



# **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

<b>1 Abstract.....</b>	<b>3</b>
1.1 Problem Definition.....	3
1.2 Solution Description.....	3
1.3 Features of the website.....	4
1.4 Important Links.....	5
<b>2 Introduction.....</b>	<b>5</b>
2.1 What is DevOps? .....	5
2.2 Why to use DevOps?.....	5
2.3 Technologies Used:.....	5
2.5 Operation Tools Used .....	6
<b>3 Architecture Diagram.....</b>	<b>7</b>
<b>4 System Configuration .....</b>	<b>6</b>
<b>5 Setting Up the Ganache Blockchain.....</b>	<b>7</b>
5.1 Installing the Ganache .....	7
5.2 Containerizing the Blockchain.....	8
<b>6 Writing Smart Contract code with Hardhat .....</b>	<b>9</b>
6.1 Writing the smart contracts.....	9
6.2 Writing the Web3.js script .....	11
6.3 Deployong the smart contact on the blockchain using deploy.js .....	11
6.4 Writing the logic with logic.js script .....	13
6.5 Configuring Hardhat.....	14
<b>7 Software Development Life Cycle Frontend (Devops) .....</b>	<b>16</b>
7.1Source Code Management .....	16
7.2 Building the React Application .....	17
7.3 Containerization (Frontend) .....	18
7.4 Code Walkthrough .....	18
<b>8 Software Development Life Cycle Backend (Devops).....</b>	<b>19</b>
8.1 Source Code Management .....	19
8.2 Writing the server code .....	19
8.3 Containerization.....	21
<b>9 Orchestration of Containers using Docker-compose.....</b>	<b>23</b>
9.1 Writing the docker compose files .....	23
<b>10 Configuring the CI/CD pipeline using Jenkins.....</b>	<b>25</b>

10.1 Writing the Jenkinsfile.....	25
10.2 Writing the Ansible Playbook(Deploy.yml) .....	28
10.3 Writing the inventory file.....	28
10.4 Setting Up the Jenkins Master Node .....	29
10.5 Running the pipeline .....	31
<b>11 Testing and Results.....</b>	<b>36</b>
<b>12 Challenges Faced.....</b>	<b>72</b>
<b>13 Scope for Future Work.....</b>	<b>72</b>
<b>14 References .....</b>	<b>72</b>

# 1 Abstract

## 1.1 Problem Definition

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 fixes and easier maintenance as compared to traditional methods.

### 2.3 Technologies Used:

Frontend: ReactJs

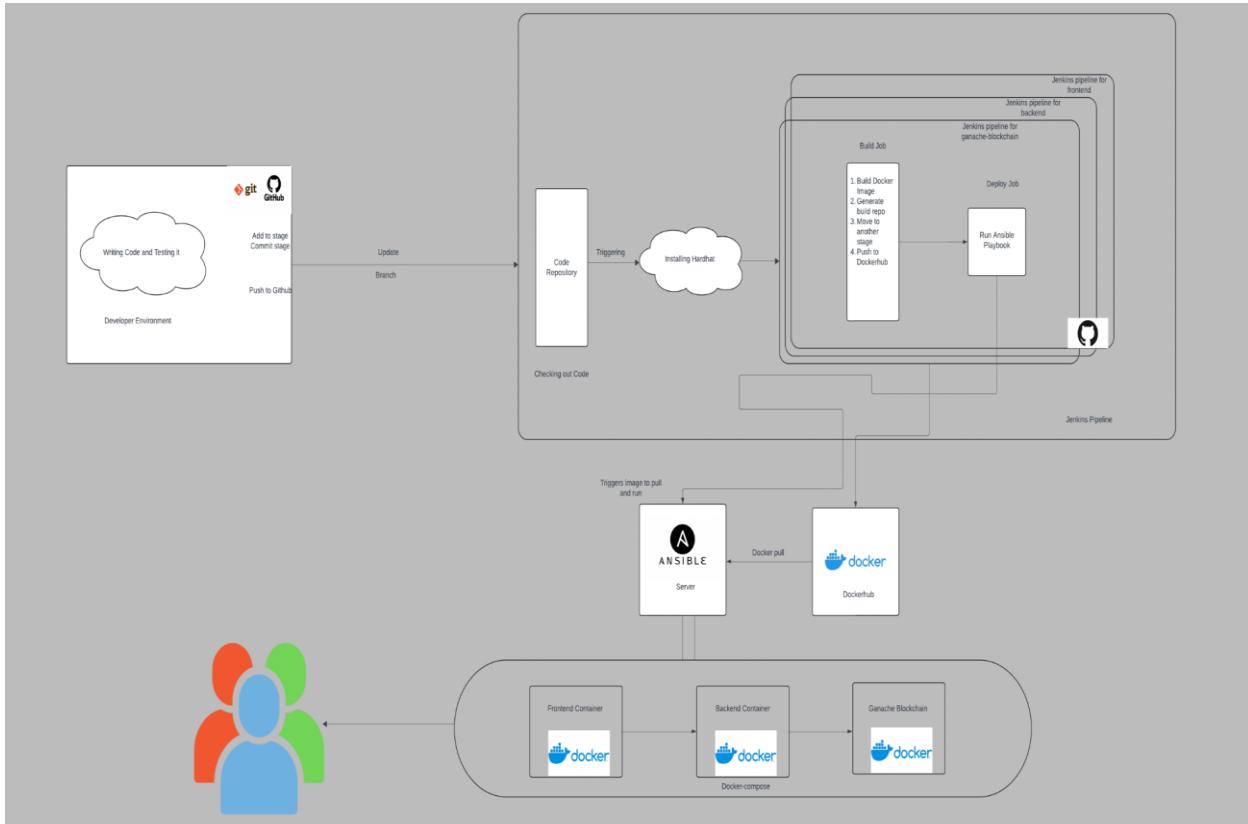
Backend: NodeJs, Solidity

Databases: Ethereum Blockchain

### 2.5 Operation Tools Used

- **SCM** : Git and GitHub
- **CI/CD** : Jenkins
- **Containerization** : Docker
- **Build Tool** : NPM (Node Package 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>
```

```
Ganache CLI v6.12.2 (ganache-core: 2.13.2)
(node:1) [DEP0040] DeprecationWarning: The `punycode` module is deprecated. Please use a userland alternative instead.
(Use `node --trace-deprecation ...` to show where the warning was created)

Available Accounts
_____
(0) 0xD6EEeE76F25B379627C71EaE34B32fe239C9aCAa (100 ETH)
(1) 0xa8A942A96bd67aec15ce28B03de587E7affcC025 (100 ETH)
(2) 0x6d388DbA4E8421c0120F55B385713f8aE8332cB3 (100 ETH)
(3) 0x76F773Fc3Cf2666321cC3537ba774DA0697A3CE0 (100 ETH)
(4) 0x1BF06494e5B195d89830b4c458127E2100ad1fb2 (100 ETH)
(5) 0xFbEc18Cd03eb19F46fec3f67257Fb6B182bc7A6a (100 ETH)
(6) 0x5883044e3870574a8F4D727a0aF9Ed56a1B11365 (100 ETH)
(7) 0x0FECD9D1Febd52f61AC43C5f02C7BA4Fc1DD341d (100 ETH)
(8) 0x145D731665d3d64046c1BC840F1caaFC51d938F9 (100 ETH)
(9) 0x28B28207d50A2A77C12f49BbC782F6e4a84E4722 (100 ETH)

Private Keys
_____
(0) 0x5d446f0d308789ed82e01417e34390b6e61ea0385a742cd72d3c63afc2c4ea3c
(1) 0xc29b7ca025832d5907d7b42449186b1783ffb220a4173341a836520c40fd1a73
(2) 0x90162cf3593e427224304cf18a43260817c7183a2d58153fb72769f2d82908e3
(3) 0x85e2bb996e2b45535650cc3e7b5ef88479d938ed75f01c6baF6929dad3db220
(4) 0xdca9305210c8d614@01f34d65fdd48b020a7e5b79ada40455ce30761936a05a7e
(5) 0x6a683733f0d1e1349d97f9c7009ca6f72115780f898fa62ce04db54327374e69
(6) 0xb57d169b68806c99eec18b7d48e0ffa1815bb2f67828549083d7a3de4aad64f
(7) 0xad4f979ad449835b3429960f2c046d795393dc96da89455efcc697da5f0a66ae
(8) 0x2a6f2d43bfff36ace3aceb16a9885cd9d53edd87ff305f3978aff86f4833142ea
(9) 0x86ea4cdf77a5a6f8716cd9340f7ac346ec7aedbc14765620b9569b971995f63e

HD Wallet
_____
Mnemonic: digital super field employ pulse exact guard suggest brick divide make rug
Base HD Path: m/44'/60'/0'/{account_index}

Gas Price
_____
200000000000
```

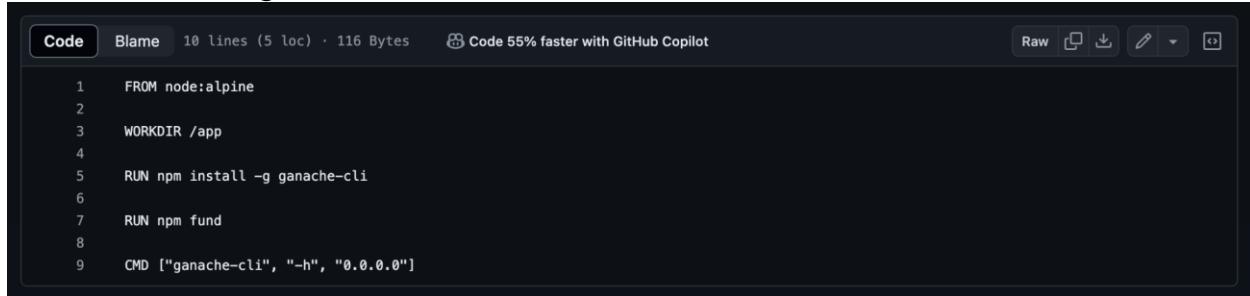
This is the screenshot of our private blockchain that got created after I ran the command ganache-cli -h at localhost. As we can see in the diagram, there are testing accounts along with their private keys in it. We will use these account address with private keys when we will be testing our application.

## 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:



A screenshot of a GitHub code editor interface. At the top, there are tabs for 'Code' (which is selected), 'Blame', and '10 lines (5 loc) · 116 Bytes'. There is also a note 'Code 55% faster with GitHub Copilot'. On the right side, there are icons for 'Raw', 'Copy', 'Download', 'Edit', and 'Preview'. The main area contains the following Dockerfile code:

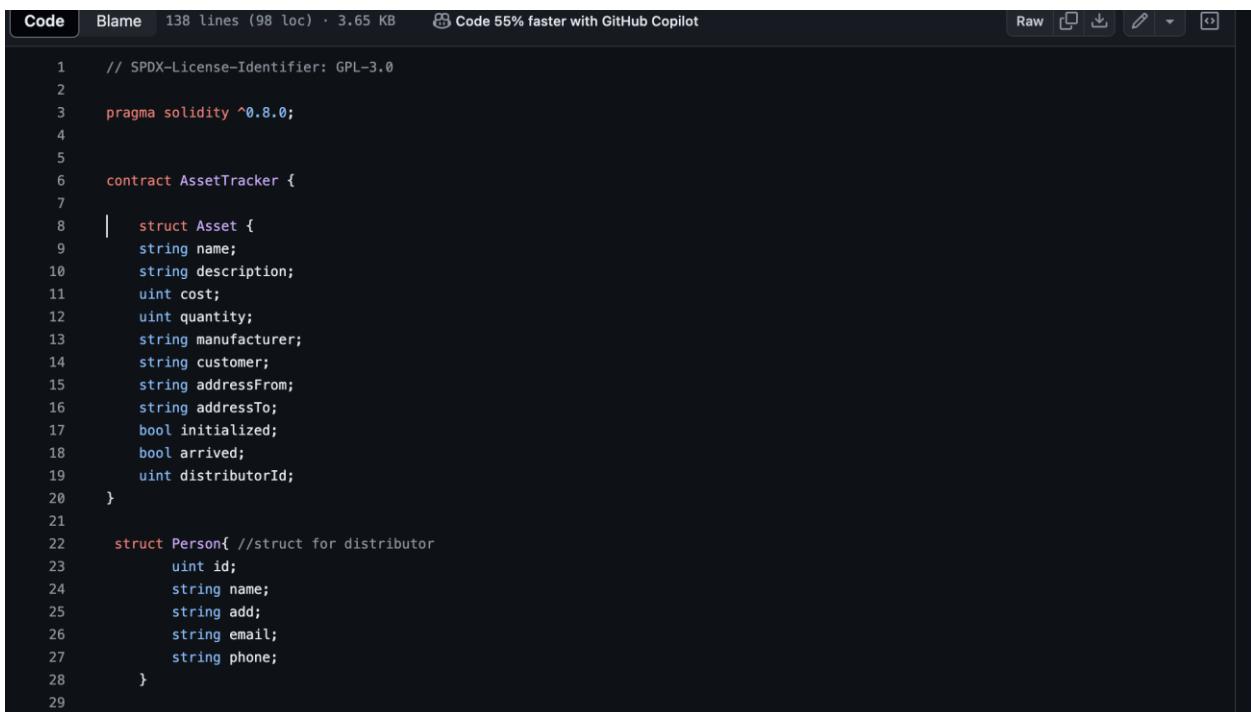
```
1 FROM node:alpine
2
3 WORKDIR /app
4
5 RUN npm install -g ganache-cli
6
7 RUN npm fund
8
9 CMD ["ganache-cli", "-h", "0.0.0.0"]
```

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.



The screenshot shows a GitHub code editor interface with a dark theme. At the top, there are tabs for 'Code' (which is selected), 'Blame', and '138 lines (98 loc) · 3.65 KB'. A GitHub Copilot icon is also present. On the right side, there are standard file operations buttons: 'Raw', 'Copy', 'Download', 'Edit', and 'View'. The main area contains the following Solidity code:

```
1 // SPDX-License-Identifier: GPL-3.0
2
3 pragma solidity ^0.8.0;
4
5
6 contract AssetTracker {
7     struct Asset {
8         string name;
9         string description;
10        uint cost;
11        uint quantity;
12        string manufacturer;
13        string customer;
14        string addressFrom;
15        string addressTo;
16        bool initialized;
17        bool arrived;
18        uint distributorId;
19    }
20
21
22 struct Person{ //struct for distributor
23     uint id;
24     string name;
25     string add;
26     string email;
27     string phone;
28 }
29
```

```

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")),"Distributor already exists");
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 language 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  const {Web3} = require("web3");
2
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 };

```

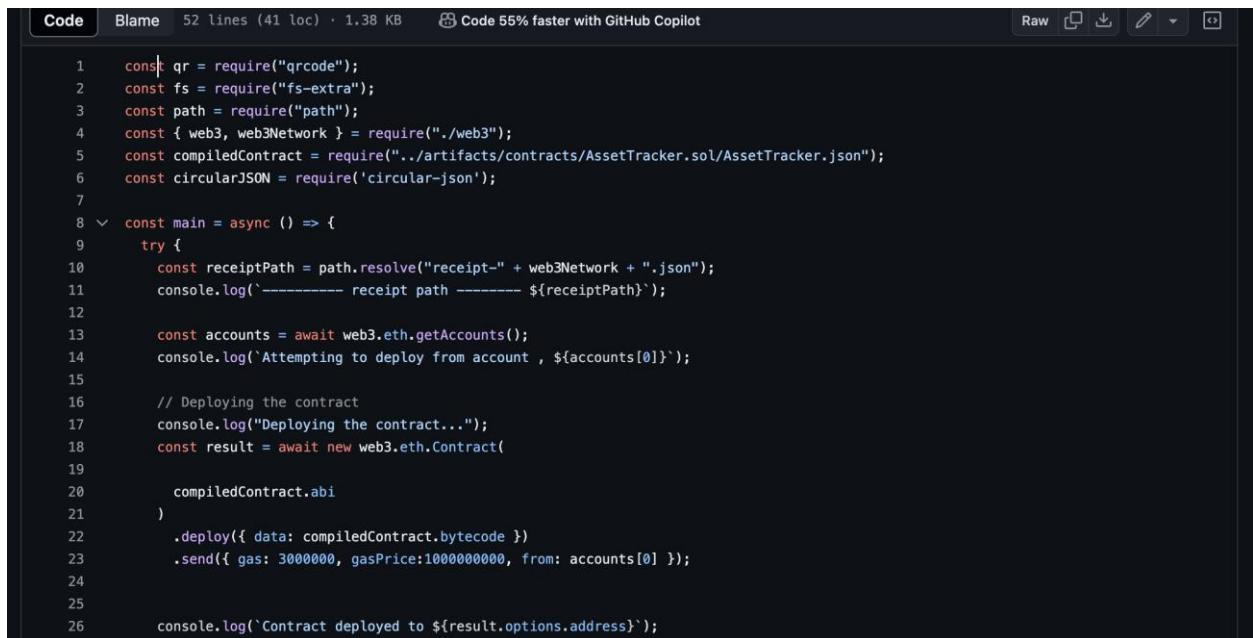
As per our given configuration details, the hardhatNetworkURL is the URL on which the our Ganache Blockchain is running.

**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:



The screenshot shows a GitHub Copilot interface with a dark theme. At the top, there are tabs for 'Code' (which is selected), 'Blame', and 'Raw'. Below the tabs, it says '52 lines (41 loc) · 1.38 KB' and 'Code 55% faster with GitHub Copilot'. On the right side, there are icons for 'Raw', 'Copy', 'Download', 'Edit', and 'Share'. The main area contains the following Node.js code:

```
1 const qr = require("qrcode");
2 const fs = require("fs-extra");
3 const path = require("path");
4 const { web3, web3Network } = require("./web3");
5 const compiledContract = require("../artifacts/contracts/AssetTracker.sol/AssetTracker.json");
6 const circularJSON = require('circular-json');
7
8 const main = async () => {
9   try {
10     const receiptPath = path.resolve("receipt-" + web3Network + ".json");
11     console.log(`----- receipt path ----- ${receiptPath}`);
12
13     const accounts = await web3.eth.getAccounts();
14     console.log(`Attempting to deploy from account , ${accounts[0]}`);
15
16     // Deploying the contract
17     console.log("Deploying the contract...");
18     const result = await new web3.eth.Contract(
19
20       compiledContract.abi
21     )
22     .deploy({ data: compiledContract.bytecode })
23     .send({ gas: 3000000, gasPrice:1000000000, from: accounts[0] });
24
25
26     console.log(`Contract deployed to ${result.options.address}`);
}
```

The screenshot shows a GitHub Copilot interface with a dark theme. At the top, there are tabs for 'Code' (which is selected), 'Blame', and 'Raw'. Below the tabs, the code is displayed in a monospaced font. The code is a Node.js script for deploying a smart contract using Web3.js. It includes imports for 'Web3' and 'path', defines a 'main' function to handle deployment logic, and a 'runMain' function to execute it. The code uses promises and try-catch blocks to manage errors during the deployment process.

```
8  const main = async () => {
22    .deploy({ data: compiledContract.bytecode })
23    .send({ gas: 3000000, gasPrice:1000000000, from: accounts[0] });
24
25
26    console.log(`Contract deployed to ${result.options.address}`);
27
28    console.log("hi");
29    const serialised = circularJSON.stringify(result.options);
30
31    // Writing the receipt to file
32    fs.writeFileSync(receiptPath, result.options);
33
34    console.log("Receipt saved successfully");
35    return serialised;
36  } catch (error) {
37    console.error(error);
38    return error;
39  }
40};
41
42 const runMain = async () => {
43   try {
44     await main();
45     process.exit(0);
46   } catch (error) {
47     console.log(error);
48     process.exit(1);
49   }
50 };
51
52 runMain();
```

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.

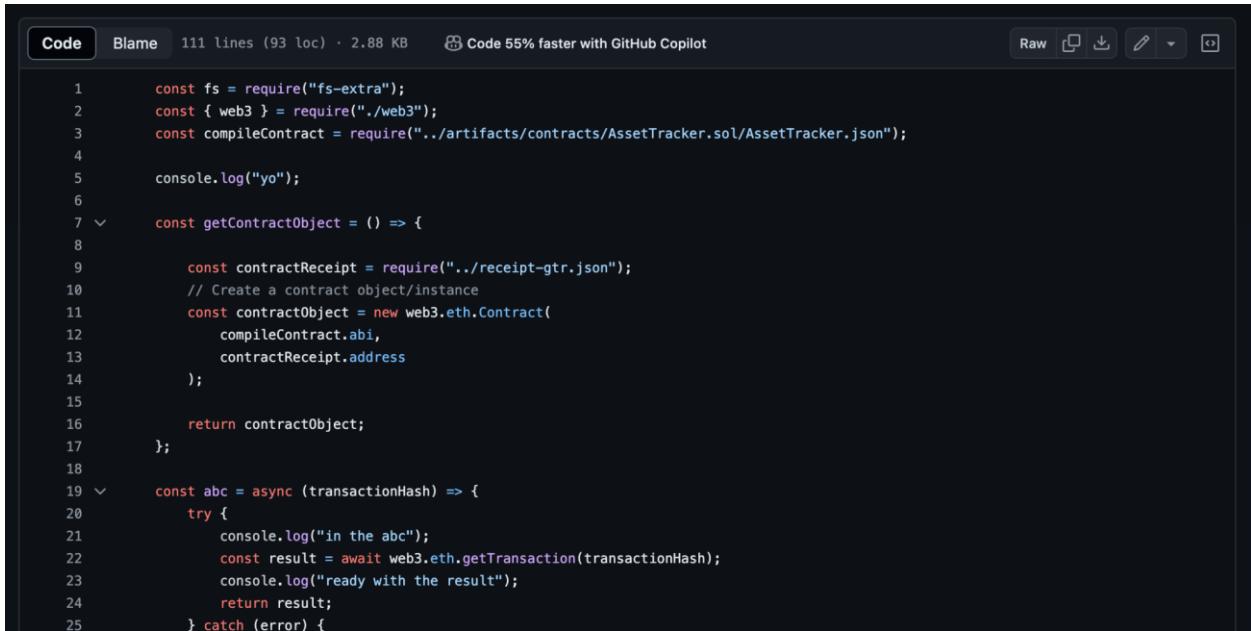
AssetTracker.json is the compiled binary produced during the compilation process.

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



The screenshot shows a GitHub Copilot interface with the following code:

```
1  const fs = require("fs-extra");
2  const { web3 } = require("./web3");
3  const compileContract = require("../artifacts/contracts/AssetTracker.sol/AssetTracker.json");
4
5  console.log("yo");
6
7  const getContractObject = () => {
8
9      const contractReceipt = require("../receipt-gtr.json");
10     // Create a contract object	instance
11     const contractObject = new web3.eth.Contract(
12         compileContract.abi,
13         contractReceipt.address
14     );
15
16     return contractObject;
17 };
18
19 const abc = async (transactionHash) => {
20     try {
21         console.log("in the abc");
22         const result = await web3.eth.getTransaction(transactionHash);
23         console.log("ready with the result");
24         return result;
25     } catch (error) {
```

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**

```
└─# npm install --save-dev hardhat --force
npm WARN using --force Recommended protections disabled.

added 269 packages in 30s

53 packages are looking for funding
  run `npm fund` for details

└─(root㉿kali-linux-2022-2)-[~/home/parallels/mba]
└─# hardhat --version
hardhat: command not found

└─(root㉿kali-linux-2022-2)-[~/home/parallels/mba]
└─# npx hardhat
888   888           888 888           888
888   888           888 888           888
888   888           888 888           888
8888888888 8888b. 888d888 .d88888 88888b. 8888b. 8888888
888   888     "88b 888P" d88" 888 888 "88b     "88b 888
888   888 .d888888 888   888 888 888 .d888888 888
888   888 888 888 Y88b 888 888 888 888 888 Y88b.
888   888 "Y888888 888   "Y88888 888 "Y888888 "Y888

Welcome to Hardhat v2.22.4

? What do you want to do? ...
▶ Create a JavaScript project
Create a TypeScript project
Create a TypeScript project (with Viem)
Create an empty hardhat.config.js
Quit
```

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 your 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/workspace]
# 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
```

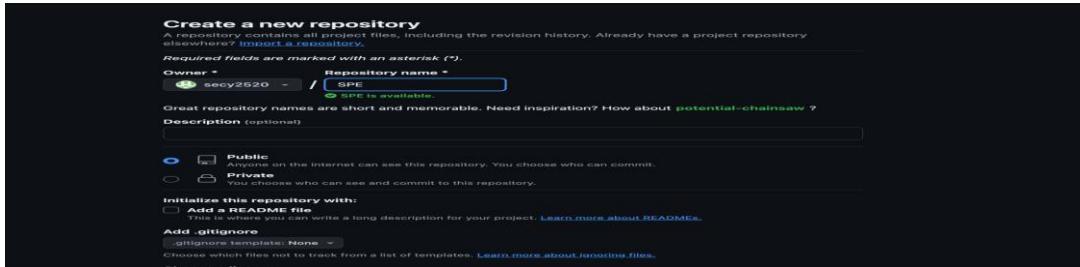
#### Start React App

```
$ 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
```

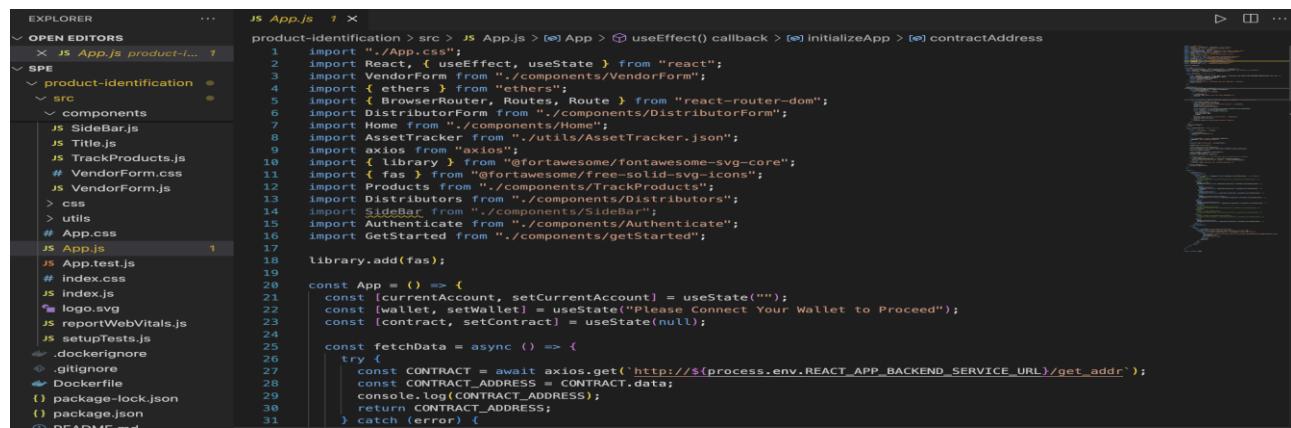
## 7.4 Containerization (Frontend)

Code Blame 11 lines (9 loc) · 300 Bytes GitHub Copilot Raw

```
1 # Use Nginx image as a base image
2 FROM nginx:alpine
3 WORKDIR /app
4 # Copy the built React app from the build folder to the Nginx web server directory
5 COPY build /usr/share/nginx/html
6
7 # Expose port 80 to the outside world
8 EXPOSE 80
9
10 # Command to run the Nginx server
11 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



The screenshot shows a code editor interface with the following details:

- EXPLORER:** Shows the project structure:
  - OPEN EDITORS: `App.js`
  - SPE: `product-identification`
  - src: `SideBar.js`, `Title.js`, `TrackProducts.js`, `VendorForm.css`, `VendorForm.js`, `css`, `utils`, `App.css`
  - App.js
  - App.test.js
  - index.css
  - index.js
  - logo.svg
  - reportWebVitals.js
  - setupTests.js
  - .dockerignore
  - .gitignore
  - Dockerfile
  - package-lock.json
  - package.json
  - README.md
- Code Editor:** The `App.js` file is open, showing the following code:

```
1 import './App.css';
2 import React, { useEffect, useState } from "react";
3 import VendorForm from "./components/VendorForm";
4 import { ethers } from "ethers";
5 import { BrowserRouter, Route } from "react-router-dom";
6 import DistributorForm from "./components/DistributorForm";
7 import Home from "./components/Home";
8 import AssetTracker from "./utils/AssetTracker.json";
9 import axios from "axios";
10 import { library } from "@fortawesome/fontawesome-svg-core";
11 import { fas } from "@fortawesome/free-solid-svg-icons";
12 import Products from "./components/TrackProducts";
13 import Distributors from "./components/Distributors";
14 import SideBar from "./components/SideBar";
15 import Authenticate from "./components/Authenticate";
16 import GetStarted from "./components/getStarted";
17
18 library.add(fas);
19
20 const App = () => {
21   const [currentAccount, setCurrentAccount] = useState("");
22   const [wallet, setWallet] = useState("Please Connect Your Wallet to Proceed");
23   const [contract, setContract] = useState(null);
24
25   const fetchData = async () => {
26     try {
27       const CONTRACT = await axios.get(`http://${process.env.REACT_APP_BACKEND_SERVICE_URL}/get_addr`);
28       const CONTRACT_ADDRESS = CONTRACT.data;
29       console.log(CONTRACT_ADDRESS);
30       return CONTRACT_ADDRESS;
31     } catch (error) {
```

- 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**



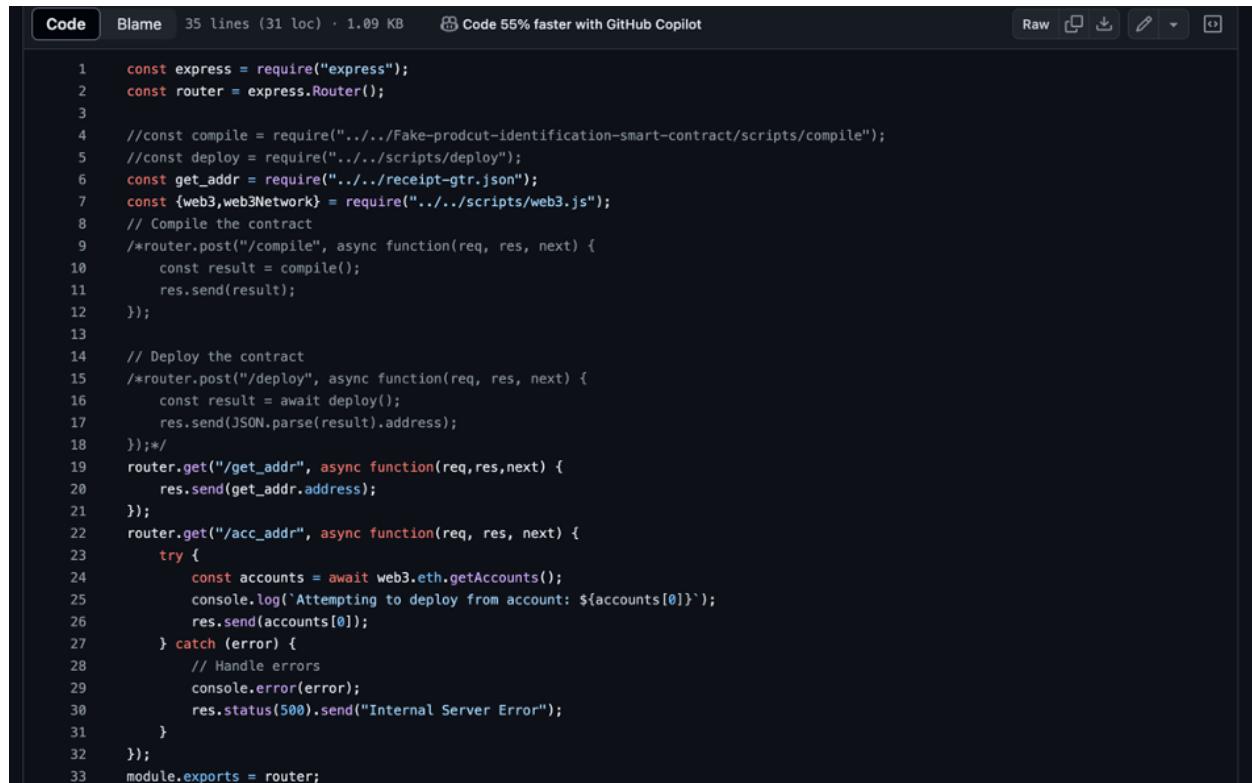
The screenshot shows a code editor interface for a file named 'Index.js'. The code is written in JavaScript and defines a web server using the Express framework. It includes imports for express, bodyParser, and cors, and requires two API route files: 'contract-API' and 'smart-contract-API'. The server is configured to listen on port 4000, use CORS, and parse JSON bodies. It also sets up bodyParser.urlencoded() with specific limits and extended parameters. Finally, it uses the routes defined in the imported files.

```
1 const express = require("express");
2 const bodyParser = require("body-parser");
3 const cors = require("cors");
4 const app = express();
5
6
7 const contractAPIRoutes = require("./contract-API");
8 const smartContractAPIRoutes = require("./smart-contract-API");
10 const port = 4000;
12
13 app.use(cors());
14 app.use(bodyParser.json());
15 app.use(
16   bodyParser.urlencoded({
17     limit:"50mb",
18     extended:false,
19     parameterLimit:50000
20   })
21 );
22
23 // use the routes specified in route folder
24
25 app.use("/", contractAPIRoutes);
26
27 app.use("/",smartContractAPIRoutes);
```

- **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**



The screenshot shows a GitHub Copilot interface with the following details:

- Code tab selected.
- Blame tab.
- 35 lines (31 loc) · 1.09 KB.
- Code 55% faster with GitHub Copilot.
- Raw, Copy, Download, Edit, Undo, Redo buttons.

```
1 const express = require("express");
2 const router = express.Router();
3
4 //const compile = require("../Fake-produt-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 like this:

Code Blame 16 lines (13 loc) · 721 Bytes  Code 55% faster with GitHub Copilot

```
FROM node:alpine
WORKDIR /app
#RUN apk update && apk upgrade && apk add --no-cache bash git openssh
RUN apk add --no-cache --repository http://dl-cdn.alpinelinux.org/alpine/edge/main --repository http://dl-cdn.alpinelinux.org/alpine/edge/community bash git openssh
RUN apk update && apk upgrade && apk add --no-cache bash git openssh
RUN apk update && apk add python3 && apk add icu-libs krb5-libs libgcc libintl libssl3 krb5-libs gcc make g++ krb5-dev
RUN apk update
RUN npm install -g npm@10.5.2
RUN npm install "express" "cors" "fs-extra" "web3" "qrcode" "circular-json"
ADD ./server ./server
ADD ./artifacts ./artifacts
ADD ./scripts ./scripts
CMD ["sh", "-c", "node ./scripts/deploy.js && node ./server/routes/index.js"]
```

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.

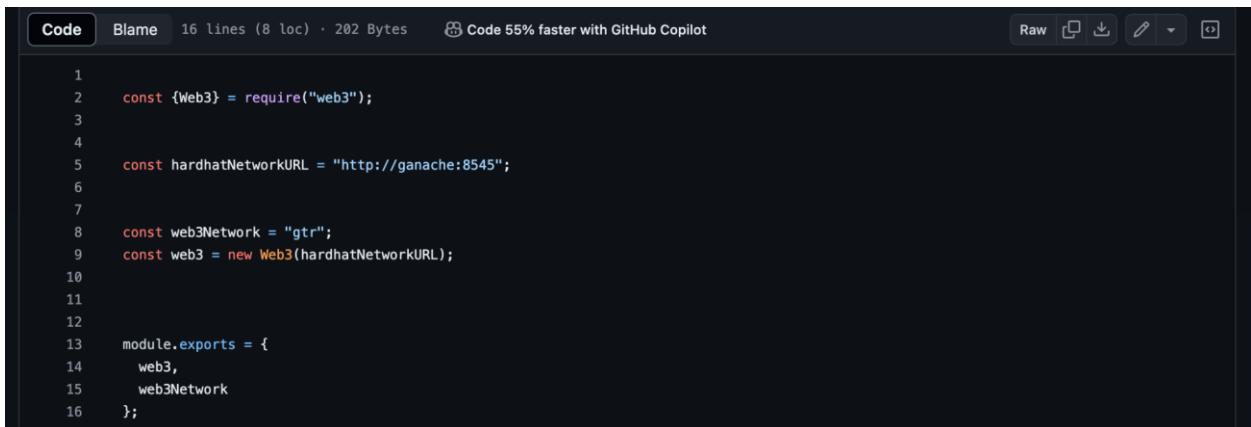
**scripts** directory contains all those scripts which are useful for interaction in the backend.

# 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:

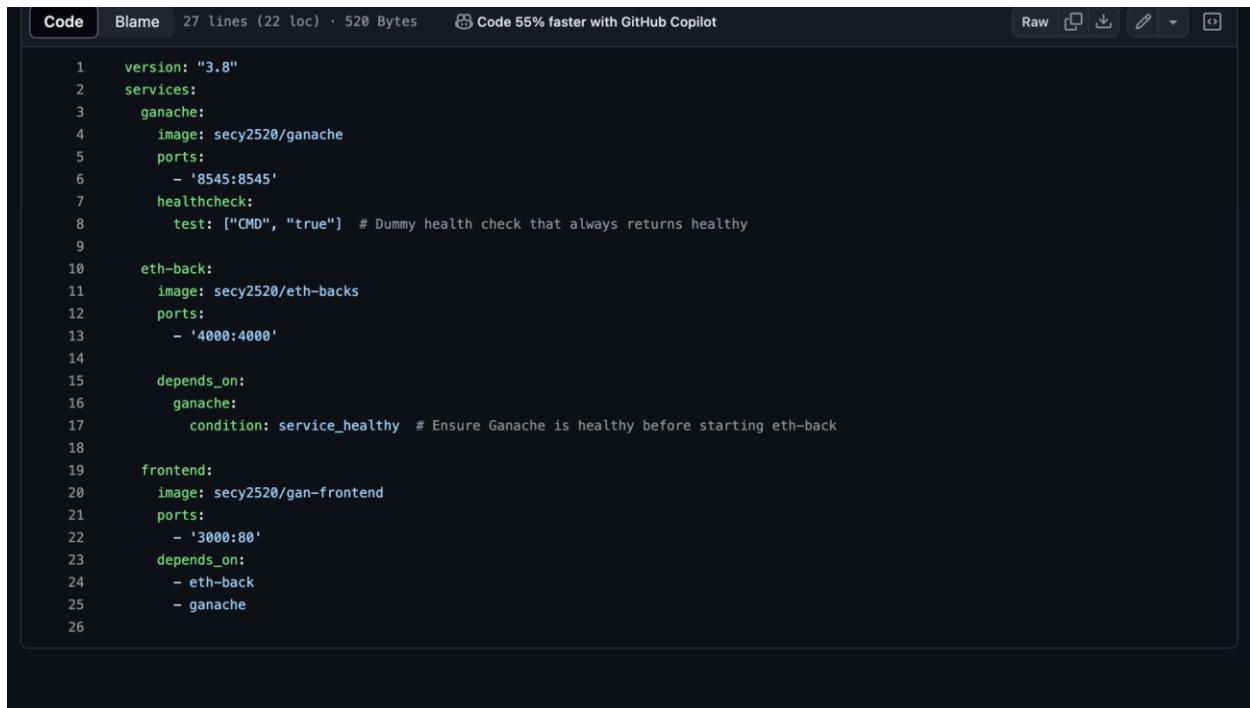
- All three containers have a dependency among them i.e. they should execute in a particular order otherwise one of the container might fail to get start.
- Frontend is dependent on backend and backend is dependent on blockchain.
- First the blockchain will be deployed. As soon as the blockchain is available, 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



The screenshot shows a GitHub code editor interface. At the top, there are tabs for 'Code' (which is selected), 'Blame', and 'Raw'. Below the tabs, it says '16 lines (8 loc) · 202 Bytes'. There is also a note 'Code 55% faster with GitHub Copilot'. On the right side, there are download and edit buttons. The code itself is a snippet of JavaScript:

```
1 const {Web3} = require("web3");
2
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 };
```

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



The screenshot shows a GitHub Copilot interface with a dark theme. At the top, there are tabs for 'Code' (which is selected), 'Blame', and 'Code 55% faster with GitHub Copilot'. Below the tabs, the code editor displays a Docker Compose file. The file defines three services: 'ganache', 'eth-back', and 'frontend'. The 'ganache' service has an image of 'secy2520/ganache', port 8545 mapped to 8545, and a healthcheck command of '["CMD", "true"]'. The 'eth-back' service has an image of 'secy2520/eth-backs', port 4000 mapped to 4000, and depends on 'ganache' with a condition of 'service\_healthy'. The 'frontend' service has an image of 'secy2520/gan-frontend', port 3000 mapped to 80, and depends on both 'eth-back' and 'ganache'. The code is numbered from 1 to 26.

```
1 version: "3.8"
2 services:
3   ganache:
4     image: secy2520/ganache
5     ports:
6       - '8545:8545'
7     healthcheck:
8       test: ["CMD", "true"] # Dummy health check that always returns healthy
9
10  eth-back:
11    image: secy2520/eth-backs
12    ports:
13      - '4000:4000'
14
15    depends_on:
16      ganache:
17        condition: service_healthy # Ensure Ganache is healthy before starting eth-back
18
19  frontend:
20    image: secy2520/gan-frontend
21    ports:
22      - '3000:80'
23    depends_on:
24      - eth-back
25      - ganache
26
```

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:

Code Blame 127 lines (115 loc) · 3.86 KB ⚡ Code 55% faster with GitHub Copilot

```
1 pipeline {
2     agent any
3
4     tools {
5
6         git "Default"
7         nodejs 'nodejs'
8
9     }
10    environment {
11        DOCKER_GAN_NAME = 'ganache'
12        DOCKER_ETH_NAME = 'eth-backs'
13        DOCKER_FRONTEND = 'gan-frontend'
14        GITHUB_REPO_URL = 'https://github.com/Priyansuvaish/SPE.git'
15    }
16
17    stages {
18        stage('Checkout') {
19            steps {
20                script {
21                    git branch: 'main', url: "${GITHUB_REPO_URL}"
22                    sh 'docker --version'
23                }
24            }
25        }
26    }
27}
```

```

26     stage('Installing hardhat')
27     {
28         steps {
29             script {
30                 dir("/var/lib/jenkins/workspace/eth-project/eth-backs/")
31                 {
32                     sh 'npm install --save-dev hardhat --force'
33                     sh 'node -v'
34                 }
35             }
36         }
37     }
38 }
39 stage('Building frontend code') {
40     steps {
41         script {
42             dir("/var/lib/jenkins/workspace/eth-project/product-identification/")
43                 {
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                 {
57                     sh 'npx hardhat compile'
58                 }
59         }
60     }
61 }
```

```

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 stage('Push Docker Images') {
91     steps {
92         script{
93             docker.withRegistry('', 'docker') {
94                 sh 'docker tag eth-backs secy2520/eth-backs:latest'
95                 sh 'docker tag ganache secy2520/ganache:latest'
96                 sh 'docker tag gan-frontend secy2520/gan-frontend:latest'
97                 sh 'docker push secy2520/eth-backs'
98                 sh 'docker push secy2520/ganache'
99                 sh 'docker push secy2520/gan-frontend'
100            }
101        }
102    }
103 }
```

```

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 }
104 stage('Removing all present images and the containers ') {
105   steps {
106     script{
107
108       sh 'docker system prune -f'
109
110     }
111   }
112 }
113 stage('Run Ansible Playbook') {
114   steps {
115     script {
116       ansiblePlaybook(
117         playbook: 'deploy.yml',
118         inventory: 'inventory'
119       )
120     }
121   }
122 }
123 }
124
125
126 }
127 }
```

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.

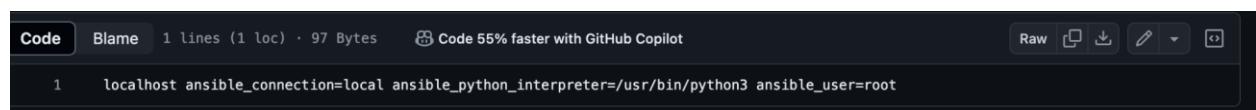
## 10.2 Writing the Ansible Playbook(Deploy.yml)



```
Code Blame 8 lines (8 loc) · 218 Bytes GitHub Copilot
1 - name: Run Docker Compose script
2   hosts: localhost
3   # become: yes
4   tasks:
5     - name: Start Docker Compose services
6       shell: docker-compose up
7       args:
8         chdir: /var/lib/jenkins/workspace/eth-project/
```

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



```
Code Blame 1 lines (1 loc) · 97 Bytes GitHub Copilot
1 localhost ansible_connection=local ansible_python_interpreter=/usr/bin/python3 ansible_user=root
```

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 Jenkins 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.

The screenshot shows the Jenkins Manage Jenkins > Plugins page with a search bar at the top containing the text "docker". Below the search bar, there is a table listing several Jenkins plugins related to Docker:

Name	Enabled
Docker API Plugin 3.3.4-86.v39b_a_5ede342c This plugin provides <a href="#">docker-java</a> API for other plugins. <a href="#">Report an issue with this plugin</a>	<input checked="" type="checkbox"/> <a href="#">3.3.4...</a> <a href="#">X</a>
This plugin is up for adoption! We are looking for new maintainers. Visit our <a href="#">Adopt a Plugin</a> initiative for more information.	
Docker Commons Plugin 439.va_3cb_0a_6a_fb_29 Provides the common shared functionality for various Docker-related plugins. <a href="#">Report an issue with this plugin</a>	<input checked="" type="checkbox"/> <a href="#">X</a>
Docker Compose Build Step Plugin 1.0 Docker Compose plugin for Jenkins <a href="#">Report an issue with this plugin</a>	<input checked="" type="checkbox"/> <a href="#">X</a>
Docker Pipeline 572.v950f58993843 Build and use Docker containers from pipelines. <a href="#">Report an issue with this plugin</a>	<input checked="" type="checkbox"/> <a href="#">X</a>
Docker plugin 1.6.1 This plugin integrates Jenkins with Docker <a href="#">Report an issue with this plugin</a>	<input checked="" type="checkbox"/> <a href="#">1.6.1</a> <a href="#">X</a>
Docker Slaves Plugin 1.0.7 Uses Docker containers to run Jenkins build agents. <a href="#">Report an issue with this plugin</a>	<input checked="" type="checkbox"/> <a href="#">X</a>
docker-build-step 2.11 This plugin allows to add various docker commands to your job as build steps. <a href="#">Report an issue with this plugin</a>	<input checked="" type="checkbox"/>

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

The screenshot shows the Jenkins plugin manager interface. A search bar at the top contains the text "node". Below it, a list item for "NodeJS Plugin 1.6.1" is shown. The "Enabled" status is indicated by a blue toggle switch with a checkmark. To the right of the switch is a red circular icon with a white "X". Below the list item, there is a link to "Report an issue with this plugin".

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

The screenshot shows the Jenkins plugin manager interface. A search bar at the top contains the text "Ansib". Below it, a list item for "Ansible plugin 307.va\_1f3ef06575a\_" is shown. The "Enabled" status is indicated by a blue toggle switch with a checkmark. To the right of the switch is a red circular icon with a white "X". Below the list item, there is a link to "Report an issue with this plugin".

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

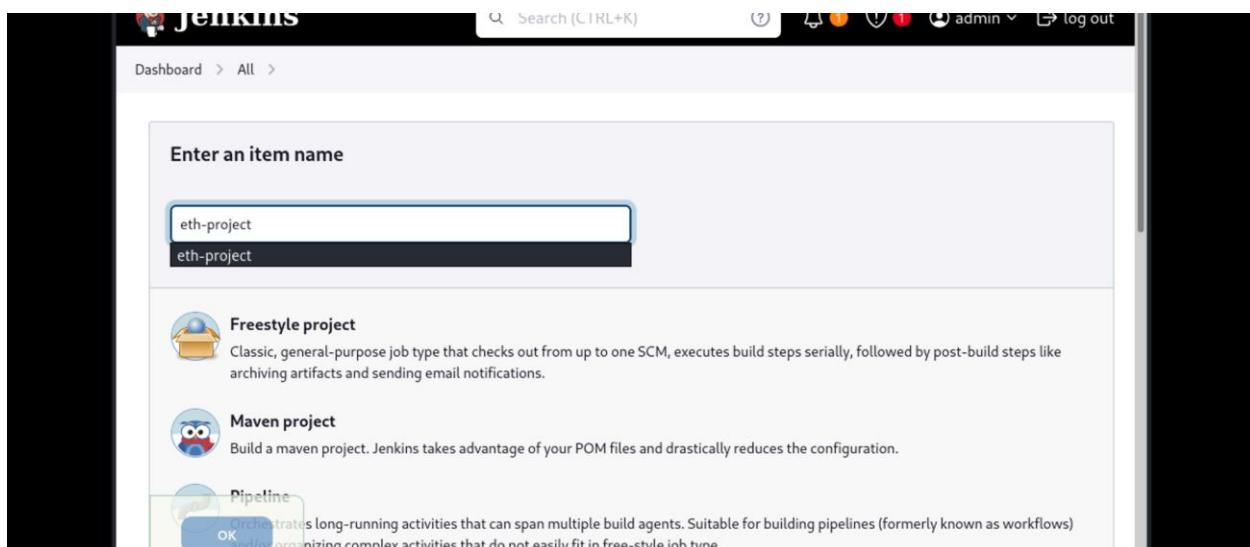
The screenshot shows the Jenkins "Credentials" management page. The URL in the browser is "Dashboard > Manage Jenkins > Credentials". The page title is "Credentials". There is a table with columns: Type, P, Store, Domain, ID, and Name. One row is visible, showing "System" under Type, "docker" under ID, and "secy2520/\*\*\*\*\*" under Name. The "Domain" column shows "(global)".

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

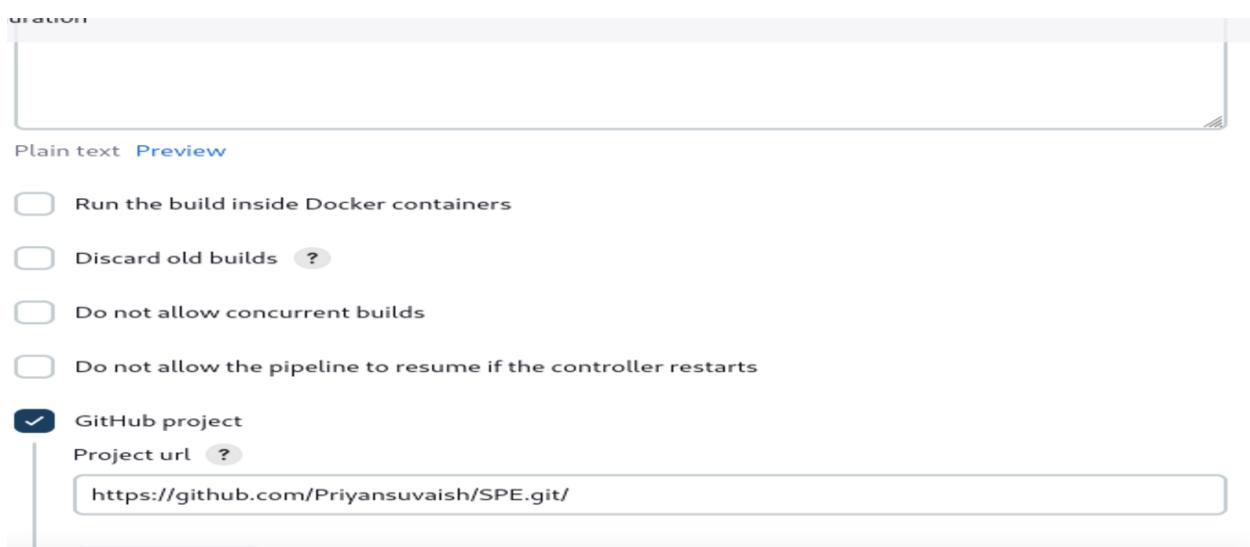
## 10.5 Running the pipeline

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

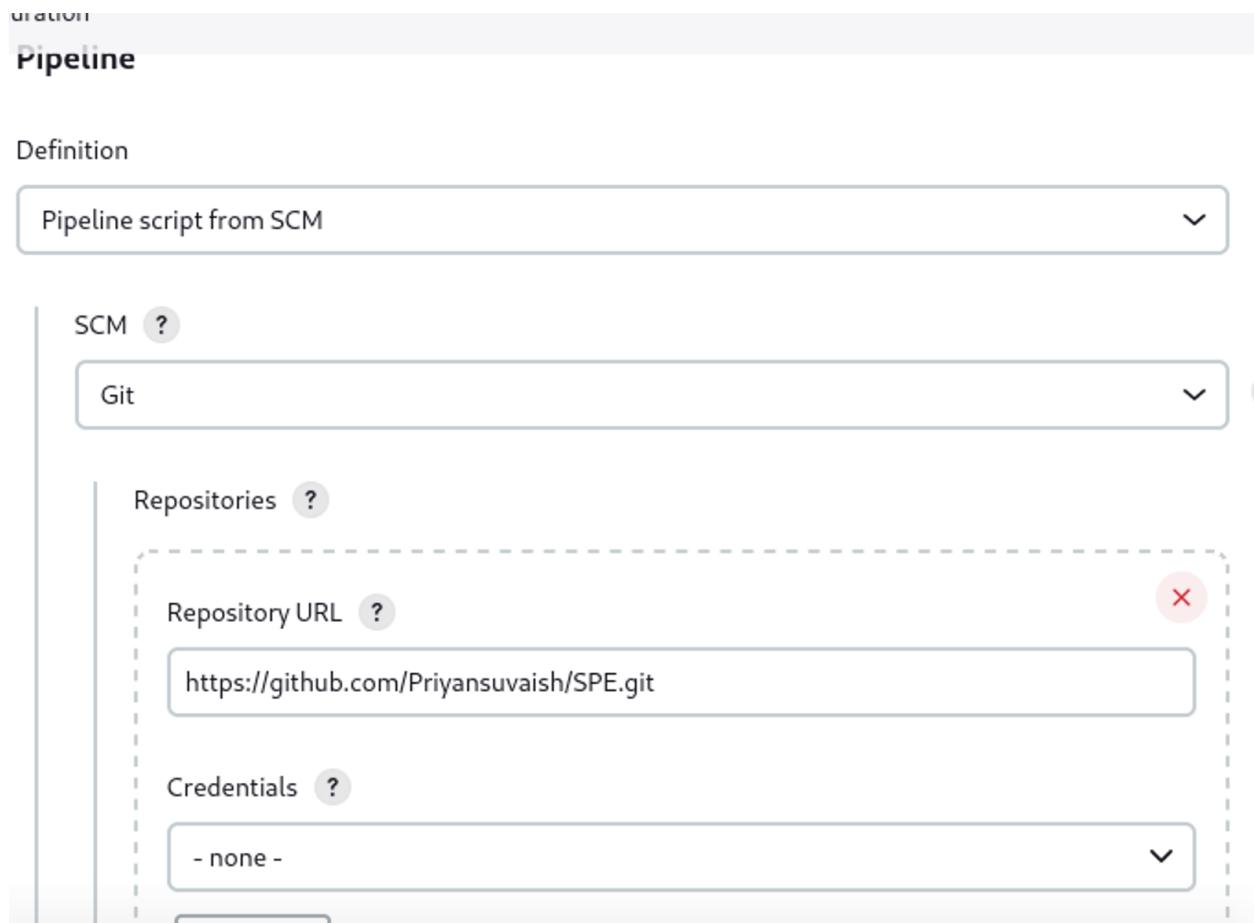
- Go to dashboard and click on **New Item**.
- After entering the item name, choose the Job type as **Pipeline**. In our case, we choose the job name as **eth-project**.



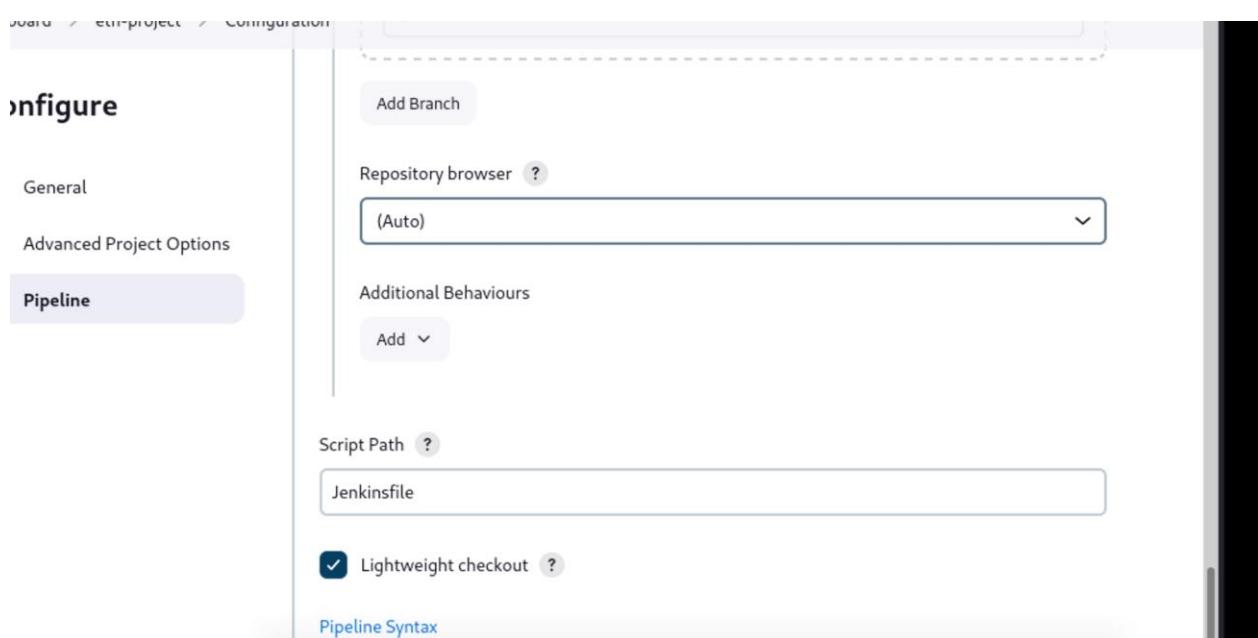
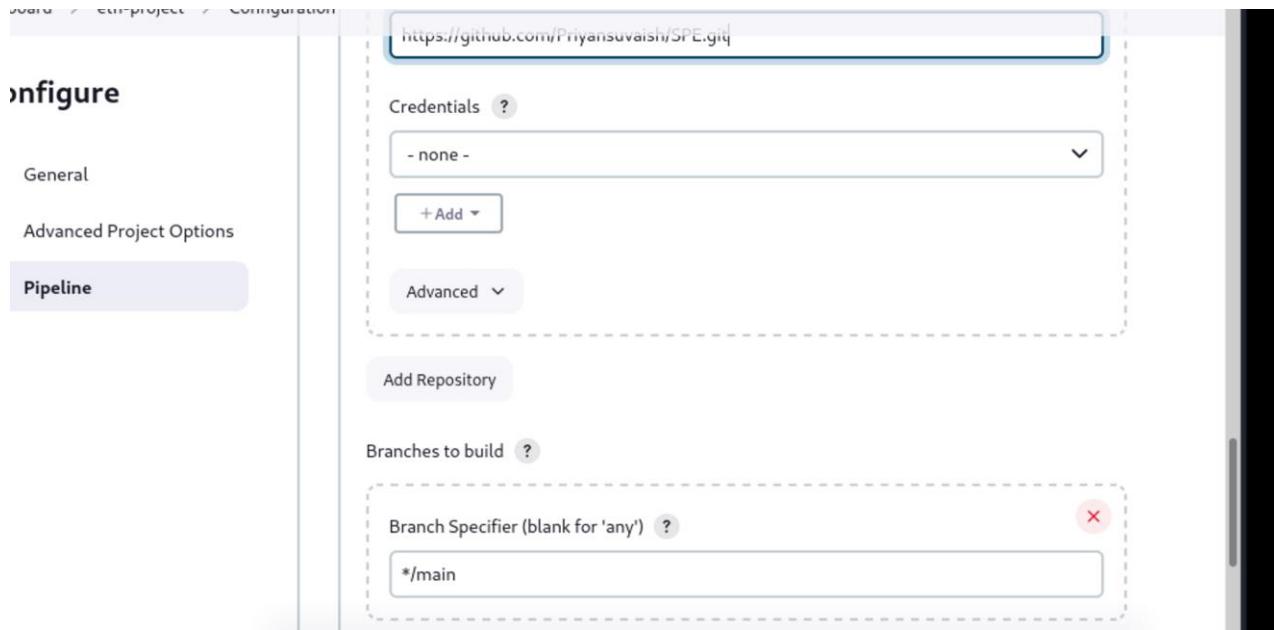
- After making the pipeline, lets configure the pipeline. To do this, click on the Job and since our project is hosted on github, pass the URL of the project in the **Github Project** section



- 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 like this:



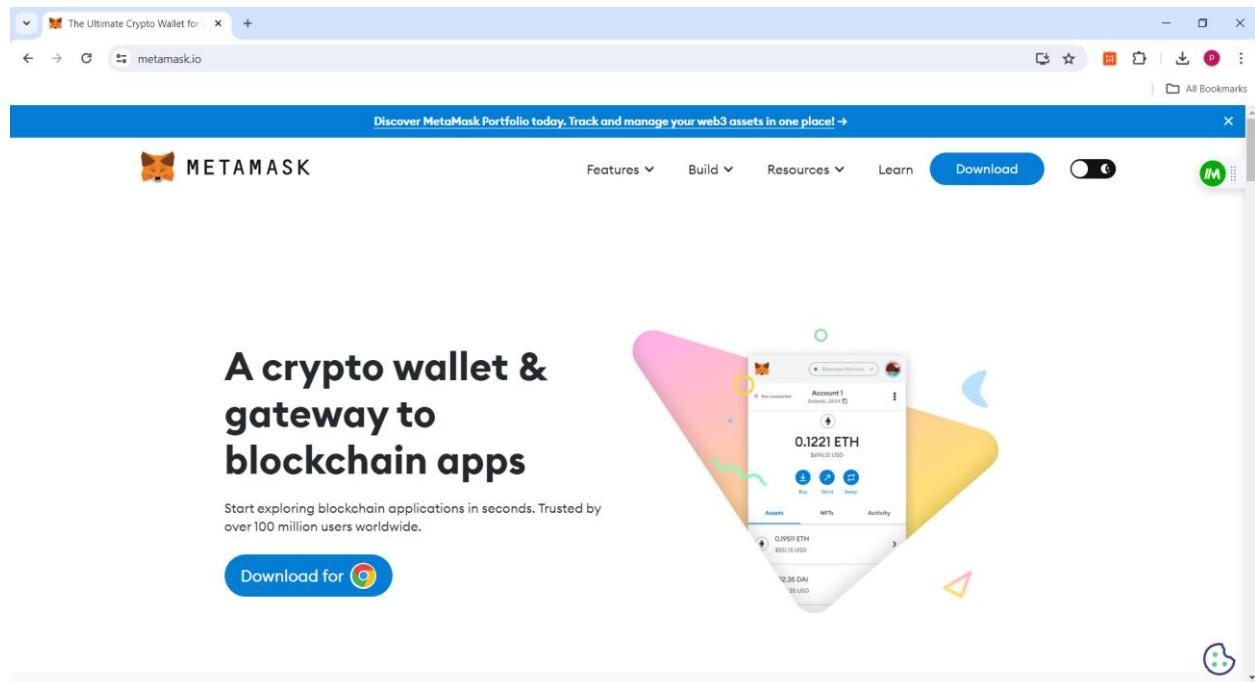
All the stages marked with green shows that pipeline got build without any conflict between packages and hence we were successful in doing that.

## 11 Testing and Results

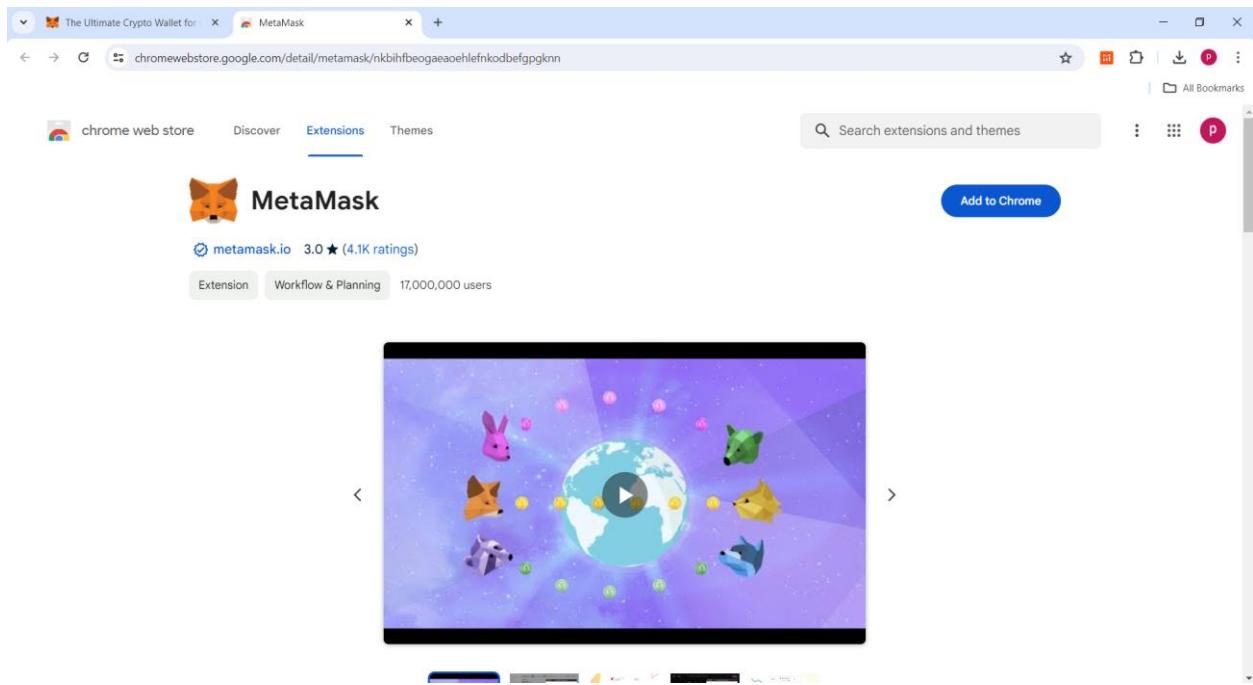
These are the some of the screen shots almost covers all major functionalities:

### 11.1 Setting up the Metamask1

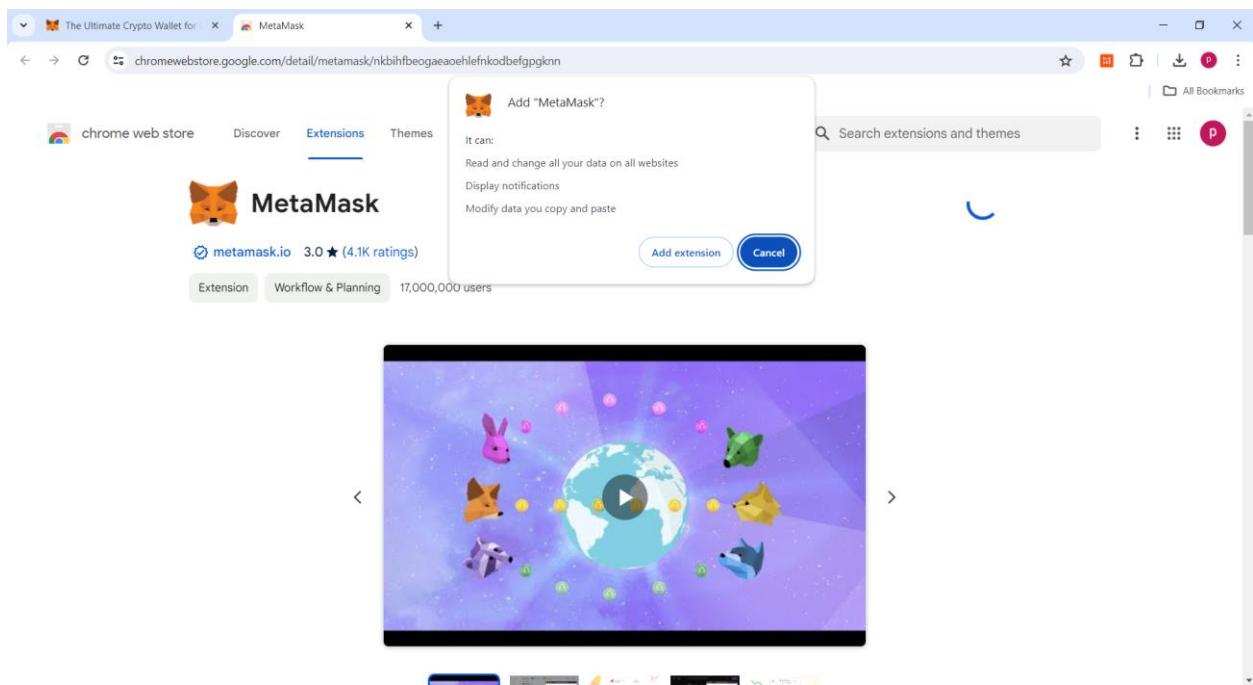
Open the browser and in the address bar at the top of the browser, type 'metamask.io' and press 'Enter'.



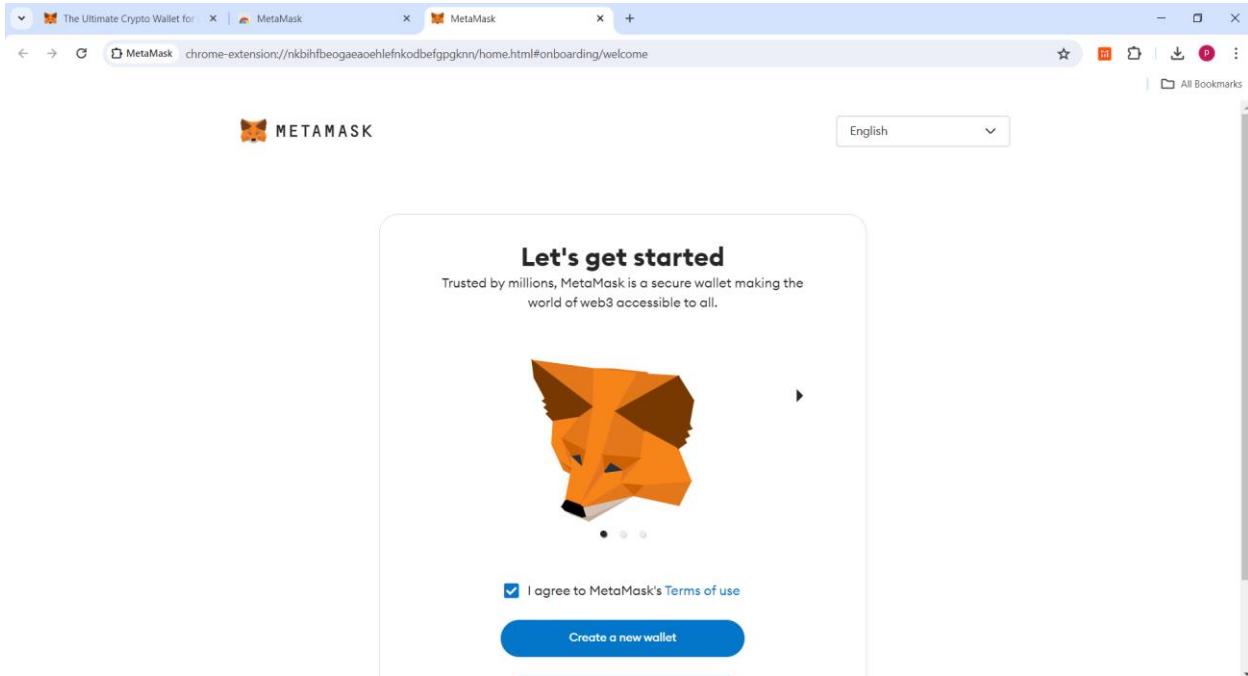
Click on 'Download for chrome'



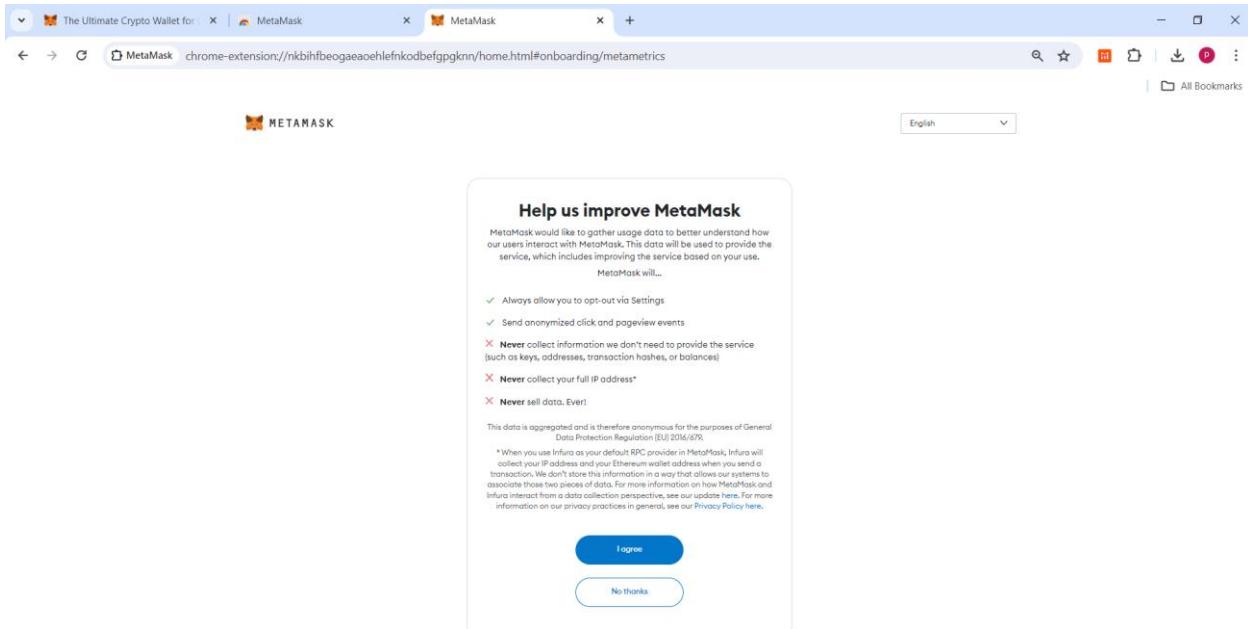
Click on 'Add to chrome'



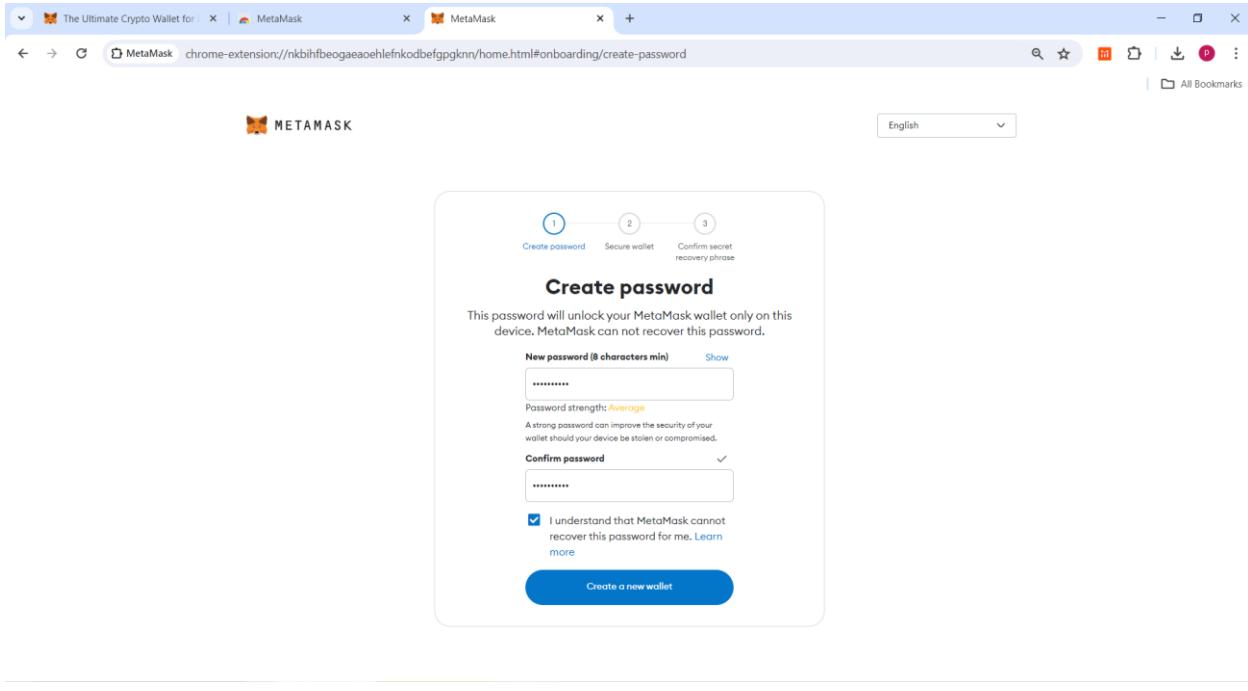
After downloading MetaMask, check the "I agree to MetaMask's Terms of use" box.



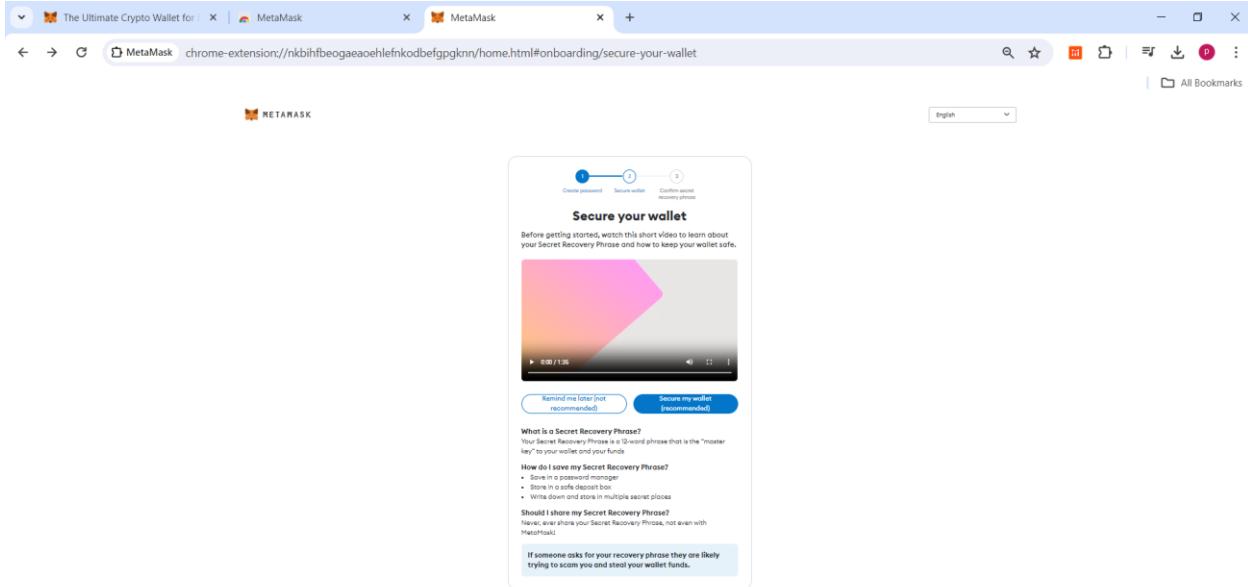
Click on 'Create a new wallet'



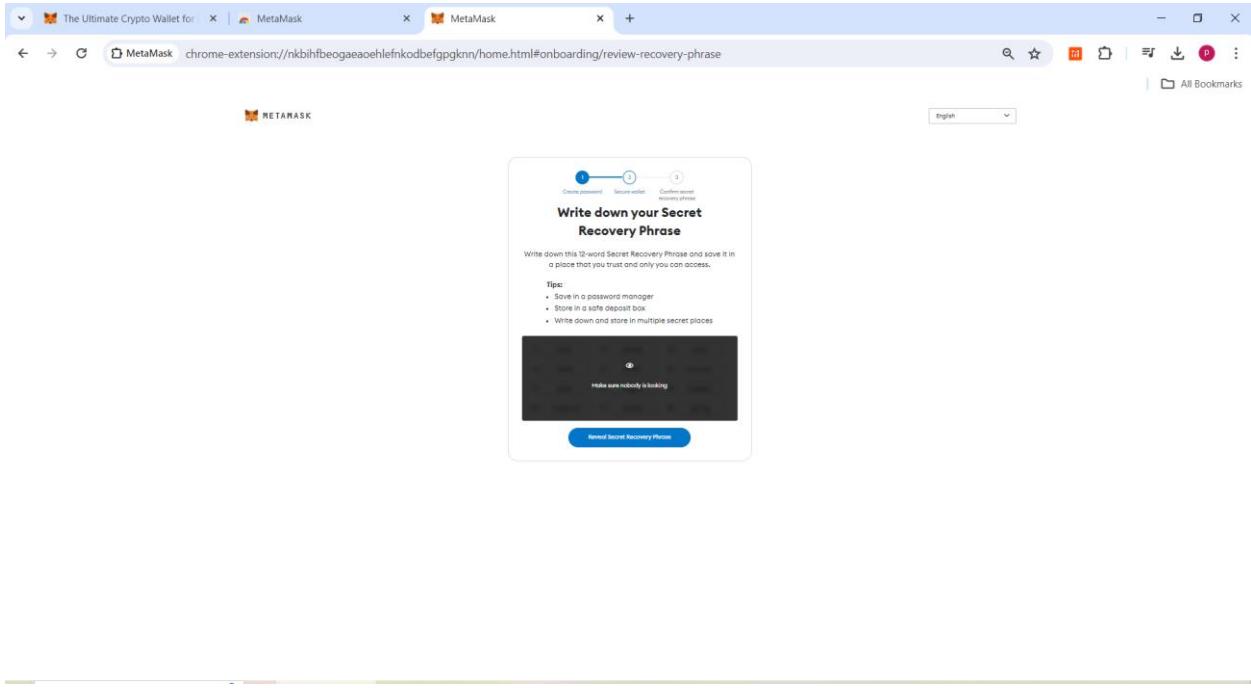
Click on 'I agree'



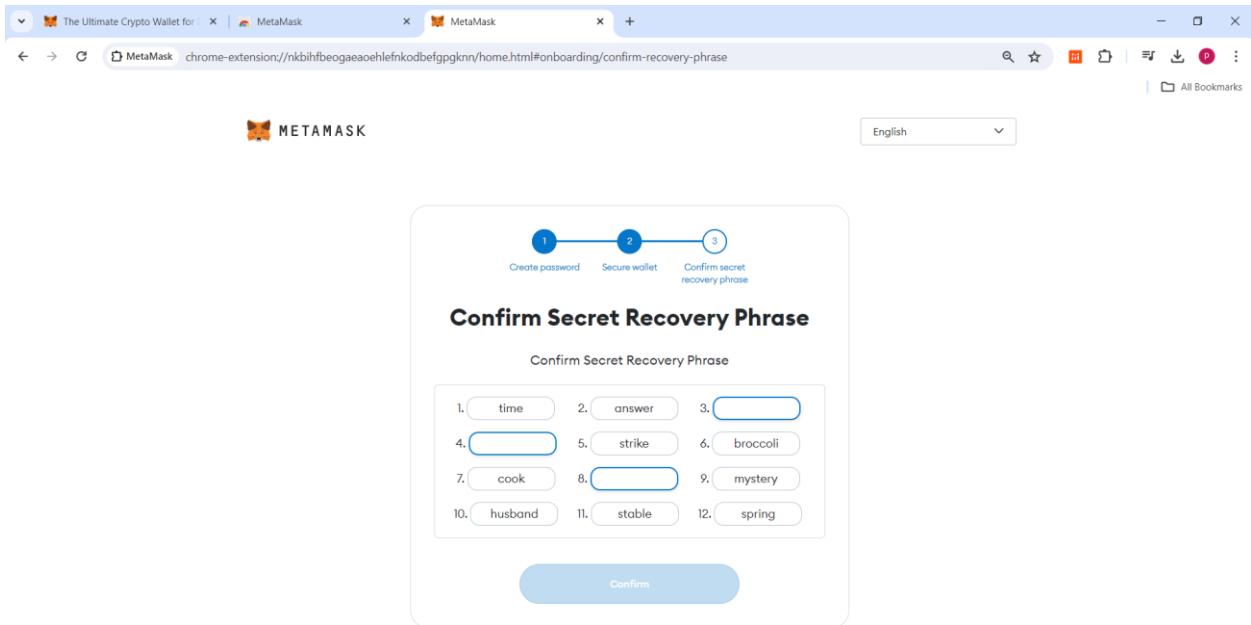
Fill in the details, create a password, check the "I understand that MetaMask cannot recover this password for me" box, then click "Create a new wallet."



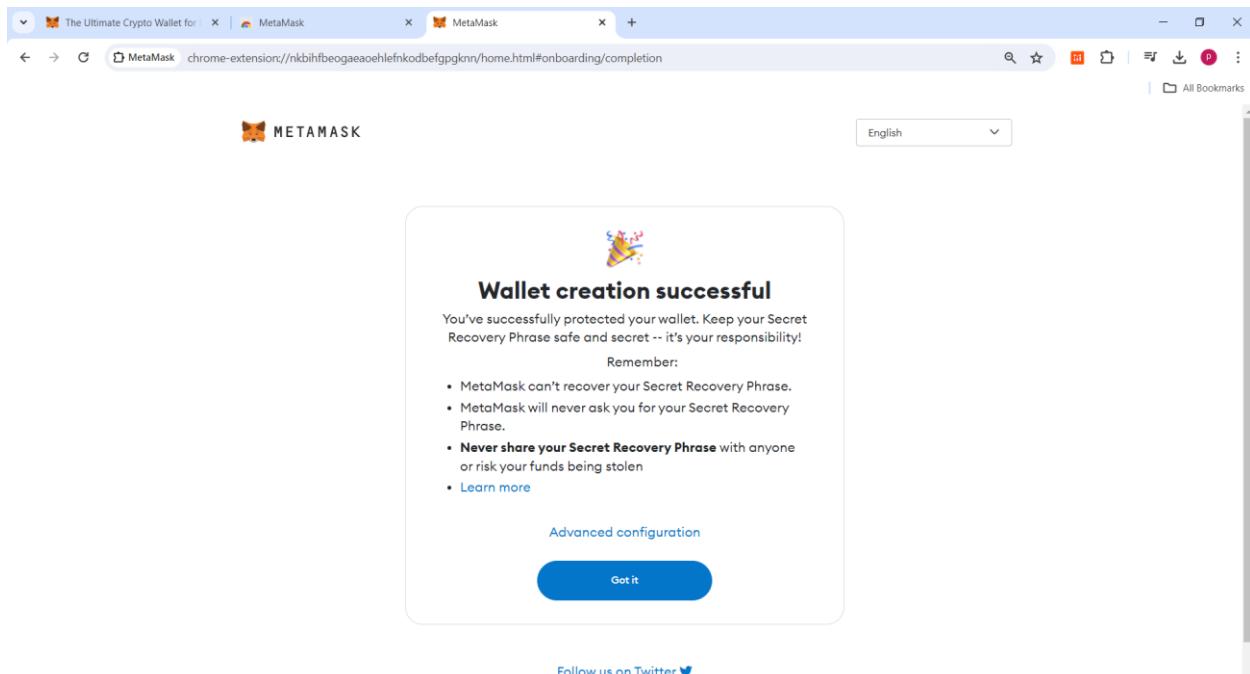
Click on 'Secure my wallet'



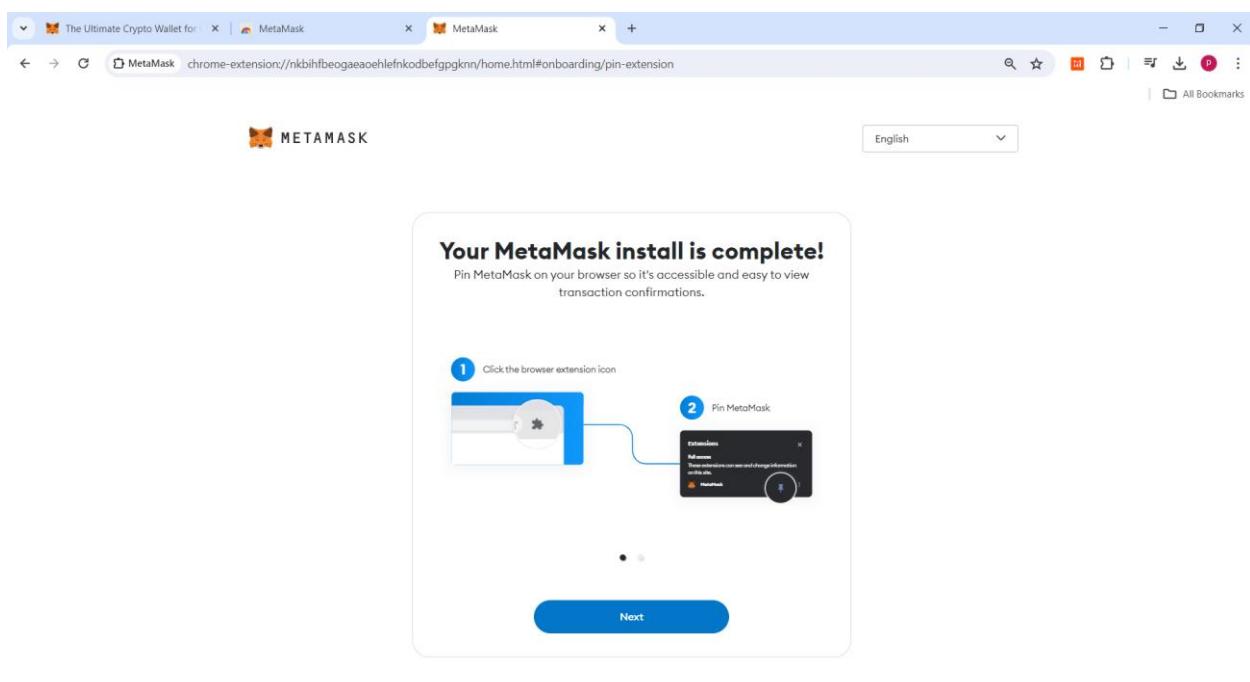
Click on 'Reveal Secret Recovery Phrase' and take the screenshot of the key then press 'Next'.



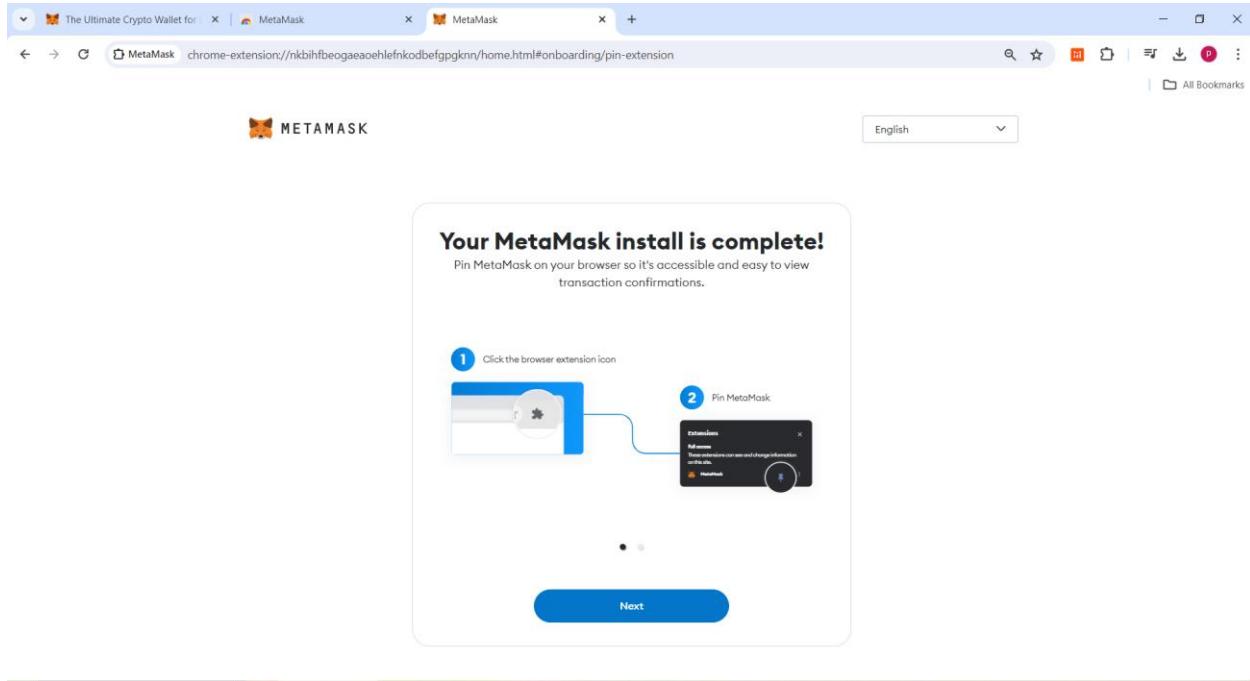
Fill the box as per the screenshot and click on 'Confirm'.



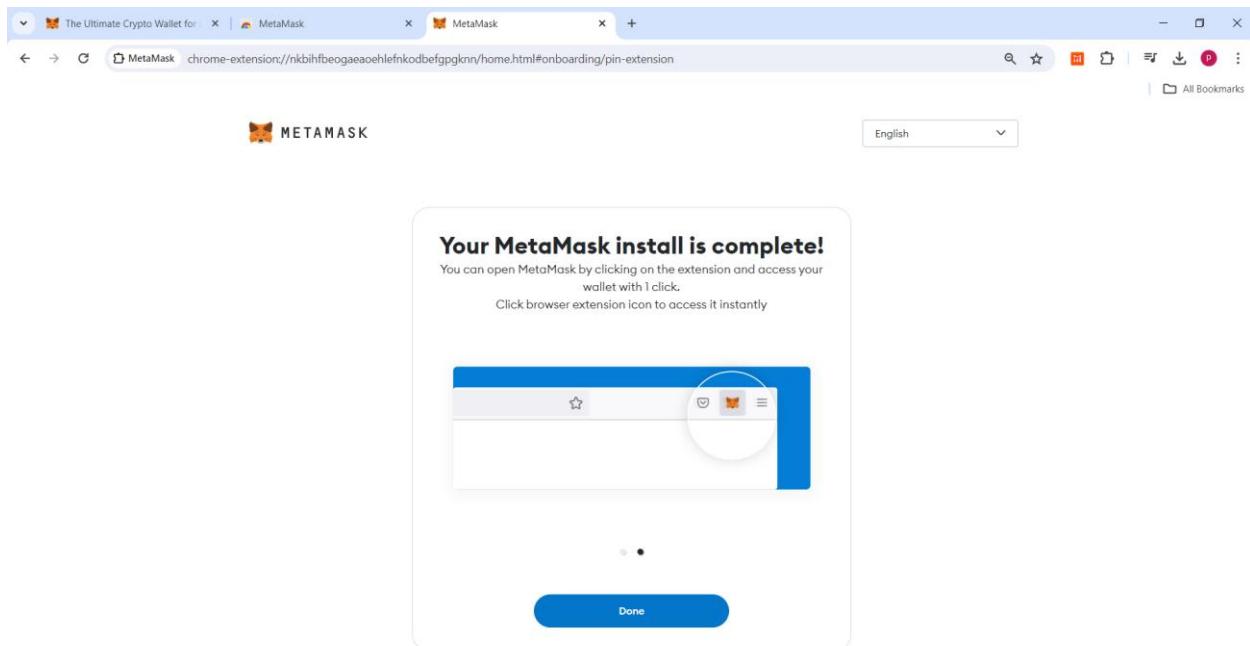
Click on 'Got it'



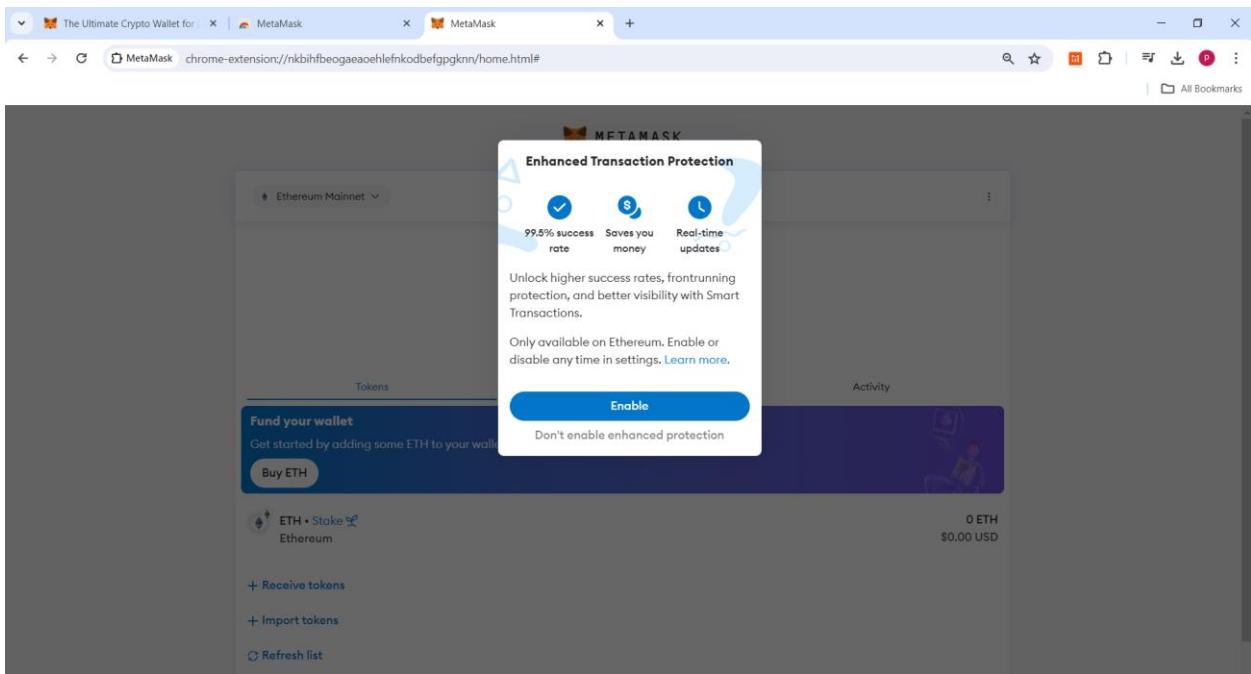
Click on 'Next'



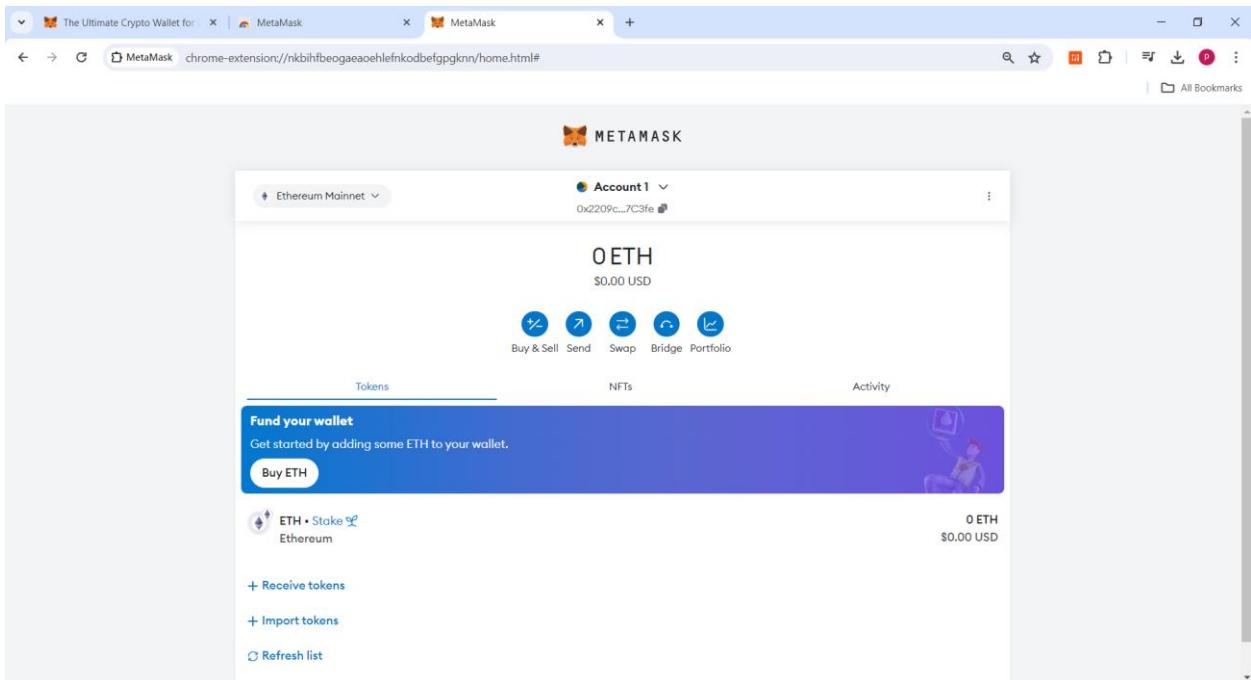
Click on 'Done'



Click on 'Enable'

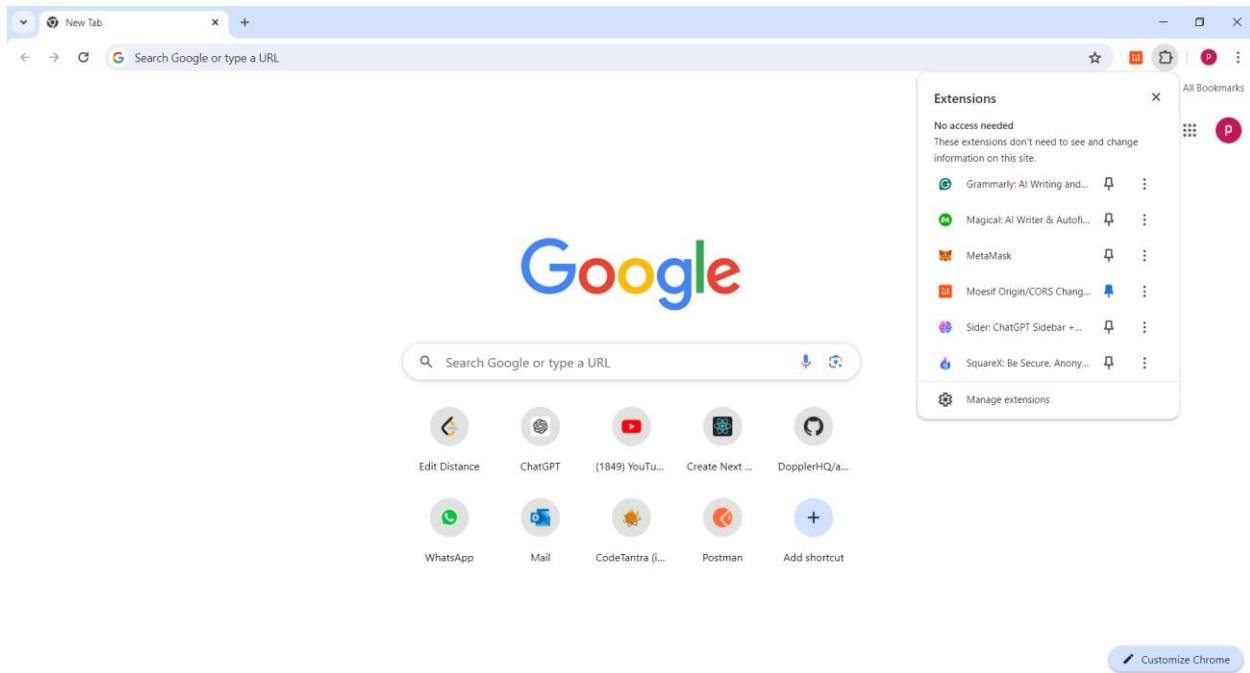


Metamask setup successfully.

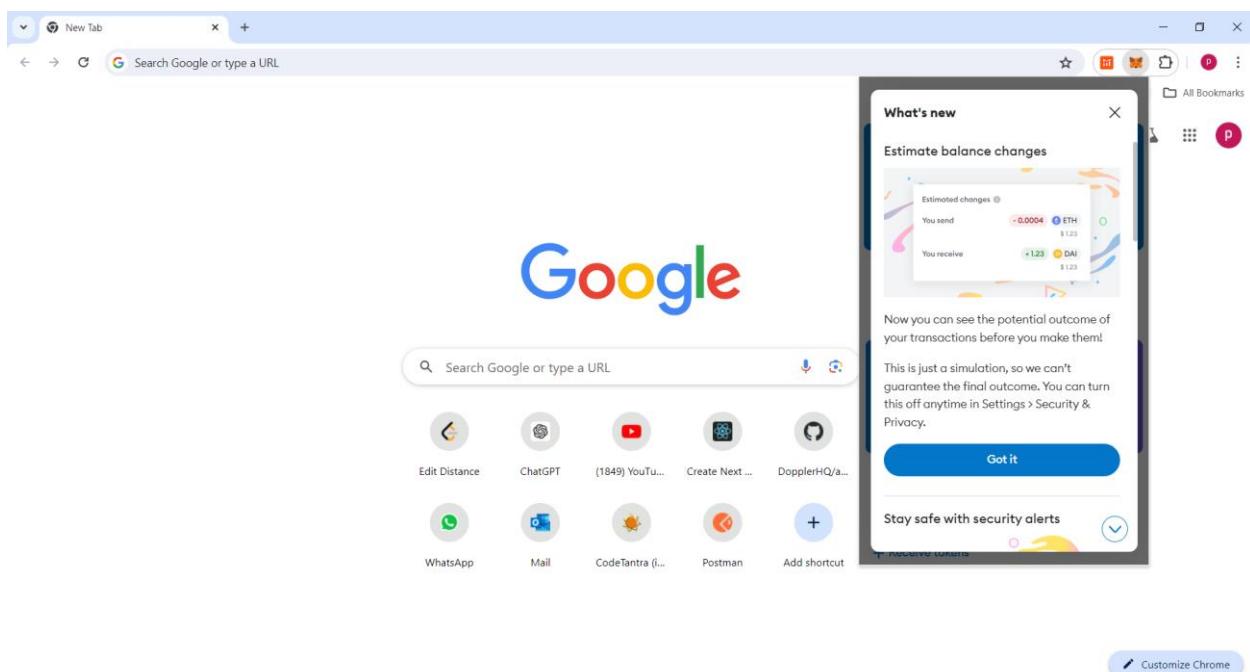


## 11.2 Setting up Server in Metamask

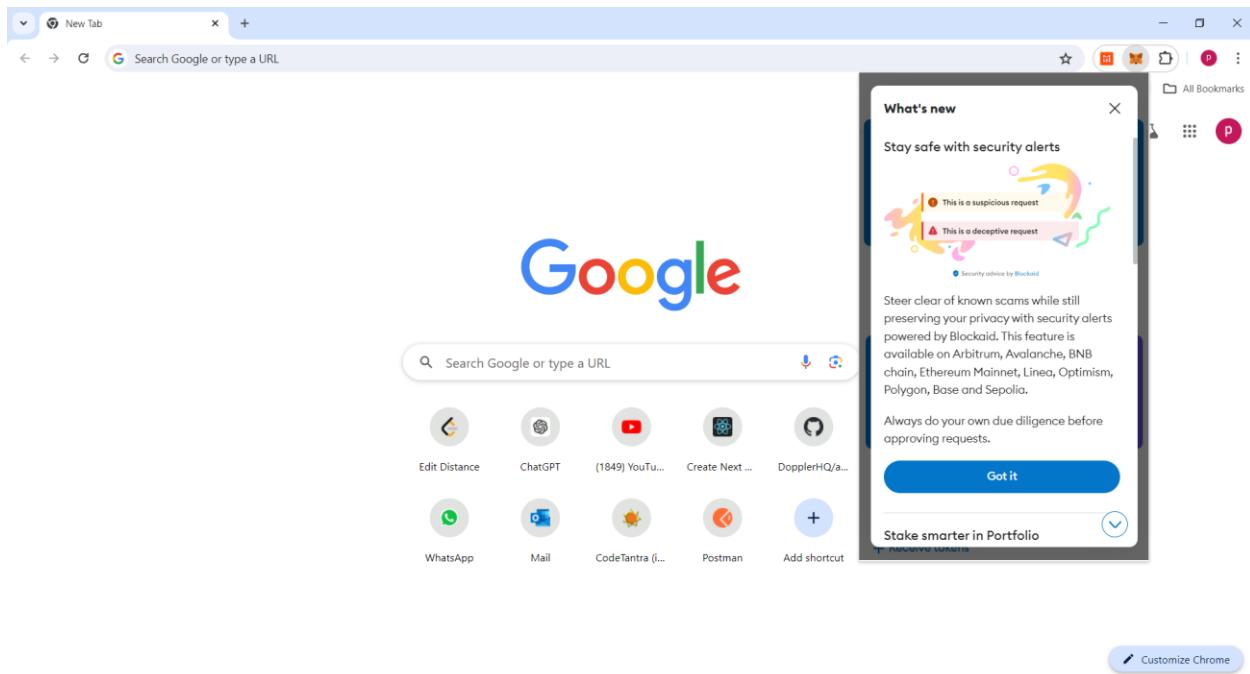
Open the browser and click on the 'extension'



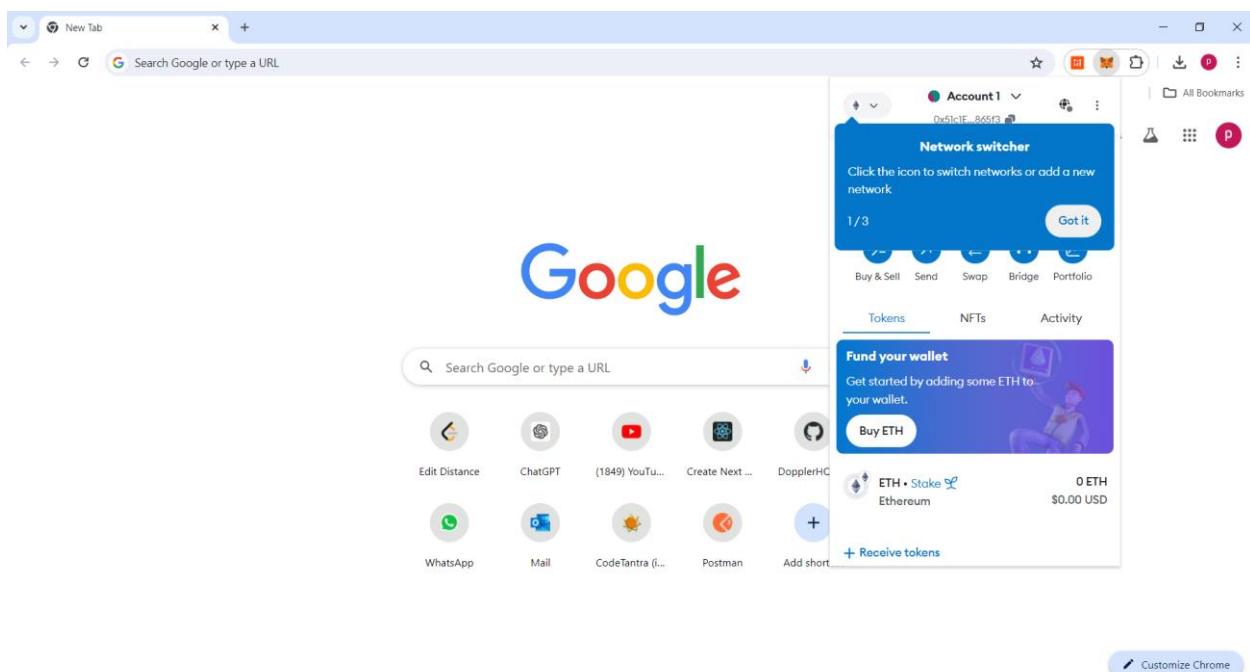
Click on the 'Metamask'



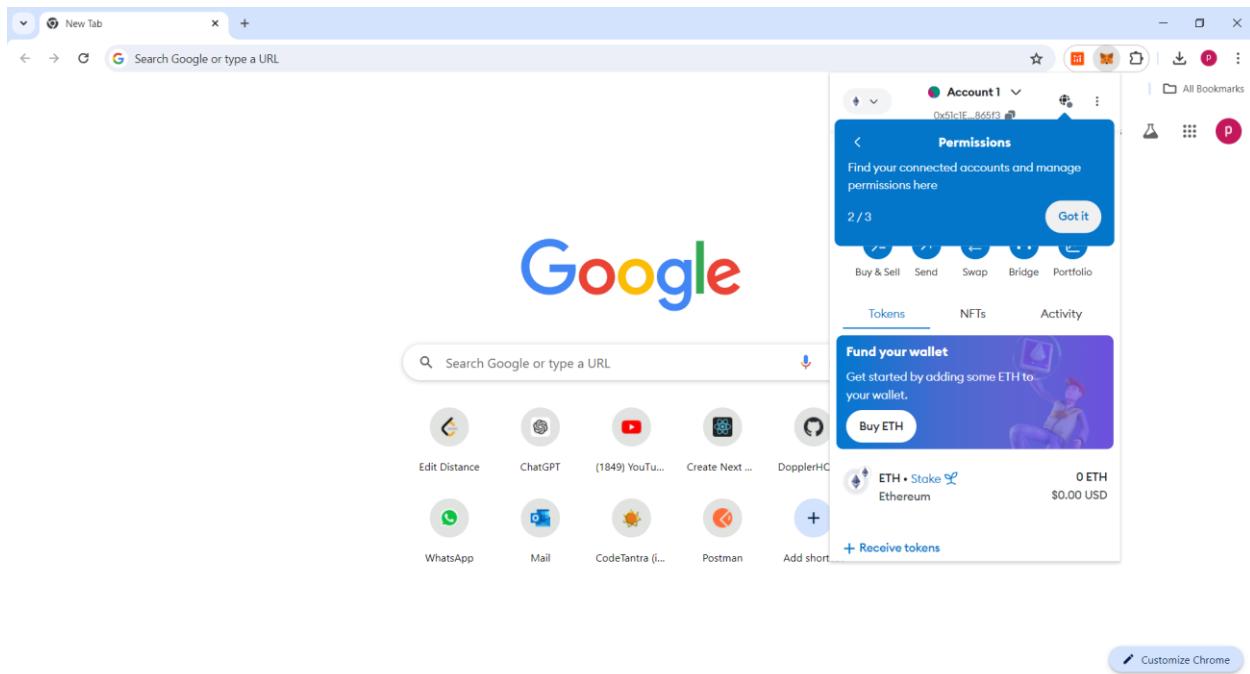
Click on 'x'



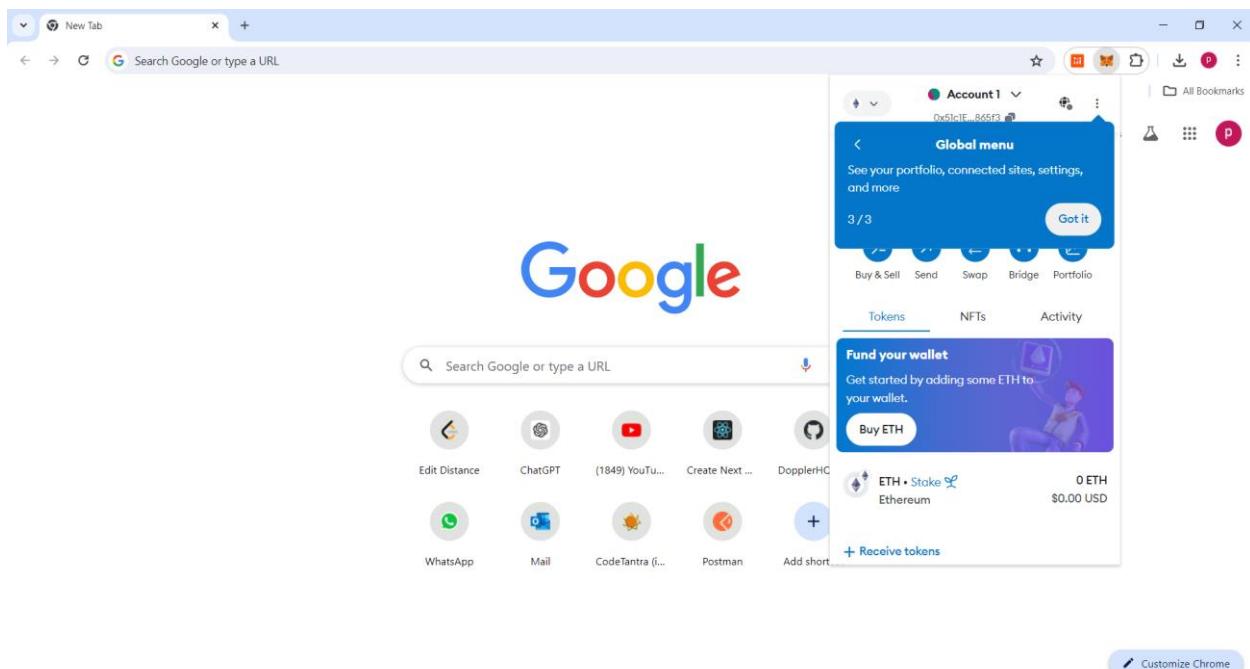
Click on 'Got it'

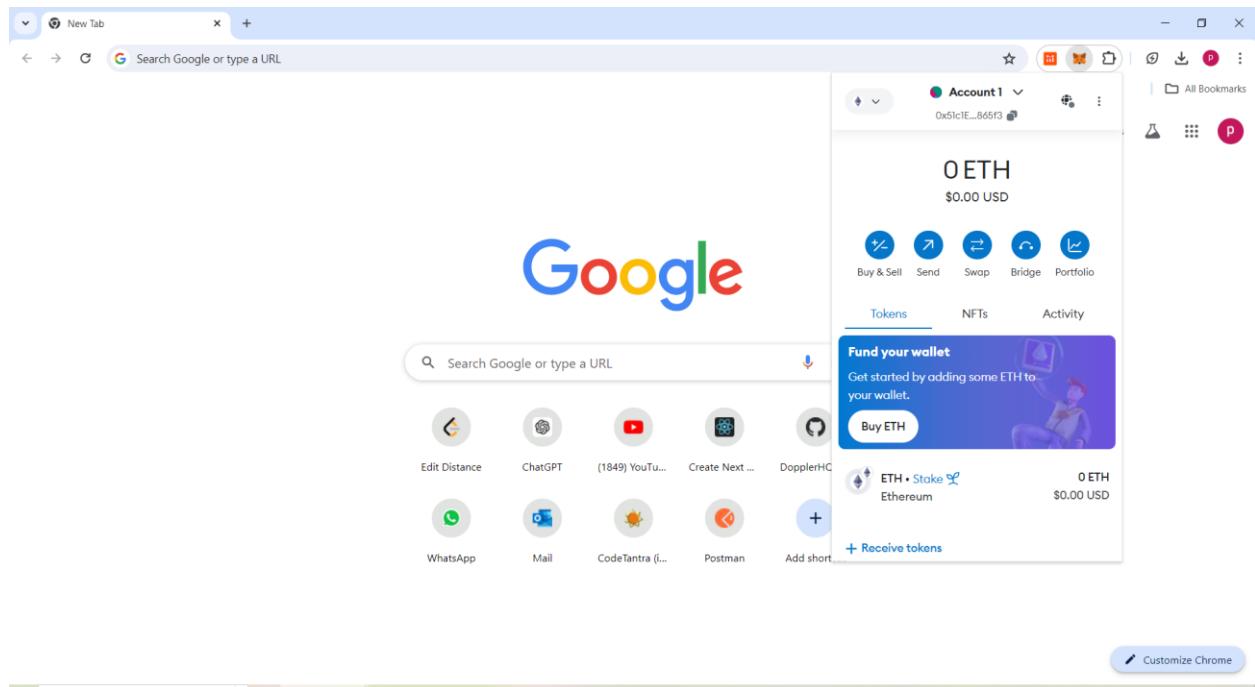


Again click on 'Got it'

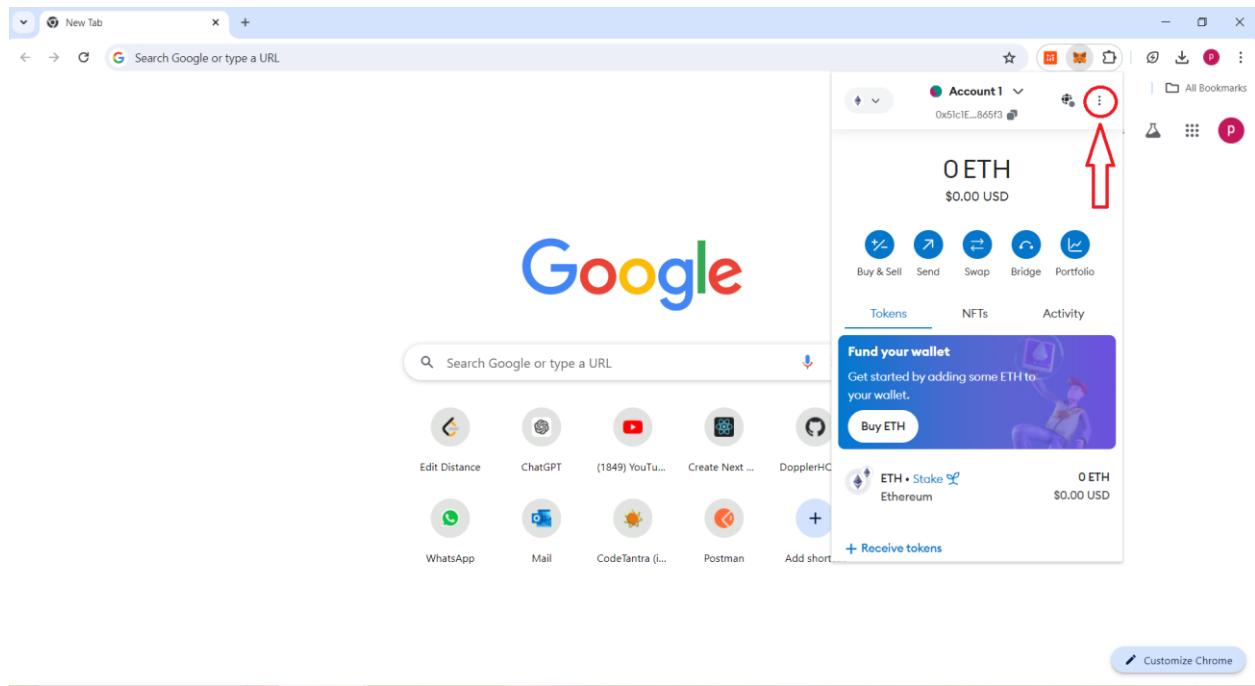


Again click on ‘Got it’

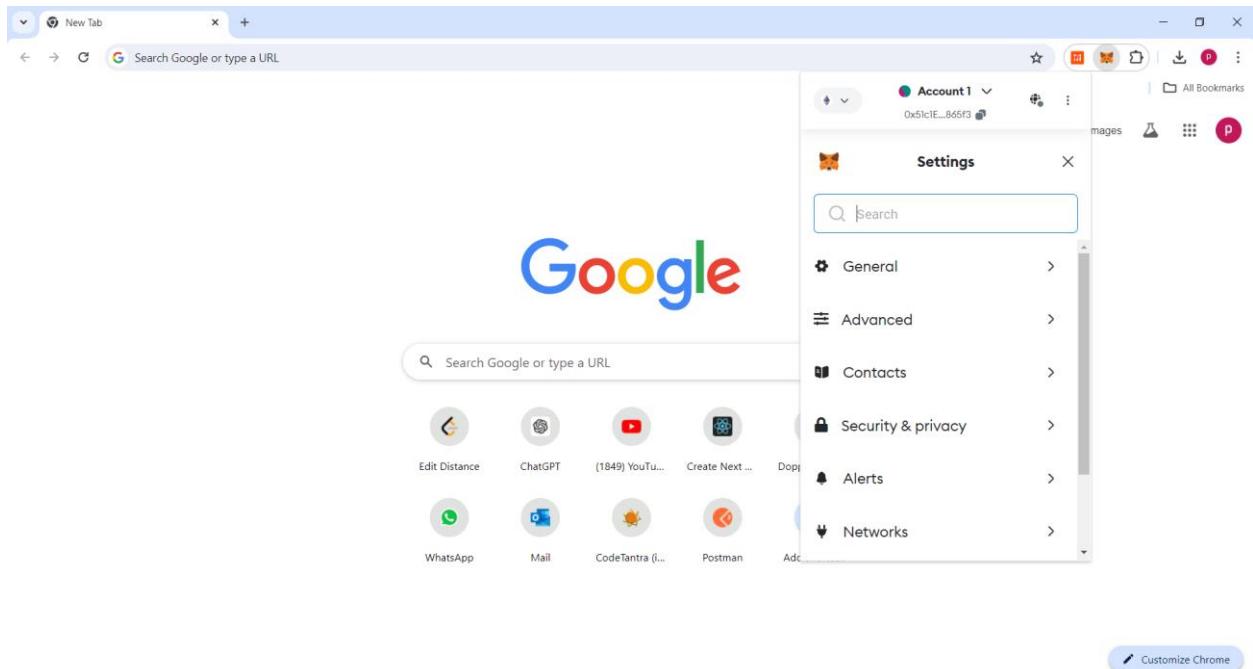




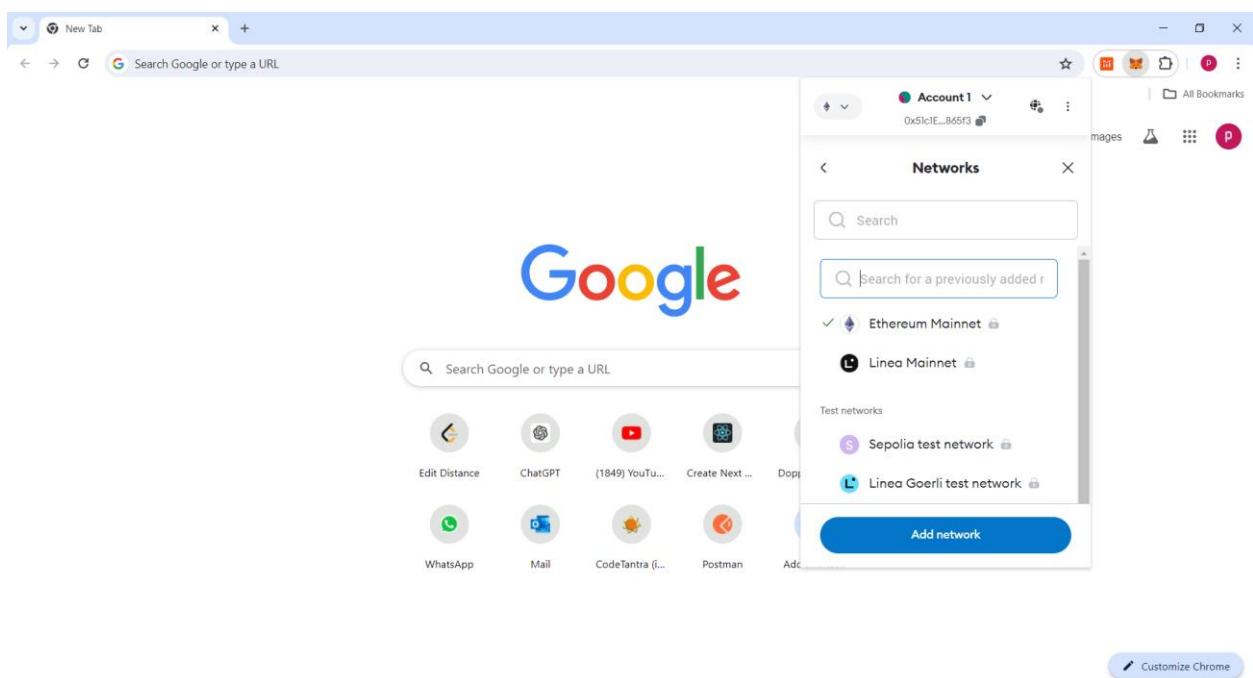
Click on 'three dot' that are marked in the figure



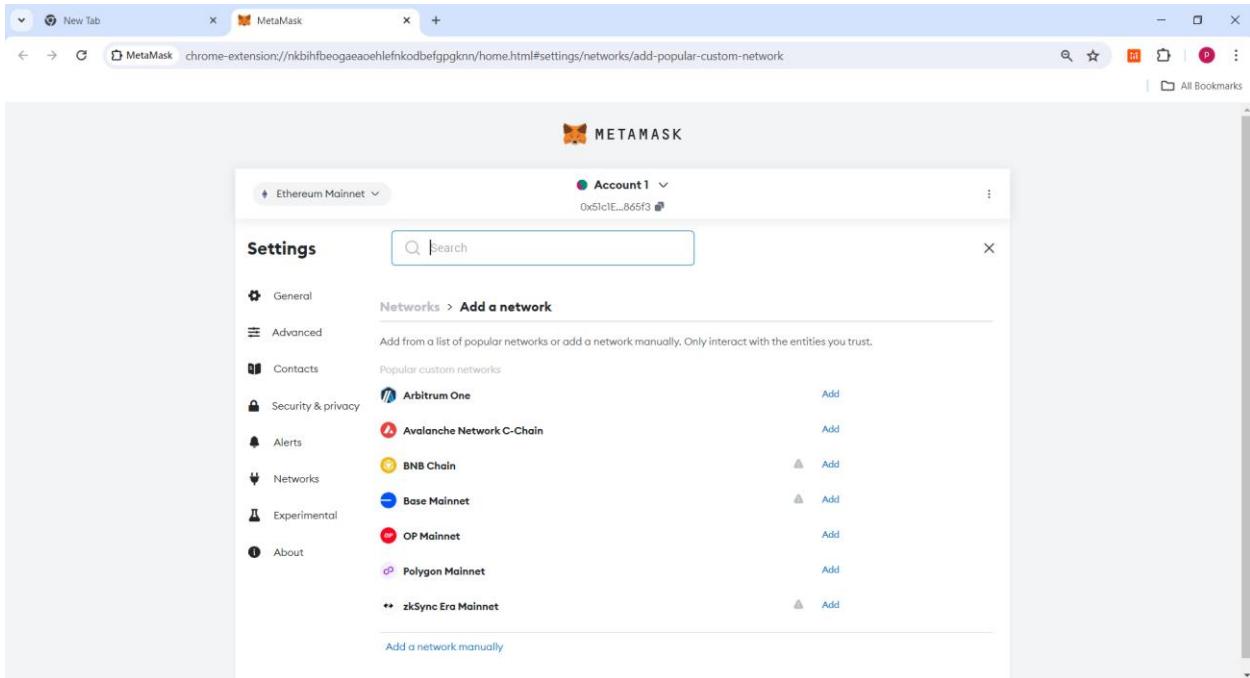
**Click on ‘Settings’**



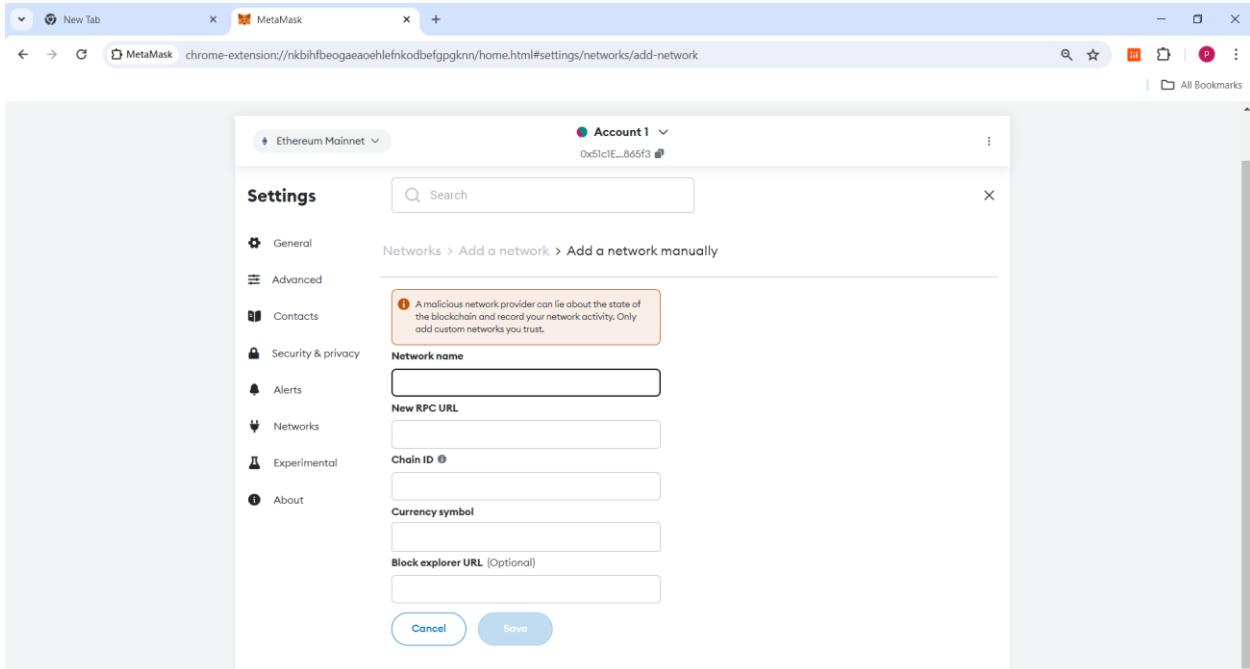
**Click on ‘Networks’**



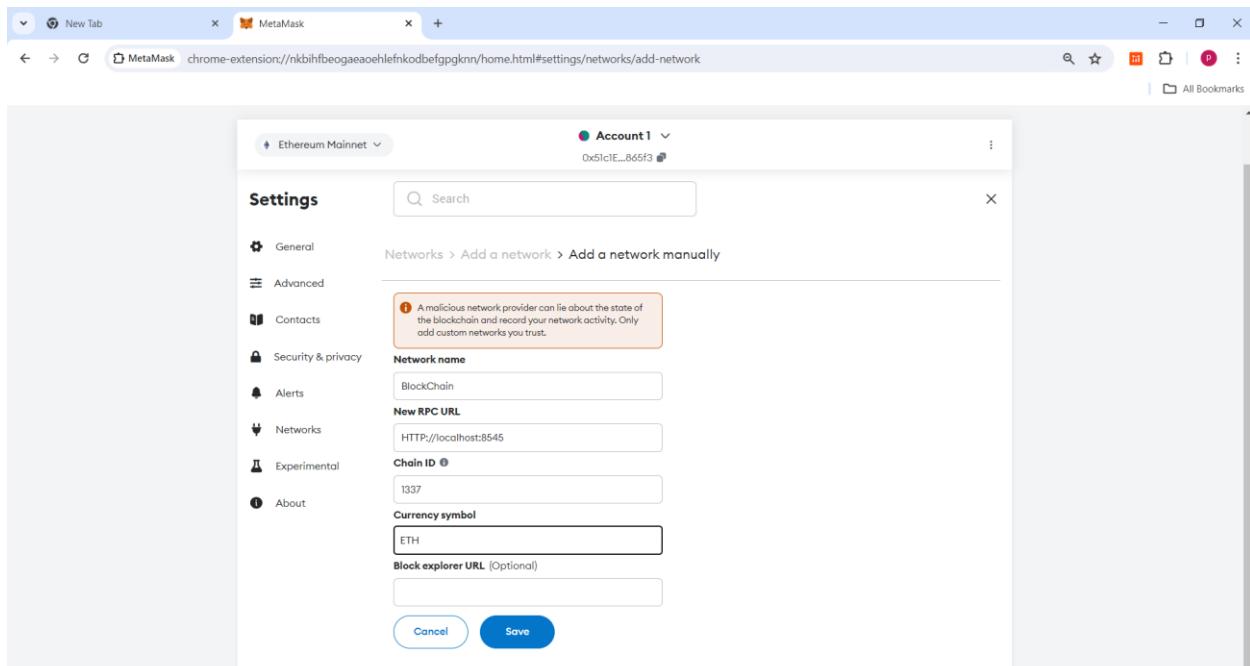
Click on 'Add network'



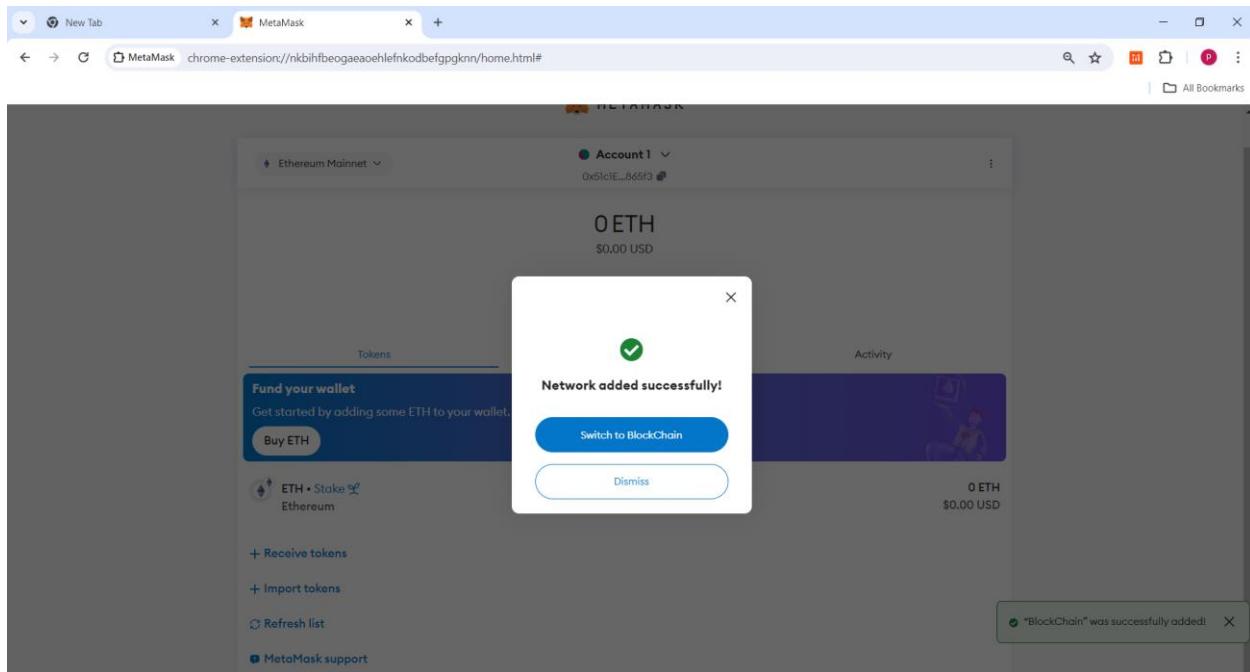
Click on 'Add a network manually'



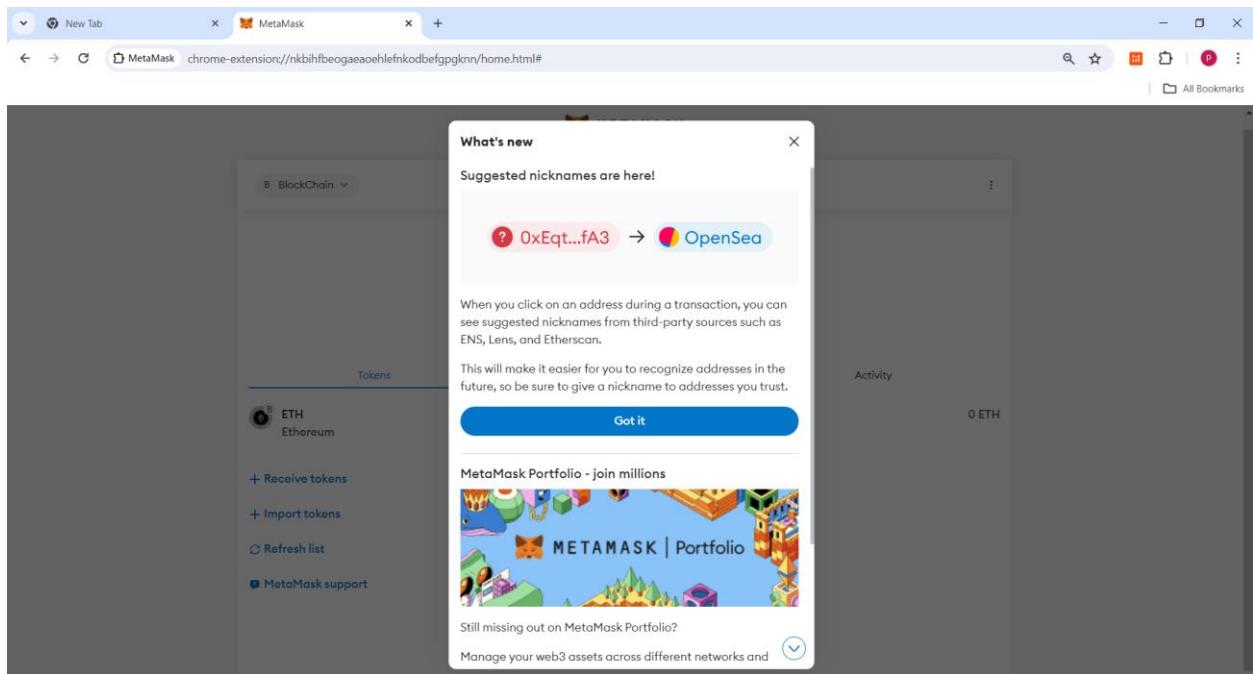
Fill all the details as shown below in the picture and before filling make sure that all the container are running.



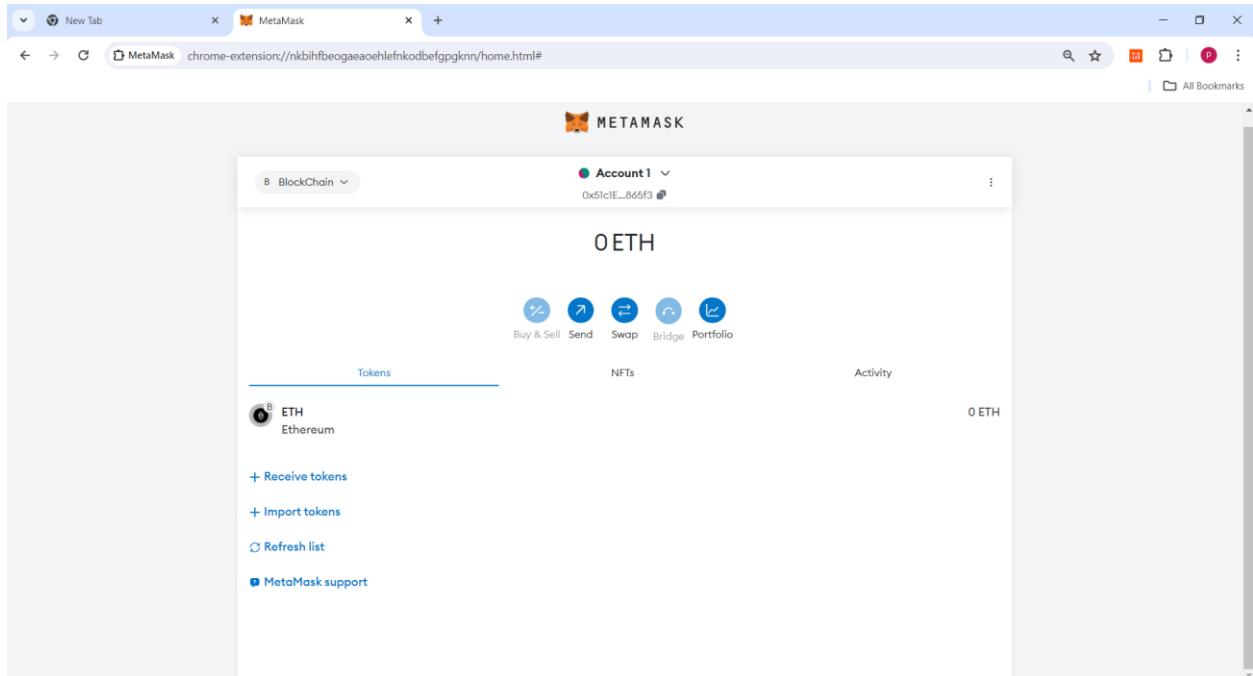
Click on 'Save'



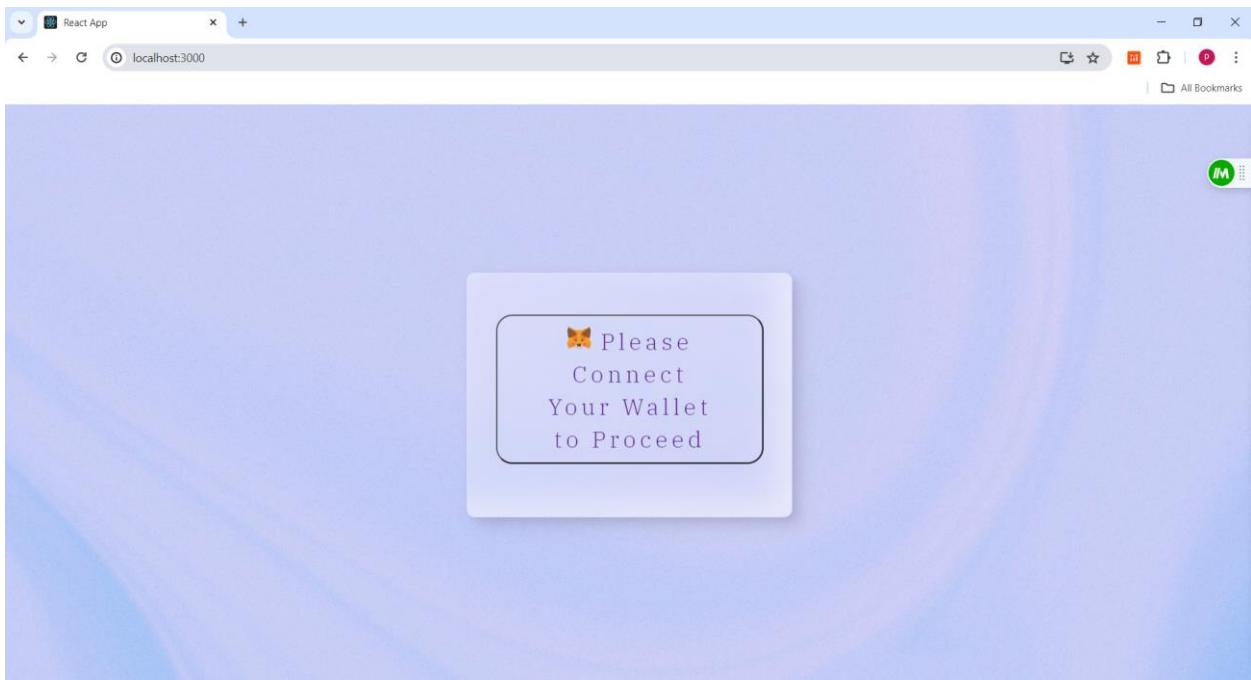
Click on 'Switch to BlockChain'



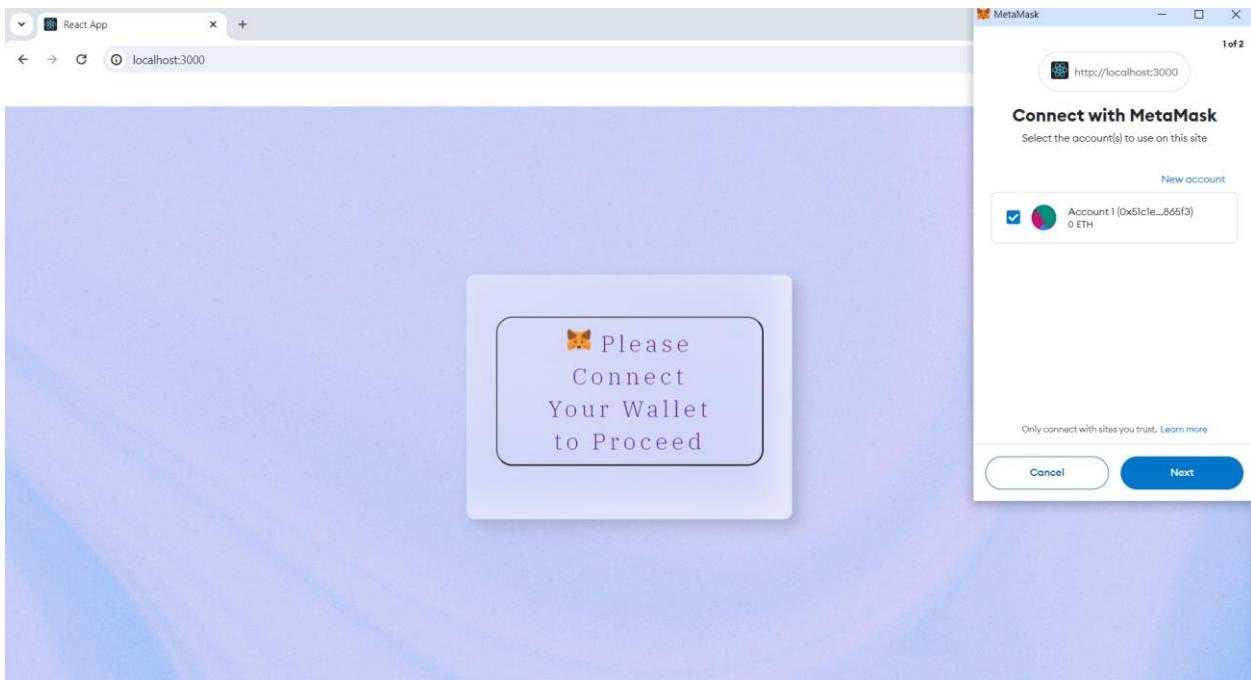
Click on 'x' and server setup completed.



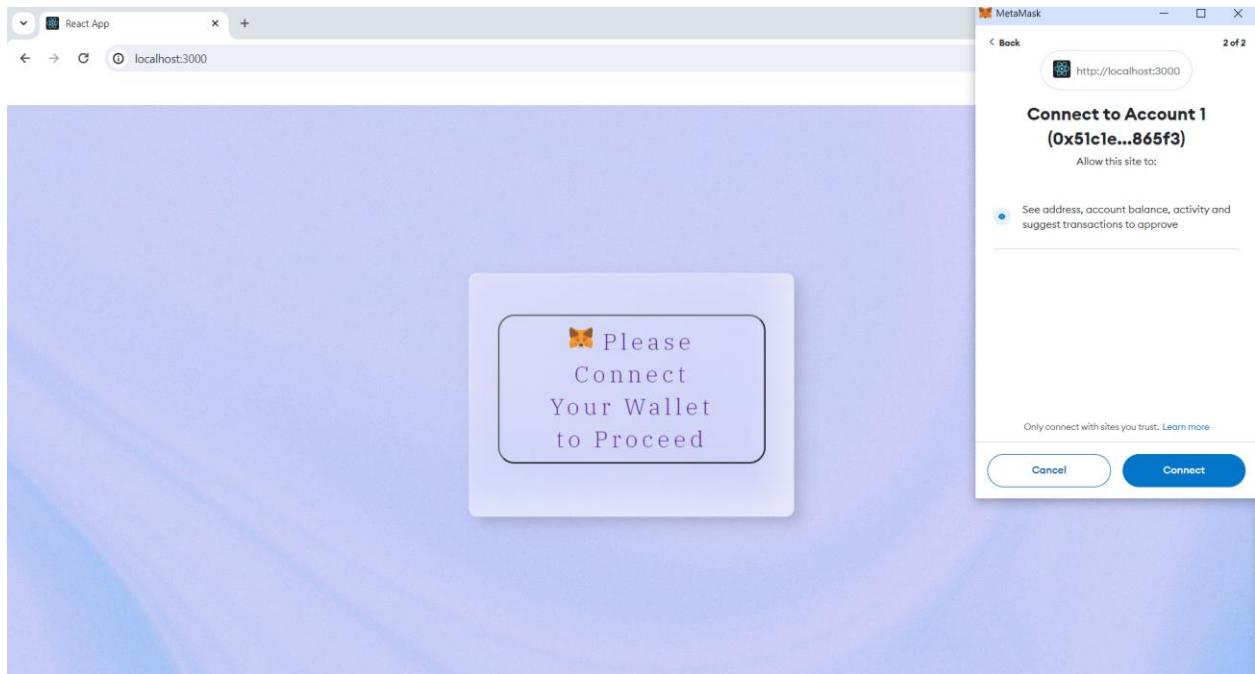
Now open the application URL '<http://localhost:3000/>'



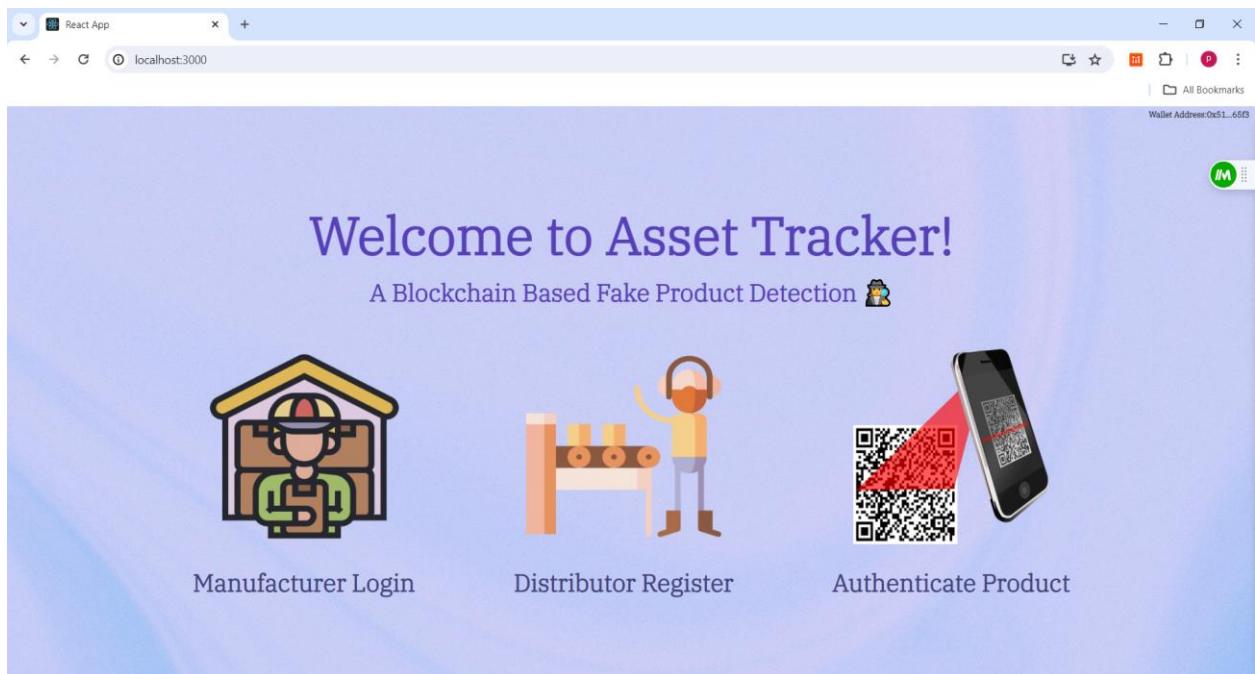
Click on 'Please Connect Your Wallet to Proceed'



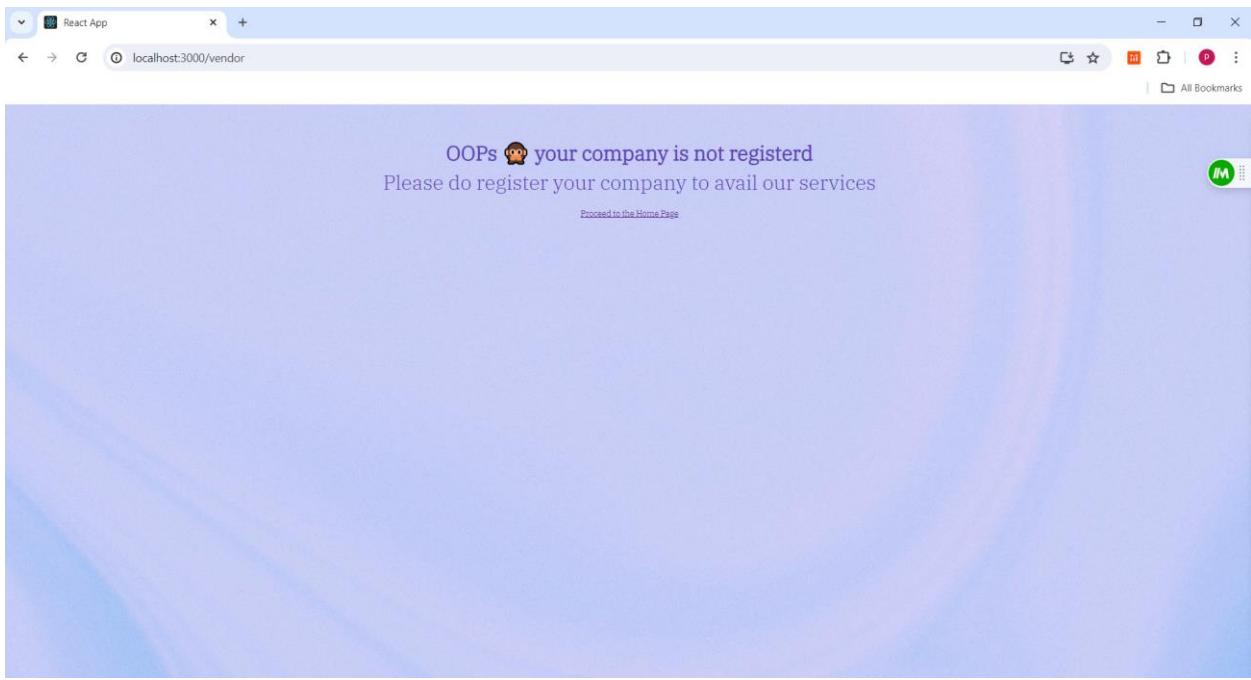
Click on 'Next'



Click on 'Connect'

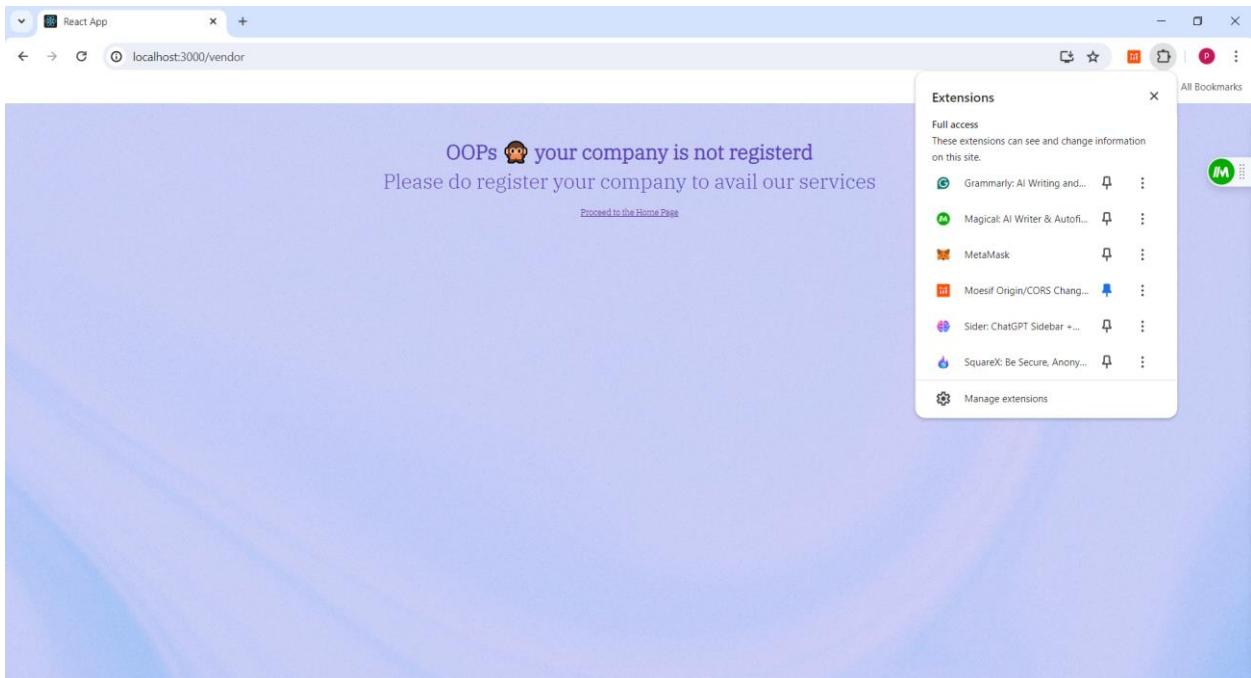


Click on 'Manufacturer Login'

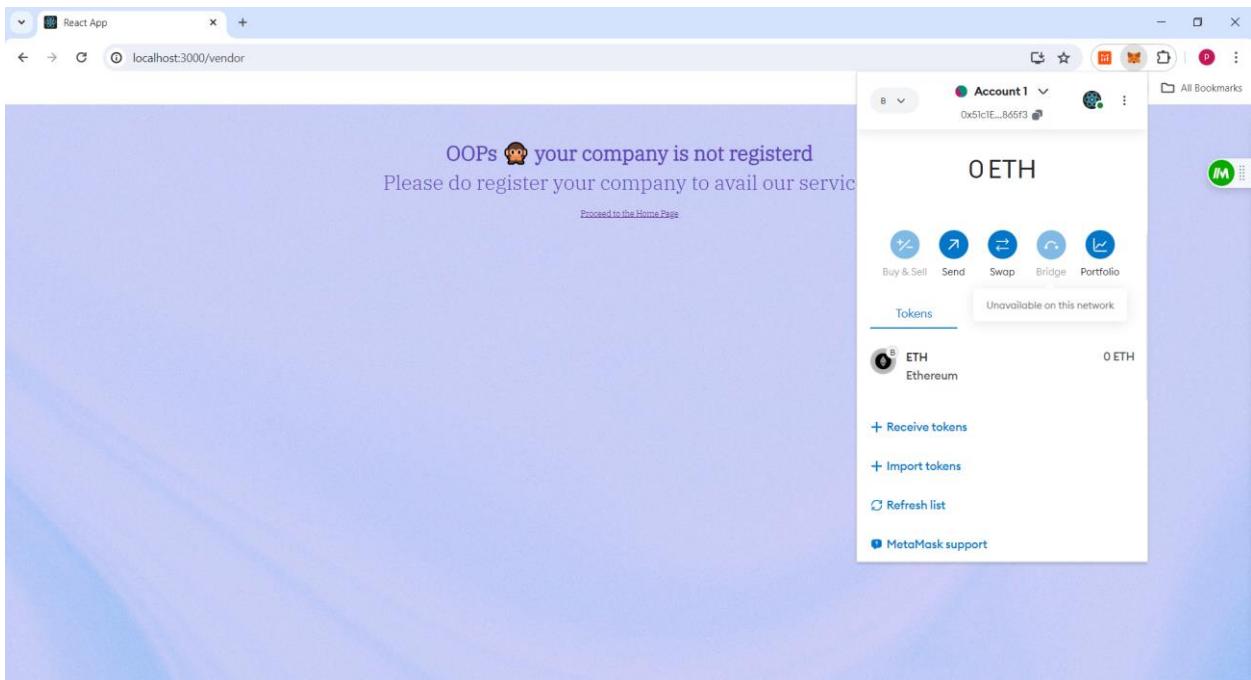


Since the given account is not registered, we will add the registered account of the manufacturer.

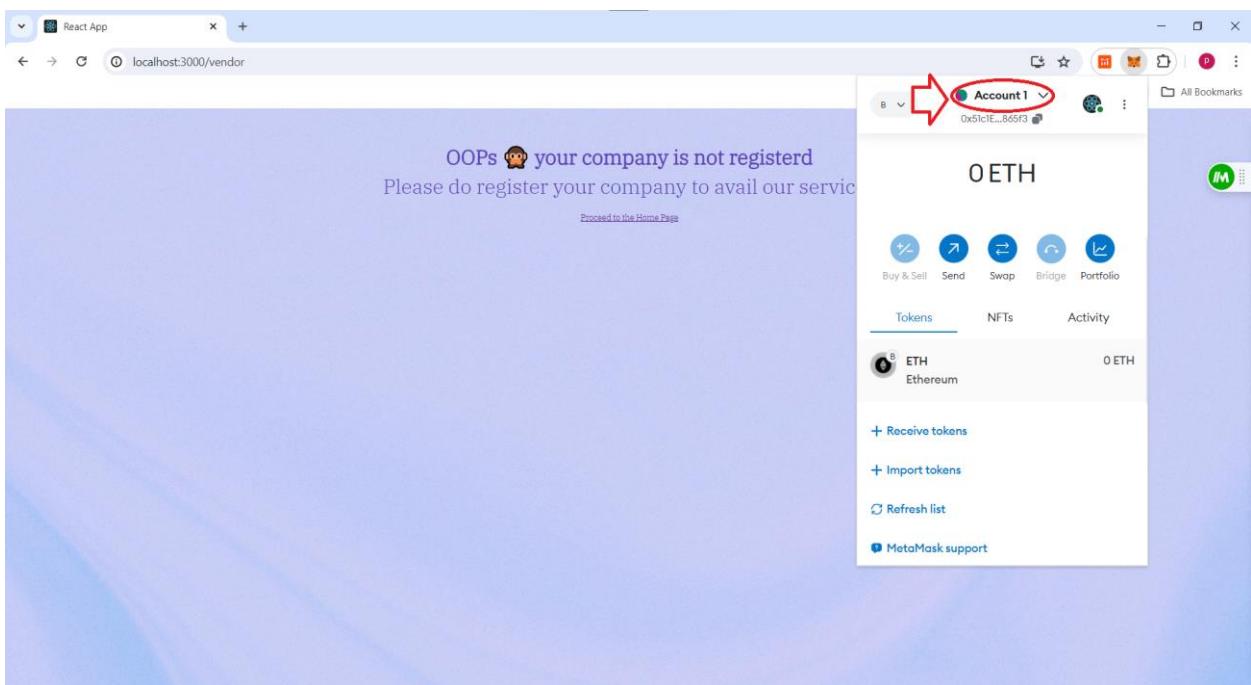
Open the browser and click on the 'extension'



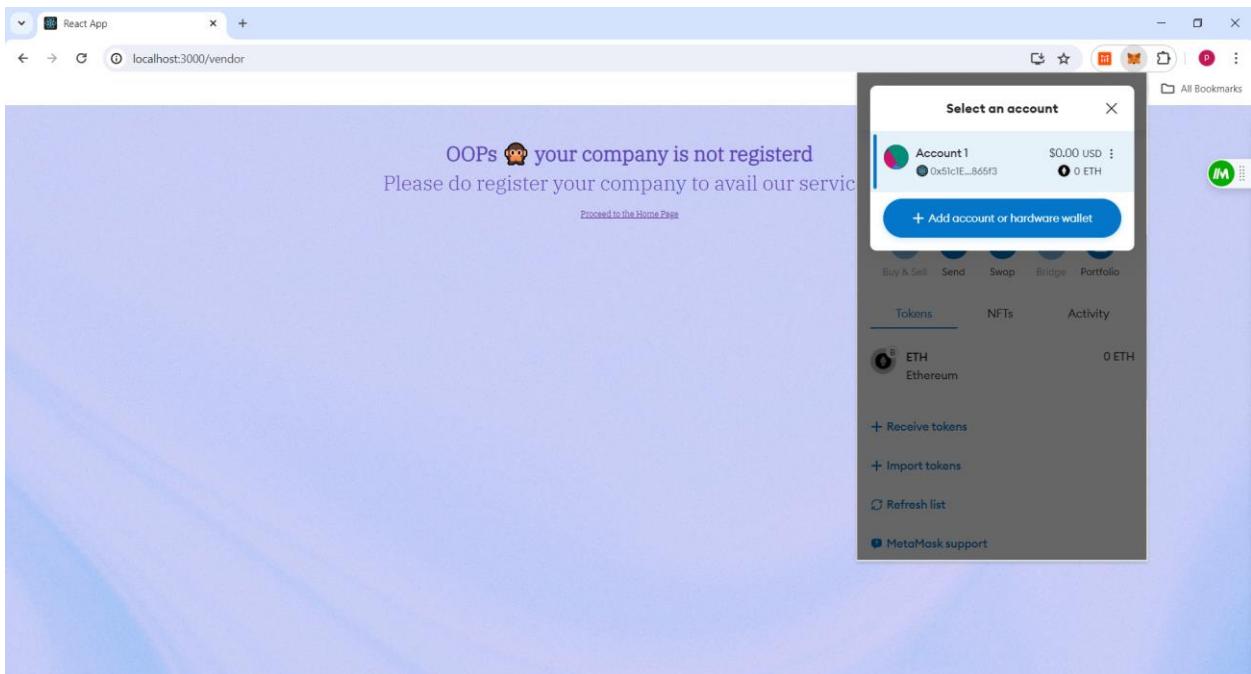
Click on the 'Metamask'



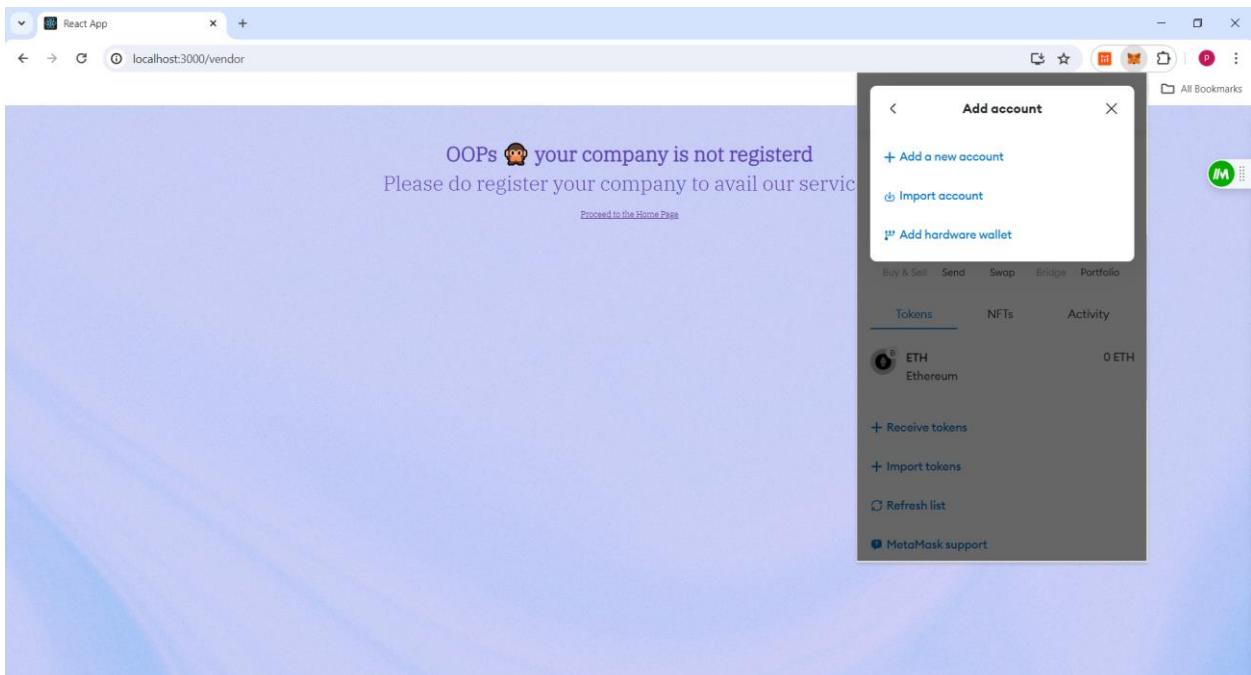
Click on red mark as shown below



Click on 'Add account'



Click on 'Import account'



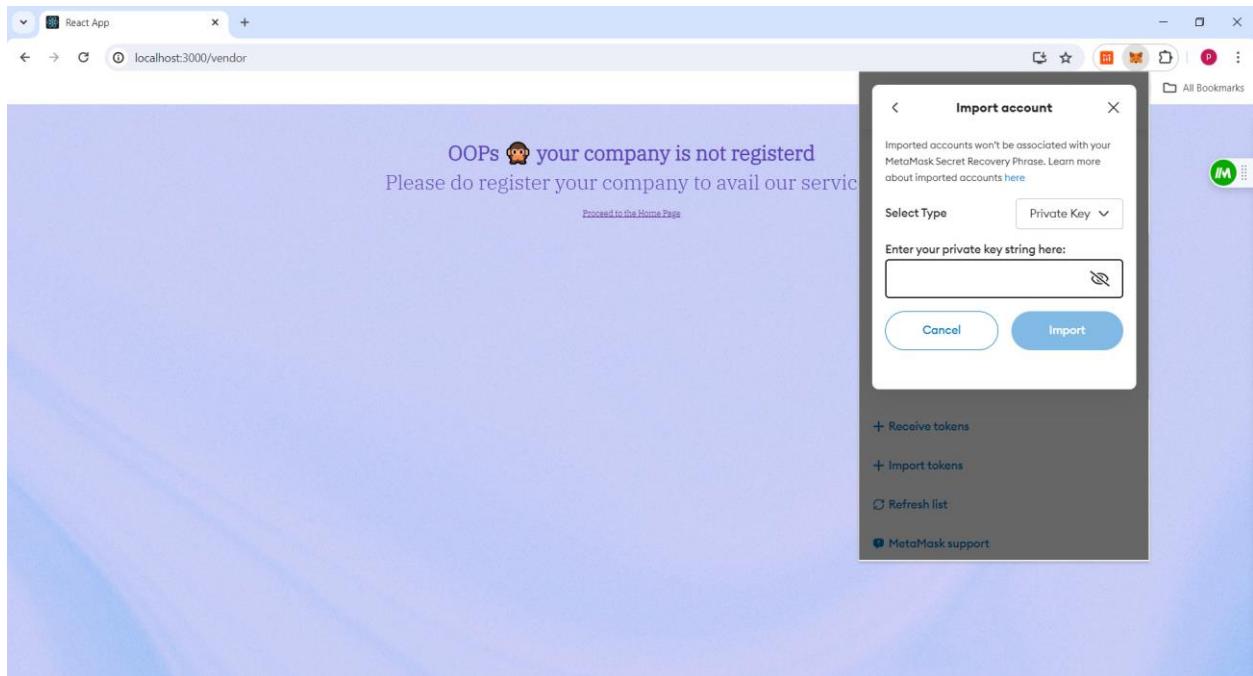
Copy the private key from the terminal where program is running

```

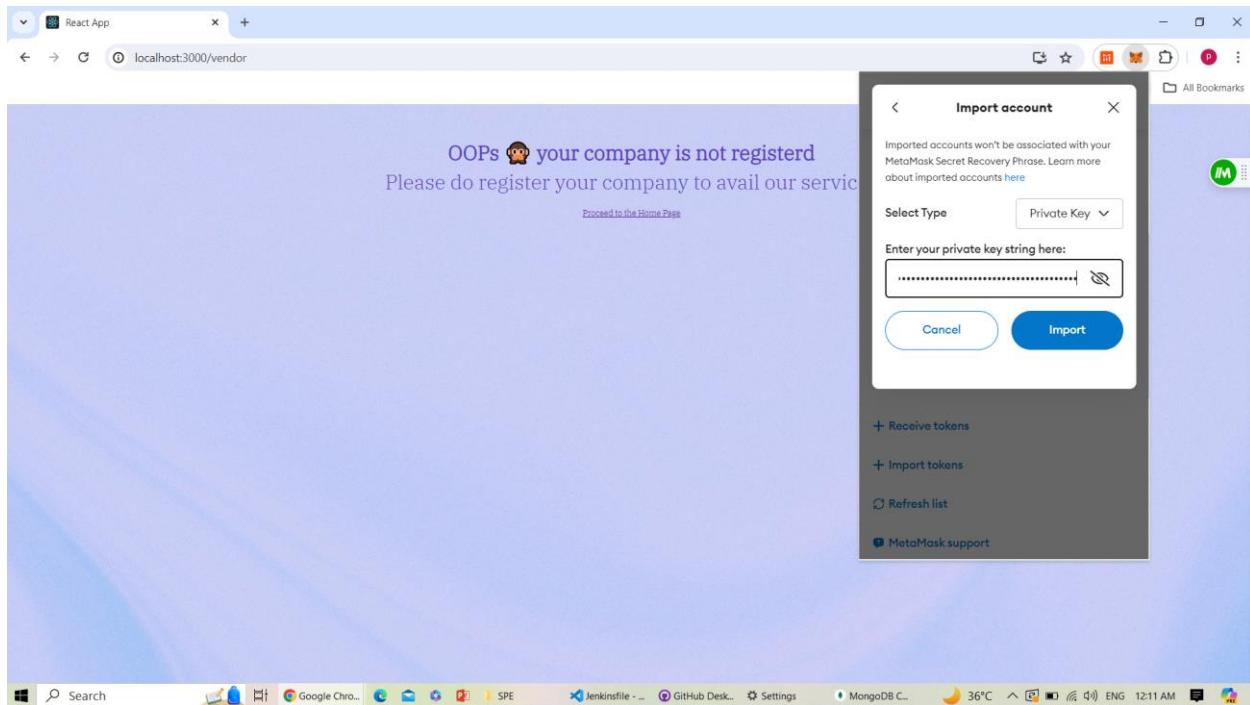
I eth-back The requested image's platform (linux/arm64/v8) does not match the detected host platform (linux/amd64/v3) and no specific platform was requested 0.0s
- Container spe-frontend-1
  Created 0.2s
Attaching to eth-back-1, Frontend-1, ganache-1
ganache-1 | Ganache CLI v6.12.2 (ganache-core: 2.13.2)
ganache-1 | (node:1) [DEP0040] DeprecationWarning: The `punycode` module is deprecated. Please use a userland alternative instead.
ganache-1 | (Use `node --trace-deprecation ...` to show where the warning was created)
ganache-1 | Available Accounts
ganache-1 | =====
ganache-1 | (0) 0xaafef437fb2ACa407905e98bb8bA24640338E685d (100 ETH)
ganache-1 | (1) 0x226ABC03f9c6c9bf24d36d629987e5cfbC592a1 (100 ETH)
ganache-1 | (2) 0x43465058969fc07de90dec0D95181C0108c53 (100 ETH)
ganache-1 | (3) 0x710fF715aef69f0F8c845f271a70a10e56f5e9a (100 ETH)
ganache-1 | (4) 0x3cb5e5e81D000abf79549A28ef31f6686Ac070d (100 ETH)
ganache-1 | (5) 0xte-e7B8-9eb0Aa324955e4AAA91C7EA466C63ea2 (100 ETH)
ganache-1 | (6) 0x8Aa147E70E3a176Eb543006e12f3c2534a3b5a (100 ETH)
ganache-1 | (7) 0xb5a027881fF7946a222624838AbAf2949551d (100 ETH)
ganache-1 | (8) 0x000000Cf3a309488B840cB7ec8971Cb529012 (100 ETH)
ganache-1 | (9) 0xb7788078589f1E1Df2e839A4ce811380Ef92a (100 ETH)
ganache-1 | Private Keys
ganache-1 | =====
ganache-1 | (0) 0xa3b831f10a928fedb3aa5b04a31f7952bf52be0910a574c15b7a7333b6d69d1
ganache-1 | (1) 0x010ca9a12278c64542497f5c705a50a3a5b20a20772000009484c664c
ganache-1 | (2) 0xada78fb2d75a602ba98d0c28bef636e69987d0e5a876d1e2dd3f889d49f3285
ganache-1 | (3) 0x58419463e50261e78c89a8eac337d7c78896c5d6954d76c9d265c7502ad038e
ganache-1 | (4) 0x4a859cc0ee2b23b2d50053bb0e13db6bc30f95eb547d188e945c23201848
ganache-1 | (5) 0xe5161cd3c309e7676c7d655c88b4a3de05f2d9d1a58edc9a87065bd709c78
ganache-1 | (6) 0xc73e0a2c025d3215b6ceb58826b2ad0bc82251d7c2b76062d953d2d89e176
ganache-1 | (7) 0x4626earff5235eaf432e0e0f196d84d9b4555fe1b2006329fb365a12709bbe
ganache-1 | (8) 0x0f7c5d47a03013ae3da10bfa52a80c3898e207452089f9aa42c1ee0213
ganache-1 | (9) 0x6e7939a462aa39ff145236aa3a066e0560f9e0cff2daf112724f39769adceb6
ganache-1 | HD Wallet
ganache-1 | =====
ganache-1 | Mnemonic: steel canvas artefact air cover girl layer orange upon pumpkin diary soap
ganache-1 | Base HD Path: m/44'/60'/0'/0/{account_index}
ganache-1 | Gas Price

```

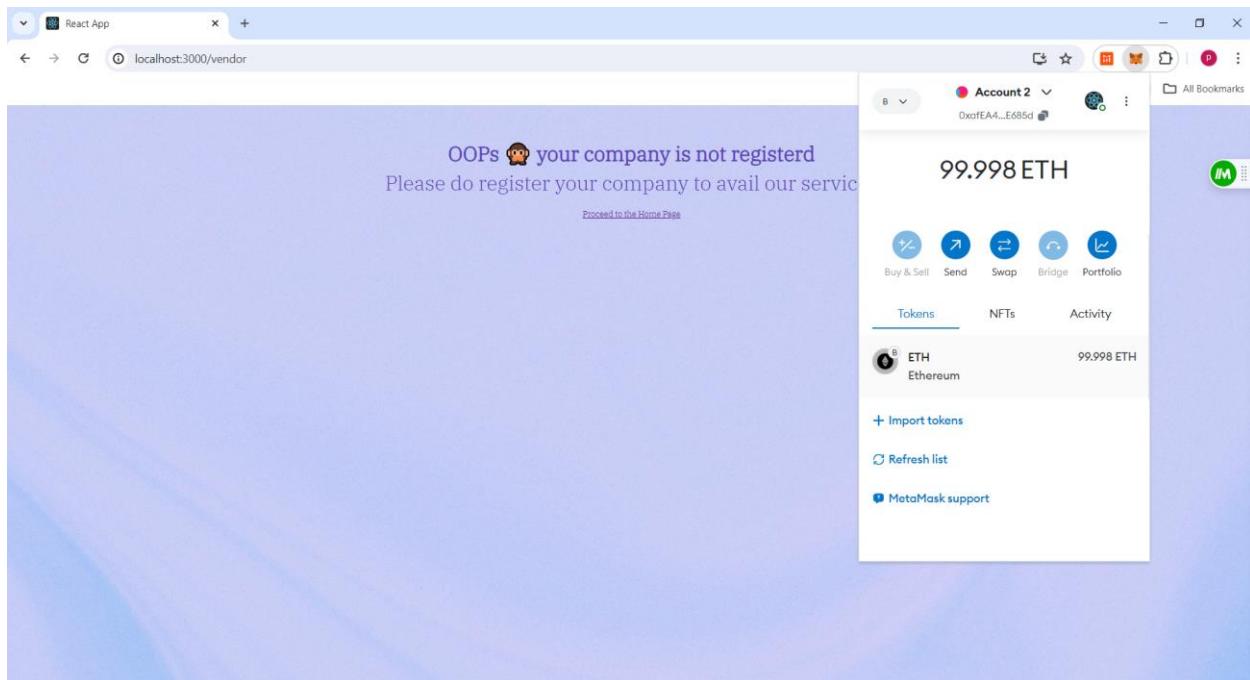
Ln 24, Col 14 Spaces: 4 UTF-8 CRLF Groovy



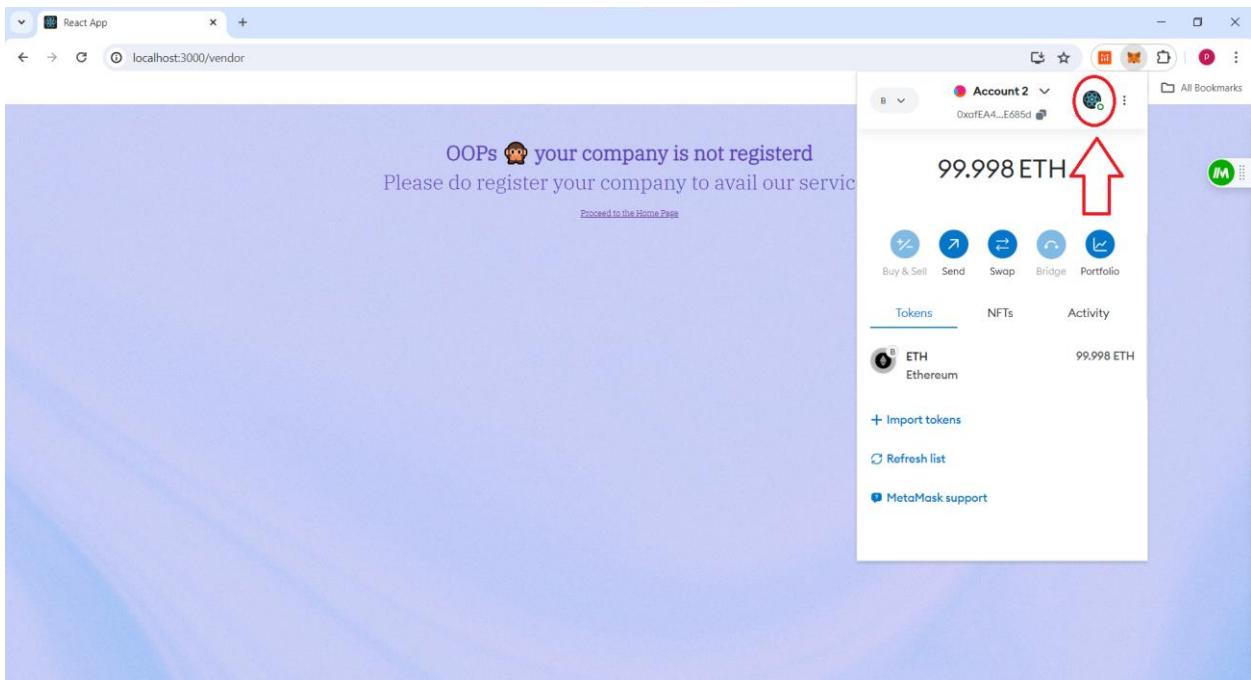
Paste that copy private key



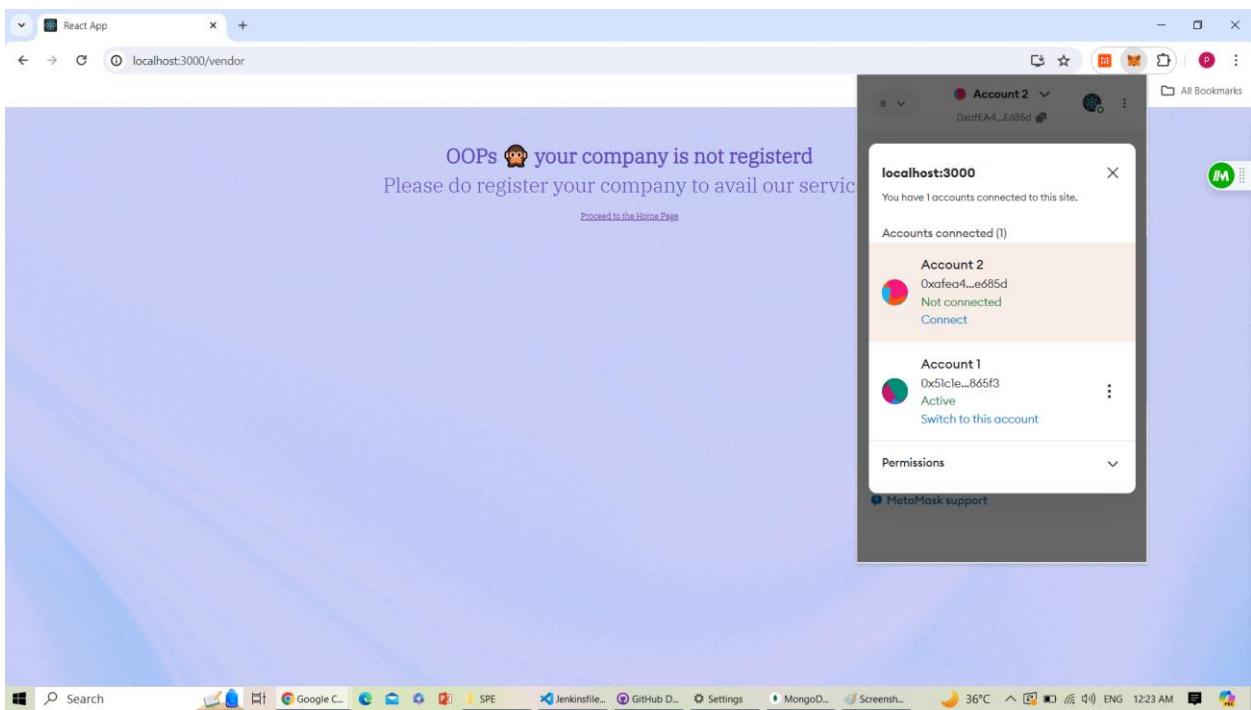
Click on 'Import'



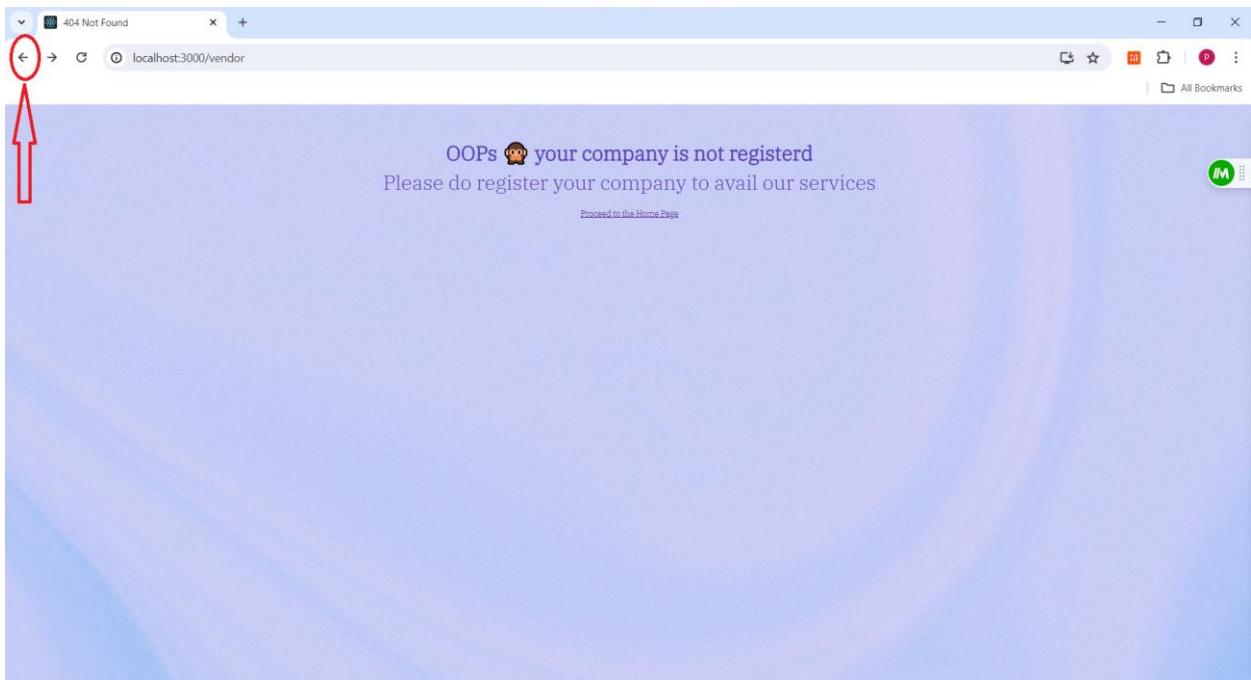
Click on red mark as shown below



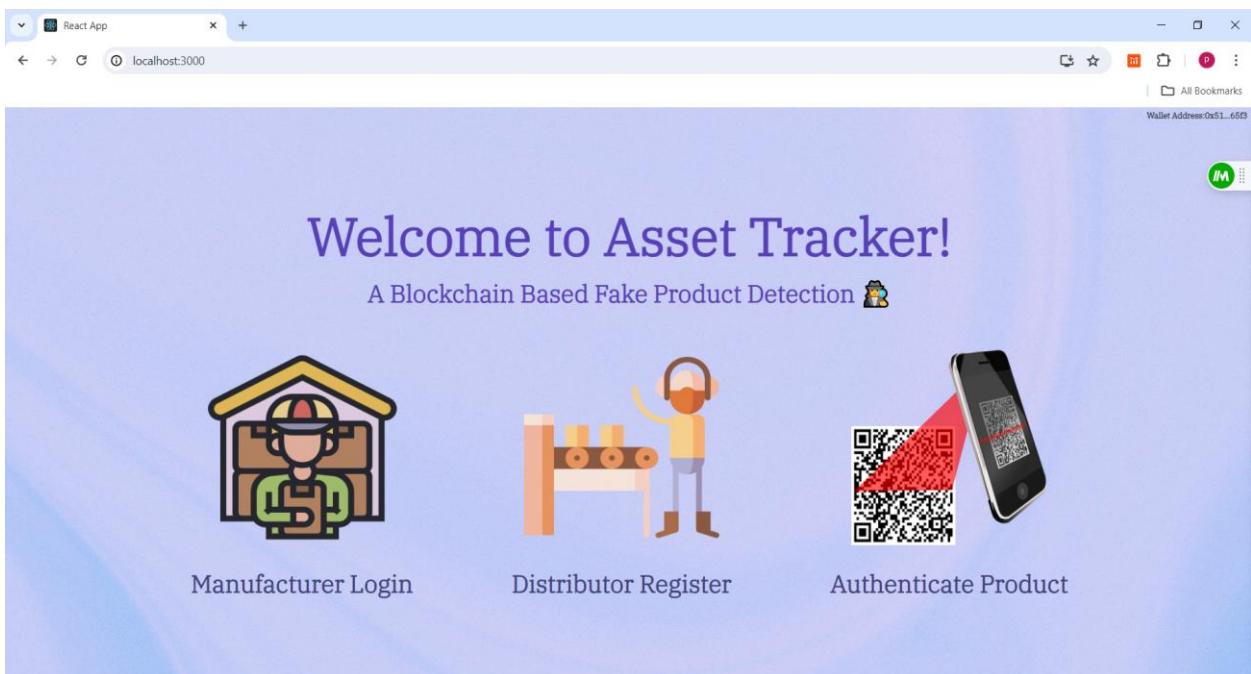
Click on 'Connect'



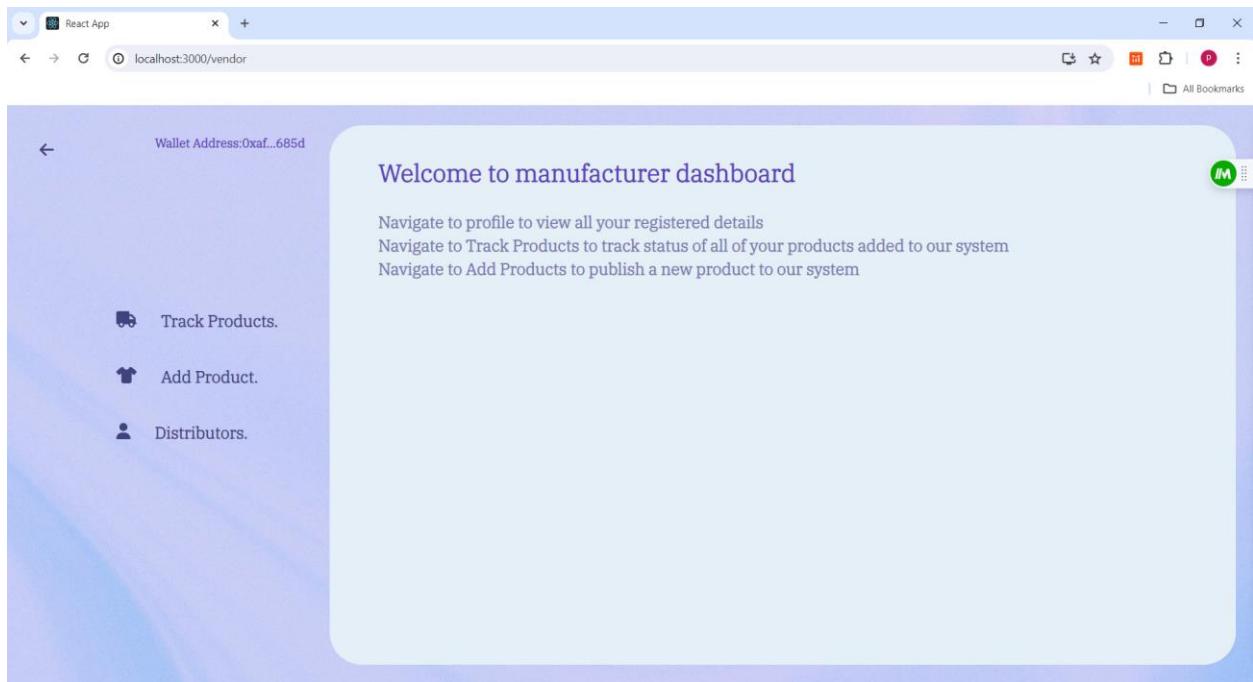
Click on 'Go back' of the browser.



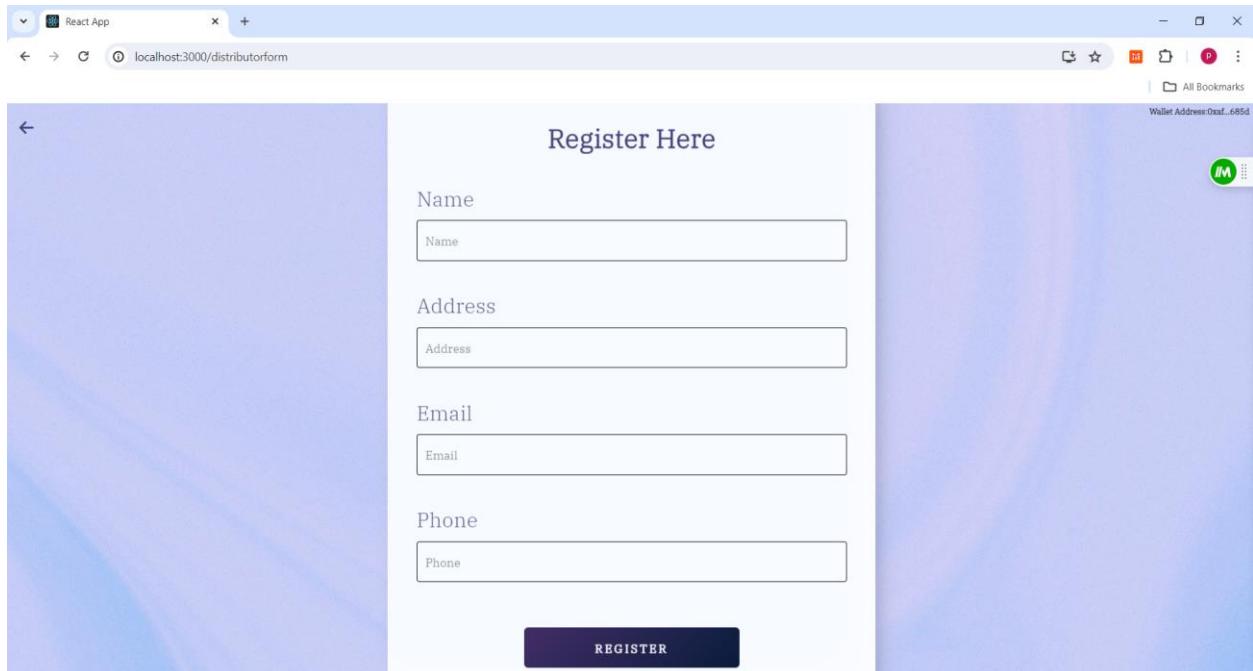
Press F5 key and then click on 'Manufacturer Login'



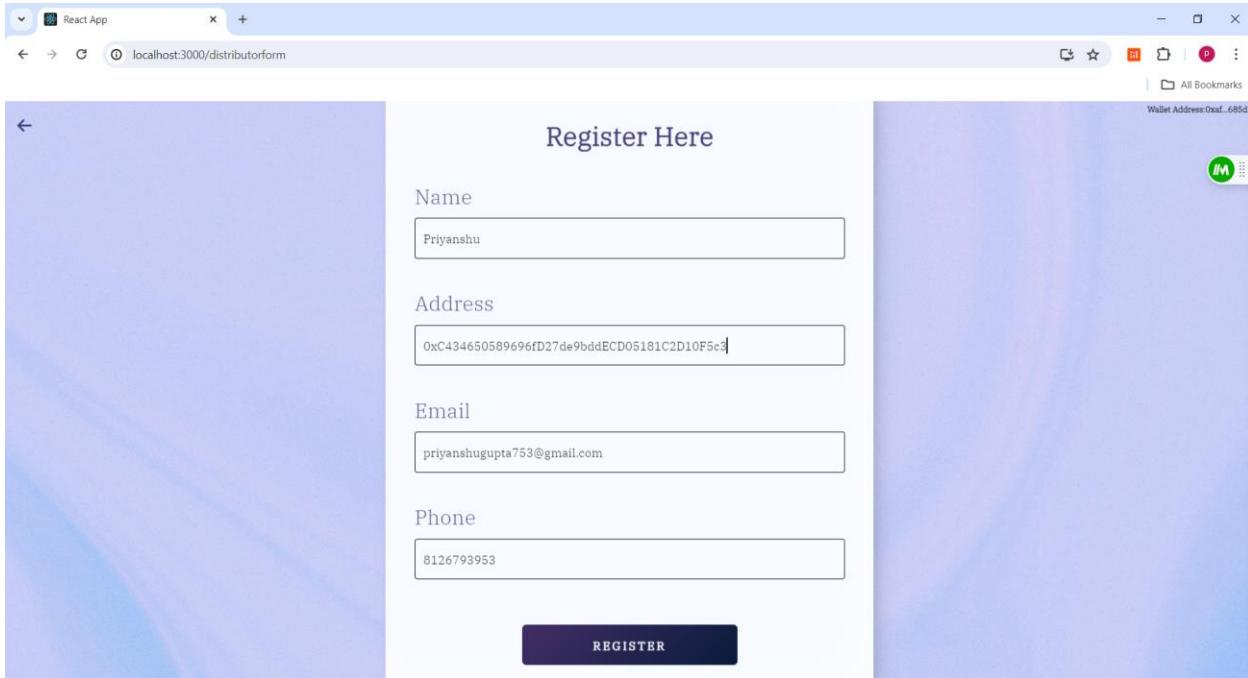
Dashboard of the registered manufacturer



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.

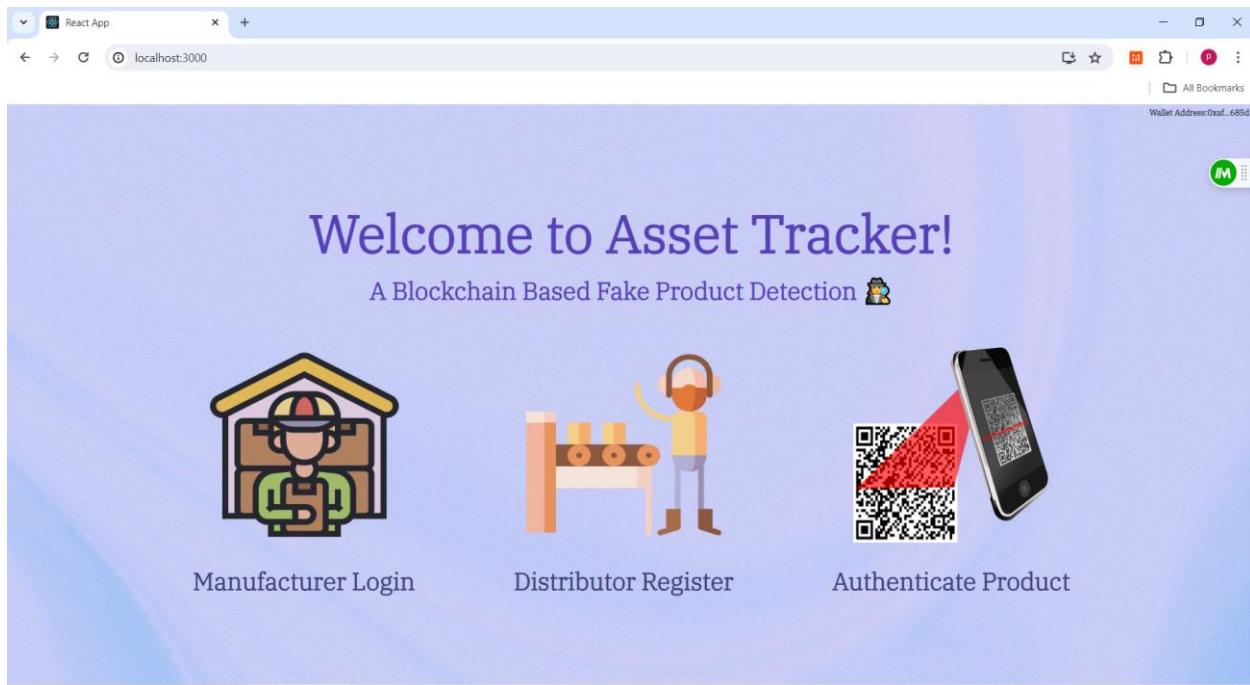
**Click on 'REGISTER'**

The screenshot shows a browser window for a React application at `localhost:3000/distributorform`. The page displays a registration form titled "Register Here" with fields for Name, Address, Email, and Phone. The "Name" field contains "Priyanshu", the "Address" field contains "0xC434650589696fD27de9bddECD05181C2D10F5c3", the "Email" field contains "priyanshugupta753@gmail.com", and the "Phone" field contains "8126793953". Below the form is a "REGISTERING..." button. To the right of the browser is the MetaMask extension interface, showing a transaction for "0x8f5c0...7444E : INSERT DISTRIBUTOR". It displays the estimated gas fee as 0.003523 ETH and the total amount as 0.003523 ETH. At the bottom of the screen is a Windows taskbar with various icons.

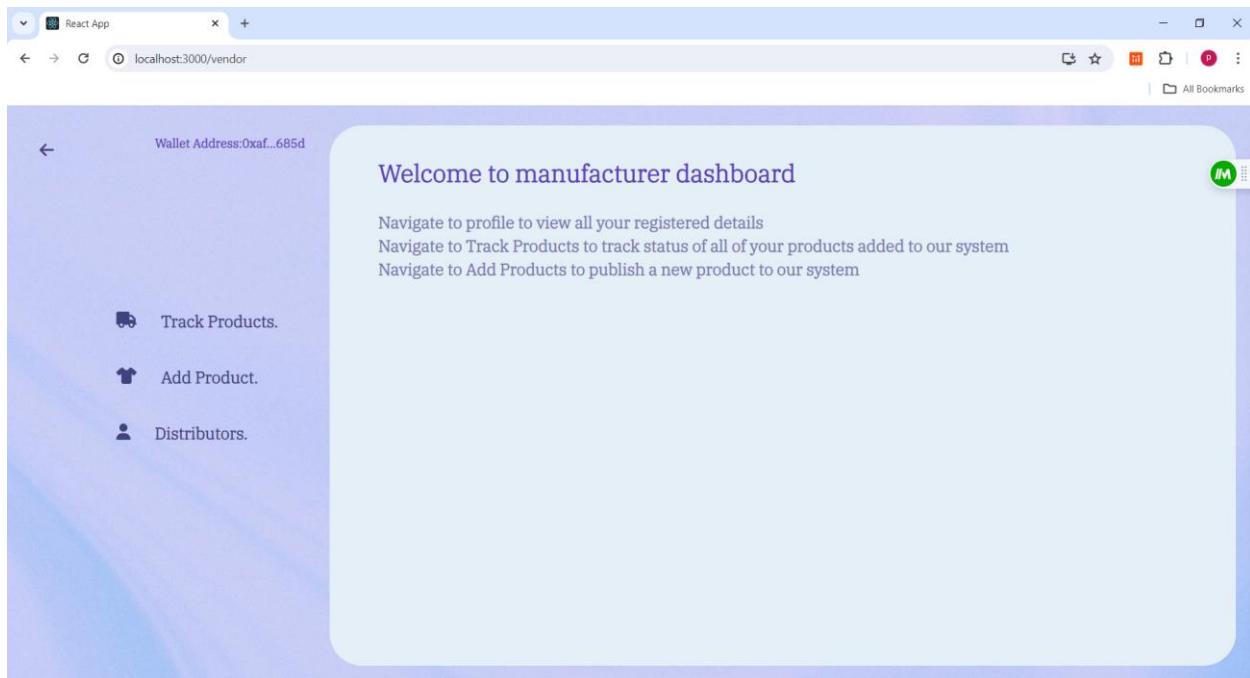
Click on 'Confirm'. The distributor registered successfully.

The screenshot shows the same browser window after the registration process. A modal dialog box is displayed with the message "Your have successfully Registered..! You will get a mail if vendor assigns you a dispatch order". Below the message is a link "Proceed to the Home Page". The rest of the registration form and the MetaMask interface are visible in the background. The Windows taskbar is also present at the bottom.

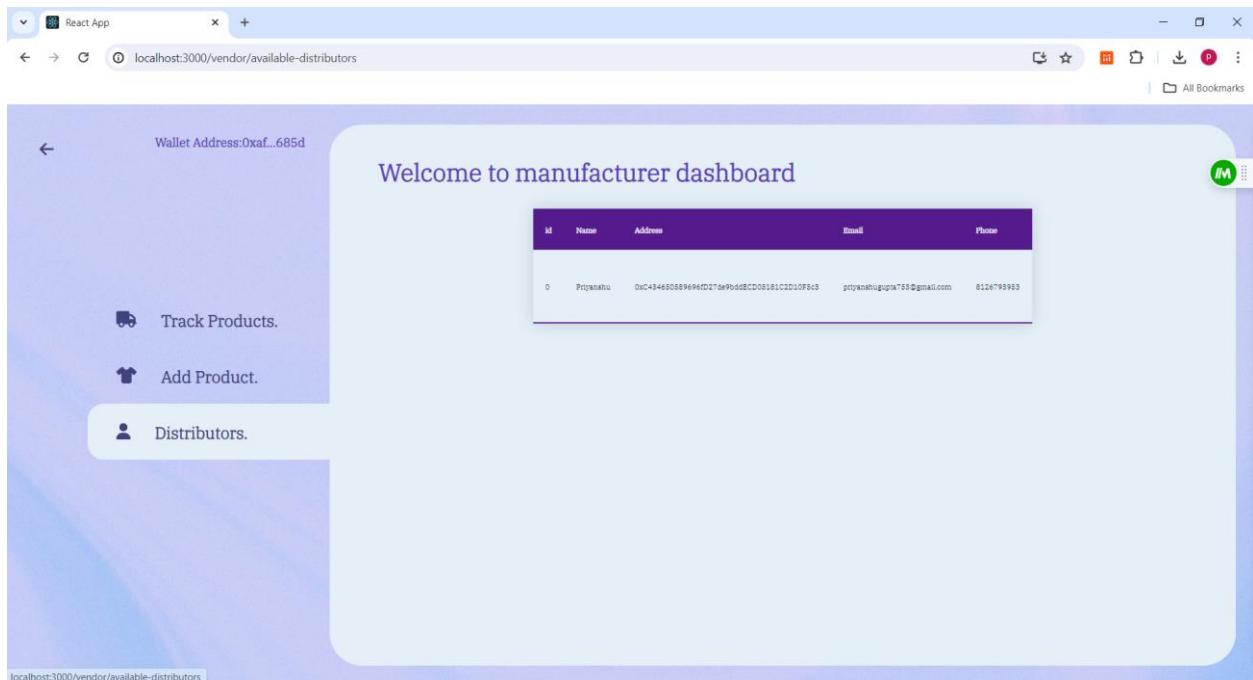
Click on 'Proceed to the Home Page'



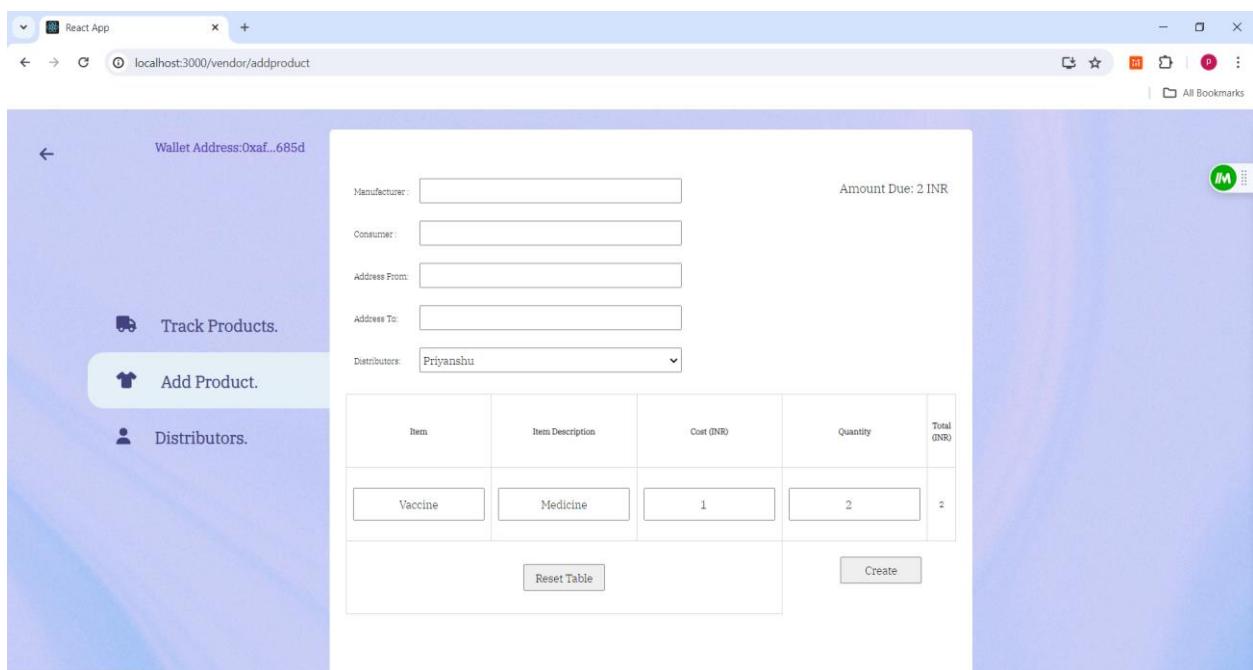
Click on 'Manufacturer Login'



Click on the 'Distributor'. To see the added distributor.



Click on 'Add Product'. To add the product.



Fill the details as shown below:

React App

localhost:3000/vendor/addproduct

Wallet Address:0xaf...685d

**Track Products.**

**Add Product.**

**Distributors.**

Manufacturer: Samsung Amount Due: 24999 INR

Consumer: Amit

Address From: Noida, Uttar Pradesh

Address To: Delhi

Distributors: Priyanshu

Item	Item Description	Cost (INR)	Quantity	Total (INR)
Samsung S6 lite tablet	Immerse yourself into a c	1	24999	24999

Reset Table Create

The MetaMask interface shows a transaction titled "0x8f5c0...7444E : CREATE ASSET". It displays the estimated gas fee (0.0107322 ETH), total amount (0.01073222 ETH), and max amount (0.01073222 ETH). Buttons for "Reject" and "Confirm" are visible at the bottom.

Click on 'Create'

React App

localhost:3000/vendor/addproduct

Wallet Address:0xaf...685d

**Track Products.**

**Add Product.**

**Distributors.**

Manufacturer: Samsung Amount Due: 24999 INR

Consumer: Amit

Address From: Noida, Uttar Pradesh

Address To: Delhi

Distributors: Priyanshu

Item	Item Description	Cost (INR)	Quantity	Total (INR)
Samsung S6 lite tablet	Immerse yourself into a c	1	24999	24999

Reset Table Creating...

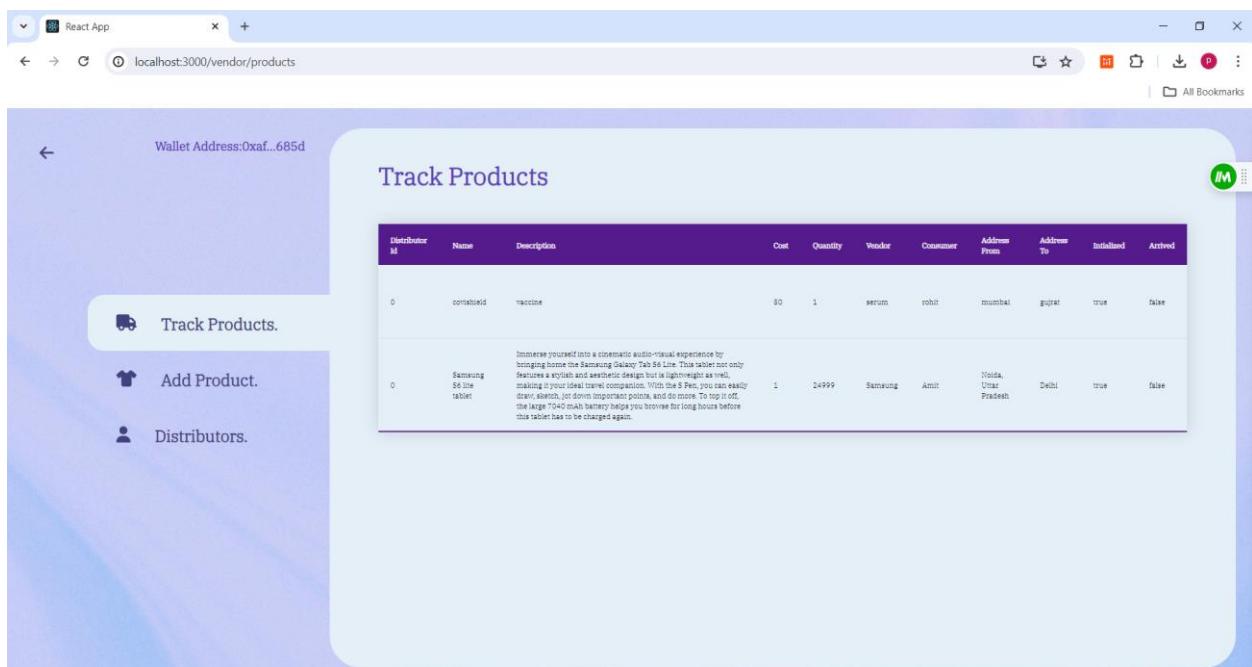
The MetaMask interface shows the transaction status as "Creating...". The "Confirm" button is still visible.

Click on 'Confirm'.

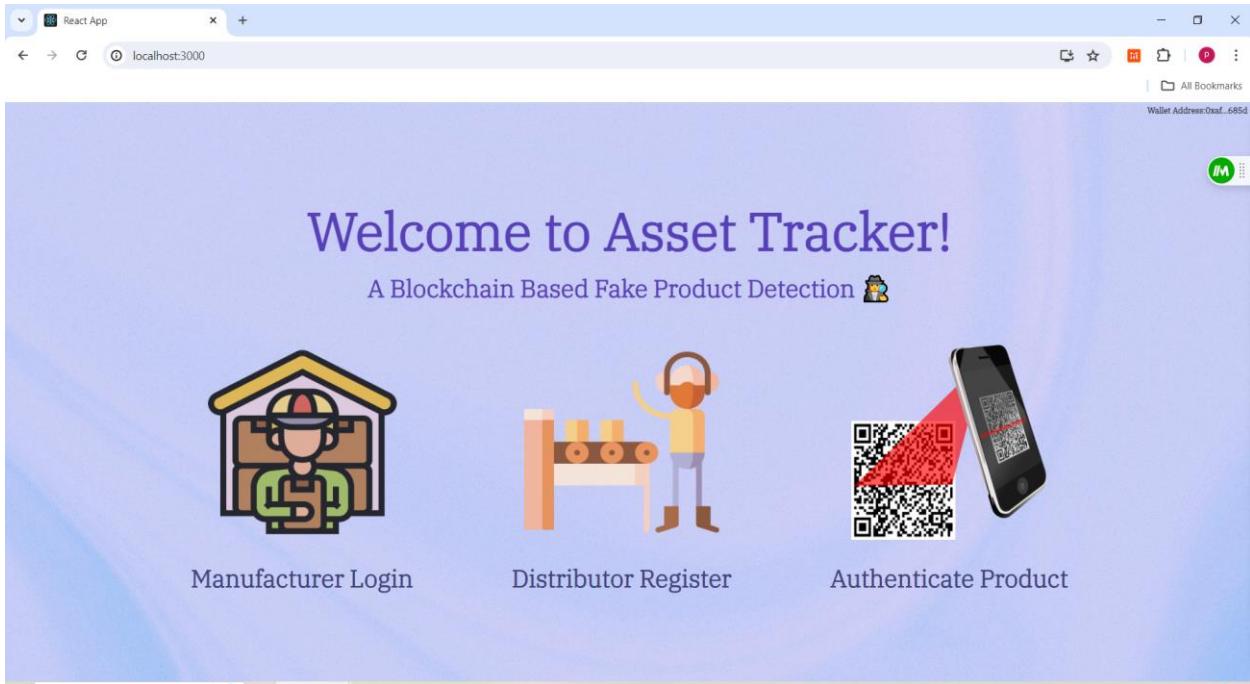


Download the QR code by right click on it and the save as image.

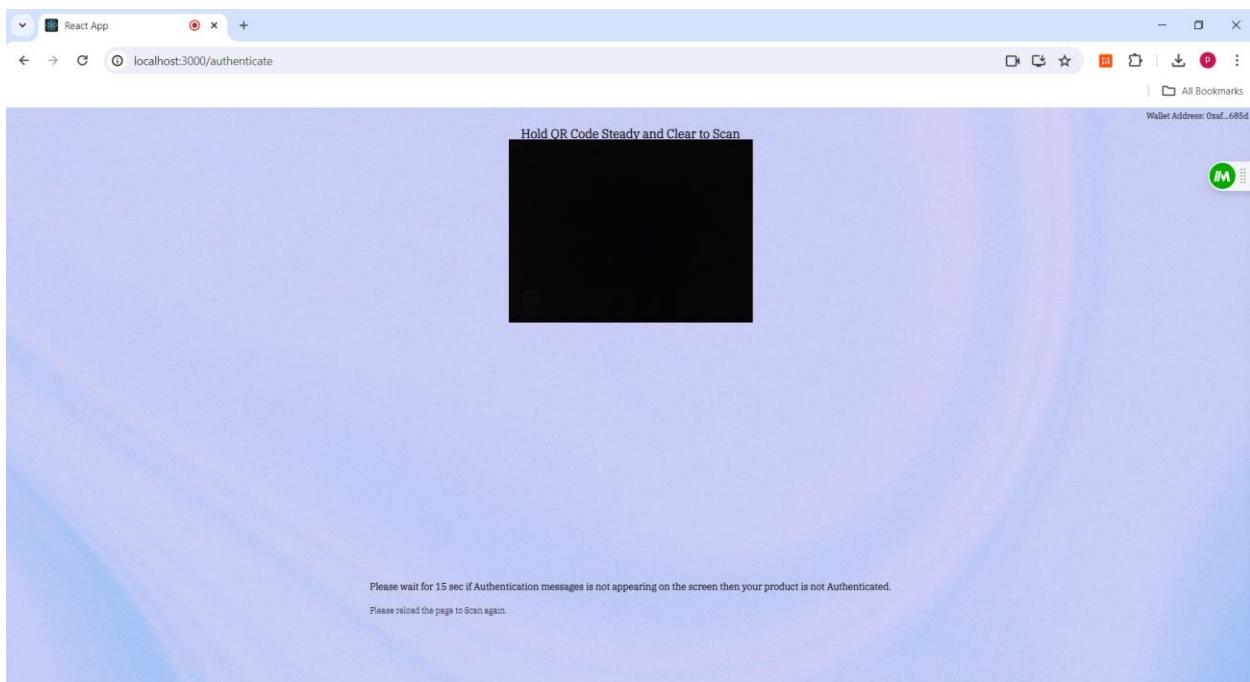
Click on 'Track Products' to view the added product. The 'arrived' column will show 'false' for that product since it has not been delivered or authenticated by the consumer.



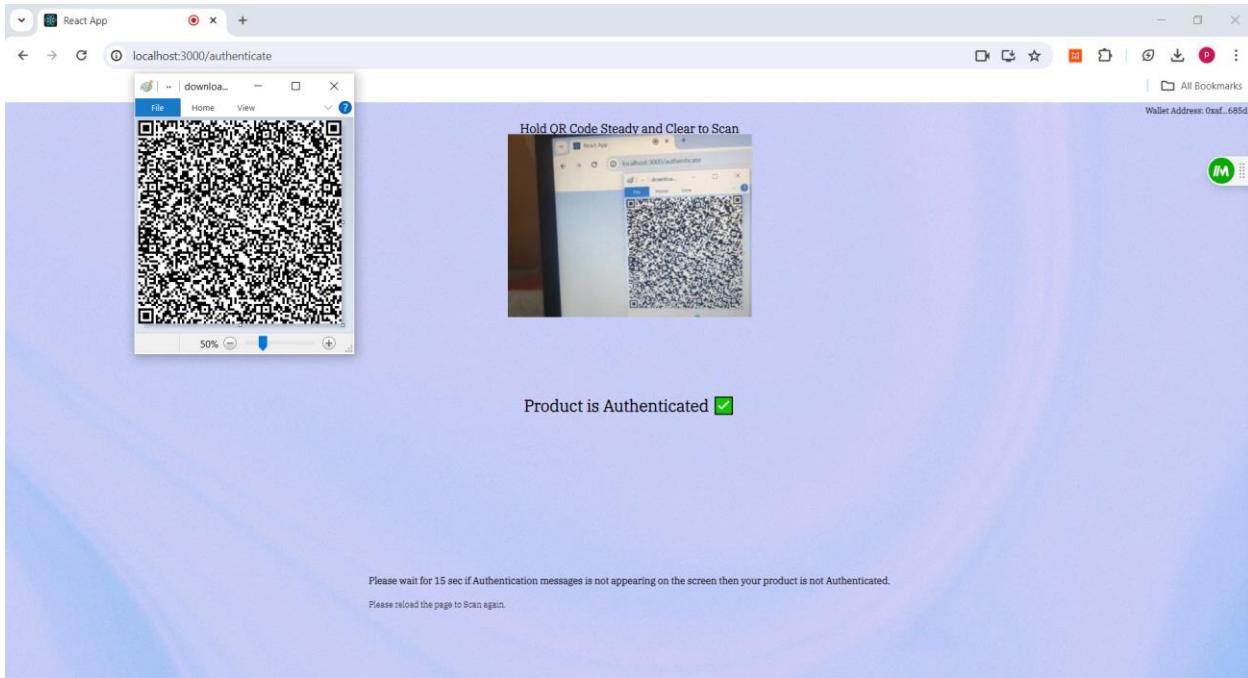
Return back to Home page for product authentication.



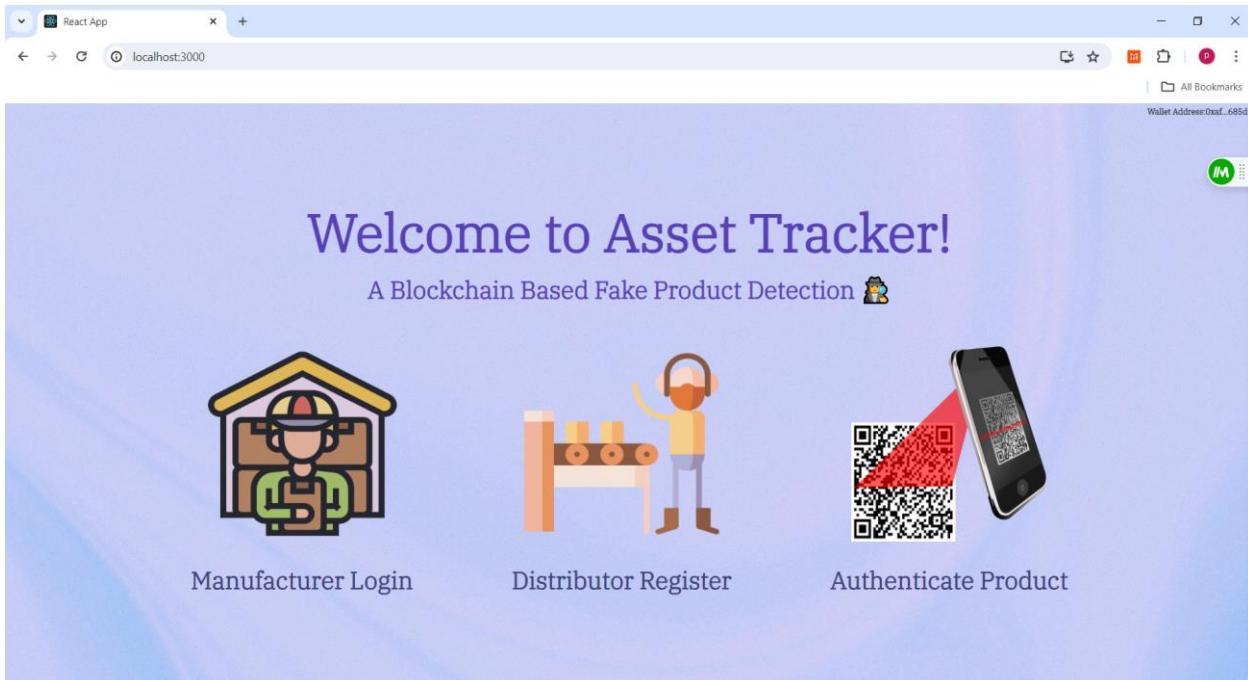
Click on 'Authenticate Product'

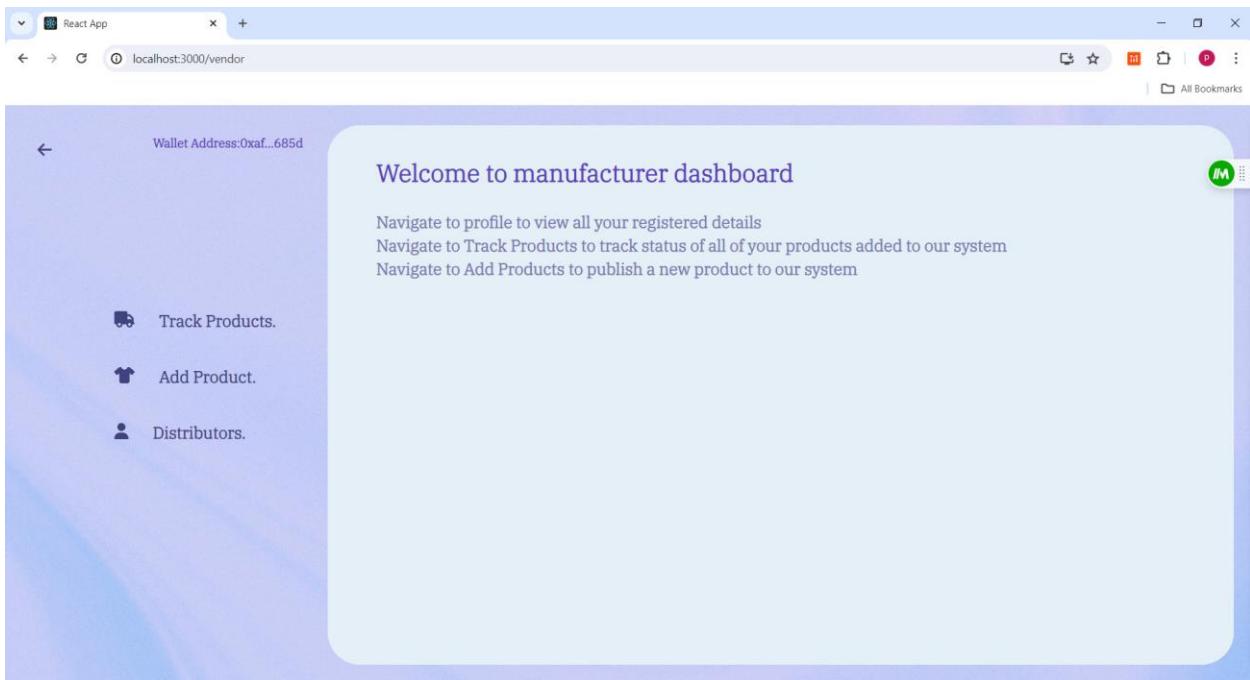


Now scan the downloaded QR code. If the QR code has not been tampered with, it will show that the product is authenticated; otherwise, it will not.



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.





The screenshot shows a web browser window titled "React App" with the URL "localhost:3000/vendor/products". The header includes the wallet address "Wallet Address:Oxaf...685d". The main content is a table titled "Track Products". The table has a purple header row with columns: Distributor Id, Name, Description, Cost, Quantity, Vendor, Consumer, Address From, Address To, Initiated, and Arrived. There are two data rows:

Distributor Id	Name	Description	Cost	Quantity	Vendor	Consumer	Address From	Address To	Initiated	Arrived
0	consaheld	vaccine	50	1	serum	robin	mumbai	gujarat	true	false
0	Samsung Tab S6	Immerse yourself into a cinematic audio-visual experience by bringing home the Samsung Galaxy Tab S6 Line. This tablet not only features a stylish and aesthetic design but is lightweight as well. It also has a 10.5 inch Super AMOLED display and a 10.5 inch S Pen for a smooth writing and drawing experience. Its 7040 mAh battery helps you browse for long hours before this tablet has to be charged again.	24999	1	Samsung	Amit	Uttar Pradesh	Delhi	true	true

The sidebar on the left is identical to the one in the first screenshot, featuring "Track Products.", "Add Product.", and "Distributors." options. The footer of the page shows the URL "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://codingwithmanni.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>