CCMP_606

PROJECT REPORT

Student Information

KARTHIK RAJA KOTHANDAN
000502802
kothandan2969@saskpolytech.ca

PRATEEK KAUL
000514655
kaul4818@saskpolytech.ca

Github Repository:
https://github.com/karthikraja213/todoApp.git

1. PROBLEM DEFINITION

The modern world relies heavily on productivity tools, with to-do list apps being among the most common tools used to manage tasks and organize daily activities. However, most existing to-do list apps are centralized, relying on private servers to store user data, which may lead to issues such as:

- Users must trust the platform to keep their data secure and intact which would be an issue of trust and transparency.
- Centralized systems are prone to data loss or tampering by either platform owners or malicious actors which can result in questionable Data Integrity.
- User data can be accessed, sold, or shared without permission and this can hinder data Privacy.

Public User Perspective: Users need a tool that can ensure the integrity and transparency of their tasks, where they are sure their data won't be modified or lost due to server outages or breaches.

Business Perspective: A decentralized to-do list app offers value by providing transparency and immutability of data through blockchain technology. Businesses that adopt this model can attract users who prioritize data security and ownership while exploring blockchain's potential in daily-use applications.

2. DESIGN AND ARCHITECTURE

Contract Name: ToDoList

Interfaces and Libraries

- **Solidity**: Smart contracts will be written in Solidity to interact with the Ethereum blockchain.
- **OpenZeppelin Libraries**: To ensure secure and well-tested contract functionalities such as Ownable and Enumerable mappings.
- **Ethers.js/Web3.js**: These libraries will be used to connect the front-end application with the Ethereum blockchain.

Functionality

The contract will have the following key functions and modifiers:

- **createTask(string memory description)**
  - Allows users to add a new task to the blockchain.
  - **Modifier**: public – any user can create a task.
- **editTask(uint256 taskId, string memory newDescription)**
  - Allows the owner of the task to edit its description.
  - **Modifier**: public – but only the task creator can call this function (onlyOwner).
- **markTaskComplete(uint256 taskId)**
  - Marks the specified task as complete.
  - **Modifier**: public – task owner only (onlyOwner).
- **deleteTask(uint256 taskId)**
  - Soft deletes a task by marking it as inactive.
  - **Modifier**: public – task owner only.
- **getAllTasks()**
  - Retrieves all tasks created by a user, including completed and active tasks.
  - **Modifier**: public view – any user can view their tasks.

- **getTaskDetails(uint256 taskId)**
  - Fetches the details of a specific task (description, status, creation date, etc.).
  - **Modifier**: public view – open to any user.
- **taskCount()**
  - Returns the total number of tasks created by a user.
  - **Modifier**: public view.

Contract Roles

- **Task Owner**: The user who creates the task will be the task owner and will have full control over editing, marking as complete, and deleting the task.

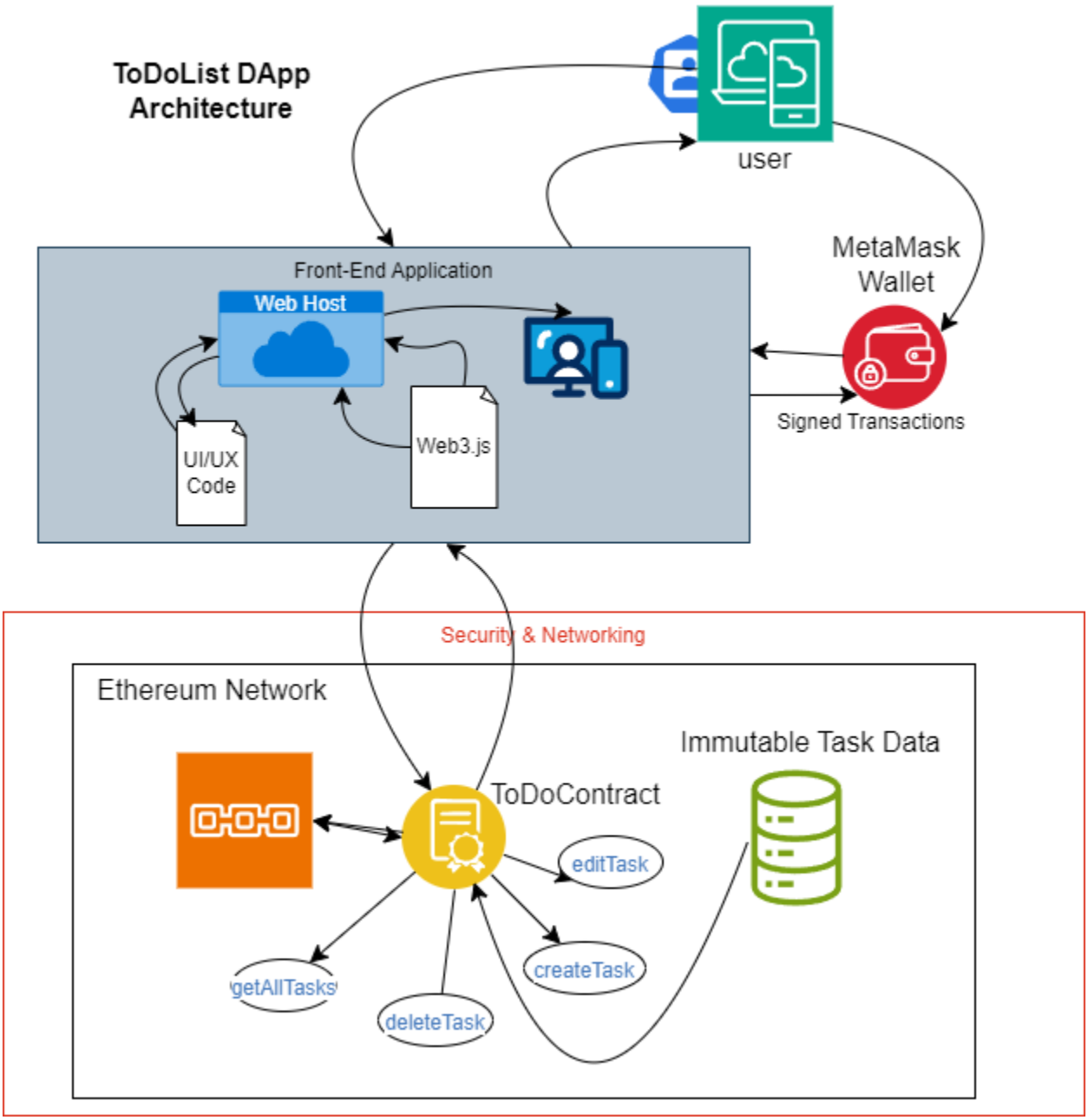
Design Components

- **Roles**: The contract will assign task ownership to the user who creates the task.
- **Event Emitters**: Functions such as createTask, markTaskComplete, and deleteTask will emit events for tracking changes on the blockchain.

Front-End Application Interaction

- The front-end will interact with the Ethereum blockchain via Ethers.js or Web3.js.
- Users will connect to the DApp using MetaMask or a similar Ethereum wallet.
- Users will be able to create, edit, delete, and mark tasks as complete using a simple UI that sends transactions to the smart contract.
- The front-end will retrieve task data from the blockchain and display it in real-time.

Architecture Diagram



3. IMPLEMENTATION

Tools and Technologies

- **Remix IDE**: Used for writing, compiling, and deploying the Solidity smart contract.
- **Metamask**: Acts as the wallet provider, allowing users to connect to the Ethereum blockchain and sign transactions.
- **Web3.js**: Facilitates communication between the frontend and the Ethereum blockchain.
- **HTML, CSS, and JavaScript**: Used to build a responsive and user-friendly frontend interface.

Development Process

1. Smart Contract Development

The smart contract was written in Solidity using the Remix IDE. It includes functions for:

- **Task Creation**: Allows users to create tasks, either one by one or in batches.
- **Task Editing**: Enables users to update task descriptions.
- **Task Completion**: Marks tasks as completed.
- **Task Soft Deletion**: Marks tasks as inactive instead of deleting them permanently.
- **Task Retrieval**: Allows users to fetch tasks filtered by status (active or inactive).

The contract was deployed to the Sepolia testnet using Remix's integrated deployment tools.

2. Frontend Development

The frontend was built using HTML, CSS, and JavaScript:

- **HTML**: Defines the structure of the user interface.
- **CSS**: Adds styling, including a background image, task list formatting, and button designs.

- **JavaScript**: Manages interactions, such as adding tasks, editing them, and communicating with the blockchain using Web3.js.

Key UI features include:

- Input fields for adding tasks.
- Buttons for editing, completing, and deleting tasks.
- A task list dynamically updated with data fetched from the blockchain.

3. Communication with Blockchain

Web3.js was used to integrate the frontend with the Ethereum blockchain:

- **Connecting to Metamask**: The app requests access to the user's wallet to sign transactions.
- **Interacting with the Smart Contract**:
    - Functions like createTask, editTask, and deleteTask are invoked from the frontend.
    - Events emitted by the smart contract are listened to for real-time updates.

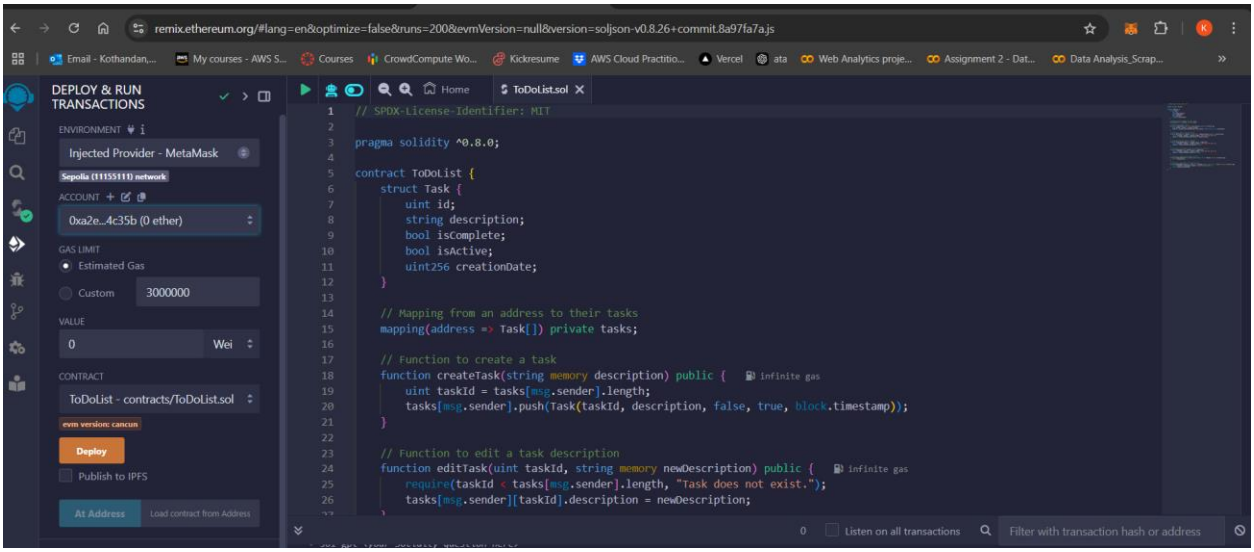
4. Wallet Integration

Metamask was integrated to:

- Allow users to log in using their Ethereum wallet.
- Handle transaction signing for task-related operations.
- Provide access to the Ethereum network (testnet).

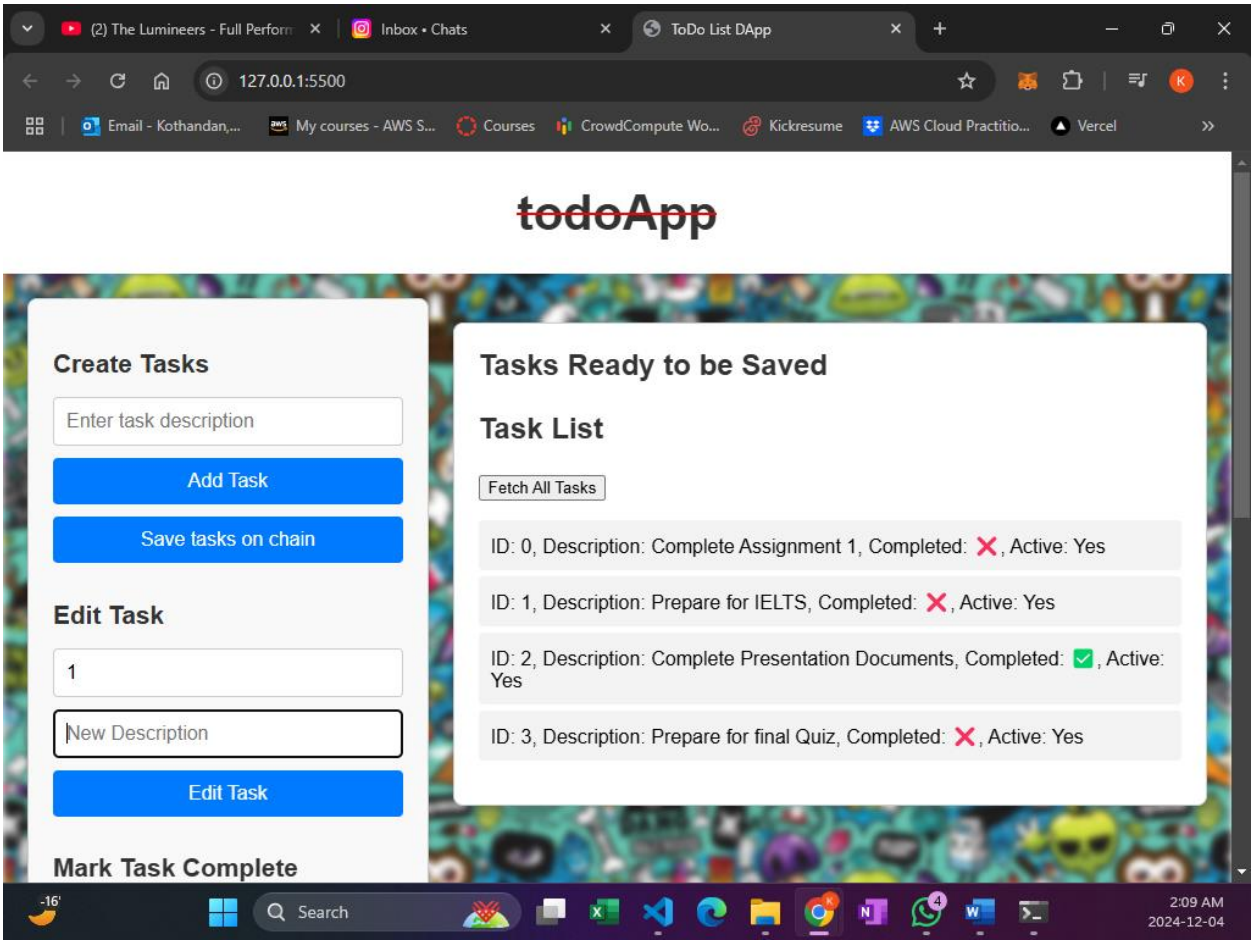
```
createTaskForm.addEventListener("submit", async (e) => {
    e.preventDefault();

    if (tasks.length === 0) {
        alert("No tasks to submit. Please add tasks first.");
        return;
    }

    try {
        const accounts = await web3.eth.getAccounts();

        // Submit all tasks in one transaction
        await contract.methods.createTasks(tasks).send({ from: accounts[0] });

        alert("All tasks submitted successfully!");
        tasks = []; // Clear local tasks
        taskList.innerHTML = ""; // Clear UI list

    } catch (error) {
        if (error.code === 4001) {
            alert("Transaction canceled by the user.");
        } else {
            console.error("Error submitting tasks:", error);
            alert("Please Log into Metamask to submit tasks");
        }
    }
});
```

1 CONTRACT COMMUNICATION WITH WEB3 CODE SNIPPET

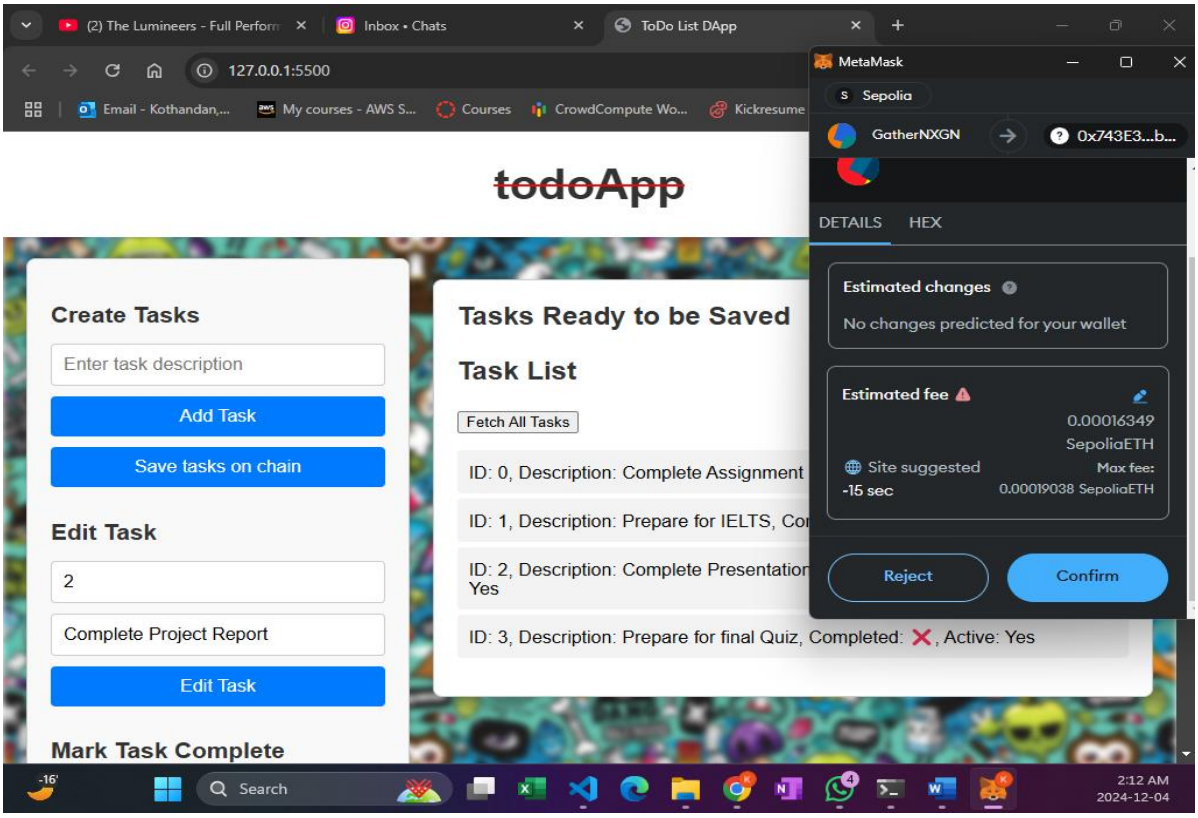


2 CONTRACT DEPLOYED WITH REMX IDE

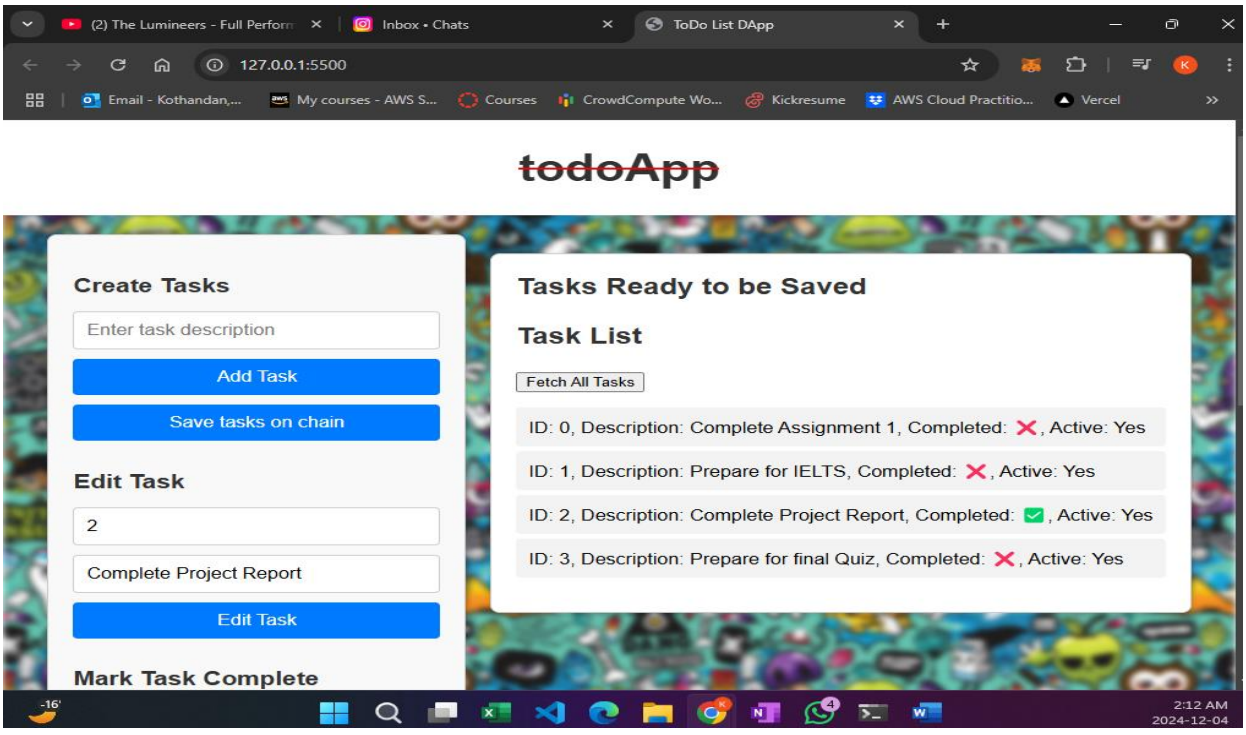
4. RESULTS



3 USER INTERFACE



4 SEND TRANSACTIONS WITH METAMASK



5 UPDATED RESULTS IN REAL-TIME

5. CONCLUSION

This Decentralized To-Do List DApp uses the transparency, security, and immutability of blockchain technology to provide users with a tool they can trust for managing tasks. By storing task data on the blockchain, the app eliminates concerns about data integrity, manipulation, or unauthorized access, offering users full control over their tasks while promoting a decentralized model of productivity tools. As a next step, we can begin implementing the smart contracts and developing the front-end integration.