**Internal Exam**

**AIM:**

Install and configure the following development setup tools to implement Blockchain development. (Set up Blockchain Development Environment)

* MetaMask (Wallet)
* Ganache Local Private Blockchain Network
* Go-Ethereum (Geth) Client
* Truffle framework
* Hardhat framework

Study and configure all testnets available in MetaMask, and also setup a custom network using Ganache.

**Output:**

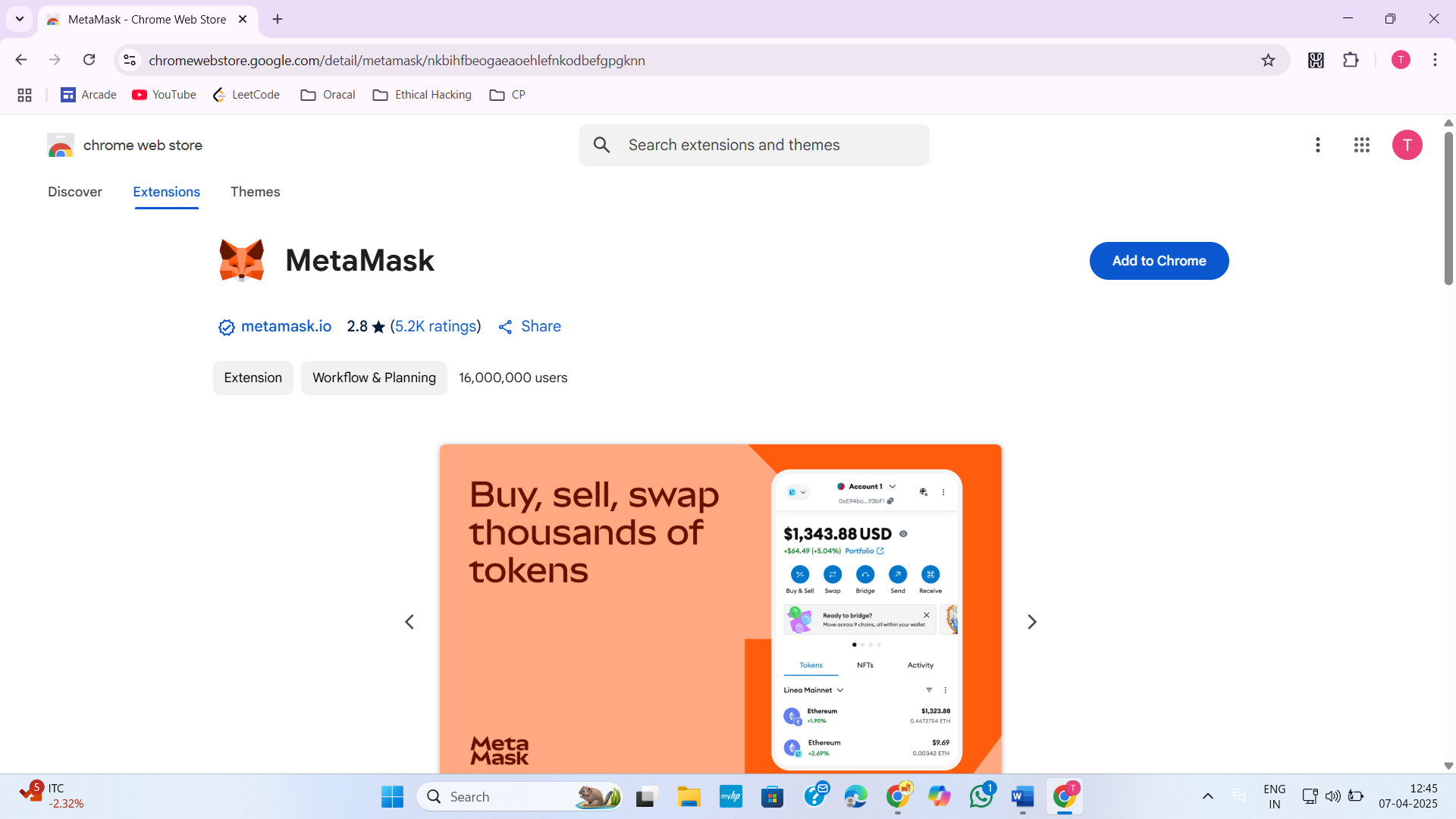


Figure 1:Open Chrome web store and search MetaMask

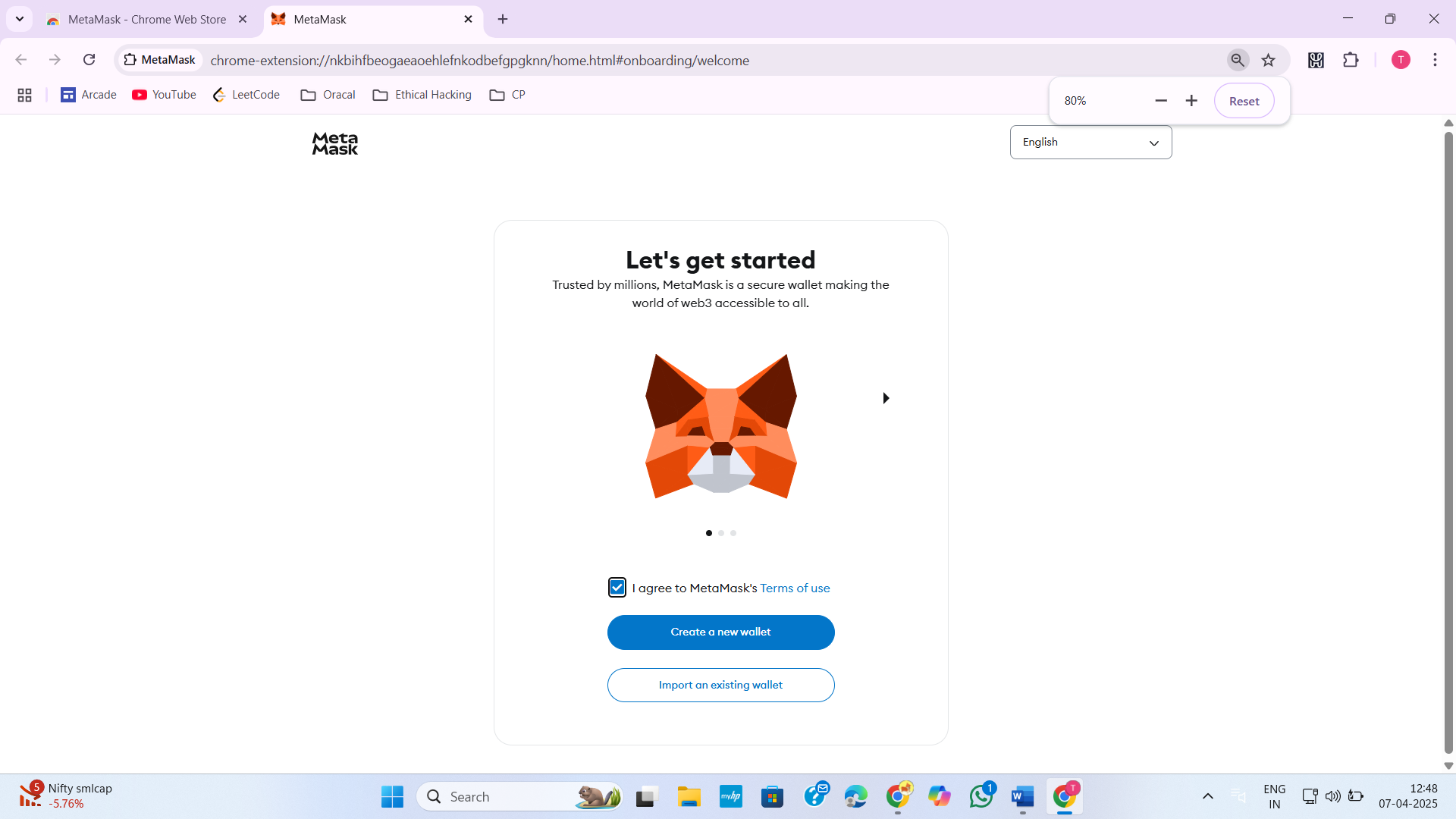
****

Figure 2:Download MetaMask and create a new wallet

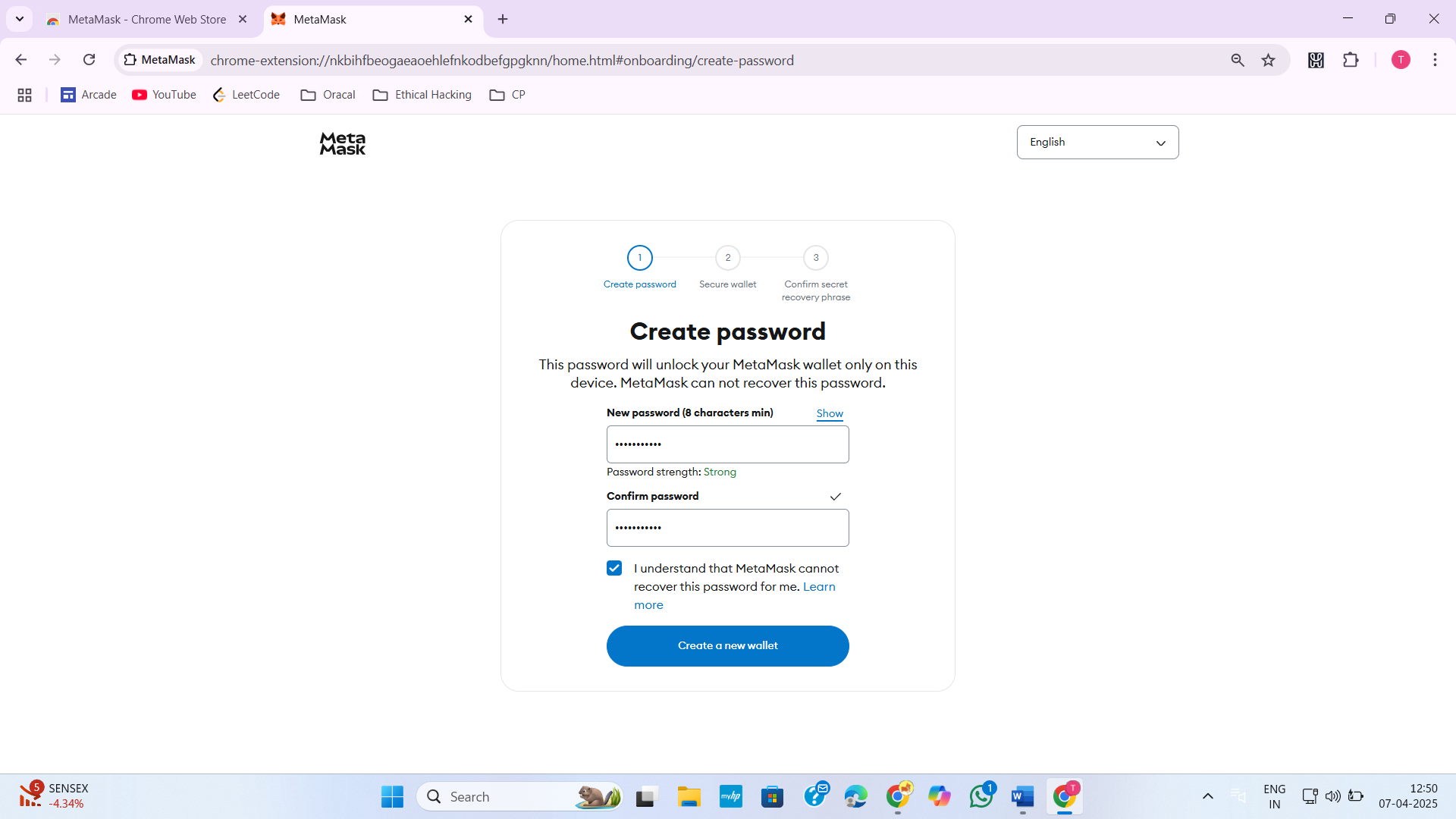


Figure 3:Set a Password for my wallet

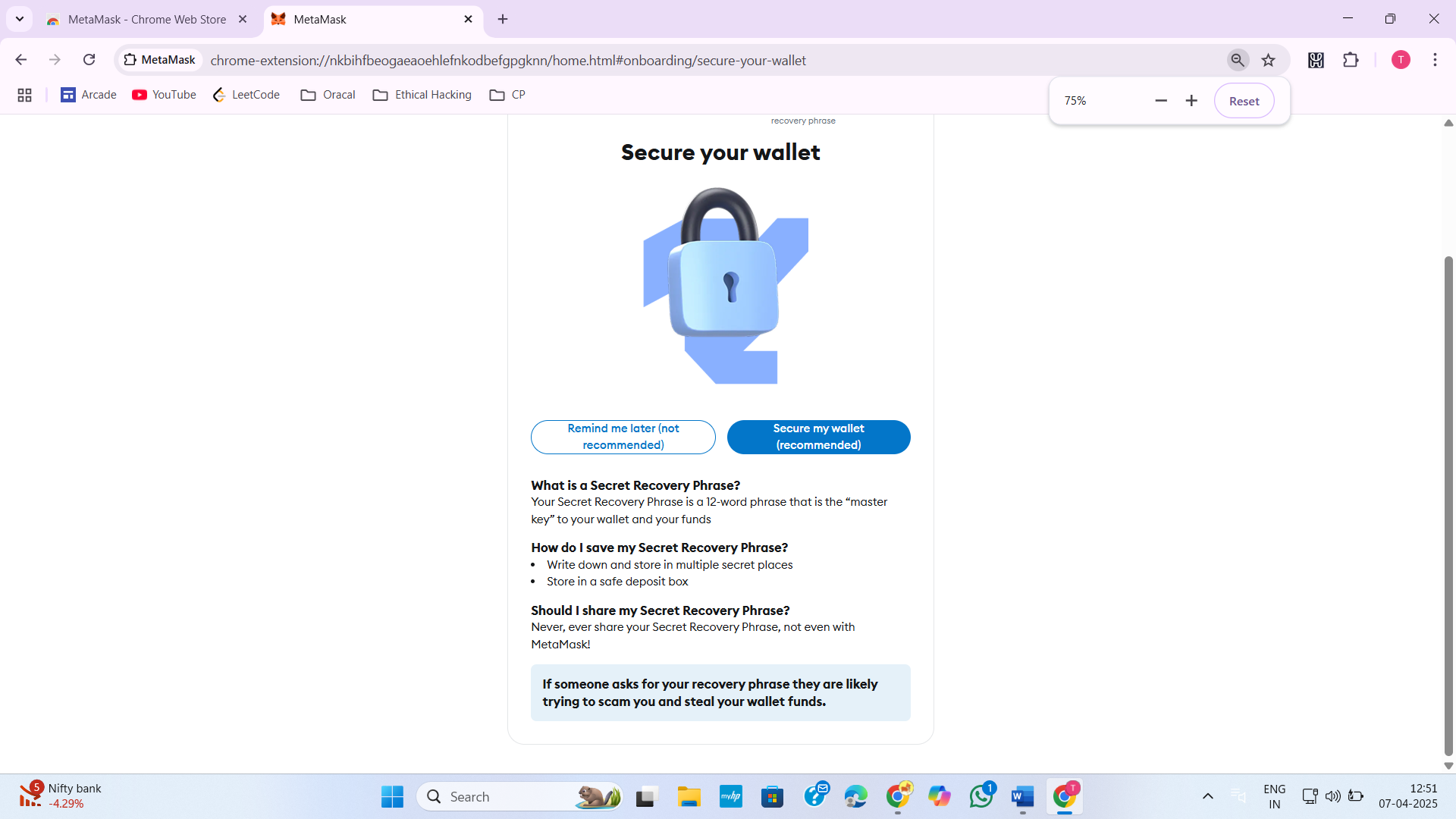


Figure 4:Secure a wallet

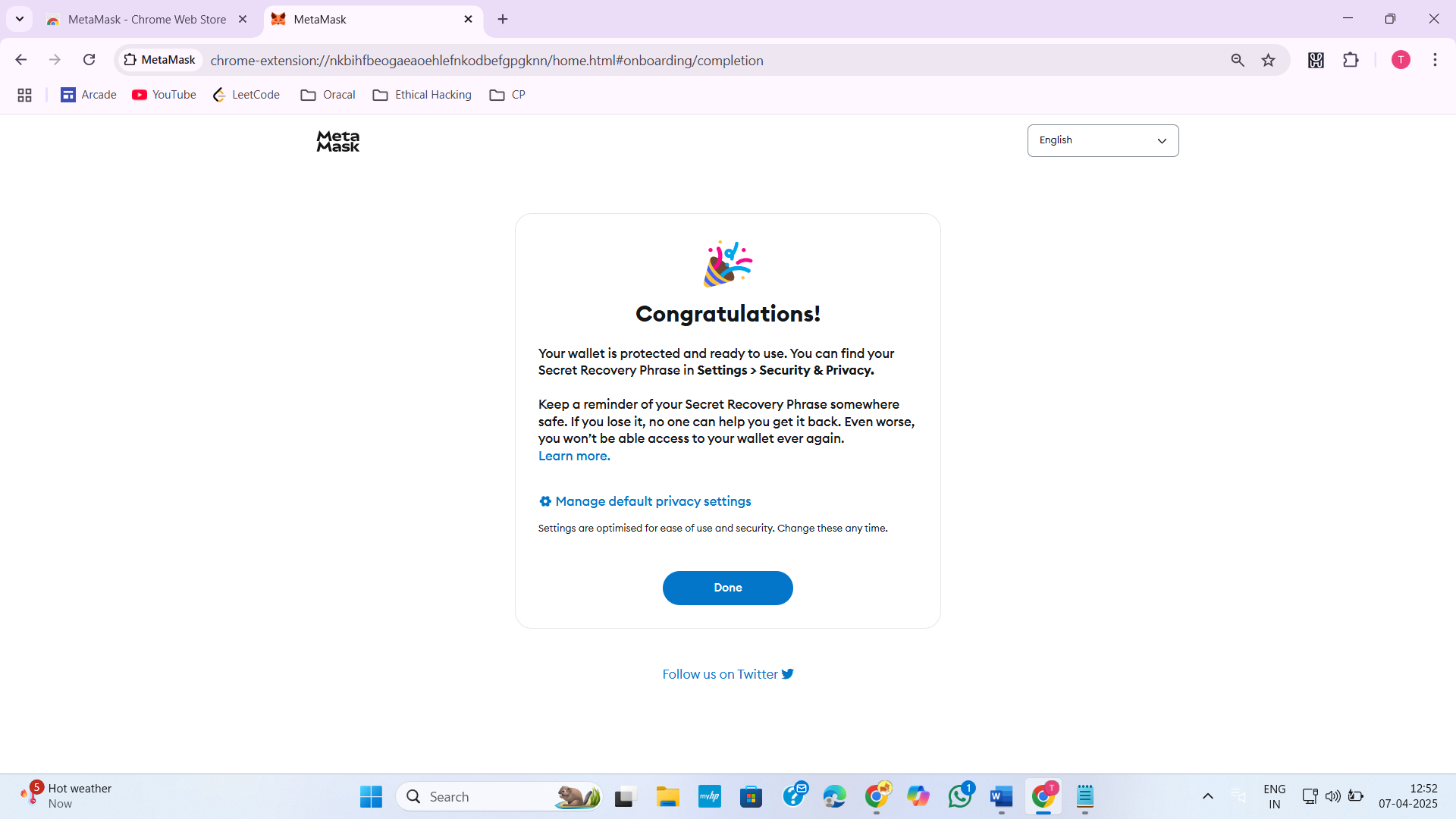


Figure 5:MetaMask wallet created successfully

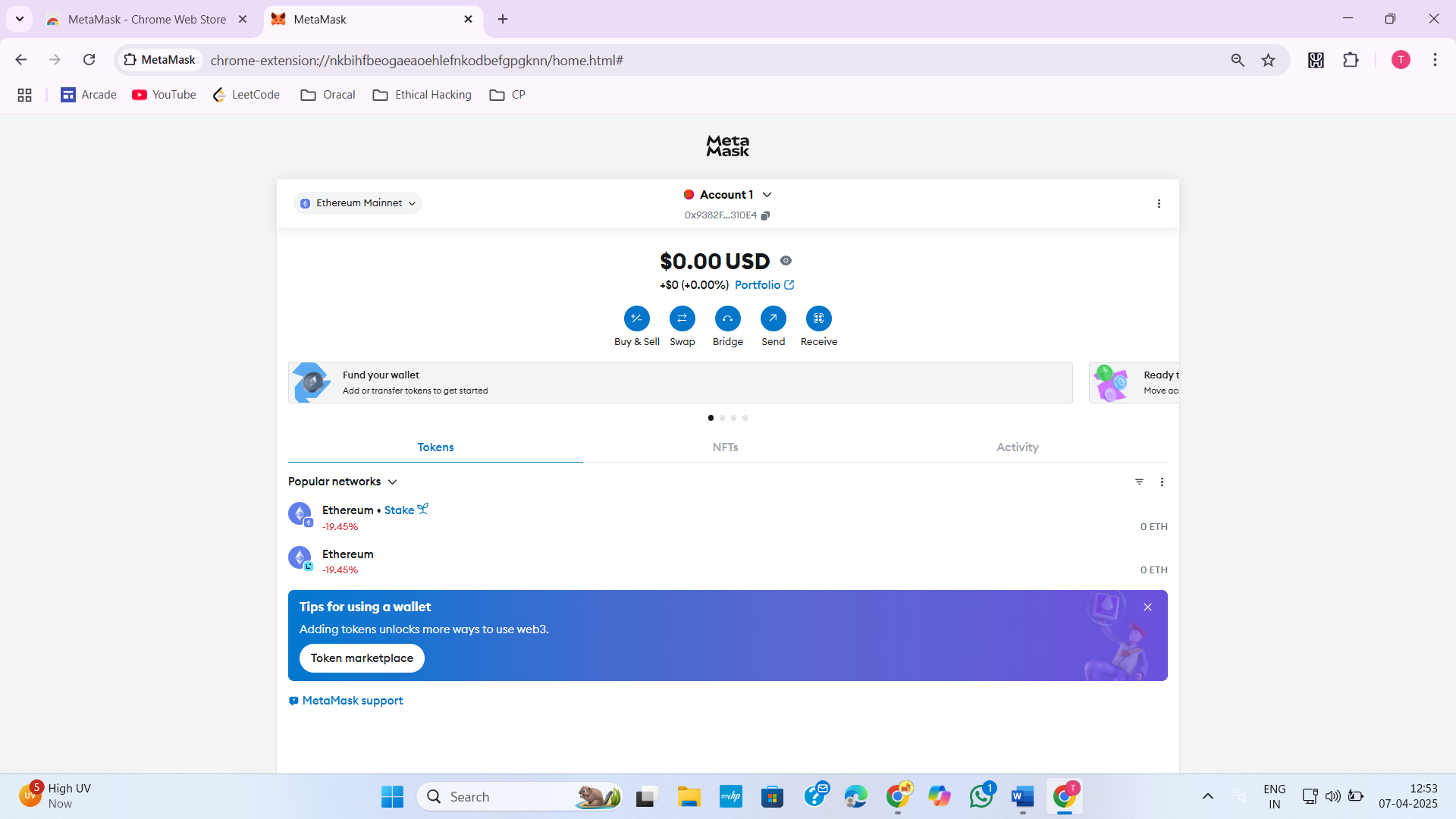


Figure 6:Homepage of MetaMask

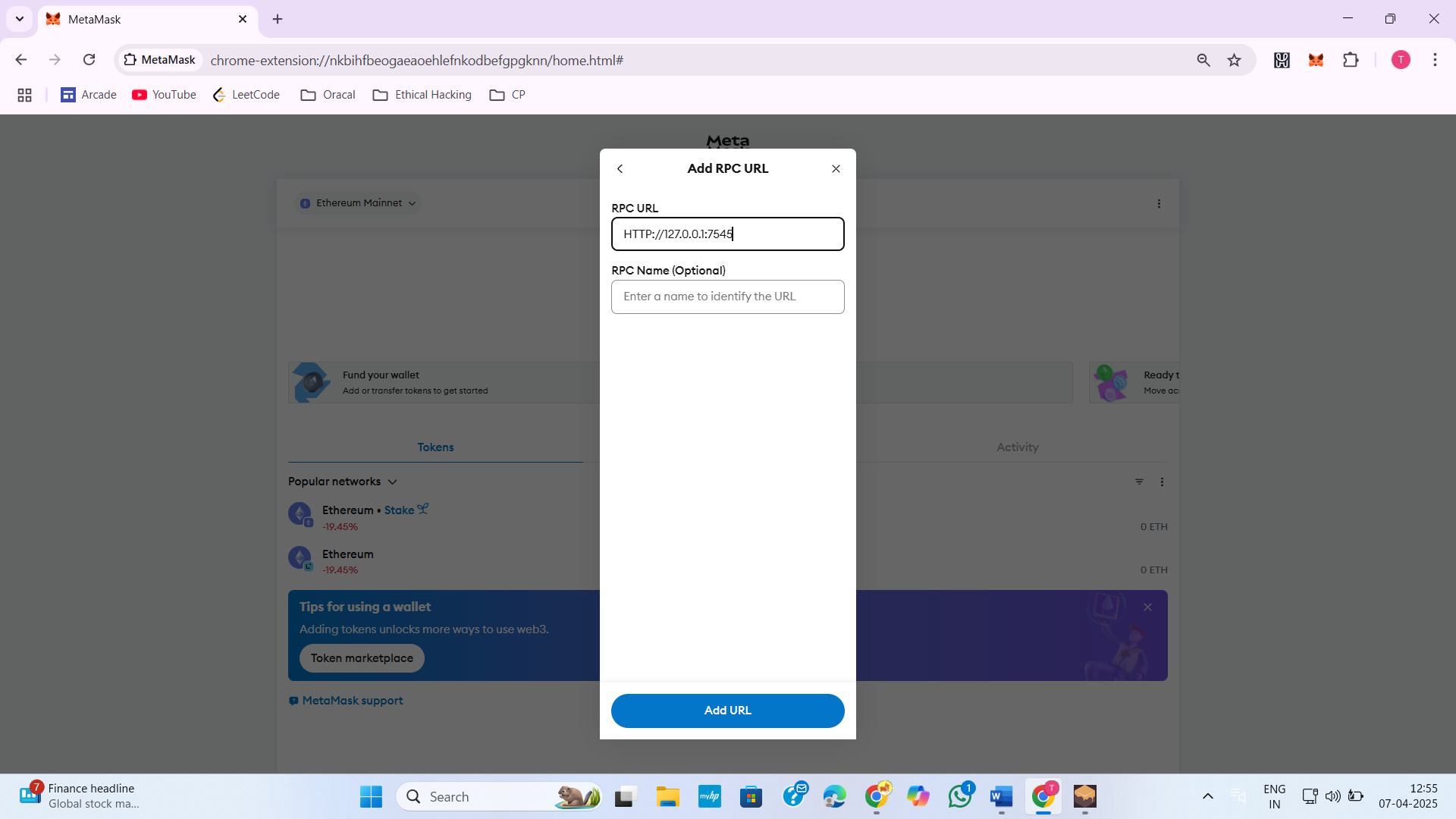


Figure 7:Add RPC URL for Ganache Local Private Blockchain Network

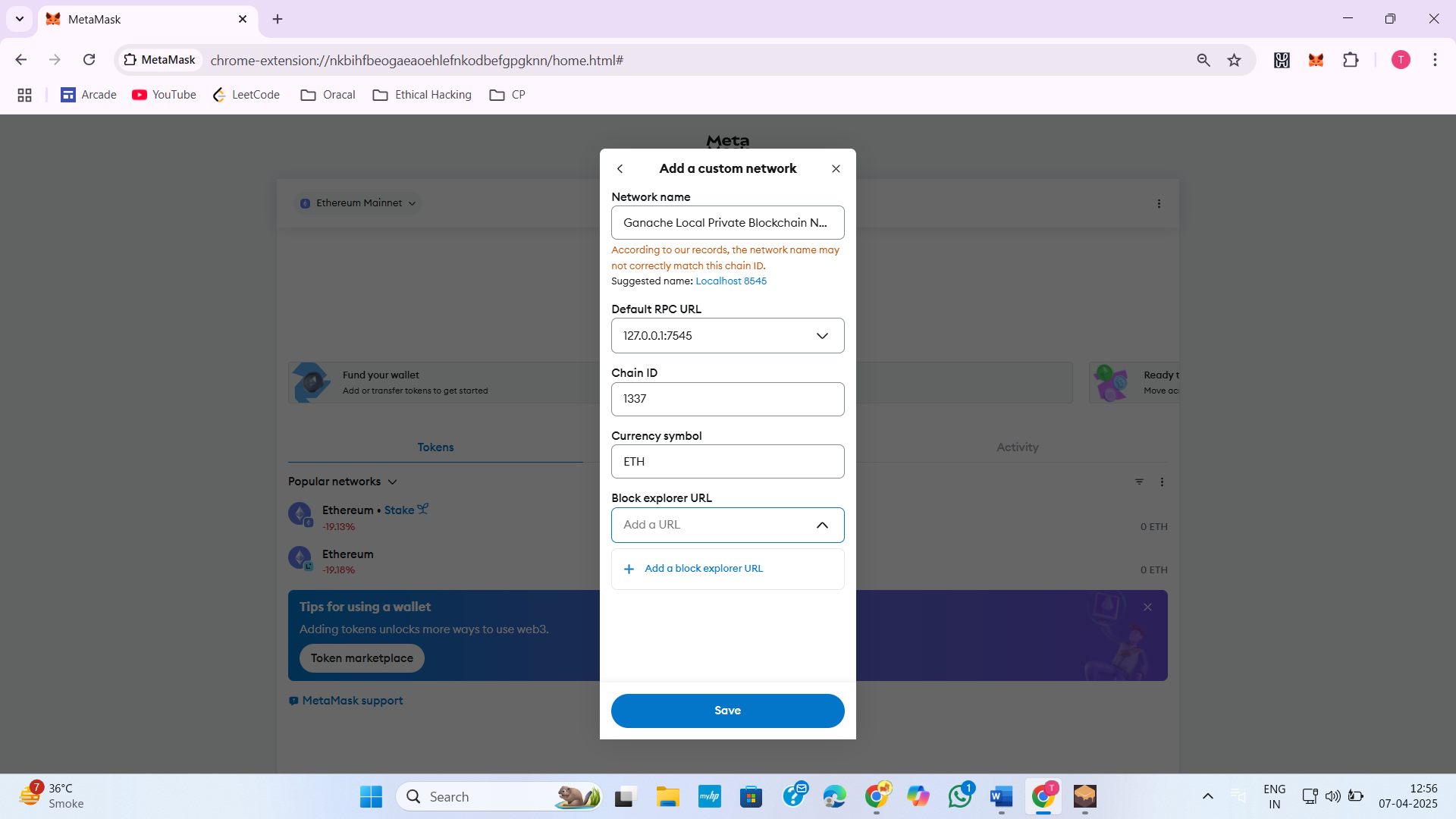


Figure 8:Add Ganache Network

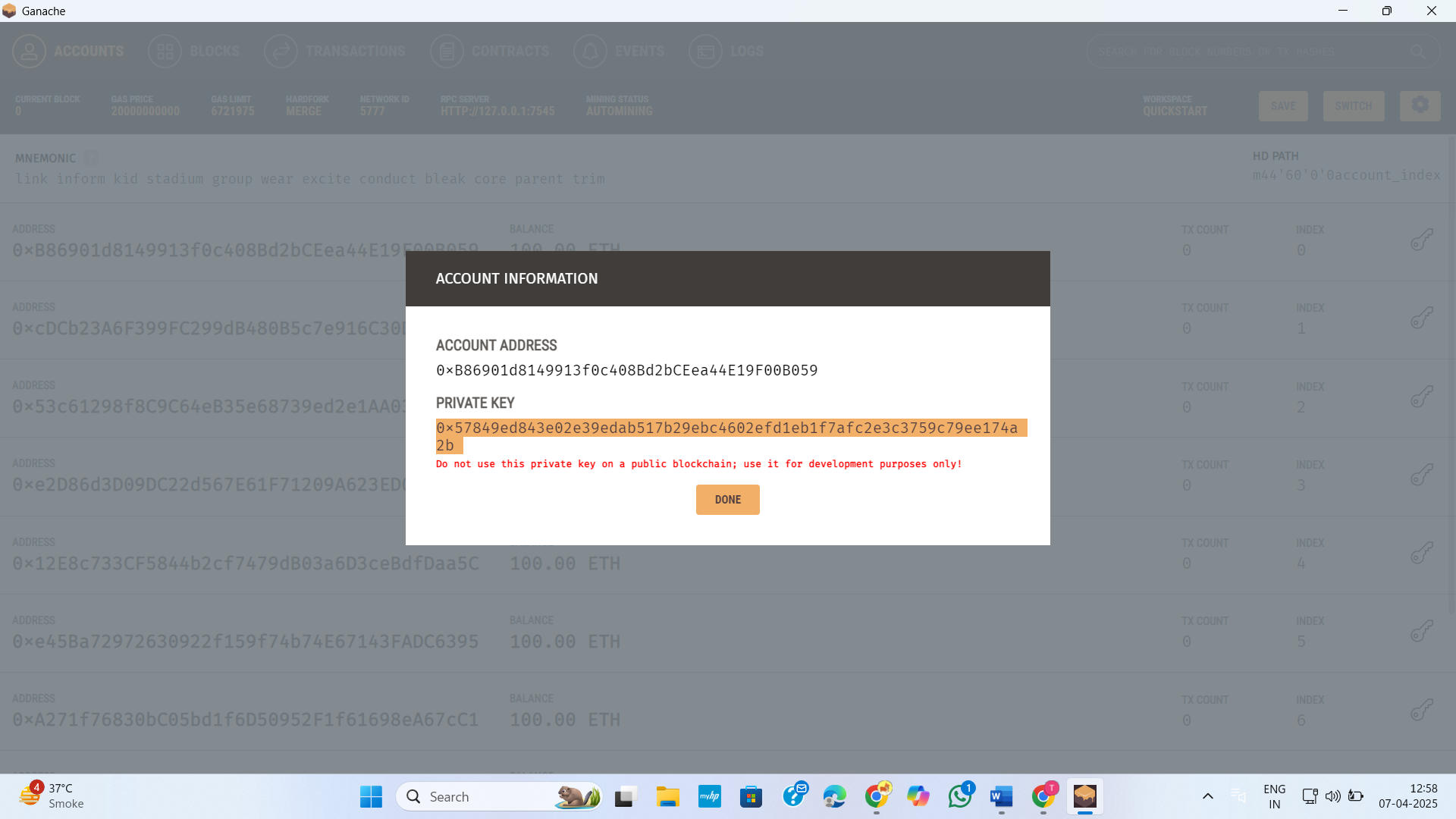


Figure 9:Take Private key from Ganache network an import account

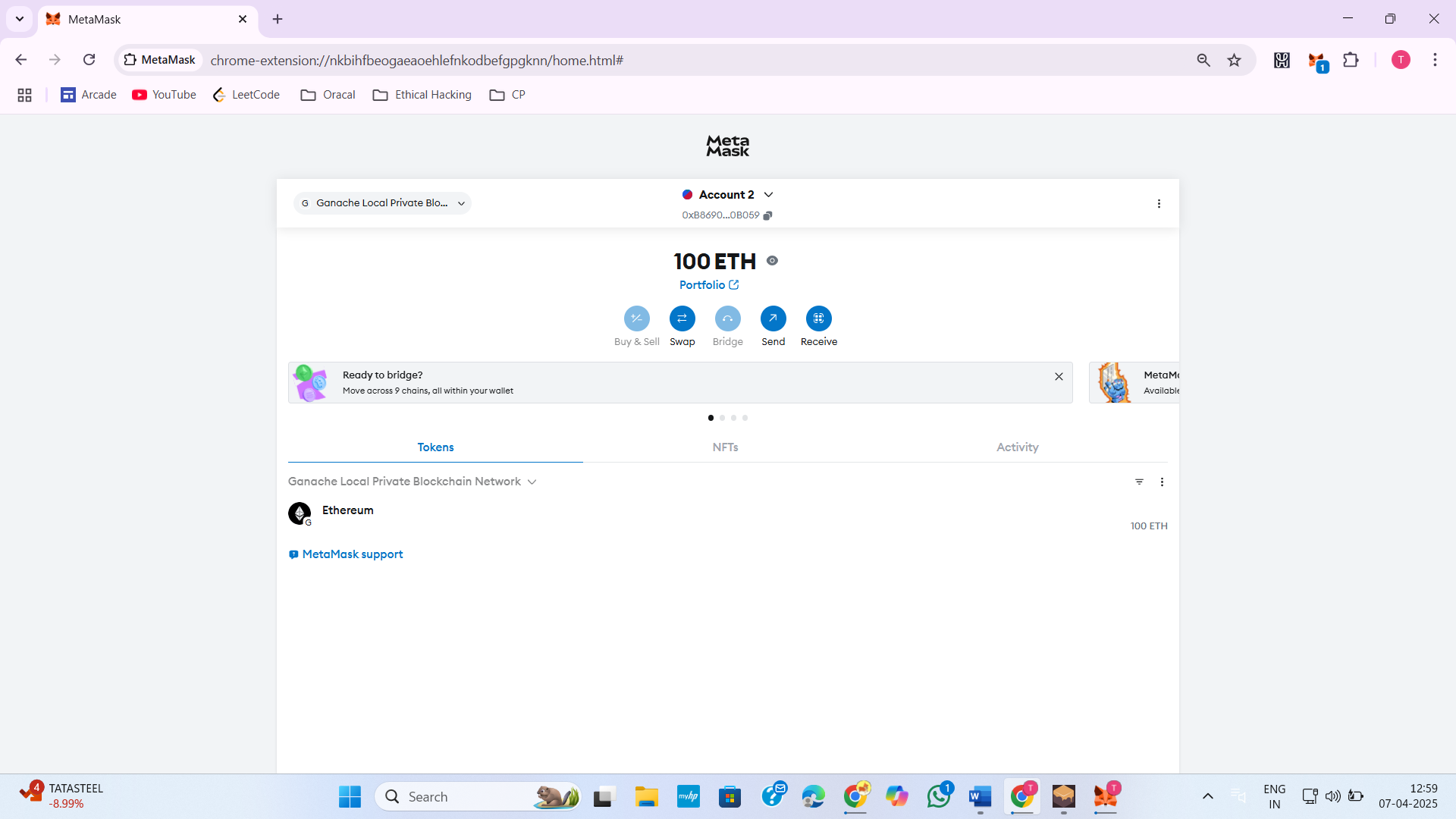


Figure 10:Account imported successfully

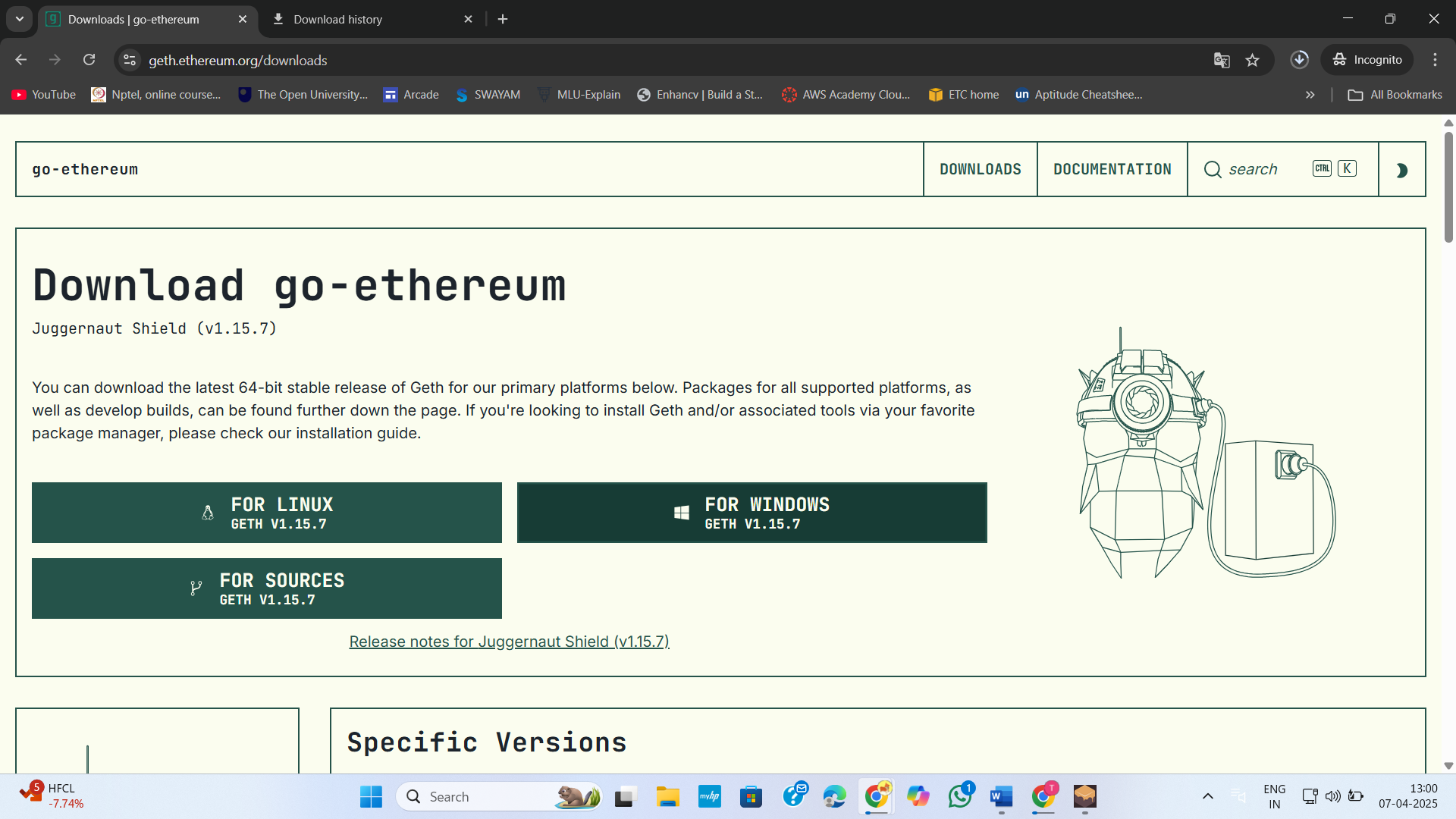


Figure 11:Download go-Ethereum for windows

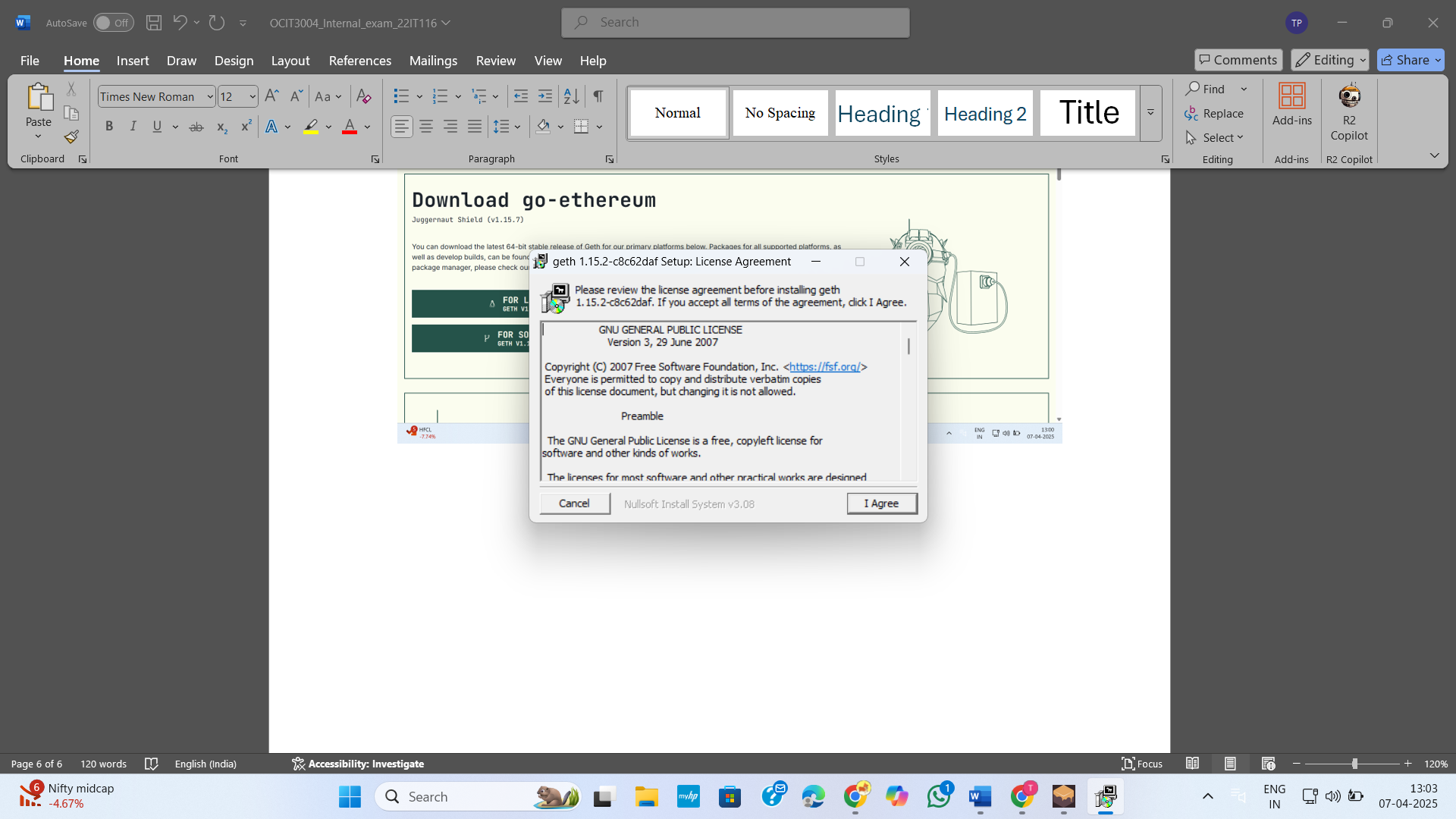


Figure 12:Install go-Ethereum

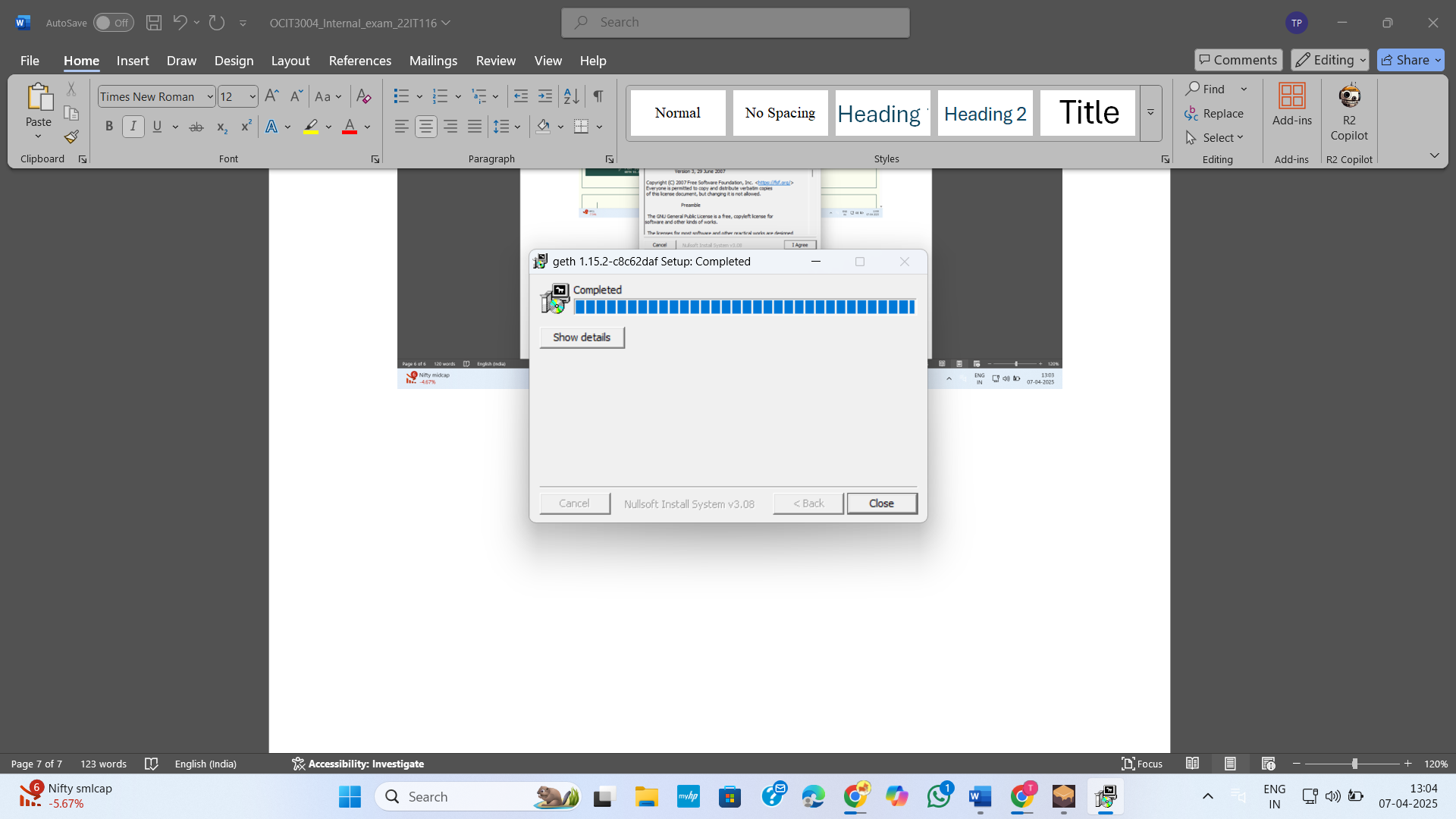


Figure 13:Install completed for Geth

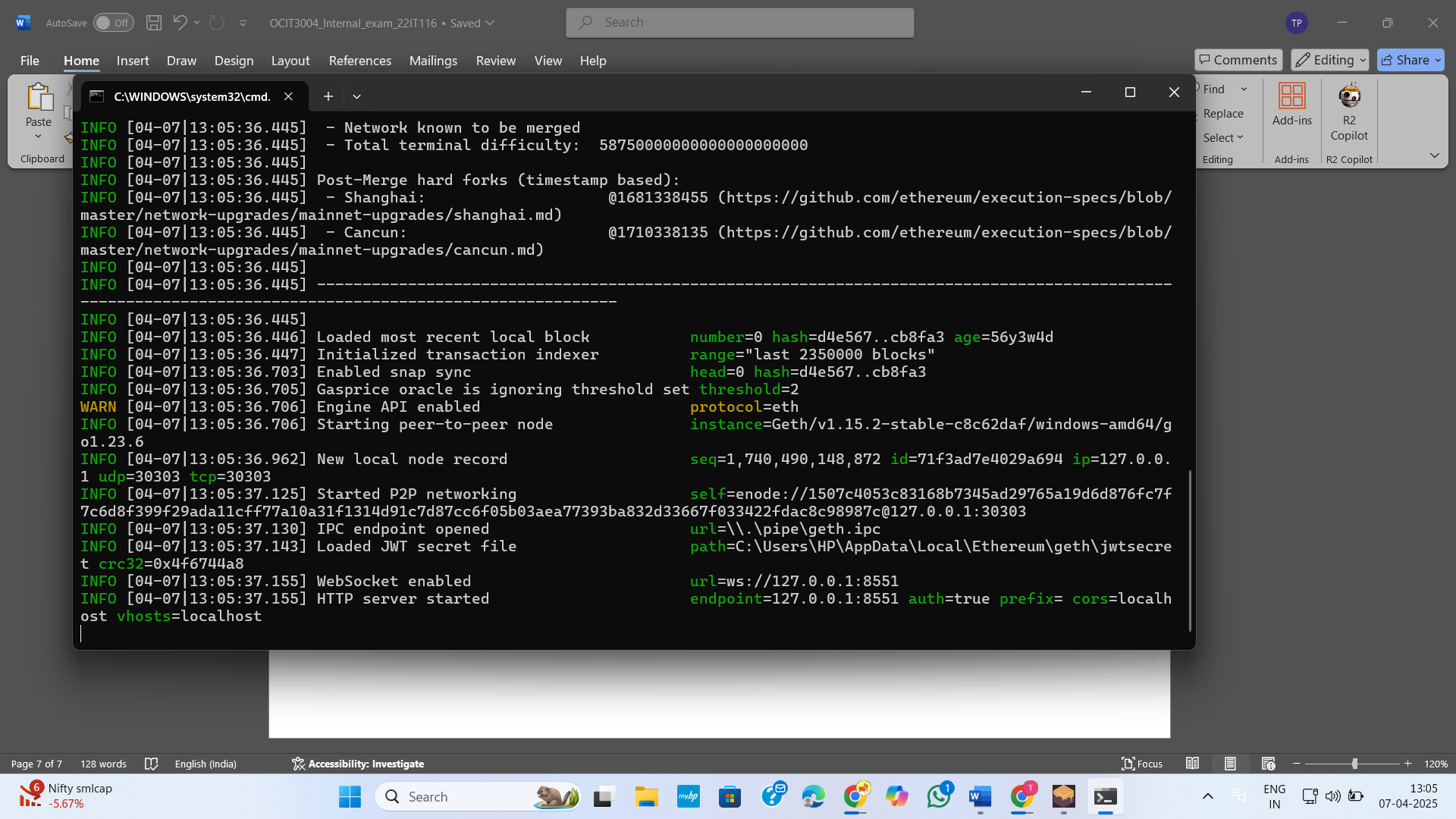


Figure 14:Open cmd and run Geth

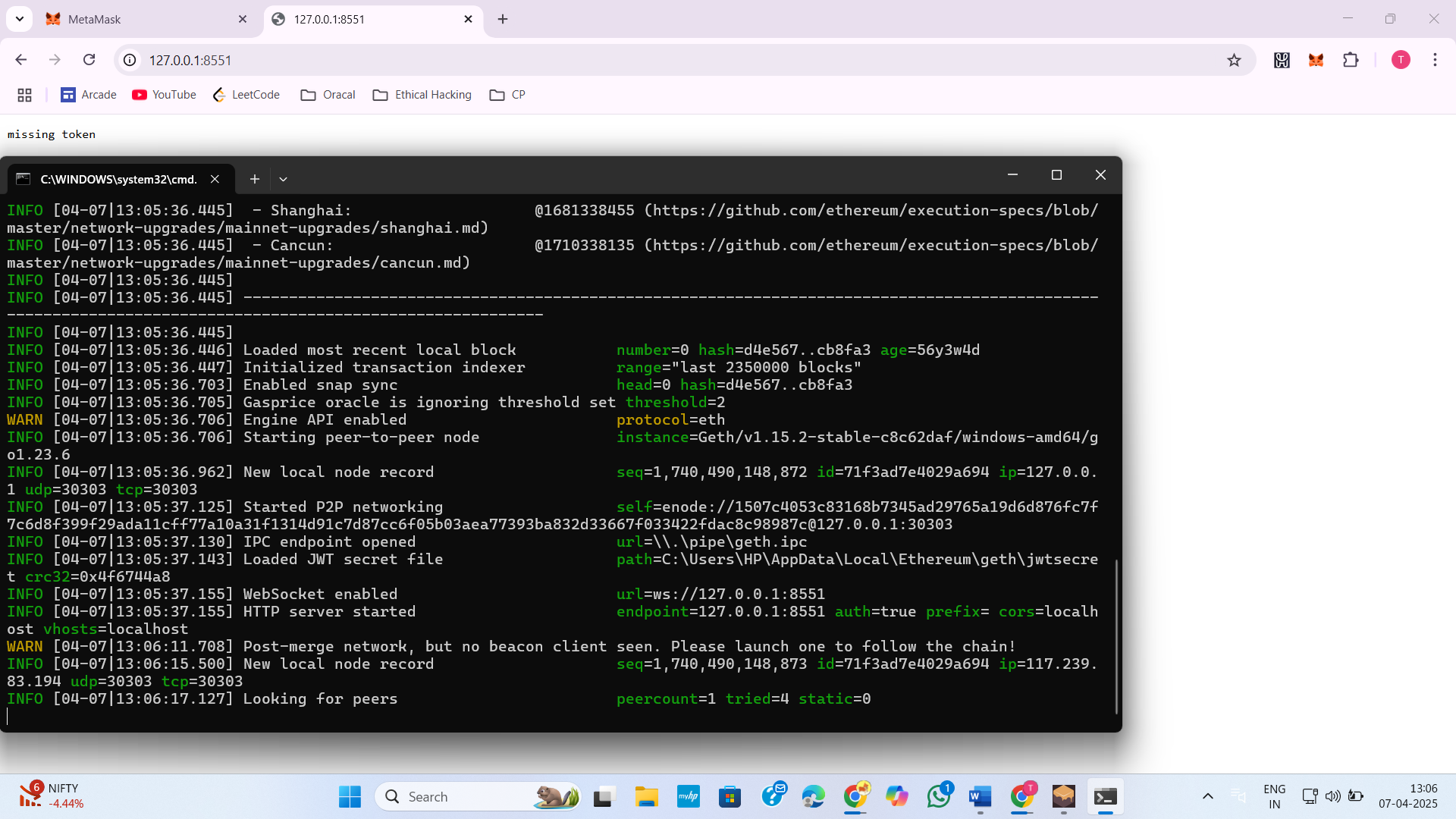


Figure 15:If we try to access HTTP server using chrome my record add in Geth

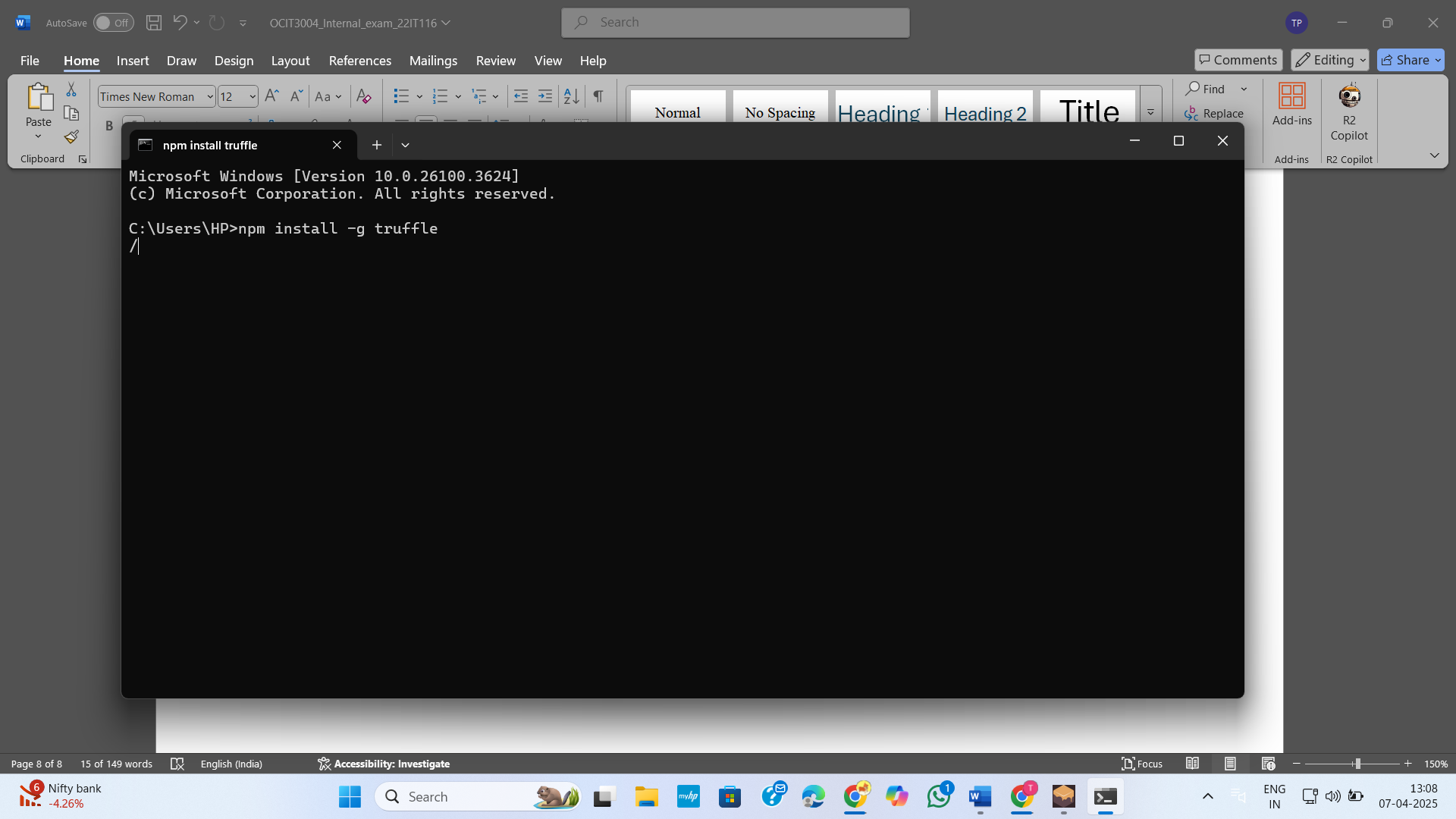


Figure 16:Install truffle using **npm install -g truffle**

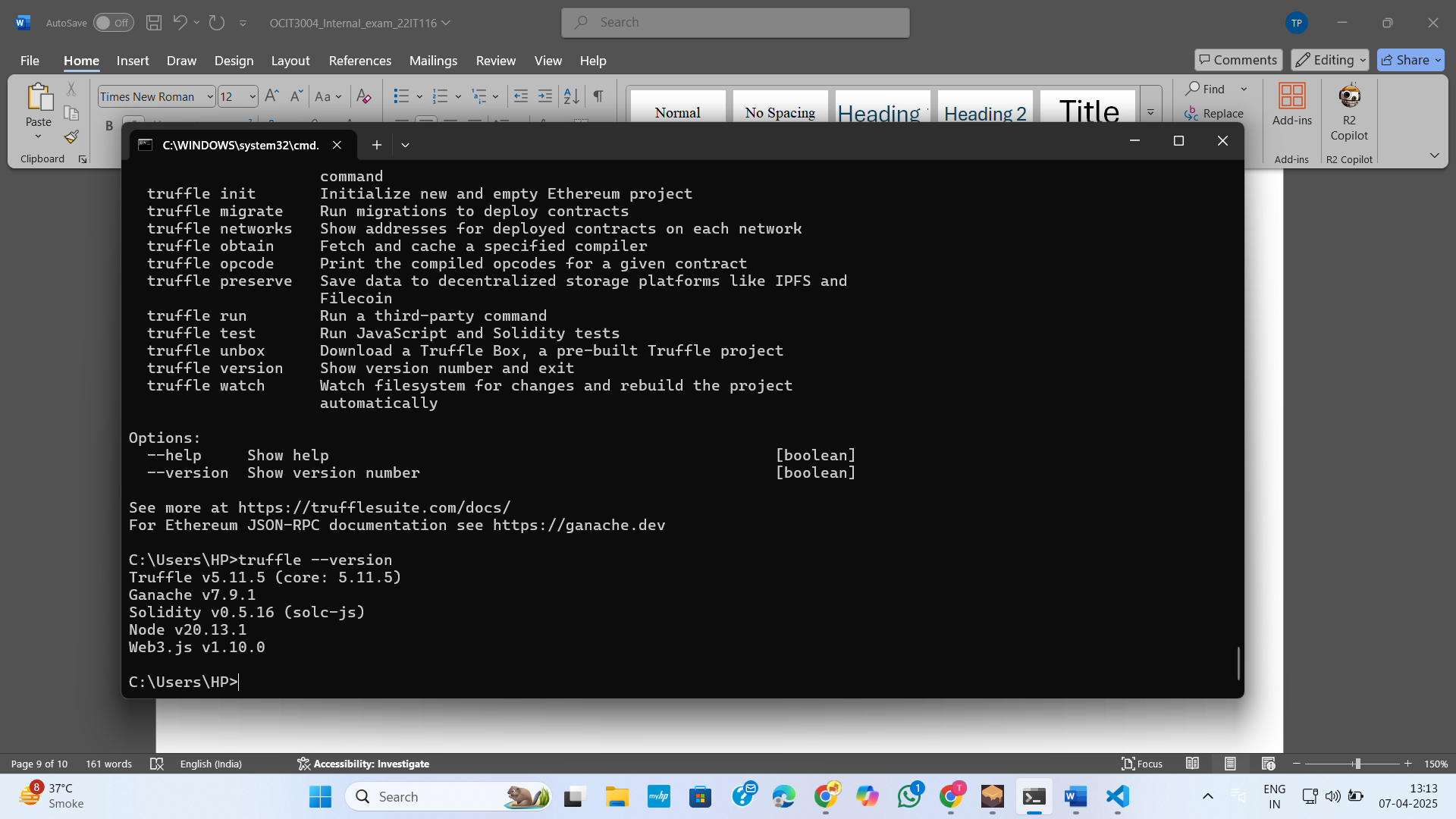


Figure 17:Truffle install successfully



Figure 18:Create a folder for hardhat

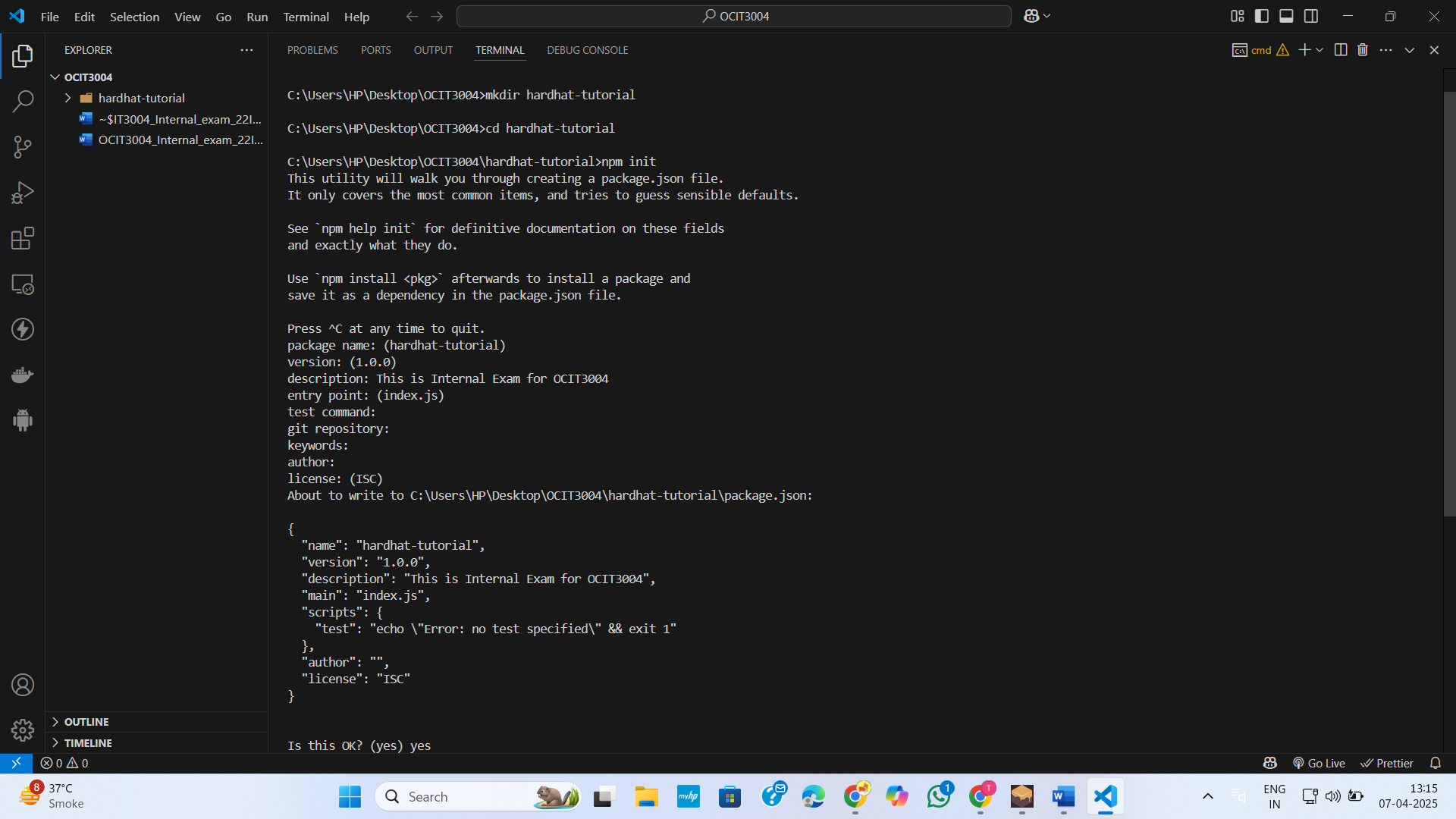


Figure 19:Run **npm init**

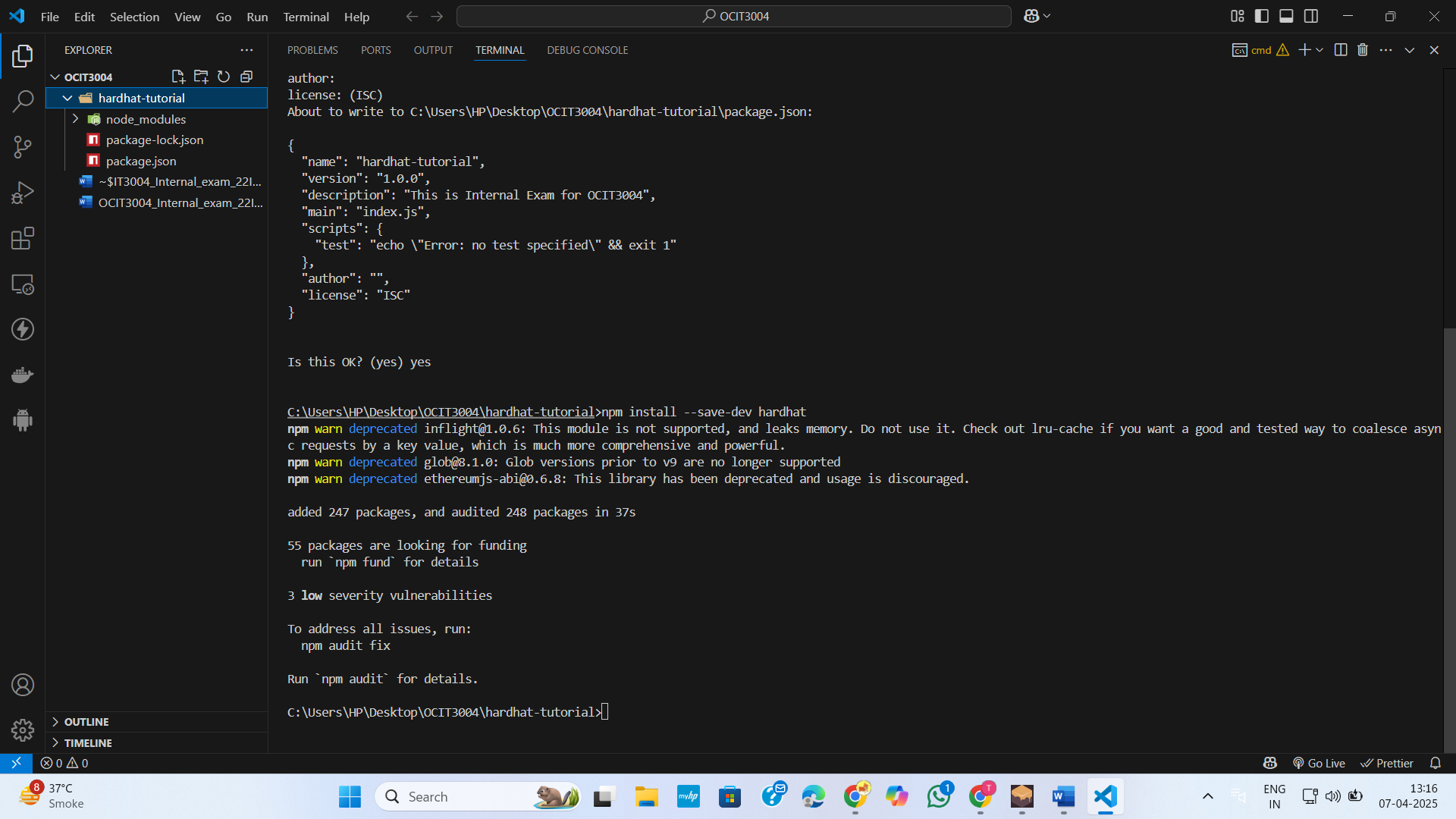


Figure 20:Suceessfully installed hardhat

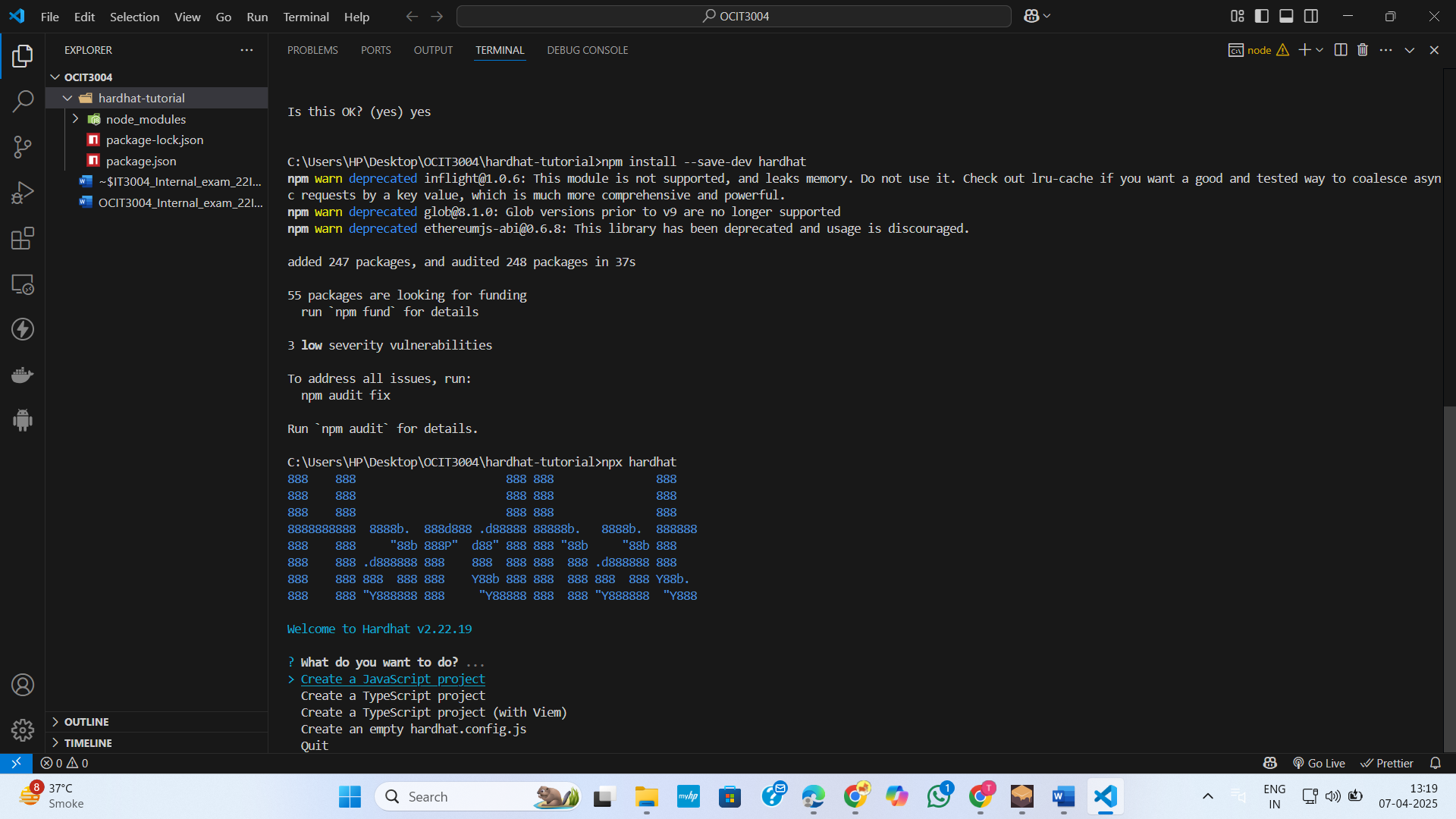


Figure 21: Initialize a new Hardhat project

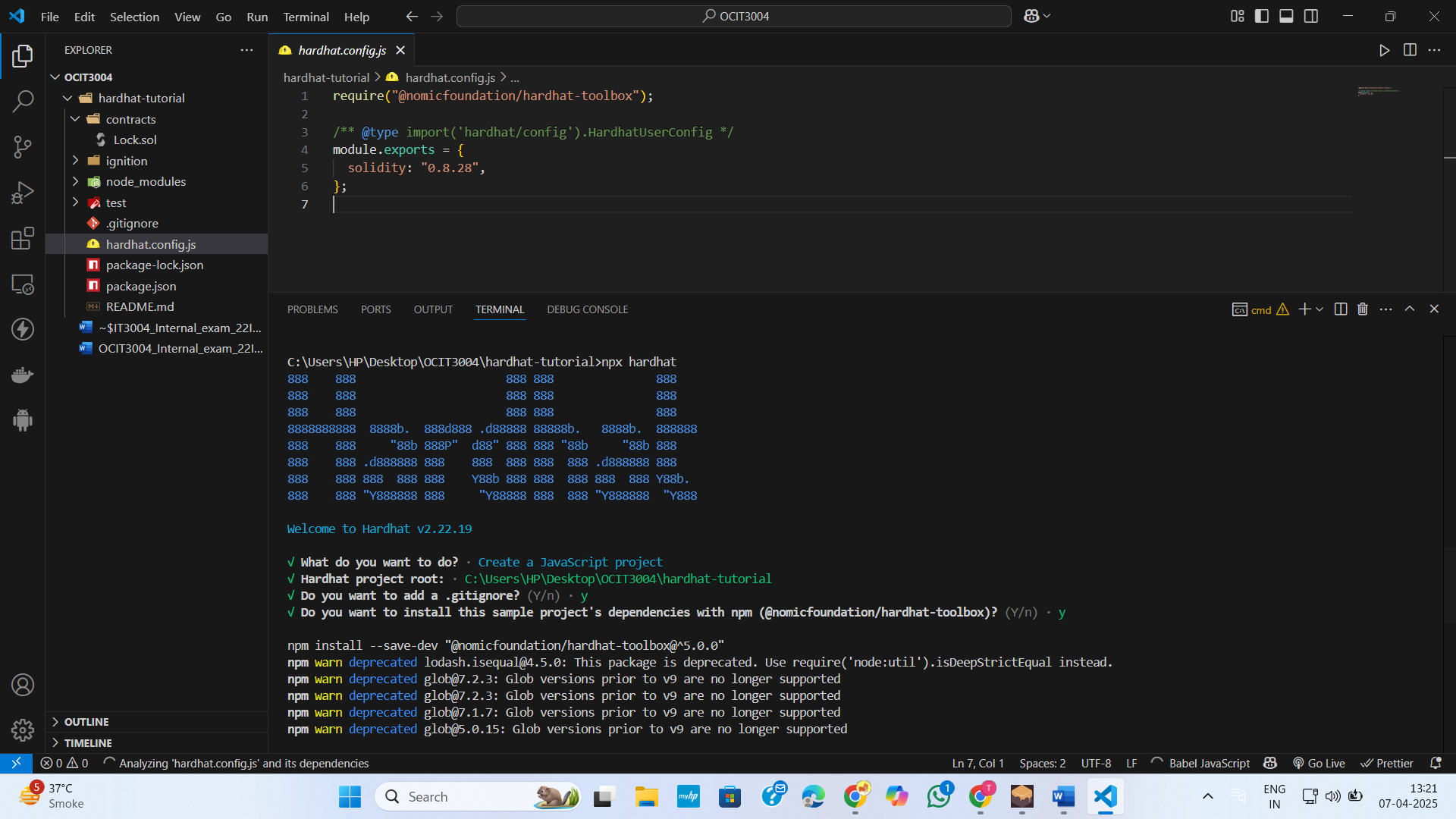


Figure 22:Successfully created hardhat project

**AIM:**

Decentralized Voting System:

* Design a Solidity smart contract that allows multiple candidates to be voted on by eligible voters.
* Each voter can vote only once.
* At the end of the election, the contract should allow anyone to query the winner.
* Ensure that the contract is secure and prevents malicious actions, such as double voting.

**Code:**

// SPDX-License-Identifier: MIT

pragma solidity ^0.8.0;

contract E\_voting {

    struct Candidate {

        string name;

        uint voteCount;

    }

    struct Voter {

        bool isEligible;

        bool isVoted;

        uint vote;

    }

    address public owner;

    mapping(address => Voter) public voters;

    Candidate[] public candidates;

    bool public electionEnded = false;

    modifier onlyOwner() {

        require(msg.sender == owner, "Only owner can call this function");

        \_;

    }

    modifier isNotVoted(){

        require(!voters[msg.sender].isVoted, "message");

        \_;

    }

    modifier isEligibleVoter(){

        require(voters[msg.sender].isEligible,"Not Eligible voter.");

        \_;

    }

    constructor(){

        owner = msg.sender;

    }

    function addCandidate(string memory name) public onlyOwner{

        candidates.push(Candidate(name,0));

    }

    function authorizeVoter(address voterAddress) public onlyOwner {

        voters[voterAddress].isEligible = true;

    }

    function vote(uint candidateIndex) public isNotVoted isEligibleVoter {

        require(candidateIndex < candidates.length, "Invalid candidate index");

        voters[msg.sender].isVoted = true;

        voters[msg.sender].vote = candidateIndex;

        candidates[candidateIndex].voteCount += 1;

    }

    function endElection() public onlyOwner {

        electionEnded = true;

    }

    function getWinner() public view returns (string memory winnerName) {

        require(electionEnded, "Election is not yet ended");

        uint maxVotes = 0;

        uint winnerIndex = 0;

        for (uint i = 0; i < candidates.length; i++) {

            if (candidates[i].voteCount > maxVotes) {

                maxVotes = candidates[i].voteCount;

                winnerIndex = i;

            }

        }

        winnerName = candidates[winnerIndex].name;

    }

}

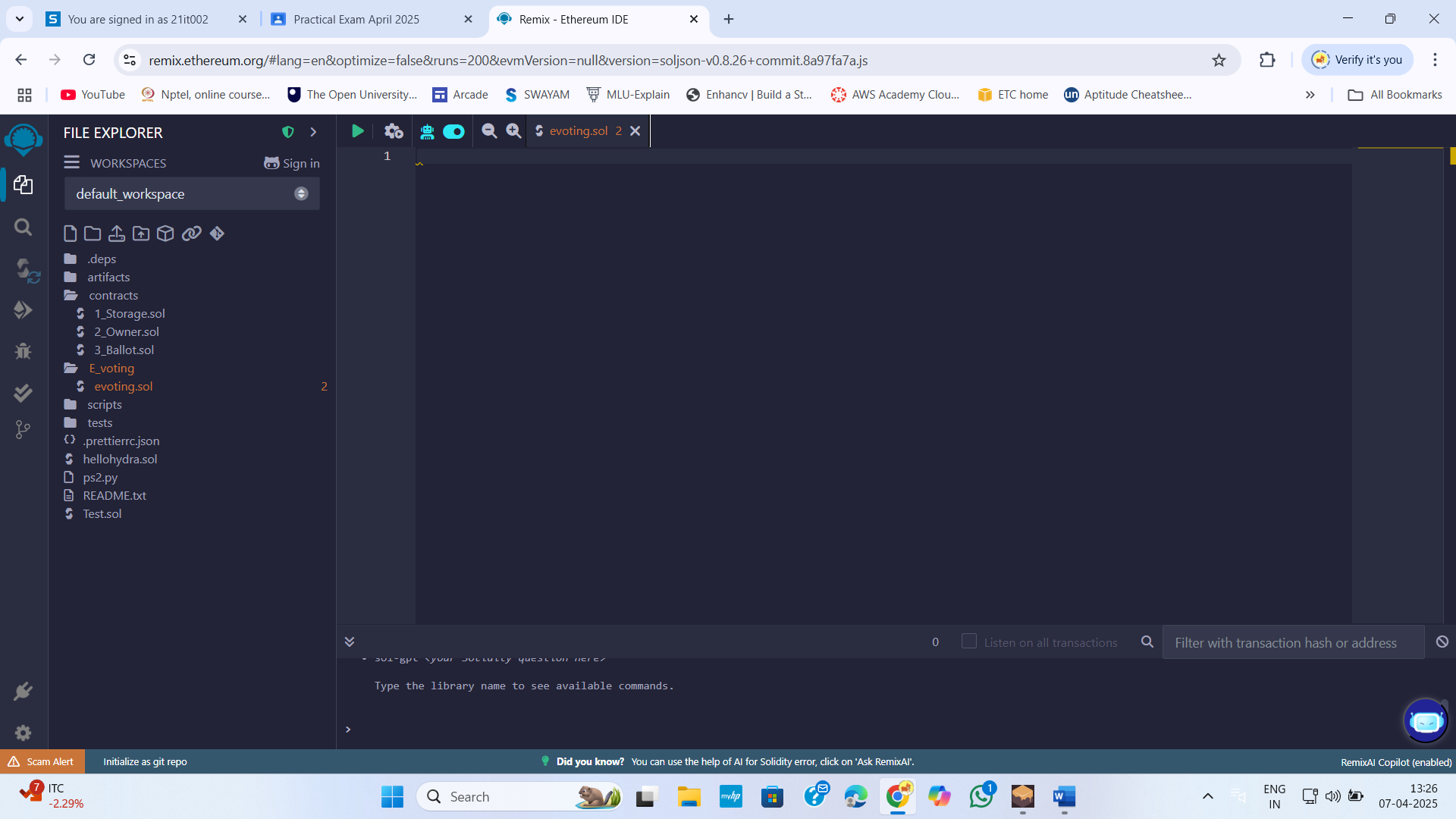
****

Figure 23:In remix IDE create a Folder E\_voting and in that folder create a evoting.sol smart contract file

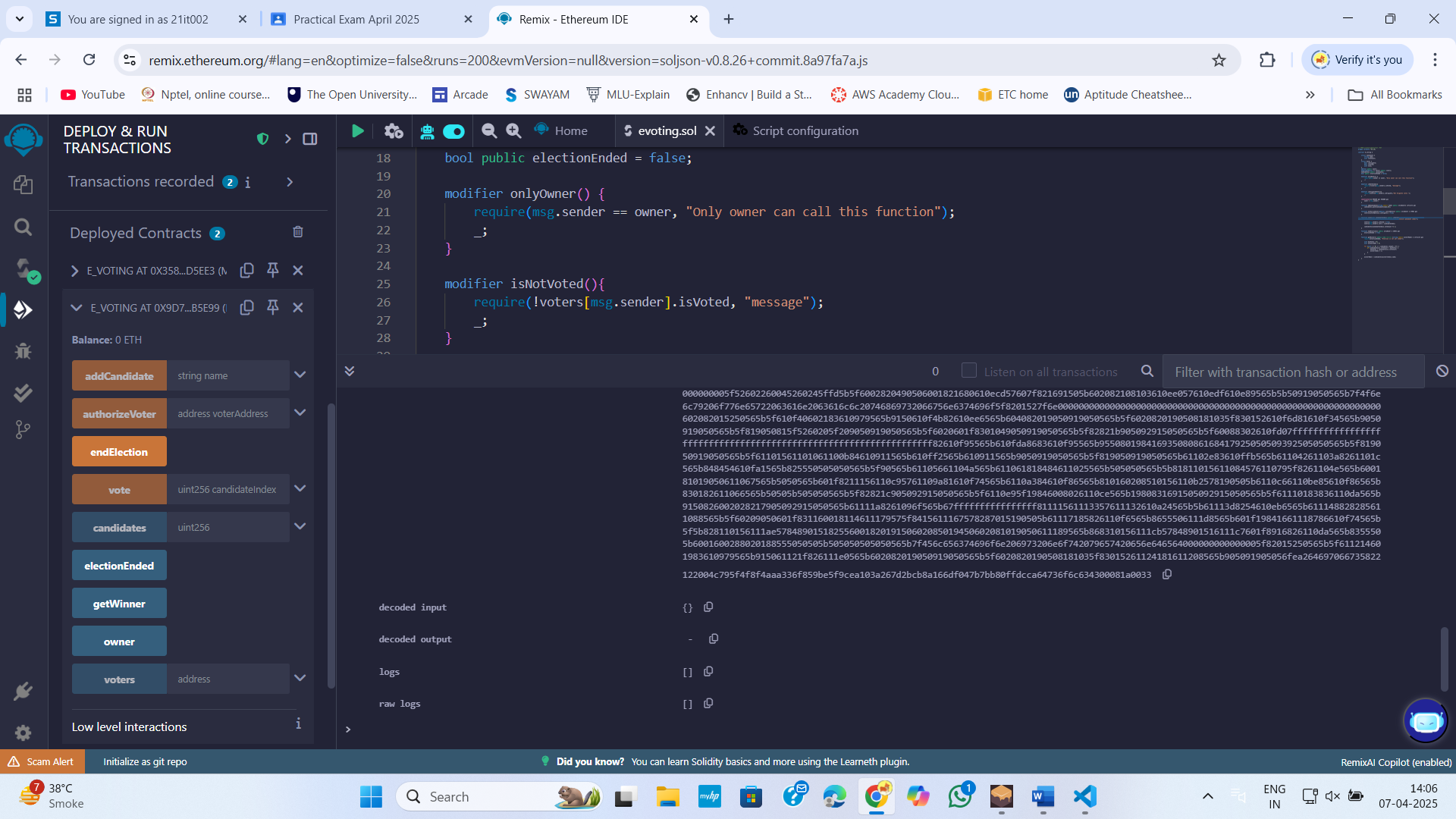


Figure 24:Compile ,Run and Deployed Smart contract



Figure 25:Add user