const Blockchain = require('./blockchain');

const Block = require('./block');

describe('Blockchain', () => {

//const blockchain = new Blockchain;

let blockchain, newChain, originalChain;

beforeEach(() => {

blockchain = new Blockchain();

newChain = new Blockchain();

originalChain = blockchain.chain;

});

it('contains a chain Array instance', ()=> {

expect(blockchain.chain instanceof Array).toBe(true);

});

it('starts with a genesis block', ()=> {

expect(blockchain.chain[0]).toEqual(Block.genesis());

});

it('adds a new block to the chain', ()=> {

const newData = 'foo bar';

blockchain.addBlock({data: newData});

expect(blockchain.chain[blockchain.chain.length - 1].data).toEqual(newData);

});

describe('isValidChain()', () => {

describe('when the chain does not start with the genesis block', () => {

it('returns false', () => {

blockchain.chain[0] = {data: 'fake-genesis'};

expect(Blockchain.isValidChain(blockchain.chain)).toBe(false);

});

});

describe('when the chain starts with the genesis block and has multiple blocks', () => {

beforeEach(() => {

blockchain.addBlock({data: 'Bears'});

blockchain.addBlock({data: 'Beets'});

blockchain.addBlock({data: 'Battlestar Galactica'});

});

describe(' and a lastHash reference has changed', () => {

it('returns false', () => {

//blockchain.addBlock({'Bears'});

//blockchain.addBlock({'Beets'});

//blockchain.addBlock({'Battlestar Galactica'});

blockchain.chain[2].lastHash = 'broken-lastHash';

expect(Blockchain.isValidChain(blockchain.chain)).toBe(false);

});

});

});

describe(' and the chain contains a block with an invalid field', () => {

it('returns false', () => {

//blockchain.addBlock({'Bears'});

//blockchain.addBlock({'Beets'});

//blockchain.addBlock({'Battlestar Galactica'});

blockchain.chain[2].data = 'some-cad-and-evil data';

expect(Blockchain.isValidChain(blockchain.chain)).toBe(false);

});

});

describe(' and the chain does not contain any invalid blocks', () => {

it('returns true', () => {

//blockchain.addBlock({'Bears'});

//blockchain.addBlock({'Beets'});

//blockchain.addBlock({'Battlestar Galactica'});

expect(Blockchain.isValidChain(blockchain.chain)).toBe(true);

});

});

});

});

describe('replaceChain()', () => {

let errorMock, logMock;

beforeEach(() => {

errorMock = jest.fn();

logMock = jest.fn();

global.console.error = errorMock;

global.console.log = logMock;

});

describe('when the new chain is not longer',() => {

beforeEach(() => {

newChain.chain[0] = {new: 'chain'};

blockchain.replaceChain(newChain.chain);

});

it('does not replace the chain', () => {

expect(blockchain.chain).toEqual(originalChain);

});

it('logs an error', () => {

expect(errorMock).toHaveBeenCalled();

});

});

describe('when the new chain is longer', () => {

describe('when the new chain is longer', () => {

beforeEach(() => {

newChain.addBlock({data: 'Bears'});

newChain.addBlock({data: 'Beets'});

newChain.addBlock({data: 'Battlestar Galactica'});

});

describe('and the chain is invalid',() => {

beforeEach(() => {

newChain.chain[2].hash = 'some-fake-hash';

blockchain.replaceChain(newChain.chain);

});

it('does not replace the chain', () => {

expect(blockchain.chain).toEqual(originalChain);

});

it('logs an error', () => {

expect(errorMock).toHaveBeenCalled();

});

});

describe('and the chain is valid',() => {

beforeEach(() => {

blockchain.replaceChain(newChain.chain);

});

it('replaces the chain', () => {

expect(blockchain.chain).toEqual(newChain.chain);

});

it('logs about the chain replacement', () => {

expect(logMock).toHaveBeenCalled();

});

});

});

});

});