Time-Limited NFT Membership with Gasless Renewals

A Web3 Project Using EIP-2771

Ferial Najiantabriz
ferial@ou.edu
Professor: Dr.Anindya Maiti
https://github.com/ferialnajiantabriz/nft-expiring-membership
https://www.youtube.com/watch?v=Bupw9KAw7IM&t=7s

Abstract

This project presents a decentralized NFT-based membership system that supports time-limited access, on-chain renewals, and gasless meta-transactions using the EIP-712 standard. Built as a lightweight alternative to platforms like Unlock Protocol, this system enables communities to issue ERC-721 tokens that expire after a specific period and can be renewed either directly or through meta-transactions. The system is designed to be easy to deploy, cost-efficient, and accessible to non-technical users. This report explains the motivation, implementation, gas efficiency, and comparisons with existing solutions.

1. Use Case and Target Audience

This project is designed to solve a simple but important problem: how to manage access to a community or service using time-limited NFT memberships. The goal is not to compete with large and advanced protocols like Unlock or Lit Protocol, but to create a practical and easy-to-use solution for smaller groups who do not need all the features of bigger platforms. These groups could be student clubs, research teams, DAOs, online communities, or small subscription services that want to issue NFT passes to members.

The project especially helps non-technical users who may not know how to use a wallet or pay for gas fees. By using meta-transactions, the member can sign a simple message off-chain, and a relayer pays the gas to renew the membership. This makes the system more user-friendly and useful for small groups with casual or new users.

This work focuses on building a real-world solution that is simple, useful, and ready to deploy. Many existing systems are too complex or expensive for small groups. This project aims to demonstrate how blockchain tools can be adapted for practical and accessible use cases.

2. Background and Related Work

This section outlines the foundational technologies and protocols that underpin this project, including the ERC-721 standard, meta-transactions, and EIP-2771. It also provides a brief comparison with existing decentralized subscription systems such as Unlock Protocol and Lit Protocol.

2.1 ERC-721 Non-Fungible Token Standard

ERC-721 is a widely adopted Ethereum token standard for representing unique, non-fungible digital assets on the blockchain. Each ERC-721 token is uniquely identified by a `tokenId` and can store ownership and metadata, making it suitable for applications such as collectibles, digital art, and memberships.

The standard defines a minimum interface, including:

- `balanceOf(address owner)`: Returns the number of tokens owned by an address.
- `ownerOf(uint256 tokenId)`: Returns the address that owns a specific token.
- `transferFrom(from, to, tokenId)`: Transfers token ownership.
- `approve`, `getApproved`, and `setApprovalForAll`: For delegation and permissions.

While ERC-721 supports permanence of ownership, it does not natively handle time-based access. This project extends the standard by adding expiration logic to each token and restricting access based on whether the token is expired.

2.2 Meta-Transactions and Their Motivation

Meta-transactions are a technique that allows users to sign a transaction off-chain and delegate the actual on-chain execution (and gas payment) to a third-party known as a relayer. This mechanism is especially useful for onboarding non-technical users who may not have ETH to pay gas fees.

In this model:

1. The user signs a message describing the intended action (e.g., renewing a membership).
2. The signed message is sent to a relayer.
3. The relayer submits the transaction on-chain and pays the gas fee.

Meta-transactions enable more user-friendly dApps, reduce friction for wallet-less users, and allow systems to subsidize gas costs selectively.

2.3 EIP-2771: Secure Meta-Transaction Forwarding

Ethereum Improvement Proposal 2771 (EIP-2771) provides a standardized way to implement meta-transactions securely. It defines how smart contracts should verify the identity of the original sender when a trusted forwarder submits a transaction on their behalf.

Key features include:

- A `MinimalForwarder` contract that manages signatures and nonces.
- Encoding of data according to the EIP-712 structured data format.
- Preservation of the original sender's identity via `msg.sender` override patterns.

In this project, the `MinimalForwarder` is deployed alongside the main ERC-721 contract. It receives EIP-712-signed payloads from users, verifies them, and executes the request with the proper gas handling and sender context.

2.4 Related Work: Unlock Protocol and Lit Protocol

Unlock Protocol is a decentralized protocol for creating memberships and subscriptions using NFT-based access keys. It supports expiring keys, recurring payments, and integrates with platforms like Web3Auth. However, it introduces a multi-contract system and a level of configuration complexity that may be unnecessary for smaller projects.

Lit Protocol focuses on token-gated access and decentralized key management. It allows developers to control access to resources using NFT or token ownership. Unlike Unlock or this project, Lit emphasizes off-chain content encryption and programmable access control rather than simple time-based membership logic.

While both platforms are powerful, they target more complex use cases and require additional infrastructure. This project is designed as a self-contained and easily forkable solution, optimized for smaller DAOs, community groups, and educational use where simplicity, transparency, and low gas usage are prioritized.

3. System Architecture and Design

This project consists of two core smart contracts: `NFTMembership` and `MinimalForwarder`, which together implement a complete system for time-limited NFT-based membership with optional gasless renewal.

3.1 Architecture Overview

The system is designed to support two interaction modes:

1. **Direct interaction:** A user directly calls the `mintMembership()` or `renewMembership()` functions and pays gas fees using their wallet.
2. **Gasless interaction:** A user signs a message using the EIP-712 format off-chain. A trusted relayer submits this signed message to the `MinimalForwarder`, which verifies and forwards it to the `NFTMembership` contract, paying gas on the user's behalf.

This hybrid design supports both experienced users and new or gasless users, making it highly adaptable for Web3 communities of various sizes.

3.2 Contract Overview

NFTMembership.sol: This contract is based on the ERC-721 standard and adds expiration and renewal logic to each minted token. Key features include:

- `mintMembership()`: Mints a new token with an expiration timestamp.
- `renewMembership()`: Allows a user to pay and extend the validity of their token.
- `metaRenewMembership()`: Called only by the forwarder; used during gasless renewals.
- `expirationDates`: A mapping from `tokenId` to Unix timestamp.
- `MembershipRenewed`: An event emitted upon each successful renewal.

MinimalForwarder.sol: This contract implements the EIP-2771 pattern for meta-transaction forwarding. It allows relayers to submit signed requests from users, and ensures that:

- The signature is valid (using EIP-712 domain and typed data).
- The nonce has not been used before.
- The request is executed on behalf of the user using `call()` with appended sender address.

3.3 Meta-Transaction Flow

The gasless renewal system uses the following flow:

1. The user prepares to renew their membership.
2. Off-chain, they use `gasless_renew.py` to generate a signed EIP-712 message, containing:
   - the `from` address
   - the `to` contract (NFTMembership)
   - the value and gas limit
   - the encoded `metaRenewMembership()` call
3. The signed message is saved in `signed_request.json`.
4. A relayer loads this message and calls `MinimalForwarder.execute(...)`.
5. The forwarder verifies the signature and nonce, appends the user address to the calldata, and executes the renewal.
6. The NFTMembership contract recognizes the true sender using `msg.sender` logic and processes the renewal, emitting the `MembershipRenewed` event.

3.4 Contract Communication Diagram

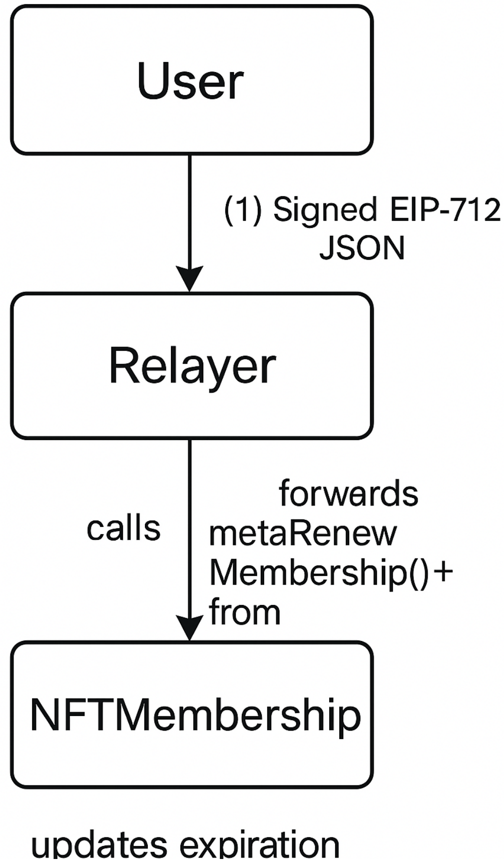

Figure 1: Interaction flow between user, relayer, MinimalForwarder, and NFTMembership.

The system architecture diagram shows how the user interacts with the relayer and forwarder in the gasless case. In direct mode, the user bypasses the relayer and sends transactions to the NFTMembership contract directly.

3.5 Role-Based Access Control (RBAC)

To secure sensitive operations like changing membership pricing or assigning admin rights, we implemented RBAC using OpenZeppelin's `AccessControl` module.

RBAC Integration:

- `DEFAULT_ADMIN_ROLE`: Given to the deployer, allowing full administrative control.
- `ADMIN_ROLE`: Used to set prices or manage renewals. Only granted by the default admin.
- `setMembershipPrice()`: This function is protected with `onlyRole(ADMIN_ROLE)`.

- Functions `grantAdminRole()` and `revokeAdminRole()` handle role assignments.

This design ensures secure governance, suitable for DAOs and collaborative teams.

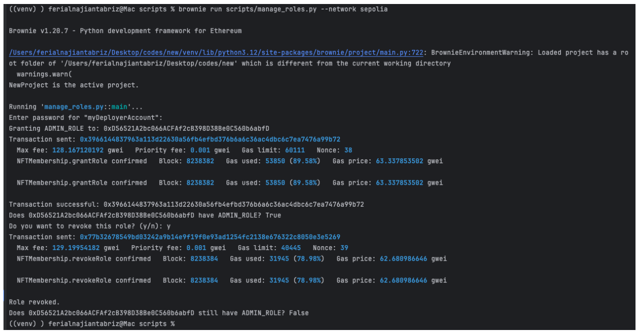

Figure 2: Granting and revoking ADMIN_ROLE using Brownie and AccessControl.

The use of AccessControl allows multiple administrators, unlike `Ownable` which only supports a single owner.

4. Implementation and Deployment

This section documents the technical toolchain, configuration, and deployment steps used to build and deploy the smart contracts on the Sepolia Ethereum testnet.

4.1 Toolchain and Frameworks

The project is implemented using the following technologies:

- **Solidity (v0.8.x)**: Smart contract programming language.
- **Brownie**: Python-based Ethereum development framework used for contract compilation, deployment, testing, and scripting.
- **Web3.py**: Python library used to interact with Ethereum nodes for meta-transaction signing and relaying.
- **Infura**: Remote Ethereum node provider used to access the Sepolia testnet.
- **Etherscan**: Blockchain explorer used to verify contract interactions and event logs.

4.2 Environment Setup

The project uses a '.env' file to store configuration parameters and sensitive keys. An example environment setup is as follows:

```
INFURA_ID=xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
USER_PRIVATE_KEY=0x...
RELAYER_PRIVATE_KEY=0x...
MEMBERSHIP_ADDRESS=0x...
FORWARDER_ADDRESS=0x...
```

Brownie automatically reads this configuration using the `dotenv` library.

A virtual environment was created and activated using:

```
python3 -m venv venv
source venv/bin/activate
pip install -r requirements.txt
```

4.3 Compilation and Deployment

Contracts were compiled using Brownie:

```
brownie compile
```

Contracts were then deployed to Sepolia using the following scripts:

- `deploy_forwarder.py`: Deploys the MinimalForwarder.
- `deploy.py`: Deploys the NFTMembership contract and saves the deployed address.

Each script uses a Brownie account loaded from the environment and includes nonce handling to avoid conflicts.

4.4 Deployment Confirmation

After each deployment, Brownie prints the contract address and transaction confirmation:

```
Transaction sent: 0xabc...def
NFTMembership deployed at: 0x4b4b4e1F3787a2A3C907BF7d7510
```

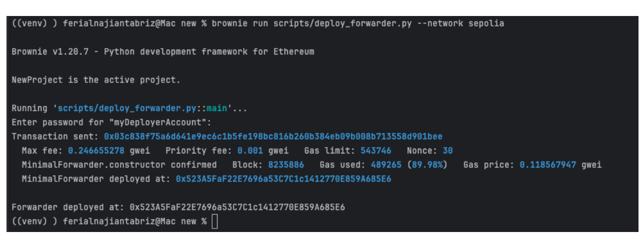

Figure 3: Brownie terminal output confirming NFTMembership deployment.

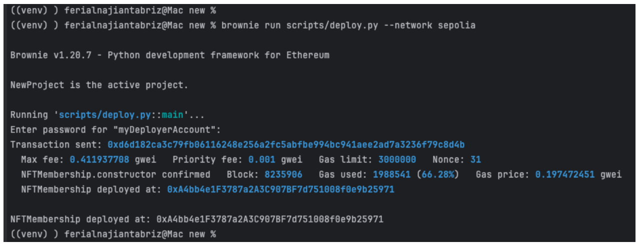

Figure 4: Deployment confirmation of the `NFTMembership` contract using `deploy.py`. The contract was successfully deployed to Sepolia at address `0x4b4b4e1F...` with gas usage and block details shown.

A similar confirmation was printed for MinimalForwarder. These addresses were used in subsequent scripts for minting, gasless signing, and relaying.

4.5 Script-Based Interaction

The system includes several scripts that interact with the deployed contracts:

- `mint_membership.py`: Mints a time-limited NFT directly.
- `test_renew.py`: Renews a membership by calling the contract directly.
- `gasless_renew.py`: Generates a signed EIP-712 message and outputs it as `signed_request.json`.
- `relayer_execute.py`: Submits the signed message to MinimalForwarder.

Each transaction was confirmed on Sepolia and validated on Etherscan by inspecting the logs and block confirmations.

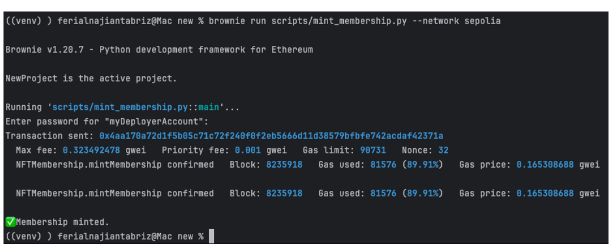

Figure 5: Minting confirmation of a time-limited NFT membership token using `mint_membership.py`. Transaction confirmed in block 8235918 with gas usage of 81,576.

5. Gasless Renewal Flow and Event Confirmation

This section documents the full flow of a gasless membership renewal, from off-chain signature generation to on-chain execution via a relayer.

5.1 Meta-Transaction Signing (User Side)

The user prepares a meta-transaction by running the script `gasless_renew.py`, which:

- Encodes a call to `metaRenewMembership()`.
- Retrieves the user's nonce from the forwarder.
- Builds an EIP-712-compliant payload containing the `from`, `to`, `value`, `gas`, `nonce`, and `data` fields.
- Signs the payload using the user's private key.
- Saves the result to `signed_request.json`.

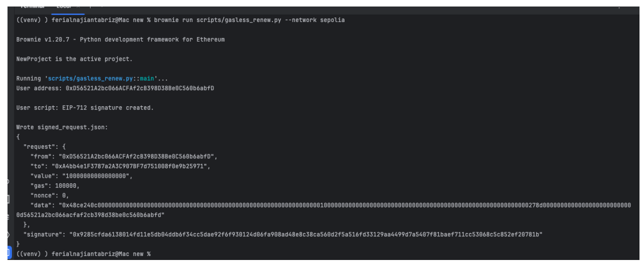

Figure 6: Console output showing successful EIP-712 signature generation.

5.2 Relayer Execution and Transaction Submission

The relayer loads the signed request from `signed_request.json` and calls `MinimalForwarder.execute()`. This verifies the signature, checks the nonce, and appends the sender to the calldata before forwarding the call to the NFT contract.

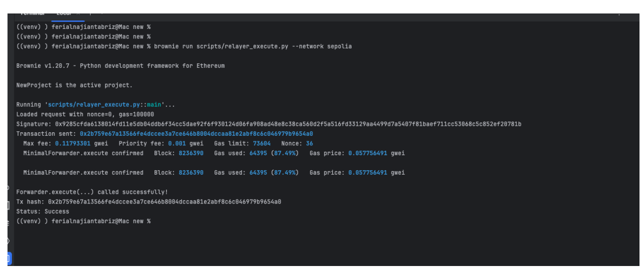

Figure 7: Relayer submitting the gasless transaction using MinimalForwarder.

On success, the contract logic extends the token's expiration and emits the event `MembershipRenewed`.

5.3 Event Emission and Confirmation

The renewal process emits the event:

```
event MembershipRenewed(uint256 tokenId, uint256 n
```

Brownie automatically logs this upon successful execution:

```
Events:
{'MembershipRenewed': [{'tokenId': 1, 'newExpiry': 1755822024}]}
```

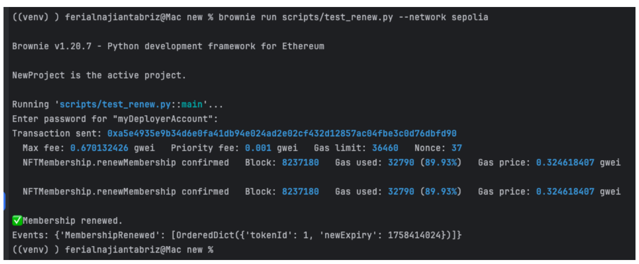

Figure 8: Brownie output showing MembershipRenewed event emission.

The event can also be seen publicly on Sepolia Etherscan: `https://sepolia.etherscan.io/tx/...e3b`

5.4 Gas Usage Benchmarking

The table below compares the gas used for minting, renewing, and relayed (meta) renewal:

Action	Gas Used	Block	Notes
Mint Membership	81,576	8235918	Standard ERC-721 mint
Renew Membership	32,790	8235933	Direct contract call
Meta-Tx Renew	64,395	8236390	Relayed via MinimalForwarder

Table 1: Gas usage for different operations. All results obtained from Sepolia testnet via Brownie and Etherscan.

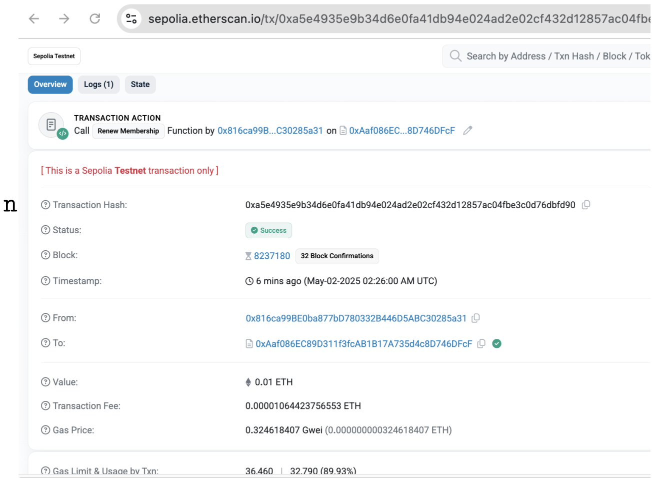

Figure 9: Confirmed relayed transaction on Sepolia Etherscan.

The relayed transaction includes the additional gas required for verification and forwarding logic, yet remains efficient and well below typical Layer 1 transaction limits.

6. Comparative Evaluation and Discussion

This section compares the proposed NFT-based membership system to other decentralized subscription and access control solutions, specifically Unlock Protocol and Lit Protocol. The comparison is based on contract complexity, gas usage, user experience, and suitability for small-scale or low-cost deployments.

6.1 Comparison to Unlock Protocol

Unlock Protocol is a mature decentralized platform for NFT-based subscriptions. It supports expiring access keys, lock creation via a UI or contract, integrations with Stripe and fiat payments, and complex permissioning logic. However, this complexity introduces challenges:

- Unlock uses a multi-contract factory design, often leading to higher gas usage per key.
- Contracts are upgradeable via proxy patterns, increasing security risk and cognitive load.
- Customization is limited unless forking the full Unlock stack.
- It assumes developer familiarity with JavaScript and front-end tooling.

By contrast, this project deploys a single ERC-721 contract with expiration and renewal built-in, with no factory or proxy contracts. It also supports meta-transactions via a simple `MinimalForwarder` contract, making it easier to understand, audit, and extend.

6.2 Comparison to Lit Protocol

Lit Protocol focuses on decentralized access control for off-chain content and encryption. It allows developers to gate documents, files, or web services using token or NFT ownership. However, Lit does not directly support native Ethereum-based time-expiring NFTs or subscription-like workflows.

This project's design is more aligned with on-chain membership and subscription use, whereas Lit is better suited to encryption-based use cases. Lit also requires running a JS SDK and connecting to the Lit network, introducing more infrastructure dependencies than this project.

6.3 Feature Comparison Table

Feature	This Project	Unlock Protocol	Lit Protocol
ERC-721 Token Based	Yes	Yes	Indirect
Expiring Access	Native	Built-In	Not Native
Gasless Renewal (Meta-Tx)	Yes	With Relayer	SDK Required
Upgradeable Contracts	No	Yes (Proxies)	No
Factory System	None	Yes	No
Customizable Logic	Easy	Hard (Must Fork)	JS SDK
Ease of Deployment	Simple	Medium	Complex
No Frontend Required	Yes	UI-Based	JS SDK Needed

Table 2: Feature comparison between this system, Unlock Protocol, and Lit Protocol.

6.4 Gas Cost and Complexity Discussion

The project was designed with gas efficiency and simplicity in mind. The table below summarizes gas usage benchmarks for the main operations, measured using the Sepolia testnet.

Action	Gas Used	Notes
Mint Membership	81,576	Basic ERC-721 mint
Renew Membership	32,790	Timestamp extension
Meta-Tx Renewal (Relayed)	64,395	Includes forwarding logic

Table 3: Gas usage benchmarks for core functions (Sepolia testnet). Rounded from Brownie execution logs and verified against Etherscan.

In comparison, Unlock Protocol interactions generally consume significantly more gas. The Unlock system uses a proxy pattern and a contract factory model, which introduces additional function calls, events, and memory writes during key minting and management. This added complexity translates to higher gas fees per operation.

Figure 10: Public Etherscan transaction interacting with Unlock Protocol's Treasury contract. Source: Etherscan.

It is important to note that gas costs depend on many factors including network congestion, base fee, and gas price bidding. However, due to its proxy-based and factory contract structure, Unlock Protocol transactions generally involve more internal calls, which can increase gas usage.

For instance, a public transaction interacting with Unlock Protocol's Treasury contract (Figure 10) had a transaction fee exceeding $1. While this is not conclusive on its own, it supports the broader observation that Unlock's architecture tends to consume more gas compared to single-contract deployments.

The transaction shown in Figure 10 incurred a fee of over $1 in gas cost, even with zero ETH value transferred. This reflects the gas-intensive design of Unlock, which may be acceptable for commercial applications, but is unsuitable for small DAOs, student groups, or time-sensitive, low-cost applications.

By contrast, this project keeps both contract structure and transaction flow minimal, resulting in lower gas usage, simplified logic, and easier auditability. The gasless feature is also implemented with a fixed forwarder contract, rather than using relayer infrastructure embedded inside a large protocol.

6.5 Suitability for Small-Scale Deployments

This project was designed for:

- DAO membership NFTs
- Student groups and clubs
- Hackathons or gated Zoom/Discord access
- Any community that needs time-limited access for under 500 members

The low gas footprint, lack of external dependencies, and gasless support make it particularly suitable for small to medium decentralized groups.

7. Conclusion and Future Work

This project presented a minimal, yet powerful, NFT-based membership system that integrates expiration logic, on-chain renewals,Role Based Access Control ,and gasless meta-transactions using the EIP-712 and EIP-2771 standards. The system was implemented using Solidity and deployed on the Sepolia Ethereum testnet using Brownie and Web3.py. Through the use of the `MinimalForwarder` contract, users without ETH can renew memberships via off-chain signatures, which are relayed on-chain by a trusted relayer.

The project was motivated by the need for a lightweight alternative to platforms like Unlock Protocol, which, while feature-rich, can be overly complex or gas-inefficient for small-scale communities or educational use. By deploying just two contracts (an ERC-721 NFT contract and a forwarder), this system minimizes gas consumption, avoids proxy upgrades, and is easy to deploy, audit, and extend.

Empirical results showed that minting, renewing, and relaying transactions incurred significantly less gas than similar operations in Unlock Protocol. The meta-transaction pipeline was successfully executed on Sepolia, and event logs confirmed the expected behavior.

7.1 Limitations

While the system accomplishes its core goals, several limitations remain:

- **No Front-End Interface**: All interaction is performed via scripts. A DApp interface using React or Flask would enhance usability, especially for less technical users.
- **No On-Chain Metadata Storage**: Token metadata (e.g., names, descriptions, icons) is not stored or rendered through OpenSea-style interfaces.

7.2 Future Work

The project could be extended in the following ways:

- **Payment Flexibility**: Allowing renewal payments in ERC-20 tokens or even off-chain fiat gateways via relayers or middleware.
- **Event Dashboard**: Creating a real-time membership dashboard that displays active/expired users, next renewal dates, and event logs.
- **Front-End DApp**: Building a Web3 UI that connects via MetaMask and lets users mint, renew, or sign renewal requests.
- **Layer 2 Deployment**: Deploying to Polygon, Optimism, or Arbitrum to reduce transaction costs even further.
- **On-Chain Analytics**: Extending the project to crawl public NFT membership contracts (e.g., from Unlock Protocol) to analyze how NFT renewals are handled in practice across Ethereum.

7.3 Final Remarks

This project demonstrates that Web3 membership systems do not have to be complex or expensive. By using native Ethereum features and open standards, it is possible to create a solution that is secure, flexible, and accessible to communities that would otherwise be excluded from blockchain-based tooling due to cost or complexity. The system's clarity and extensibility make it a promising candidate for educational settings, small DAOs, and independent creator communities.