic.js is the script which is acting as the service layer serving to the Smart-contract API.js in the Backend code. All the calls made by smart-contractAPI.js will be served by this logic.js



Here this is the screenshot of the logic,js where we have defined logics for all the APIs that will interface with the underlying blockchain.

6.5 Configuring Hardhat

In order to deploy the blockchain application, we need to build our smart contract code so that it can be used to deploy on the blockchain network. Hence in order to do build our code, we will be using one of the Etheruem Development platforms, ***Hardhat.***

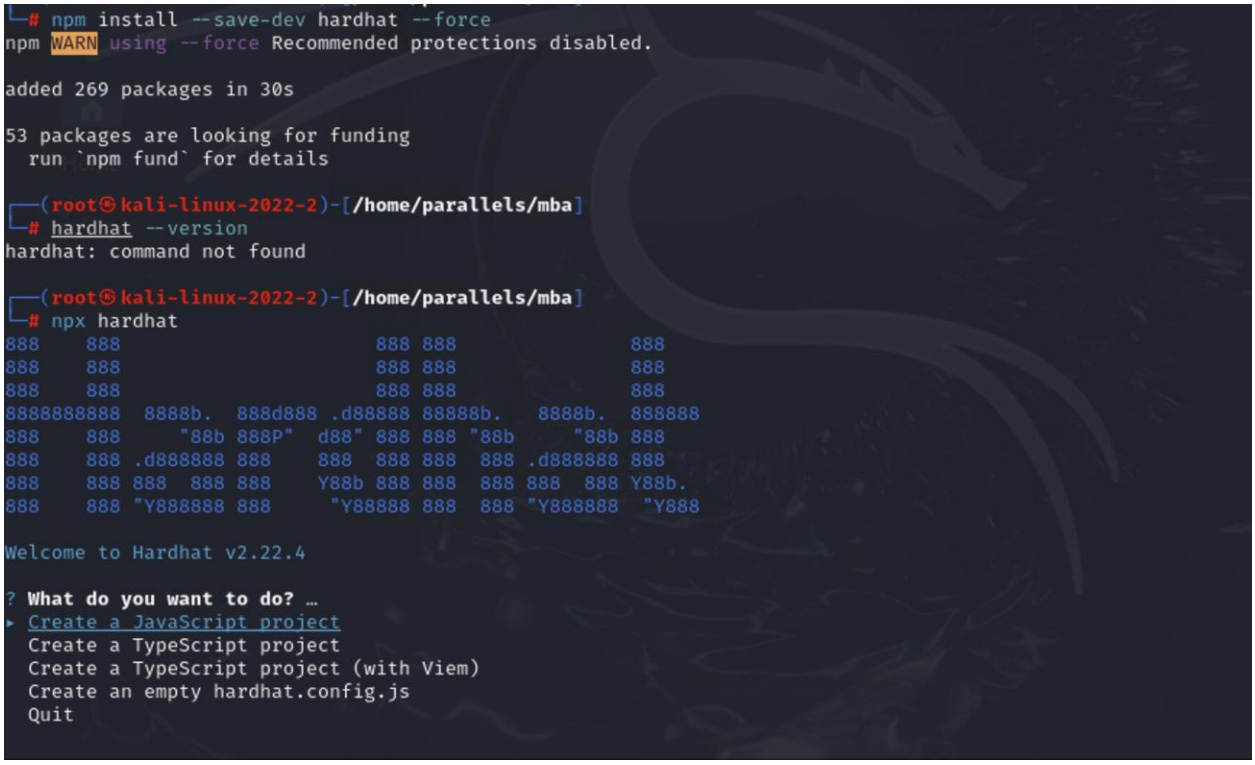
There are many other platforms available in the market but we still chose Hardhat because its more flexible and customizable than Truffle, and it's suitable for complex tasks.
In order to install hardhat, we will be installing it through npm.

1. Go to the directory where you want to configure it as a Hardhat directory and then install it by :

$ npm install –save-dev hardhat

2. After running it successfully, just run the command

$ npx hardhat



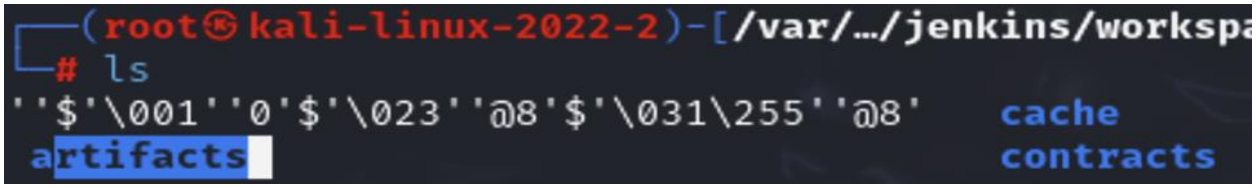
It is the output screen that you will get after running the hardhat. If this appears, that means the hardhat is successfully installed in the present working directory and its running.

Now, in order to build you application using Hardhat, you need to initialise the Hardhat project.

- Run the ***$ npx hardhat***. A screen like above will appear.
- Create an ***empty hardhat.config.js***
- Now after you do this, your current directory will get initialised as the Hardhat directory.
- Now just transfer all your smart contract code files to this directory.
- After doing all this, run the command

$ npx hardhat compile.

- If this command gets successful in compiling the contracts, a folder with name ***artifacts*** which will have all binaries associated with the code and essential for building the application.


(root㉿kali-linux-2022-2)-[/var/…/jenkins/workspa
ls
''$'\001''0'$'\023''@8'$'\031\255''@8'
artifacts
cache
contracts

7 Software Development Life Cycle Frontend (DevOps)

7.1 ReactJS (Front End)

React is a free and open-source front-end JavaScript library for building user interfaces based on UI components. It is maintained by Meta and a community of individual developers and companies.

Install NodeJS and Node Package Manager (NPM)

In order to run React, Node environment shall be installed before starting,

```
$ sudo apt-get install nodejs
```

```
$ sudo apt-get install npm
```

Create React App

```
$ npx create-react-app
```

Start React App

```
$ npm start
```

7.2 Source Code Management

Source control Management is used for tracking the file changes history, source code etc. It helps us in many ways in keeping the running project in a structured and organized way. In this project, we'll be using **GitHub** as SCM Tool which is a cloud-based version control system. Here we'll be focusing on the frontend part and the SCM setup, steps, workflow for **backend** is in the backend section under SCM.

a. Now let's create a GitHub repository for source code management.

b.Initialize git in the local folder and push the initial commit to the created repository

Commands:

- **git init** -> to initialize git in the current directory
- **git add .** -> staging the changes
- **git remote add origin** https://github.com/Priyansuvaish/SPE.git-> Setting the remote for fetch and push
- **git commit -m <Message>** -> to commit changes with message
- **git push origin master** -> Pushing the changes to the master branch of remote origin.

7.3 Building the React Application

- npm install -> Installs all the dependencies from *package.json* file.
- npm start -> Starts the application at the specified url and port

```
Compiled successfully!

You can now view asset-tracker in the browser.

  Local:            http://localhost:3000
  On Your Network:  http://192.168.29.22:3000

Note that the development build is not optimized.
To create a production build, use npm run build.

webpack compiled successfully
```

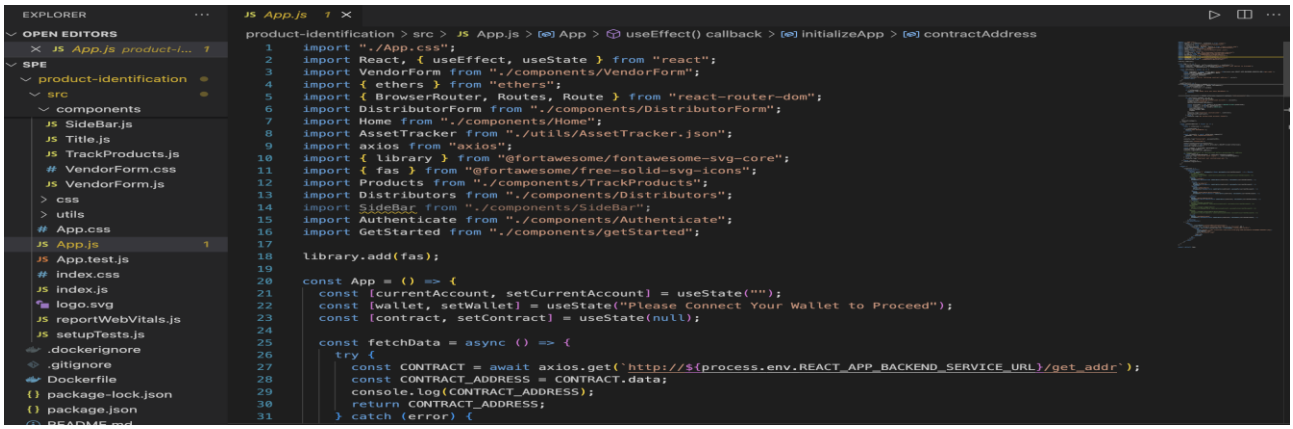
7.4 Containerization (Frontend)

Code Blame 11 lines (9 loc) · 300 Bytes Code 55% faster with GitHub Copilot Raw

```
# Use Nginx image as a base image
FROM nginx:alpine
WORKDIR /app
# Copy the built React app from the build folder to the Nginx web server directory
COPY build /usr/share/nginx/html

# Expose port 80 to the outside world
EXPOSE 80

# Command to run the Nginx server
CMD ["nginx", "-g", "daemon off;"]
```

As you can see in the above Dockerfile, we are using the base image nginx:alpine since our whole code is written in NodeJS. In the next step, we are copying the build file what we have created to the ***/usr/share/nginx/html*** folder inside the docker image. We just then expose the port 80 of the container as the application would be starting off at this port and then just executing the command ***nginx -g daemon off*** to run the nginx server.

7.5 Code Walkthrough



- All the building components are stored in /src/components directory

- All the Services which are responsible for API Calls are stored in **/src/api** directory
- All the React Utilities are in **/src/react-utils** directory.

8 Software Development Life Cycle Backend (DevOps)

Backend uses the express framework and NodeJs for creating a server. Express uses the Node.js runtime to make a server easily.

8.1 Source Code Management

We use git for managing source code, following commands can be used to create the backend repository.

Commands:

- **git init** -> to initialize git in the current directory
- **git add .** -> staging the changes
- **git remote add origin** https://github.com/Priyansuvaish/SPE.git > Setting the remote for fetch and push
- **git commit -m <Message>** -> to commit changes with message
- **git push origin master** -> Pushing the changes to the master branch of remote origin.

8.2 Writing the server code

- **Index.js file**

- **Contract-API.js**

```
25  app.use("/", contractAPIRoutes);
26
27  app.use("/",smartContractAPIRoutes);
28
29  app.use(function(err, req,res, next){
30      res.status(422).send({error: err.message});
31  });
32
33
34
35
36  app.listen( port, function(){
37      console.log(`Listening to the port ${port} .....`);
38  });
```

- **SmartContract-API.js**

Code | Blame | 35 lines (31 loc) · 1.09 KB | Code 55% faster with GitHub Copilot | Raw

```
1   const express = require("express");
2   const router = express.Router();
3
4   //const compile = require("../../Fake-prodcut-identification-smart-contract/scripts/compile");
5   //const deploy = require("../../scripts/deploy");
6   const get_addr = require("../../receipt-gtr.json");
7   const {web3,web3Network} = require("../../scripts/web3.js");
8   // Compile the contract
9   /*router.post("/compile", async function(req, res, next) {
10      const result = compile();
11      res.send(result);
12  });
13
14  // Deploy the contract
15  /*router.post("/deploy", async function(req, res, next) {
16      const result = await deploy();
17      res.send(JSON.parse(result).address);
18  });*/
19  router.get("/get_addr", async function(req,res,next) {
20      res.send(get_addr.address);
21  });
22  router.get("/acc_addr", async function(req, res, next) {
23      try {
24          const accounts = await web3.eth.getAccounts();
25          console.log(`Attempting to deploy from account: ${accounts[0]}`);
26          res.send(accounts[0]);
27      } catch (error) {
28          // Handle errors
29          console.error(error);
30          res.status(500).send("Internal Server Error");
31      }
32  });
33  module.exports = router;
```

These are the 3 core files that are actually serving the whole backend code to interact with the underlying smart contract code or the logic layer which is lying at the blockchain level.

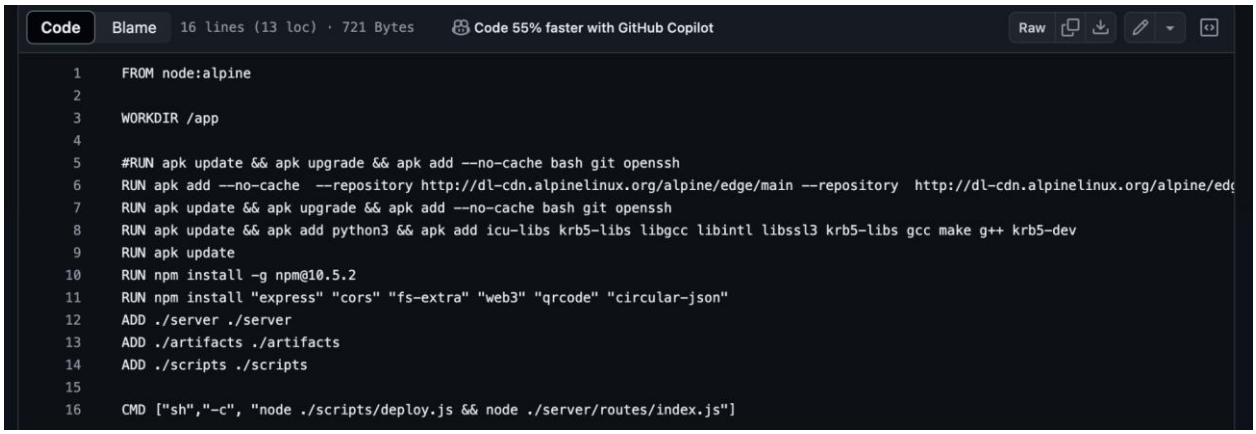
- **Smart contract-API.js** - In this file, we wrote all the APIs that are needed to extract information through smart-contract code or feed into the blockchain through the written smart contract code. By calling this smart contract-API.js in the main file, we are actually making call directly at the blockchain level or doing the transactions.

- **Contract-API.js**– In this file, we wrote all the APIs related to deploying the smart contract code on the blockchain.

- **Index.js** – It is kind of indexing file which is calling the other two files inside it and serving it publicly.

8.3 Containerization

In order to containerise the whole backend code, we need to make sure that all the dependencies related to what we imported in our backend code is correctly installed and there are no conflicts between the packages. The containerisation code looks something likes this:



As we can see, we correctly installed all the needed packages with node:alpine as the main base image. After doing all this, we actually copied some important files from the current directory to the container which includes;

- server
- artifacts
- scripts

server directory is the directory which contains all the above three scripts that we just used to interact with the backend.

artifacts directory contains all the compiled code build files that are actually be used for the production.

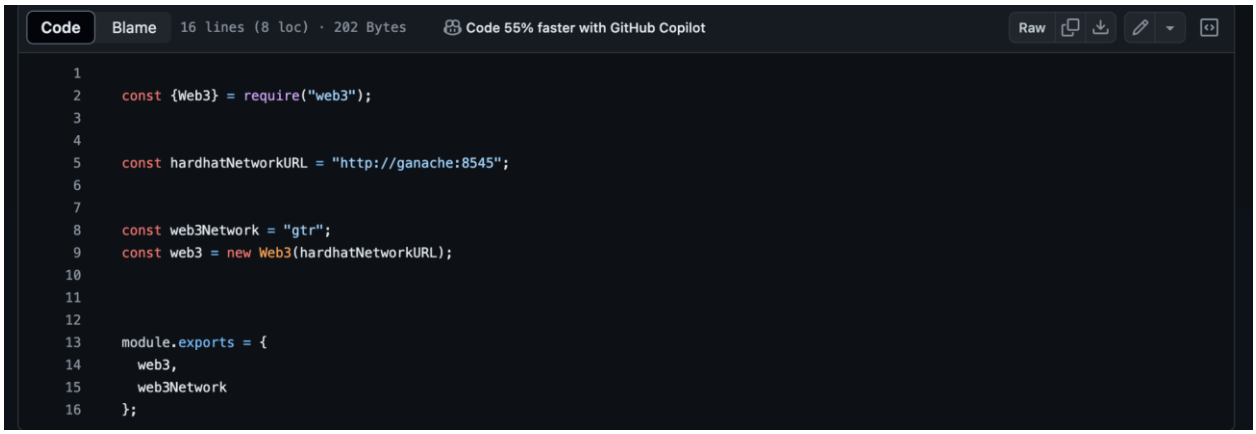
scripts directory contains all those scripts which are useful for interaction in the backend.

9 Orchestration of Containers using Docker-compose

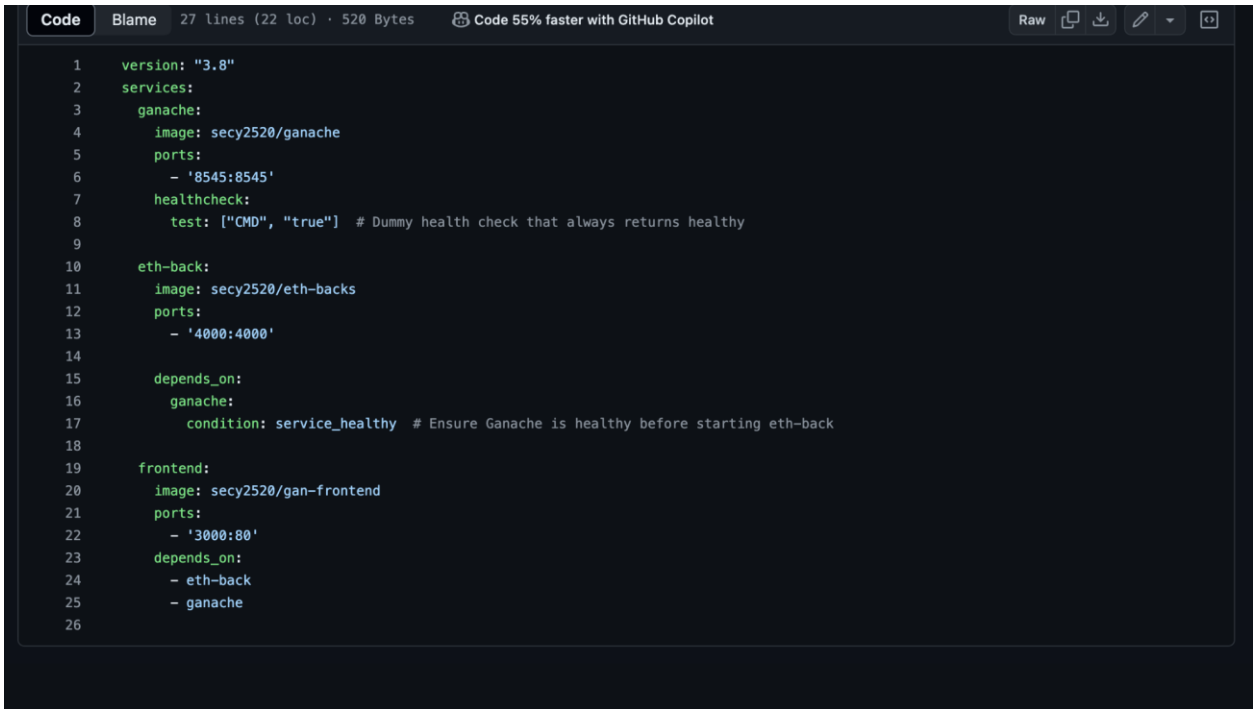
9.1 Writing the docker compose files

Since we have actually containerise our three units of application i.e. Frontend, Backend and the Blockchain. Now the challenge comes to us is how are we gonna establish communication among each of them. Some of our observations were:

- All three containers have a dependency among them i.e. they should execute in a particular order otherwise one of the container might fail to get start.
- Frontend is dependent on backend and backend is dependent on blockchain.
- First the blockchain will be deployed. As soon as the blockchain is available, now the web3.js should know about the fact the URL of the deployed blockchain. Since blockchain is deployed using one of the containers named as “ganache”, we have passed http://ganache:8545 in the web3.js file in order to let both the containers i.e. ganache and backend to communicate with each other



Looking after all the important observation, we came with the final design of our docker-compose file which looks something likes this:



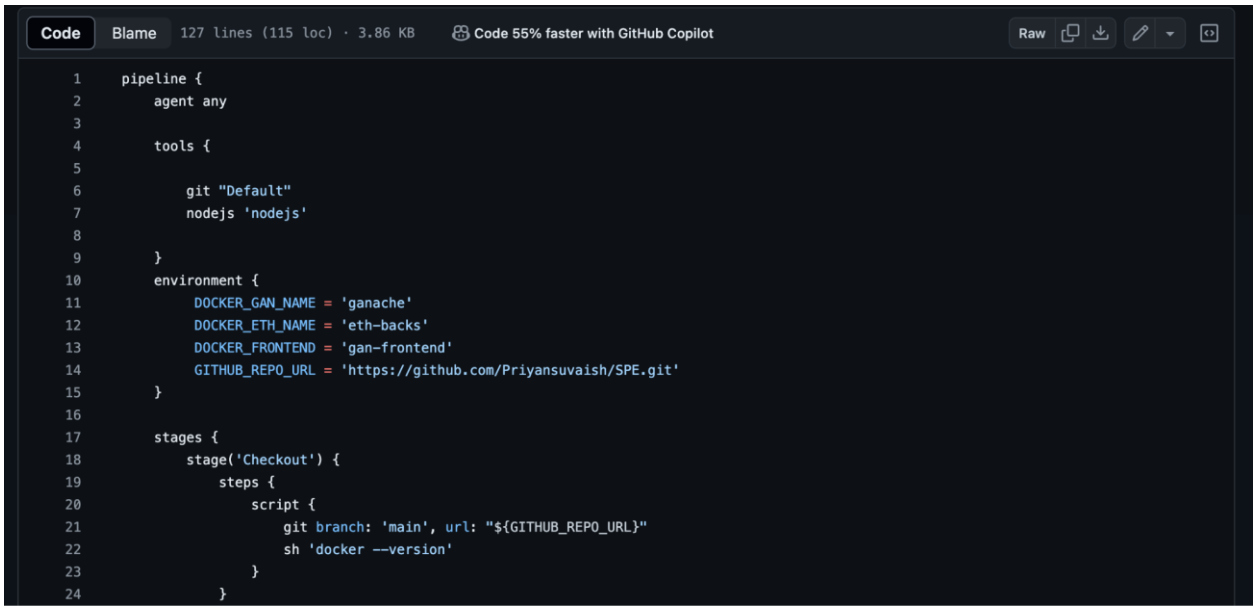

```yaml
version: "3.8"
services:
  ganache:
    image: secy2520/ganache
    ports:
      - '8545:8545'
    healthcheck:
      test: ["CMD", "true"]  # Dummy health check that always returns healthy

  eth-back:
    image: secy2520/eth-backs
    ports:
      - '4000:4000'

    depends_on:
      ganache:
        condition: service_healthy  # Ensure Ganache is healthy before starting eth-back

  frontend:
    image: secy2520/gan-frontend
    ports:
      - '3000:80'
    depends_on:
      - eth-back
      - ganache

```

As we can see, we clearly established a dependency among all the containers in an order so that they can execute properly.

10 Configuring the CI/CD pipeline using Jenkins

10.1 Writing the Jenkinsfile

In order to write the Jenkinsfile for automating the tasks, we used declarative syntax in order to define the stages. Here is the Jenkinsfile which we have written:




```
pipeline {
    agent any

    tools {

        git "Default"
        nodejs 'nodejs'

    }
    environment {
         DOCKER_GAN_NAME = 'ganache'
         DOCKER_ETH_NAME = 'eth-backs'
         DOCKER_FRONTEND = 'gan-frontend'
         GITHUB_REPO_URL = 'https://github.com/Priyansuvaish/SPE.git'
    }

    stages {
        stage('Checkout') {
            steps {
                script {
                    git branch: 'main', url: "${GITHUB_REPO_URL}"
                    sh 'docker --version'
                }
            }
```

```
26            stage('Installing hardhat')
27            {
28                steps {
29                    script
30                    {
31                    dir("/var/lib/jenkins/workspace/eth-project/eth-backs/")
32                     {
33                        sh 'npm install --save-dev hardhat --force'
34                         sh 'node -v'
35
36                     }
37                    }
38                }
39            }
40            stage('Building frontend code') {
41                steps {
42                    script {
43                        dir("/var/lib/jenkins/workspace/eth-project/product-identification/") {
44                            sh 'npm install --force'
45
46                            sh 'CI=false npm run build'
47                        }
48                    }
49                }
50            }
51
52           stage('Building and Testing Backend code') {
53                steps {
54                    script {
55                    dir("/var/lib/jenkins/workspace/eth-project/eth-backs/") {
56                    sh 'npx hardhat compile'
57                    }
58                }
59            }
```

```
71
72            stage('Build Docker Image for the ganache Blockchain') {
73                steps {
74                    script {
75                        // Build Docker image
76                        docker.build("${DOCKER_GAN_NAME}", '-f /var/lib/jenkins/workspace/eth-project/dockerfile_ganache .')
77                        // }
78                    }
79                }
80            }
81          stage('Build Docker Image for the Backend') {
82                steps {
83                    script {
84                        // Build Docker image
85                           dir("/var/lib/jenkins/workspace/eth-project/eth-backs/") {
86                        docker.build("${DOCKER_ETH_NAME}", '-f /var/lib/jenkins/workspace/eth-project/eth-backs/docker_backend .')
87                           }
88                    }
89                }
90            }
91            stage('Push Docker Images') {
92                steps {
93                    script{
94                        docker.withRegistry('', 'docker') {
95                        sh 'docker tag eth-backs secy2520/eth-backs:latest'
96                        sh 'docker tag ganache secy2520/ganache:latest'
97                        sh 'docker tag gan-frontend secy2520/gan-frontend:latest'
98                        sh 'docker push secy2520/gan-frontend'
99                        sh 'docker push secy2520/eth-backs'
100                       sh 'docker push secy2520/ganache'
101                       }
102                    }
103               }
```

```
                    sh 'docker tag ganache secy2520/ganache:latest'
                    sh 'docker tag gan-frontend secy2520/gan-frontend:latest'
                    sh 'docker push secy2520/gan-frontend'
                    sh 'docker push secy2520/eth-backs'
                    sh 'docker push secy2520/ganache'
                    }
                }
            }
        }
        stage('Removing all present images and the containers ') {
            steps {
                script{

                   sh 'docker system prune -f'

                }
            }
        }
        stage('Run Ansible Playbook') {
            steps {
                script {
                    ansiblePlaybook(
                        playbook: 'deploy.yml',
                        inventory: 'inventory'
                     )
                }
            }
        }


    }
}
```

Before entering into the stages, we defined some tools and environment variables that are essential for smooth running of our application:

Tools include:

git: "Default" as per the configuration setup in the Jenkins Master Node.

nodejs 'nodejs' as per the version of Nodejs installed in the Jenkins Master Node.

Environment Variables:

Lets go through all the stages stage by stage:

Stage 1: In the stage1, we first checkout the code from the github repository.

Stage 2: In the stage 2, we then installed the hardhat that will be used a building tool in our application stages

Stage 3: We built the frontend code by first traversing to the target directory and then doing the npm build in it using the command:

'npm install --force'

'CI=false npm run build'

Stage 4: In this stage, we built the frontend code by doing ***'npm install –force'*** to install the required libraries without any conflict of dependencies and then building the frontend code.

Stage 5: In this stage, we built the backend solidity code ***'npx hardhat compile'*** which will generate the artifacts for building the solidity code

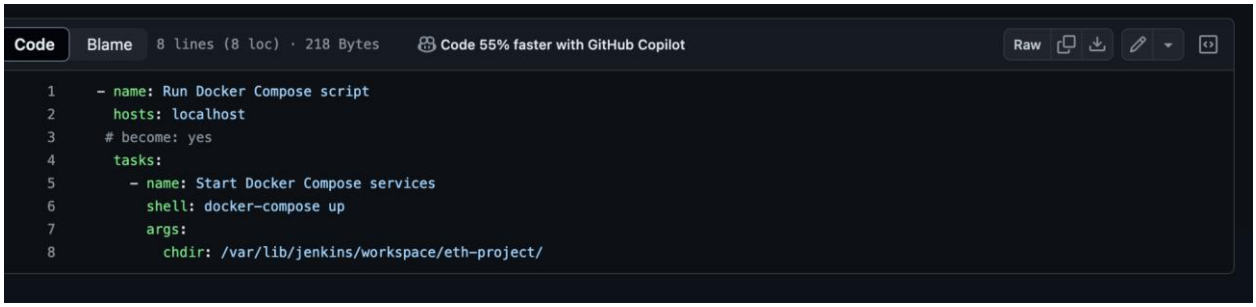
Stage 6-9: In all these stages, we actually build the docker images for all the component separately i.e. frontend, backend and ganache blockchain.

Stage 7: In stage 7, we actually push all these docker images in the dockerhub after a successful build.

Stage 8: In stage 8, we removed any dangling images that will be dangling with the currently build images.

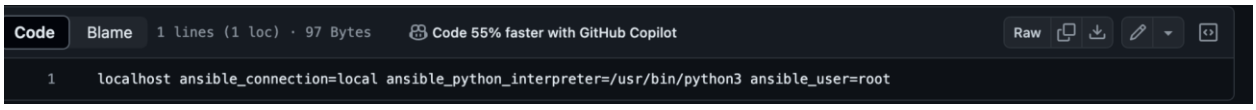
Stage 9: In stage 9, finally run the playbook with ***deploy.yml*** as the playbook and ***inventory.ini*** with both of them present in the same directory.

10.2 Writing the Ansible Playbook(Deploy.yml)



Our Ansible playbook script consists of only one task i.e. running the required Docker-compose script from the destination directory.

10.3 Writing the Inventory file



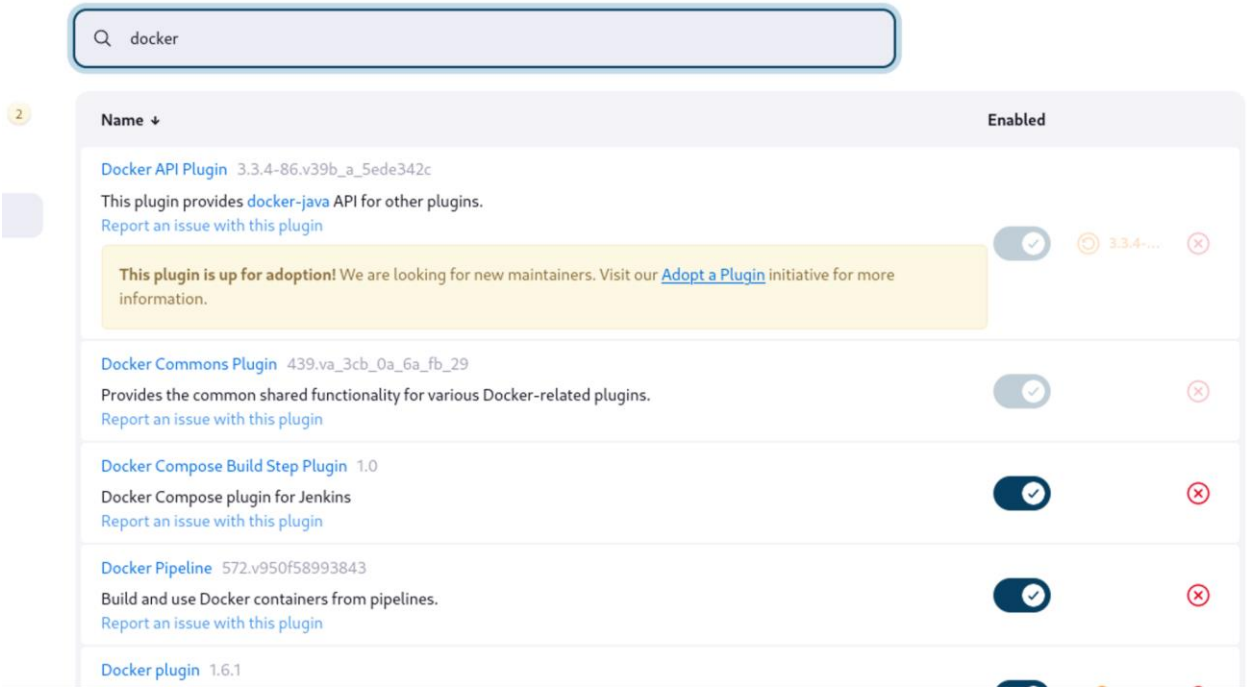
As clear from the above script, this script is responsible for mentioning the hosts on which ansible playbook should run on. As you can see from the above script, it mentions, the ansible-connection to be local and runs the python_interpreter from the installed binary in the /usr/bin/python3

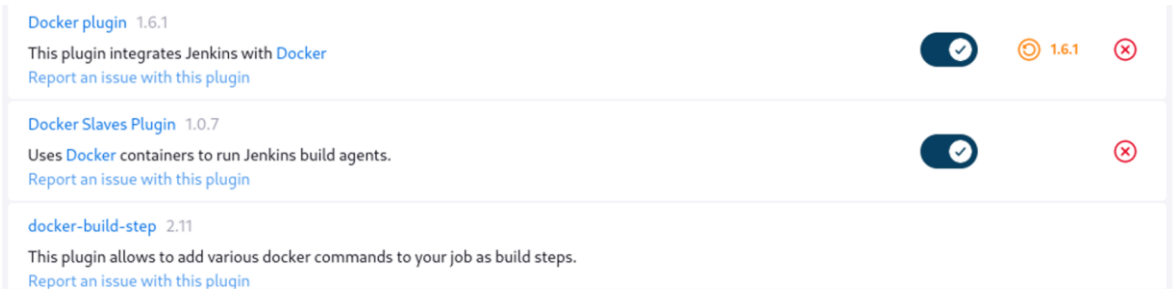
directory. The ansible_user is given as root that justifies that we are using the root user to run the playbook script on the Ansible node.

10.4 Setting Up the Jenkins Master Node

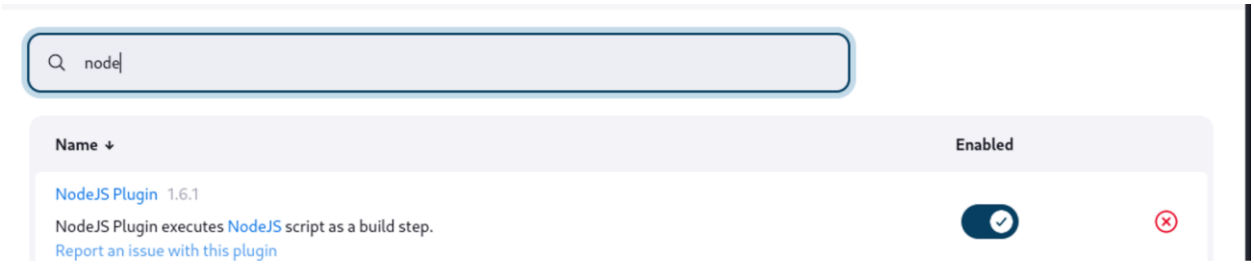
Before coming to this step, make sure that you have Jekins installed in your master node. After the successful installation of Jenkins on your Master node i.e. where you want to run the Jenkins CI/CD pipeline, the following steps should be followed in order to replicate the same scenario:

- Go to ***Dashboard > Manage Jenkins > Plugins*** install all these plugins for docker.

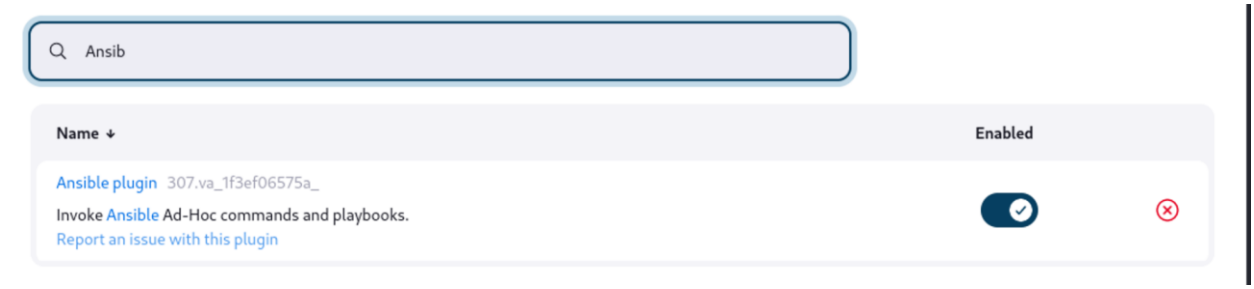




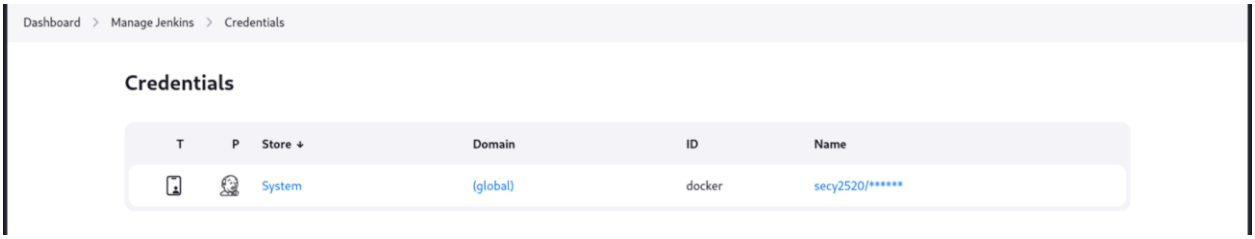
- After the successful installation of the above plugins, install the following ***NodeJs plugin.***



- Since we are using Ansible for deployment, install the below required plugin for Ansible Deployment.



- After all these plugins have been installed, go to ***Dashboard > Manage Jenkins > Security > Credentials*** and make Credentials to access docker from the Jenkins client.

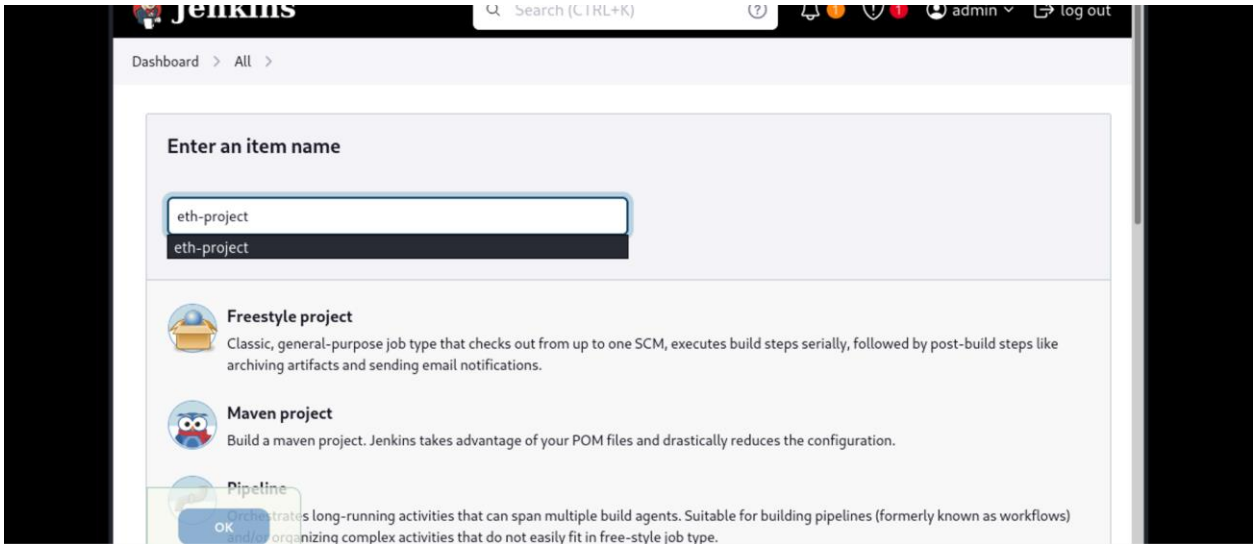


After completing all the above required steps, you are good to go to run a pipeline in the next steps.

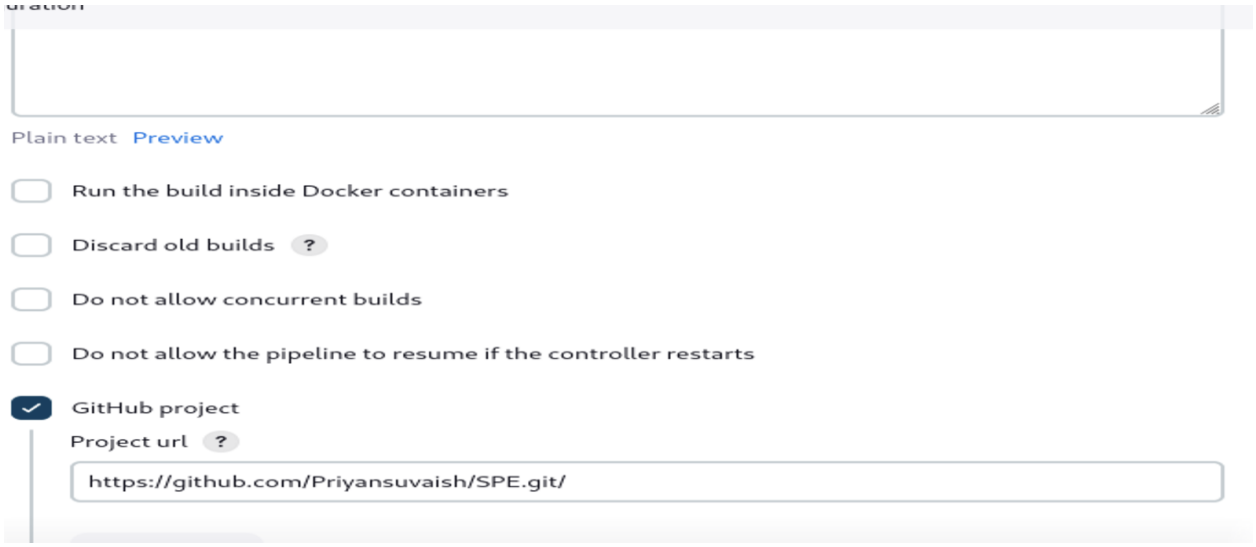
10.5 Running the pipeline

After all the above steps have been successfully implemented, lets configure the job for running the pipeline

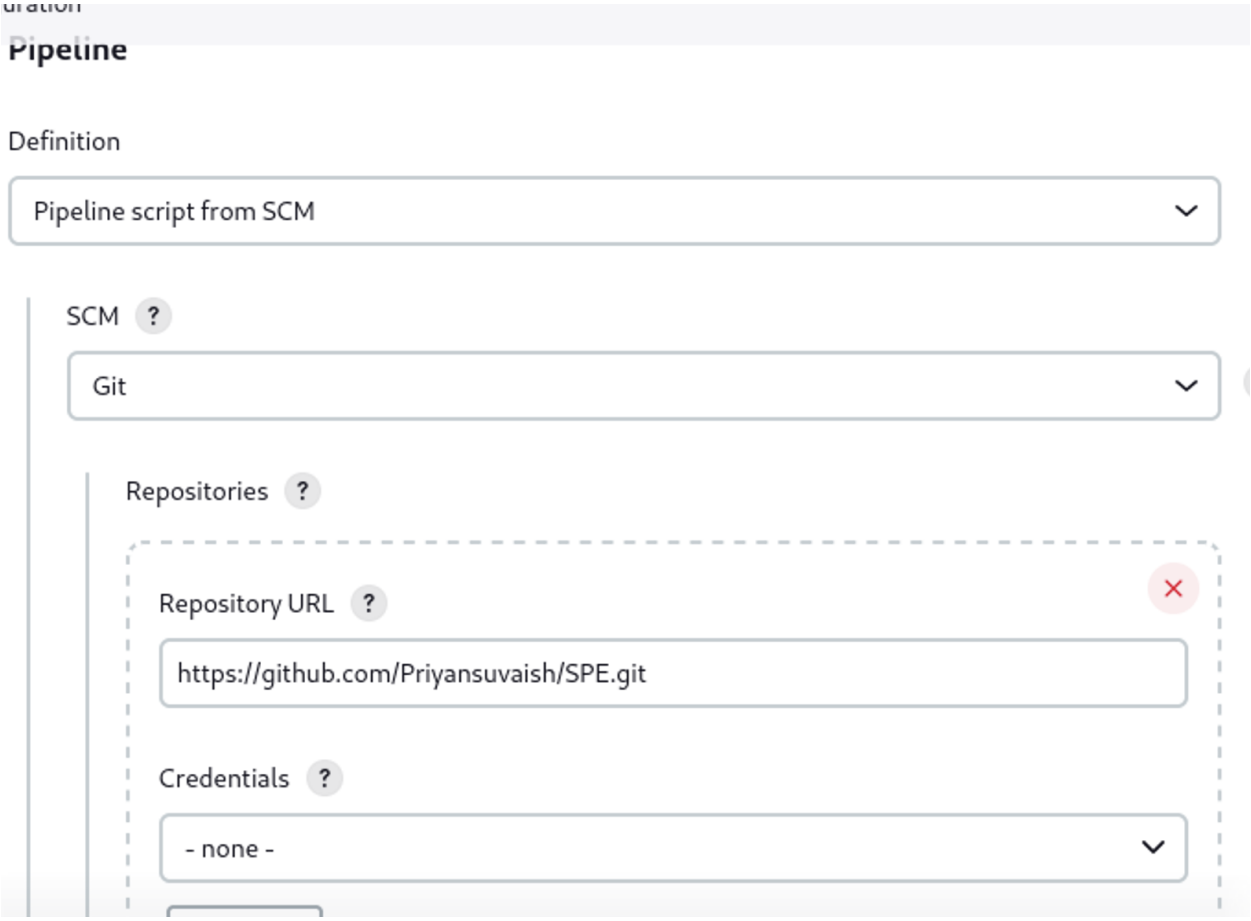
- Go to dashboard and click on ***New Item.***
- After entering the item name, choose the Job type as ***Pipeline.*** In our case, we choose the job name as ***eth-project.***



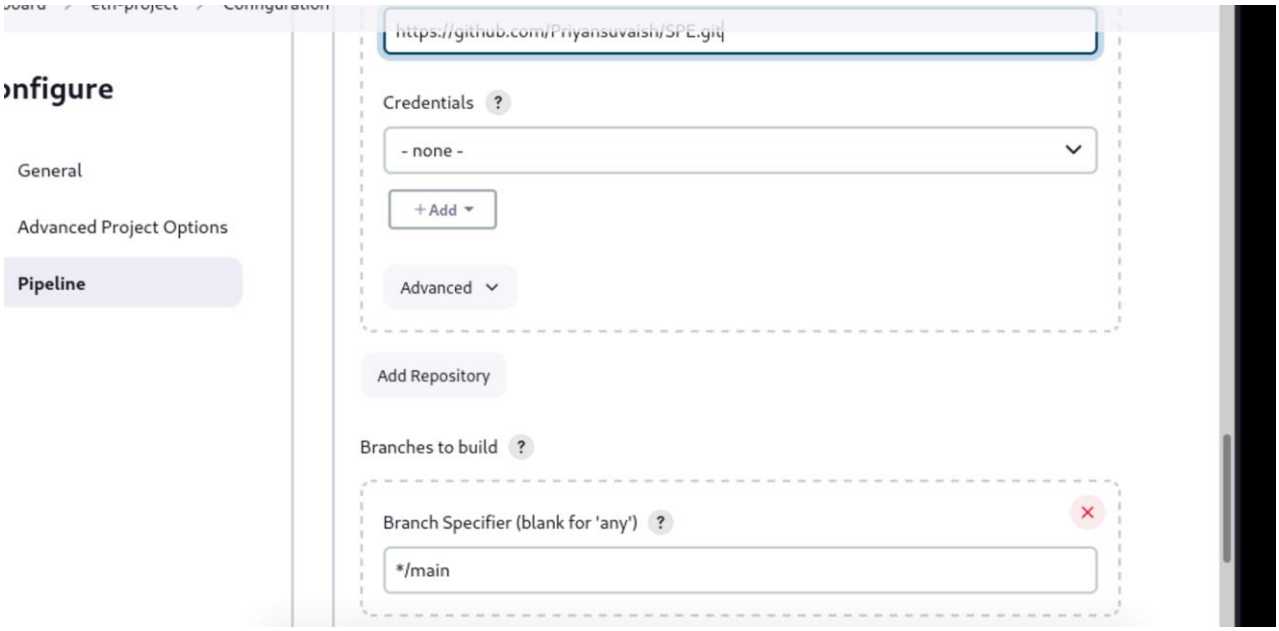
- After making the pipeline, lets configure the pipeline. To do this, click on the Job and since our project is hosted on github, pass the URL of the project in the ***Github Project*** section

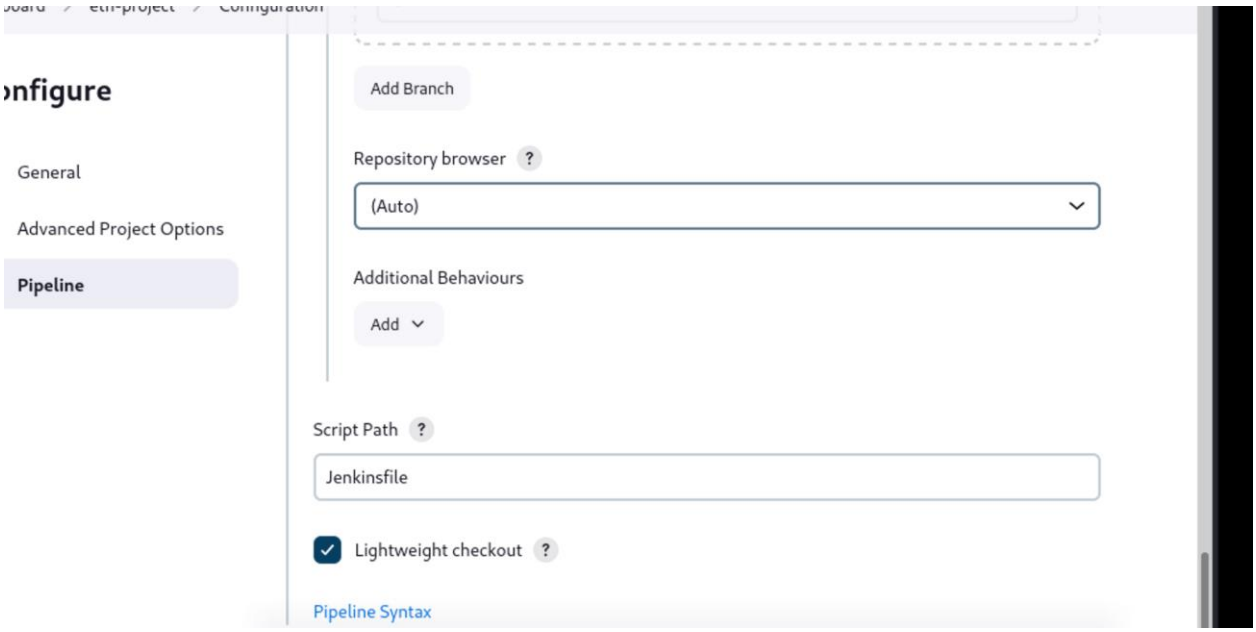


- Then move to the Pipeline section. Choose ***Pipeline script from SCM option.***
- Then choose ***SCM as Git.***
- In the repositories section, enter the repo URL in the github and add any credentials if it is private repo.
- Then add what branches to build and the path of the Jenkinsfile script.



Finally after doing all the above settings, the final configuration file looks something like this





- After setting up the above configurations correctly, then build the pipeline. After getting the successful pipeline build, it looks something likes this:

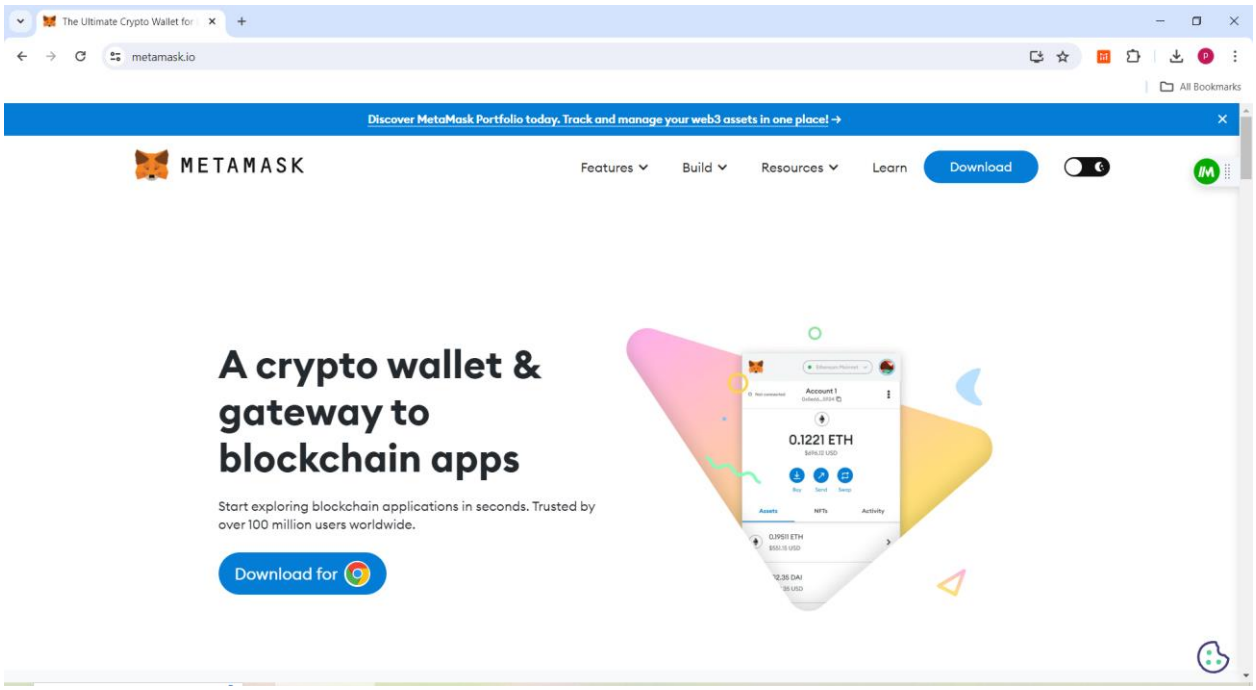
All the stages marked with green shows that pipeline got build without any conflict between packages and hence we were successful in doing that.

11 Testing and Results

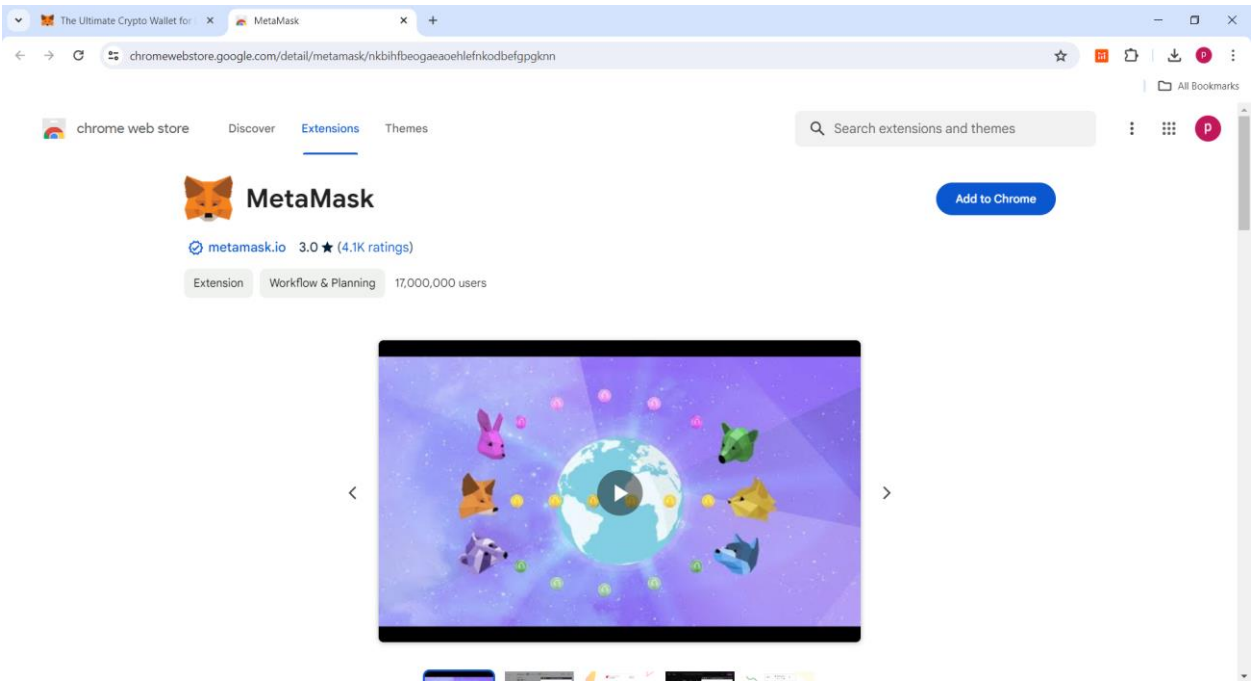
These are the some of the screen shots almost covers all major functionalities:

11.1 Setting up the Metamask1

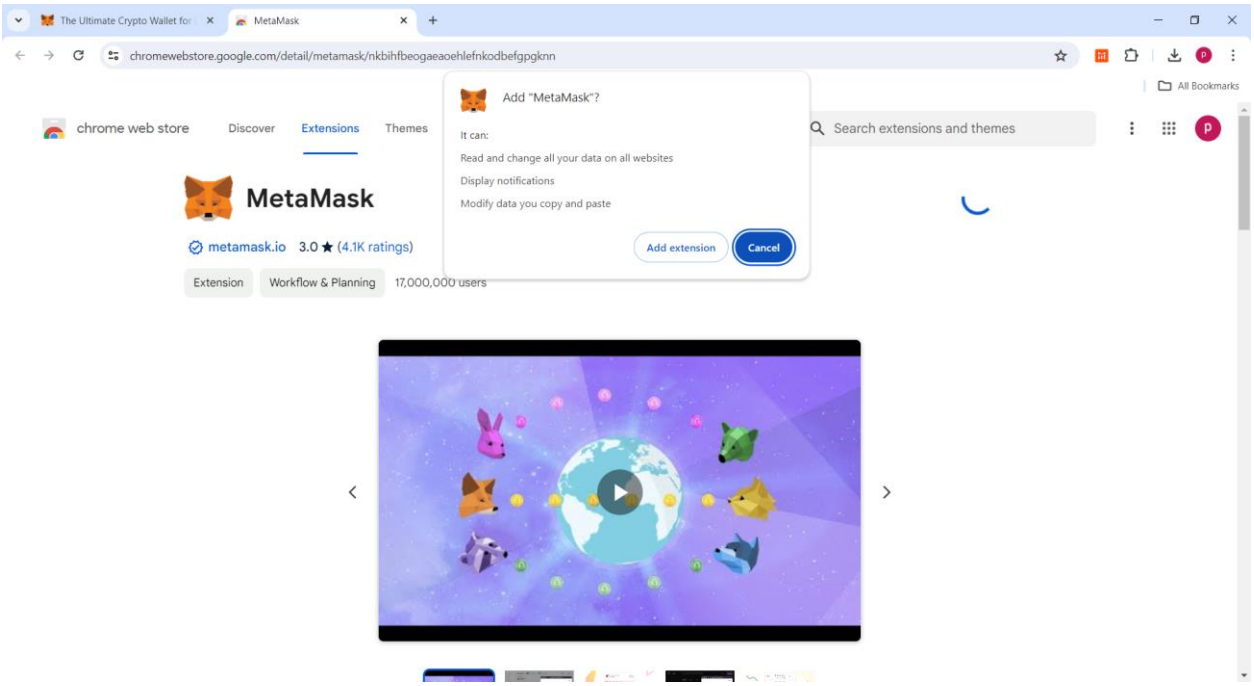
Open the browser and in the address bar at the top of the browser, type ‘metamask.io’ and press ‘Enter’.



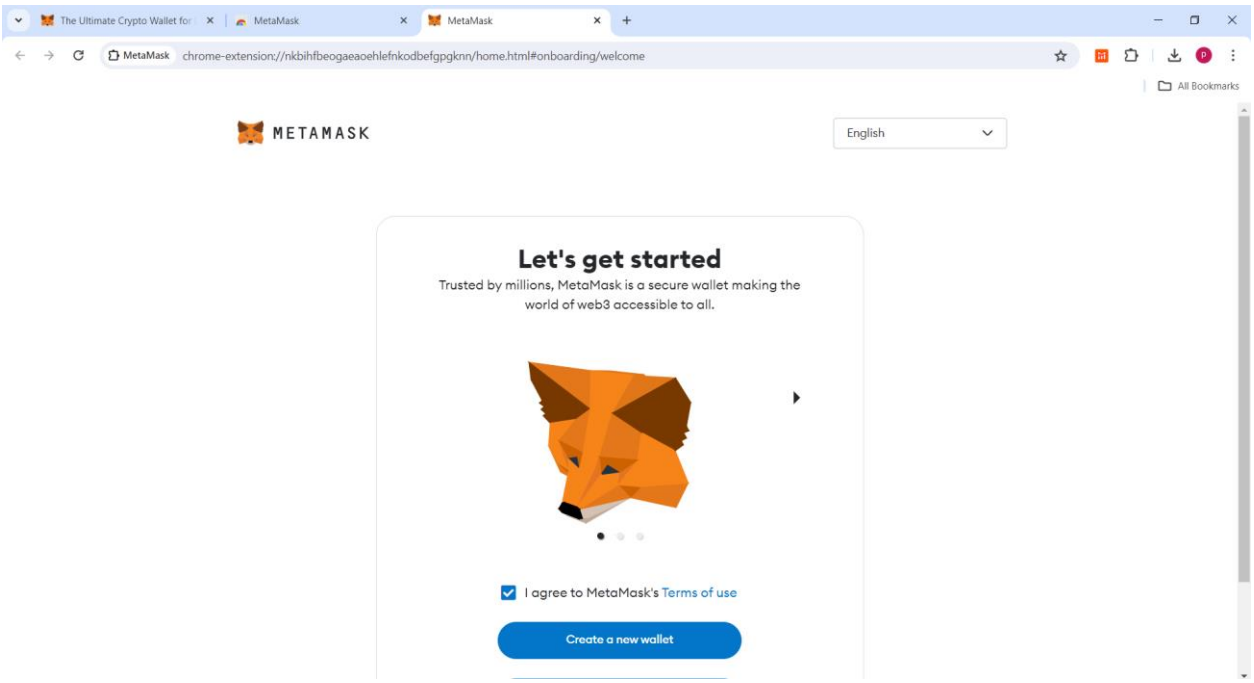
Click on ‘Download for chrome’



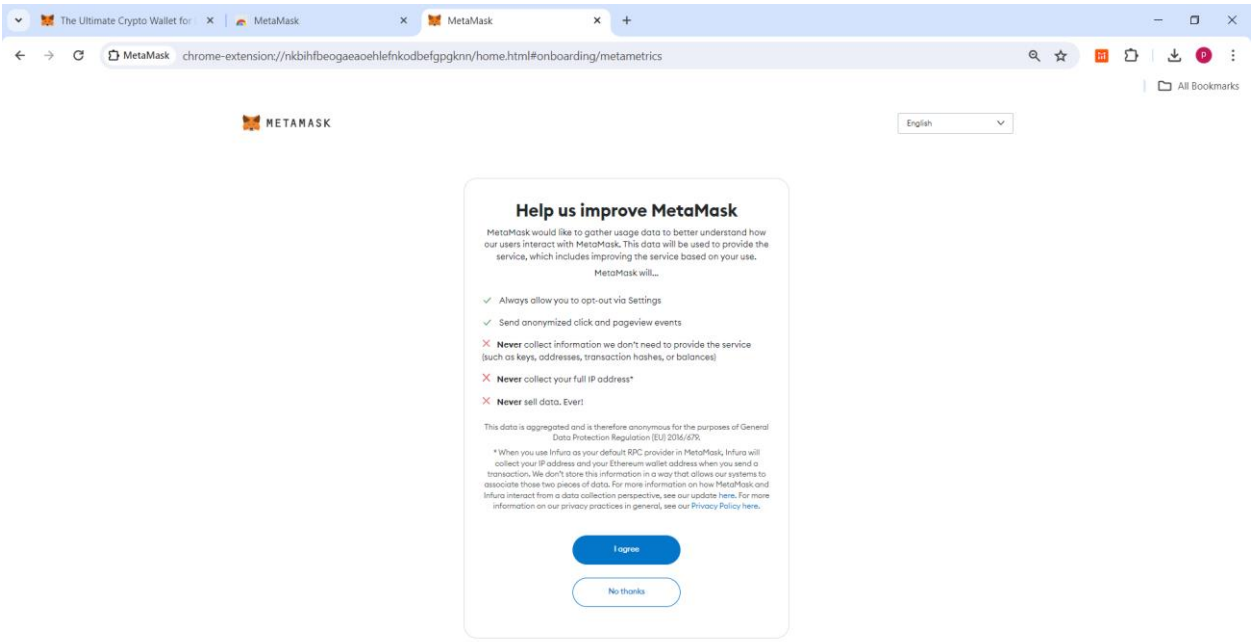
Click on ‘Add to chrome’



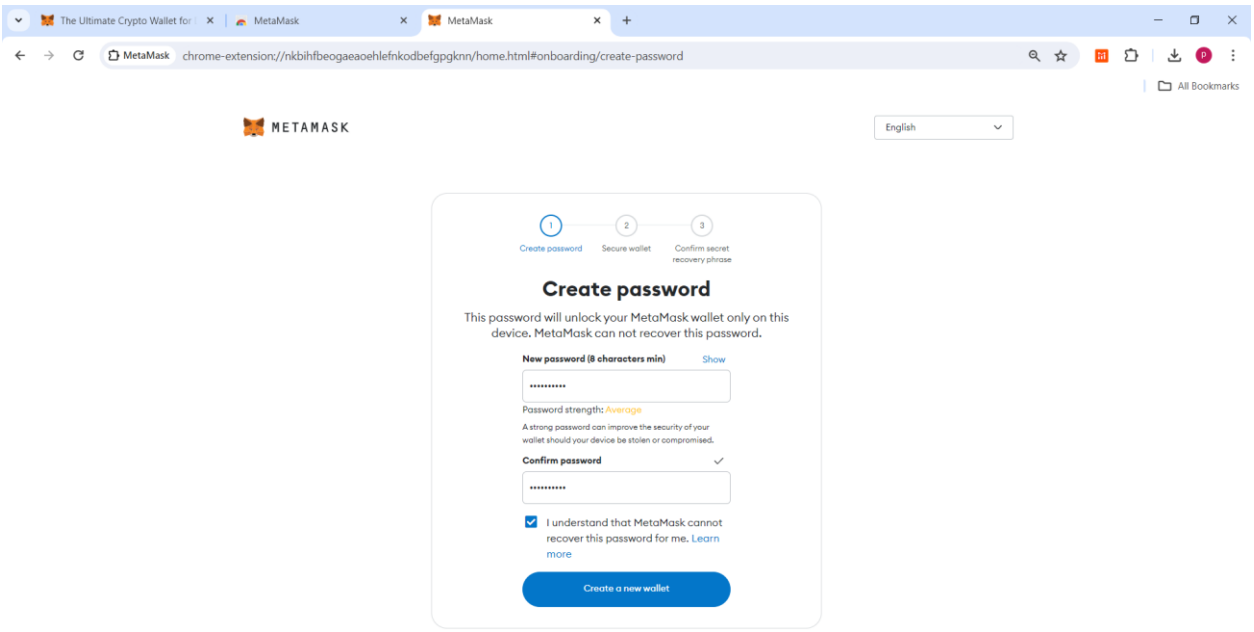
After downloading MetaMask, check the "I agree to MetaMask's Terms of use" box.



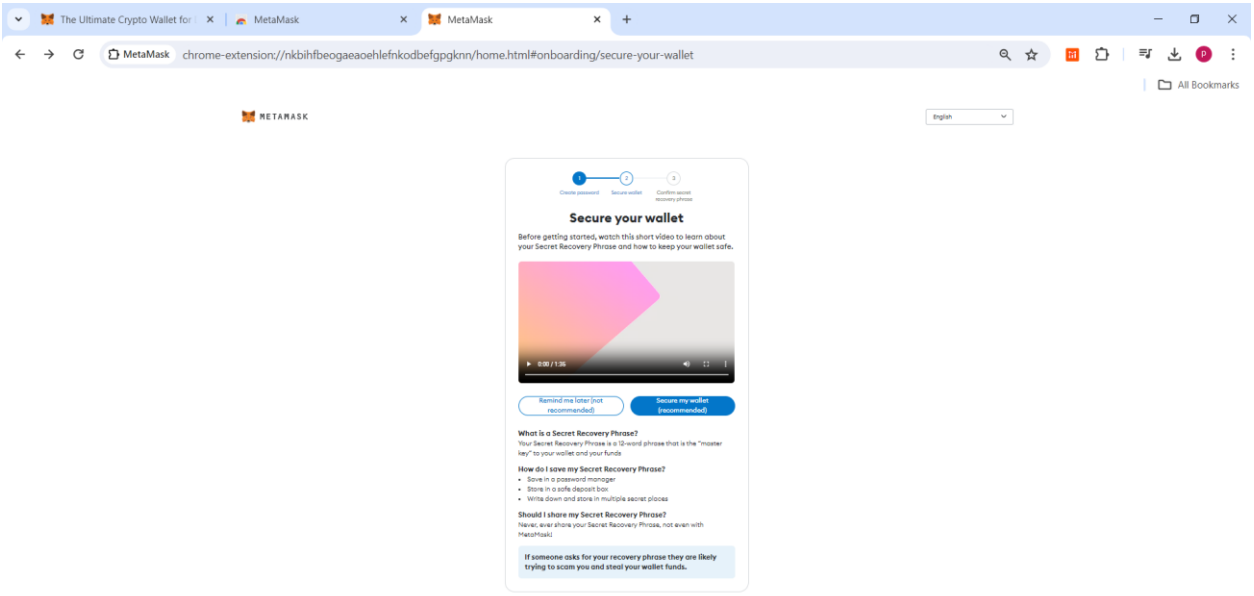
Click on ‘Create a new wallet’



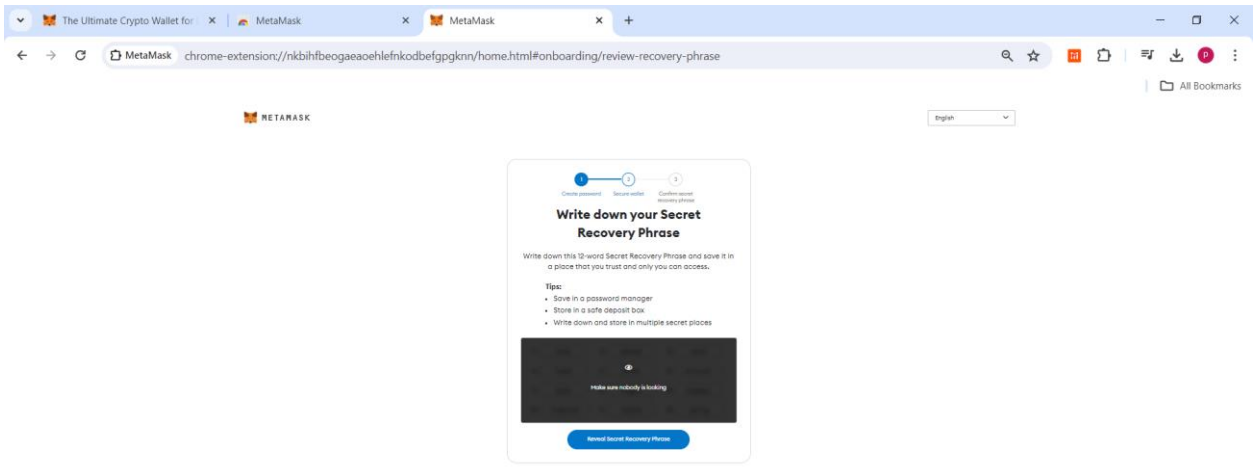
Click on ‘I agree’



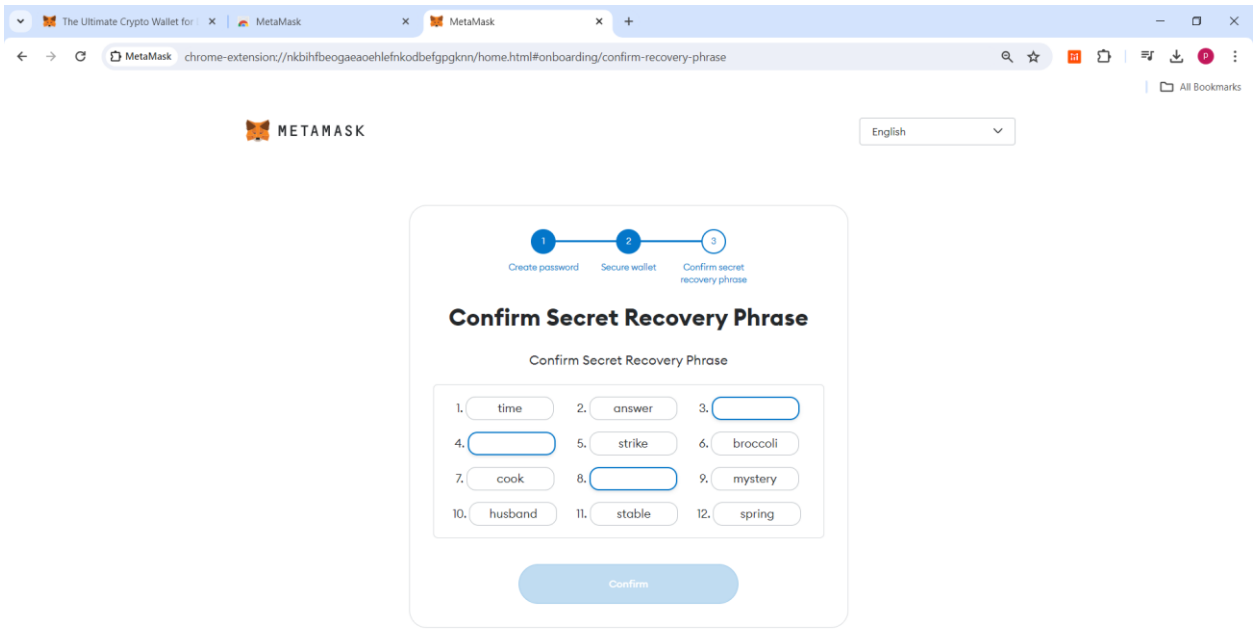
Fill in the details, create a password, check the "I understand that MetaMask cannot recover this password for me" box, then click "Create a new wallet."



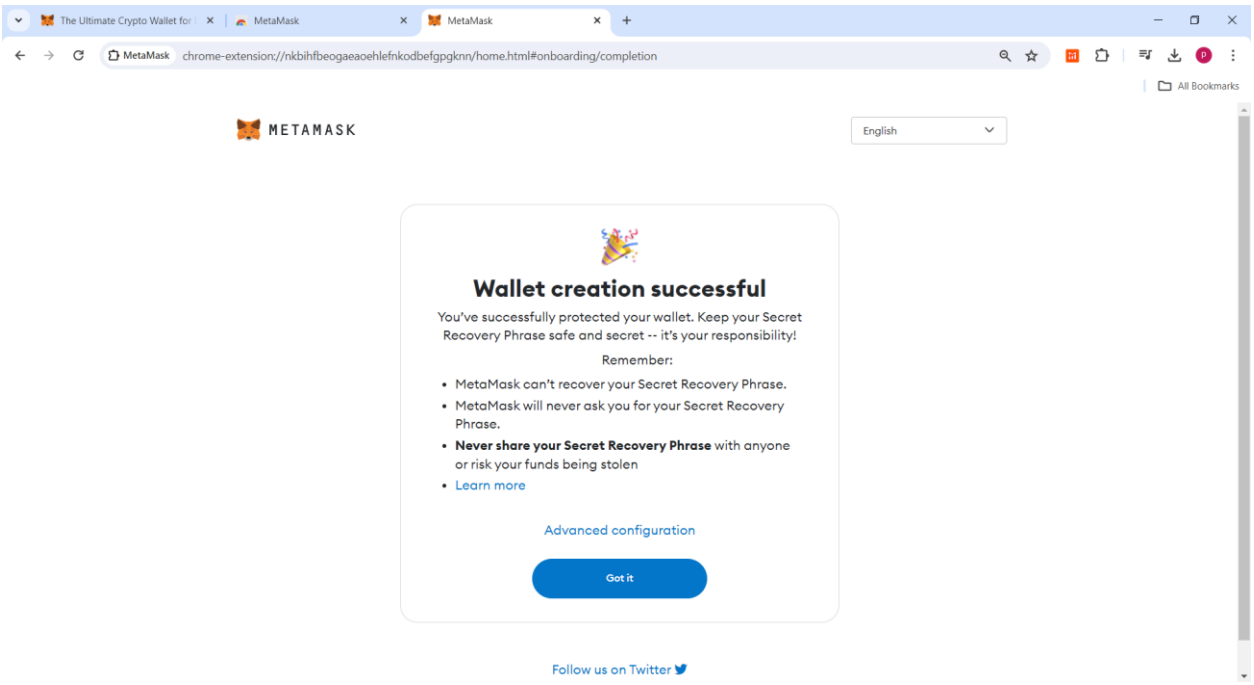
Click on ‘Secure my wallet’



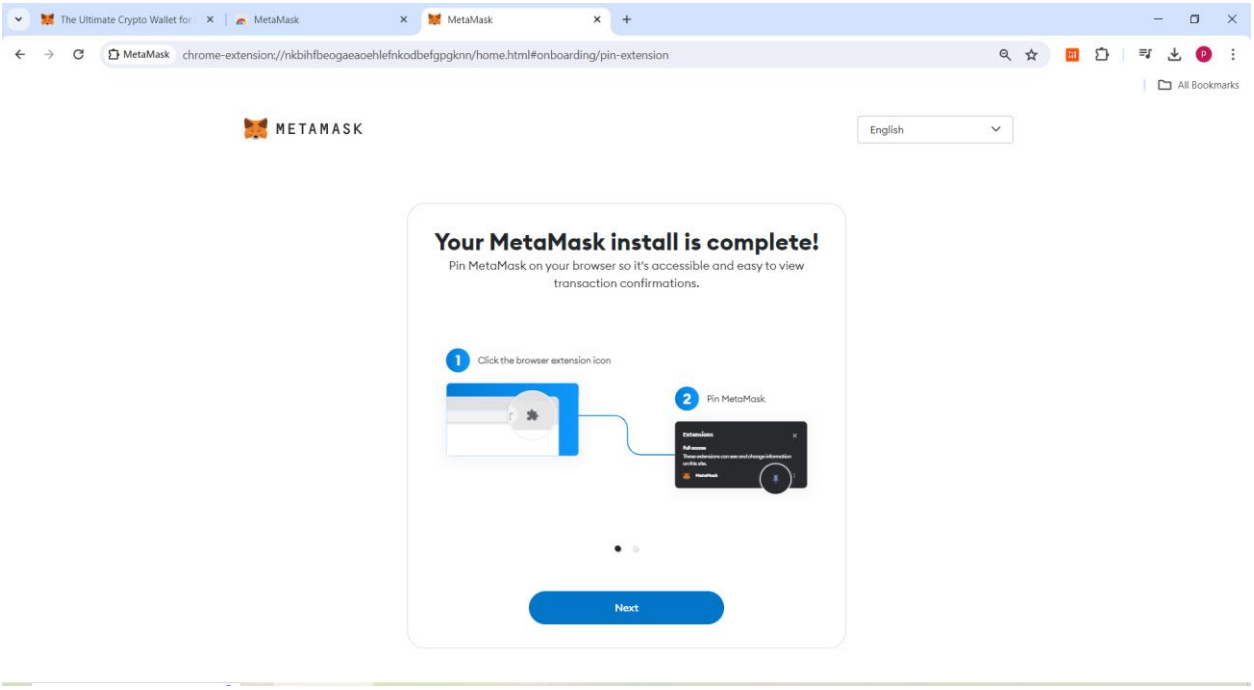
Click on ‘Reveal Secret Recovery Pharse ’ and the take the screenshot of the key then press ‘Next’.



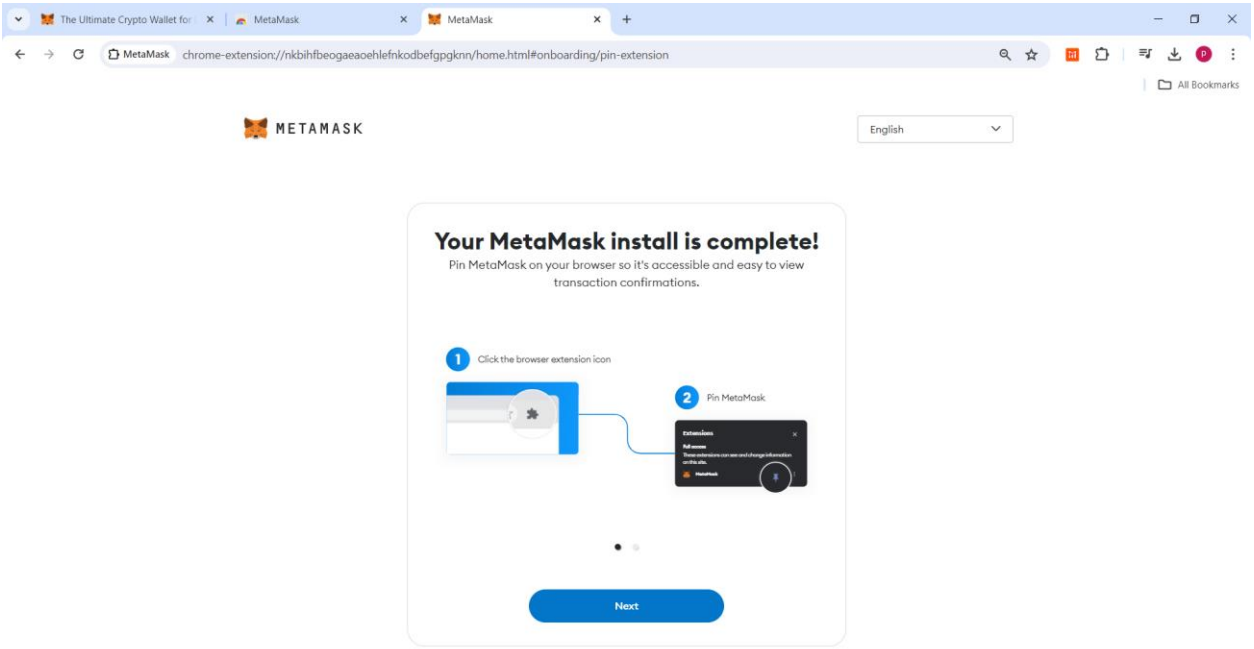
Fill the box as per the screenshot and the click on ‘Confirm’.



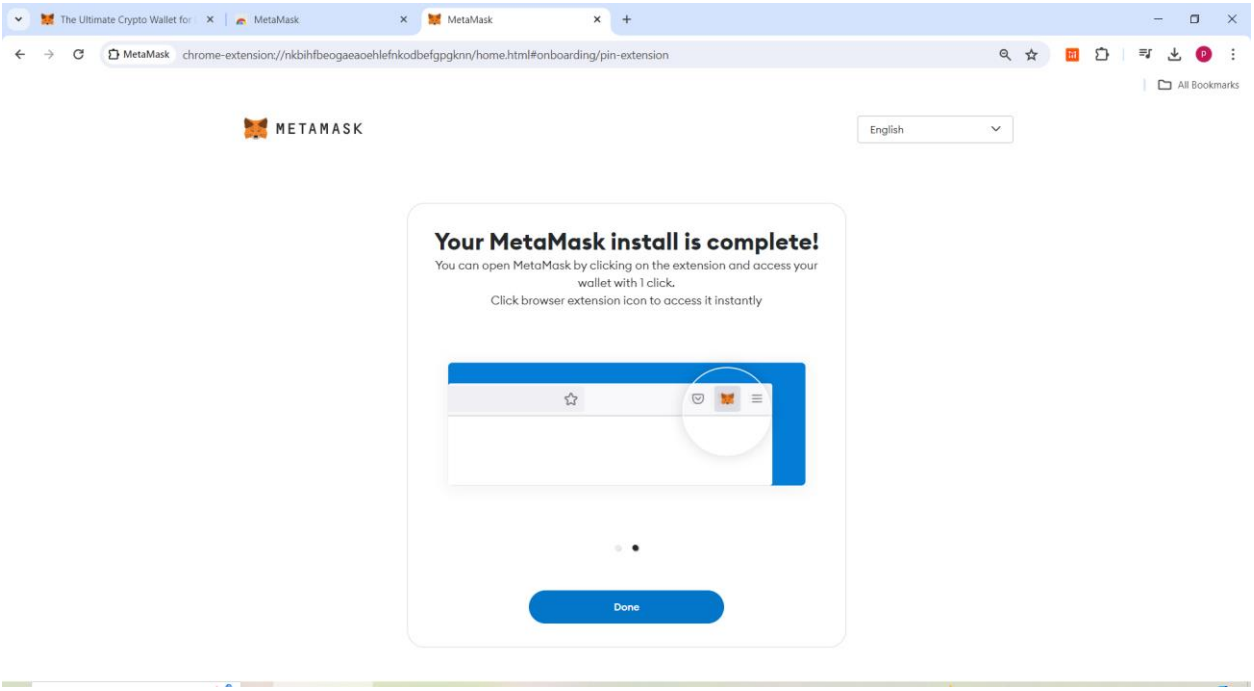
Click on ‘Got it’



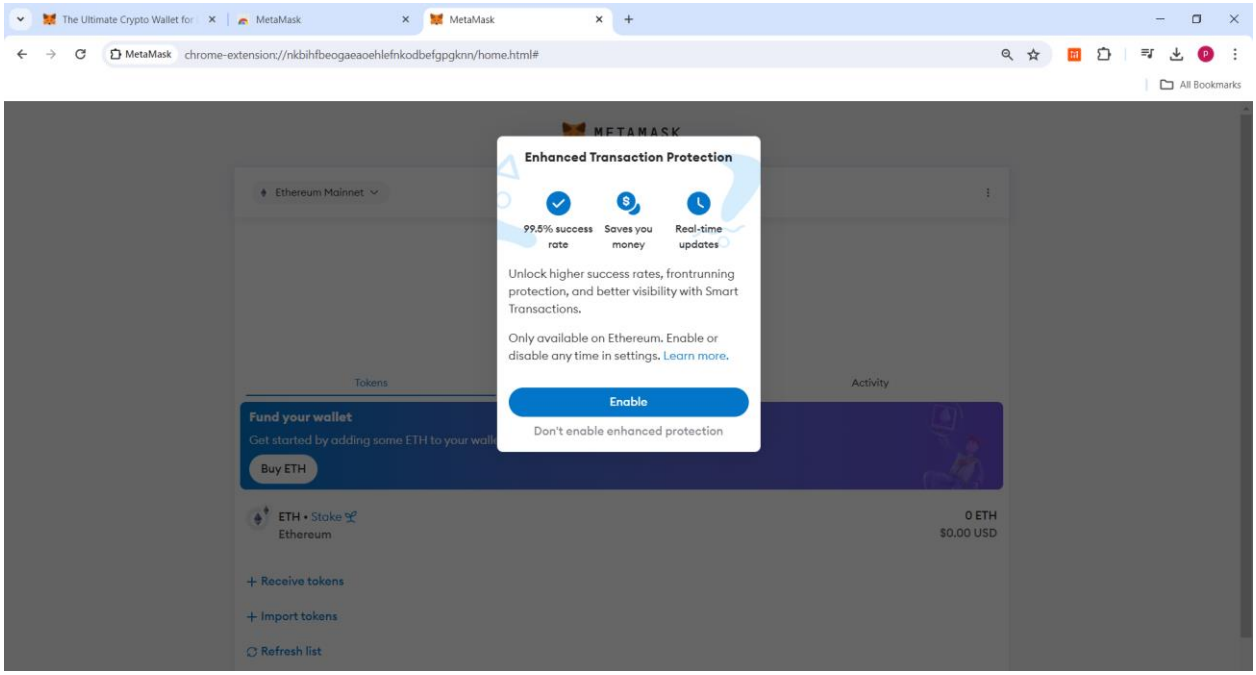
Click on ‘Next’



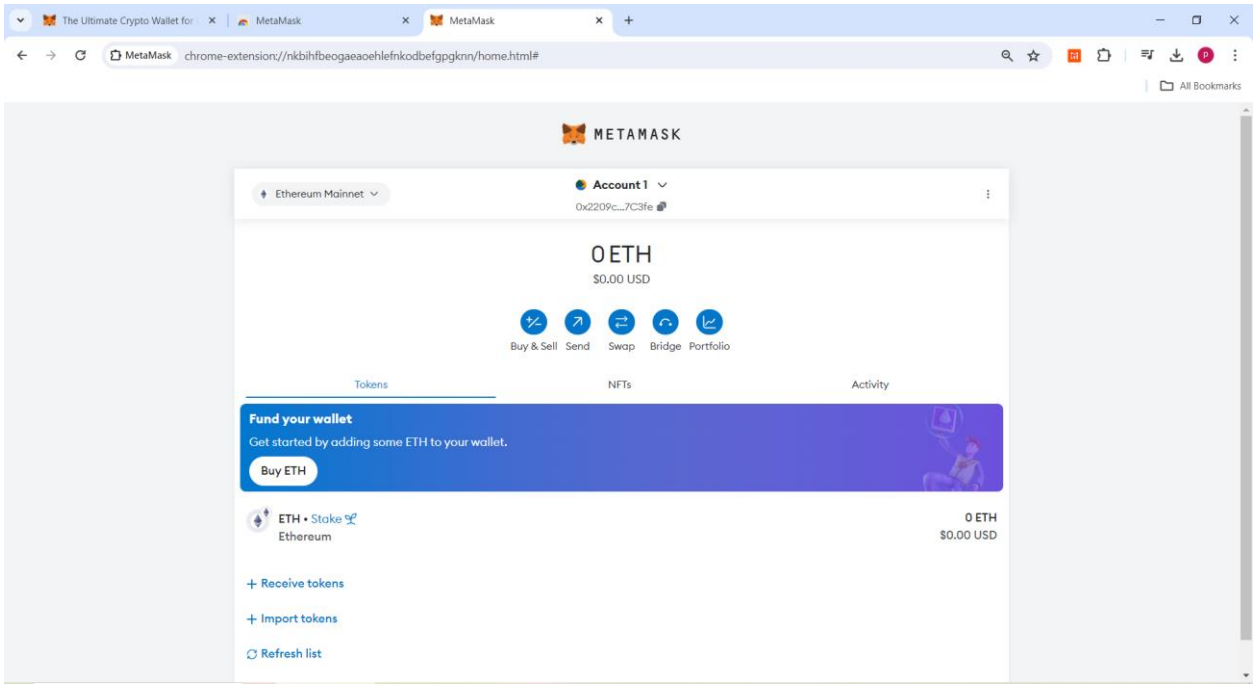
Click on ‘Done’



Click on ‘Enable’

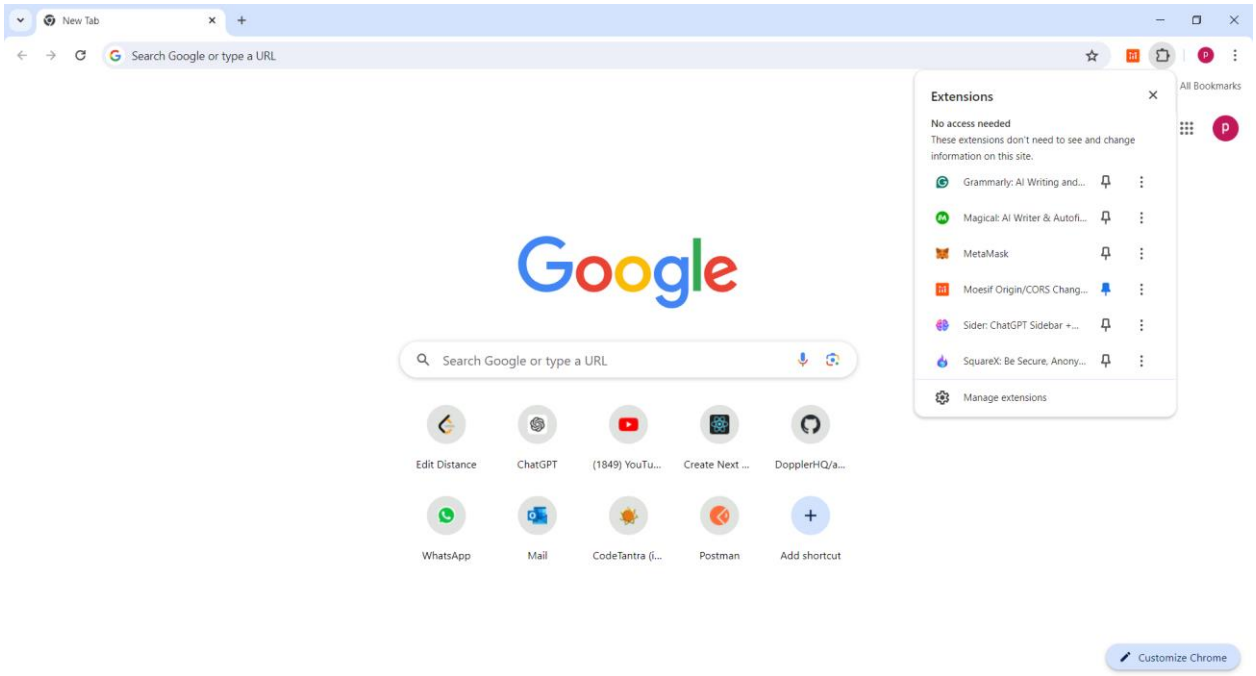


Metamask setup successfully.

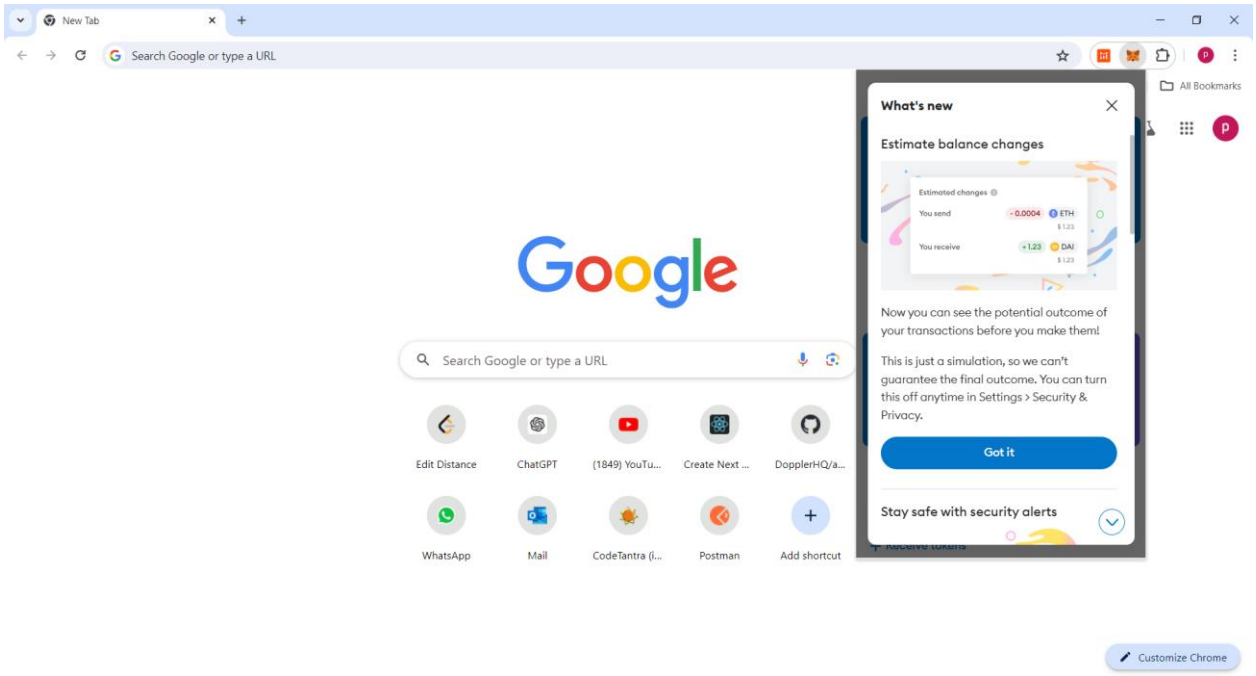


11.2 Setting up Server in Metamask

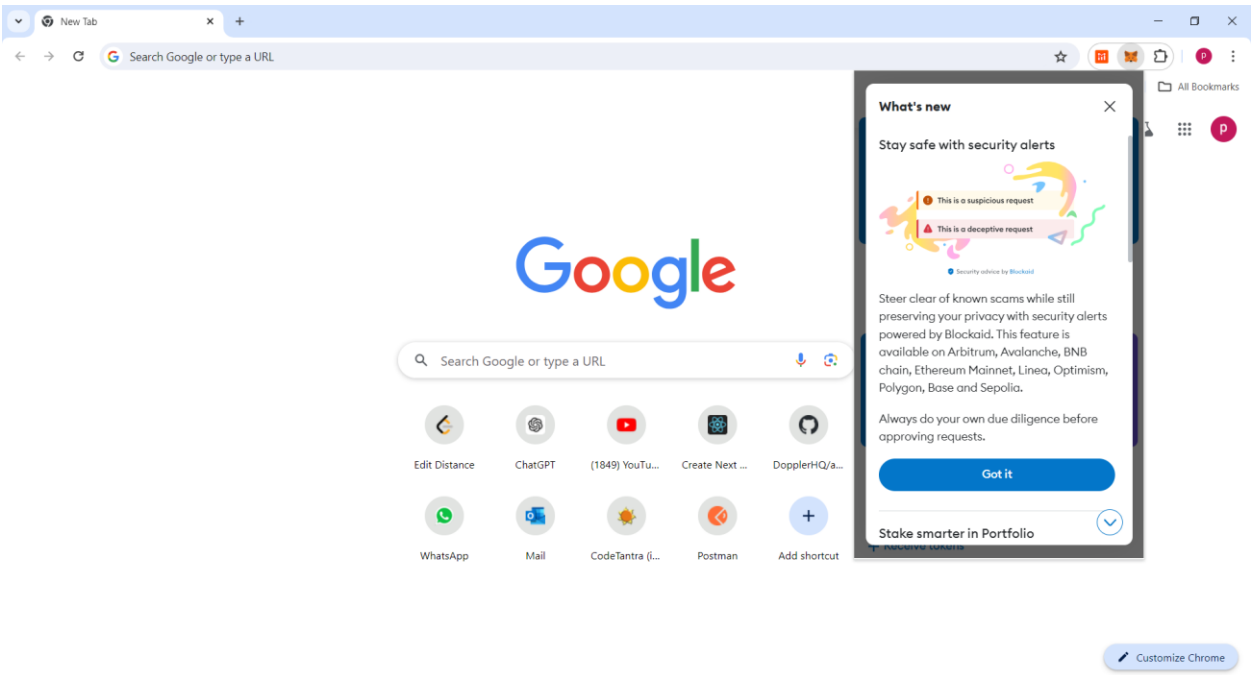
Open the browser and click on the ‘extension’



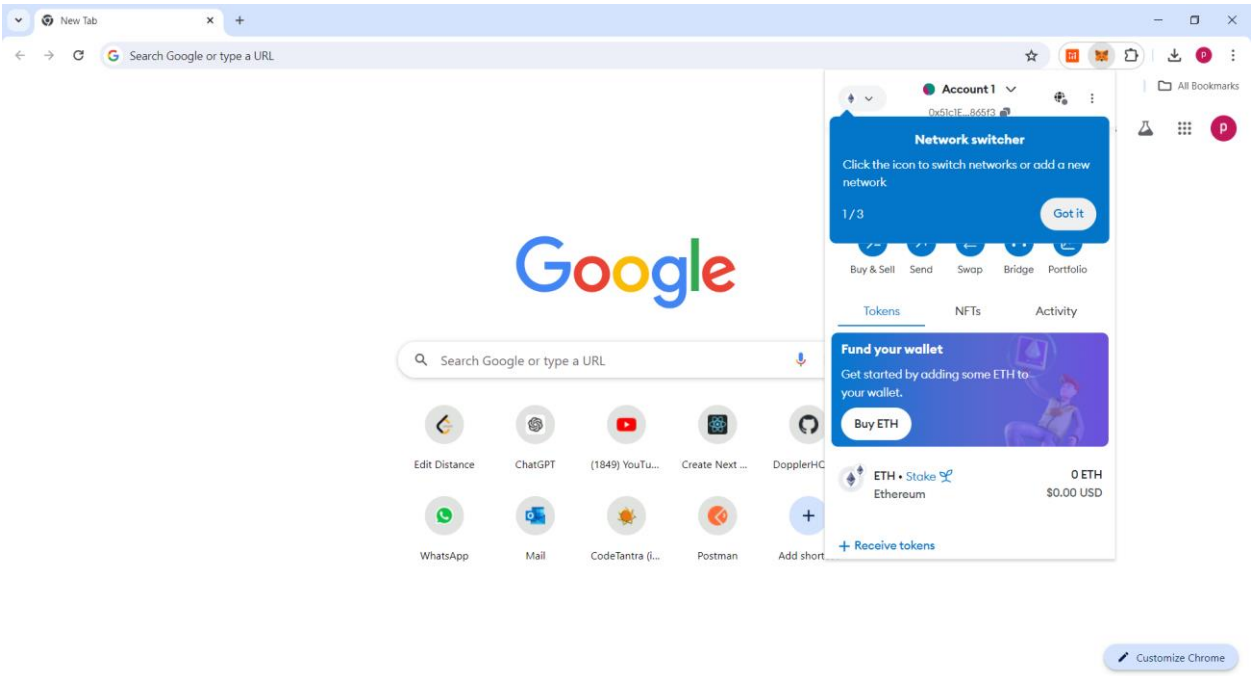
Click on the ‘Metamask’



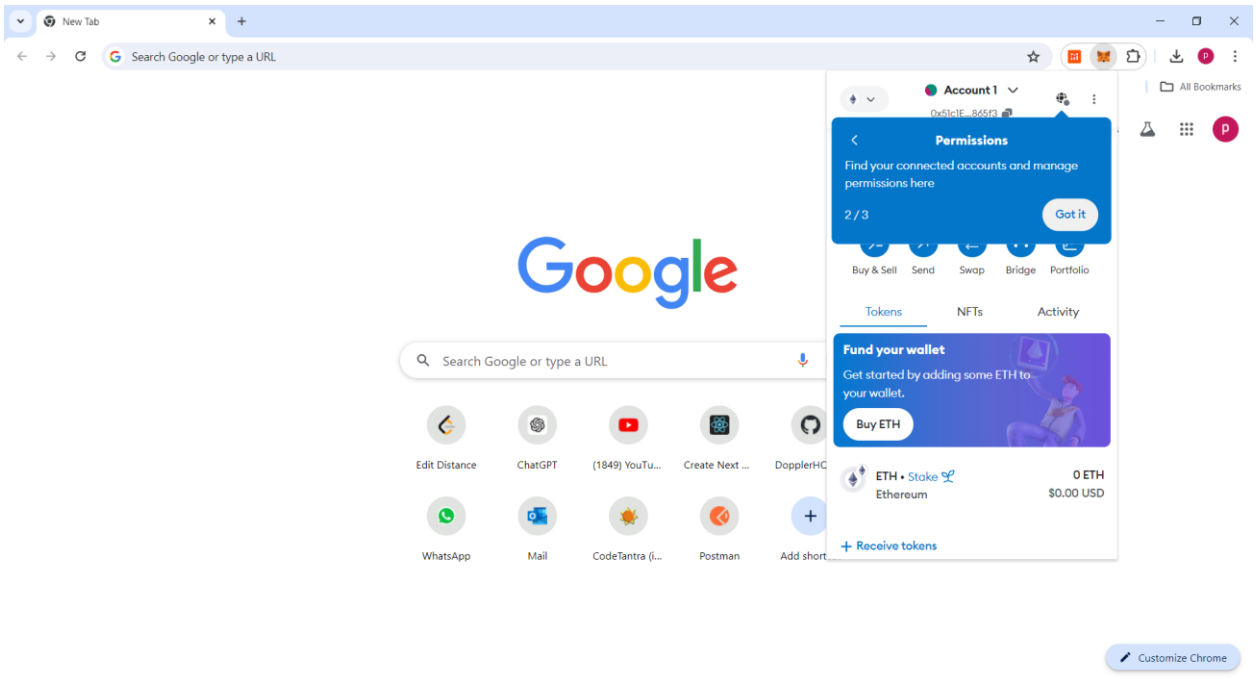
Click on ‘x’



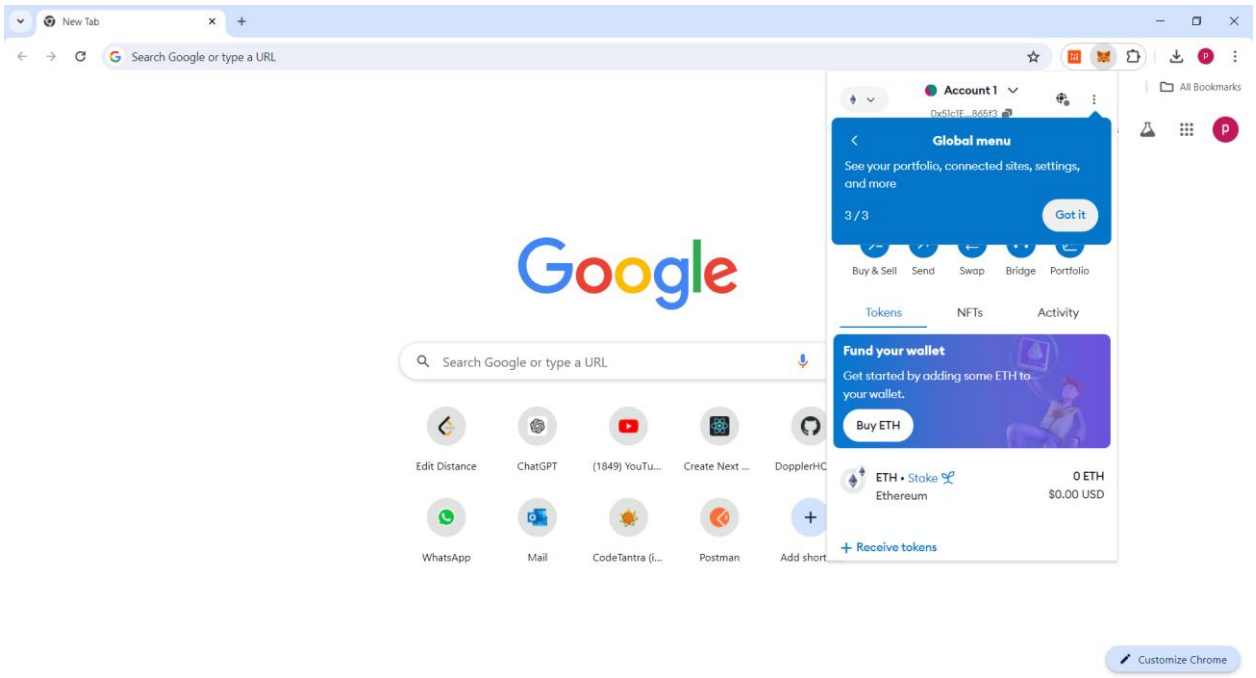
Click on ‘Got it’

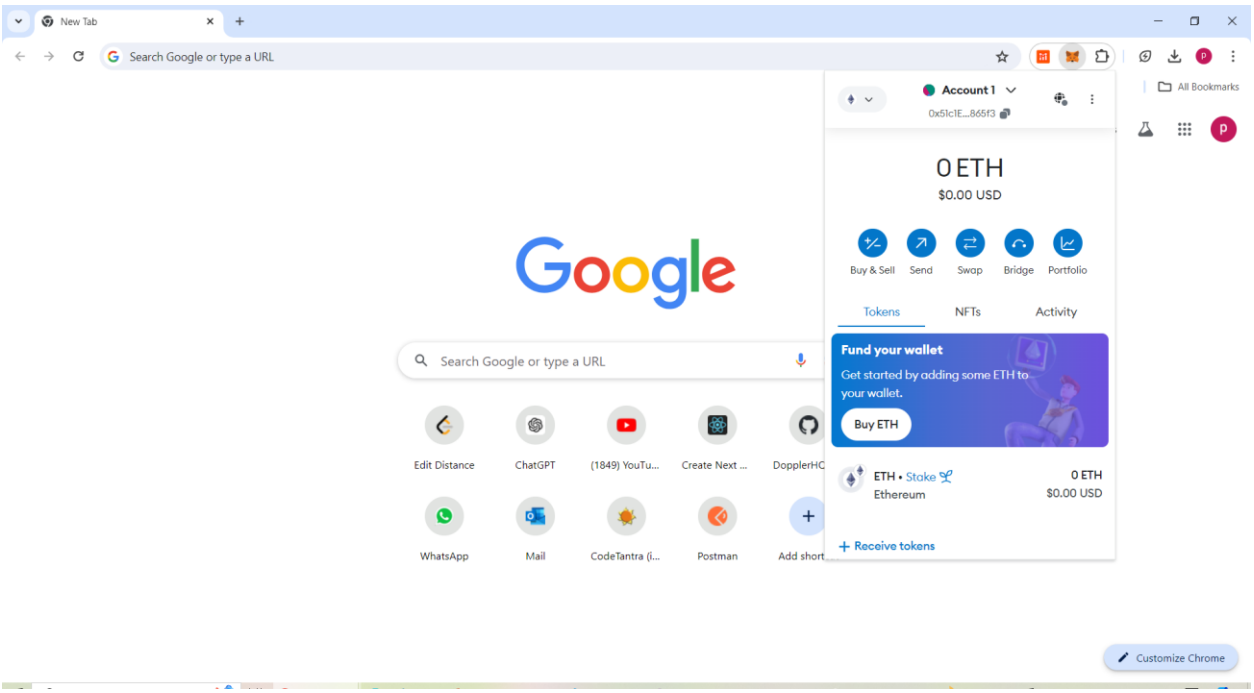


Again click on ‘Got it’

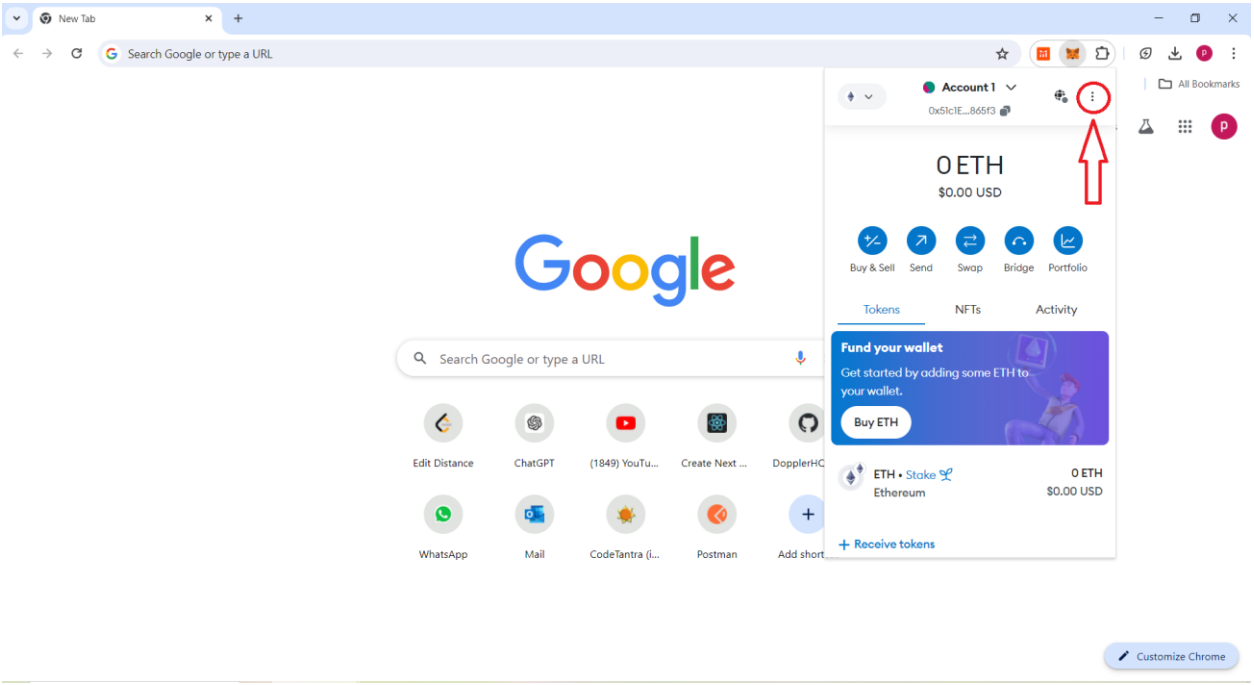


Again click on ‘Got it’

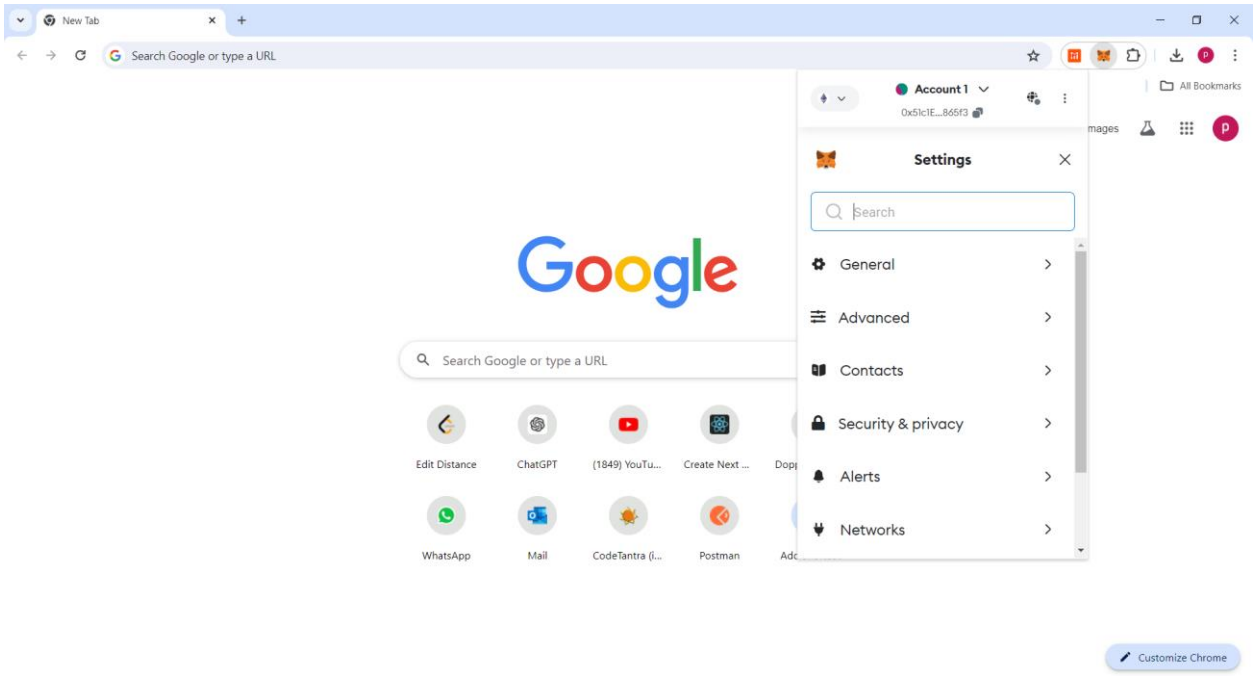




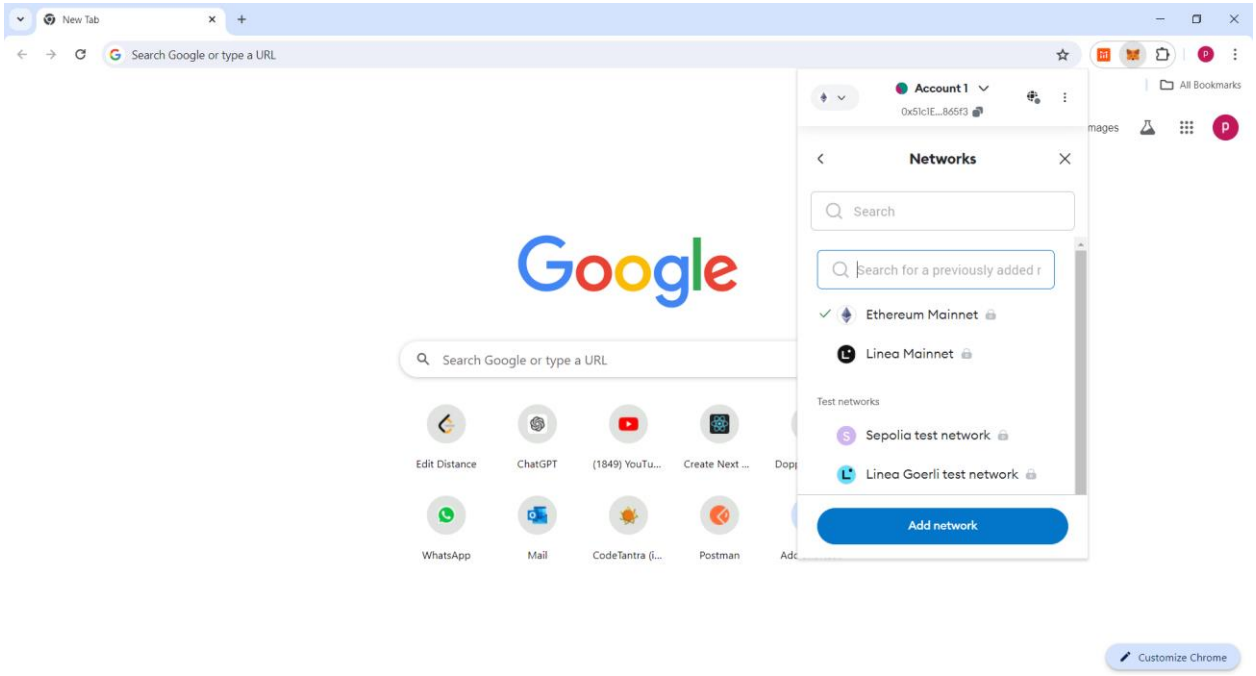
Click on ‘three dot’ that are marked in the figure



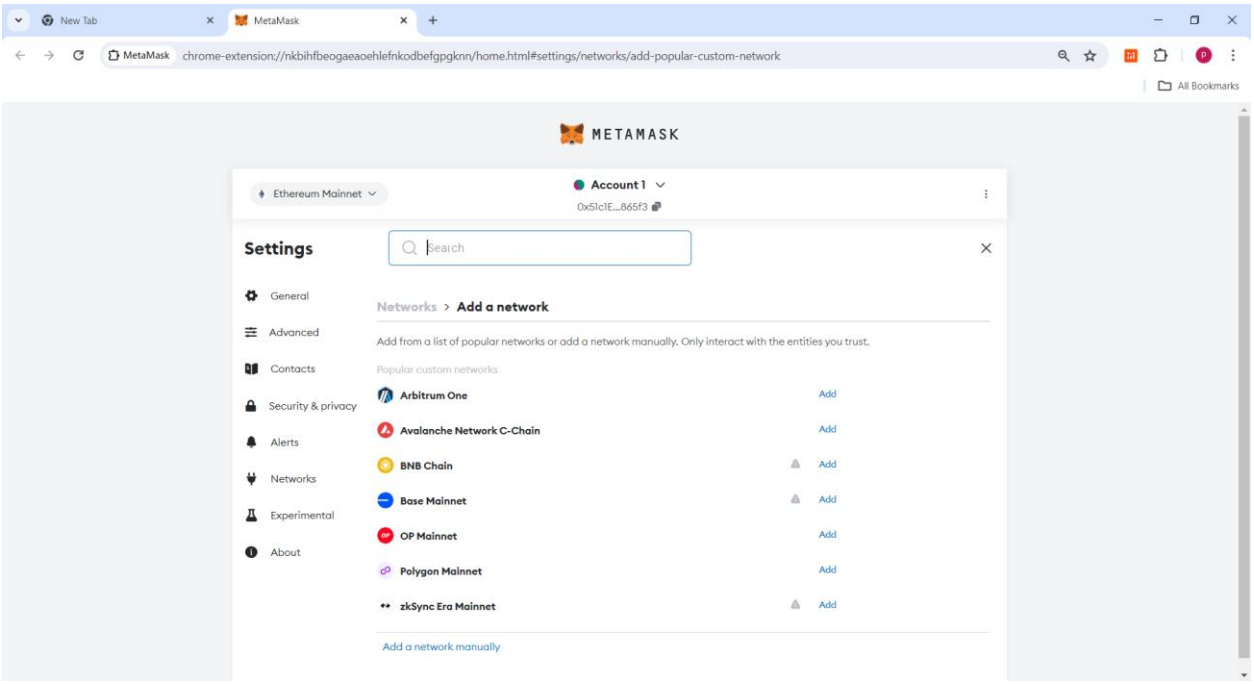
Click on ‘Settings’



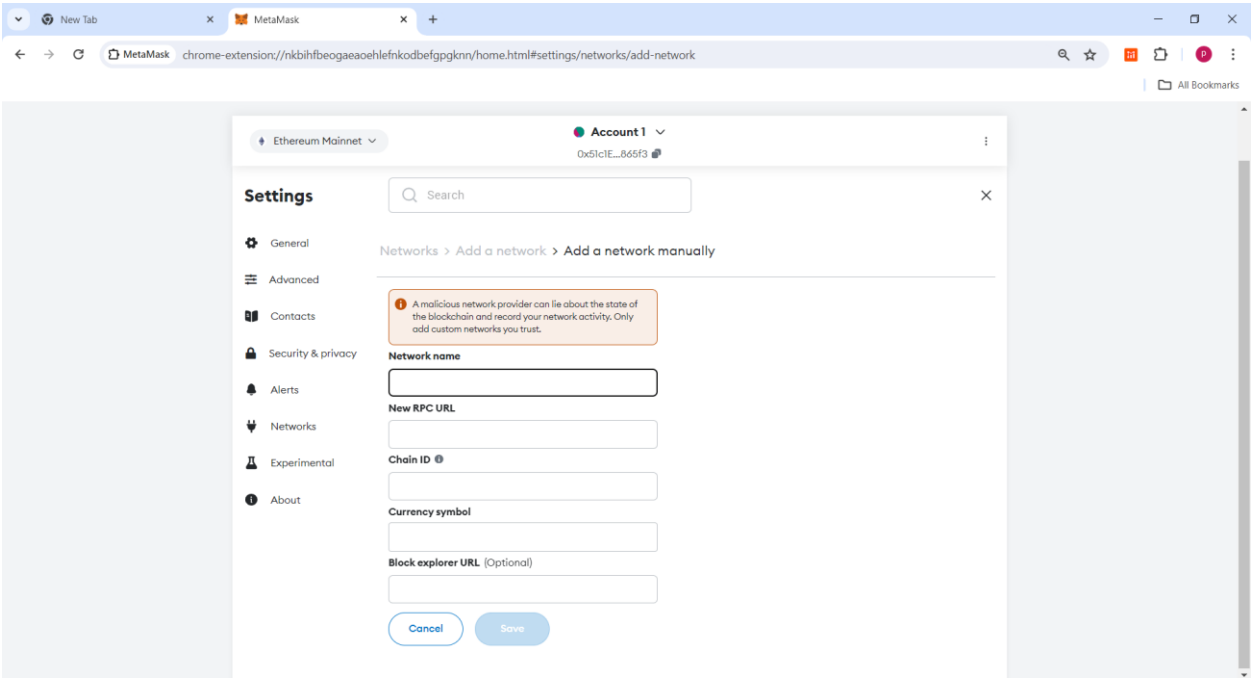
Click on ‘Networks’



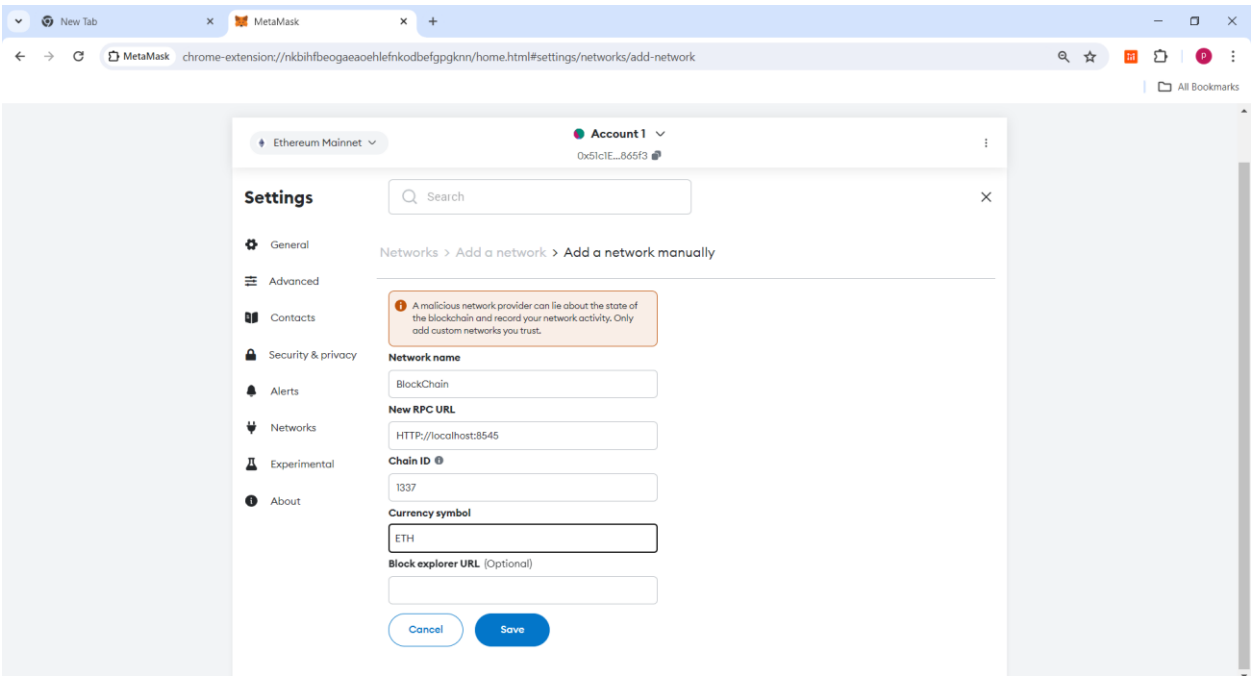
Click on ‘Add network’



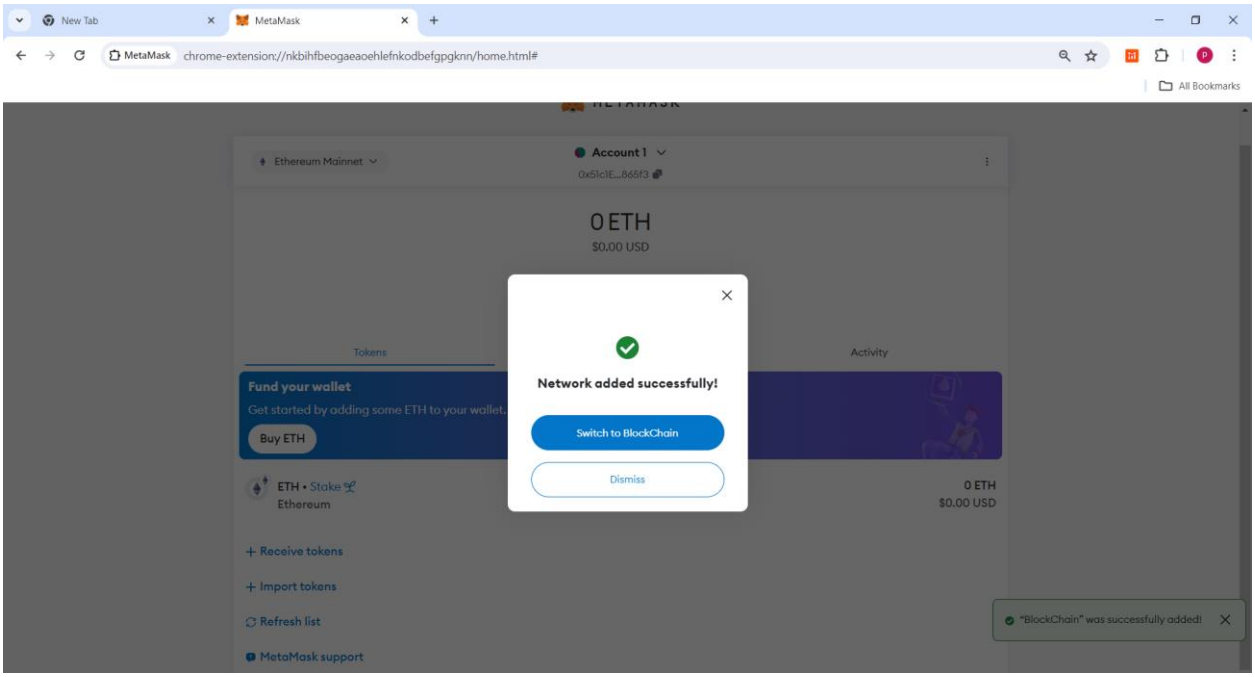
Click on ‘Add a network manually’



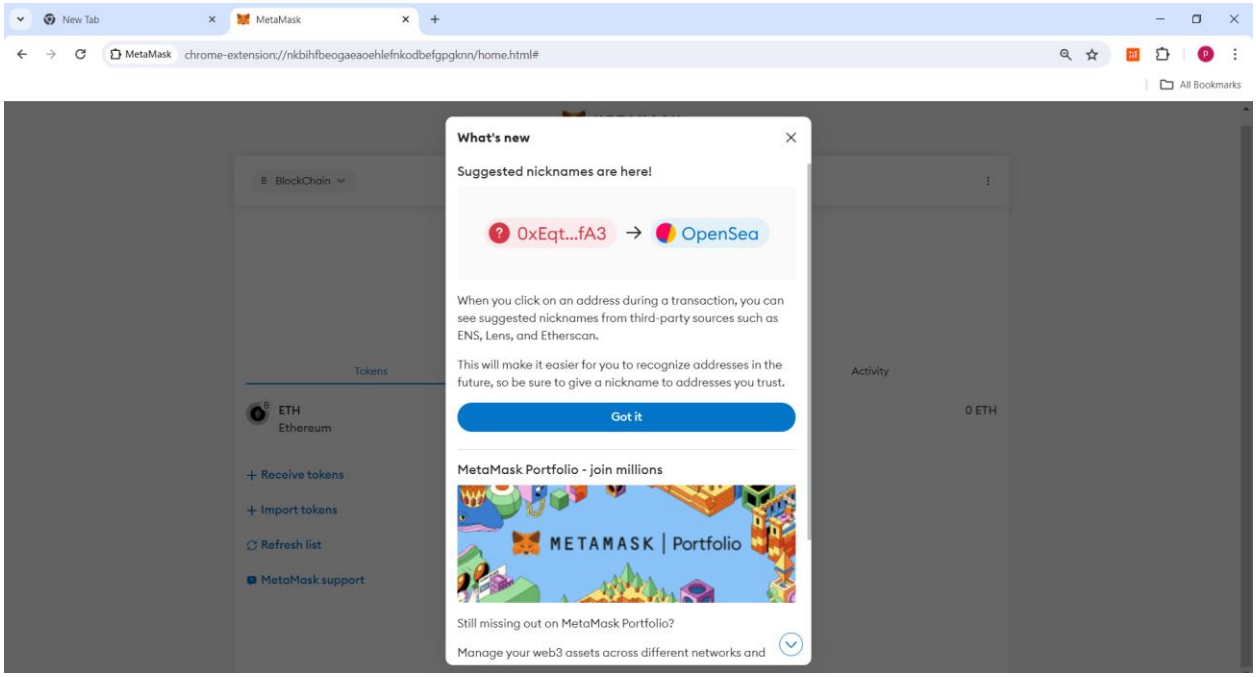
Fill all the details as shown below in the picture and before filling make sure that all the container are running.



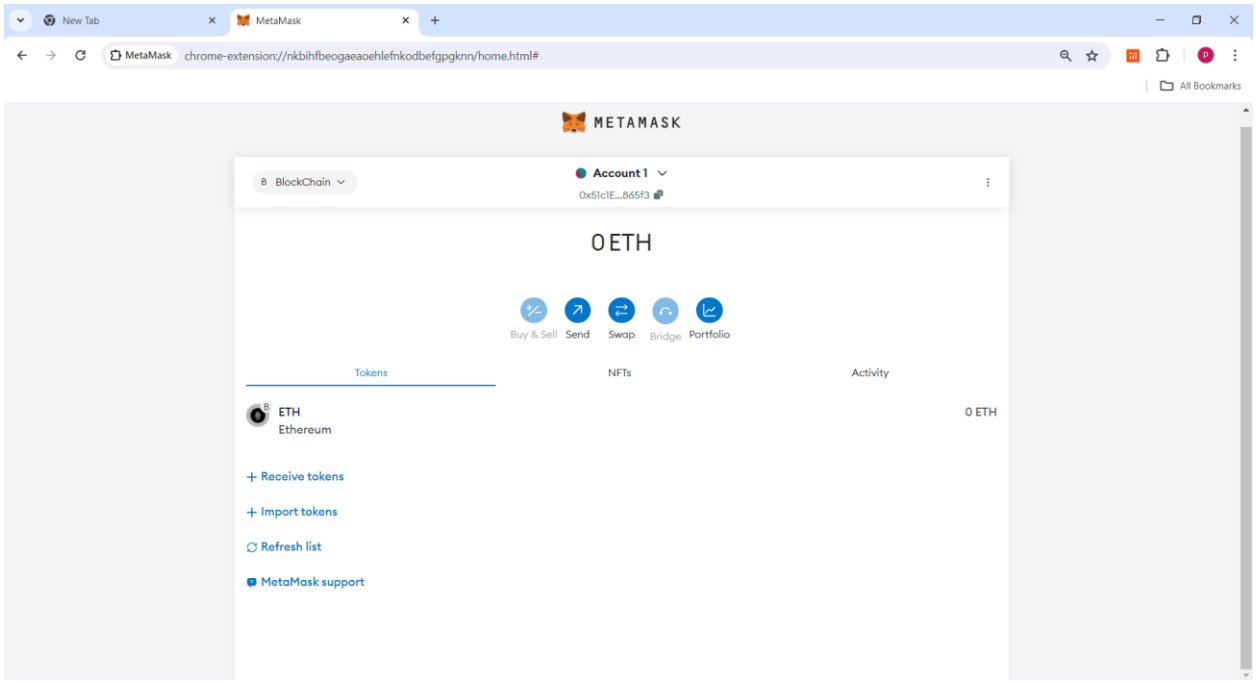
Click on ‘Save’



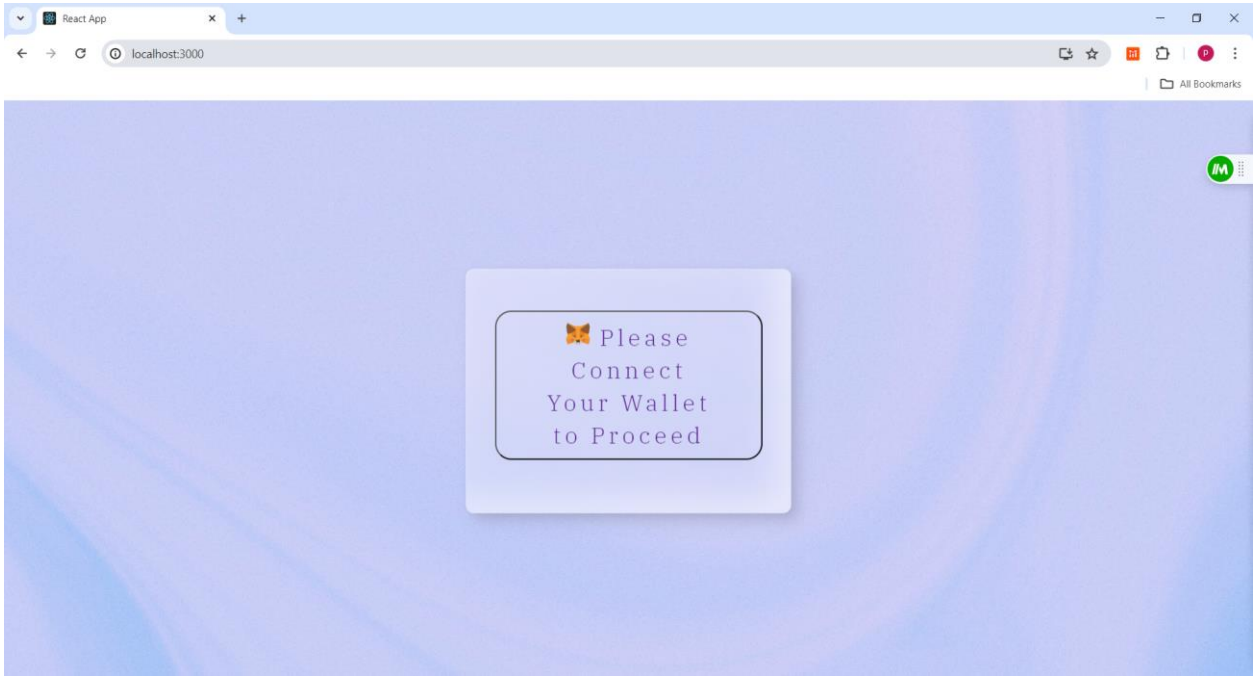
Click on ‘Switch to BlockChain’



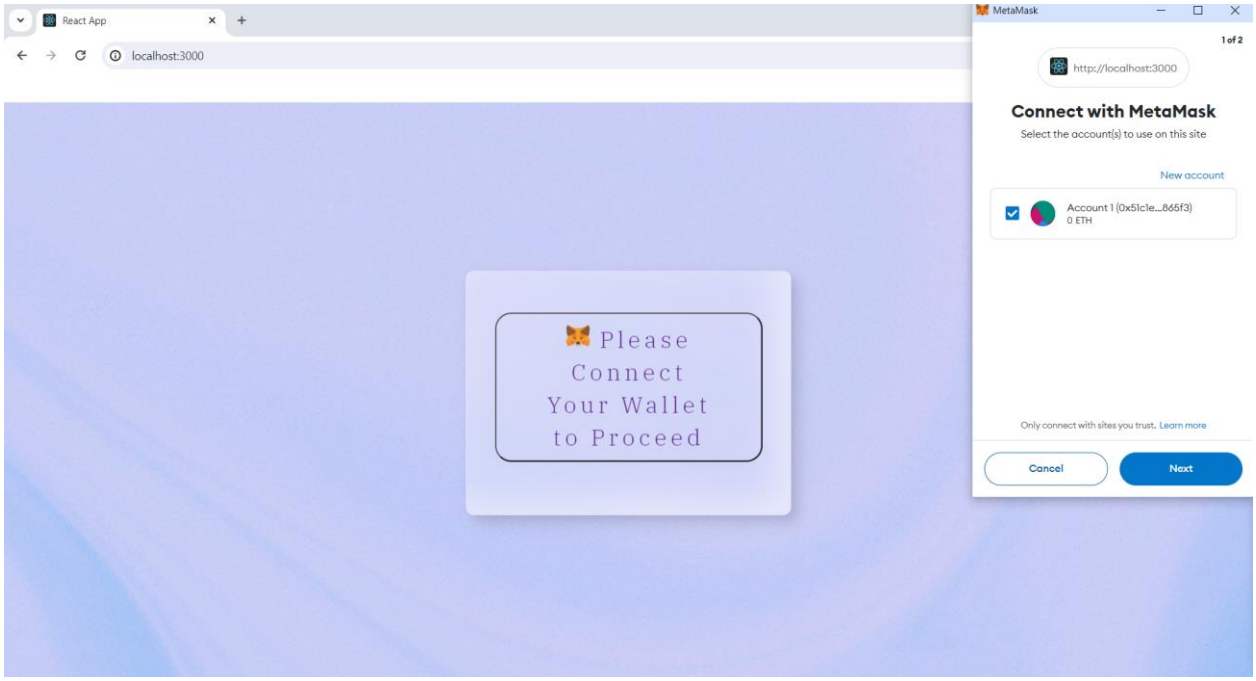
Click on ‘x’ and server setup completed.



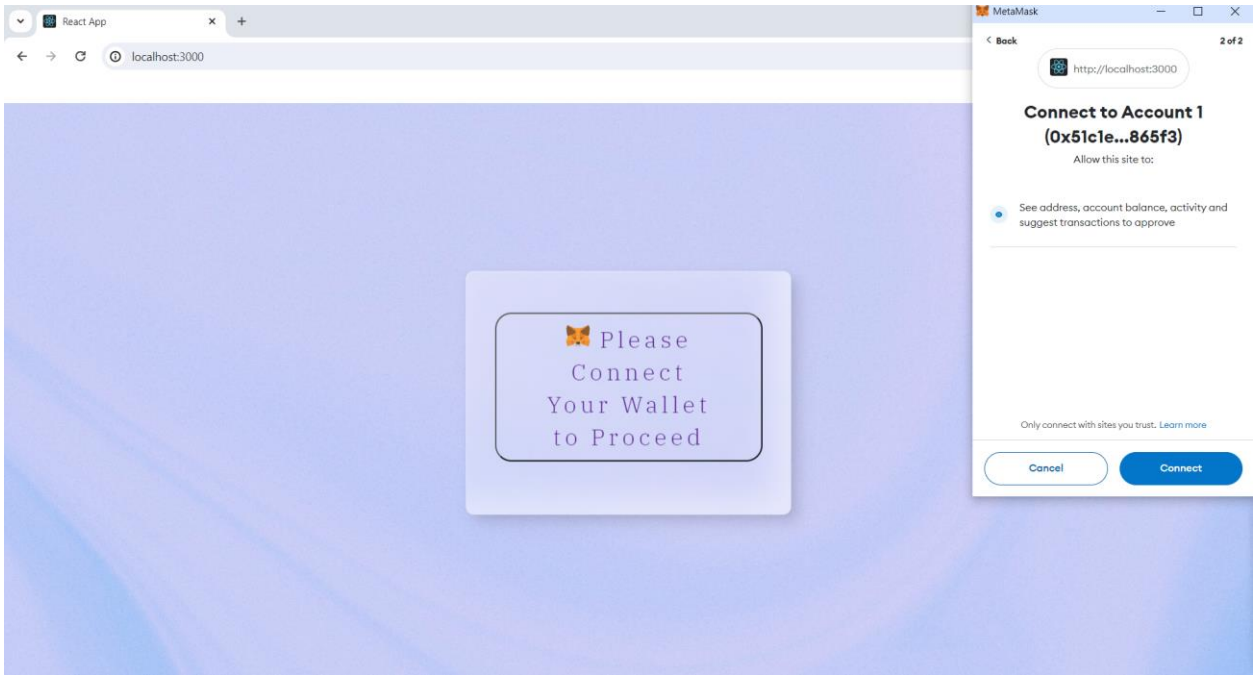
Now open the application URL ‘http://localhost:3000/’



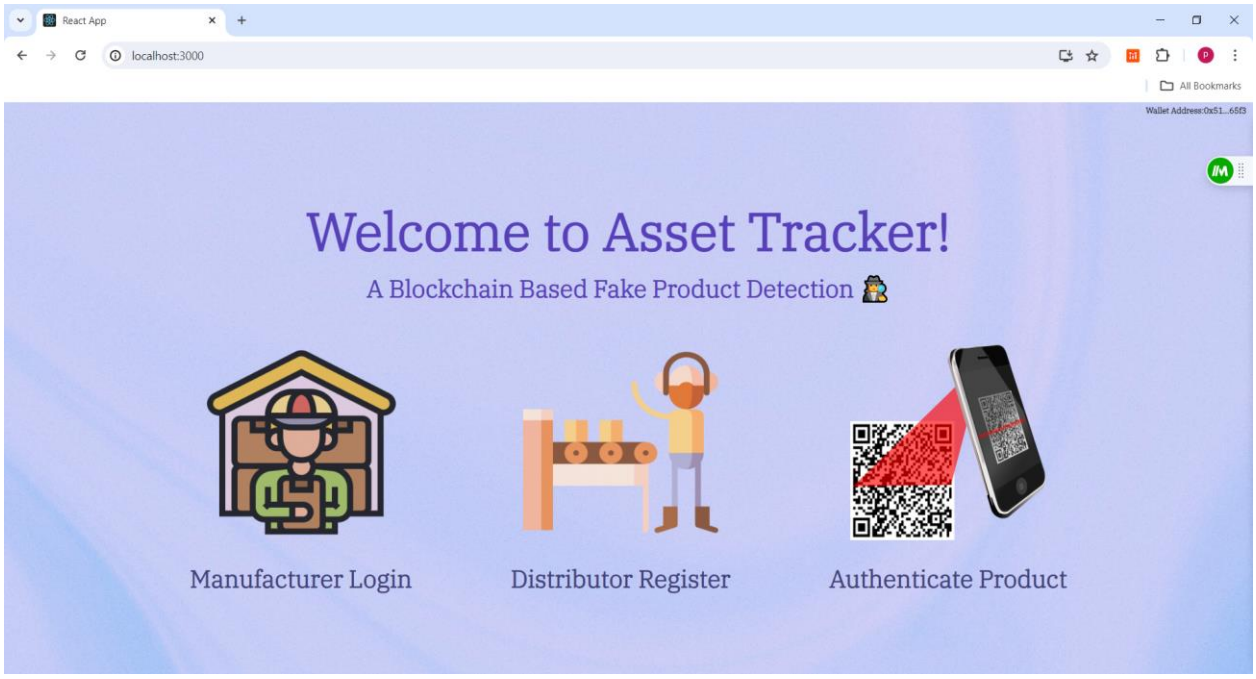
Click on ‘Please Connect Your Wallet to Proceed’



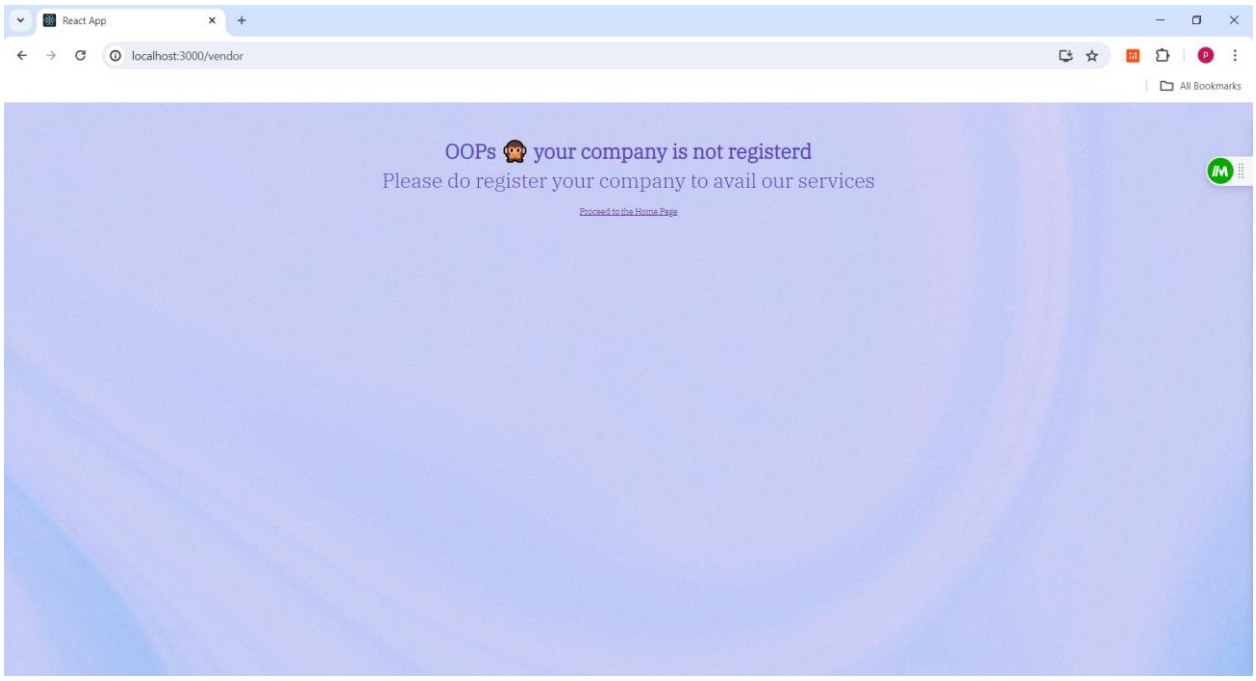
Click on ‘Next’



Click on ‘Connect’

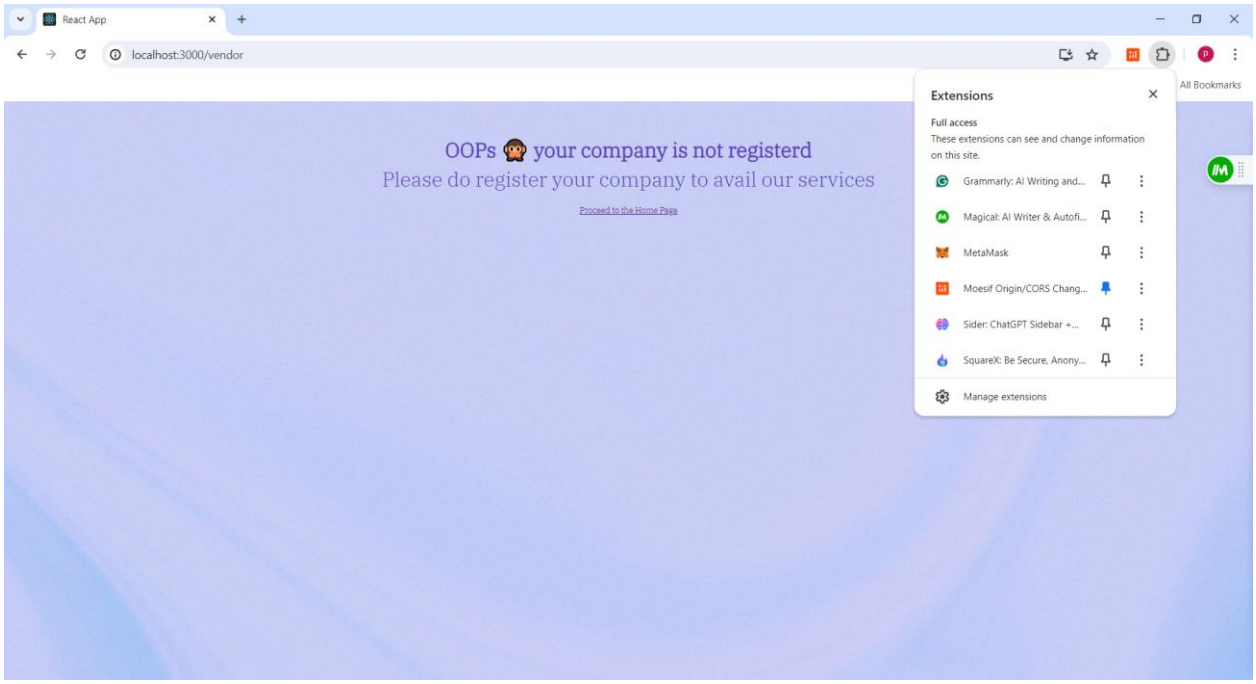


Click on ‘Manufacturer Login’

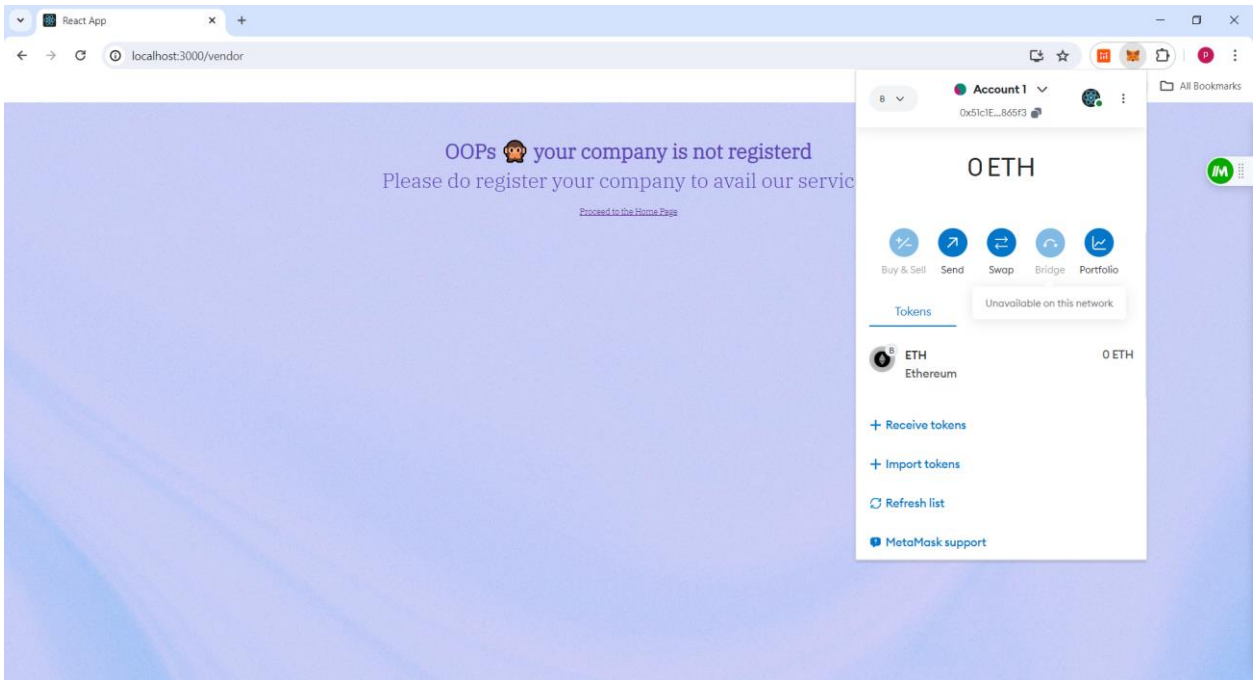


Since the given account is not registered, we will add the registered account of the manufacturer.

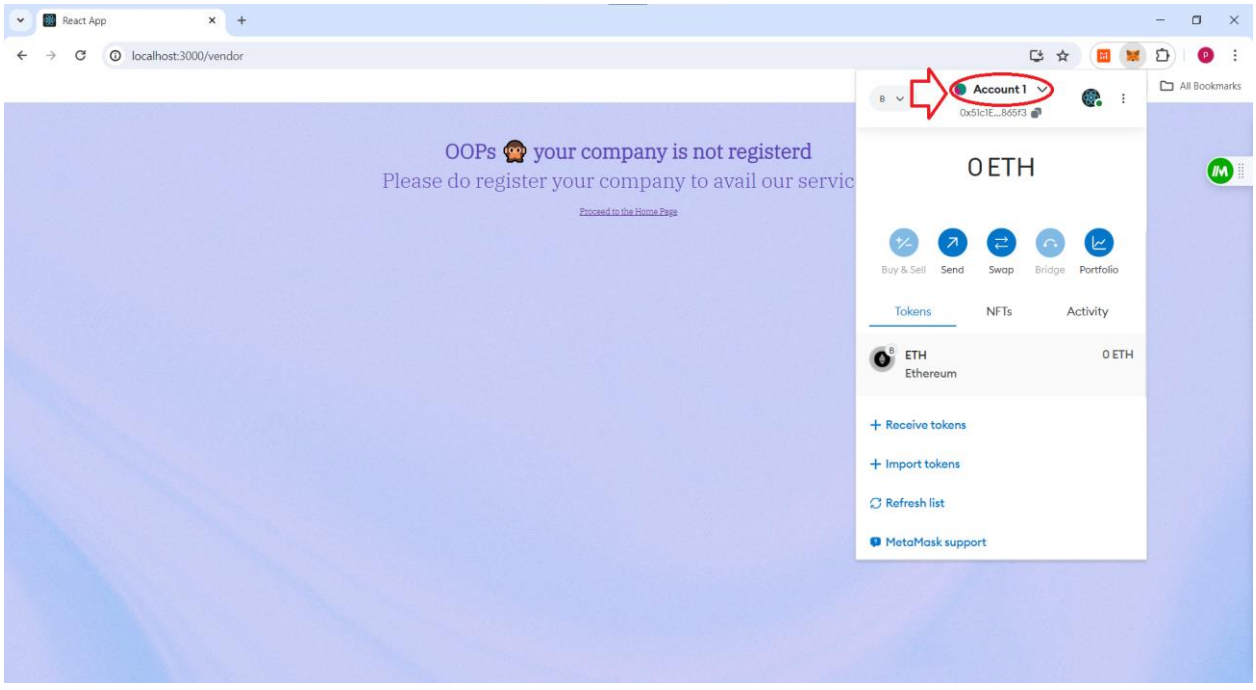
Open the browser and click on the ‘extension’



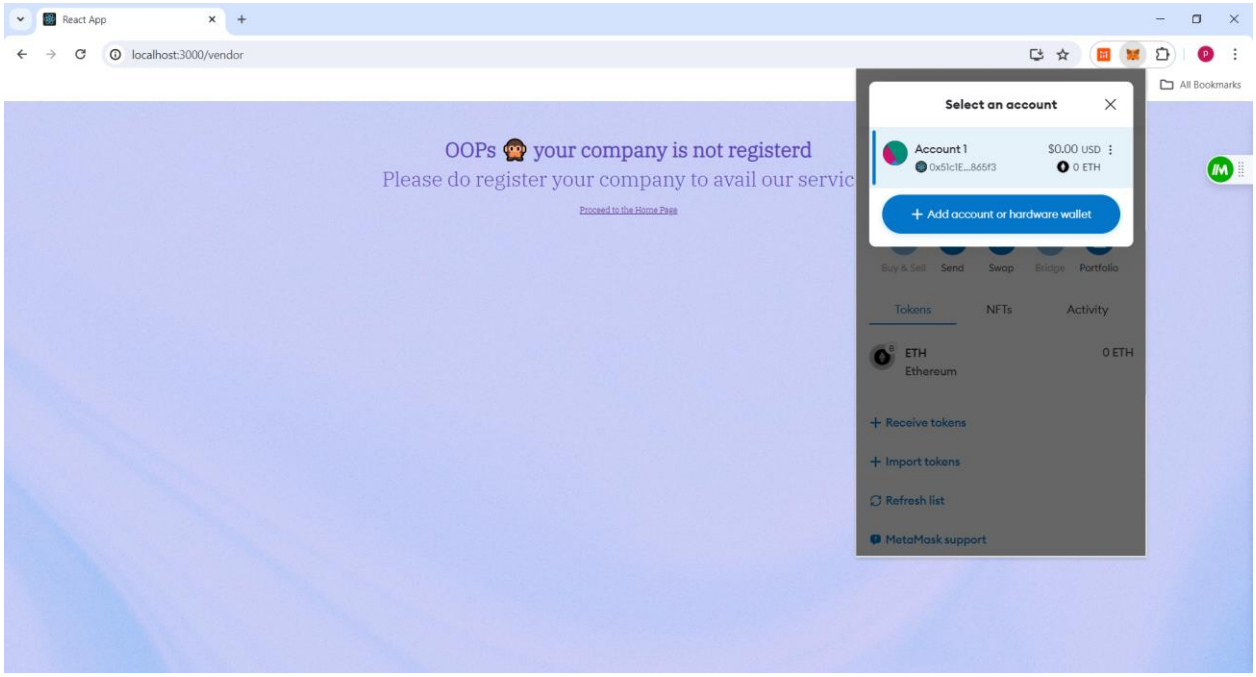
Click on the ‘Metamask’



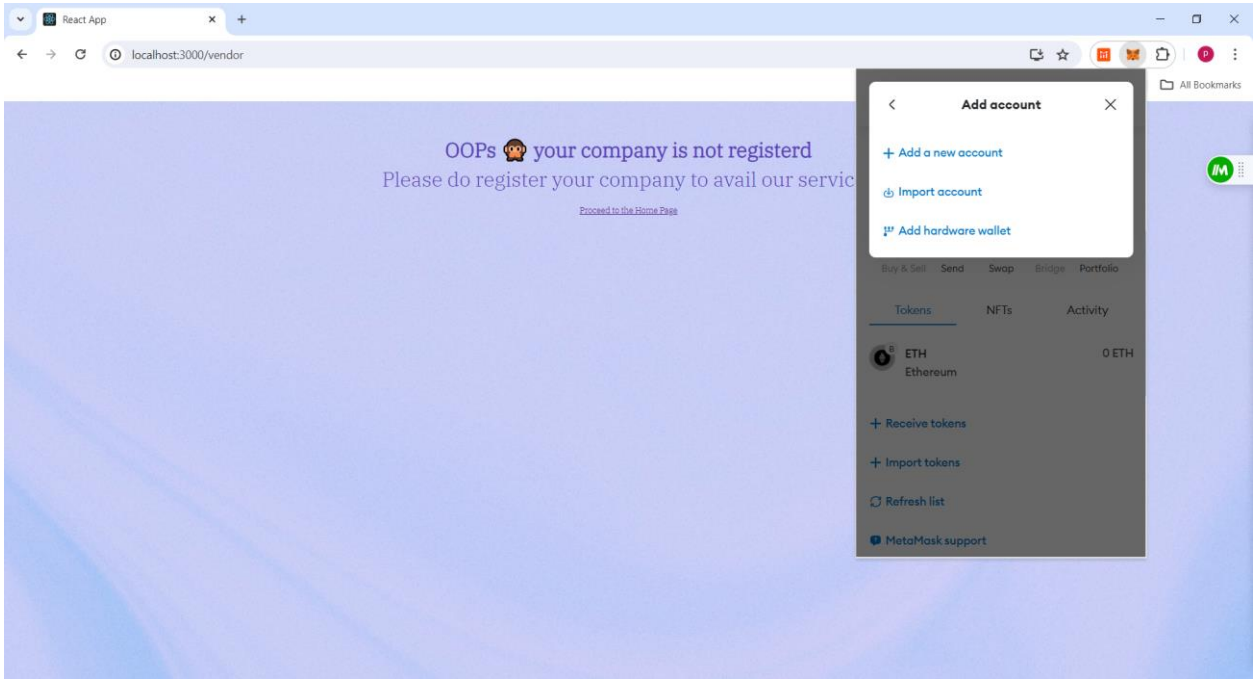
Click on red mark as shown below



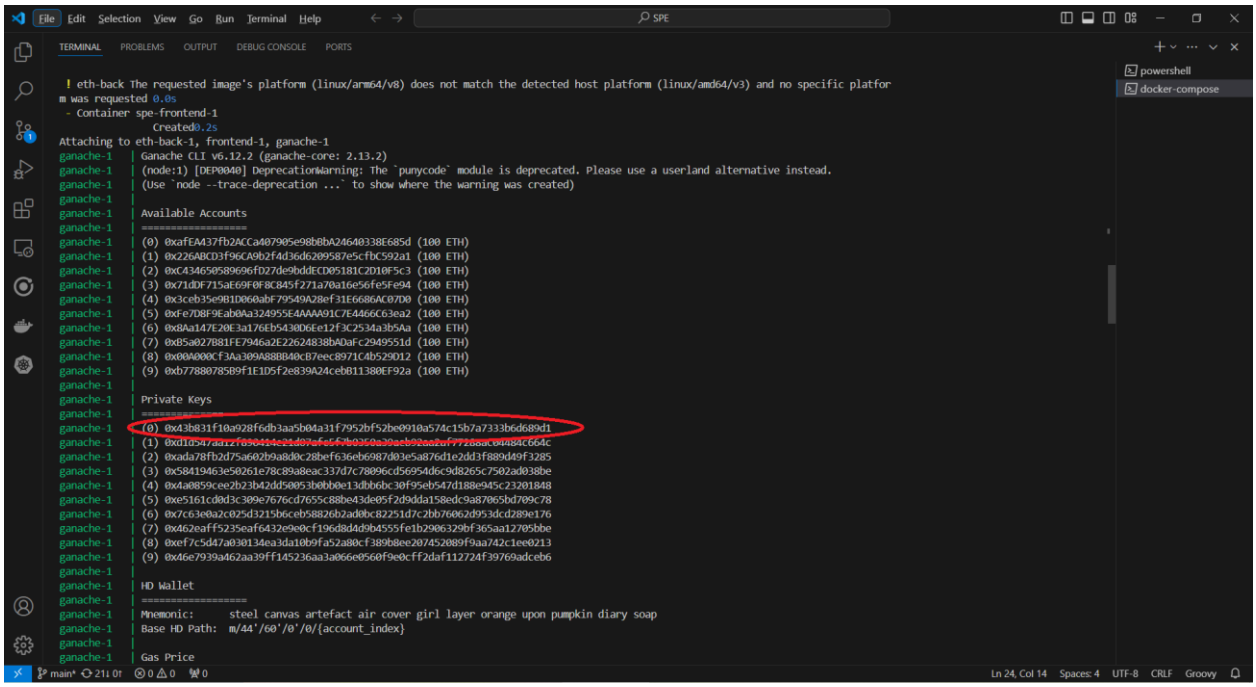
Click on ‘Add account’

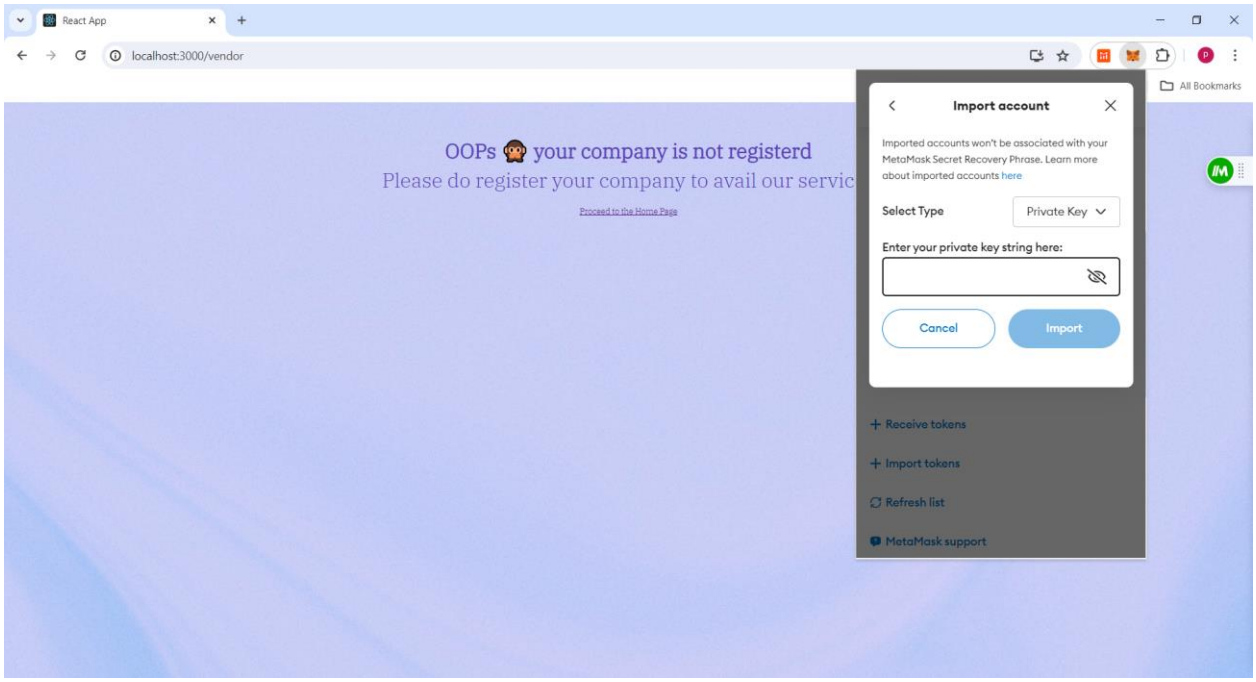


Click on ‘Import account’

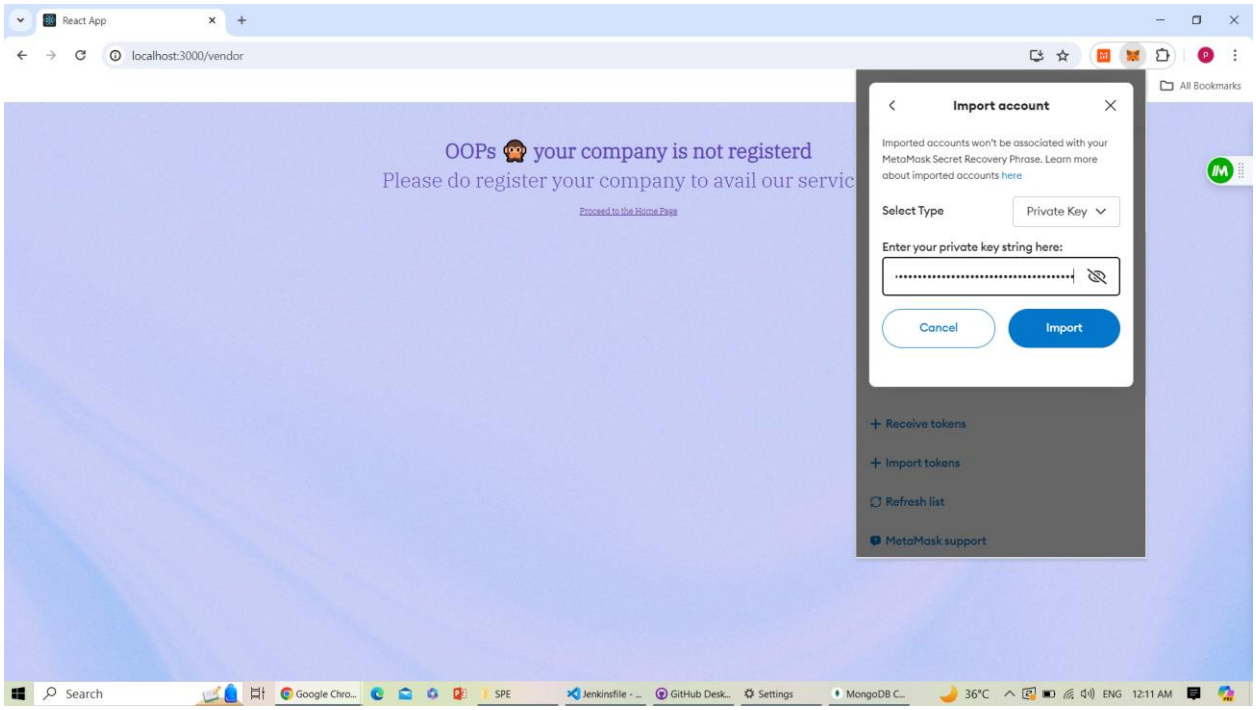


Copy the private key from the terminal where program is running

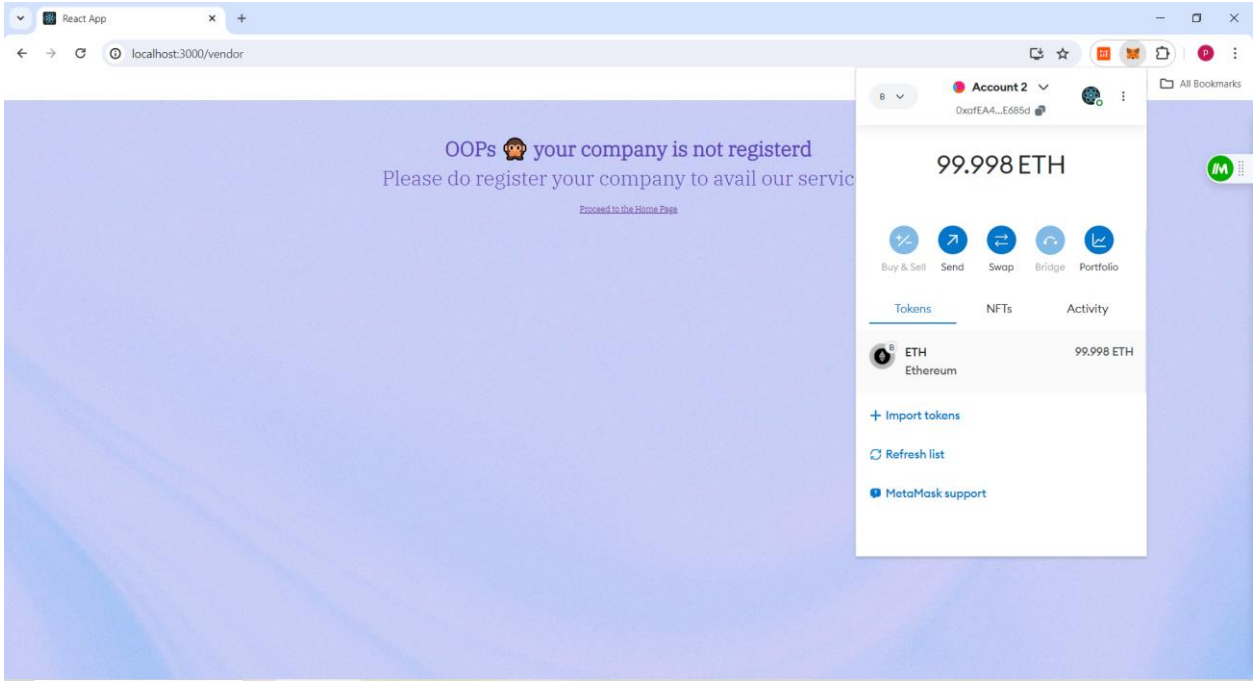




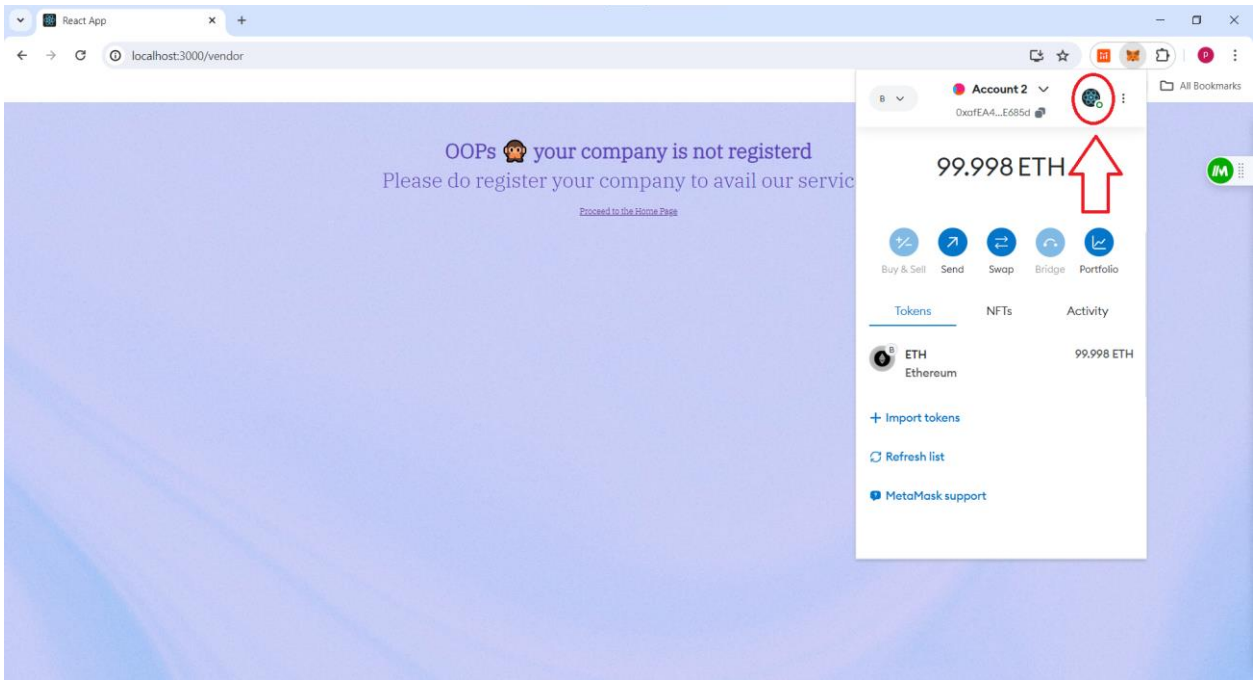
Paste that copy private key



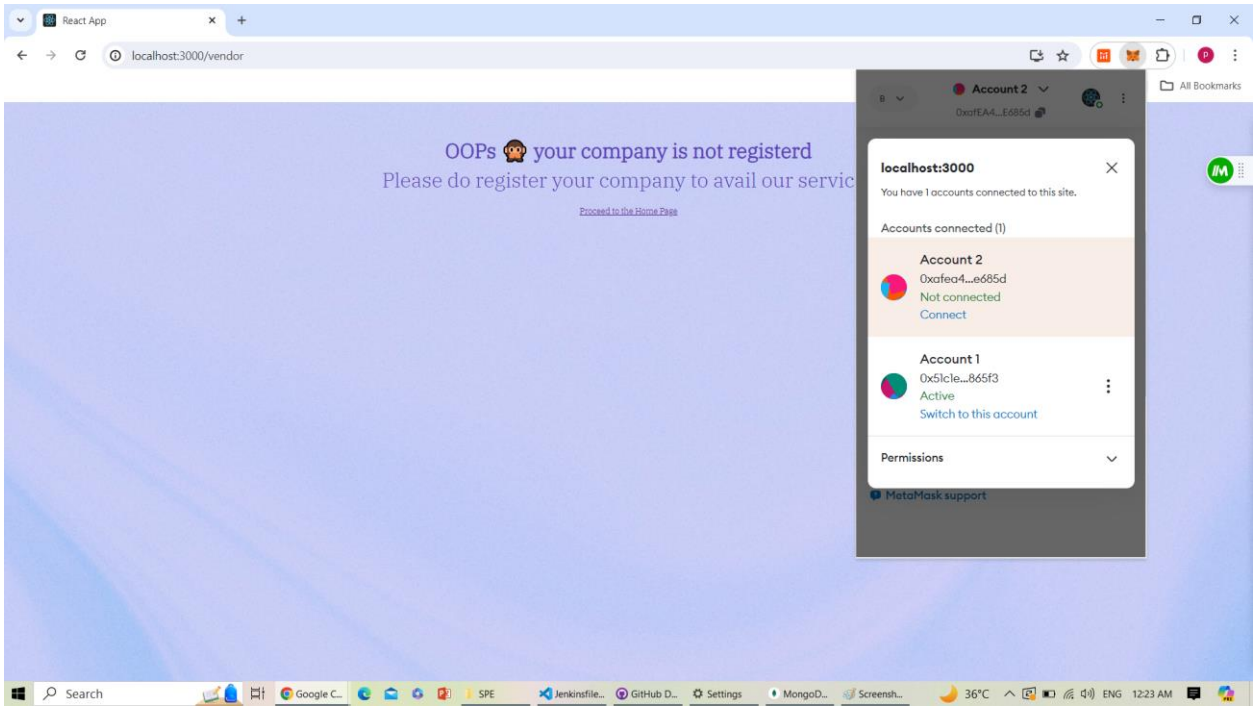
Click on ‘Import’



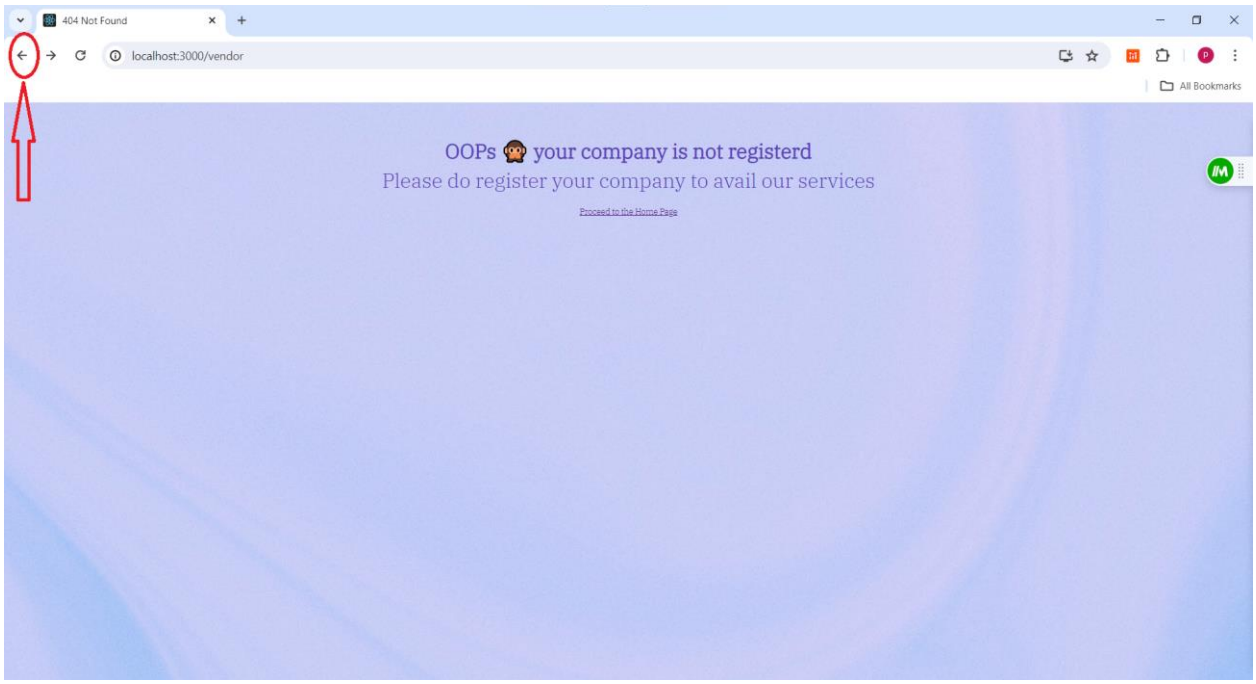
Click on red mark as shown below



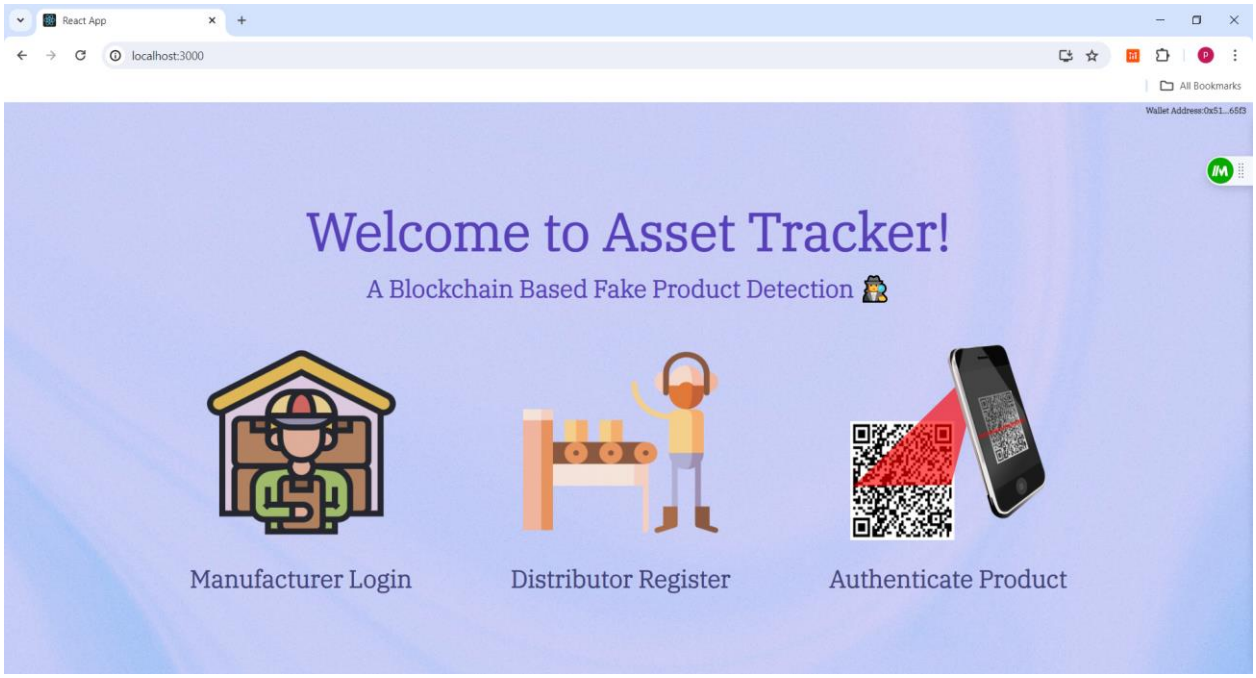
Click on ‘Connect’



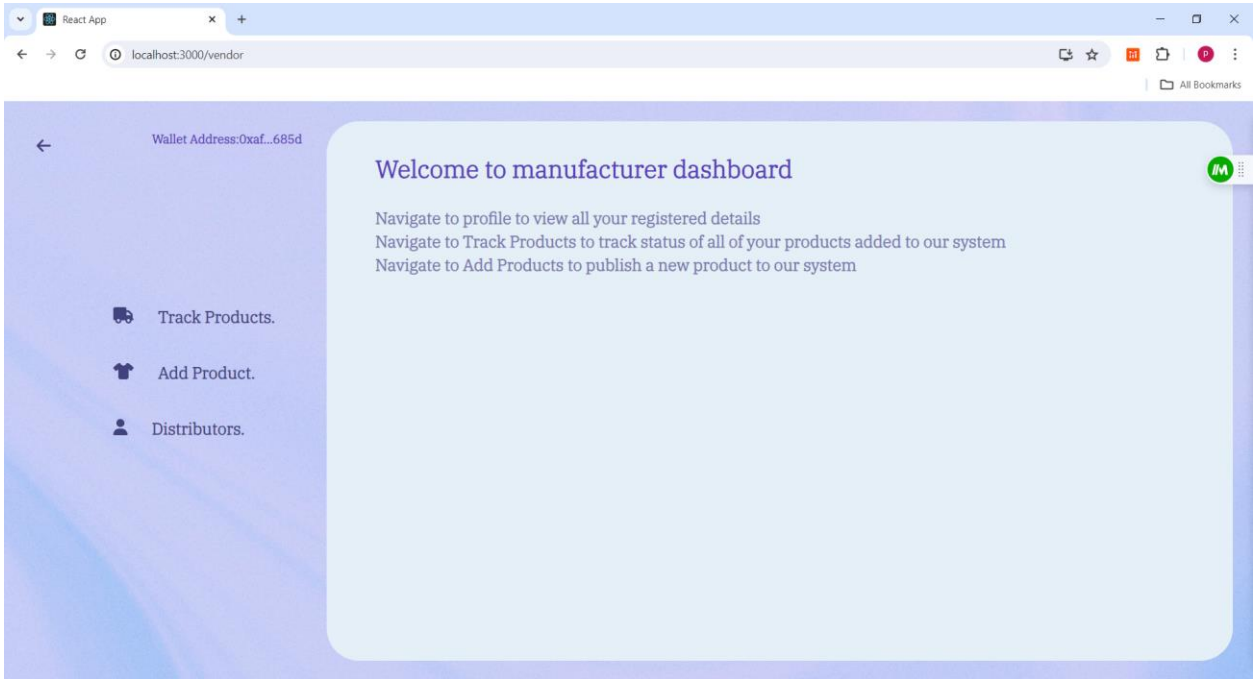
Click on ‘Go back’ of the browser.



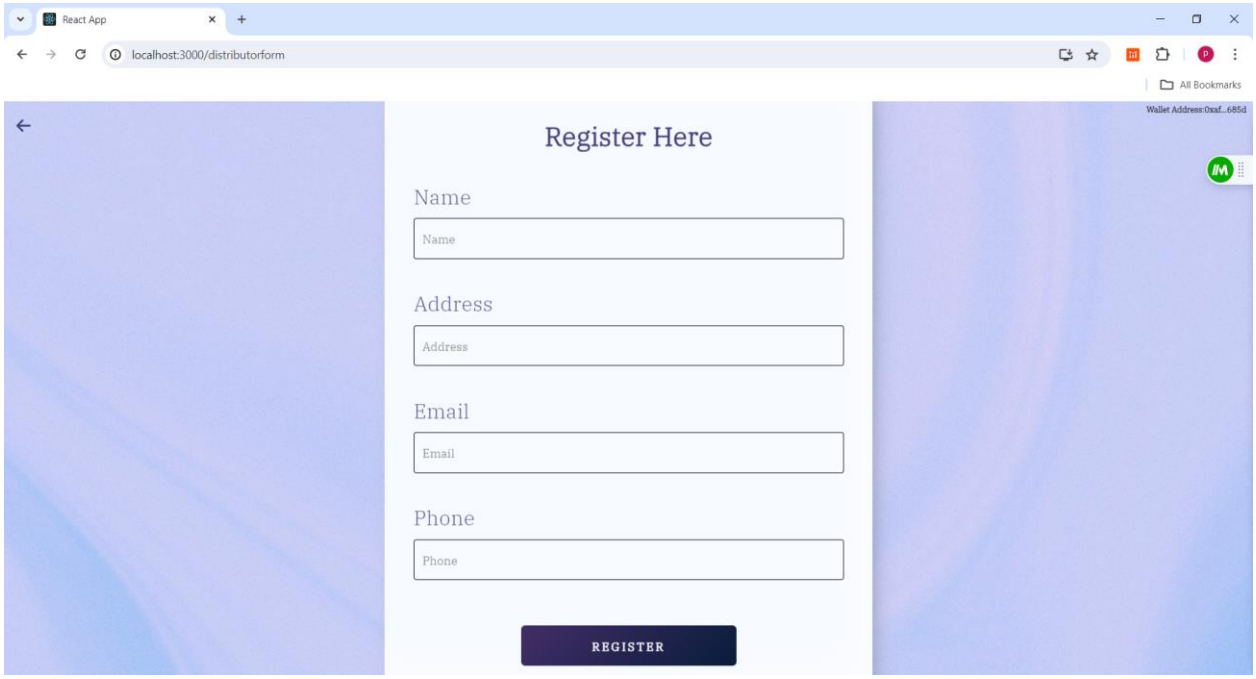
Press F5 key and then click on ‘Manufacturer Login’



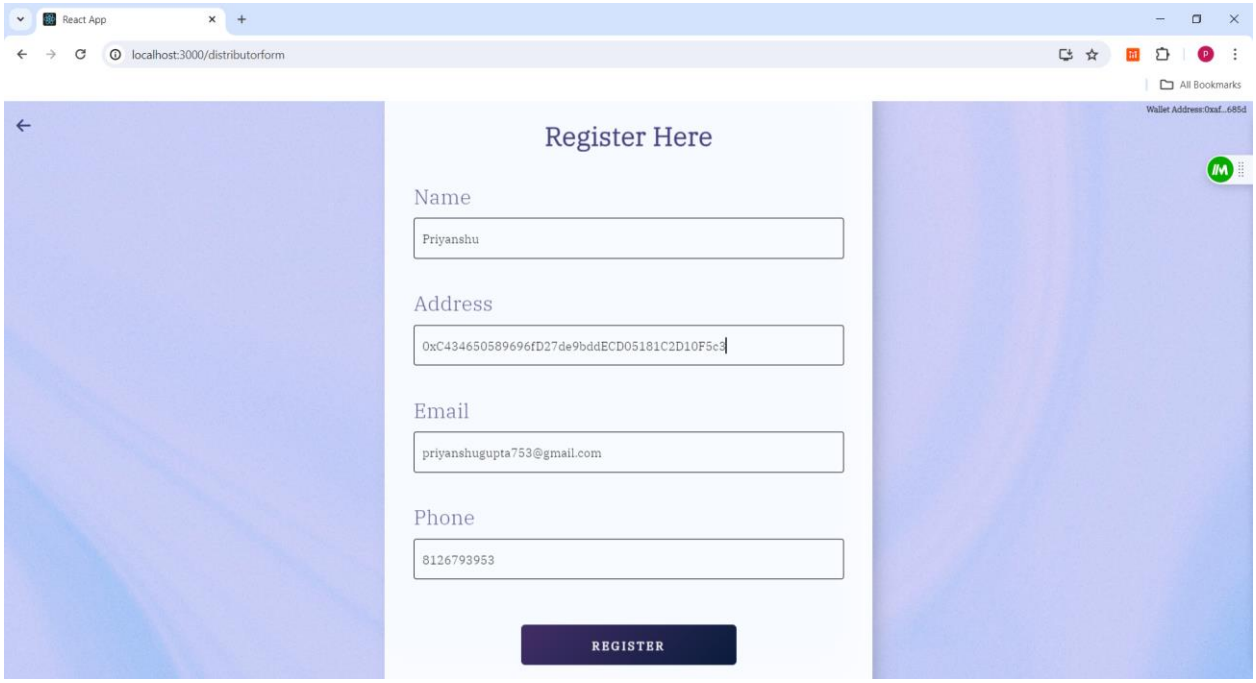
Dashboard of the registered manufacturer



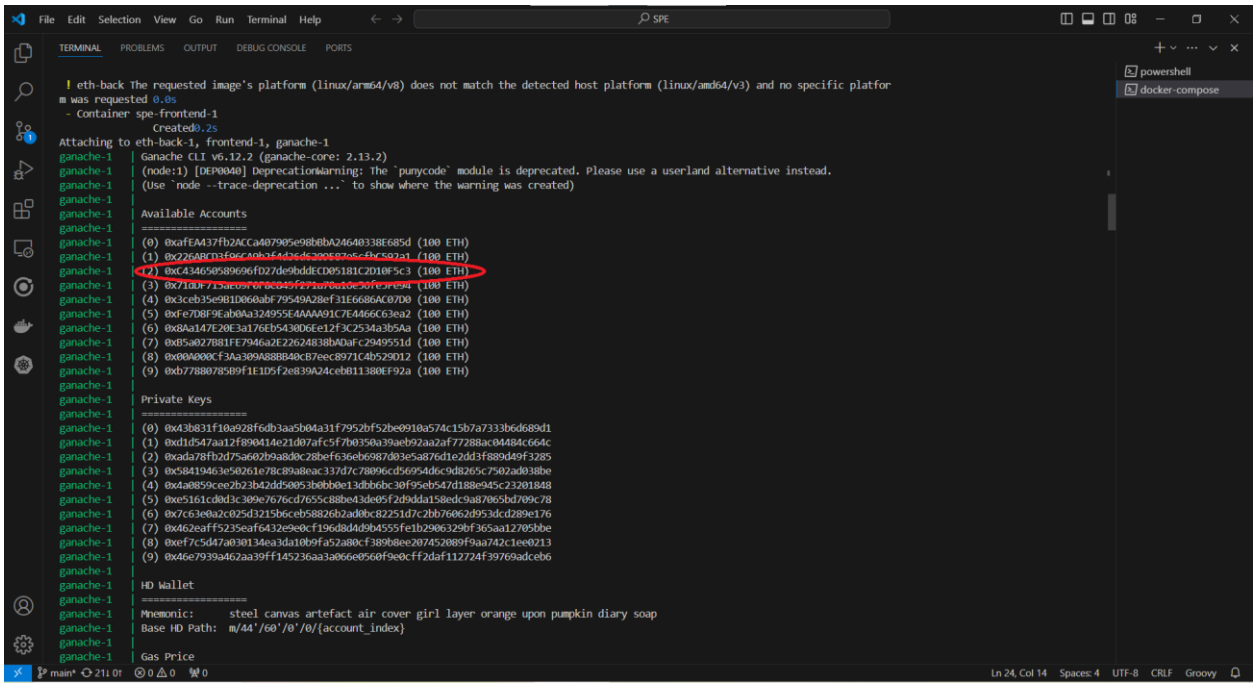
To add the product, the distributor needs to be registered. Click on ‘Go back’ in the browser, then click on ‘Distributor Register’.



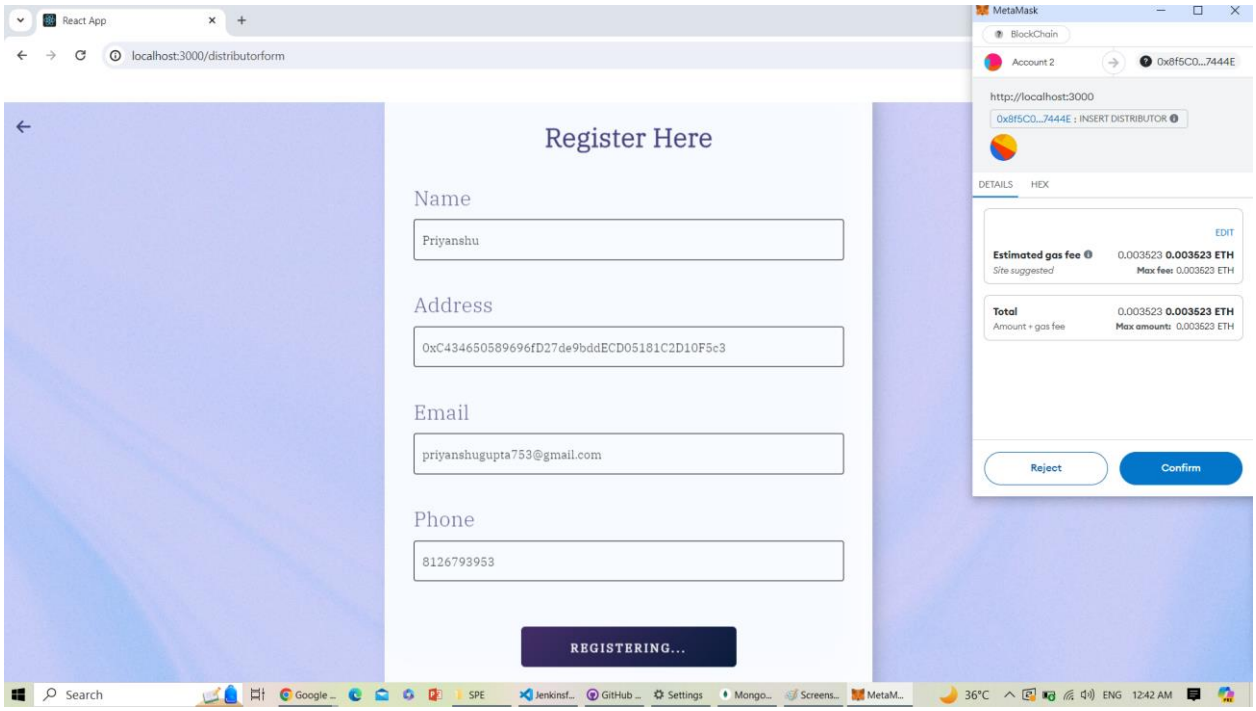
Fill the detail as shown below:



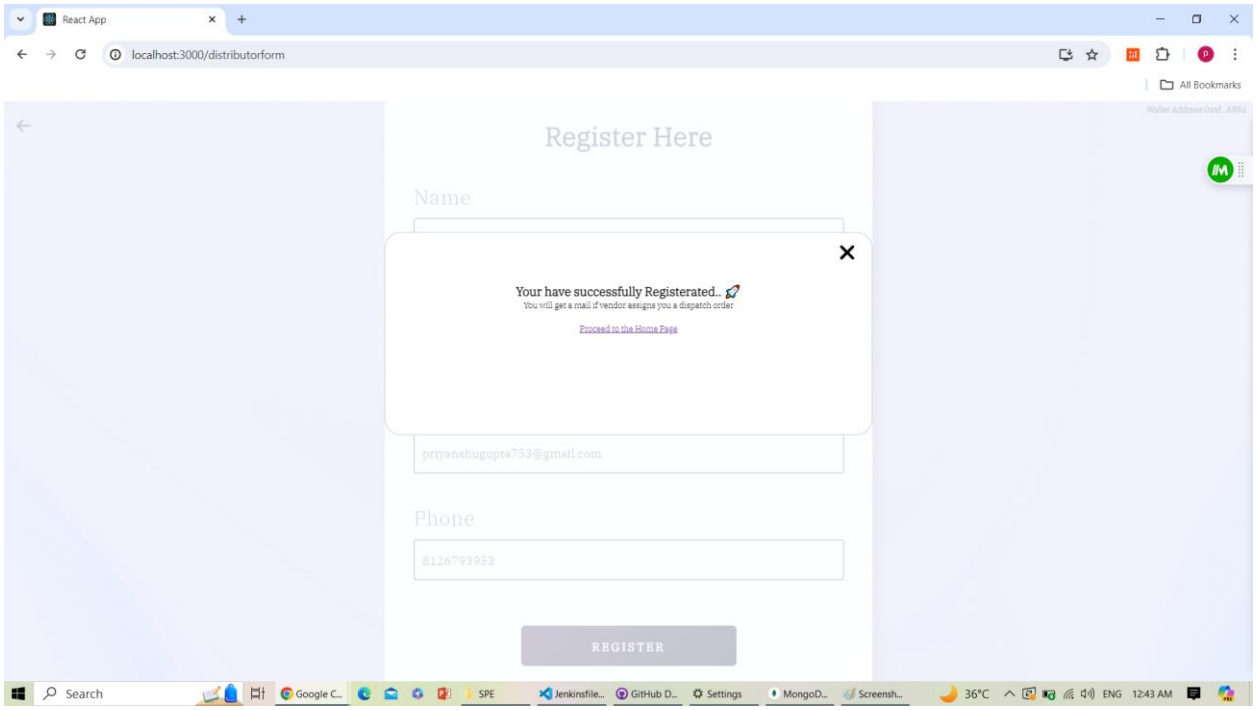
For the address part, copy any address of the account from the terminal that is available.



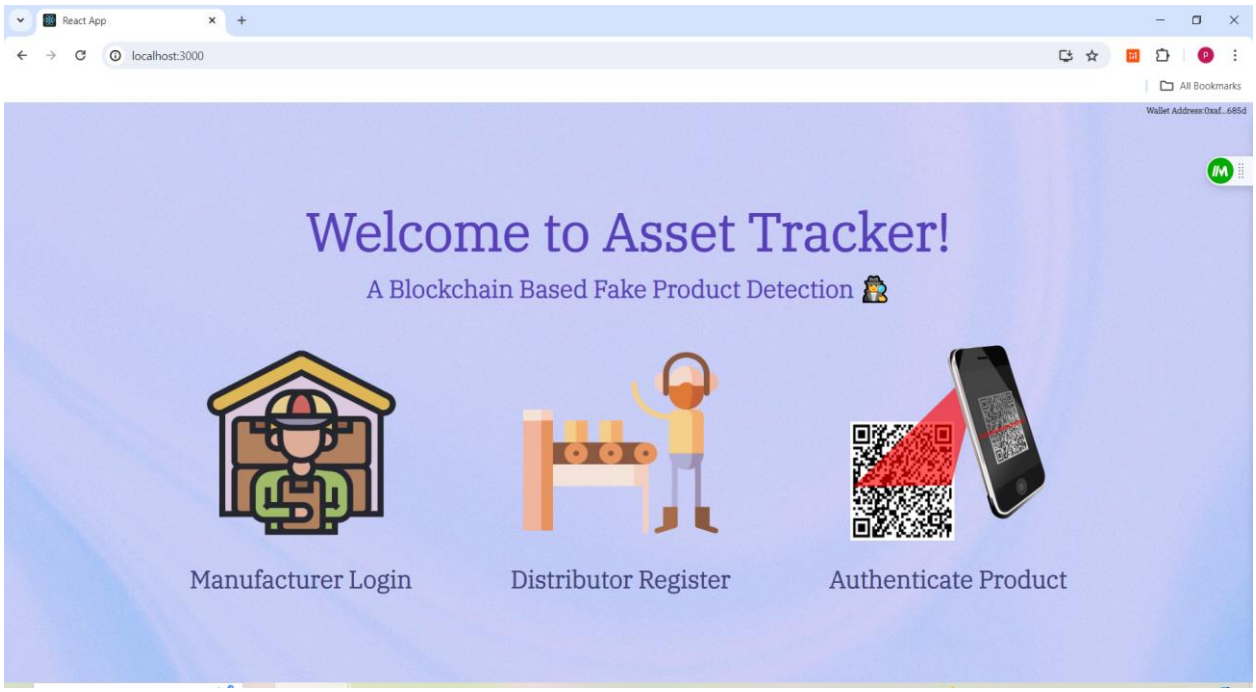
Click on ‘REGISTER’



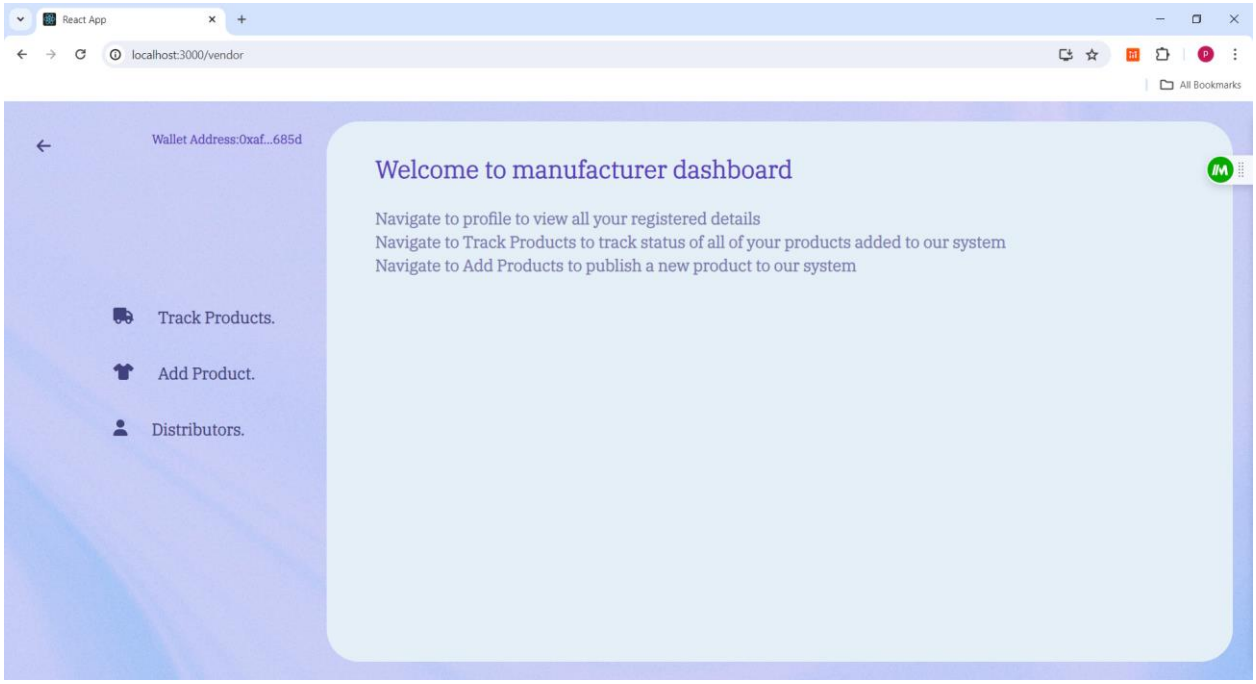
Click on ‘Confirm’. The distributor registered successfully.



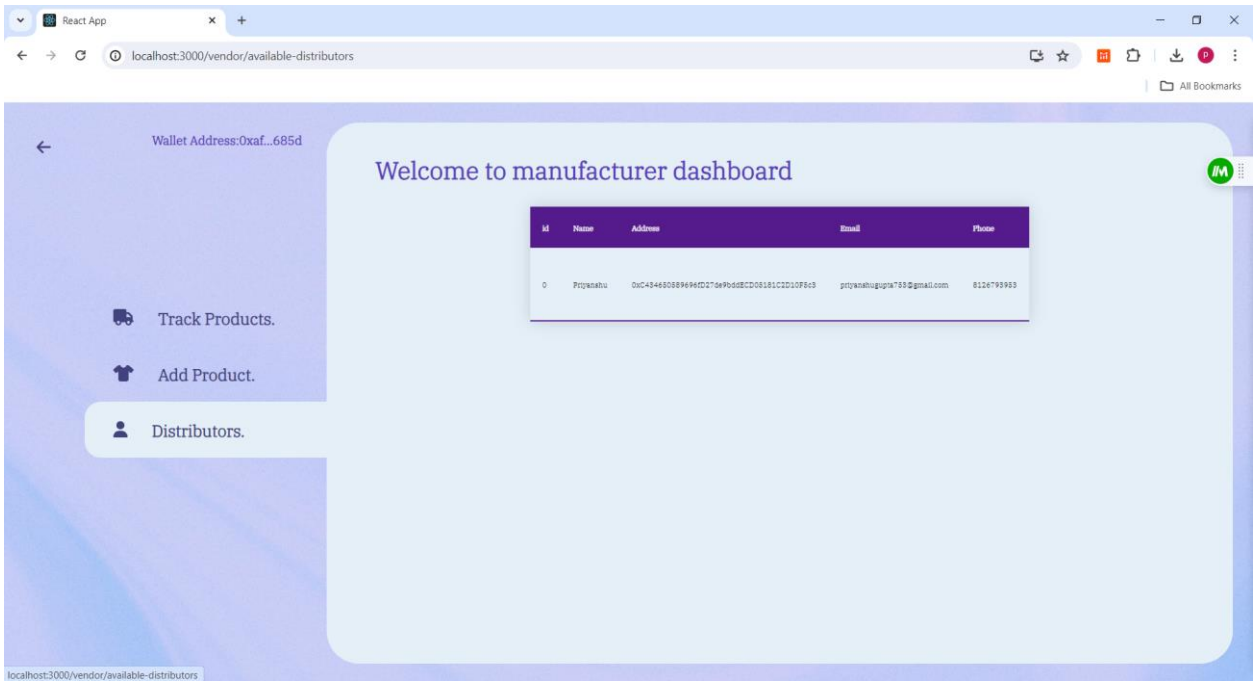
Click on ‘Proceed to the Home Page’



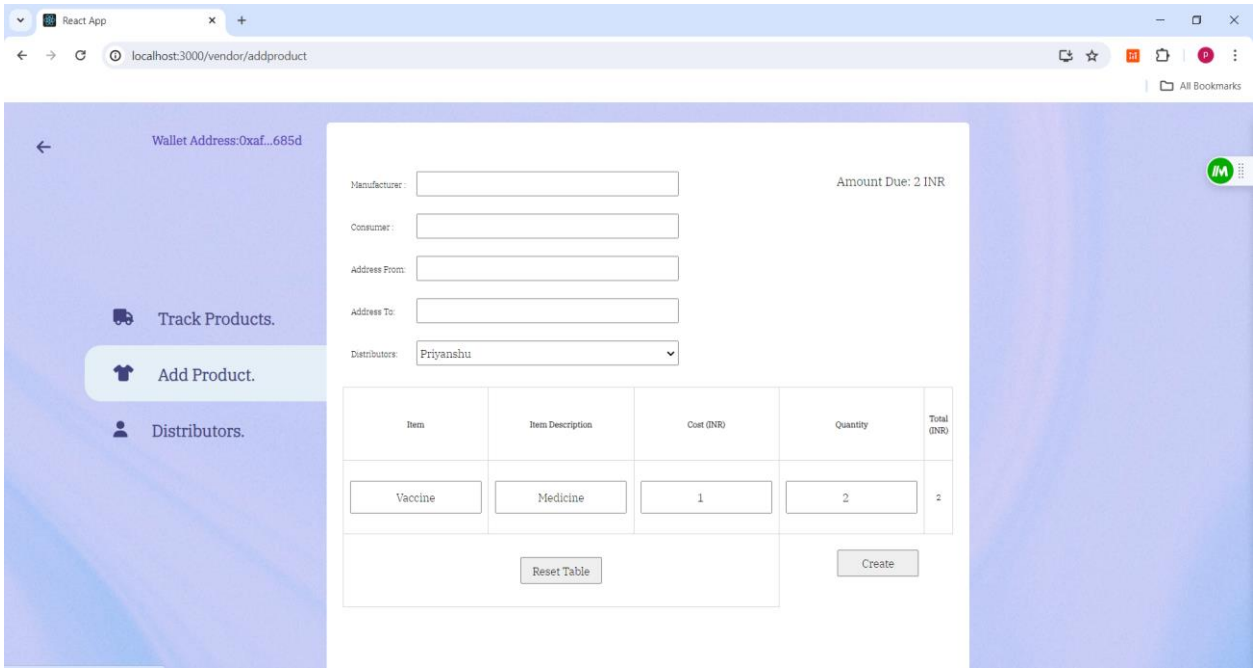
Click on ‘Manufacturer Login’



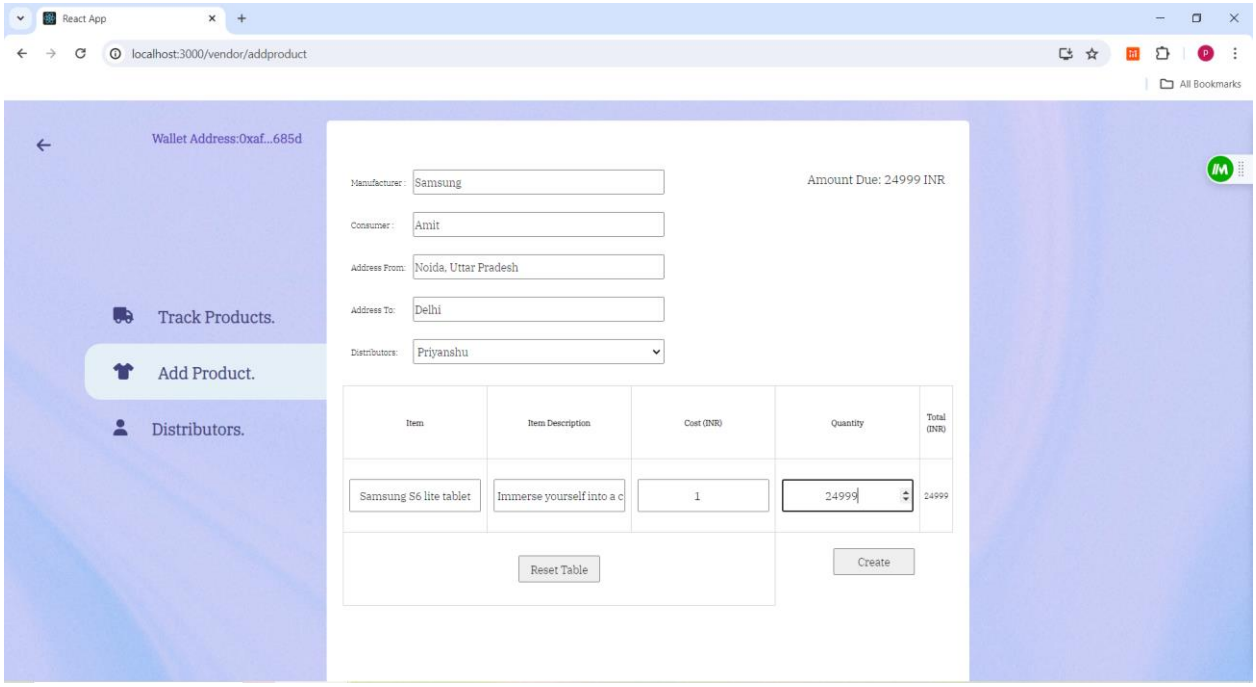
Click on the ‘Distributor’. To the see the added distributor.



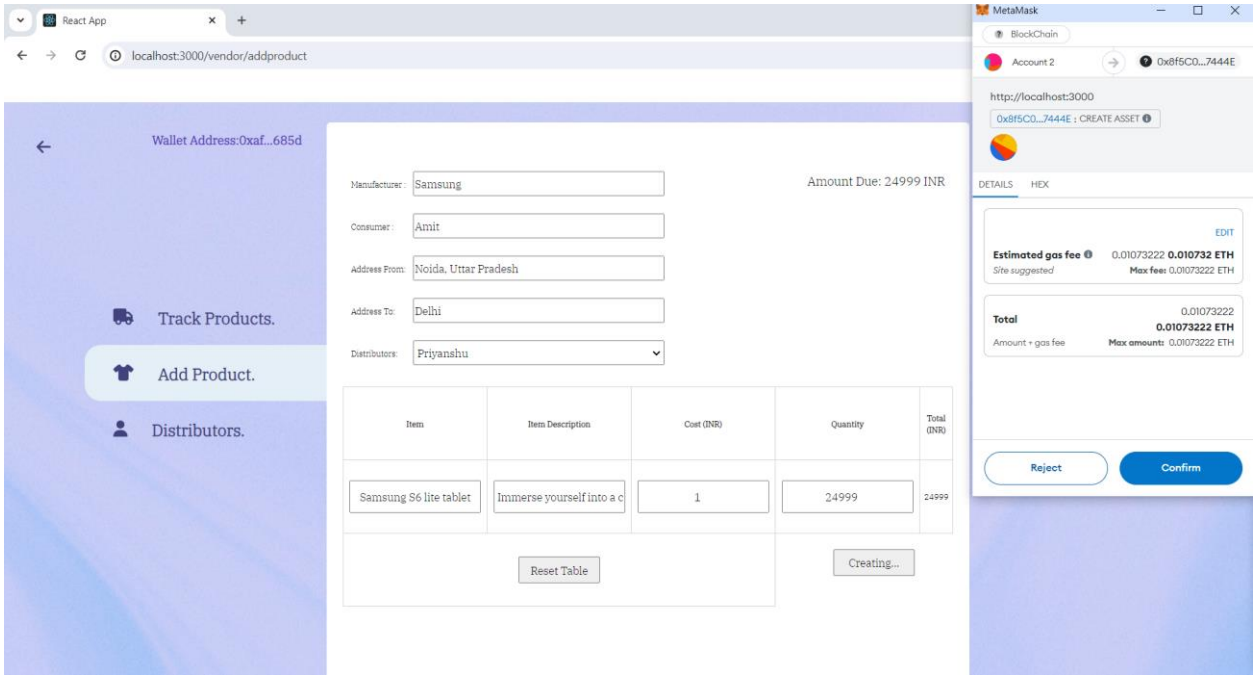
Click on ‘Add Product’. To add the product.



Fill the details as shown below:



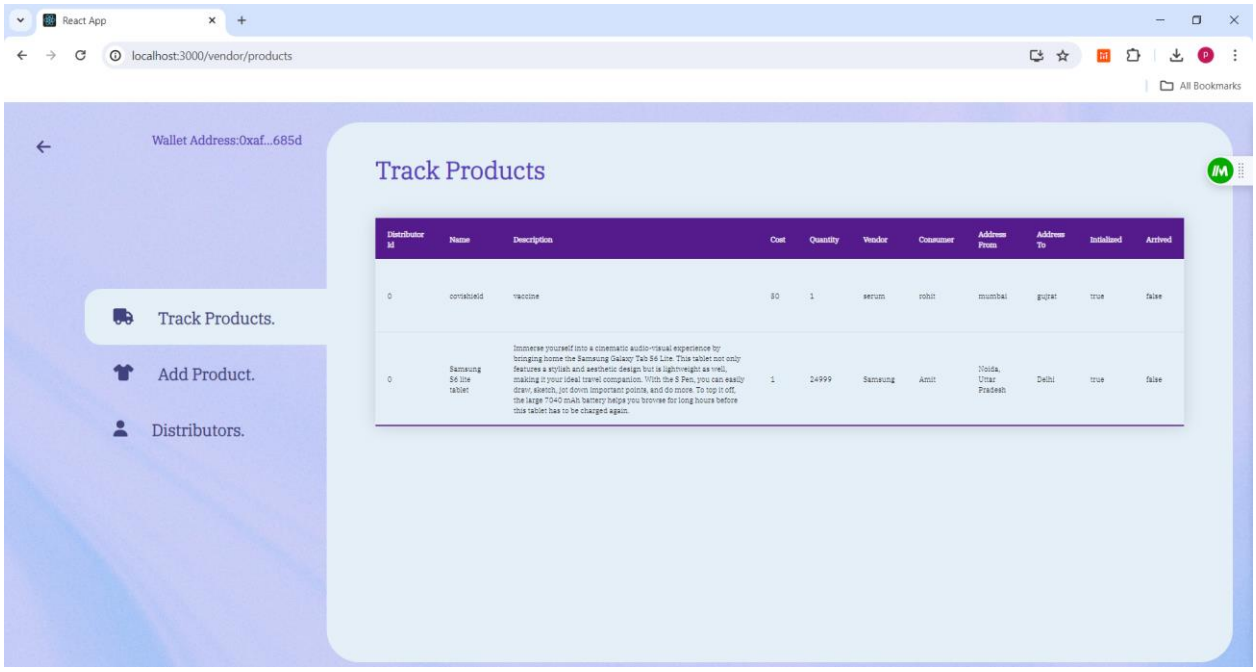
Click on ‘Create’



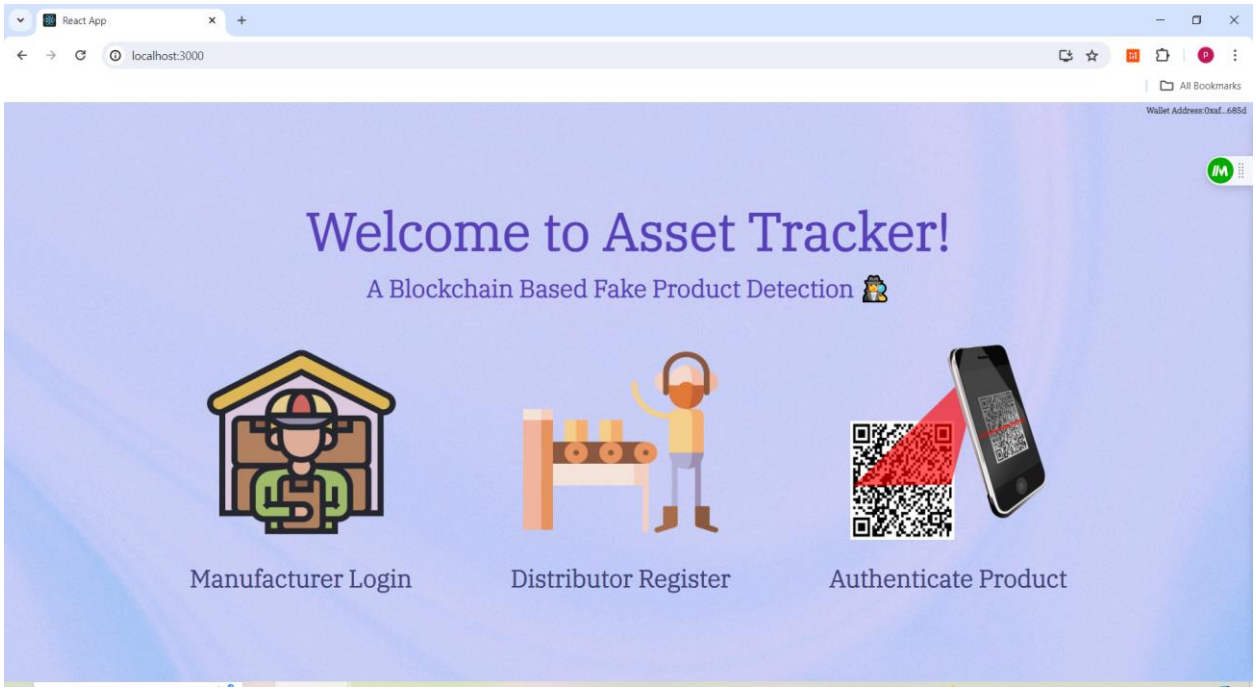
Click on ‘Confirm’.

Download the QR code by right click on it and the save as image.

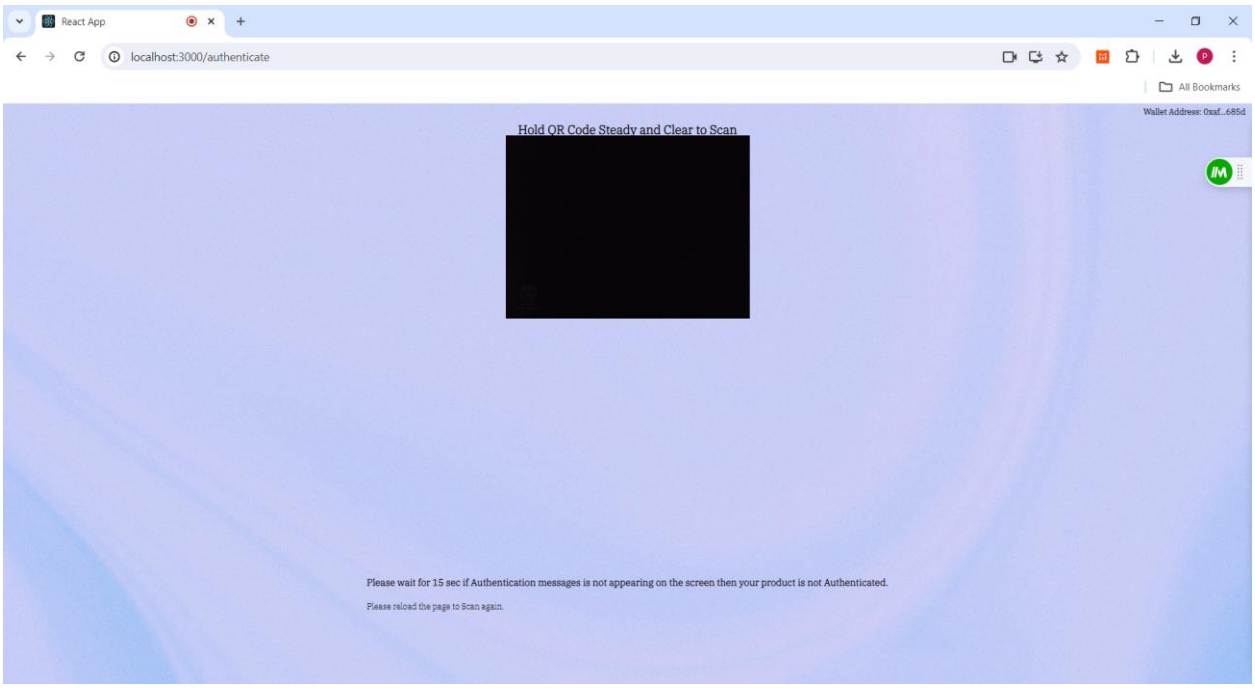
Click on 'Track Products' to view the added product. The 'arrived' column will show 'false' for that product since it has not been delivered or authenticated by the consumer.



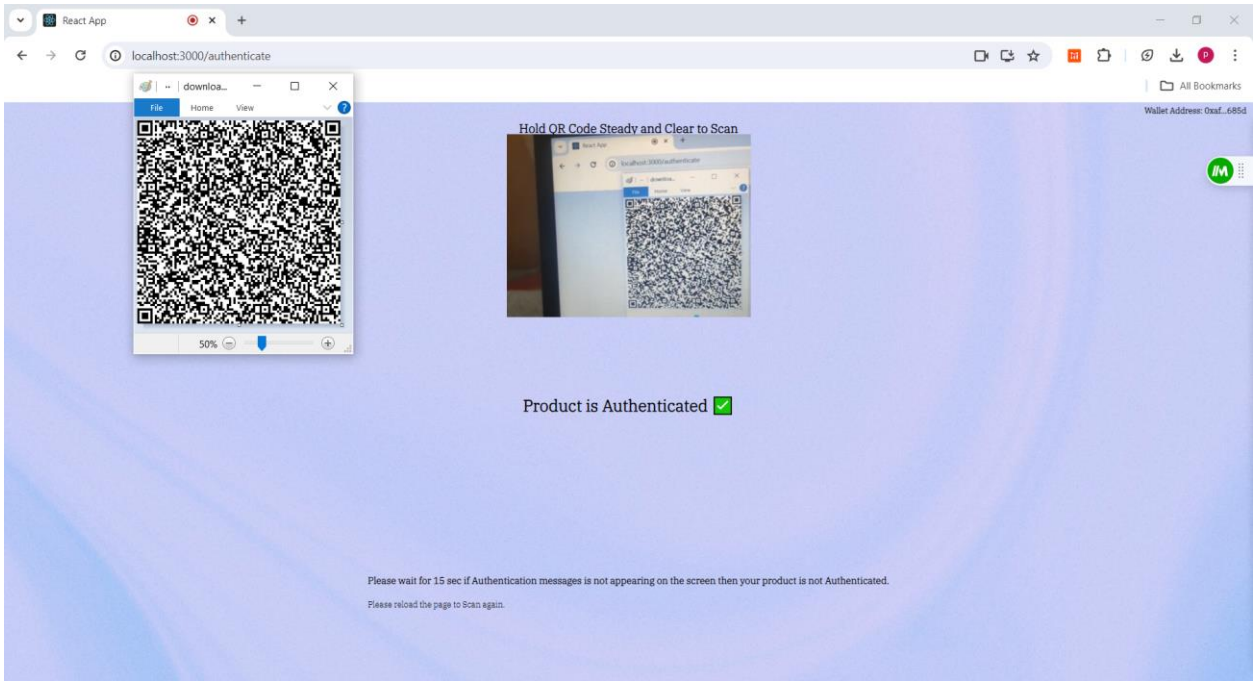
Return back to Home page for product authentication.



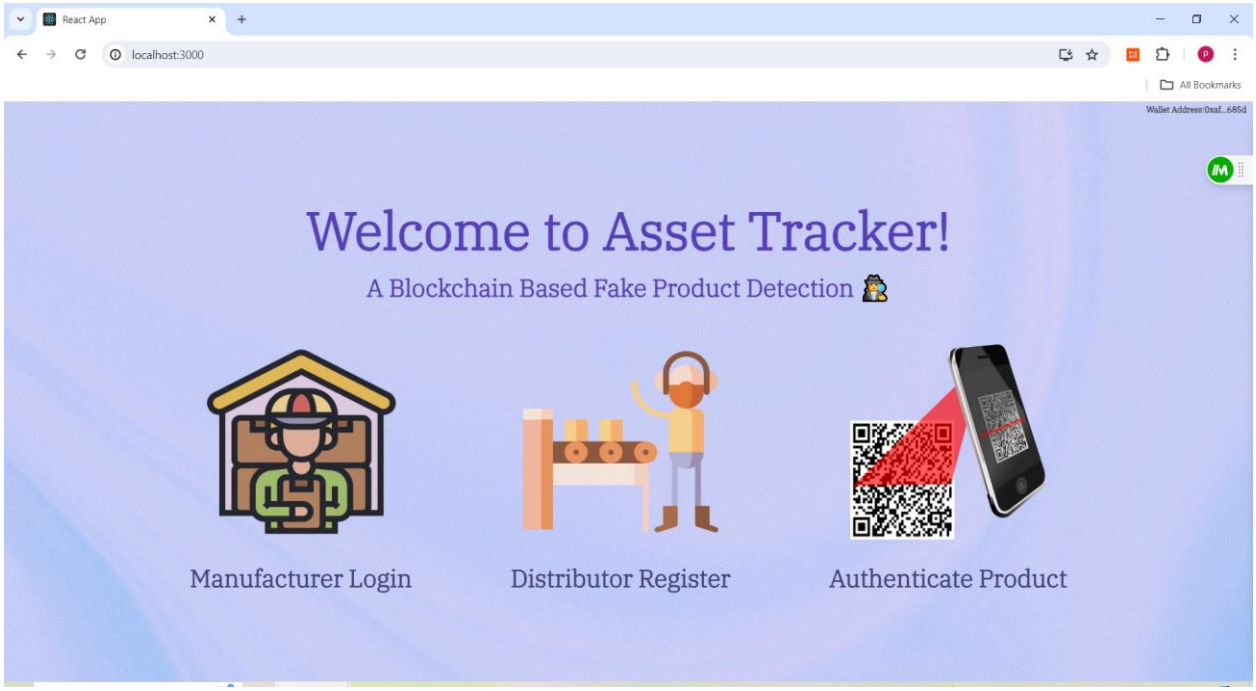
Click on ‘ Authenticate Product’

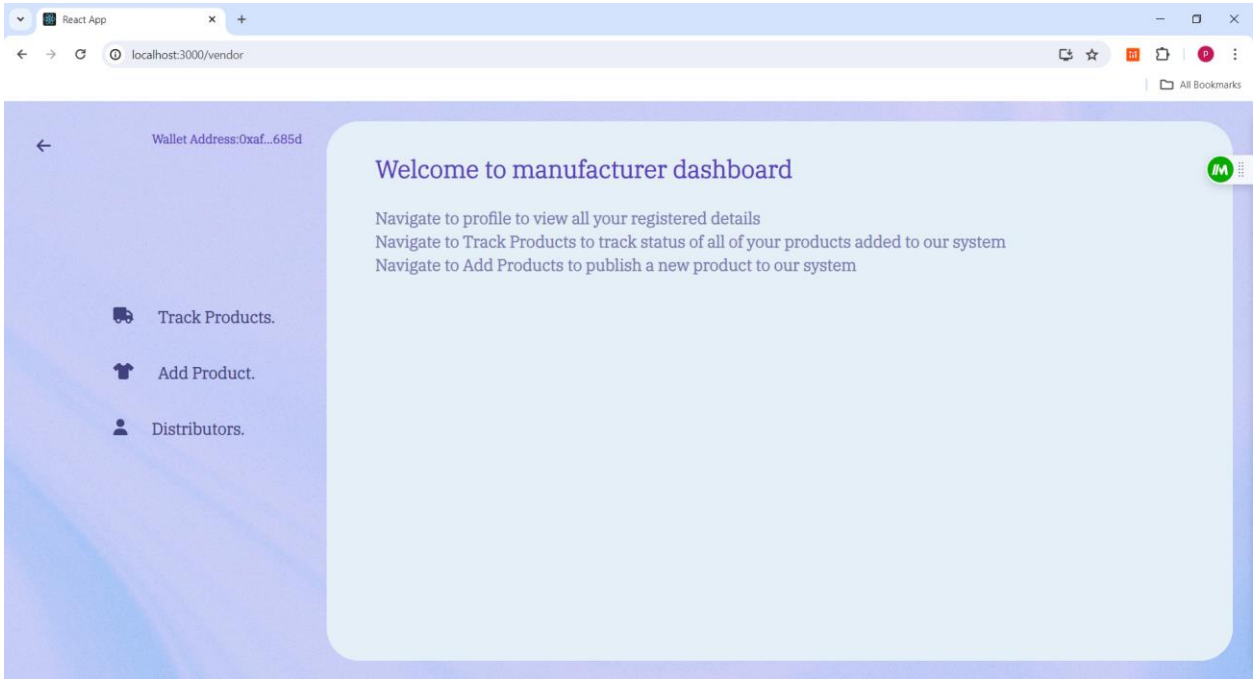


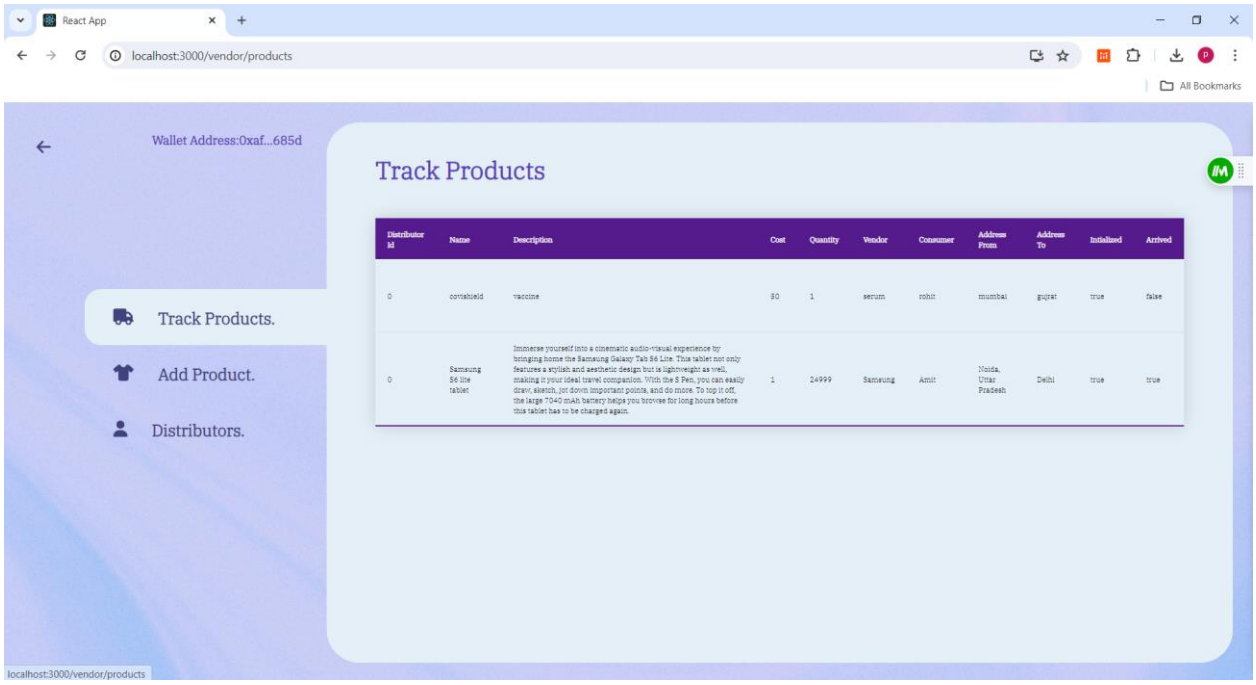
Now scan the downloaded QR code. If the QR code has not been tampered with, it will show that the product is authenticated; otherwise, it will not.



Now, go to the Home Page and click on the Manufacturer Login again to check the 'Track Products' arrival column. Since the product is authenticated, it should display as true.




React App
localhost:3000/vendor
All Bookmarks
Wallet Address:0xaf...685d
Welcome to manufacturer dashboard
Navigate to profile to view all your registered details
Navigate to Track Products to track status of all of your products added to our system
Navigate to Add Products to publish a new product to our system
Track Products.
Add Product.
Distributors.


React App
localhost:3000/vendor/products
All Bookmarks
Wallet Address:0xaf...685d
Track Products
Distributor Id
Name
Description
Cost
Quantity
Vendor
Consumer
Address From
Address To
Initialised
Arrived
0
covishield
vaccine
80
1
serum
rohit
mumbai
gujrat
true
false
0
Samsung S6 lite tablet
Immerse yourself into a cinematic audio-visual experience by bringing home the Samsung Galaxy Tab S6 Lite. This tablet not only features a stylish and aesthetic design but is lightweight as well, making it your ideal travel companion. With the S Pen, you can easily draw, sketch, jot down important points, and do more. To top it off, the large 7040 mAh battery helps you browse for long hours before this tablet has to be charged again.
1
24999
Samsung
Amit
Noida, Uttar Pradesh
Delhi
true
true
Track Products.
Add Product.
Distributors.
localhost:3000/vendor/products

12 Challenges Faced

1. While setting up the local blockchain and to communicate with frontend is a little bit difficult task for us.
2. Containerising the blockchain inside the container was one of the most difficult task that we encountered.
3. After containerising, getting all the containers communicate with each other, we faced a little difficulty in that phase.
4. After networking, building the containers and automating the whole pipeline was a difficult task for us.
5. We also faced challenges during the pipeline and app build process for the backend.

13 Scope for Future Work

In this project, we demonstrated a prototype showcasing how containerization can effectively scale solutions built on Blockchain. This technology can be particularly useful in areas where supply chain issues persist and companies face ongoing challenges. By employing containerization in conjunction with Blockchain, organizations can quickly halt the delivery of counterfeit products within their supply chains. This approach provides a swift resolution to these issues, thereby helping companies maintain their reputation in the market.

14 References

1. https://web3js.readthedocs.io/en/v1.2.9/

2. https://medium.com/daox/three-methods-to-transfer-funds-in-ethereum-by-means-of-solidity-5719944ed6e9

3. https://codingwithmanny.medium.com/how-to-dockerize-your-hardhat-solidity-contract-on-localhost-a45424369896

4. https://ethereum.org/en/developers/docs/accounts

5. https://hardhat.org/
6. https://docker.com

7. https://medium.com/hackernoon/create-an-ethereum-dapp-with-react-and-docker-211223005f17

8. https://docs.soliditylang.org/en/develop/types.html