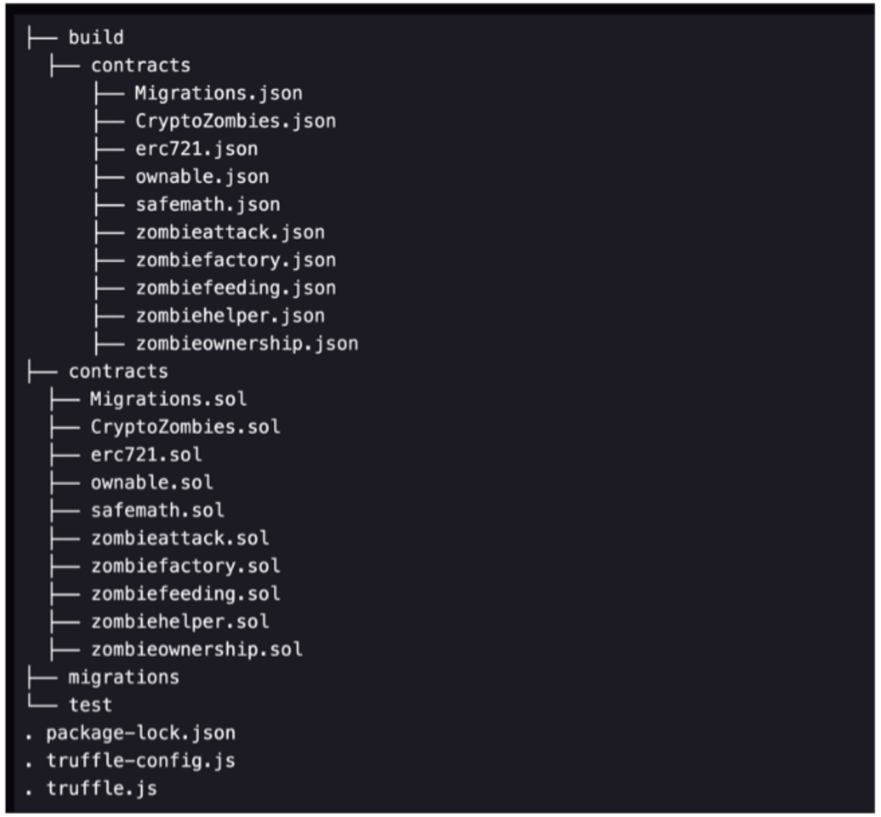
Testing Smart Contracts with Truffle

chapter 1



Truffle Famework는 솔리디티 코드(스마트 컨트랙트)를 로컬 환경에서 보다 쉽게 컴파일하고 배포할 수 있는 프레임워크

chapter 2

```
1  const CryptoZombies = artifacts.require("CryptoZombies");
2  contract("CryptoZombies", (accounts) => {
3      it("should be able to create a new zombie", () => {
4
5      })
6  })
7
```

스마트 컨트랙트를 컴파일하면 json 파일이 생성(build artifacts) =>contract abstraction return

contract() function
두 개를 인자로 받는데 어떤걸 테스트 할지와 실제 테스트 할 콜 백 함수를 인자로 받는다.

chapter 3

```
let [alice, bob] = accounts;
it("should be able to create a new zombie", async () => {
})
```

Canache(가나슈) : 테스트와 개발을 위한 목적으로 만들어진 이더리움의 가상 블록체인 시스템

chapter 4

```
const CryptoZombies = artifacts.require("CryptoZombies");
const zombieNames = ["Zombie 1", "Zombie 2"];
contract("CryptoZombies", (accounts) => {
    let [alice, bob] = accounts;
    it("should be able to create a new zombie", async () => {
        const contractInstance = await CryptoZombies.new();
        |
    })
})
```

CryptoZombies는 contract abstraction

따라서 새로운 object를 만들어 줘야 함

chapter 5

새로운 좀비 만들기 🧟

```
it("should be able to create a new zombie", async () => {
    const contractInstance = await CryptoZombies.new();
    const result = await contractInstance.createRandomZombie(zombieNames[0], {from: alice});
    assert.equal(result.receipt.status, true);
    assert.equal(result.logs[0].args.name,zombieNames[0]);
})
```

artifacts.require 는 log 를 생성한다.
result.logs[0].args.name
이름을 검색할 수 있다.

result.receipt.status == true => 거래 성공

assert.equal()

{from : alice} = > msg.sender

chapter 6

```
beforeEach(async () => {
    contractInstance = await CryptoZombies.new();
});
it("should be able to create a new zombie", async () => {
    const result = await contractInstance.createRandomZombie(zombieNames[0], {from: alice});
    assert.equal(result.receipt.status, true);
    assert.equal(result.logs[0].args.name,zombieNames[0]);
})
it("should not allow two zombies", async () => {
})
```

test 진행 시에 새로운 instance 필요

Mocha에서는 before(), after(), beforeEach(),afterEach()의 4가지 함수(hook)를 제공하여 테스트 코드 전후를 제어할 수 있음.

테스트가 실행되기 전에 어떤 것을 실행하려면, 코드는 beforeEach() 속에 넣어야 함.

chapter 7

```
async function shouldThrow(promise) {
  try {
      await promise;
     assert(true);
  }
  catch (err) {
      return;
  }
assert(false, "The contract did not throw.");

}

module.exports = {
  shouldThrow,
};
```

```
it("should not allow two zombies", async () => {
    await contractInstance.createRandomZombie(zombieNames[0], {from: alice});
    await utils.shouldThrow(contractInstance.createRandomZombie(zombieNames[1], {from: alice}));
})
```

```
Contract: CryptoZombies
    ✓ should be able to create a new zombie (129ms)
    ✓ should not allow two zombies (148ms)


  2 passing (1s)
```

chapter 8

<좀비 전송>

시나리오 1:

앨리스(소유주)가 자신의 주소, 밥의 주소, 그리고 전송하고자 하는 좀비 아이디를transfe
rFrom 의 인자로 전달해 호출하는 방식이다.

시나리오 2:

앨리스가 밥의 주소와 좀비 아이디를 통해서 approve 를 호출 그리고 그 계약은 밥이 좀
비를 데려가도록 승인받은 것을 저장 => 앨리스나 밥이 transferFrom을 부를 때 contrac
t는 msg sender가 앨리스나 밥의 주소와 동일한지 확인 => 동일하면 좀비 전송

chapter 9

시나리오 1

```
context("with the single-step transfer scenario", async () => {
    it("should transfer a zombie", async () => {
        const result = await contractInstance.createRandomZombie(zombieNames[0], {from: alice});
        // alice 가 만든 새로운 좀비에 대한 객체.
        const zombieId = result.logs[0].args.zombieId.toNumber();
        // log를 통해서 좀비 아이디를 들고 옴.
        await contractInstance.transferFrom(alice, bob, zombieId, {from: alice});
        // alice -> bob 좀비 전송
        const newOwner = await contractInstance.ownerOf(zombieId);
        // 새로운 소유자 설정.
        assert.equal(newOwner, bob);
        // 확인
    })
})
```

chapter 10

시나리오 2-1

```
it("should approve and then transfer a zombie when the approved address calls transferForm", async () => {
    const result = await contractInstance.createRandomZombie(zombieNames[0], {from: alice});
    const zombieId = result.logs[0].args.zombieId.toNumber();
    await contractInstance.approve(bob, zombieId, {from: alice});
    // bob을 approve 를 먼저 한다. (alice가)
    await contractInstance.transferFrom(alice, bob, zombieId, {from: bob});
    // 그 후 bob 이 transferFrom 을 call
    const newOwner = await contractInstance.ownerOf(zombieId);
    assert.equal(newOwner,bob);
})
```

chapter 11

시나리오 2-2

```
it("should approve and then transfer a zombie when the owner calls transferForm", async () => {
    const result = await contractInstance.createRandomZombie(zombieNames[0], {from: alice});
    const zombieId = result.logs[0].args.zombieId.toNumber();
    await contractInstance.approve(bob, zombieId, {from: alice});
    await contractInstance.transferFrom(alice, bob, zombieId, {from: alice});
    //alice 가 call
    const newOwner = await contractInstance.ownerOf(zombieId);
    assert.equal(newOwner,bob);
})
```

chapter 12

Time traveling

```
function _createZombie(string _name, uint _dna) internal {
  uint id = zombies.push(Zombie(_name, _dna, 1, uint32(now + cooldownTime), 0, 0)) - 1;
  zombieToOwner[id] = msg.sender;
  ownerZombieCount[msg.sender] = ownerZombieCount[msg.sender].add(1);
  emit NewZombie(id, _name, _dna);
}
```

=> 쿨 타임 존재

```
const secondZombieId = result.logs[0].args.zombieId.toNumber();
await time.increase(time.duration.days(1)); // 시간 증가
await contractInstance.attack(firstZombieId, secondZombieId, {from: alice});
```

=> evm_increaseTime
evm_mine
메소드를 통해서 시간 증가

chapter 13

Chai Assertion Library

```
it("zombies should be able to attack another zombie", async () => {
    let result;
    result = await contractInstance.createRandomZombie(zombieNames[0], {from: alice});
    const firstZombieId = result.logs[0].args.zombieId.toNumber();
    result = await contractInstance.createRandomZombie(zombieNames[1], {from: bob});
    const secondZombieId = result.logs[0].args.zombieId.toNumber();
    await time.increase(time.duration.days(1));
    await contractInstance.attack(firstZombieId, secondZombieId, {from: alice});
    expect(result.receipt.status).to.equal(true);
})
```

expect().to.equal()

감사합니다.