Blockchain Reference Sheet

Carlo Supina

"Blockchain technology and dApps have the ability to decentralize power from existing authorities through the use of smart contracts, cryptocurrencies, and asset ownership. This shift will change current businesses and economic and social paradigms. Transaction costs and barriers to entry will be reduced in various industries. The result will likely lead to an increase in economic exchange and prosperity."

– ConsenSys

Terms:

Blockchain Distributed ledger technology allowing peer to peer transactions

Bitcoin The first decentralized digital currency, uses a distributed ledger (blockchain)

Altcoin Cryptocurrencies that aren't Bitcoin

Ethereum Blockchain based distributed computing platform

Ether The currency used on the Ethereum platform

ICO (Initial Coin Offering) Campaign where a percentage of a new cryptocurrency is sold to early backers

Gas Processing cost of an Ethereum transaction and computing cost of smart contract

EVM (Ethereum Virtual Machine) Decentralized runtime environment for smart contracts on the network

Smart Contract Code that runs on a blockchain (runs on the EVM for Ethereum)

Solidity Language for writing smart contracts

dApp Distributed application

Hashing Sending data through a one-way function that returns an output of characters that is unique to the input

Hash Resulting output of putting data through a hashing function

Ropsten Name of the official Ethereum test network

Proof of Work Method of proving a block through solving mathematical problems using processing power

Mining Using processing power to attempt to solve the proof of work algorithms

Proof of Stake Method of proving a block through deterministically choosing a creator based on wealth

The DAO A decentralized autonomous organization that was created using smart contracts, victim of an exploitation attack

Ethereum Classic The original Ethereum network that resulted from the hard fork after the DAO attack

Vitalik Buterin Co-founder of Ethereum

Satoshi Nakamoto Name used by the unkown person/s who designed Bitcoin

Satoshi (unit) A hundred millionth of a bitcoin (0.00000001 BTC)

Wei 10^{-18} of one ether

ABI (Application Binary Interface) Defines the details of how data is encoded and decoded from the machine code

Links:

General:

Ethereum Project Official Ethereum website

Etherscan Ethereum block explorer

Goldman Sachs Blockchain Presentation Excellent visual representation of blockchain

The DAO Attack An attacker stole $50 million worth of ether by exploiting a bug in a smart contract

Metamask Chrome plugin for a browser based Ethereum node

Coin Market Cap Market capitalizations of cryptocurrencies

ConsenSys Software foundry focused on developing peer to peer applications

Crypto Kitties Popular distributed game made to run on the EVM

Development:

Truffle Framework Framework for Ethereum development

Geth Command line interface for running an Ethereum node implemented with Golang

Ganache Node.js based Ethereum client for testing smart contracts

OpenZeppelin Framework of secure smart contracts written in Solidity

Truffle Angular 2 App Starter Angular 2 app that uses the Truffle framework

Crypto Zombies Codecademy like course for learning to write smart contracts in solidity

Setting Up a Development Environment

Software

Node.js Javascript runtime

NPM Package manager for Node.js

Python 3 Interpreted scripting language

Web3.py Python interface for interacting with Ethereum ecosystem

Flask Framework for web development with Python

Postman GUI client for testing APIs

Overview

The development method that I use will follow this general workflow:

- Run Ganache-cli in a separate terminal and leave it running in the background.
- Write a smart contract using Solidity
- Create a Flask server for making it easier to interact with the blockchain
- Create a file of helper functions for making it easier to interact with a smart contract
- Write tests and put them in the "`if name == '__main__'`" section
- Refine smart contract (solidity)
- Add any new smart contract functions to API
- Edit tests if needed...

Ganache-Cli

Ganache-cli can be thought of as a simulated blockchain that can exist and run entirely on your own computer. This means that instead of having to wait around 30 seconds for the current block to be mined every time a transaction is sent, your transactions are mined instantly allowing for much faster testing. To install ganache-cli enter the following command into either your bash terminal or command prompt:

```
npm install -g ganache-cli
```

I would recommend opening another terminal for running ganache-cli that you can later hide. After you enter the command to begin running it, you don't need to do anything else with it. To start running ganache-cli simply enter the following command into your terminal:

```
ganache-cli
```

Writing Smart Contracts

Smart contracts refer to code that is run on the blockchain. For Ethereum they are most commonly written in using the language Solidity, however, there are alternatives. For now I don't plan on explaining Solidity in this document, but I may add it in the future. Here is example smart contract written in Solidity that I will be using in the demo.

```
pragma solidity ^0.4.0;


contract Epics {

    //counter of the number of students
    uint256 total_students = 0;
    //key value pair of addresses to Students, allows Students to be searched by address
    mapping(address => Student) students;
    //array to store the addresses of all students
    address[] public student_addrs;
    //struct for a Student, to simplify - a categorized group of variables, a custom type
    struct Student{
        address addr;
        address team;
        string name;
        uint8 age;
        bool registered;
    }

    //register a student
    function registerStudent(address _addr, string _name, uint8 _age){
        //assert that the address has not already been registered
        Student storage s = students[_addr];
        assert(!s.registered);

        //set a key value pair in the mapping and initialize the struct variables
        students[_addr] = Student({addr: _addr, team: 0, name: _name, age: _age, registered: true});
        //push the added address onto the array
        student_addrs.push(_addr);
        total_students ++;
    }

    //set the students team
    function setTeam(address _addr, address _team_addr){
        //assert that the student and team have been registered
        Student storage s = students[_addr];
        Team storage t = teams[_team_addr];

        assert(s.registered);
        assert(t.registered);

        //set the student’s team to the address specified
        students[_addr].team = _team_addr;
        //push the student’s address onto the teams array of students
        teams[_team_addr].students.push(_addr);
    }

    //you can return multiple values, all values from the struct returned at once
    function getStudent(address _addr) constant returns (address, address, string, uint8, bool){
        Student storage s = students[_addr];
        return(s.addr, s.team, s.name, s.age, s.registered);
    }

    //return array of all student addresses registered
    function getStudentAddresses() constant returns(address[]){
        return student_addrs;
    }


    uint256 total_teams = 0;
    mapping(address => Team) teams;
    address[] public team_addrs;
    struct Team{
        address addr;
        string name;
        address[] students;
        bool registered;
    }

    function registerTeam(address _addr, string _name){
        Team storage t = teams[_addr];
        assert(!t.registered);

        teams[_addr] = Team({addr: _addr, name: _name, students: new address[](0), registered: true});
        team_addrs.push(_addr);
        total_teams ++;
    }

    function getTeam(address _addr) constant returns (address, string, address[], bool){
        Team storage t = teams[_addr];
        return(t.addr, t.name, t.students, t.registered);
    }

    function getTeamAddresses() constant returns(address[]){
        return team_addrs;
    }
}
```

Setup and Helper Functions

Web3 module is the foundation of interacting with an Ethereum blockchain. The following code uses this module for functions that connect to the RPC client and compiling, initializing and deploying the contract on the blockchain. The code also has helper functions for writing and reading from the blockchain and more functions for converting certain contract calls into dictionaries.

```
from web3 import Web3, HTTPProvider
import json
from subprocess import Popen, PIPE


def declare_contract():
    return input("Enter the path to your contract (.sol): ")


def connect_to_rpc():
    """Connect to an RPC.
    Enter the ip and port of the rpc when prompted.
    Returns the Web3 objects created from the specified provider.
    """
    provider = "http://"

    ip = input("IP of  provider: ")
    provider += "127.0.0.1" if ip == "" else ip

    provider += ":"

    port = input("Port of provider: ")
    provider += "8545" if port == "" else port

    print("Connecting to provider: " + provider)
    web3 = Web3(HTTPProvider(provider))
    return web3


def initialize_contract(web3, contract_source):
    contract = create_contract(web3, contract_source)

    deployer_address = input("Enter the address of the deployer: ")
    deployer_address = web3.eth.accounts[0] if deployer_address == "" else deployer_address

    gas = input("Enter the desired gas for the contract: ")
    gas = 3000000 if gas == "" else int(gas)

    contract_address = deploy_contract(web3, contract, deployer_address, gas)
    print("The contract address is: " + contract_address)
    return contract_address


def get_contract_bytecode(file_path):
    p = Popen(["solc", "--optimize", "--bin", file_path], stdin=PIPE, stdout=PIPE, stderr=PIPE)

    bytecode, err = p.communicate()

    return "0x" + str(bytecode).split("\\n")[-2]


def get_contract_bytecode_runtime(file_path):
    p = Popen(["solc", "--optimize", "--bin-runtime", file_path], stdin=PIPE, stdout=PIPE, stderr=PIPE)
    bytecode, err = p.communicate()

    return "0x" + str(bytecode).split("\\n")[-2]


def get_contract_abi(file_path):
    p = Popen(["solc", "--optimize", "--abi", file_path], stdin=PIPE, stdout=PIPE, stderr=PIPE)

    abi, err = p.communicate()

    return json.loads(str(abi).split("\\n")[-2])


def create_contract(web3, file_path):
    return web3.eth.contract(abi=get_contract_abi(file_path),
                             bytecode=get_contract_bytecode(file_path),
                             bytecode_runtime=get_contract_bytecode_runtime(file_path))


def deploy_contract(web3, contract, deployer_address, gas):
        tx_hash = contract.deploy(transaction={"from": deployer_address, "gas": gas})
        receipt = web3.eth.getTransactionReceipt(tx_hash)
        return receipt["contractAddress"]


def create_contract_instance(web3, abi):
    return web3.eth.contract(abi=abi)


def read_chain(contract_instance, contract_address, function_name, *args):
    contract_call = contract_instance.call({"to": contract_address})

    if function_name == "getStudent":
        return contract_call.getStudent(args[0])
    if function_name == "getTeam":
        return contract_call.getTeam(args[0])
    if function_name == "getStudentAddresses":
        return contract_call.getStudentAddresses()
    if function_name == "getTeamAddresses":
        return contract_call.getTeamAddresses()


def write_chain(contract_instance, from_address, contract_address, function_name, * args):
    contract_transact = contract_instance.transact({"from": from_address, "to": contract_address, "gas": 100000

    if function_name == "registerStudent":
        return contract_transact.registerStudent(args[0], args[1], args[2])
    if function_name == "registerTeam":
        return contract_transact.registerTeam(args[0], args[1])
    if function_name == "setTeam":
        return contract_transact.setTeam(args[0], args[1])


def student_to_dict(info):
    info_strings = ['address', 'team', 'name', 'age', 'registered']
    student_dict = {}

    for istr, i in zip(info_strings, info):
        student_dict[istr] = i
    return student_dict


def team_to_dict(info):
    info_strings = ['address', 'name', 'students', 'registered']
    team_dict = {}

    for istr, i in zip(info_strings, info):
        team_dict[istr] = i

    return team_dict
```

Creating a RESTful API

Creating an API will make it easier to test your smart contract functions, especially if you are working on a team. The Python Flask framework will make it easy to create one. The main two requests that are should be part of your API are the GET and the POST requests. The GET will be typically used whenever you need to call a accessor function in your smart contract. The POST request will be typically used whenever you need to call a mutator function in your smart contract.

```
from flask import Flask, jsonify, request
from flask_cors import CORS
from epics_functions import *

global CONTRACT_SOURCE
global WEB3
global CONTRACT_ADDRESS
global CONTRACT_ABI


app = Flask(__name__)
CORS(app)


@app.route('/students/addresses', methods=['GET'])
def get_student_addresses():
    instance = create_contract_instance(WEB3, CONTRACT_ABI)

    info = read_chain(instance, CONTRACT_ADDRESS, "getStudentAddresses")

    return jsonify({'result': info})


@app.route('/teams/addresses', methods=['GET'])
def get_team_addresses():
    instance = create_contract_instance(WEB3, CONTRACT_ABI)

    info = read_chain(instance, CONTRACT_ADDRESS, "getTeamAddresses")

    return jsonify({'result': info})


@app.route('/students', methods=['POST'])
def register_student():
    """
    :return: json including the transaction hash and the student data
    """

    instance = create_contract_instance(WEB3, CONTRACT_ABI)

    receipt = write_chain(instance, WEB3.eth.accounts[0], CONTRACT_ADDRESS, "registerStudent", request.json["ad
                          request.json["name"], request.json["age"])
    student = read_chain(instance, CONTRACT_ADDRESS, "getStudent", request.json['address'])

    student_dict = student_to_dict(student)

    output = [{'transaction': receipt,
              'student': student_dict}]

    return jsonify({'result': output})


@app.route('/teams', methods=['POST'])
```

```
def register_team():
    """
    :return: json including the transaction hash and the team data
    """

    instance = create_contract_instance(WEB3, CONTRACT_ABI)

    receipt = write_chain(instance, WEB3.eth.accounts[0], CONTRACT_ADDRESS, "registerTeam", request.json["addre
                          request.json["name"])
    team = read_chain(instance, CONTRACT_ADDRESS, "getTeam", request.json['address'])

    team_dict = team_to_dict(team)

    output = [{'transaction': receipt,
              'team': team_dict}]

    return jsonify({'result': output})


@app.route('/students/<address>', methods=['GET'])
def info_student(address):
    """
    :param address: Ethereum address of a registered student
    :return: json of the specified student's data
    """
    instance = create_contract_instance(WEB3, CONTRACT_ABI)
    info = read_chain(instance, CONTRACT_ADDRESS, "getStudent", address)

    output = [student_to_dict(info)]

    return jsonify({'result': output})


@app.route('/teams/<address>', methods=['GET'])
def info_team(address):
    """
    :param address: Ethereum address of a registered team
    :return: json of the specified team's data
    """
    instance = create_contract_instance(WEB3, CONTRACT_ABI)
    info = read_chain(instance, CONTRACT_ADDRESS, "getTeam", address)

    output = [team_to_dict(info)]

    return jsonify({'result': output})


@app.route('/students', methods=['GET'])
def all_students():
    """
    :return: json of all of the students in the system
    """

    instance = create_contract_instance(WEB3, CONTRACT_ABI)
    addresses = read_chain(instance, CONTRACT_ADDRESS, "getStudentAddresses")
    output = []

    for a in addresses:
        info = read_chain(instance, CONTRACT_ADDRESS, "getStudent", a)
        output.append(student_to_dict(info))
    return jsonify({"result": output})


@app.route('/teams', methods=['GET'])
def all_teams():
    """
    :return: json of all of the teams in the system
    """

    instance = create_contract_instance(WEB3, CONTRACT_ABI)
    addresses = read_chain(instance, CONTRACT_ADDRESS, "getTeamAddresses")
    output = []

    for a in addresses:
        info = read_chain(instance, CONTRACT_ADDRESS, "getTeam", a)
        output.append(team_to_dict(info))

    return jsonify({"result": output})


@app.route('/students/<address>/team', methods=['POST'])
def set_team(address):

    instance = create_contract_instance(WEB3, CONTRACT_ABI)

    receipt = write_chain(instance, WEB3.eth.accounts[0], CONTRACT_ADDRESS, "setTeam", address, request.json["

    student = read_chain(instance, CONTRACT_ADDRESS, "getStudent", address)
    student_dict = student_to_dict(student)

    output = [{'transaction': receipt,
              'student': student_dict}]

    return jsonify({'result': output})


if __name__ == '__main__':
    WEB3 = connect_to_rpc()
    CONTRACT_SOURCE = declare_contract()
    CONTRACT_ADDRESS = initialize_contract(WEB3, CONTRACT_SOURCE)
    CONTRACT_ABI = get_contract_abi(CONTRACT_SOURCE)
    app.run(host='0.0.0.0')
```