# A Simple Example of TureChain DApp

## Overview:

This article guides you through a pet store tracking system. It is hoped that an address on the ture chain will be associated with the pet to track the adoption of the pet. The main job is to write smart contracts and front-end logic.

It is suitable for people who have some knowledge of HTML and JavaScript, but who are noob to DApp development.

With this simple DApp example, we'll show you:

* Setting up the development environment for DAp
* Creating Truffle projects using Truffle Box
* Writing Intelligent contracts for DApp
* Compile and deploy smart contracts
* Test intelligence contract
* Create a user interface to interact with smart contracts

## Development Environment

There’s the components:

* Development tool: Truffle
* Server: Node.js
* Smart Contract Language: Solidty
* Javascript Library: Web3.js
* TureChain Private Network

Firstly we need to install Node.js as another development tool depens on it.

Run commands in terminal(Example in osx):

brew install node

npm install -g truffle

After the installation is complete, verify that the installation is successful by executing the command truffle version at the terminal.

truffle version

## Creating a project

If this is your first time using truffle, you’ll have to take care of some initial setup. Namely, you’ll need to auto-generate some code that establishes a truffle project - a collection of settings for an instance of Truffle, including smart contracts source, options and migrations settings .

1. Create a project directory

**mkdir pet-shop && cd pet-shop**

1. Auto-genrate project struct

**truffle init**

1. This article uses the existing pet-shop, in Truffle Box to build the basic project structure.

**truffle unbox pet-shop**

Let’s look at what init created:

**contracts/:** contains Solidity source files for smart contracts. One important contract, Migrations.sol, will be discussed later.

**migrations/:** Truffle uses migration systems to handle smart contract deployments. Migration is a deployment script that changes the state of an application contract.

**test/:** contains JavaScript and Solidity testing for Smart contracts

**truffle.js:** Truffle profile Pet Store Truffle Box has additional files and folders.

## Coding contracts

Write smart contracts to implement back-end logic and storage to start DApp.

1. Create a new file called Adoption.sol in the contracts/ directory.
2. Add the following to the file:

pragma solidity ^0.4.17;

contract Adoption {

}

1. Create an array of variables of type adopters, of address and length of 16. Later, we'll write a function to return the entire array for use in our UI.

address[16] public adopters;

1. Create a pet adoption function named adopt

function adopt(uint petId) public returns (uint) {

require(petId >= 0 && petId <= 15);

adopters[petId] = msg.sender;

return petId;

}

1. Create a function to retrieve adopters, getAdopters

function getAdopters() public view returns (address[16]) {

return adopters;

}

The view keyword in the function declaration indicates that the function does not modify the state of the contract.

## Compile and deploy smart contracts

Solidity is a compiled language, which means we need to compile our Solidity into bytecode for execution.

1. In the terminal, make sure that you are in the root directory that contains the DApp directory and enter the: **truffle compile**, similar to the following output:

Compiling ./contracts/Migrations.sol...

Compiling ./contracts/Adoption.sol...

Writing artifacts to ./build/contracts

1. Create a new file called 2\_deploy\_contracts.js in the migrations/ directory. There is a JavaScript file in the migrations/ directory. This handles the deployment of Migrations.sol contracts to observe subsequent smart contract migrations.
2. Add the following to the 2\_deploy\_contracts.js file:

var Adoption = artifacts.require("Adoption");

module.exports = function(deployer) {

deployer.deploy(Adoption);

};

1. Modify the truffle.js, configuration for locally built initial chained private networks.

development: {

host: "127.0.0.1",

port: 7545,

network\_id: "\*" // Match any network id

},

1. deploy the contract to the TrueChain private network. With the command: **Truffle migrate**

## Test smart contract

Truffle is very flexible in smart contract testing because tests can be written in JavaScript or Solidity. In this article, we will write tests using Solidity.

1. Create a new file called TestAdoption.sol in the test/ directory.
2. Add the following to the TestAdoption.sol file:

pragma solidity ^0.4.17;

import "truffle/Assert.sol";

import "truffle/DeployedAddresses.sol";

import "../contracts/Adoption.sol";

contract TestAdoption {

// the test address of contract

Adoption adoption = Adoption(DeployedAddresses.Adoption());

// pet id

uint expectedPetId = 8;

//the adopter of pet

address expectedAdopter = this;

}

1. To test the adopt () function, add the following to the TestAdoption.sol smart contract:

function testUserCanAdoptPet() public {

uint returnedId = adoption.adopt(expectedPetId);

Assert.equal(returnedId, expectedPetId, "Adoption of the expected pet should match what is returned.");

}

1. Test to retrieve the owner of a single pet

function testGetAdopterAddressByPetId() public {

address adopter = adoption.adopters(expectedPetId);

Assert.equal(adopter, expectedAdopter, "Owner of the expected pet should be this contract");

}

1. Test to retrieve all pet owners

function testGetAdopterAddressByPetIdInArray() public {

// store adopter in memory rather than constract

address[16] memory adopters = adoption.getAdopters();

Assert.equal(adopters[expectedPetId], expectedAdopter, "Owner of the expected pet should be this contract");

}

1. Execute test with command **truffle test**

Using network 'development'.

Compiling ./contracts/Adoption.sol...

Compiling ./test/TestAdoption.sol...

Compiling truffle/Assert.sol...

Compiling truffle/DeployedAddresses.sol...

TestAdoption

✓ testUserCanAdoptPet (91ms)

✓ testGetAdopterAddressByPetId (70ms)

✓ testGetAdopterAddressByPetIdInArray (89ms)

3 passing (670ms)

## Develop front-end UI and intelligent contract interactions

Pet store Truffle Box comes with code for the front end of the application. This code exists in the src/ directory. Front-end unused build systems (webpack,grunt, etc.) make it as easy to use as possible. The structure of the application already exists; we will fill the unique functionality of the block chain. In this way, you can master this knowledge and apply it to your own front-end development.

1. Open src/js/app.js. in a text editor
2. Instantiation web3. There is a global App object to manage our application, load the pet data in the init (), and then call the function initWeb3 (). The web3.js library interacts with the ethernet block chain. It can retrieve user accounts, send transactions, interact with smart contracts, etc.

// init web3

if (typeof web3 !== 'undefined') {

App.web3Provider = web3.currentProvider;

web3 = new Web3(web3.currentProvider);

} else {

// use Web3.providers config provider

App.web3Provider = new Web3.providers.HttpProvider('http://127.0.0.1:7545');

web3 = new Web3(App.web3Provider);

}

1. Instantiate a contract. Still in src/js/app.js, remove multi-line comments from initContract and replace them with the following:

$.getJSON('Adoption.json', function(data) {

// get contract and init with truffle-contract

var AdoptionArtifact = data;

App.contracts.Adoption = TruffleContract(AdoptionArtifact);

// set provider

App.contracts.Adoption.setProvider(App.web3Provider);

// use contracts to retrieve and mark adopted pets

return App.markAdopted();

});

1. Get adopted pets and update UI. Still in src/js/app.js, remove multi-line comments from markAdopted and replace them with the following:

var adoptionInstance;

App.contracts.Adoption.deployed().then(function(instance) {

adoptionInstance = instance;

return adoptionInstance.getAdopters.call();

}).then(function(adopters) {

for (i = 0; i < adopters.length; i++) {

if (adopters[i] !== '0x0000000000000000000000000000000000000000') {

$('.panel-pet').eq(i).find('button').text('Success').attr('disabled', true);

}

}

}).catch(function(err) {

console.log(err.message);

});

1. Handle the adopt () function. Still in / src/js/app.js, remove multi-line comments from handleAdopt and replace them with the following:

var adoptionInstance;

web3.eth.getAccounts(function(error, accounts) {

if (error) {

console.log(error);

}

var account = accounts[0];

App.contracts.Adoption.deployed().then(function(instance) {

adoptionInstance = instance;

// Execute adopt as a transaction by sending account

return adoptionInstance.adopt(petId, {from: account});

}).then(function(result) {

return App.markAdopted();

}).catch(function(err) {

console.log(err.message);

});

});

Congratulations By learning this example, now you are a DApp developer.