

# Attendance Marking System

**Title:** *A Decentralized Merkle-Based Attendance marking and verification DApp*

**Author:** Arpan Barik

**Date:** Nov 18, 2025

## Abstract (Summary)

A decentralized application which provides tamper-evident recording and later verification of class/session attendance on Ethereum testnet (Sepolia). Teachers (or third parties) may subsequently verify any attendee's presence without reconstructing full histories on-chain. The architecture leverages: (1) an AttendanceSystem contract (for class/session lifecycle, root storage, and proof verification); (2) an AttendanceToken contract acting as a restricted minter (authorized system only) for attendance badges; (3) a server that collects addresses, builds sorted-pair Merkle trees, pins finalized records to IPFS (Pinata), and exposes proof endpoints; and (4) a React/TypeScript frontend using viem/wagmi for gasless simulation and transaction flow.

## Table of Contents

1. Introduction
  - 1.1 Problem Statement
  - 1.2 Objectives
  - 1.3 Scope and Limitations
2. System Architecture
  - 2.1 Overview
  - 2.2 Component Responsibilities
3. Technology Stack
4. Smart Contract Development
  - 4.1 Design Patterns & Principles
  - 4.2 Core Contracts & Functions
  - 4.3 Security Considerations
  - 4.4 Deployment Process
5. Frontend Implementation
6. Results and Discussion
7. Conclusion and Recommendations
8. References

## 1. Introduction

### 1.1 Problem Statement

Traditional attendance systems depend on centralized databases vulnerable to manipulation, opaque auditing, and single points of failure. Verifying historical attendance for a specific individual or date requires trusting the administrator's records. The challenge: provide an immutable, easily verifiable record of who attended a session—without storing large datasets or personally identifiable information directly on-chain, and enabling independent verification at any later time.

Decentralization uniquely addresses:

- **Integrity:** Merkle root commits the entire set; any alteration of off-chain records invalidates proofs.
- **Transparency:** Anyone can recompute the root from the IPFS JSON and compare to the stored commitment.
- **Self-sovereign claims:** Students claim tokens directly; teachers cannot forge claims on their behalf.
- **Auditability:** A transaction hash on Etherscan becomes a permanent anchor for each finalized session.

### 1.2 Objectives

- Record class sessions and attendee addresses securely.
- Generate deterministic Merkle trees and store only roots + metadata on-chain.
- Pin canonical session JSON to IPFS and link via CID in finalization transactions.
- Provide Merkle proof-based claim flow to mint a non-transferable attendance token.
- Allow off-chain and on-chain verification of any student's attendance.
- Ensure extensibility for per-date session indexing and future privacy-oriented improvements.

### 1.3 Scope and Limitations

#### Scope:

- Single-chain (Sepolia testnet) deployment.
- Attendance tracking by wallet address (no identity abstraction).
- One Merkle tree per finalized class.
- Non-transferable ERC-721 token issuance for claimed attendance.

#### Limitations:

- Privacy: addresses are public.
- Replay safety tied to leaf salts (chainId, contract); porting to new contracts invalidates existing proofs unless roots are migrated.
- No formal third-party audit yet; relies on internal reviews and OpenZeppelin libraries.
- Trust in server for canonical ordering and IPFS pinning (although tampering breaks chain root consistency).
- Single leaf format version; changes require versioned handling.

## 2. System Architecture

### 2.1 Overview

Logical components:

1. **Frontend (React/TypeScript):** Provides teacher dashboard (register, create class, rotate off-chain codes, finalize, verify) and student dashboard (mark attendance, claim token).
2. **Server (Node/Express):** In-memory or persistence layer for temporary attendee capture, Merkle tree construction (sorted pairs), proof generation, IPFS pinning (Pinata), and finalize transaction orchestration.
3. **Smart Contracts (Solidity):**
  - AttendanceSystem: Manages class metadata, finalization (root, total, content hash, CID), Merkle proof verification, and gating logic.
  - AttendanceToken: ERC-721 derivative with restricted minting (soulbound semantics).
4. **IPFS (Pinata):** Stores canonical JSON { chainId, contract, uniqueCode, attendees[], timestamp }.
5. **Blockchain (Ethereum Sepolia):** Persists minimal immutable commitments (root + metadata) and token mint events.

Data Flow (Finalization):

Frontend (Teacher) → Server: request finalize → Server builds Merkle tree → Pin JSON to IPFS → Returns CID, root, total, contentHash → Frontend simulates contract call → MetaMask finalizes → On-chain event logs root + CID → Teacher UI displays root & CID.

Claim Flow:

Student marks attendance → Address added off-chain → Teacher finalizes (root includes address) → Student fetches proof from server → Submit claim → Contract verifies proof → Mints soulbound token.

### 2.2 Component Responsibilities

- **Leaf Construction:** keccak256(abi.encode(chainId, contractAddress, uniqueCode, studentAddress)).
- **Merkle Tree:** Server uses merkletreejs with sortPairs: true.
- **On-chain Storage:** mapping(string => ClassFinalization) or extended structure storing root, total, contentHash, cid.
- **Token Mint:** AttendanceSystem authorized as minter; token ID increments per successful claim.

## 3. Technology Stack

- **Blockchain:** Ethereum Sepolia testnet.
- **Smart Contracts:** Solidity ^0.8.20, OpenZeppelin (ERC721, Ownable, MerkleProof).
- **Development Framework:** Hardhat + Ignition module deployment, Etherscan verification plugin.
- **Frontend:** React, TypeScript, Vite, wagmi/viem for RPC interactions & simulations.
- **Server:** Node.js, Express, Pinata SDK for IPFS pinning, merkletreejs for tree construction.
- **Hashing:** Keccak256 (Solidity intrinsic / viem / merkletreejs).
- **Wallet Integration:** MetaMask.

- **Dependency Control:** npm with lock file; environment variables for chainId, contract address, Pinata JWT.

## 4. Smart Contract Development

### 4.1 Design Patterns & Principles

- **Separation of Concerns:** AttendanceSystem handles logic; AttendanceToken purely stores mintable attendance badges (SBT style).
- **Access Control:** Only registered teachers can finalize; only system (minter) can call token mint.
- **Minimal On-chain Data:** Root, counts, content hash, CID — no full arrays.
- **Deterministic Salting:** Include block.chainid + address(this) + uniqueCode to prevent cross-contract replay.
- **Event Emission:** ClassFinalizedWithCID logs parameters for Etherscan transparency.
- **Soulbound Pattern:** Override transfer hooks or omit exposed transfer functions (or revert on transfer attempts) to keep tokens non-transferable.

### 4.2 Core Contracts & Functions

AttendanceSystem:

- registerTeacher(): Marks msg.sender eligible for class creation/finalization.
- createClass(string uniqueCode, timestamp ...): Initializes class record.
- finalizeClassWithCID(string uniqueCode, bytes32 root, uint32 total, bytes32 contentHash, string cid): Stores Merkle root & metadata; emits event.
- verifyAttendance(string uniqueCode, address student, bytes32[] proof) view returns (bool): Recomputes leaf; verifies proof.
- claim(string uniqueCode, bytes32[] proof): Validates proof and mints attendance token if not already claimed.
- getFinalization(string uniqueCode) view: Returns root, total, contentHash, cid (and optionally timestamp).

AttendanceToken:

- setMinter(address system) onlyOwner: Authorizes AttendanceSystem.
- mintAttendance(address to, string uniqueCode): Called only by authorized system; mints new token ID.
- Override/guard transfers: Enforce soulbound semantics (implementation-dependent).

Merkle Proof:

- Uses sorted pair hashing; server ensures ordering, contract relies on OpenZeppelin MerkleProof.verify.

### 4.3 Security Considerations

- **Reentrancy:** Not critical (no external calls during state changes except safe ERC-721 mint); still could apply ReentrancyGuard if extended.
- **Input Validation:** Reject zero roots, empty CID, duplicate class codes.

- **Replay Protection:** Leaf salts incorporate chain and contract address.
- **Authorization:** Teacher registration gating; minter restriction in token contract.
- **Proof Correctness:** Sorted pairs requirement documented; mismatch triggers Invalid proof revert.
- **Gas Efficiency:** Store fixed-size data; avoid arrays. Single SSTORE per field on finalization.
- **Upgradability Risks:** Absent; migrations require manual data bridging.
- **External Libraries:** OpenZeppelin audited components reduce implementation attack surface.

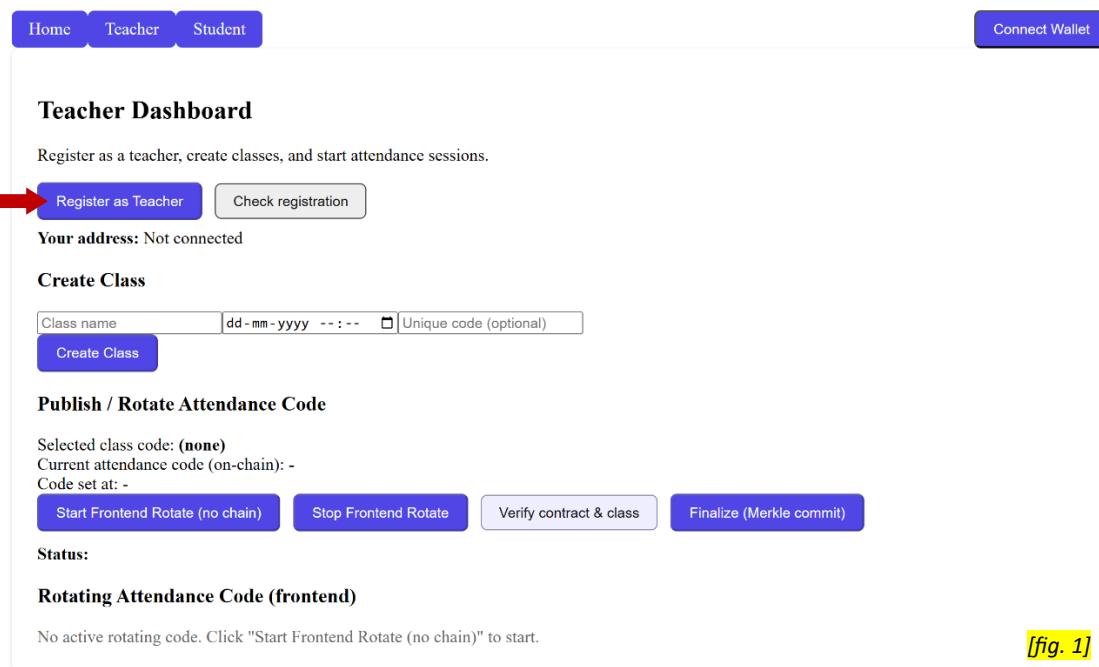
#### 4.4 Deployment Process

- **Compilation:** npx hardhat compile.
- **Ignition Deployment:** npx hardhat ignition deploy [AttendanceSystem.ts](http://\_vscodecontentref\_/0) --network sepolia --deployment-id AttendanceDeployment-v2.
- **Verification:** npx hardhat verify --network sepolia <AttendanceSystemAddress> after flatten or artifact availability.
- **Address Propagation:** Update frontend contracts.ts and server env CONTRACT\_ADDRESS, restart processes.
- **Pinata Setup:** Provide PINATA\_JWT in .env; server pins JSON and obtains CID prior to finalization.

#### 5. Frontend Implementation

The interface is split into Teacher and Student tabs. Teacher workflow:

1. Register as teacher (button (see fig. 1) triggers transaction; status reflected (see fig. 2)).



## Teacher Dashboard

Register as a teacher, create classes, and start attendance sessions.

Your address: 0x2d3c8aAda7EC0f595B087D0074fB6A307d91E1b6

Note: This address is already registered as a teacher.

### Create Class

Class name	dd-mm-yyyy	-- : --	<input type="checkbox"/>	Unique code (optional)
<input type="button" value="Create Class"/>				

### Publish / Rotate Attendance Code

Selected class code: (none)

Current attendance code (on-chain): -

Code set at: -

Status:

#### Rotating Attendance Code (frontend)

No active rotating code. Click "Start Frontend Rotate (no chain)" to start.

[fig. 2]

2.Create class (inputs: course code, optional unique session code, date/time); (see fig. 3).

## Teacher Dashboard

Register as a teacher, create classes, and start attendance sessions.

Your address: 0x2d3c8aAda7EC0f595B087D0074fB6A307d91E1b6

Note: This address is already registered as a teacher.

### Create Class

Class name	dd-mm-yyyy	-- : --	<input type="checkbox"/>	Unique code (optional)
<input type="button" value="Create Class"/>				

### Publish / Rotate Attendance Code

Selected class code: (none)

Current attendance code (on-chain): -

Code set at: -

Status:

#### Rotating Attendance Code (frontend)

No active rotating code. Click "Start Frontend Rotate (no chain)" to start.

[fig. 3]

\*[Click 'Create Class' and confirm transaction (see fig. 4)]

The screenshot shows the Teacher Dashboard interface. At the top, there are three tabs: Home, Teacher (which is selected), and Student. On the right, it shows the address 0x2d3c...E1b6. Below the tabs, the title "Teacher Dashboard" is displayed. A note says "Register as a teacher, create classes, and start attendance sessions." There are three buttons: "Registered" (highlighted with a red arrow), "Check registration", and "Raw Inspect". Below these is a text field showing "Your address: 0x2d3c8aAda7EC0f595B087D0074fB6A307d91E1b6" and a note "Note: This address is already registered as a teacher." Under the "Create Class" section, there is a form with fields "CSET212", "19-11-2025 22:59", and "Unique code (optional)". A blue button labeled "Create Class" is highlighted with a red arrow. To the right, a MetaMask window titled "Transaction request" is open, showing the transaction details: "Estimated changes No changes", "Network Sepolia", "Request from HTTP localhost:5173", "Interacting with Alert > 0x05A0e...9AaCA", "Network fee 0.0003 s SepoliaETH", and two buttons "Cancel" and "Confirm". A yellow box labeled "[fig. 4]" is positioned at the bottom right.

\*[A unique class code is displayed for this particular class (see fig. 5)]

This screenshot shows the Teacher Dashboard after a class has been created. The "Create Class" section now displays a unique class code "CLS91swywmc" in the "Unique code (optional)" field. The "Create Class" button is highlighted with a red box. The "Publish / Rotate Attendance Code" section shows the selected class code "CLS91swywmc" highlighted in a red box. The "Status" field shows the transaction hash: "Transaction submitted: 0x197c287e1c056d9c28ccf60e1e456b3f7837c0224ce3cf514b2909e6b1e9e26f". A yellow box labeled "[fig. 5]" is positioned at the bottom right.

3. Start off-chain rotation (see fig. 6);(local ephemeral code updates for marking).

Teacher Dashboard

Register as a teacher, create classes, and start attendance sessions.

Registered Check registration Raw Inspect

Your address: 0x2d3c8aAda7EC0f595B087D0074fB6A307d91E1b6  
Note: This address is already registered as a teacher.

Create Class

CSET212	19 - 11 - 2025 22 : 59	CLS91swywmc
Create Class		

**Publish / Rotate Attendance Code**

Selected class code: **CLS91swywmc**  
Current attendance code (on-chain): -  
Code set at: -

**Start Frontend Rotate (no chain)** Stop Frontend Rotate Verify contract & class Finalize (Merkle commit)

**Status:** Transaction submitted: 0x197c287e1c056d9c28ccf60e1e456b3f7837c0224ce3cf514b2909e6b1e9e26f

**Rotating Attendance Code (frontend)**

No active rotating code. Click "Start Frontend Rotate (no chain)" to start.

[fig. 6]

\*[This 6-digit alpha-numerical code (see fig. 7: A) is projected on the class/session screen]

Teacher Dashboard

Register as a teacher, create classes, and start attendance sessions.

Registered Check registration Raw Inspect

Your address: 0x2d3c8aAda7EC0f595B087D0074fB6A307d91E1b6  
Note: This address is already registered as a teacher.

Create Class

CSET212	19 - 11 - 2025 22 : 59	CLS91swywmc
Create Class		

**Publish / Rotate Attendance Code**

Selected class code: **CLS91swywmc**  
Current attendance code (on-chain): -  
Code set at: -

**Start Frontend Rotate (no chain)** **Stop Frontend Rotate** Verify contract & class **Finalize (Merkle commit)**

**Status:** Starting frontend rotation (no chain)... B C

**Rotating Attendance Code (frontend)**

A 81DQHY

[fig. 7]

\*[All present attendees will see the rotating Attendance code being projected and mark their attendance on the student page (student page workflow discussed below)]

\*[After all attendees have marked their attendances, teacher can 'Stop Frontend Rotate' (see fig. 7: B)]

4. Finalize: triggers server finalize endpoint; obtains CID/root; simulates contract; prompts MetaMask (see fig. 8); on success displays root + total + CID link (see fig. 9).

The screenshot shows the Teacher Dashboard interface. At the top, there are three tabs: Home, Teacher (which is selected), and Student. A blue header bar on the right displays the address "0x2d3c...E1b6". Below the tabs, the dashboard has sections for "Teacher Dashboard", "Create Class", and "Publish / Rotate Attendance Code". In the "Create Class" section, a class code "CSET212" is entered, along with a date and time "19 - 11 - 2025 22 : 59" and an attendance code "CLS91swywmc". A "Create Class" button is present. In the "Publish / Rotate Attendance Code" section, a note says "Selected class code: CLS91swywmc". Below this are buttons for "Start Frontend Rotate (no chain)", "Stop Frontend Rotate", "Verify contract & class", and "Finalize (Merkle commit)". A status message "Status: Submitting finalize tx... attendees=1" is shown. To the right of the dashboard, a MetaMask extension window titled "Transaction request" is open, showing details about the transaction being submitted. The transaction is estimated to have "No changes" and is interacting with the "Sepolia" network via "localhost:5173". The "Finalize (Merkle commit)" button on the dashboard corresponds to the "Confirm" button in the MetaMask dialog.

[fig. 8]

This screenshot shows the Teacher Dashboard after the finalize operation. The layout is identical to fig. 8, with tabs for Home, Teacher (selected), and Student, and a header address bar "0x2d3c...E1b6". The "Create Class" and "Publish / Rotate Attendance Code" sections are visible. In the "Publish / Rotate Attendance Code" section, the status message now reads "Status: Finalized: root=0xa9b1f19c3e72ed7fa5093543ac2c0d9b0ad3394be879231630889cb996cbd99e total=1 | cid=QmPEkJrjhewLacLGfvYPfrG4wHd6Gg2mh5HWqgaZSbGU". This message is highlighted with a red box. Below the status message, the "Rotating Attendance Code (frontend)" section is present with the same instructions as fig. 8.

[fig. 9]

Student workflow:

1. Enter class code after the teacher has created a class; mark attendance (see fig. 10); (server collects address).

The screenshot shows a student dashboard interface. At the top, there are three tabs: Home, Teacher, and Student, with 'Student' being the active tab. In the top right corner, there is a blue button with the text '0x3649...722f'. Below the tabs, the title 'Student Dashboard' is displayed. A sub-instruction 'Join a class and mark your attendance.' is present. A text input field labeled 'Enter class unique code' contains the value '0x36491EFEFE2842F6d50f94622883b19Af8B8722f'. A blue button labeled 'Mark Attendance' is visible. Below this, a status message 'Connected: 0x36491EFEFE2842F6d50f94622883b19Af8B8722f' is shown. A section titled 'Live Attendance' follows. Under 'Live Attendance', the instruction 'Pick the currently projected code' is given. A note says 'Click "Mark Attendance" to start viewing rotating codes.' A blue button labeled 'Claim on-chain (after finalization)' is present. The status 'Status:' is followed by '[fig. 10]'.

- \*[Choose the correct code among the four (the one being projected in that instant by the teacher); (see fig. 11)]

The screenshot shows a student dashboard interface, identical to Fig. 10 but with a red rectangular box highlighting the 'Pick the currently projected code' section. This section contains four blue buttons with the text '1CPUU6', 'MM30HK', 'Q5VT6N', and 'IIUM9X'. Below this section is a blue button labeled 'Claim on-chain (after finalization)'. The status 'Status: Attendance marked!' is shown at the bottom. To the right of the status message is '[fig. 11]'. The top navigation bar, connected status, and live attendance section are also visible.

2. After finalization (on teacher's side), fetch Merkle proof from server via claim button (see fig. 12); simulate claim; confirm transaction (see fig. 13); token minted and visible in wallet (see fig. 14), and on etherscan (see fig. 15).

[Home](#)
[Teacher](#)
[Student](#)
0x3649...722f

## Student Dashboard

Join a class and mark your attendance.

Mark Attendance

**Connected:** 0x36491EFEFE2842F6d50f94622883b19Af8B8722f

### Live Attendance

**Live code:** -

Pick the currently projected code

→
Claim on-chain (after finalization)

**Status:** Attendance marked!

[fig. 12]

The screenshot shows a student dashboard interface and a MetaMask wallet extension dialog.

**Student Dashboard:**

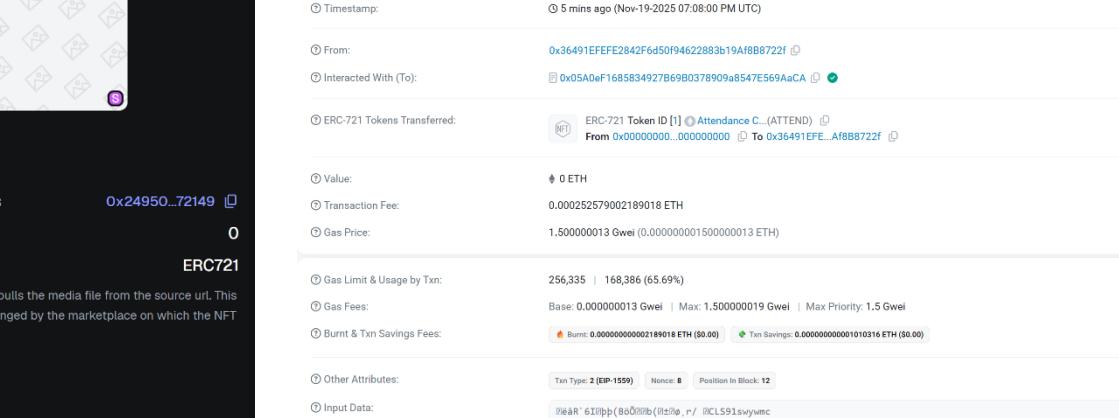
- Header: Home, Teacher, Student
- Title: Student Dashboard
- Text: Join a class and mark your attendance.
- Input field: CLS91swywmc
- Button: Mark Attendance
- Text: Connected: 0x36491EFEFE2842F6d50f94622883b19Af8B8722f
- Section: Live Attendance
- Text: Live code: -
- Text: Pick the currently projected code
- Button: Claim on-chain (after finalization)
- Status: Submitting claim tx...

**MetaMask Transaction Request Dialog:**

- Header: MetaMask Account 3
- Title: Transaction request
- Text: Estimated changes ⓘ
- Text: You receive + #1 📡 Attendance
- Text: Network: Sepolia
- Text: Request from: △ HTTP localhost:5173
- Text: Interacting with: Alert > 0x05A0e...9AaCA
- Buttons: Cancel, Confirm

[fig. 13]

[fig. 13]



The screenshot shows the MetaMask interface displaying a transaction for an NFT. The transaction details are as follows:

- Contract address: 0x24950...72149
- Token ID: 0
- Token standard: ERC721
- Disclaimer: MetaMask pulls the media file from the source url. This url sometimes gets changed by the marketplace on which the NFT was minted.
- Transaction Hash: 0xa4fdb7168da95797702d7892304b4d542d9a0c8d59479bdac96a1e56e52de7a4
- Status: Success
- Block: 9663719 | 26 Block Confirmations
- Timestamp: 5 mins ago (Nov 19 2025 07:08:00 PM UTC)
- From: 0x36491EFEFE2842F6d50f94622883b19Af8B8722f
- Interacted With (To): 0x05A0ef1685834927B69B0378909a8547E569AaCA
- ERC-721 Tokens Transferred:
  - Value: 0 ETH
  - Transaction Fee: 0.000252579002189018 ETH
  - Gas Price: 1,500,000 Gwei (0.0000000150000013 ETH)
- Gas Limit & Usage by Txn: 256,335 | 168,386 (65.69%)
- Gas Fees:
  - Base: 0.000000013 Gwei | Max: 1.500000019 Gwei | Max Priority: 1.5 Gwei
  - Burnt & Txn Savings Fees:
    - Burnt: 0.00000000002189018 ETH (\$0.00)
    - Txn Savings: 0.00000000001010316 ETH (\$0.00)
- Other Attributes: Txn Type: 2 (EIP-1559) | Nonce: 8 | Position In Block: 12
- Input Data: 0xaar`6I@pb(8b0@#b@#p,r/ @CL591svyvmc

[fig. 14] [fig. 15]

[fig. 15]

### 3. Displayed status message.

The screenshot shows a web-based student dashboard. At the top, there are three tabs: 'Home', 'Teacher', and 'Student'. The 'Student' tab is active. In the top right corner, there is a purple button with the text '0x3649...722f'. Below the tabs, the title 'Student Dashboard' is displayed in bold. A sub-instruction 'Join a class and mark your attendance.' is present. A text input field contains the value 'CLS91swywmc'. Below it is a blue button labeled 'Mark Attendance'. To the right, a status message 'Connected: 0x36491EFEFE2842F6d50f94622883b19Af8B8722f' is shown. Under the heading 'Live Attendance', there is a section for 'Live code: -' and a note 'Pick the currently projected code'. A blue button labeled 'Claim on-chain (after finalization)' is visible. At the bottom, a red-bordered box displays the status 'Status: Claimed on-chain!'. The entire interface has a light gray background with rounded corners.

## 6. Results and Discussion

### Functional Outcomes

- Successful class creation, single-attendee and multi-attendee finalization (root correctness confirmed manually).
- Accurate Merkle proof acceptance with empty proof for single-leaf sessions.
- AttendanceToken minted only through authorized system; transfer attempts blocked (soulbound behavior).
- Etherscan transaction displays decoded input (root, total, contentHash, CID); event log retrievable.

### Verification Robustness

- Recomputed roots from IPFS JSON matched on-chain roots—demonstrating integrity of off-chain record binding.
- “Invalid proof” surfaced correctly during mismatch (e.g., stale contract address or wrong salt) enabling rapid debugging.

### Performance

- Gas usage per finalization within a modest range; dominated by storage writes (root, counts, hashes).
- Claim operations low-cost (Merkle verification over short proof arrays;  $O(\log n)$  scaling).

### Problem Fit

The application satisfies the initial need for immutable attendance verification: teacher cannot retrospectively alter an already committed root without creating a new transaction; any tampering with the IPFS JSON invalidates contentHash and Merkle root consistency.

## **Limitations Observed**

- Reliance on centralized server for ordering leaves (although detectable via root mismatch).
- Lack of automated batch testing across many attendees.

## **7. Conclusion and Recommendations**

### **Conclusion**

The DApp demonstrates a practical, gas-efficient pattern for decentralized attendance verification using Merkle commitments and off-chain storage. The system effectively minimizes on-chain data while preserving auditability and enabling self-sovereign token claims for participants. The integration of IPFS CIDs enhances transparency, allowing independent reconstruction and validation of attendance sets.

## **8. References**

1. OpenZeppelin Contracts v5.x Documentation – <https://docs.openzeppelin.com/contracts>
2. Ethereum Yellow Paper – <https://ethereum.github.io/yellowpaper/>
3. Hardhat Framework – <https://hardhat.org>
4. Wagmi / Viem Libraries – <https://wagmi.sh> / <https://viem.sh>
5. Merkle Trees Explained (Ethereum Blog) – <https://blog.ethereum.org> (Merkle fundamentals)
6. Pinata IPFS Service – <https://pinata.cloud>
7. Etherscan API & Verification Guides – <https://docs.etherscan.io>
8. ERC-721 Standard – <https://eips.ethereum.org/EIPS/eip-721>
9. Soulbound Token Concept – “Decentralized Society: Finding Web3’s Soul” (Buterin et al.)
10. merkletreejs Library – <https://github.com/miguelmota/merkletreejs>