

# Proxy Contract for Load Balancing and Function Delegation

Developed By :

Anjali Khandelwal

February 24, 2024

## 1 Introduction

This document provides a detailed overview of the design and implementation of a proxy smart contract that acts as a load balancer for other contracts. The proxy contract serves as the entry point for all requests, delegating them to specific implementation contracts based on the function requirement. The primary objectives of this project are to design and implement a proxy contract that maintains a registry of implementation addresses for different functionalities, implement a fallback function for function delegation, and ensure efficient and secure function delegation.

Following are the objectives of the project:

**Proxy Contract Design:** This section outlines the design of the proxy contract, including the creation of a registry of implementation addresses for different functionalities and the implementation of a fallback function.

**Function Delegation:** The document explains how the proxy contract forwards requests to the appropriate implementation contract and supports updates to implementation addresses without disrupting the proxy contract or requiring redeployment.

**Security and Efficiency:** The document discusses the incorporation of security measures to prevent unauthorized updates to the contract registry and optimize for gas efficiency and minimize execution costs for delegated functions.

Additionally, the document includes the requirements, points to consider, and testing strategies to ensure correct function delegation and execution.

## 2 Design Decisions

### 2.1 Architecture

**Contract Architecture:** The project utilizes a proxy contract design pattern to act as a load balancer for other contracts. It maintains a registry of implementation addresses

for different functionalities such as token transfers, staking, and voting. To make the proxy contract act as a load balancer for various contracts we have used a dynamic registry to keep track of implementations mapped to the function signatures of the implementation contract functions. The moderator of the proxy contract has access to add, update and delete the function signatures from the registry.

## 2.2 Security Considerations

**Access Control:** The proxy contract includes role-based access control for registry updates, ensuring that only authorized accounts can modify the contract registry. The proxy contract consists of two roles:

1. **Admin:** The deployer of all the contracts as well as the initializer of the moderator address.
2. **Moderator:** The person responsible for the updation of the dynamic registry of implementation contract addresses mapped to function signatures of the implementation contract. The moderator is not allowed to delegate call to implementation contract via fallback for security reasons.

## 2.3 Scalability and Performance

**Gas Efficiency:** The contract is optimized for gas efficiency to minimize execution costs for delegated functions. The project uses only functions required for accessing the registry and modifying it saving unnecessary storage inside the contract.

**Dynamic Registry:** The contract includes a dynamic registry of function signatures and corresponding contract addresses to support updates, additions, and removals without disrupting the proxy contract or requiring redeployment.

# 3 Challenges Faced

## 3.1 Technical Challenges

**Nonce Management:** Ensuring correct nonce management was a challenge when sending multiple transactions from the same account. Currently this issue is addressed via sending the transactions individually to avoid incorrect nonce issues. Further in the future we can implement a Nonce Manager to keep track of the transaction count and include it with the sent transaction.

# 4 Implementation

## 4.1 Code Overview

1. The contract code is divided into multiple files for clarity and maintainability.
2. The contract is implemented using Solidity, a high-level programming language for writing smart contracts on the Ethereum blockchain.
3. The project is developed and thoroughly tested using the Hardhat Framework.

## 4.2 Key Features

**Fallback Function:** The contract includes a fallback function that delegates function calls to the appropriate implementation contract address based on the function signature present in the registry.

**Dynamic Registry:** The contract maintains a dynamic registry of function signatures and corresponding contract addresses, allowing for updates without requiring redeployment. Functions are created for addition, updation and removal of function signatures and their corresponding implementations, which can be called only by the moderator of the contract.

**Role-based access controls:** The contract consists of two assigned roles admin and moderator.

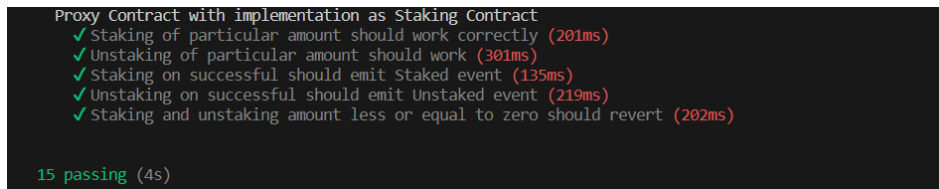
**Storage Slots:** Storage slots are used to allocate specific memory slots to admin and moderator addresses avoiding memory clashes and overwrite of data

## 4.3 Testing

**Unit Testing:** Unit tests are written using the Hardhat testing framework to ensure correct function delegation and execution.

**Integration Testing:** Integration tests are performed to verify the interaction between the proxy contract and the implementation contracts.

Screenshot for the test cases:



```
Proxy Contract with implementation as Staking Contract
✓ Staking of particular amount should work correctly (201ms)
✓ Unstaking of particular amount should work (301ms)
✓ Staking on successful should emit Staked event (135ms)
✓ Unstaking on successful should emit Unstaked event (219ms)
✓ Staking and unstaking amount less or equal to zero should revert (202ms)

15 passing (4s)
```

Figure 1: Passing of all the test cases

## 4.4 Version Control

The project uses Git for version control, with a main branch for stable releases and feature branches for ongoing development.

## 5 Detailed Interaction Guide

The steps to setup the project and interacts with the code is provided below:

NOTE:

- To interact with an already deployed contract with implementation contracts already present in the project, utilize the scripts/deployedContractCall.js and call the required function.

- Make sure to replace the proxy contract address in .env file to the address of proxy contract over testnet.
- Following are the addresses of contracts deployed over sepolia testnet:  
 Proxy Contract Address: '0x7bBf9Ea62EAEFbAe90f1C17b66c5F7C63B7d0a2B'  
 Token Address: '0x899f46e4e57216F0EEeD7D56602098a517Cb212E' Staking Address : '0xB059A07138dA863FA0E0E77FFE6542a2d62c46fc'

#### Deployment and Interaction Steps for the project

1. Clone the project from the github repository: [Github Repository](#)
2. Execute the command '**npm install**' - This would install all the required packages and modules to your project.
3. Create a .env file and create the required variables.
4. Assign values to the variables leaving Proxy Contract Address and Implementation Contract Address as they will be assigned after deploying the corresponding contracts over the network.
5. In the hardhat.config.js file configure the network you want to use and set it as default network.
6. In the contracts folder add the code for the implementation contract with which you want to interact make sure to use initialize function if contract needs to be initialized to a state beforehand.  
 The sample code for the same is provided in the Token.sol contract for reference.
7. Replace the implementation contract name with the name of your implementation contract(eg. Token to be replaced with XYZImplementation) in scripts/deploy.js, scripts/proxyInteraction.js, scripts/delegate.js and test/proxyTest.js
8. Your project is completely setup now.
9. Execute the command '**npx hardhat run scripts/deploy.js**' to execute deployment script for the contracts. The console will display the addresses of contracts deployed, copy them and paste in the .env file accordingly.
10. Next execute the command '**npx hardhat run scripts/proxyInteraction.js**'  
 – This will add function signatures to the registry of proxy contract. Replace the function signature with the signature of your implementation contract while calling 'testcases(functionName)' from main() method.
11. Execute '**npx hardhat run scripts/delegate.js**' (Make sure to add the function signature of the required function in proxyInteraction.js and call the function in delegate.js) – This will delegate the call via proxy to implementation contracts.
12. To execute the tests execute command '**npx hardhat test**'

## 6 Deployment and Maintenance

### 6.1 Deployment Process

- The contract is deployed using Hardhat, a development environment for Ethereum smart contracts.
- Deployment scripts are used to automate the deployment process and ensure consistency across different environments.

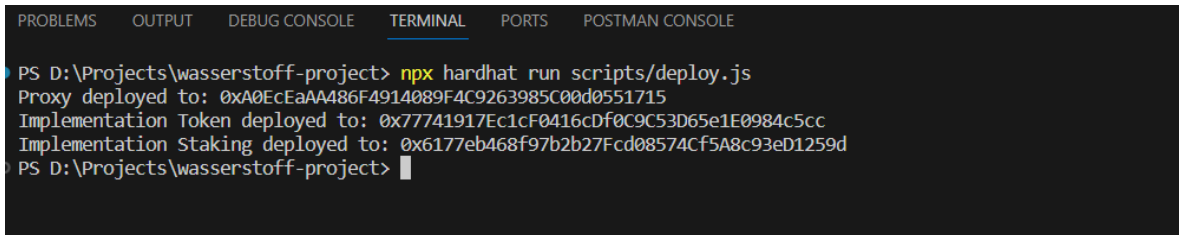
### 6.2 Maintenance

**Updates:** Updates to the contract are managed through a versioning system to ensure backward compatibility.

## 7 Results

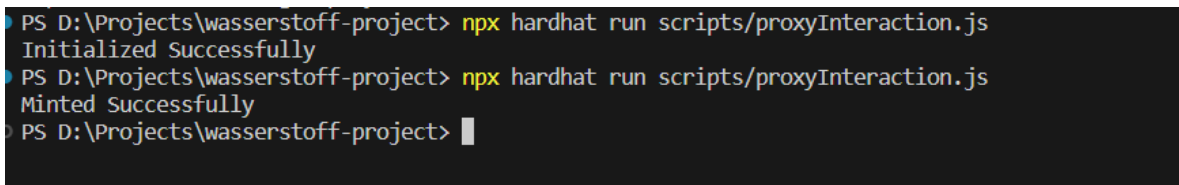
The project was successfully developed and deployed to the Ethereum Sepolia Testnet. The contract addresses were retrieved and successful interaction with the implementation contract via the proxy contract was setup.

Below are the screenshots of the process:



```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS  POSTMAN CONSOLE
PS D:\Projects\wasserstoff-project> npx hardhat run scripts/deploy.js
Proxy deployed to: 0xA0EcEaAA486F4914089F4C9263985C00d0551715
Implementation Token deployed to: 0x77741917Ec1cF0416cDf0C9C53D65e1E0984c5cc
Implementation Staking deployed to: 0x6177eb468f97b2b27Fcd08574Cf5A8c93eD1259d
PS D:\Projects\wasserstoff-project>
```

Figure 2: Successful deployment of the contracts



```
PS D:\Projects\wasserstoff-project> npx hardhat run scripts/proxyInteraction.js
Initialized Successfully
PS D:\Projects\wasserstoff-project> npx hardhat run scripts/proxyInteraction.js
Minted Successfully
PS D:\Projects\wasserstoff-project>
```

Figure 3: Successful entry addition for function signatures to proxy contract

## 8 Conclusion

The development process for Task 2 of the Wasserstoff Blockchain Interviews (2024) involved careful consideration of design decisions, security measures, and performance optimizations. Challenges were addressed through collaboration and problem-solving, resulting in a robust and efficient solution.

Detailed document for interaction with the project is in the readme.md file.

```
PS D:\Projects\wasserstoff-project> npx hardhat run scripts/delegate.js  
0xbea7fd269fbae8e3ac5afd6739560aef2351e65d05cc240ac7f091904c51d886  
PS D:\Projects\wasserstoff-project> npx hardhat run scripts/delegate.js  
0x0e89b706990e310cc8f9812ae038c308257f1fcc97fce981901edecbcb649948  
PS D:\Projects\wasserstoff-project> □
```

Figure 4: Successful Interaction with implementation via proxy contract

## 9 References

1. [Hardhat Documentation](#)
2. [Solidity Documentation](#)
3. [Ethers Documentation](#)
4. [Alchemy Docs](#)
5. [Openzeppelin Contracts: Github](#)

## 10 Future Scope

1. Additional functionality of allowing the user to interact via user interfaces.
2. Nonce Management System can be implemented.

## 11 Appendices

1. [Github Repository: https://github.com/anjalikh99/Wasserstoff-Task-2-2024-Blockchain-Interviews](https://github.com/anjalikh99/Wasserstoff-Task-2-2024-Blockchain-Interviews)
2. Deployed Contract on Sepolia Testnet:
  - **Proxy Contract:** 0x7bBf9Ea62EAEFbAe90f1C17b66c5F7C63B7d0a2B
  - **Implementation Token:** 0x899f46e4e57216F0EEeD7D56602098a517Cb212E
  - **Implementation Staking:** 0xB059A07138dA863FA0E0E77FFE6542a2d62c46fc