DA51 Lab session Dapp

Ethereum Decentralized Application (Dapp)

J. Gaber, gaber@utbm.fr, jaafar.gaber@gmail.com

Target set of this session:
After completion of this session, you will successfully know:

- how to develop Ethereum Blockchain Applications
- how to install and use Ethereum Decentralized Application Frameworks
- you will know the steps to develop an Ethereum smart contract for a blockchain marketplace.

Truffle Suite is the most widely used set of tools to simplify the process of developing smart contracts and DApps.

DApps are decentralized applications which have smart contacts deployed to an Ethereum network at the back end and a web UI as the front end.

I. Setting up the development environment

you need to install following toolkits: Node.js, npm, and Truffle Suite.

1. to check if they are already installed (see lab session 3) or to test your installation once done, bring up a terminal session and execute the command node -v (resp. npm -v) to print out the version of Node (resp. npm) which have been installed:

node -v
npm -v

2. install Truffle with npm

npm install –global truffle

3. to check execute the command truffle version

truffle version

ps: There is a bug for Windows for version Truffle 5.1.15, you'd better install truffle 5.1.10 using the commands:
npm uninstall --global truffle
npm install --global truffle@5.1.10

4. To list the set of commands, use the command truffle help

truffle help

5. How to know about all installed global modules or packages with detail (recall you are using npm)?

npm ls -gl

6. What's the difference between npm packages and modules?

Hint: https://docs.npmjs.com/about-packages-and-modules

A package is a file or directory that is described by a package.json file. A package must contain a package.json file to be published to the npm registry.

Modules allow you to split an application into separate files instead of having your entire application in one file. This concept is also present in other languages such as the C language using the include statement, or Python with the import statement, etc.

The npm registry contains packages, many of which are also Node modules, or contain Node modules. Since modules are not required to have a `package.json` file, not all modules are packages. Only modules that have a `package.json` file are also packages.

What is package.json ?
For more details on file package.json, head to https://nodejs.org/en/knowledge/getting-started/npm/what-is-the-file-package-json/

II. The Truffle Suite

The Truffle Suite provides a suite of tools to write smart contracts with the solidity language, to test the smart contracts and to deploy them to the blockchain.

7. The npm list command prints to stdout the versions of packages that are installed, as well as their dependencies in a tree structure. Use the command npm list -gl and identify the tools we have.

npm list -gl

8. What's the purpose of Ganache?

Hint: https://trufflesuite.com/docs/ganache/

Ganache is a personal blockchain for rapid Ethereum and Corda distributed application development. You can use Ganache across the entire development cycle; enabling you to develop, deploy, and test your DApps in a safe and deterministic environment.

Ganache provides 10 accounts preloaded with 100 fake Ether (ETH). Each account has a unique address and a private key.

9. What's the purpose of web3.js?

Hint: https://github.com/ethereum/web3.js/

Web3.js is the Ethereum JavaScript API that let you interacts with Ethereum blockchain nodes, local or remote, using a HTTP, IPC or WebSocket connection. It can retrieve user accounts, send transactions, interact with smart contracts, and more.

Recall that the traditional front-end client written in HTML/CSS/JavaScript connects and interacts with a backend HTTP server. On the other hand, the smart contract client needs to connect to a Ethereum blockchain node.

10. What's the purpose of MetaMask?

MetaMask is a crypto wallet and gateway to blockchain DApps. It is available as a browser extension *(for Chrome and Firefox) and as a mobile app. It comes with a key vault, secure login, and token wallet - everything you need to manage your digital assets.*

see lab session 4, for Firefox as Add-ons or Chrome extension. You will see a Fox icon appear at the top right of the navigation bar.

11. What's the purpose of Lite-Server?
Hint: https://github.com/johnpapa/lite-server

A Light-weight development HTTP server Bundled with Truffle Box, under folder petshop/node_modules/lite-server.

12. What's the purpose of Gulp?
Hint: https://gulpjs.com/

A Task Runner (Build Tool) to Automate the Workflow.

13. To edit your source code, you can use your preferred IDE, e.g., VS Code, Remix, …
You can install support for Solidity via "Extensions". Hint: https://trufflesuite.com/blog/build-on-web3-with-truffle-vs-code-extension/
Remix is a web IDE (i.e., an on-line editor) for Solidity development, instant testing, and deployment, https://remix.ethereum.org/

III. Truffle Box

In what follows, you will develop your DApp with Truffle Box, which provides several boilerplates, examples, and project templates, https://trufflesuite.com/docs/truffle/quickstart/

We will begin as a first DApp example with the well-known Pet-Shop Truffle Box: https://trufflesuite.com/boxes/pet-shop/

1) Creating a Truffle project using a Truffle Box

14. To create a Truffle Project, first launch a terminal and create a project directory and call it called petshop

mkdir petshop
cd petshop

ps: avoid working with directory in Dropbox folder or Google Drive folder (see https://ethereum.stackexchange.com/questions/24863/why-are-my-contracts-showing-as-not-deployed-in-truffle/139332#139332).

15. Download and unpack the Truffle box Pet-Shop into this project directory petshop.
use the following command: truffle unbox pet-shop

16. What is the output of this command?

The boilerplate codes are unpacked into the petshop directory created.

17. Describe the Truffle's directory structure and content?

contracts/	*contains the solidity source file (.sol) for smart contracts. The Pet-Shop box provides a smart contract called "Migrations.sol" used for deployment.*
migrations/	*Truffle uses a migration system to deploy smart contracts. A Migration is a special smart contract that keeps track of changes.*
test/	*contains the test scripts (written in JavaScript or Solidity) for the smart contracts.*
node_modules/	*contains the node.js dependencies.*
src/	*contains client-side programs in HTML/CSS/JS and related resources such as images and fonts.*
truffle-config.js	*the Truffle configuration file.*

Migrations are JavaScript files that help you deploy contracts to the Ethereum network.
See https://trufflesuite.com/docs/truffle/how-to/contracts/run-migrations/

The Migrations contract keeps track of which migrations were done on the current network. The Migrations contract stores (in last_completed_migration) a number that corresponds to the last applied "migration" script, found in the migrations folder. So, as this Migrations contract stores the number of the last deployment script applied, Truffle will not run those scripts again.

IV. Writing the smart contract

Let us now write our smart contract. Under the contracts directory, create a solidity source file and call it Adoption.sol:

contracts > Adoption.sol

```
// SPDX-License-Identifier: MIT
pragma solidity >=0.4.22 <0.9.0;
//pragma solidity ^0.5.16;

// pragma provides information to the compiler to specify the Solidity version.
//The caret (^) denotes minimum version of 0.4.22, but lower than the next version of 0.9.0

contract Adoption {
    // Declare an array of 16 Ethereum addresses as the adopter of each pet
    address[16] public adopters;

    // Adopting a pet
    function adopt(uint petId) public returns (uint) {
        require(petId >= 0 && petId <= 15, "Pet-ID out of range");
        adopters[petId] = msg.sender;  // address of account/smart contract that calls this function is given by msg.sender.
        return petId;
    }

    // Retrieving the adopters
    function getAdopters() public view returns (address[16] memory) {
        return adopters;
    }
    //The keyword view indicates that the function will not modify the state of the contract.

}

```

Which function defines the business logic of the smart contract?
The function adopt()

V. Compiling and migrating the smart contract

18. Compile the smart contract
truffle compile

What is the output?
Hint: examine the petshop/build/contracts/ directory.

The outputs are Artifacts Adoption.json and Migrations.json, written to petshop\build\contracts directory.

19. Examine the Migrations.sol smart contract code:

contracts > Migrations.sol

```
1   // SPDX-License-Identifier: MIT
2   pragma solidity >=0.4.22 <0.9.0;
3
4   contract Migrations {
5     address public owner = msg.sender;
6     uint public last_completed_migration;
7
8     modifier restricted() {
9       require(
10        msg.sender == owner,
11        "This function is restricted to the contract's owner"
12      );
13      _;
14    }
15
16    function setCompleted(uint completed) public restricted {
17      last_completed_migration = completed;
18    }
19  }
20
```

The Migrations contract keeps track of which migrations were done on the current network. Once a migration is done, Truffle will store its reference number in the Migrations contract. So as Truffle will not run those scripts again.

The Migrations contract stores this number that corresponds to the last applied "migration" script, found in the migrations folder, in the attribute last_completed_migration.

Once you've created a new contract or your DApp may need to have a modified contract deployed, you create a new migration script with an increased number. Once the migration is done, Truffle will store the reference number in the Migrations contract.

Note that a migration is a deployment script. In the Migrations directory, there is currently a JavaScript called 1_initial_migration.js. Note that the filename is prefixed with a number. The numbered prefix is required to record whether the migration ran successfully (see https://trufflesuite.com/docs/truffle/how-to/contracts/run-migrations/).

20. Examine its code

```
migrations > JS 1_initial_migration.js
1   var Migrations = artifacts.require("./Migrations.sol");
2
3   module.exports = function(deployer) {
4     deployer.deploy(Migrations);
5   };
6
```

and indicate which contract is deployed by this first migration?
this code deploys the contracts/Migrations.sol.

21. Create the second migration script called 2_deploy_contracts.js in the Migrations directory, with the following code, to deploy your Adoption smart contract:

```
migrations > JS 2_deploy_contracts.js
1
2   var Adoption = artifacts.require("Adoption");
3
4   module.exports = function(deployer) {
5     deployer.deploy(Adoption);
6   };
7
```

22. What is the purpose of this script?
to deploy our Adoption smart contract.

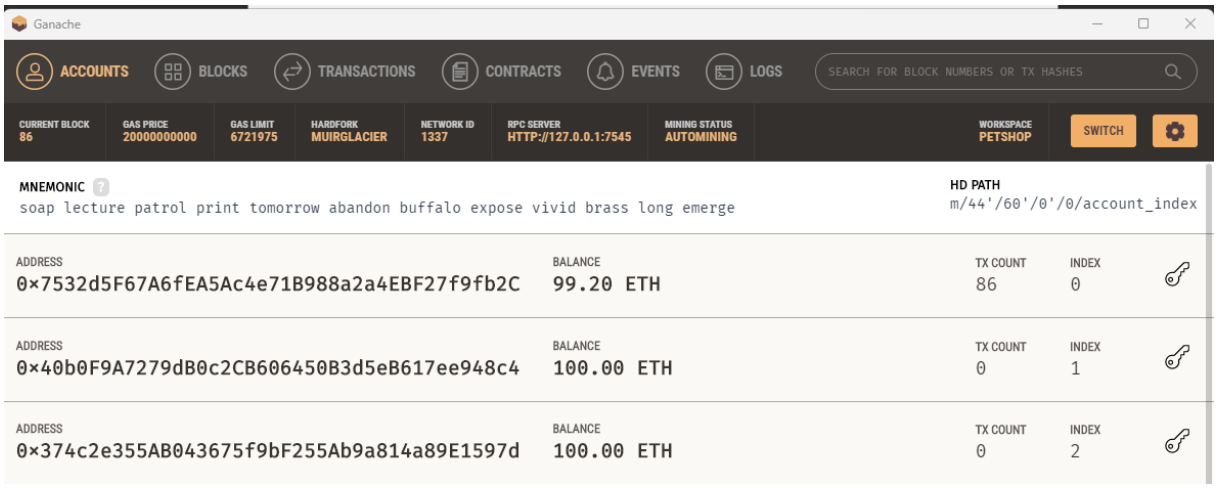
VI. Migrate to the Ganache Local Personal Blockchain

Before we can migrate our smart contract to a blockchain, we need a blockchain running. We will use Ganache, a local personal blockchain for Ethereum development, i.e., to deploy contracts, develop applications, and run tests.

Ganache provides 10 accounts preloaded with 100 fake Ether (ETH). Each account has a unique address and a private key.

Launch Ganache, then click on New Workspace Button. Fill in the required fields, in Workspace Name, enter petshop, then click Add Project and select the file petshop/truffle-config.js, and finally Save Workspace.

This will generate a blockchain running locally on port 7545.



Examine and inspect the Ganache console, by clicking the different tabs of its menu:

ACCOUNTS: a panel that shows 10 accounts with 100 ETH. Each account has a 20-byte address and a private key.
BLOCKS: to inspect the blocks, a block (Block 0) was created.
TRANSACTION: panel to inspect transactions (no transactions for the moment).
CONTRACTS: panel that shows the contracts. Here you have your two contracts Adoption and Migrations not deployed yet.
EVENTS: panel to show events
LOGS: panel to show the logs and error messages.

you also have a bar below the menu showing information such as the current block, Gas price, gas limit, the network id, the RPC server, ….

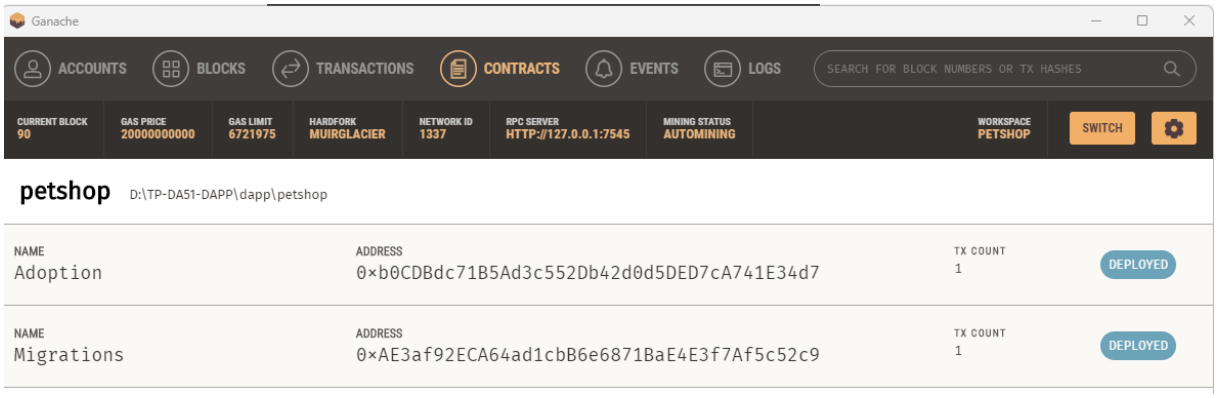
23. Head back to your terminal and execute the command to deploy (i.e., migrate) your smart contracts to the blockchain

truffle migrate

you will get output like the following:

```
2_deploy_contracts.js
=====================

   Deploying 'Adoption'
   --------------------
   > transaction hash:    0xe9841828f7429ac16152d67554f835b07c7b766969c0e5573a2d9f4d74a68cef
   > Blocks: 0            Seconds: 0
   > contract address:    0x3da151a352d2407B5570128CA315de5b52356573
   > block number:        89
   > block timestamp:     1668795794
   > account:             0x7532d5F67A6fEA5Ac4e71B988a2a4EBF27f9fb2C
   > balance:             99.19510814
   > gas used:            226052
   > gas price:           20 gwei
   > value sent:          0 ETH
   > total cost:          0.00452104 ETH


   > Saving migration to chain.
   > Saving artifacts
   -------------------------------------
   > Total cost:          0.00452104 ETH


Summary
=======
> Total deployments:   2
> Final cost:          0.0083599 ETH



D:\Dropbox\EnseigDropbox\DA51-A21\Part I\TD TP\TP\dapp\petshop>
```

and the Ganache console shows now the new state of the blockchain.



24. Check out the other tabs: Blocks, Transactions, Accounts and Logs.

VII. Testing the smart contract

The Truffle framework provides testing support. The Truffle test scripts can be written in JavaScript or Solidity.

Let us write our test script in Solidity.

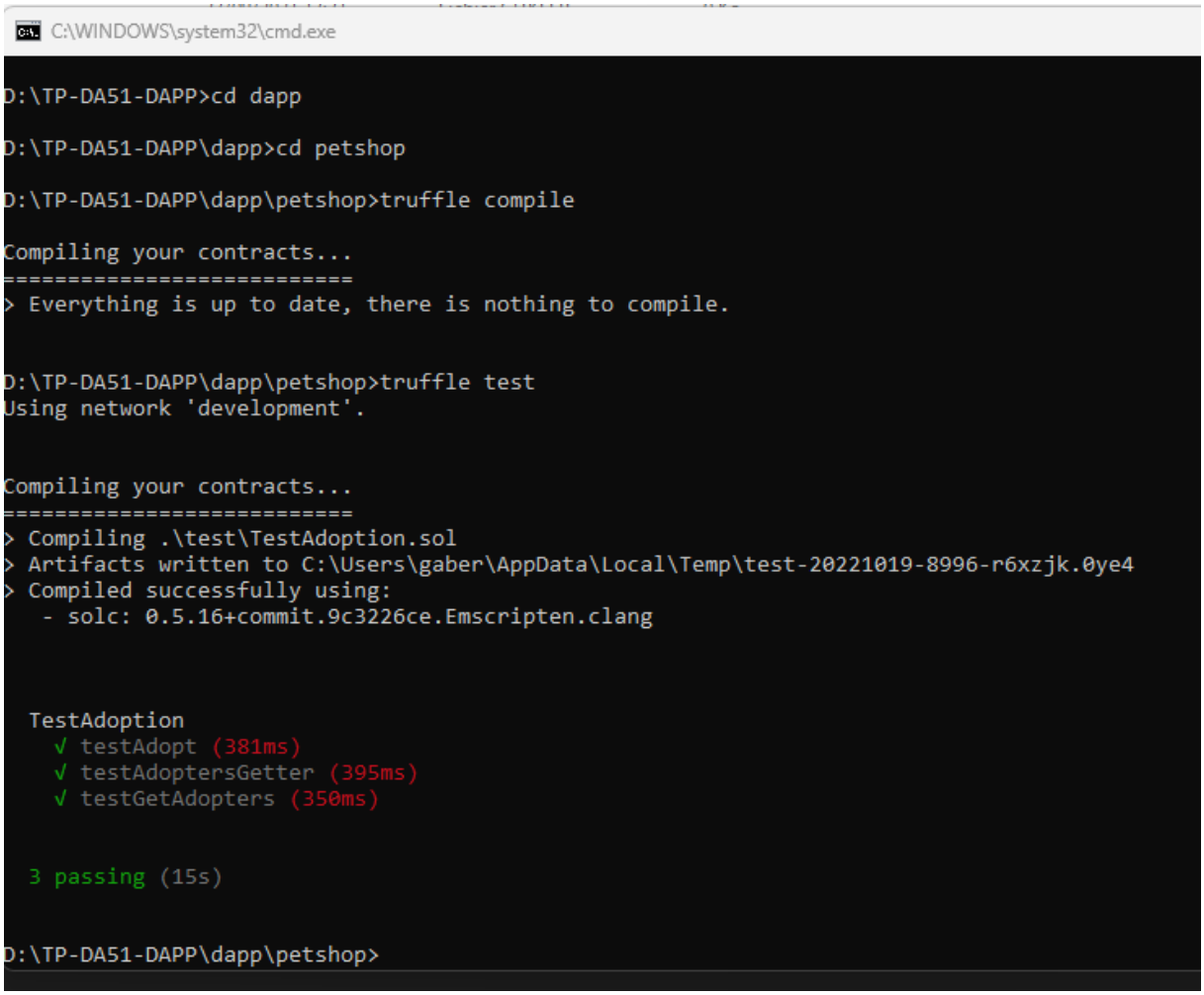
25. Create a solidity source file in the test directory and call it TestAdoption.sol, with the following code:

test > TestAdoption.sol

```
// SPDX-License-Identifier: MIT
pragma solidity >=0.4.22 <0.9.0;

//pragma solidity ^0.5.0;

import "truffle/Assert.sol";
//this smart contract provides the various assertions, such as Assert.equal() used in the script.

//import "../contracts/Assert.sol";

import "truffle/DeployedAddresses.sol";
//This smart contract gets the address of the deployed instance of contract by Truffle to the blockchain.

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

contract TestAdoption {
    // Deploy an instance of Adoption smart contract for testing. Adoption() returns its address.
    Adoption adoption = Adoption(DeployedAddresses.Adoption());
    // The id of the pet that will be used for testing
    uint expectedPetId = 8;
    // The expected owner of adopted pet is this contract; The address of TestAdoption contract that will be the sender of the transactions.
    address expectedAdopter = address(this);

    // Test the adopt() function
    function testAdopt() public {
        uint returnedId = adoption.adopt(expectedPetId);
        Assert.equal(returnedId, expectedPetId, "Adoption of the expected pet should match what is returned.");
    }

    // Test adopters public getter to retrieve a single pet's owner
    function testAdoptersGetter() public {
        address adopter = adoption.adopters(expectedPetId);
        Assert.equal(adopter, expectedAdopter, "Owner of the expected pet should be this contract");
    }

    // Test the getAdopters() function to retrieve all pet owners
    function testGetAdopters() public {
        // Store adopters in memory rather than contract's storage
        address[16] memory adopters = adoption.getAdopters();
        Assert.equal(adopters[expectedPetId], expectedAdopter, "Owner of the expected pet should be this contract");
    }
}
```

26. To run the test, head back to the terminal and use command:
Truffle test

You should get an output like what follows:


C:\WINDOWS\system32\cmd.exe
D:\TP-DA51-DAPP>cd dapp
D:\TP-DA51-DAPP\dapp>cd petshop
D:\TP-DA51-DAPP\dapp\petshop>truffle compile
Compiling your contracts...
> Everything is up to date, there is nothing to compile.
D:\TP-DA51-DAPP\dapp\petshop>truffle test
Using network 'development'.
Compiling your contracts...
> Compiling .\test\TestAdoption.sol
> Artifacts written to C:\Users\gaber\AppData\Local\Temp\test-20221019-8996-r6xzjk.0ye4
> Compiled successfully using:
- solc: 0.5.16+commit.9c3226ce.Emscripten.clang
TestAdoption
√ testAdopt (381ms)
√ testAdoptersGetter (395ms)
√ testGetAdopters (350ms)
3 passing (15s)
D:\TP-DA51-DAPP\dapp\petshop>

VIII. Creating a user interface to interact with the smart contract

Now that we have written our smart contract, deployed it to our local test blockchain, and tested the functions of the smart contract via the console, it's time to create a client-side UI.

The Pet-Shop Truffle box has already a front-end within the src/ directory. Bootstrap's boilerplate was used to layout the web page.

what is exactly boilerplate HTML?

Let us make a digression on Boilerplate html.

By making use of some boilerplate code, you can speed up the development of the client-side UI. So, what is exactly boilerplate HTML?

When defining the HTML for the web pages in your DApp, you could start entirely from scratch or you can download some already defined HTML, referred as a boilerplate code.

To download some boilerplate HTML and CSS files, among the locations available you have http://www.initializr.com/

Once you generate and download your boilerplate code (e.g., using Bootstrap), you incorporate this boilerplate code within your application and customize it to suit your purposes.

The Pet-Shop Truffle box has already a front-end Bootstrap's boilerplate within the src/ directory. You can also of course generate your own boilerplate code and customize it later for next project.

End of the digression.

Since, the Pet-Shop Truffle box has already a front-end Bootstrap's boilerplate within the src/ directory, open the index.html located in the src directory and browse it.

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <!-- The above 3 meta tags *must* come first in the head; any other head content must come *after* these tags -->
    <title>Pete's Pet Shop</title>

    <!-- Bootstrap -->
    <link href="css/bootstrap.min.css" rel="stylesheet">

    <!-- HTML5 shim and Respond.js for IE8 support of HTML5 elements and media queries -->
    <!-- WARNING: Respond.js doesn't work if you view the page via file:// -->
    <!--[if lt IE 9]>
      <script src="https://oss.maxcdn.com/html5shiv/3.7.3/html5shiv.min.js"></script>
      <script src="https://oss.maxcdn.com/respond/1.4.2/respond.min.js"></script>
    <![endif]-->
  </head>
  <body>
    <div class="container">
      <div class="row">
        <div class="col-xs-12 col-sm-8 col-sm-push-2">
          <h1 class="text-center">Pete's Pet Shop</h1>
          <hr/>
          <br/>
        </div>
      </div>

      <div id="petsRow" class="row">
        <!-- PETS LOAD HERE -->
      </div>
    </div>

    <div id="petTemplate" style="display: none;">
      <div class="col-sm-6 col-md-4 col-lg-3">
        <div class="panel panel-default panel-pet">
          <div class="panel-heading">
            <h3 class="panel-title">Scrappy</h3>
          </div>
          <div class="panel-body">
            <img alt="140x140" data-src="holder.js/140x140" class="img-rounded img-center" style="width: 100%;"
                 src="https://animalso.com/wp-content/uploads/2017/01/Golden-Retriever_6.jpg" data-holder-rendered="true">
            <br/><br/>
            <strong>Breed</strong>: <span class="pet-breed">Golden Retriever</span><br/>
            <strong>Age</strong>: <span class="pet-age">3</span><br/>
            <strong>Location</strong>: <span class="pet-location">Warren, MI</span><br/><br/>
            <button class="btn btn-default btn-adopt" type="button" data-id="0">Adopt</button>
          </div>
        </div>
      </div>
    </div>

    <!-- jQuery (necessary for Bootstrap's JavaScript plugins) -->
    <script src="https://ajax.googleapis.com/ajax/libs/jquery/1.12.4/jquery.min.js"></script>
    <!-- Include all compiled plugins (below), or include individual files as needed -->
    <script src="js/bootstrap.min.js"></script>
    <script src="js/web3.min.js"></script>
    <script src="js/truffle-contract.js"></script>
    <script src="js/app.js"></script>
  </body>
</html>

```

27. Examine the JavaScript/jQuery "src/js/app.js".

the global App object starts and manages the application.

init() loads the animal data and then call the initWeb3() function.

The current file has some functions to be completed.

28. Replace "app.js" with the provided file.

The next step is interacting with the DApp in a browser.

IX. Configure the MetaMask Wallet

29. See Lab session 4 to install and configure MetaMask Wallet.

MetaMask Wallet is a browser extension for managing digital assets via the private key.

30. Connect MetaMask to your local Ganache blockchain

Via Network on MetaMask, Select Custom RPC, in Network Name, enter a name MonGanache, in New RPC URL, enter http://localhost:7545, where the Ganache RPC server is listening, then Save.

You should get Account 1 mapped to Ganache first account with about 100 ETH.

31. To import an account from Ganache

In the Ganache console, select an account, e.g., index 4, click the Key icon (on the right) to show the private key. Copy the private key.
Head to the Fox icon on the browser to pop up MetaMask. Click the Circle icon. Select Import Account. Paste the copied private key and Import.

You should get a new account called Account 2 imported from your Ganache with 100 ETH.

X. Set up the lite-Server

32. The lite-server is configured in bs-config.json. Locate this file.

The configuration indicates that the src directory, which contains the HTML/CSS/JS, and the build/contracts directory, which contains the smart contract artifacts, form the root directory of the server.

33. Examine the file package.json

Note that for the npm run dev command launches the lite-server, in package.json the scripts object is included to add a dev alias to lite-server.

XI. Run your Pet-Shop Dapp

34. Restart the ganache to reset the state.

35. Head to your terminal, and launch the command truffle migrate --reset to reset the smart contracts

truffle migrate --reset

36. start the lite-server:

npm run dev

on the web page, select a pet, then click Adopt. MetaMask will show the transaction details, click Confirm. The Adopt button changes to Success and disabled.