DECENTRALIZED JOB BOARD

BY ALLIES

Name	Roll No
Rachakonda Chandrahasa	230001065
Rahul Kumar	230001066
Darpan Nayak Tejavath	230001022
Gajendra Singh Rana	230004016
Thikmanik Nongrang	230001077
Sai Abhilash Dash	230005041
Mannuru Praneetha	230005025

1. Introduction

The Job Board platform is a decentralized application (dApp) developed using Solidity and deployed on the Ethereum blockchain. It enables clients to post jobs, escrow payments, and release funds upon satisfactory work completion. Freelancers can apply for jobs, complete tasks, and receive ratings, all without relying on a centralized intermediary.
Key features include:

- Clients posting jobs with deadlines and budgets
- Freelancers applying and being accepted by clients
- Secure escrow and payment release via smart contracts
- Refunds for unfulfilled jobs after deadlines
- Rating system for maintaining freelancer reputation

2. System Architecture Overview

The Job Board platform follows a modular and layered architecture with the following key components:

- **Smart Contract Layer**
  The core logic resides in `JobBoard.sol`, a Solidity smart contract deployed on the Ethereum blockchain. It manages job postings, applications, escrow payments, work completion, and payment release. All actions are recorded immutably on-chain, ensuring trust and decentralization.

- **Frontend Interface**
  Built using a modern JavaScript framework (e.g., React or Next.js), the frontend provides an intuitive UI for employers and freelancers. Users can interact seamlessly with features like job posting, application, submission, rating, and payment release.

- **Interaction Layer**
  Integration is handled via `web3.js` or similar libraries to connect MetaMask with the smart contract. It enables read/write operations and listens for smart contract events to update the UI in real time.

This architecture ensures complete decentralization, removes intermediaries, and enhances security and transparency.

3. Smart Contract Design

The `JobBoard.sol` contract handles the business logic and state management of the decentralized freelance platform. It ensures transparency, minimizes trust assumptions, and automates interactions through the Ethereum blockchain.

3.1 Data Structures

- **Job**
  Encapsulates all necessary information about a freelance job:

  - `id`: Unique job ID.
  - `title`: Job title.
  - `description`: Detailed description of the job.
  - `budget`: Payment offered for completion.
  - `deadline`: UNIX timestamp for submission deadline.

    - `employer`: Address of the job creator.
    - `freelancer`: Address of the assigned freelancer.
    - `isCompleted`: Boolean flag indicating work completion.
    - `ratingGiven`: Boolean indicating if a rating has been provided.
    - `rating`: Numerical performance rating (1 to 5).

- **FreelancerRating**
  Tracks the overall performance of each freelancer:
    - `total`: Total of all ratings received.
    - `count`: Number of jobs rated.
    - `lastRated`: Timestamp of the last rating.

- **JobStatus (Enum)**
  Represents the lifecycle of a job:

    - `Open`: Available for applications.
    - `Assigned`: Freelancer has been hired.
    - `Completed`: Work has been submitted and payment processed.

Key Functions:

- **Job Posting**: Allows employers to post new job listings with essential details.
- **Job Application**: Freelancers can apply for available jobs, and once assigned, they are given a deadline.
- **Work Submission**: Freelancers can mark the work as completed, triggering the escrowed funds release.
- **Escrow & Payment**: Employers escrow funds upfront, which are released upon successful work completion and a rating.
- **Refund Mechanism**: If a job is not completed by the deadline, employers can refund their escrowed funds.

Security Mechanisms:

- **Modifiers**: Restrict access to certain actions, ensuring that only the employer or freelancer can perform specific tasks.

- **Access Control**: Ensures that actions like posting jobs or making payments are only performed by authorized users.

Events:

Events like **JobPosted**, **FundsEscrowed**, **WorkCompleted**, and **FreelancerRated** track state changes, ensuring a transparent and auditable flow of actions on the blockchain.

3.1 **Escrow and Payment Workflow**

The **Escrow and Payment Workflow** in the JobBoard contract ensures secure and transparent financial transactions between employers and freelancers.

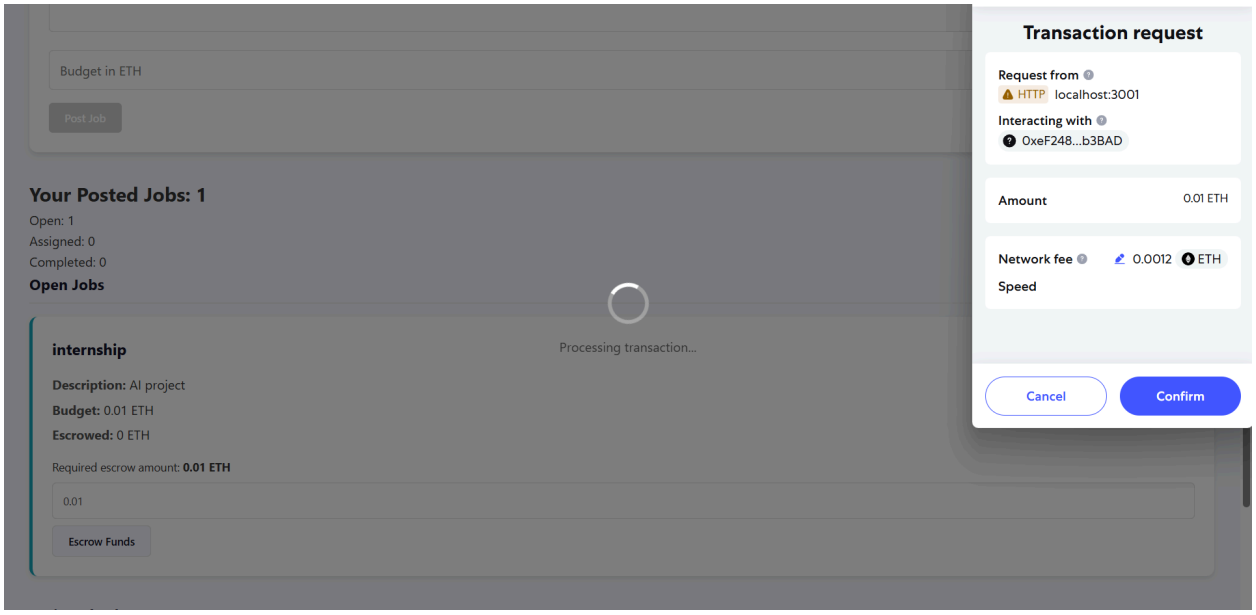
Job Lifecycle:

1. **Job Creation**: The employer posts a job with a specified budget, which is stored on the blockchain.

2. **Escrow of Funds**: Upon job posting, the employer escrows the agreed amount in ETH by sending it to the contract. These funds are held securely in the contract until the job's completion.

3. **Work Completion**: Once the freelancer completes the job, they mark it as "work completed." The employer then reviews the work.

4. **Payment Release**: If the employer approves the work and provides a rating (between 1-5), the escrowed funds are released to the freelancer.

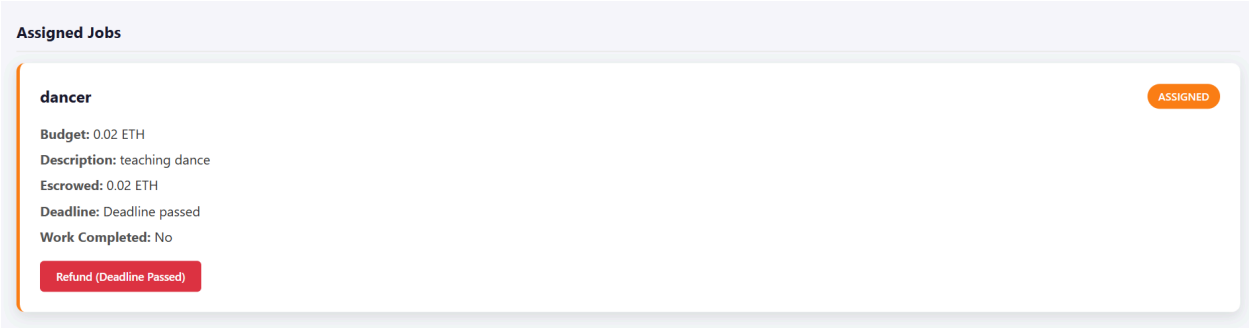


The employer will escrow the funds

Refund Conditions:

If the freelancer fails to complete the job by the set deadline, the employer can request a refund. The contract ensures that if the work isn't completed within the deadline and no funds have been released, the employer can reclaim the escrowed amount.

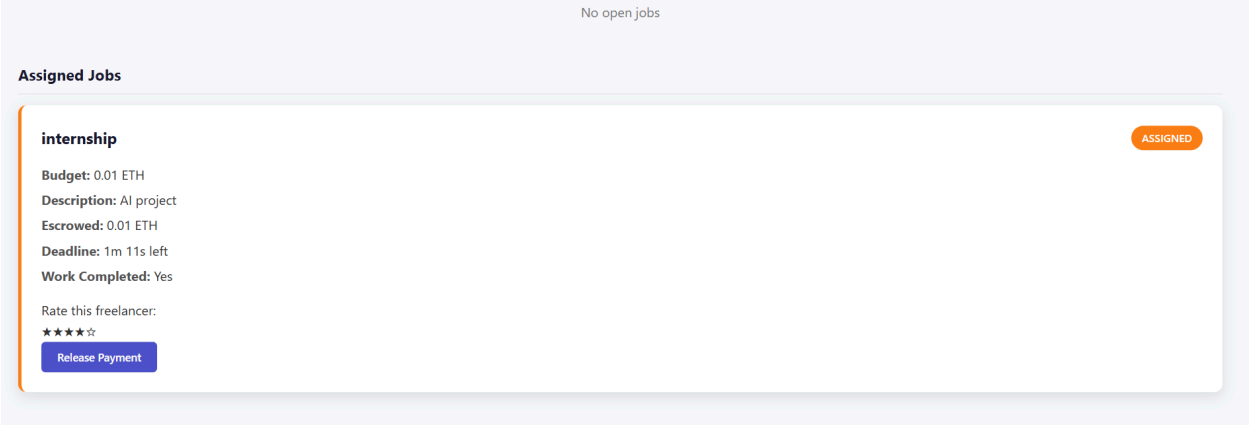
This workflow ensures a fair transaction environment, protecting both employers and freelancers by holding funds securely and releasing them only upon mutually agreed conditions.



If the Job is not done by the freelancer, employer is refunded the escrowed amounts

Freelancer Rating System:

The **Freelancer Rating System** in the JobBoard contract enables employers to rate freelancers after the completion of a job, ensuring a transparent reputation system.



The employer rates the freelancer and then releases the payment

Rating Logic:

After the freelancer marks the work as complete, the employer provides a rating between 1 and 5. This rating is then stored in the job record, and the freelancer's overall reputation is updated accordingly.

Average Rating Calculation:

The contract maintains a **FreelancerRating** struct that tracks the total ratings, rating count, and last rating timestamp for each freelancer. The average rating is calculated as the total ratings divided by the rating count.

FreelancerRating storage rating = freelancerRatings[job.freelancer];

rating.totalRatings += _rating;

rating.ratingCount += 1;

The rating is shown in the freelancer Dashboard.

The **Frontend Interface** of the JobBoard platform is built using modern technologies like **React** for dynamic user interfaces ,is organized into key sections:

- **JobList**: Displays available jobs with their details.
- **JobDetails**: Shows detailed information about a job, including employer and freelancer details.
- **JobForm**: Allows users to post new jobs.
- **Dashboard**: Provides a personalized view for both employers and freelancers.

Wallet Integration:

To enable seamless blockchain interactions, **MetaMask** and **WalletConnect** are integrated, allowing users to connect their wallets directly to the platform for job posting, payment, and rating actions.

State Management:

For efficient state management, **React Context API** is used to handle global states, ensuring smooth transitions between components and a responsive UI.

The frontend seamlessly connects with the smart contract on the blockchain, offering an intuitive, secure, and easy-to-use interface for employers and freelancers alike.

Here's a simplified and structured version of all the sections in bullet-point format for clarity and easy inclusion in a technical report or documentation:

4. Blockchain Interaction Logic

- Uses **web3.js** to connect the frontend with the Ethereum smart contract.
- Enables key actions: job posting, applying, marking work complete, releasing payment, and rating.
- Handles **transaction lifecycle**: sending, waiting for confirmations, and updating the UI.

- Implements **error handling**: detects and displays issues like insufficient funds or failed transactions.
- Uses **loading indicators** and **notifications** to inform users during transactions.

5. Deployment & Environment Setup

- **Smart contract deployment** is done using **Truffle** to the **Sepolia testnet**.
- **Frontend** is hosted locally during development and can be deployed via **Vercel** for production.
- **Environment variables** (e.g., contract address, Infura ID) are stored in a `.env` file for secure access.
- Deployment setup ensures secure, scalable, and test-friendly development.

6. Testing and Debugging

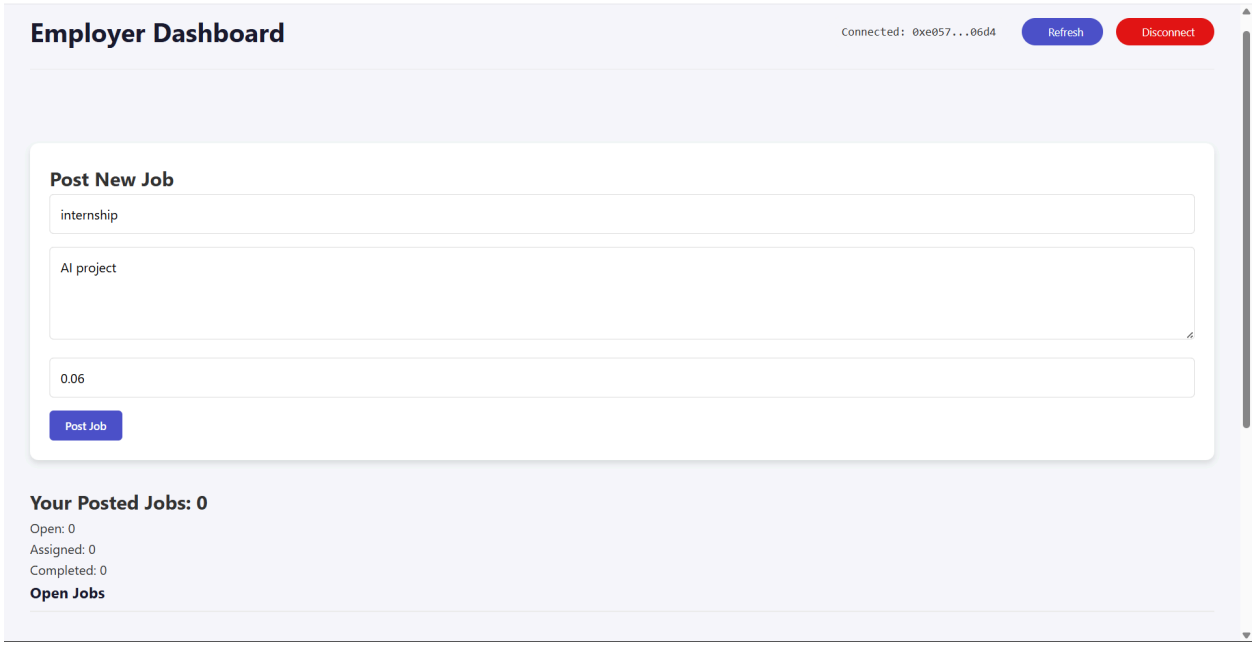
- Uses **Truffle** and **Ganache** for smart contract testing.
- **Unit tests** check core features: job posting, applications, payments, and ratings.
- **Manual frontend testing** simulates real user actions for end-to-end verification.
- Tests cover **edge cases**: no escrowed funds, double ratings, invalid transitions.

7. Security Considerations

- Uses `.call()` for ETH transfers, with **success checks** to prevent loss of funds.

- Implements **role-based access control**:
    - `onlyEmployer`: restricts payment and refund actions.
    - `onlyFreelancer`: restricts job completion marking.
- Adds **deadline logic** to allow refunds if a job is not completed on time.
- Frontend restricts UI actions based on **contract state**, ensuring secure interactions.

Employer Dashboard:



Freelancer Dashboard: