Decentralized Identity Verification

TEAM: ThirdLevel

ROLL NO	NAME
230001050	**Mani Mathur**
230001069	**Rohan**
230001062	**Parth Agarwal**
230041019	**Maahir Arora**
230041017	**Kushagra Mishra**
230005026	**Pranav Manurakar**

1. Introduction

Decentralized Identity Verification is a decentralized application (dApp) developed using Solidity for the Ethereum blockchain. This project aims to address privacy, security, and compliance challenges by eliminating reliance on centralized authorities and giving users full control over their personal data. The system ensures privacy, data ownership, and verifiable credentials through smart contract-based management.

The platform facilitates:

- **Users** to register their identity and submit verification data.
- **Organizations** to review and verify user identities on-chain.
- **Storage** of only essential verification metadata on-chain, protecting user privacy.
- **Decentralized access** to verified identity status without revealing sensitive data.
- **Revocation and update** of verifications by trusted organizations.
- **Verification status tracking** for individuals and entities.
- **Trust layer creation** between users and verifiers via transparent verification logs.

2. Contract Structure and Key Components

The IdentityVerification contract is written in Solidity (version ^0.8.24) and implements a minimal, on-chain identity registry and verification flow. Its major components are:

2.1 Mappings

2.1.1 Registered:

mapping(bytes32 => uint8) public registered;

Maps an identity hash to a flag (0 = not registered, 1 = registered), ensuring each identity can only be registere once.

2.1.2 isVerified:

mapping(bytes32 => bool) public isVerified;

Tracks whether a registered identity hash has been verified (true) or not (false).

2.2 State Variables:

2.2.1 **verifier** (address):

```
address public verifier;
```

The only address authorized to perform identity verification (e.g., a government body). Set once at deployment

2.3 Events:

2.3.1 IdentityRegistered(bytes32 indexed identityHash) :

Emitted whenever a new identity hash is successfully registered.

2.3.2 IdentityVerified(bytes32 indexed identityHash) :

Emitted when the authorized verifier marks a registered identity as verified

2.4 Constructor :

```
constructor(address _verifier) {

  verifier = _verifier;

}
```

Initializes the contract by setting the verifier address. Only this address can later call verifyIdentity.

3. Contract Functional Overview

3.1 Identity Registration

Function: registerIdentity(bytes32 identityHash)

- Anyone can call this function to register a new identity hash, provided it hasn't been registered before (require(registered[identityHash] == 0)).
- On success, it marks the hash as registered (registered[identityHash] = 1) and emits an IdentityRegistered event

3.2 Identity Verification

Function: *verifyIdentity(bytes32 identityHash*)

- Only the designated verifier address (set in the constructor) can invoke this (require(msg.sender == verifier)).
- It checks that the identity is already registered (registered[identityHash] == 1) and not yet verified (!isVerified[identityHash]).
- Marks the identity as verified (isVerified[identityHash] = true) and emits an IdentityVerified even.

3.3 Status Queries

Solidity's public mappings automatically generate getter functions for both registration and verification status:

- **registered(bytes32)** → returns 0 (not registered) or 1 (registered).
- **isVerified(bytes32)** → returns false (not verified) or true (verified).

4. Gas Optimization:

The **Decentralized Identity Verification** contract implements several optimizations to ensure minimal gas consumption while preserving functionality and security:

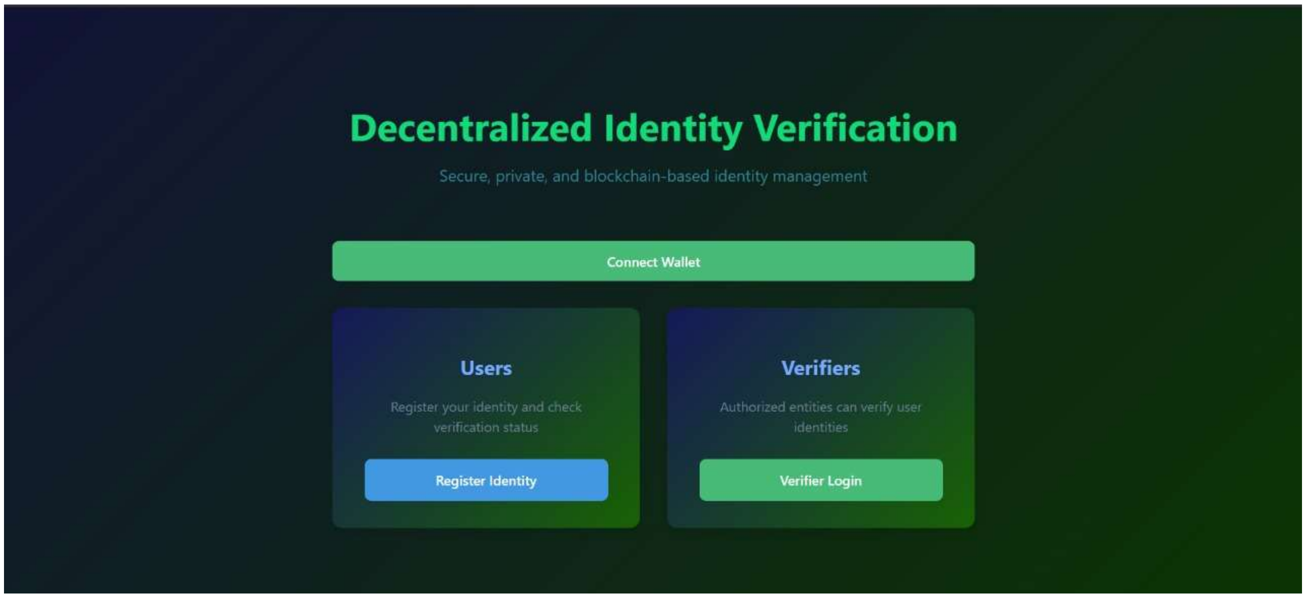
1. **Simple and Packed Storage Layout**
   The contract uses a minimal number of mappings (registered, isVerified) without redundant data structures. This reduces storage slots and improves access efficiency.
2. **Single Responsibility Updates**
   State changes—such as updating verification or registration status—occur only once per identity, and only when necessary. This limits costly SSTORE operations.
3. **Restricted Access and Conditions**
   Both registerIdentity and verifyIdentity functions use precise require statements to short-circuit execution early and prevent unnecessary computation or storage writes.
4. **No Loops or Arrays**
   The contract avoids dynamic arrays and loops altogether, preventing gas costs from scaling with the number of identities registered or verified.
5. **Leverages Public Mappings for Free Getters**
   By declaring mappings as public, Solidity automatically generates getter functions (e.g., registered(identityHash), isVerified(identityHash)), which can be called off-chain with zero gas cost.

5. Security Measures

This application integrates several key security mechanisms to ensure safe on-chain identity handling and prevent misuse:

1. **Role-Based Access Control**
   Only the designated verifier address is allowed to perform identity verifications. This is enforced using a strict require(msg.sender == verifier) check, preventing unauthorized accounts from marking identities as verified.
2. **Input Validation**
   All inputs are rigorously validated. Functions like registerIdentity and verifyIdentity check for logical conditions (e.g., identity not already registered or verified) using require statements to prevent redundant or malicious transactions.
3. **Immutable Verifier Assignment**
   The verifier is assigned once during contract deployment and cannot be changed later, minimizing the risk of privilege escalation or role tampering.
4. **No External Calls / Ether Transfers**
   The contract is purely registry-based and handles no Ether or external contract interactions, effectively eliminating risks such as reentrancy attacks or issues from unsafe Ether transfers.
5. **Privacy by Design**
   Identities are referenced on-chain only via hashed values (bytes32 identityHash), ensuring that no sensitive personal data (e.g., names, emails) is stored or exposed. This supports compliance with privacy best practices and mitigates data leakage risks.

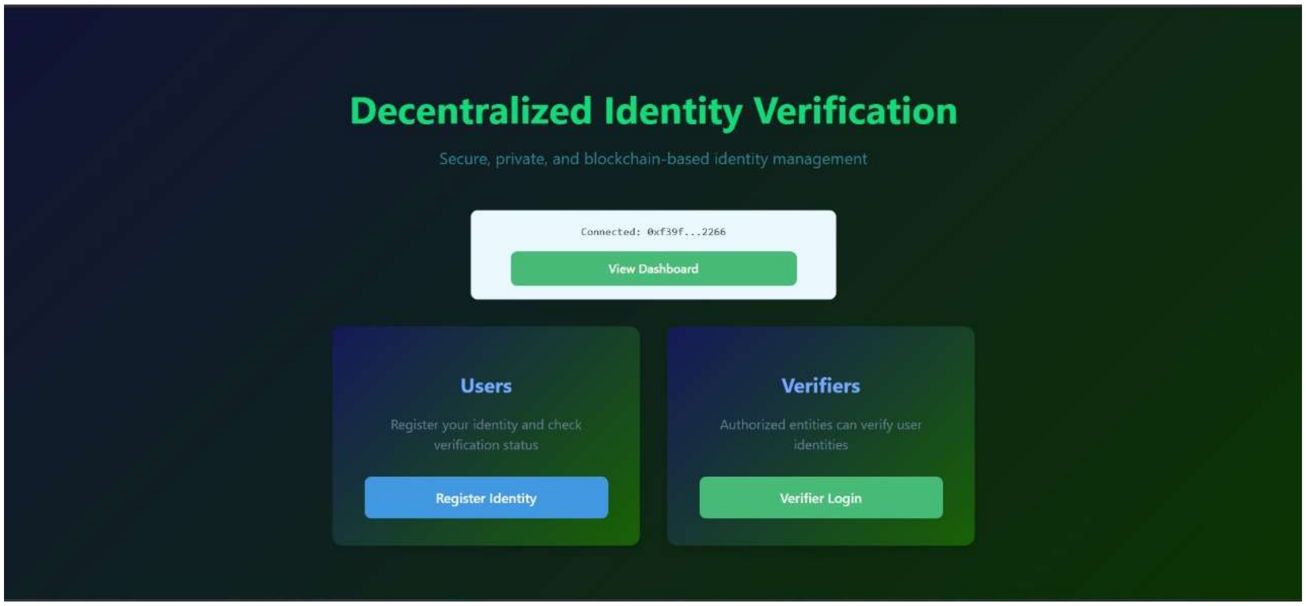
6. How It Works:


Decentralized Identity Verification
Secure, private, and blockchain-based identity management
Connect Wallet
Users
Register your identity and check verification status
Register Identity
Verifiers
Authorized entities can verify user identities
Verifier Login

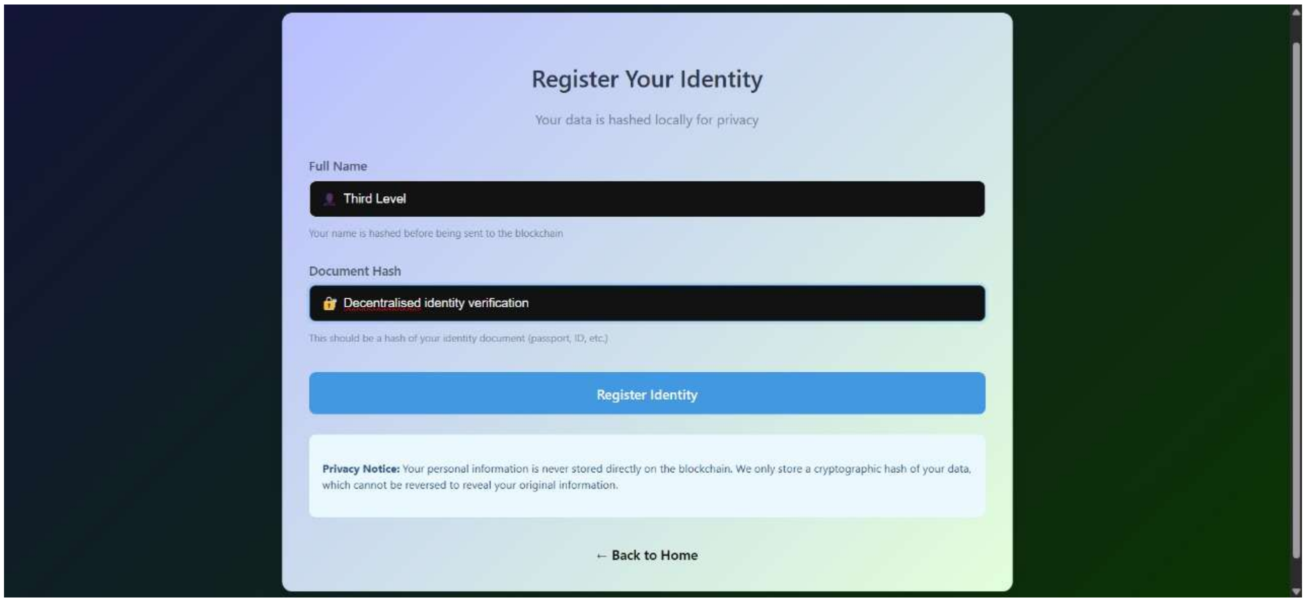
This is the landing page where users must first connect their wallet to begin. It ensures secure access via blockchain credentials. After connecting, users can choose to either:

- Register Identity (for individuals)
- Verifier Login (for authorized verifiers)

This page starts the identity verification process by linking the user to their blockchain wallet.


Decentralized Identity Verification
Secure, private, and blockchain-based identity management
Connected: 0xf39f...2266
View Dashboard
Users
Register your identity and check verification status
Register Identity
Verifiers
Authorized entities can verify user identities
Verifier Login

After connecting their wallet, the user sees their address displayed and can access the dashboard. They can choose to register their identity as a user or log in as a verifier to check others' identities.



Register Your Identity

Your data is hashed locally for privacy

Full Name: Third Level

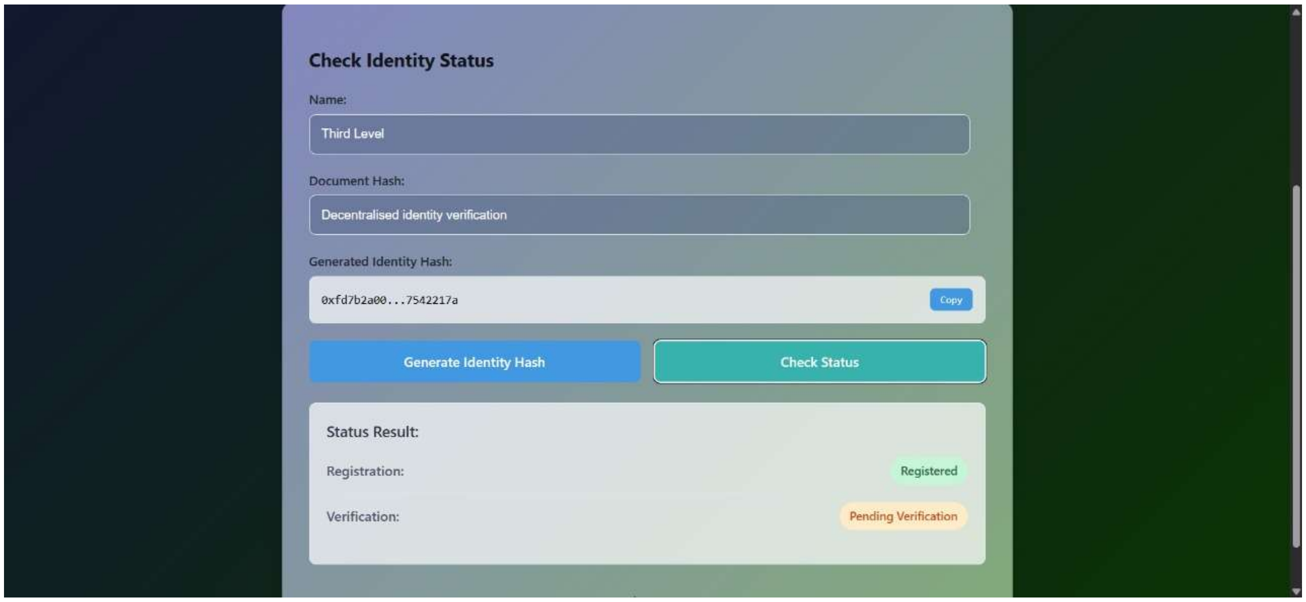
Your name is hashed before being sent to the blockchain

Document Hash: Decentralised identity verification

This should be a hash of your identity document (passport, ID, etc.)

Privacy Notice: Your personal information is never stored directly on the blockchain. We only store a cryptographic hash of your data, which cannot be reversed to reveal your original information.

This is the registration page which lets the user register their identity by entering their full name and a document hash. The data is hashed locally to protect privacy before being sent to the blockchain.



Check Identity Status

Name: Third Level

Document Hash: Decentralised identity verification

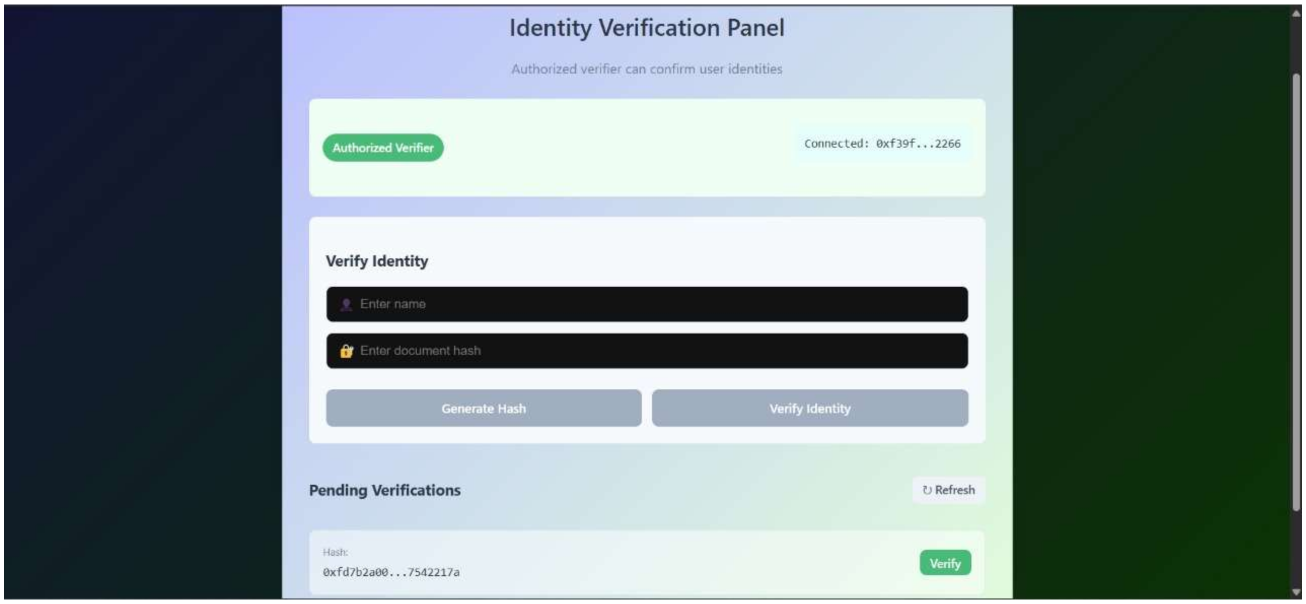
Generated Identity Hash: 0xfd7b2a00...7542217a

Status Result:

Registration: Registered

Verification: Pending Verification

This is the dashboard page which lets the user to check the status of their verification. The users can also generate the identity hash in the dashboard.

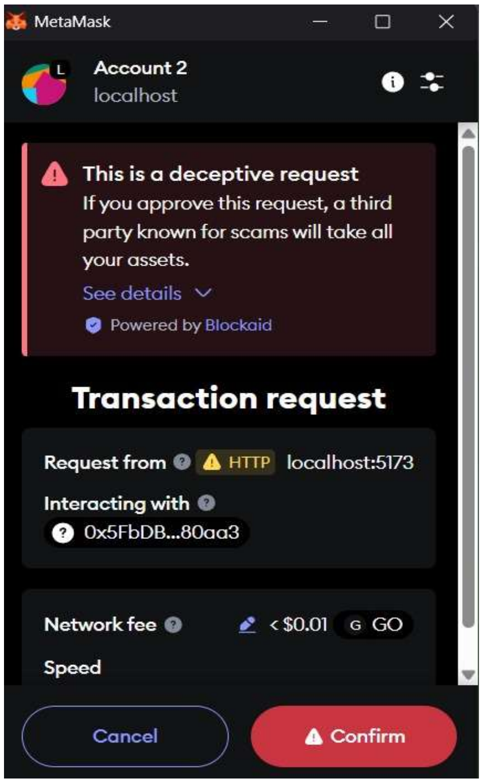

Identity Verification Panel

Authorized verifier can confirm user identities

Connected: 0xf39f...2266

Hash:
0xfd7b2a00...7542217a

Verification Panel
Only authorized verifiers can access this page by clicking **"Verifier Login."** Here, they can view identity submissions and update verification statuses on the blockchain. Additionally, verifiers can prioritize specific users by generating their identity hash using their name and document hash if known, enabling faster processing for high-priority cases.


This is a deceptive request
If you approve this request, a third party known for scams will take all your assets.

Transaction request

Request from HTTP localhost:5173

Interacting with 0x5FbDB...80aa3

Network fee < $0.01 GO

MetaMask is used in this application as the crypto wallet for users and verifiers. It connects to the decentralized app (dApp), lets users register or verify identities, and handles all blockchain transactions securely. Every action like identity registration or verification is confirmed through MetaMask.

Github repo: https://github.com/KMan943/Decentralized_Identity_verification