System Manual for Decentralized Healthcare System DApp

Table of Contents

1. Introduction

The Decentralized Healthcare System DApp is designed to leverage blockchain technology to enhance healthcare records management. This application aims to provide a secure, transparent, and efficient way for patients, doctors, and administrators to interact with healthcare data while ensuring patient privacy and data integrity. By utilizing Ethereum smart contracts, the DApp facilitates various functionalities such as patient registration, medical record management, and access control for healthcare professionals.

2. Project Structure

The project consists of two primary components: the backend, which includes the smart contracts, and the frontend, which provides the user interface for interaction.

2.1 Backend (Smart Contracts)

The backend is implemented using Solidity and consists of the following contracts:

- **Healthcare System Contract**: This contract manages the core functionalities of the healthcare system, including user registrations and medical record handling.
- **Lock Contract**: This contract provides a mechanism for locking funds until a specified unlock time, allowing for time-based withdrawals.

2.2 Frontend (User Interface)

The frontend is developed using React and Vite, offering an interactive and user-friendly interface for users to engage with the DApp. It includes components for patient registration, doctor registration, medical record updates, and access management.

3. Installation

3.1 Prerequisites

Before setting up the project, ensure you have the following installed on your machine:

- **Node.js**: A JavaScript runtime that allows you to run JavaScript code server-side. Version 14 or higher is recommended.
- **npm**: The Node package manager that comes with Node.js, used for managing project dependencies.
- **yarn (optional)**: An alternative package manager that can be used instead of npm.
- **MetaMask**: A browser extension that serves as a cryptocurrency wallet and allows interaction with Ethereum-based applications.
- **Hardhat**: A development environment for Ethereum that facilitates smart contract development, testing, and deployment.

3.2 Setting Up the Backend

1. **Navigate to the Backend Directory**: Open your terminal and change to the project directory:

bash command

```
cd MyDapp
```

2. **Install Dependencies**: Install the necessary dependencies for the project using npm:

bash command

```
npm install
```

This command will read the **package.json** file and install all required packages.

3. **Compile the Smart Contracts**: Use Hardhat to compile the smart contracts, which will generate the necessary artifacts for deployment:

bash command

```
npx hardhat compile
```

This command will create a directory named **artifacts** containing the compiled contract files.

4. **Deploy the Smart Contracts**: To deploy the contracts to a local Ethereum network, first, start the Hardhat node:

bash command

```
npx hardhat node
```

This command will launch a local Ethereum network on your machine.

In a new terminal window, run the deployment script:

bash command

```
npx hardhat run scripts/deploy.js --network localhost
```

This command will deploy the **Healthcare System** and **Lock** contracts to the local network and output the contract addresses.

3.3 Setting Up the Frontend

1. **Navigate to the Frontend Directory**: Change to the frontend directory:

bash command

```
cd frontend
```

2. **Install Frontend Dependencies**: Install the frontend dependencies using npm:

bash command

npm install

This will set up all the necessary libraries for the React application.

3. **Configure Environment Variables**: If required, create a **.env** file in the frontend directory to store sensitive information such as contract addresses or API keys.

4. Usage

4.1 Running the Application

1. **Start the Frontend Development Server**: Use the command below to start the development server for the frontend application:

bash command

npm run dev

This command will compile the React application and serve it locally.

2. **Access the Application**: Open your web browser and navigate to **http://localhost:3000**. This URL will display the user interface of the DApp, allowing users to interact with the system.

4.2 User Roles

The application supports three distinct user roles, each with specific permissions and functionalities:

- **Patient**:
    - Can register themselves in the system.
    - Has the ability to update their medical records by uploading new data.
    - Can grant or revoke access to their medical records for specific doctors, ensuring control over who can view their sensitive information.
- **Doctor**:
    - Can register in the system upon approval from an admin.
    - Has the capability to view the medical records of patients who have granted them access, facilitating better patient care and treatment.
- **Admin**:
    - Responsible for registering new doctors in the system.
    - Has the authority to manage user accounts and oversee the overall functionality of the DApp, ensuring compliance with healthcare regulations.

5. Smart Contracts Overview

5.1 Healthcare System Contract

This contract is the backbone of the DApp, providing essential functions for managing healthcare data:

- **Functions**:

- **registerDoctor(address _doctor)**: Allows the admin to register a new doctor by providing their Ethereum address.
- **registerPatient()**: Enables patients to self-register, creating their unique profile in the system.
- **updateMedicalRecord(string memory _ipfsHash)**: Allows patients to update their medical records by providing an IPFS hash that points to the stored data.
- **grantAccessToDoctor(address _doctor)**: Permits patients to grant access to their medical records to a specific doctor, enhancing collaborative care.
- **viewMedicalRecord(address _patient)**: Allows doctors to view a patient's medical record if access has been granted, ensuring that they have the necessary information for treatment.

6. Troubleshooting

6.1 Common Issues

- **MetaMask Connection Issues**: Ensure that MetaMask is connected to the correct network (e.g., localhost) and that the account has sufficient funds for transactions.
- **Contract Deployment Failures**: If contracts fail to deploy, check the console for error messages and ensure that the Hardhat node is running.

6.2 Debugging Steps

- Use console logs within the smart contracts to trace function calls and state changes.
- Check the browser console for any frontend errors that may indicate issues with the React application.

7. Future Enhancements

7.1 Lock Contract

This contract is designed to manage funds securely, providing a mechanism for time-based withdrawals:

- **Functions**:
    - **withdraw ()**: Allows the contract owner to withdraw funds only after a specified unlock time has passed, ensuring that funds are not accessed prematurely.

7.2 Multi-Signature Admin Control

- Implement a multi-signature wallet for admin functions to enhance security. This would require multiple approvals for critical actions, such as registering new doctors or modifying access permissions.

7.3 Support for Multiple Medical Records Formats

- Extend the functionality to support various formats for medical records, such as images, documents, or video files, enabling a more comprehensive representation of a patient's health history.

7.4 User Feedback and Rating System

- Introduce a feedback mechanism that allows patients to rate their doctors and provide comments on their experiences. This could help improve the quality of care and accountability.

8. Conclusion

The Decentralized Healthcare System DApp provides a robust solution for managing healthcare records securely and transparently. Following this manual allows users to effectively set up, run, and interact with the application. For further assistance, users are encouraged to consult the documentation of the libraries and frameworks utilized in the project.

9. Appendices

9.1 References

- Ethereum Documentation: https://ethereum.org/en/developers/docs/
- Hardhat Documentation: https://hardhat.org/getting-started/
- React Documentation: https://reactjs.org/docs/getting-started.html