# Decentralized Q&A Platform - Project Report

***Group Members: Osama Ali Khan(cs211010), Junaid Ansar(cs211087), Rameez Raza Rizvi(cs211082)***

## Executive Summary

This project presents a decentralized Question & Answer platform built on blockchain technology, utilizing Ethereum smart contracts to create a transparent, censorship-resistant knowledge-sharing system. The platform enables users to ask questions, provide answers, and vote on content quality in a fully decentralized manner without relying on centralized authorities.

**Key Features:**

* Decentralized question posting and answering
* Community-driven voting system
* Immutable record of all interactions
* No central authority or censorship

## 1. Project Overview

### 1.1 Problem Statement

Traditional Q&A platforms like Stack Overflow, Quora, and Reddit face several challenges:

* **Centralized Control**: Single point of failure and potential censorship
* **Lack of Transparency**: Opaque moderation and ranking algorithms
* **Data Ownership**: User-generated content controlled by platform owners
* **Incentive Misalignment**: Limited rewards for quality contributions
* **Trust Issues**: Difficulty verifying authenticity of information

### 1.2 Solution Approach

Our decentralized Q&A platform addresses these issues by:

* Storing all data on blockchain for immutability and transparency
* Implementing community-driven governance through voting mechanisms
* Eliminating single points of failure through decentralization
* Providing cryptographic proof of all interactions
* Enabling direct peer-to-peer knowledge exchange

### 1.3 Technical Architecture

The system consists of three main components:

1. **Smart Contract Layer**: Solidity contracts managing core logic
2. **Frontend Interface**: Web-based user interface for interaction
3. **Blockchain Infrastructure**: Ethereum network for deployment

## 2. Technical Implementation

### 2.1 Smart Contract Design

#### Core Data Structures

struct Question {

uint id; // Unique identifier

address asker; // Address of question author

string title; // Question title

string description; // Detailed question content

string category; // Category of Question

uint voteCount; // Number of upvotes received

uint timestamp; // Time when question was asked

}

struct Answer {

uint questionId; // Reference to parent question

address responder; // Address of answer author

string text; // Answer content

uint voteCount; // Number of upvotes received

uint timestamp;

}

#### Key Functions

| Function | Purpose | Access Control |
| --- | --- | --- |
| askQuestion() | Submit new questions | Public |
| answerQuestion() | Provide answers to questions | Public |
| voteQuestion() | Upvote questions | Public (once per user) |
| voteAnswer() | Upvote answers | Public (once per user) |
| getQuestions() | Retrieve all questions | View function |
| getAnswers() | Get answers for specific question | View function |

#### Security Features

* **Double-voting Prevention**: Mapping structures prevent users from voting multiple times
* **Input Validation**: Require statements ensure data integrity
* **Access Control**: Public functions with appropriate restrictions

### 2.2 Frontend Implementation

#### Technology Stack

* **HTML5**: Structure and semantic markup
* **CSS3**: Styling and responsive design
* **JavaScript (ES6+)**: Interactive functionality
* **Ethers.js**: Ethereum blockchain interaction
* **MetaMask**: Wallet integration

#### User Interface Components

* Wallet connection interface
* Question submission form
* Question browsing and display
* Voting mechanisms
* Answer submission and display

### 2.3 Python Integration Options

The project supports multiple deployment approaches:

#### Option A: Python Flask Server

* Web3.py for blockchain interaction
* Flask for web server functionality
* Automatic contract compilation and deployment
* RESTful API endpoints

#### Option B: Pure Python Scripts

* Command-line interface for contract interaction
* Direct blockchain communication
* Suitable for automation and testing

## 3. Features and Functionality

### 3.1 Core Features

#### Question Management

* **Question Posting**: Users can submit questions with title, detailed description and category.
* **Question Display**: All questions visible with metadata (author, vote count, timestamp)
* **Question Voting**: Community members can upvote valuable questions

#### Answer System

* **Answer Submission**: Any user can provide answers to existing questions
* **Answer Voting**: Community-driven ranking of answer quality
* **Answer Display**: Chronological listing with vote counts

#### Voting Mechanism

* **One Vote Per User**: Prevents spam and manipulation
* **Immutable Records**: All votes permanently recorded on blockchain
* **Transparent Counting**: Vote tallies publicly verifiable

### 3.2 Advanced Features

#### Reputation System

* Implicit reputation based on vote counts
* Publicly verifiable contribution history
* Incentivizes quality content creation

#### Data Persistence

* All data stored permanently on blockchain
* No risk of data loss or platform shutdown
* Historical records always accessible

## 4. Deployment Guide

### 4.1 Prerequisites

#### Software Requirements

# Development Tools

- Web browser (Chrome/Firefox recommended)

- Remix IDE (browser-based, no installation needed)

- MetaMask browser extension

# Optional: For local blockchain testing

- Node.js (v14.0+) - for serving frontend files

- Ethereum Sipolia - for local blockchain

# Python Environment (Alternative deployment option)

- Python 3.8+

- Web3.py library

- Flask framework

- Solcx compiler

#### Infrastructure Requirements

* Ethereum node access (local or remote)
* MetaMask browser extension
* Sufficient ETH for gas fees

### 4.2 Local Development Setup

#### Step 1: Environment Preparation

# Create project directory

mkdir blockchain-qa-platform

cd blockchain-qa-platform

# Initialize Node.js project (for frontend dependencies)

npm init -y

# Install frontend dependencies

npm install ethers

#### Step 2: Smart Contract Development with Remix

1. Open [Remix IDE](https://remix.ethereum.org/) in your browser
2. Create a new file DecentralizedQA.sol
3. Copy and paste the smart contract code
4. Select Solidity compiler version 0.8.0 or higher
5. Compile the contract
6. Deploy using Remix's deployment interface

#### Step 3: Local Blockchain Setup

Use Ganache Gui for local Blockchain setup and connection

#### Step 4: Frontend Configuration

1. Update contract address in HTML file with the deployed address from Remix
2. Configure MetaMask for your chosen network (local Ganache or testnet)
3. Serve frontend files using a local server:

# Using Python

python -m http.server 8000

### 4.3 Production Deployment

#### Testnet Deployment with Remix

1. **Configure MetaMask**
2. **Obtain testnet ETH** from faucets
3. **Connect Remix to MetaMask**:
   * In Remix, go to Deploy & Run tab
   * Select "Injected Provider - MetaMask" as environment
   * Ensure MetaMask is connected to desired testnet
4. **Deploy contract** directly through Remix interface
5. **Copy contract address** and update frontend configuration

#### Mainnet Considerations

* Security audits recommended before mainnet deployment
* Gas optimization analysis using Remix's gas estimation
* Comprehensive testing on testnets required
* Consider using Remix's verification features for contract transparency

## 5. Security Analysis

### 5.1 Smart Contract Security

#### Implemented Protections

* **Reentrancy Prevention**: No external calls in state-changing functions
* **Access Control**: Appropriate function visibility modifiers
* **Input Validation**: Require statements for critical operations
* **Integer Overflow**: Solidity 0.8+ built-in protection

#### Potential Vulnerabilities

* **Gas Limit Issues**: Large arrays could cause out-of-gas errors
* **Front-running**: Vote transactions could be front-run
* **Spam Prevention**: No economic barriers to prevent spam

### 5.2 Recommended Security Enhancements

// Add economic barriers

uint256 public questionFee = 0.01 ether;

uint256 public answerFee = 0.005 ether;

// Implement time locks

mapping(address => uint256) public lastQuestionTime;

uint256 public questionCooldown = 1 minutes;

// Add reputation requirements

mapping(address => uint256) public userReputation;

uint256 public minReputationToVote = 10;

## 6. Testing Strategy

### 6.1 Unit Testing

#### Smart Contract Tests with Remix

Remix IDE provides built-in testing capabilities:

1. **Unit Testing in Remix**:
   * Create test files in the tests folder
   * Use Remix's Solidity Unit Testing plugin
   * Write tests using Remix's testing framework

// Example test file: test\_DecentralizedQA.sol

pragma solidity ^0.8.0;

import "remix\_tests.sol";

import "../DecentralizedQA.sol";

contract TestDecentralizedQA {

DecentralizedQA qa;

function beforeAll() public {

qa = new DecentralizedQA();

}

function testAskQuestion() public {

qa.askQuestion("Test Title", "Test Description");

DecentralizedQA.Question[] memory questions = qa.getQuestions();

Assert.equal(questions.length, 1, "Should have one question");

}

function testVoteQuestion() public {

qa.askQuestion("Test", "Description");

qa.voteQuestion(0);

DecentralizedQA.Question[] memory questions = qa.getQuestions();

Assert.equal(questions[0].voteCount, 1, "Should have one vote");

}

}

1. **Manual Testing through Remix Interface**:
   * Deploy contract in Remix
   * Use the deployed contract interface to test functions
   * Verify state changes and events
   * Test with different accounts

#### Frontend Testing

* User interface functionality
* MetaMask integration
* Error handling and edge cases

### 6.2 Integration Testing

#### End-to-End Workflows

1. User connects wallet
2. User asks question
3. Another user provides answer
4. Community votes on content
5. Results displayed correctly

#### Cross-browser Compatibility

* Chrome, Firefox, Safari testing
* Mobile responsiveness
* MetaMask compatibility

## 7. Performance Analysis

### 7.1 Gas Cost Analysis

| Operation | Estimated Gas Cost | USD Cost (at 50 gwei, $2000 ETH) |
| --- | --- | --- |
| Ask Question | ~65,000 gas | ~$6.50 |
| Answer Question | ~55,000 gas | ~$5.50 |
| Vote on Question | ~30,000 gas | ~$3.00 |
| Vote on Answer | ~35,000 gas | ~$3.50 |

## 8. Economic Model

### 8.1 Current Model

The current implementation operates without transaction fees or token rewards:

**Costs:**

* Gas fees for all blockchain interactions
* No additional platform fees

**Incentives:**

* Reputation through vote counts
* Community recognition
* Knowledge sharing motivation

### 8.2 Proposed Enhancements

#### Token-Based Rewards

// ERC-20 token for rewards

contract QAToken {

mapping(address => uint256) public balances;

function rewardAnswer(address answerer, uint256 votes) external {

balances[answerer] += votes \* 10; // 10 tokens per vote

}

}

#### Staking Mechanism

* Users stake tokens to ask questions
* Stakes returned with successful answers
* Discourage spam and low-quality content

## 9. Risk Assessment

### 9.1 Technical Risks

| Risk | Impact | Probability | Mitigation |
| --- | --- | --- | --- |
| Smart Contract Bugs | High | Medium | Comprehensive testing, audits |
| Gas Price Volatility | Medium | High | Layer 2 solutions, gas optimization |
| Network Congestion | Medium | Medium | Alternative networks, batching |
| Frontend Vulnerabilities | Medium | Low | Security best practices, regular updates |

### 9.2 Business Risks

| Risk | Impact | Probability | Mitigation |
| --- | --- | --- | --- |
| Low User Adoption | High | Medium | Marketing, user incentives |
| Regulatory Changes | High | Low | Legal compliance, adaptability |
| Competition | Medium | High | Unique features, community building |
| Economic Model Failure | High | Low | Iterative improvements, user feedback |

## 10. Conclusion

The **Decentralized Q&A Platform** developed in this project effectively showcases the potential of blockchain technology in building transparent, secure, and community-governed knowledge-sharing systems. By leveraging Ethereum smart contracts, the platform allows users to ask questions, submit answers, and vote on content without relying on a centralized authority.

This ensures that all interactions are permanently recorded on the blockchain, providing an immutable and verifiable history of contributions. Furthermore, the platform empowers the community to manage content quality through a decentralized voting system, making it resistant to censorship and biased moderation. Overall, the project successfully demonstrates how blockchain can be used to create open, trustworthy, and self-regulating platforms for collaborative information exchange.