APS1050H F LEC0101 20259:Blockchain Technologies and Cryptocurrencies

Final Project (Non-tutorial Version) Petshop: Executive Summary

Introduction

Our DApp is an enhanced version of the classic Pet Shop DApp (Pete's Pet Shop) decentralized application, built on Ethereum using Solidity, Web3.js, and a fully on-chain smart contract architecture. It allows users to view, adopt, and interact with pets while ensuring transparency, immutability, and trust through blockchain technology. For our DApp, **7 features have been implemented to adapt the team size of 4 people after discussion with instructors.** All 7 features enrich both the front-end user experience and the back-end smart contract logic, making the DApp more interactive and intuitive. These improvements create a more dynamic, user-driven, and functional platform for decentralized pet adoption.

Feature Implementation

1. a way of adding/registering pets (and their photos url), for a fee.

File: index.html

- Added a dedicated "Register Your Pet" form that collects name, breed, age, location, and picture URL, and maps to the front-end registration logic.
- Added visual styling and input fields to integrate seamlessly with the rest of the app.

File: app.js

- Added handleRegisterPet() to call registerPet() with all form inputs and send the required 0.01 ETH.
- Added getAndDisplayRegisteredPets() to load newly registered pets from the smart contract, assign them IDs, and append them to App.allPets.
- Updated UI rendering functions to show registered pet panels, including the image URL entered by the user.
- Added event binding in bindEvents() for the "Register Pet" button.

File: Adoption.sol

- Added RegisteredPet struct with fields for name, breed, age, location, pictureUrl, owner, and adoption status.
- Implemented registerPet() to allow users to list their own pets for adoption, enforcing a 0.01 ETH registration fee and storing new pets in registeredPets.
- Added getRegisteredPetsCount() and public access to registeredPets for front-end retrieval

2. a way to restrict the hours of operation of the PetShop DApp to business hours (9AM - 9PM EST)

File: app.js:

- Implemented isDuringBusinessHours() to compute time in EST.
- Implemented showBusinessHoursWarning() to alert users during closed hours.
- Added business-hour checks inside handleAdopt() and handleRegisterPet() so adoption/registration is blocked when outside 9 AM–9 PM EST.

3. a way of filtering*** for a list of pets (available pets not adopted already) of a specific breed, age and location etc.

File: index.html

- Added a filter panel including breed selector, location selector, age-range inputs, and status selector ("Not Adopted").
- Included a "Search" and "Reset Filter" button inside the filter panel

File: app.js

- Added applyFilter() to filter the in-memory pet list based on breed, location, age range, and adoption status.
- Added resetFilter() to clear filter fields and reload all pets.
- Added fillFilterOptions() to populate breed and location dropdowns based on current pet data.
- Linked filter buttons to their handlers in bindEvents()

4. a way of liking and unliking a pet in the Petshop, for a fee

File: index.html

- Added Like/Unlike button and Like count UI elements inside each pet card

File: app.js

- Added event binding in bindEvents() for .btn-like to trigger handleToggleLike().
- Implemented handleToggleLike() to call toggleLike() on the contract and refresh UI like counts.
- Added updateLikeDisplay() and refreshAllLikes() to dynamically update displayed like totals.

File: Adoption.sol

- Added mapping(uint => uint) petLikes and mapping(address => mapping(uint => bool)) userLiked to store like counts and per-user like status.
- Implemented toggleLike(uint petId) to charge the like fee and flip a user's like/unlike state, updating the like counter accordingly.
- Added getMyLikedPets() to return all pet IDs liked by the caller.

5. a way of recording and showing the pets liked by the user

File: index.html

- Added a "My Liked Pets" section containing a button that toggles visibility of an expandable container used to display the user's liked pets.

File: app.js

- Implemented showMyLikes() to toggle the "My Liked Pets" section and call the backend.
- Added refreshMyLikesList() to fetch liked pet IDs from getMyLikedPets() and dynamically populate the list UI.

File: Adoption.sol

- Added getMyLikedPets() to return an array of all pet IDs liked by msg.sender.
- Uses the existing userLiked[address][petId] mapping to detect which pets a user has liked.

6. a way of filtering*** for a list of pets (already adopted pets) of a specific breed, age and location etc.

File: index.html

- Included filter panel (breed selector, location selector, age-range inputs, and status selector) with the status option "Adopted Only" available.

File: app.js

- Updated applyFilter() so adoption status = “adopted” filters the in-memory list to only show pets marked as already adopted
- Updated resetFilter() restores all selector fields and reloads all pets, clearing the adopted-only view when needed.
- Updated fillFilterOptions() continues to populate dropdowns based on all available and registered pets so filtering remains consistent.
- Ensured filter buttons remain mapped in bindEvents() so searching and resetting for adopted pets works the same as for available pets.

7 . a way of donating ether to the petshop transferred from Web3Basics SendMeEther, where only the petshop owner can withdraw ether.

File: index.html

- Added a new Donation Card section with ETH input, a “Donate ETH” button, and an Owner Controls area with a “Withdraw All Donations” button.
- Integrated Bootstrap styling for clear separation and usability.

File: app.js

- Added donate() to read the donation amount from user input, convert to Wei, and send ETH to donate() from the connected wallet.
- Added withdraw() to call the owner-only withdrawal function and surface success or failure to users.
- Inserted event bindings in bindEvents() for the donation and withdrawal buttons.

File: Adoption.sol

- Implemented donate() as a payable function that accepts ETH contributions and updates totalDonations.
- Added withdraw() restricted to the contract owner, allowing owners to withdraw all accumulated donations.
- Declared owner in the constructor to store the deployer as the contract administrator

Conclusion

Overall, the deployment of seven features significantly strengthens both the smart-contract layer and the front-end interface, resulting in a more transparent, secure, and user-driven decentralized pet adoption experience.

APS1050H F LEC0101 20259:Blockchain Technologies and Cryptocurrencies

Final Project (Non-tutorial Version) Petshop: Set-Up Instruction

Set-Up Instruction

Required package and version:

Node.js version: v12.13.0
lite-server version: v2.6.1
Solidity version: v0.5.16
web3 version: v1.2.1
Truffle version: v5.1.10
Ganache version: v2.7.1
Front-End Framework: JQuery & Bootstrap

Installation and compilation:

1. Install Truffle globally.

`npm install -g truffle`

2. Compile the project.

`truffle compile`

3. Run the development console and migrate the contract in reset mode.

`truffle migrate -- reset`

4. Run the liteserver development server (outside the development console) for front-end

// Serves the front-end on http://localhost:3000 using browser

`npm run dev` or *`lite-server`*

***Petshop tutorial:** https://archive.trufflesuite.com/guides/pet-shop/