Takneek PS - Quest

70 Points

Overtime

Problem Statement:

The objective is to develop a decentralised service where workers can be allocated tasks based on their specifications and get paid upon completion. The system should meet the following requirements:

- **Worker Registration:**
    1. Individual workers can register on the platform.
    2. During registration, workers must specify their availability (no. of hours), expertise level (numerical value), and minimum acceptable hourly wage.
    3. Workers are required to link their wallet for payment processing.
    4. Workers can only work with their available number of hours. Once all allocated hours are completed (working for 1 or more tasks) , the worker will be considered deregistered from the platform.
- **Task Addition by Admin:** (Assume only one Admin)
    1. The Admin can add tasks, specifying the required time (an integer value), expertise level (numerical value), task dependencies (if any), hourly wage, deadline, and whether the task is divisible or non-divisible (partitioning required time to get smaller tasks, not less than 1 hour).
- **On-Chain Task Allocation:**
    1. Task allocation is automated and handled entirely on-chain via a smart contract.
    2. The allocation algorithm considers constraints specified by both Admin and workers. Tasks can remain unallocated if constraints are not met.
    3. Information about tasks and worker specifications should be stored on-chain to ensure transparency and trustlessness.
    4. The task allocation function can be run at most one time after every task or worker addition. (You may choose to not run task allocation and wait for another query)
    5. Task allocation is automated, which means the logic for when to perform task allocation on-chain should be pre-decided by an algorithm. Task allocation can occur any time based on predefined criteria, such as after an event (e.g., worker or task addition) or other criteria related to the buffer of unassigned tasks and workers.
- **Smart Contract Execution:**
    1. When a worker is allocated a task, a smart contract is initiated between the worker and the Admin. Payment is processed automatically after task completion and the deadline, from Admin's wallet to the worker's wallet.

Takneek PS - Quest

70 Points

All gas fees associated with contract interactions are to be incurred by the Admin.

- **API and Frontend:**
    1. Develop APIs to handle worker registration, task addition, monitoring current allocation status and wallet balance checking.
    2. Create a user interface for both Admin and workers, using the APIs.
    3. The Admin's UI should include data visualisation tools for effective task management, while both workers and admin's UI should cater to their specific tasks and payment information. There should be additional API's on Admin side to show on-chain cost incurred and other related statistics.
- **Deployment and Blockchain Integration:**
    1. Integrate with a test Ethereum blockchain (Rinkeby, Goerli etc) and Metamask wallet for consistency.
    2. Deploy the frontend and off-chain backend server, if any.

Deliverables:

1. Codebase:

- A link to the GitHub repository containing the entire codebase. The repository should include the smart contract, frontend code, off-chain backend server code.
- A README file containing a link to the deployed application, a link to the deployed smart contract, and an overview of the codebase.
- A zip file of all the source code and README.

2. Documentation:

- Comprehensive documentation that clearly describes all API endpoints. This documentation will be required to run the test script, and failure to do so will result in 0 points.
- A comprehensive documentation hosted on GitHub Pages will receive a bonus of +2 points. (upper limit remains 70 points)

Evaluation

A test script will be used to run a set of queries on each submission. There are four types of queries: `addTask`, `addWorker`, `checkStatus`, and `checkWallet`, all of which will be executed through the provided APIs. Points will be awarded based on the following criteria:

Takneek PS - Quest

70 Points

Note:

- The `checkStatus` API will be run at equal intervals of time. If the task allocation status has not changed between two consecutive `checkStatus` queries, the points for the latter check will not be counted.
- The test script will execute multiple runs with different values for the check interval.
- The average of all runs will be considered for evaluation.
- Remember Task allocation is automated, and there needs to be a pre-decided logic for when to run the task allocation algorithm on-chain.
- Strictly follow the given schema for your API endpoints to ensure consistency in evaluation.

API Schemas:

1. **addTask API:**
    a. **Request Body:**
        i. `time`: Required time for the task (integer, hours)
        ii. `expertise`: Minimum Expertise level required (integer) - higher means more expertise required
        iii. `dependencies`: Task dependencies (array of task IDs)
        iv. `wage`: Hourly wage (integer)
        v. `deadline`: Task deadline (timestamp)
        vi. `divisible`: Whether the task is divisible (boolean)
2. **addWorker API:**
    a. **Request Body:**
        i. `hours`: Number of hours available (integer)
        ii. `expertise`: Expertise level of the worker (integer)
        iii. `min_wage`: Minimum acceptable hourly wage (integer)
        iv. `wallet`: Wallet address for payment (string)
3. **checkStatus API:**
    a. **Response Body:** i. `tasks`: A list of tasks with their details
        - `task_id`: The ID of the task (string)
        - `worker_id`: The ID of the worker allocated to the task, if any (string, nullable)
        - `status`: Status of the task, whether it is completed or not (bool)
4. **checkWallet API:**
    a. **Request Body:**
        i. `worker_id`: The unique identifier of the worker whose wallet balance is being checked.

Takneek PS - Quest

70 Points

Point Breakup

1. **Completion and Correctness (30 points):**
    - Completion
        - Confirms the delivery of all required functionalities, including on-chain and off-chain services.
        - 2 Points Reserved for Documentation.
    - Correctness
        - Ensures that tasks are allocated according to the specified constraints without violating any conditions at any step.
        - Verifies that payments are processed correctly and on time.
2. **Efficiency (17 points):**
    - Evaluated based on on-chain cost minimization. (7 points)
    - The value of the tasks accomplished, calculated as the summation of the expertise required. (4 points)
    - Fair work hour distribution, measured using entropy (Number of hours assigned to workers in proportion to total assigned hours). (3 points)
    - The total money earned by all the workers. (3 points)
    - These points will be distributed relatively, with the best submission receiving full points and others receiving points relative to it.
    - Relative Points= (Score of Submission / Score of Best Submission) × Total
3. **User Interface (15 points):**
    - Evaluated based on the clarity and effectiveness of data visualisation.
    - The system's ability to handle errors gracefully. This will be assessed manually by performing random actions on the frontend.
4. **Presentation (8 Points):** At least three members from the team should be available for presentation, including at least two from Y24. Presentation time will be a maximum of 7 minutes.

Team Structure

1. Each team should have 6 members, with at least 3 from Y24 and 1 from Y23.

Submission Deadline

1. **Deadline:** 4th September, Wednesday, EOD
2. **Repository Privacy:** The GitHub repository should remain private until the submission deadline. It must be made public after the deadline has passed. No commits should be made after the deadline.