## Designing reusable smart contracts for Electronic Health Record (HER)

Designing reusable smart contracts for an Electronic Healthcare Record (EHR) system is crucial for achieving efficiency, security, and scalability in healthcare data management on a blockchain. Designing reusable smart contracts helps to achieve modularity, scalability, and reusability. The reusable smart contact in HER can then be customized and adopted across all healthcare applications. Designing reusable smart contracts for EHR systems involves a thoughtful combination of blockchain technology, healthcare domain expertise, and a deep understanding of privacy and security considerations. These contracts should not only enhance data management but also promote patient-centric control over their healthcare data while ensuring compliance with healthcare regulations. The Table 2 Smart Contract for PatientRegitry in Ethereum Solidity.

Table 2 Smart Contract for PatientRegitry in Ethereum Solidity

pragma solidity ^0.8.0;

**contract** PatientRegistry {

struct Patient {

uint256 id;

string name;

uint256 dateOfBirth;

string gender;

string contactInformation;

string medicalHistory;

string allergies;

}

uint256 public patientCount;

mapping (uint256 => Patient) public patients;

event PatientAdded (uint256 id, string name, uint256 dateOfBirth, string gender, string contactInformation);

function addPatient(string memory \_name, uint256 \_dateOfBirth, string memory \_gender, string memory \_contactInformation, string memory \_medicalHistory, string memory \_allergies) public {

patientCount++;

patients[patientCount] = Patient(patientCount, \_name, \_dateOfBirth, \_gender, \_contactInformation, \_medicalHistory, \_allergies);

emit PatientAdded(patientCount, \_name, \_dateOfBirth, \_gender, \_contactInformation);

}

**function** getPatient(uint256 \_id) public view returns (string memory, uint256, string memory, string memory, string memory, string memory) {

require(\_id > 0 && \_id <= patientCount, "Invalid patient ID");

Patient memory patient = patients[\_id];

return (patient.name, patient.dateOfBirth, patient.gender, patient.contactInformation, patient.medicalHistory, patient.allergies);

}

}

The provided Solidity code defines a smart contract named PatientRegistry that serves as a basic patient information management system on the Ethereum blockchain. Let's break down this code step by step:

* pragma solidity ^0.8.0;: This line specifies the version of the Solidity compiler that should be used.
* contract PatientRegistry { ... }: This defines the PatientRegistry smart contract.
* struct Patient { ... }: This is a data structure that represents a patient's information, including their ID, name, date of birth, gender, contact information, medical history, and allergies. This structure is used to store patient records.
* uint256 public patientCount;: This state variable keeps track of the total number of patients in the registry.
* mapping (uint256 => Patient) public patients;: This mapping associates patient IDs (of type uint256) with their respective Patient records. It allows you to retrieve patient information by their ID.
* event PatientAdded (uint256 id, string name, uint256 dateOfBirth, string gender, string contactInformation);: This event is emitted when a new patient is added to the registry. It provides important details of the added patient.
* function addPatient(string memory \_name, uint256 \_dateOfBirth, string memory \_gender, string memory \_contactInformation, string memory \_medicalHistory, string memory \_allergies) public { ... }: This function is used to add a new patient to the registry. It takes patient information as arguments and emits the PatientAdded event.
* patientCount++;: This line increments the patientCount to keep track of the total number of patients.
* patients[patientCount] = Patient(...);: This line adds a new patient to the patients mapping by associating the patient's ID with their information.
* function getPatient(uint256 \_id) public view returns (string memory, uint256, string memory, string memory, string memory, string memory) { ... }: This function allows you to retrieve a patient's information by providing their ID. It checks if the ID is valid and returns the patient's details as a tuple.
* require(\_id > 0 && \_id <= patientCount, "Invalid patient ID");: This line ensures that the provided patient ID is within the valid range of IDs stored in the registry. If the ID is out of range, it throws an error.
* Patient memory patient = patients[\_id];: This line retrieves the patient's information from the patients mapping based on the provided ID.
* return (patient.name, patient.dateOfBirth, patient.gender, patient.contactInformation, patient.medicalHistory, patient.allergies);: This returns the patient's information as a tuple.

In summary, this smart contract, PatientRegistry, allows the addition of patient records and retrieval of patient information based on their unique ID. It's a basic example of how blockchain can be used to manage sensitive healthcare data securely and transparently. Please note that in real-world scenarios, more features, access control mechanisms, and security measures should be considered for the protection of patient data and privacy. In addition, design for reusable smart contracts can boost the productivity and sustainability goals of blockchain. By separating the interface from the actual implementation, you can create other contracts that adhere to the same patient record structure and can use the IRecord interface for interoperability and reusability. For instance, if you want to create another contract that interacts with patient records, you can implement the same IRecord interface and work with patient data seamlessly. This approach promotes code reusability and maintainability.