FurSure Protocol Framework

Protocol Introduction

We have created the concept and framework for a blockchain system named **FurSure**, which will be an **ERC-20 smart contract** hosted on an existing decentralised blockchain network. FurSure serves as a **permissioned platform** for multiple parties: **Owners, Animal Shelters, Vets, and Pet Stores**. It operates through a reward-based token called *FurCoin* (symbol: FUR).

Specifics of functions and permissions will be detailed under the **Protocol Design** section. For now, we will give a high-level overview of FurSure.

Animal shelters will utilise the system to **register animals**, initialising a unique address identifier and medical record for each pet. Once a prospective adopter passes the shelter's existing regulatory checks, the shelter may **transfer ownership of the animal**. The unique address of a pet will be stored in its microchip, which is already common practice in the UK and many other countries.

Pet owners will engage with the system and **earn rewards** from veterinary practices for attending regular health checkups. This incentivises owners to prioritise the well-being of their pets, ensuring proper care. Additionally, the system includes the option for **granting trusted carers**, such as petsitters, or family members who may be looking after pets temporarily.

To address the issue of elderly or sick animals often being overlooked in shelters, we allow the shelters to set a **higher reward level for such animals**. This incentivises adopters to consider giving these end-of-life animals a home for their final years, with the system subsidising expenses.

These **rewards**, in the form of FUR, can be redeemed for **discounts at participating pet care stores** that sponsor the system. Sponsored stores contribute capital to maintain the system, and in return, FurCoin can be used at their stores to apply discounts. FurCoins may also be rewarded through spending at these stores. As our system grows, so does the market share of our sponsors, creating a mutually beneficial relationship.

An important feature of FurSure is the **Lost and Found system**, allowing pet owners to register a pet as lost on the system. If found by a member of the public, it will return the details of the registered owner, only if the pet has been registered as lost, protecting the owner from malicious use of this function. Alternatively, if the pet is taken to a vet or shelter, these trusted parties can locate the owner at any time.

Pre-existing Concepts

There are currently no successful implementations of our idea, though there have been several attempts. Each attempt or concept has not been focused on **improving the wellbeing of animals**, or gone further than designing a few basic functions.

For example, **Pawtocol (2018)**, a Canadian company, had a website with broken links and little recent activity. Their blockchain had only four main functions, including a pet NFT marketplace and a blockchain pet tag. The project appears to have failed.

We found a GitHub repo called **"Adopt-Pet"** (Swarnendu0123, 2024) created by another student, claiming to provide safe pet adoption. However, contract code was basic and insufficient, suggesting another failed attempt. These examples reassure us that there are no fully developed systems in this space, leaving room for FurSure to succeed where others have faltered.

Additionally, there has been a research paper titled **"Adoption of Pets in Distributed Network using Blockchain Technology"** Gururaj et al. (2020), which provides a theoretical basis for a pet adoption marketplace in the form of a website. However, it is simply just a secure marketplace concept, with no extensions/functionalities beyond that or regarding well-being and functions beyond the adoption process.

Business Strategy

As a **nonprofit**, FurSure's mission is **promoting responsible pet ownership**, and to **improve welfare of animals**, particularly those often overlooked in shelters. To sustain operations, we collaborate with sponsor pet stores. **Sponsors benefit** from the increased visibility and customer engagement, as owners use FUR earned through the platform to redeem discounts on pet products and services.

Our **reward-token system** incentivises pet adoption and care by **prioritising animals in need**, such as elderly or sick pets, through higher reward levels. By building strong partnerships with shelters, vets, and sponsors, FurSure creates a **self-sustaining ecosystem** that encourages support for animal welfare while ensuring transparency and accountability through blockchain technology.

Permissioned Design

A **permissioned blockchain** is perfect for FurSure because it ensures sensitive data, like veterinary records, stays private while still allowing transparency where it matters.

For example, public-facing information such as **adoption history** can remain accessible to ensure trust, but private data, like medical records, is restricted to authorised parties such as shelters and veterinarians. This balance is key to making the platform both secure and transparent.

Blockchain technology itself is essential for this project because it solves key challenges in pet adoption that traditional systems can't address. Ensuring records like **adoption histories and health data** are tamper-proof and verifiable, building trust between adopters, shelters, and sponsors. Blockchain eliminates the need for a central authority to manage these records, reducing overhead costs and creating a more equitable system. Its decentralised nature allows all stakeholders to access the same reliable data in real time, ensuring transparency and accountability throughout the process.

By limiting participation to **verified parties**, such as certified shelters and licensed vets, a permissioned blockchain guarantees the credibility of everyone involved. This kind of control isn't possible with public blockchains, where anyone can access and interact with the system. For a platform handling private records and fostering trust between adopters and shelters, this is essential.

Permissioned blockchains also make **compliance with regulations** easier, as access controls can be precisely managed. They allow the system to **scale efficiently**, with faster transactions and smoother operations compared to public blockchains. This ensures that vital actions, like transferring pet ownership or updating health records, happen quickly and securely.

Ultimately, the **permissioned design** gives FurSure the best of both worlds: **transparency for public trust** and **privacy for sensitive data**. Combined with blockchain technology's ability to create **immutable, decentralised records**, this system is uniquely positioned to transform pet adoption by making it secure, efficient, and trustworthy.

Protocol Design

Parties					
Permissioned					Default
System Manager(0)	Vet(1)	Sponsored Store(2)	Animal Shelter(3)	Owner(4)	General Public(5)

Figure 1: Protocol Parties

Functions		
Who?	**Function**	**Purpose**
Blockchain Manager	assignPermissionedRole	Assigning permissioned roles to parties (1,2,3) | *Emits to blockchain*
Blockchain Manager	revokePermissionedRole	Revoking permissioned roles from parties -> (5) | *Emits to blockchain*
Anyone	findOwner	Identify owner of pet ONLY-IF pet registered as lost
Anyone	totalSupply	Return total supply of FurCoin
Anyone	balances	Return balance of an address
Owner	registerCarer	Assign family member/ pet-sitter in care of pet | *Emits to blockchain*
Owner	deregisterCarer	Revoke registered carer from a pet assignment | *Emits to blockchain*
Owner	lostPet	Flag pet as lost | *Emits to blockchain*
Owner	foundPet	Flagging found pet | *Emits to blockchain*
Owner	transferCoin	Transferring coins to others | *Emits to blockchain*
Owner, Carer, Vet	viewRecord	Accessing pet's medical record & details
Vet, Owner (if Shelter)	amendRecord	Amending pet's medical record & details
Shelter	registerPet	Registers pet to system with relevant information | *Emits to blockchain*
Shelter	approveAsOwner	Approving user the ability to adopt | *Emits to blockchain*
Shelter	transferPet	Assigns pet to approved Owner
Shelter	changeRewardLevel	Assigns reward level multiplier to pet (default 1)
Vets & Shelter	getOwner	Identify owner of pet
Sponsor & Vet	petRewardLevel	Returns reward multiplier of pet
Sponsor & Vet	giveReward	Rewarding for checkups & spending
Sponsor	spentReward	Burning tokens spent

Figure 2: Protocol Functions

The FurSure protocol defines **clear, role-based interactions** among parties, leveraging the functionality of the blockchain to ensure **transparency**. Key features include **pet registration**, **ownership transfers**, and **caregiving permission** (Figure 2).

Shelters register pets using `registerPet` and approve owners through `approveAsOwner` and `transferPet`. Temporary caregivers are facilitated via `registerCarer` and `deregisterCarer`, enabling owners to **delegate responsibilities** to trusted individuals.

Lost pets are managed using `lostPet`, `foundPet`, and `findOwner`, allowing blockchain users to **locate and reconnect with owners** using the microchip data.

Reward mechanisms incentivise adoption and care: shelters can adjust a pet's reward level (increased rewards for elderly or sick pets) through `changeRewardLevel`, while vets and sponsors issue tokens through `giveReward`. Tokens are redeemed at sponsor stores, which burn them using `spentReward` to maintain **circulation balance**.

Permission management is handled by the blockchain manager through `assignPermissionedRol` and `revokePermissionedRole` (Figure 1). General utilities include `totalSupply` and `balances` for monitoring FurCoin activity, while `viewRecord` and `amendRecord` allow authorised parties to **access or update pet records securely**.

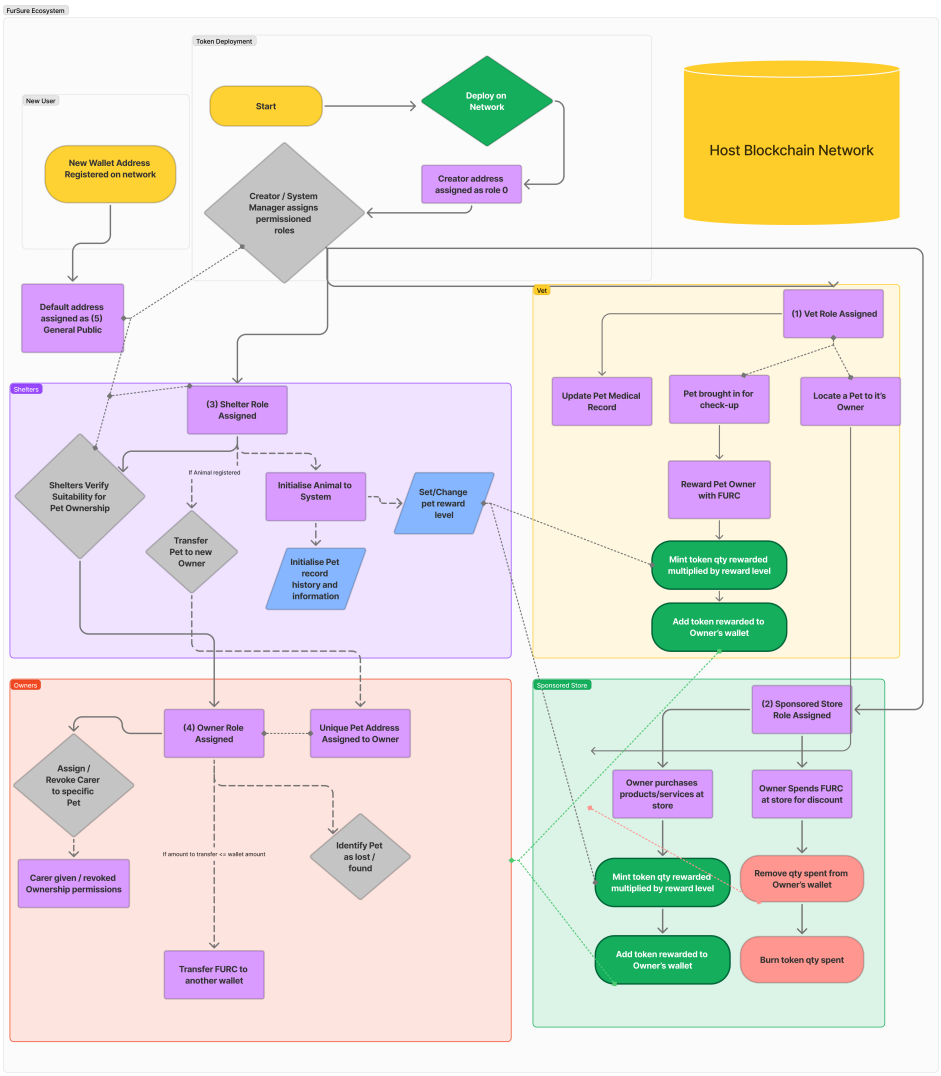

Figure 3: FurSure Ecosystem Design Flowchart

Host Network

We chose to host **FurCoin** on the **Base network**. Base is a **Layer 2 blockchain** built on the **Optimism stack** and secured by Ethereum, offering **fast and cost-efficient transactions** while maintaining the **security and decentralisation** of the Ethereum mainnet. As an Ethereum-compatible network, it fully supports **ERC-20 tokens and smart contracts**.

We chose Base because its **low gas fees and fast transaction times** make it ideal for FurSure, where frequent interactions, such as reward distributions and token redemptions, are central.

As of December 2024, the **average transaction fee** on Ethereum is approximately $6.7 per transaction (Bitbond, 2024). In contrast, Base, a Layer-2 solution built on the Optimism stack, provides substantially **lower fees**, averaging around $0.0011 per transaction (ibid).

By minimising transaction costs, Base reduces the **financial burden** on shelters, adopters, and sponsors, encouraging **wider participation**. Additionally, Base's compatibility with Ethereum ensures that FurCoin remains **interoperable with other Ethereum-based applications**, expanding its potential use cases while benefiting from the **security and decentralisation** of Ethereum mainnet.

Governance

For the governance protocol, a **Decentralised Off-Chain Governance Protocol** will be used as it can balance **inclusivity, efficiency, and practicality**. This takes the form of **Role-Based Participation**, where stakeholders are grouped into roles with the following responsibilities:

- **Veterinarians:** Act as Animal Health Experts.
- **Shelters:** Represent Pet Welfare.
- **Adopters:** Advocate for pet ownership.

Each group has **voting power proportional to their stake** in the system:

- **Veterinarians:** 40% weight, providing expert insights into health policies.
- **Shelters:** 40% weight, reflecting their direct impact on operations and pet welfare.
- **Adopters:** 20% weight, contributing community-driven feedback.

Any stakeholder can **propose changes** (e.g., new policies), and then **votes are cast** on a secure off-chain platform. Results are **aggregated and verified** via the weighted voting system. Once a proposal passes, platform administrators implement changes directly to the protocol.

To illustrate further, let's say a **sponsor pet food brand** proposes to join FurSure, offering some discounts in exchange for FUR. This proposal includes terms and conditions for all parties.

Veterinarians review the pet food brand's products to ensure they meet health and nutrition standards, **shelters** assess the partnership's potential benefits such as increased

adoption rates or funding opportunities, and **adopters** evaluate the convenience and value of the proposed discount.

Each stakeholder group casts votes aligned with their roles: veterinarians focus on product quality, shelters on operational advantages, and adopters' personal incentives. Votes are **weighted**, with veterinarians and shelters each holding 40% influence and adopters 20%. If the proposal reaches the **approval threshold**, the governance council ratifies the decision, and the platform integrates the sponsor's discount offer. Adopters can then use their **FurCoins for discounts**, enhancing the ecosystem's utility and value.

Protocol Testing

In order to test the **functionality and rigorous defensibility** of the smart contract's protocol, we deployed the use of **Hardhat tests**. We ran two main tests which we felt were crucial to pass for the smart contract.

General Lifecycle Test:

We devised a **step-by-step flow**, which we would expect an average pet to follow on our network as intended (Figure 3). This ensures that they interact with each party and make use of all intended functions. From the **inception of the pet's unique ID** via the shelters, through adoption, and then routine checkups and spending on supplies. Even testing whether the **Lost & Found function works as intended**.

Protocol Defense Testing:

In this test, we brainstormed a few **extremely critical functions** that, if someone were able to exploit them, would cripple the system. This ranged from users trying to **mint new money**, someone effectively **stealing an already registered pet**, to owners trying to **fraud rewards**.

Of course, there are many more potential exploits. However, our implementation of **modifiers** made restricting functions to specific groups or parties very easy and foolproof in the source code for the contract.

```
PS [redacted]\FurSure> npx mocha .\test\lifecycletest.js/


  FurCoin Lifecycle Test
    ✓ Should deploy FurCoin contract
    ✓ Creator should be able to assign permissioned roles
    ✓ Shelter should be able to register a pet and set reward level
    ✓ Shelter should be able to transfer a pet to an owner
    ✓ Vet should be able to provide a multiplied reward to a pet's owner
    ✓ Vet should be able to amend a pet record
    ✓ Owner should be able to spend FUR tokens at a store (Store Burns & Mints according)
    ✓ Owner should be able to flag a pet as lost
    ✓ Member of public should be able to find a lost pet and receive owner's contact number
    ✓ Owner should be able to register a pet as found and the phone number is no longer visible
    ✓ Owner should be able to transfer tokens to a member of the public as a reward
    ✓ Owner should be able to assign and revoke carer roles

  Defended Against Malicous Attacks Test
    ✓ Unauthorised user trying to mint
    ✓ Shelter attempting to use giveReward (mint) FURC
    ✓ Trying to find owner of a pet that is not registered as lost
    ✓ Member of public trying to register themselves as carer or owner
    ✓ Owner trying to give themselves a reward


  17 passing (1s)
```

Figure 4: Hardhat Testing

References

- **Bitbond (2024)** Ethereum Gas Price: Bitbond, Token Tool by Bitbond. Available at: https://tokentool.bitbond.com/gas-price/ethereum (Accessed: 04 December 2024).
- **Gururaj, H.L. et al. (2020)** 'Adoption of pets in distributed network using blockchain technology', *International Journal of Blockchains and Cryptocurrencies*, 1(2), p. 107. doi:10.1504/ijbc.2020.108996.
- **Pawtocol (2018)** Medium. Available at: https://pawtocol.medium.com/ (Accessed: 02 December 2024).
- **Swarnendu0123 (2024)** *SWARNENDU0123/adopt-pet: A blockchain based pet adopting platform.*, GitHub. Available at: https://github.com/Swarnendu0123/adopt-pet (Accessed: 29 November 2024).

Code Implementation (Word Limit Excluded)

Repository Address:
https://github.com/jackthompsondb/FurSure

Smart Contract in Solidity:
The smart contract is implemented in Solidity and includes all functionalities described in the protocol design.

Hardhat Tests:
Tests have been implemented using Hardhat

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.28;

// Info:
// Roles: Creator (0), Vet (1), Sponsor (2), Shelter (3), Owner (4),
General Public (5 or "")

contract FurCoin {
    // initial supply of token

    uint256 public supply = 1000000;
    string public name = "FurCoin";
    string public symbol = "FUR";
    uint8 public decimals = 1;

    // Token-related event
    event TokenTransferred(address indexed _from, address indexed _to,
uint256 _value);

    // Approval-related events
    event OwnerApproved(address indexed _shelter, address indexed
_newOwner);
    event PermissionedRoleApproved(address indexed _from, address indexed
_to, uint256 _role);
    event PermissionedRoleRevoked(address indexed _from, address indexed
_to, uint256 _role);

    // Pet lifecycle events
    event PetRegistered(address indexed _shelter, uint256 indexed _petId);
    event PetAdopted(address indexed _shelter, address indexed _adopter,
uint256 indexed _petId);
    event CarerRegistered(uint256 indexed petId, address indexed carer);
    event CarerDeregistered(uint256 indexed petId, address indexed carer);
    event PetLost(uint256 indexed _petId);
    event PetFound(uint256 indexed _petId);

    // create mappings
    mapping(address => uint256) public balances; // Maps an address to a
balance
```

```
    mapping(address => uint256) public roles;  // Maps an address to a
role
    mapping(uint256 => uint256) public petRewardLevel; // Maps pet ID to
its reward level
    mapping(uint256 => address) public petOwner; // Maps a pet ID to its
owner
    mapping(uint256 => address) public petCarer; // Maps a pet ID to its
carer
    mapping(uint256 => bool) public petAdopted; // Tracks if a pet ID is
adopted
    mapping(uint256 => bool) public petLost;    // Tracks if a pet ID is
lost
    mapping(uint256 => string) public petLostPhoneNumber; // Maps a pet ID
to a phone number when lost
    mapping(uint256 => PetRecord) private petRecords; // Maps a pet ID to
its record

    // pet record structure
    struct PetRecord {
        string name;
        uint256 age;
        string medicalHistory;
        bool exists; // To ensure the pet record exists before accessing
    }

    // create constructor
    constructor() {
        balances[msg.sender] = supply;
        roles[msg.sender] = 0;
    }

    // create modifiers to optimise function size and reduce redundancy
    modifier onlyCreator() {
        require(roles[msg.sender] == 0, "Only creator can use this
function");
        _;
    }
    modifier onlyVetOrShelter() {
        require(roles[msg.sender] == 1 || roles[msg.sender] == 3, "Only
Vet or Shelter can use this function");
```

```
        _;
    }
    modifier onlyVetOrSponsor() {
        require(roles[msg.sender] == 1 || roles[msg.sender] == 2, "Only
Vet or Sponsor can use this function");
        _;
    }
    modifier onlyShelter() {
        require(roles[msg.sender] == 3, "Only Shelter can use this
function");
        _;
    }
    modifier onlySponsor() {
        require(roles[msg.sender] == 2, "Only Sponsor Stores can use this
function");
        _;
    }
    modifier onlyAuthorized(uint256 _petId) {
        require(msg.sender == petOwner[_petId] || msg.sender ==
petCarer[_petId] || roles[msg.sender] == 3,
            "Access restricted to Owner, Carer, or Shelter");
        _;
    }

    function mint(uint256 _amount) public onlyCreator {
        // Function for the creator to mint new tokens
        supply += _amount;
        balances[msg.sender] += _amount;
        emit TokenTransferred(address(0), msg.sender, _amount);
    }

    function burn(address _from, uint256 _amount) public onlyCreator {
        // Function for the creator to burn tokens
        require(balances[_from] >= _amount, "Insufficient balance to
burn");
        supply -= _amount;
        balances[_from] -= _amount;
        emit TokenTransferred(_from, address(0), _amount);
    }
```

```
    function assignPermissionedRole(address _address, uint256 _role)
public onlyCreator {
        // Function for system manager to assign roles permissioned roles
        // such as Vet, Sponsor, Shelter or anything else
        require(_role >= 0 && _role <= 5, "Invalid role");
        roles[_address] = _role;
        emit PermissionedRoleApproved(msg.sender, _address, _role);
    }

    function revokePermissionedRole(address _address) public onlyCreator {
        // Function for system manager to revoke permissioned roles
        // such as Vet, Sponsor, Shelter or anything else
        uint256 previous_role = roles[_address];
        roles[_address] = 5;
        emit PermissionedRoleRevoked(msg.sender, _address, previous_role);
    }

    function getOwner(uint256 _petId) public onlyVetOrShelter view returns
(address) {
        // Function to check the owner of a pet
        return petOwner[_petId];
    }

    function registerPet(uint256 _petId, string memory _name, uint256
_age, string memory _medicalHistory) public onlyShelter {
        require(!petRecords[_petId].exists, "Pet is already registered");
        // Function for Shelters to register a new pet
        petOwner[_petId] = msg.sender;
        petAdopted[_petId] = false;
        petRewardLevel[_petId] = 1;

        // Initialize a basic pet record
        petRecords[_petId] = PetRecord({
            name: _name,
            age: _age,
            medicalHistory: _medicalHistory,
            exists: true
        });

        // Emit events for registration and record initialization
```

```
        emit PetRegistered(msg.sender, _petId);
    }

    function viewPetRecord(uint256 _petId) public view
onlyAuthorized(_petId) returns (PetRecord memory) {
        require(petRecords[_petId].exists, "Pet record does not exist");
        return petRecords[_petId];
    }

    function amendPetRecord(uint256 _petId, string memory _name, uint256
_age, string memory _medicalHistory) public onlyVetOrShelter {
        require(petRecords[_petId].exists, "Pet record does not exist");
        // Function for Vets to amend a pet's record
        petRecords[_petId].name = _name;
        petRecords[_petId].age = _age;
        petRecords[_petId].medicalHistory = _medicalHistory;
    }

    function changeRewardLevel(uint256 _petId, uint256 _rewardLevel)
public onlyShelter {
        // Function for Shelters to change the reward level of a pet
        petRewardLevel[_petId] = _rewardLevel;
    }

    function approveAsOwner(address _allowedOwner) public onlyShelter {
        // Function for Shelters to approve a new owner so they can adopt
a pet
        require(roles[_allowedOwner] != 0, "Cannot change system manager's
role");
        require(roles[_allowedOwner] != 1, "Cannot change Vet's role");
        require(roles[_allowedOwner] != 2, "Cannot change Sponsor's
role");
        roles[_allowedOwner] = 4;
        emit OwnerApproved(msg.sender, _allowedOwner);
    }

    function transferPet(uint256 _petId, address _newOwner) public
onlyShelter {
        // Function for Shelters to transfer ownership of a pet to a new
owner
```

```
        require(petOwner[_petId] == msg.sender, "Only the owner can
transfer the pet");
        petOwner[_petId] = _newOwner;
        petAdopted[_petId] = true;
        emit PetAdopted(msg.sender, _newOwner, _petId);
    }

    function registerCarer(uint256 _petId, address _carer) public {
        // Function for Owners to register a carer for a pet
        require(petOwner[_petId] == msg.sender, "Only the owner can
register a carer");
        petCarer[_petId] = _carer;
        emit CarerRegistered(_petId, _carer);
    }

    function deregisterCarer(uint256 _petId, address _carer) public {
        // Function for Owners to deregister a carer for a pet
        require(petOwner[_petId] == msg.sender, "Only the owner can
deregister a carer");
        petCarer[_petId] = address(0);
        emit CarerDeregistered(_petId, _carer);
    }

    function lostPet(uint256 _petId,string memory _phonenumber) public {
        // Function for Vets or Shelters to mark a pet as lost
        require(petOwner[_petId] == msg.sender, "Only the owner can mark a
pet as lost");
        petLost[_petId] = true;
        petLostPhoneNumber[_petId] = _phonenumber;
        emit PetLost(_petId);
    }

    function foundPet(uint256 _petId) public {
        require(petOwner[_petId] == msg.sender, "Only the owner can mark a
pet as found");
        petLost[_petId] = false;
        petLostPhoneNumber[_petId] = "";
        emit PetFound(_petId);
    }
```

```
    function findOwner(uint256 _petId) public view returns (address) {
        // Function for General Public to find the owner of a lost pet
        require(petLost[_petId] == true, "Pet is not lost");
        return petOwner[_petId];
    }

    function getRewardLevel(uint256 _petId) public view onlyVetOrSponsor
returns (uint256) {
        // Function for General Public to find the reward level of a pet
        return petRewardLevel[_petId];
    }

    function giveReward(uint256 _petId, uint256 _value) public
onlyVetOrSponsor {
        // Function for Vets or Sponsors to give a token reward to a pet's
owner
        uint256 multiplier = petRewardLevel[_petId];
        uint256 reward = _value * multiplier;
        supply += reward;
        balances[petOwner[_petId]] += reward;
        emit TokenTransferred(msg.sender, petOwner[_petId], reward);
    }

    function spendReward(address _owner, uint256 _value) public
onlySponsor {
        // Function for Owners to spend their token rewards
        require(balances[_owner] >= _value, "Insufficient balance");
        supply -= _value;
        balances[_owner] -= _value;
        emit TokenTransferred(_owner, address(0), _value);
    }

    function transfer(address _to, uint256 _value) public {
        // Function for Owners to transfer tokens to another address
        require(balances[msg.sender] >= _value, "Insufficient balance");
        balances[msg.sender] -= _value;
        balances[_to] += _value;
        emit TokenTransferred(msg.sender, _to, _value);
    }
```

```
    function getBalance(address _address) public view returns (uint256) {
        // Function for General Public to check their token balance
        return balances[_address];
    }
}
```

```
const { expect } = require("chai");
const { ethers } = require("hardhat");

describe("FurCoin Lifecycle Test", function () {
  let creator, vet, sponsor, shelter, owner, carer, finder, publicAccount;
  let furCoin;

  beforeEach(async function () {
    // Deploy FurCoin contract
    const FurCoin = await ethers.getContractFactory("FurCoin");
    furCoin = await FurCoin.deploy();
    await furCoin.waitForDeployment();

    // Get signers
    [creator, vet, sponsor, shelter, owner, carer, finder, publicAccount]
= await ethers.getSigners();

    // Assign roles
    await furCoin.connect(creator).assignPermissionedRole(shelter.address,
3); // Assign Shelter role
    await furCoin.connect(creator).assignPermissionedRole(vet.address, 1);
// Assign Vet role
    await furCoin.connect(creator).assignPermissionedRole(sponsor.address,
2); // Assign Sponsor role
    await furCoin.connect(creator).assignPermissionedRole(owner.address,
4); // Assign Owner role
    await furCoin.connect(creator).assignPermissionedRole(carer.address,
5); // Assign Member of Public role
  });

  it("Should deploy FurCoin contract", async function () {
    expect(furCoin.target).to.be.properAddress;
  });

  it("Creator should be able to assign permissioned roles", async function
() {
    await furCoin.connect(creator).assignPermissionedRole(vet.address, 1);
// Assign Vet role
    const vetRole = await furCoin.roles(vet.address);
```

```
    expect(vetRole).to.equal(1);

    await furCoin.connect(creator).assignPermissionedRole(sponsor.address,
2); // Assign Sponsor role
    const sponsorRole = await furCoin.roles(sponsor.address);
    expect(sponsorRole).to.equal(2);

    await furCoin.connect(creator).assignPermissionedRole(shelter.address,
3); // Assign Shelter role
    const shelterRole = await furCoin.roles(shelter.address);
    expect(shelterRole).to.equal(3);
  });

  it("Shelter should be able to register a pet and set reward level",
async function () {
    // Register a pet
    await furCoin.connect(shelter).registerPet(1, "Furry", 12, "Partially
blind in left eye");
    const petOwner = await furCoin.connect(shelter).getOwner(1);
    expect(petOwner).to.equal(shelter.address);

    // Set reward level for the pet
    await furCoin.connect(shelter).changeRewardLevel(1, 5);

    // View reward level for the pet as a vet
    const rewardLevelVet = await furCoin.connect(vet).getRewardLevel(1);
    expect(rewardLevelVet).to.equal(5);

    // View reward level for the pet as a sponsor
    const rewardLevelSponsor = await
furCoin.connect(sponsor).getRewardLevel(1);
    expect(rewardLevelSponsor).to.equal(5);
  });

  it("Shelter should be able to transfer a pet to an owner", async
function () {
    // Register a pet
    await furCoin.connect(shelter).registerPet(2, "Buddy", 3, "Healthy");
    const initialPetOwner = await furCoin.connect(shelter).getOwner(2);
    expect(initialPetOwner).to.equal(shelter.address);
```

```
    // Transfer the pet to the owner
    await furCoin.connect(shelter).transferPet(2, owner.address);
    const newPetOwner = await furCoin.connect(shelter).getOwner(2);
    expect(newPetOwner).to.equal(owner.address);
  });
  // Add more tests specific to FurCoin functionality here
  it("Vet should be able to provide a multiplied reward to a pet's owner",
async function () {
    // Register a pet and transfer to owner
    await furCoin.connect(shelter).registerPet(3, "Max", 10, "Deaf");
    await furCoin.connect(shelter).changeRewardLevel(3, 2);
    await furCoin.connect(shelter).transferPet(3, owner.address);

    // Vet provides reward to the pet's owner
    await furCoin.connect(vet).giveReward(3, 100);
    const ownerBalance = await furCoin.getBalance(owner.address);
    expect(ownerBalance).to.equal(200);
  });

  it("Vet should be able to amend a pet record", async function () {
    // Register a pet
    await furCoin.connect(shelter).registerPet(4, "Bella", 5, "Healthy");

    // Vet amends the pet record
    await furCoin.connect(vet).amendPetRecord(4, "Bella", 6, "Updated
medical history");

    // Verify the amended pet record
    const petRecord = await furCoin.connect(shelter).viewPetRecord(4);
    expect(petRecord.name).to.equal("Bella");
    expect(petRecord.age).to.equal(6);
    expect(petRecord.medicalHistory).to.equal("Updated medical history");
  });
  it("Owner should be able to spend FUR tokens at a store (Store Burns &
Mints according)", async function () {
    // Register a pet and transfer to owner
    await furCoin.connect(shelter).registerPet(5, "Charlie", 4,
"Healthy");
    await furCoin.connect(shelter).transferPet(5, owner.address);
```

```
    // Vet provides reward to the pet's owner
    await furCoin.connect(vet).giveReward(5, 100);
    let ownerBalance = await furCoin.getBalance(owner.address);
    expect(ownerBalance).to.equal(100);

    // Owner spends FUR tokens at a store
    await furCoin.connect(sponsor).spendReward(owner.address, 50);
    ownerBalance = await furCoin.getBalance(owner.address);
    expect(ownerBalance).to.equal(50);
  });
  it("Owner should be able to flag a pet as lost", async function () {
    // Register a pet and transfer to owner
    await furCoin.connect(shelter).registerPet(6, "Luna", 3, "Healthy");
    await furCoin.connect(shelter).transferPet(6, owner.address);

    // Owner flags the pet as lost
    await furCoin.connect(owner).lostPet(6, "123-456-7890");

    // Verify the pet is flagged as lost
    const isLost = await furCoin.petLost(6);
    expect(isLost).to.be.true;

    // Verify the lost pet phone number
    const lostPhoneNumber = await furCoin.petLostPhoneNumber(6);
    expect(lostPhoneNumber).to.equal("123-456-7890");
  });
  it("Member of public should be able to find a lost pet and receive
owner's contact number", async function () {
    // Register a pet and transfer to owner
    await furCoin.connect(shelter).registerPet(7, "Rocky", 2, "Healthy");
    await furCoin.connect(shelter).transferPet(7, owner.address);

    // Owner flags the pet as lost
    await furCoin.connect(owner).lostPet(7, "123-456-7890");

    // Member of public finds the lost pet
    const petOwner = await furCoin.connect(carer).findOwner(7);
    expect(petOwner).to.equal(owner.address);
```

```
    // Verify the lost pet phone number
    const lostPhoneNumber = await furCoin.petLostPhoneNumber(7);
    expect(lostPhoneNumber).to.equal("123-456-7890");
  });
  it("Owner should be able to register a pet as found and the phone number
is no longer visible", async function () {
    // Register a pet and transfer to owner
    await furCoin.connect(shelter).registerPet(8, "Shadow", 5, "Healthy");
    await furCoin.connect(shelter).transferPet(8, owner.address);

    // Owner flags the pet as lost
    await furCoin.connect(owner).lostPet(8, "123-456-7890");

    // Verify the pet is flagged as lost
    let isLost = await furCoin.petLost(8);
    expect(isLost).to.be.true;

    // Owner registers the pet as found
    await furCoin.connect(owner).foundPet(8);

    // Verify the pet is no longer flagged as lost
    isLost = await furCoin.petLost(8);
    expect(isLost).to.be.false;

    // Verify the lost pet phone number is no longer visible
    const lostPhoneNumber = await furCoin.petLostPhoneNumber(8);
    expect(lostPhoneNumber).to.equal("");
  });
  it("Owner should be able to transfer tokens to a member of the public as
a reward", async function () {
    // Register a pet and transfer to owner
    await furCoin.connect(shelter).registerPet(9, "Buddy", 3, "Healthy");
    await furCoin.connect(shelter).transferPet(9, owner.address);

    // Vet provides reward to the pet's owner
    await furCoin.connect(vet).giveReward(9, 100);
    let ownerBalance = await furCoin.getBalance(owner.address);
    expect(ownerBalance).to.equal(100);

    // Owner transfers tokens to a member of the public
```

```
    await furCoin.connect(owner).transfer(carer.address, 50);
    ownerBalance = await furCoin.getBalance(owner.address);
    expect(ownerBalance).to.equal(50);

    const carerBalance = await furCoin.getBalance(carer.address);
    expect(carerBalance).to.equal(50);
  });
  it("Owner should be able to assign and revoke carer roles", async
function () {
    // Register a pet and transfer to owner
    await furCoin.connect(shelter).registerPet(10, "Max", 4, "Healthy");
    await furCoin.connect(shelter).transferPet(10, owner.address);

    // Owner assigns a carer role
    await furCoin.connect(owner).registerCarer(10, carer.address);
    let assignedCarer = await furCoin.petCarer(10);
    expect(assignedCarer).to.equal(carer.address);

    // Owner revokes the carer role
    await furCoin.connect(owner).deregisterCarer(10, carer.address);

    // Verify the carer's role is set to 5 (General Public)
    const carerRole = await furCoin.roles(carer.address);
    expect(carerRole).to.equal(5);
  });
});

describe("Defended Against Malicous Attacks Test", function () {
  let creator, vet, sponsor, shelter, owner, carer, finder, publicAccount;
  let furCoin;

  beforeEach(async function () {
    // Deploy FurCoin contract
    const FurCoin = await ethers.getContractFactory("FurCoin");
    furCoin = await FurCoin.deploy();
    await furCoin.waitForDeployment();

    // Get signers
    [creator, vet, sponsor, shelter, owner, carer, finder, publicAccount]
= await ethers.getSigners();
```

```
    // Assign roles
    await furCoin.connect(creator).assignPermissionedRole(shelter.address,
3); // Assign Shelter role
    await furCoin.connect(creator).assignPermissionedRole(vet.address, 1);
// Assign Vet role
    await furCoin.connect(creator).assignPermissionedRole(sponsor.address,
2); // Assign Sponsor role
    await furCoin.connect(creator).assignPermissionedRole(owner.address,
4); // Assign Owner role
    await furCoin.connect(creator).assignPermissionedRole(carer.address,
5); // Assign Member of Public role
  });
  it("Unauthorised user trying to mint", async function () {
    await
expect(furCoin.connect(carer).mint(1000)).to.be.revertedWith("Only creator
can use this function");
  });

  it("Shelter attempting to use giveReward (mint) FURC", async function ()
{
    await furCoin.connect(shelter).registerPet(1, "Buddy", 3, "Healthy");
    await furCoin.connect(shelter).transferPet(1, owner.address);
    await expect(furCoin.connect(shelter).giveReward(1,
100)).to.be.revertedWith("Only Vet or Sponsor can use this function");
  });

  it("Trying to find owner of a pet that is not registered as lost", async
function () {
    await furCoin.connect(shelter).registerPet(2, "Max", 4, "Healthy");
    await furCoin.connect(shelter).transferPet(2, owner.address);
    await
expect(furCoin.connect(carer).findOwner(2)).to.be.revertedWith("Pet is not
lost");
  });

  it("Member of public trying to register themselves as carer or owner",
async function () {
    await furCoin.connect(shelter).registerPet(3, "Luna", 2, "Healthy");
    await furCoin.connect(shelter).transferPet(3, owner.address);
```

```
    await expect(furCoin.connect(carer).registerCarer(3,
carer.address)).to.be.revertedWith("Only the owner can register a carer");
  });

  it("Owner trying to give themselves a reward", async function () {
    await furCoin.connect(shelter).registerPet(4, "Rocky", 5, "Healthy");
    await furCoin.connect(shelter).transferPet(4, owner.address);
    await expect(furCoin.connect(owner).giveReward(4,
100)).to.be.revertedWith("Only Vet or Sponsor can use this function");
  });
});
```