

DECENTRALISED IDENTITY VERIFICATION

- SUBMITTED BY:
KESHAV SINGHAL (230001039)
KUMAR AYAMAN (230001044)
KUNAL GOURV (230002036)
KOTA SANJAY KUMAR (230001042)
KUMAR PRINCE (230008019)
GIREESH KUMAR REDDY (230005013)

**SUBMITTED TO :
PROF. SUBHRA MAZUMDAR**

TABLE OF CONTENT

BLOCKCHAIN PRESENTATION



01 Smart Contracts

02 Data Structures Used

03 Gas & Memory Efficiency

04 Deployment Details

05 References



OVERVIEW

This project aims to create a decentralized identity verification system. Users register hashed identities on the blockchain, while authorized verifiers confirm these identities.

A Solidity smart contract stores verification statuses immutably. The system prioritizes privacy by hashing sensitive data before sending it to the blockchain.

SMART CONTRACTS

CODING LANGUAGE-SOLIDITY

ROLE-BASED ACCESS CONTROL

- Only verifiers and the contract owner can verify identities.
- The owner can add or remove verifiers.
- Ensures only trusted parties can validate identities.

IDENTITY REGISTRATION

- Users can register their identity by submitting a hashed version of their personal data.
- The hash is stored on-chain to protect user privacy.

IDENTITY VERIFICATION

- Verifiers can validate registered identities.
- Each identity can only be verified once.
- This step confirms that a user's off-chain identity matches their on-chain hash.

SMART CONTRACTS

4. IDENTITY CHECK FUNCTION

Public function allows anyone to check if an identity hash is registered and/or verified.

5. DECENTRALIZED TRUST SYSTEM

- Multiple verifiers can exist.
- No single point of failure.
- Owner can delegate trust to other entities.

```
mapping(address => bool) public verifiers;

function addVerifier(address _verifier) external onlyOwner { ... }
function removeVerifier(address _verifier) external onlyOwner { ... }

modifier onlyVerifier() { ... }
```

DATA STRUCTURES USED

STRUCTURE FOR STORING IDENTITY INFORMATION

Each identity is stored as an instance of the structure *Identity* as defined below:

Field	Datatype	Purpose
IdentityHash	Bytes32	Store the hashed document info
DocumentType	Enum	Store the type of document
isRegistered	Bool	Whether the identity is registered or not
isVerified	Bool	Whether the identity is verified or not

DATA STRUCTURES USED



MAPPING 1: IDENTIFIERS

- It is a public mapping from bytes32 to *Identity*
- It maps the identity hash to the whole *Identity* structure containing the details of that transaction

MAPPING 2: VERIFIERS

- It is a public mapping from address to bool
- It maps each Ethereum account address to a True/False value that is true for authorized verifiers and false otherwise

GAS & MEMORY EFFICIENCY

USE OF FIXED-SIZE HASHES OVER STRINGS

Identity hashes are stored as *bytes32* instead of *string*, hence cheaper to store and compare

EFFICIENT MAPPING USAGE

- mapping(bytes32 → Identity) ensures O(1) lookup (no need for loops)
- apping(address → bool) verifiers used for quick role checks

STRUCT PACKING

The use of structure ensures that each instance is packed within 2 storage slots (32 bytes each) hence saving memory

Field	Datatype	Bytes
IdentityHash	Bytes32	32
DocumentType	Enum	1
isRegistered	Bool	1
isVerified	Bool	1

FRONTEND

01 CLIENT FOLDER CONTAINS FRONTEND PART

1 Register Identity Page

- Users enter personal info (name, ID type/number).
- A keccak256 hash is generated locally for privacy.
- Only the hash is stored on-chain, protecting raw user data.

2 Check Identity Page

- Users can:
- Input details again to regenerate the hash and check status.
- Or directly input an existing hash to check if registered/verified.
- Ensures transparency and self-verification.

3 Verify Identities Page

- For authorized verifiers only.
- Verifiers input a hash to mark the identity as verified on-chain.

🔑 Roles & Access

- Smart contract restricts access:
- Owner: full control.
- Verifier: can verify IDs.
- UI reflects role-based access in real-time.

```
✓ client
  > node_modules
  ✓ public
    <> index.html
  ✓ src
    ✓ components
      JS AdminPanel.js
      JS IdentityChecker.js
      JS RegistrationFor...
      JS VerificationPane...
    ✓ contracts
      {} Context.json
      {} IdentityVerificati...
      {} Ownable.json
      JS App.js
      # index.css
      JS index.js
      {} package-lock.json
      {} package.json
```

WEB3

1. Blockchain Interaction with Smart Contract

- Web3 allows your frontend to communicate with a deployed smart contract through the following functions:
- `web3.eth.getAccounts()` — Gets the current user's wallet address.
- `contract.methods.<functionName>().call()` — Calls read-only methods (no gas).
- `contract.methods.<functionName>().send({ from: account })` — Executes transactions that write to the blockchain (require gas and MetaMask confirmation).

2. Blockchain Interaction with Smart Contract

In both `RegistrationForm` and `IdentityChecker`:

js

Copy Edit

```
web3.utils.soliditySha3({ type: 'string', value: name }, { type: 'string', value: documentId })
```

This ensures:

- ✓ No raw personal data is stored on-chain.
- ✓ Identity is linked via hash only, preserving user privacy.

3.Roles and Access Control

Admin (AdminPanel)

Adds/removes verifier addresses using:

```
js                                                                    Copy Edit
contract.methods.addVerifier(address).send({ from: account });
contract.methods.removeVerifier(address).send({ from: account });
```

Verifier (VerificationPanel):

- Confirms an identity is valid using:

```
js                                                                    Copy Edit
contract.methods.verifyIdentity(hash).send({ from: account });
```

Only addresses previously added by the admin can call this function successfully.....

4. User Registration (RegistrationForm)

- A user registers their identity using:

```
js Copy Edit  
  
contract.methods.registerIdentity(hash, docType).send({ from: account });
```

- The hash is generated locally and stored immutably on the blockchain.
- The documentType is an enum-like integer (e.g., 0 = Passport).

5. Status Verification (IdentityChecker)

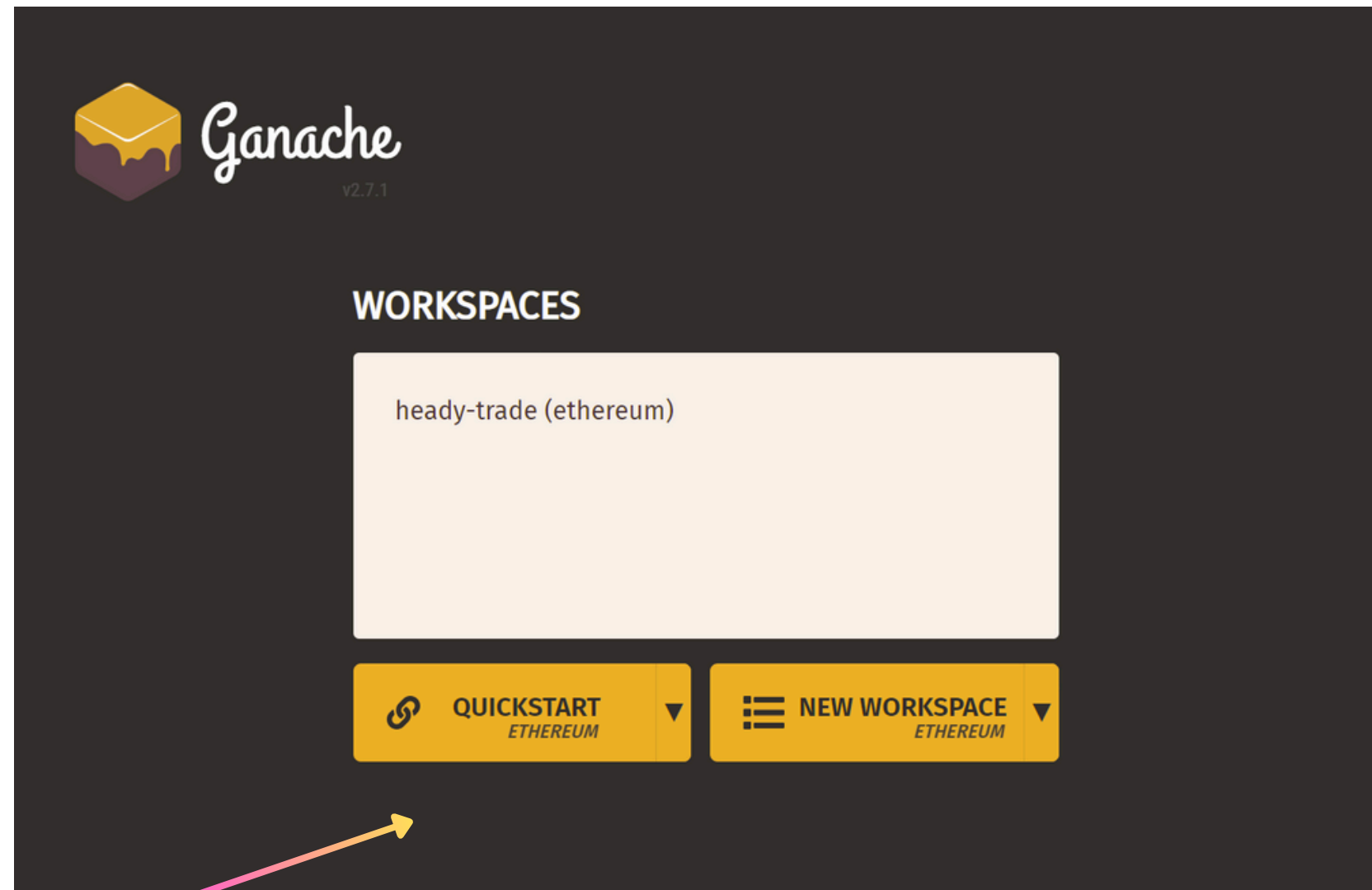
- Any user can check if a hash is:
 - Registered
 - Verified
- Using:

```
js Copy Edit  
  
contract.methods.checkIdentity(hash).call();
```

GANACHE SETUP-

1.Download ganache and install and start in your system









link(<https://archive.trufflesuite.com/ganache/>)










2.Click on quickstart

NETWORK ID

RPC NETWORK

CURRENT BLOCK 0	GAS PRICE 20000000000	GAS LIMIT 6721975	HARDFORK MERGE	NETWORK ID 5777	RPC SERVER HTTP://127.0.0.1:7545	MINING STATUS AUTOMINING	WORKSPACE QUICKSTART	SAVE	SWITCH	
MNEMONIC ? toss toast fever celery salute ordinary analyst govern wrestle lift wash salad								HD PATH m44'60'0'0account_index		
ADDRESS 0xd40d0b586C71438201aE122f1318e8011BE58A91		BALANCE 100.00 ETH		TX COUNT 0		INDEX 0				
ADDRESS 0x8cDCb09c0904E26B17e864A809006253EebF21CE		BALANCE 100.00 ETH		TX COUNT 0		INDEX 1				
ADDRESS 0x76324961fD0a2956a0fa23C5c1eA6A6c7f69A5B8		BALANCE 100.00 ETH		TX COUNT 0		INDEX 2				
ADDRESS 0x80C48F9cB19A9D594F1F373369260c4A609867A4		BALANCE 100.00 ETH		TX COUNT 0		INDEX 3				
ADDRESS 0x3F9c8E7761D27BBC61B644B3ed8c9f76EF054051		BALANCE 100.00 ETH		TX COUNT 0		INDEX 4				
ADDRESS 0x880d73F37213816FBb7882D59aa9EF968C11f951		BALANCE 100.00 ETH		TX COUNT 0		INDEX 5				
ADDRESS 0x2D1Cd7eeEAc57D41ee831C454bEdb5f5BF592A55		BALANCE 100.00 ETH		TX COUNT 0		INDEX 6				

CLICK ON THIS TO GET PRIVATE KEYS

ADDRESS	BALANCE	TX COUNT	INDEX	
0xd40d0b586C71438201aE122f1318e8011BE58A91	100.00 ETH	0	0	
ADDRESS	BALANCE	TX COUNT	INDEX	
0x8cDCb09c0904E26B17e864A809006253EebF21CE	100.00 ETH	0	1	
ADDRESS	BALANCE	TX COUNT	INDEX	
0x76324961fD0a2956a0fa23C5c1eA6A6c7f69A5B8	100.00 ETH	0	2	
ADDRESS	BALANCE	TX COUNT	INDEX	
0x80C48F9cB19A9D594F1F373369260c4A609867A4	100.00 ETH	0	3	
ADDRESS	BALANCE	TX COUNT	INDEX	
0x3F9c8E7761D27BBC61B644B3ed8c9f76EF054051	100.00 ETH	0	4	
ADDRESS	BALANCE	TX COUNT	INDEX	
0x880d73F37213816FBb7882D59aa9EF968C11f951	100.00 ETH	0	5	
ADDRESS	BALANCE	TX COUNT	INDEX	

ACCOUNT INFORMATION

ACCOUNT ADDRESS

0xd40d0b586C71438201aE122f1318e8011BE58A91

PRIVATE KEY

0xd4f978b05a0dabcc136f53d623d31aef718f6c0a4ea1ba9ba283ad933b6b049b

Do not use this private key on a public blockchain; use it for development purposes only!

DONE

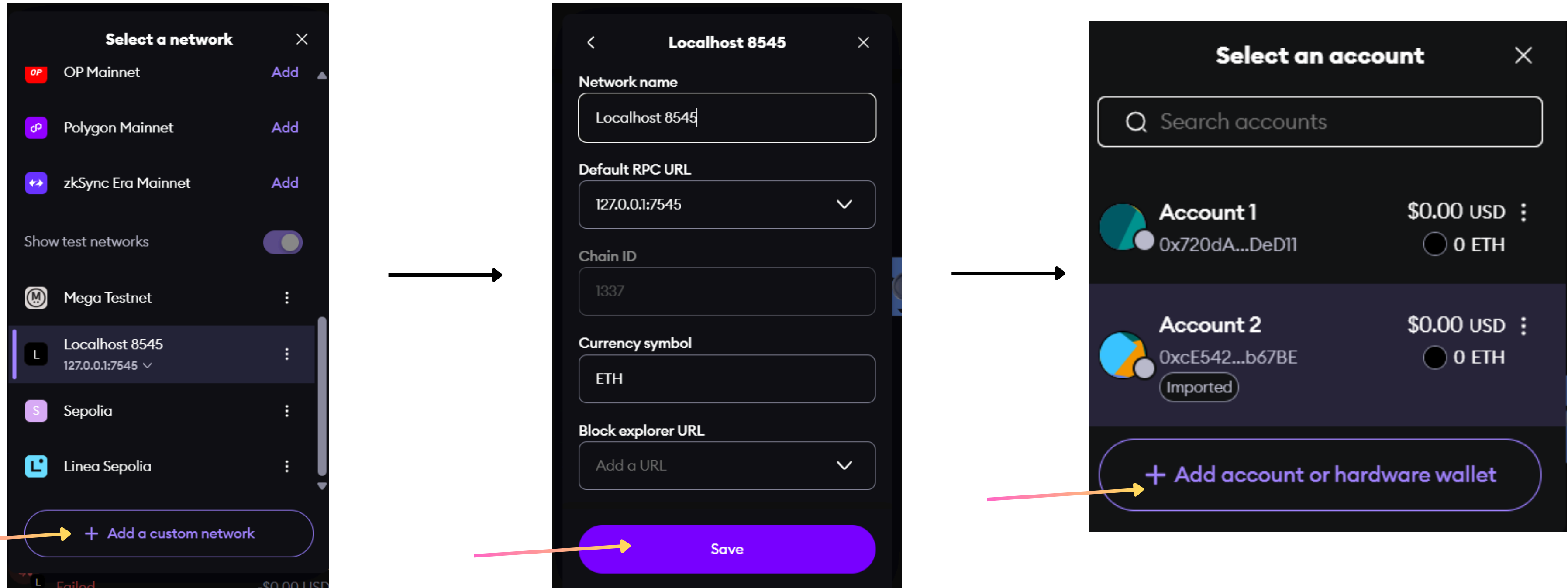
TRUFFLE-

- Truffle is a development framework for building decentralized applications (dApps) on Ethereum.
- Provides tools for writing, testing, deploying, and managing smart contracts.
- Simplifies contract migration, testing, and interaction with the Ethereum blockchain.
- Integrates with Ganache for local blockchain development and supports Web3.js for frontend integration.
- Offers a standardized project structure and automated processes to streamline the development workflow.

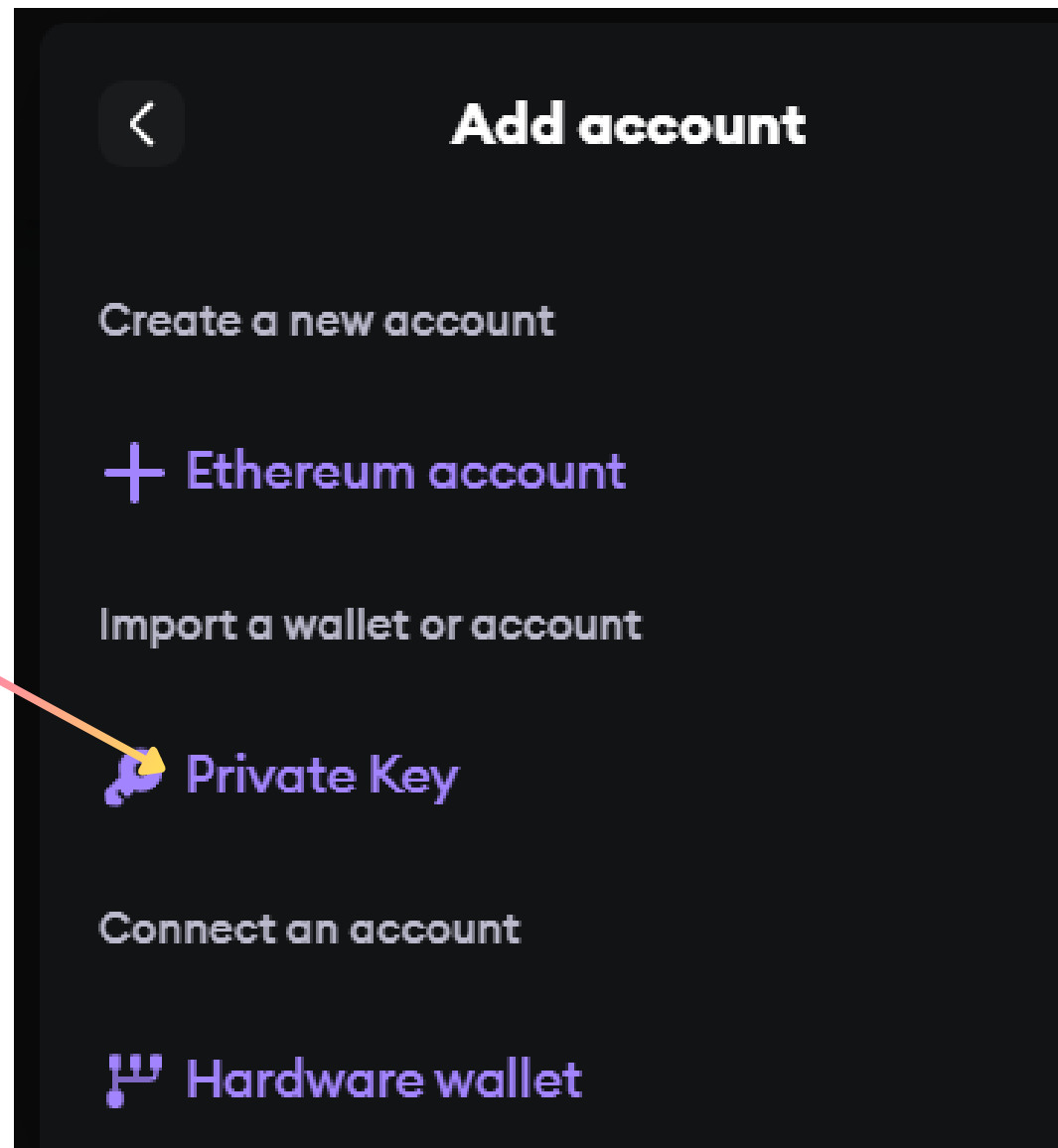
TO RUN TRUFFLE WRITE THIS IN TERMINAL-

- `truffle compile`
- `truffle migrate --reset`

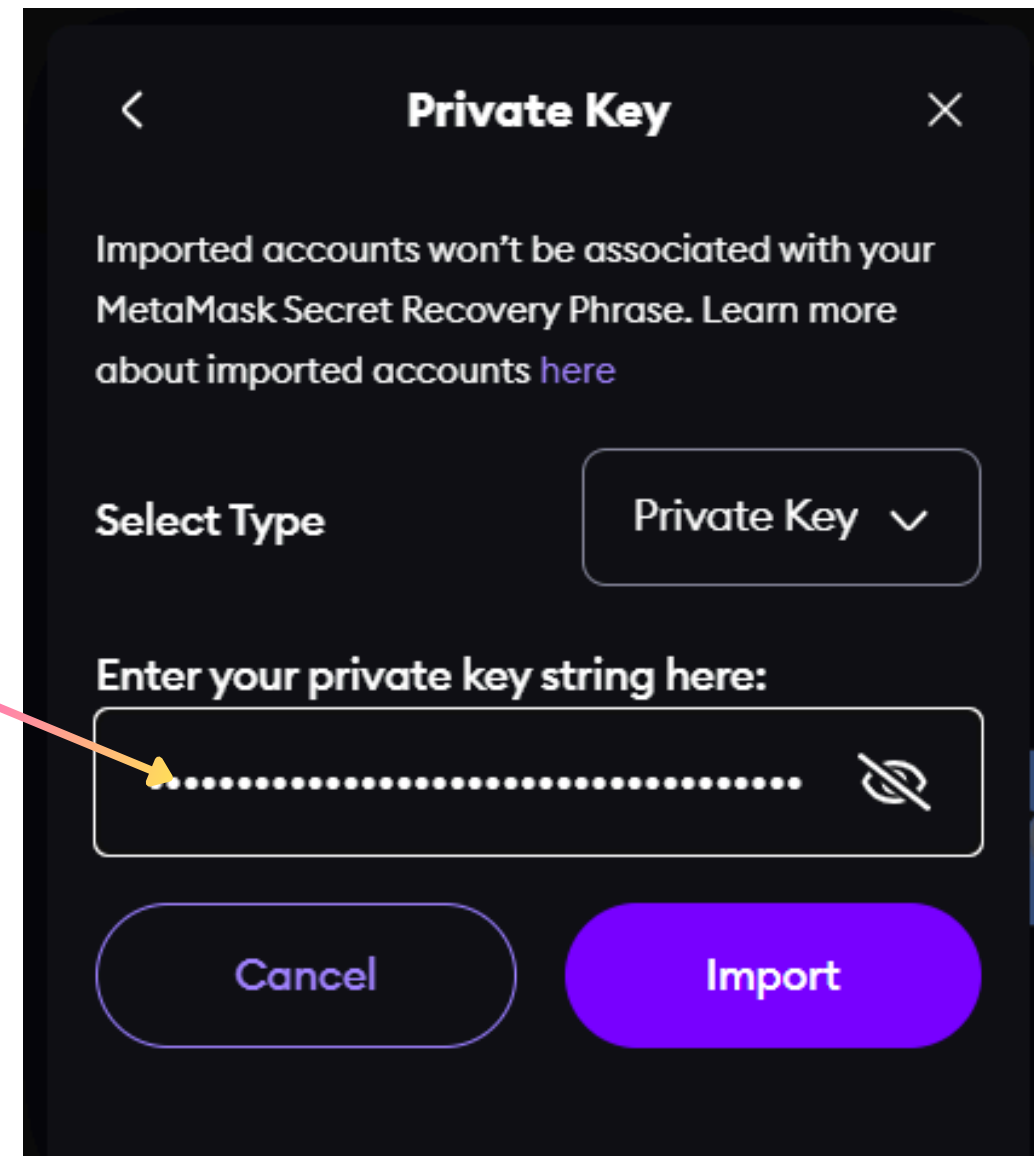
METAMASK CONNECTION



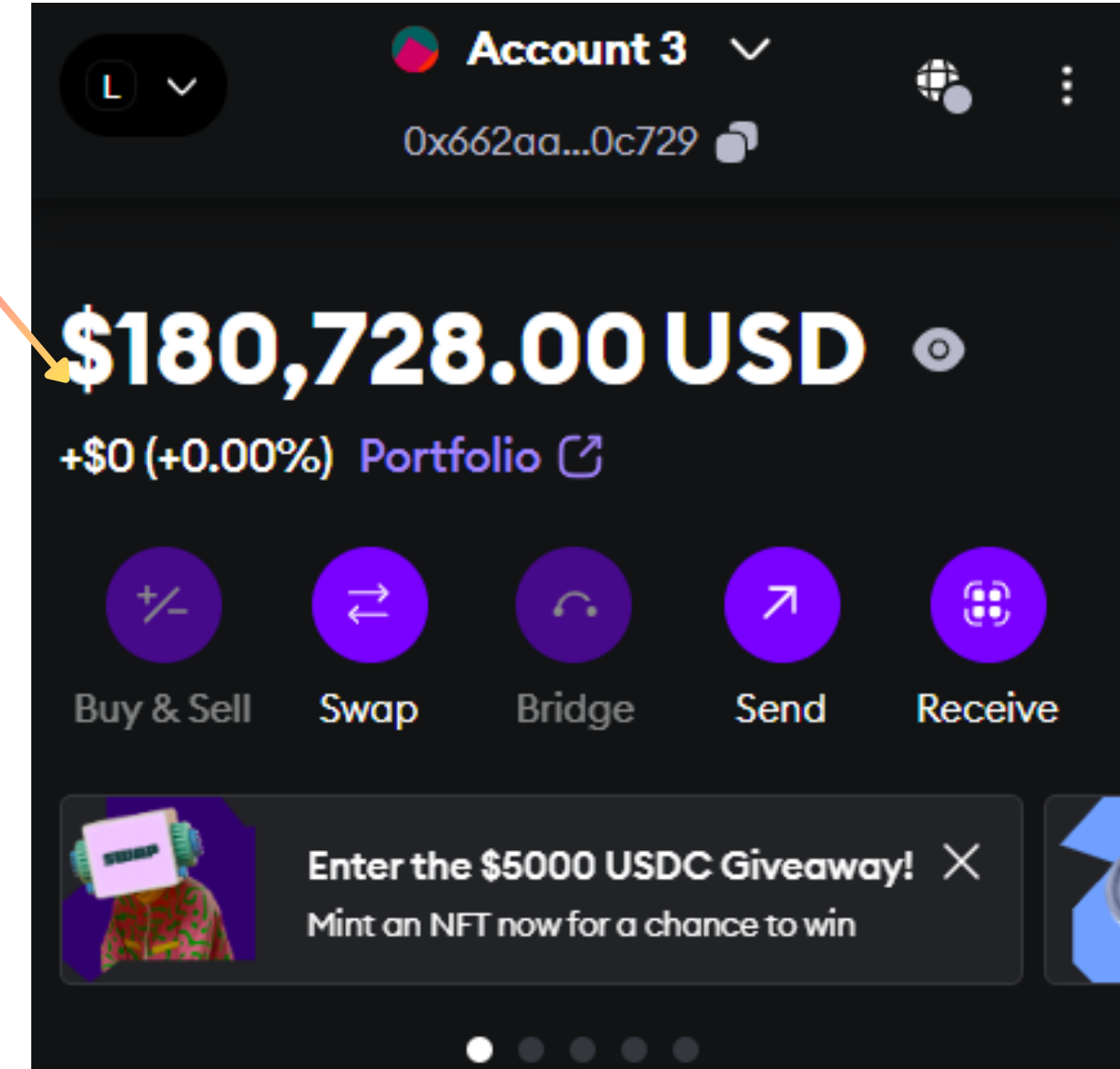
To connect MetaMask to a Ganache instance, you need the RPC network ID, the Ganache server's RPC URL (e.g., <http://localhost:7545>), and the private keys for the accounts generated by Ganache to sign transactions and interact with the blockchain.



click on the private key



enter the private key



ACCOUNT IS ADDED AND BALANCE IS SHOWN

REFERENCES



1) A DOCUMENTATION ON WEB3

web3.js - Ethereum JavaScript API

2) TUTORIAL FOR SOLIDITY PROGRAMMING

Naz Dumanskyy - YouTube

3) AI TOOLS

- ChatGPT
- DeepSeek

THANK YOU

