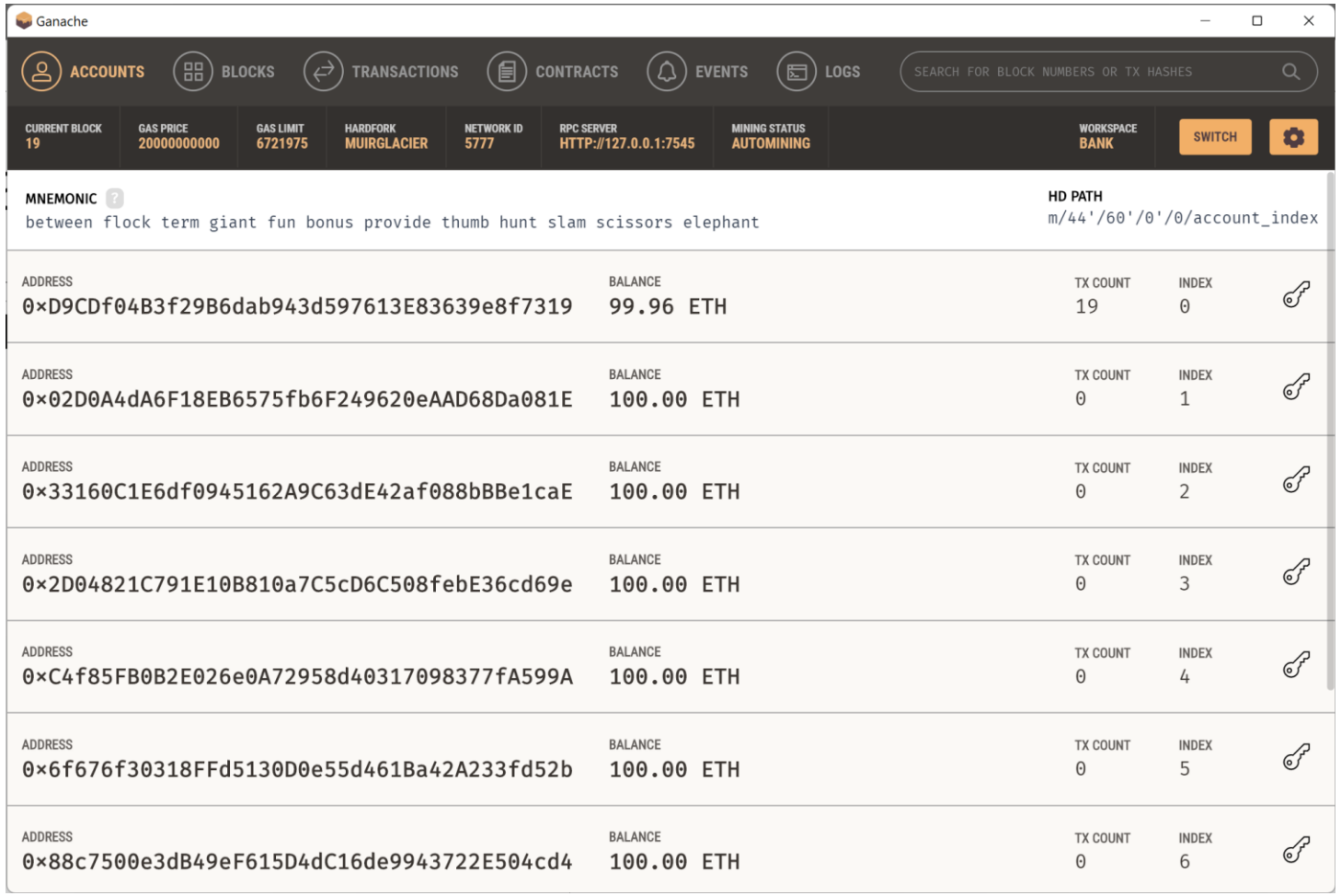
Nakamoto Bank Inc.

Ganache

Provides a simple development environment that allows us to code against.



Contracts written in Solidity

Every contract starts with a version constraint

```
pragma solidity ^0.5.0;
```

We use the ^ (caret) restriction to ensure only major version 0 can be used

Contract details

A dictionary that contains addresses and their respective amount.

The two methods utilize solidity's require methods that serves as validation for the input.

Lastly a view method, that is able to show us the current balance of a the asking user.

```
pragma solidity ^0.5.0;

contract Bank {
    mapping(address => uint) public balances;

    function deposit(uint amount) public {
        require(amount >= 0);
        balances[msg.sender] += amount;
    }

    function withdraw(uint amount) public {
        uint balance = balances[msg.sender];
        require(balance >= amount);
        balances[msg.sender] -= amount;
    }

    function getAccount() public view returns (uint) {
        return balances[msg.sender];
    }

}
```

Migration

We migrate the contract onto the blockchain to make it available to end users, using the “truffle migrate” command

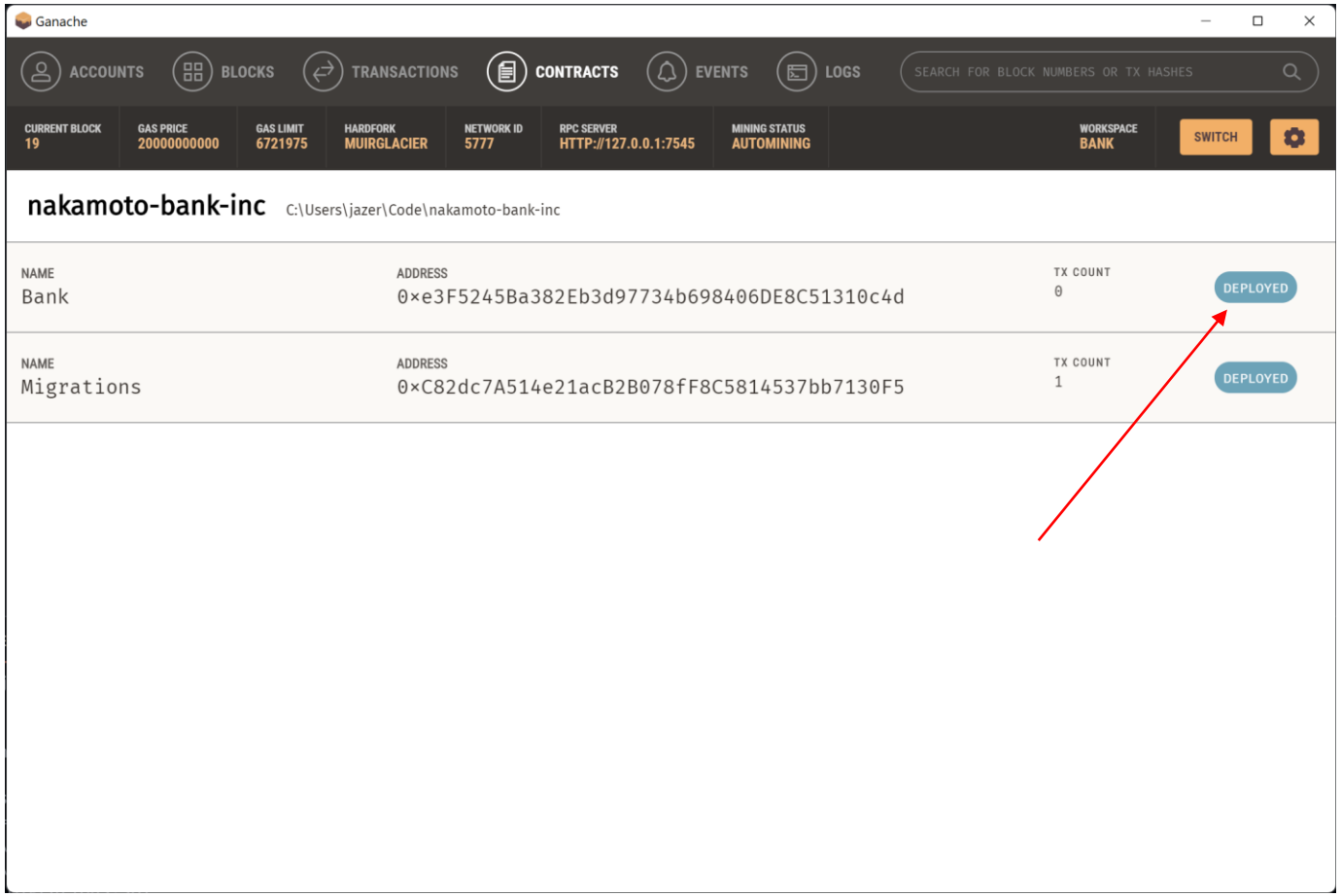
```
var Bank = artifacts.require("Bank");

module.exports = function(deployer) {
    deployer.deploy(Bank);
}
```

Migration

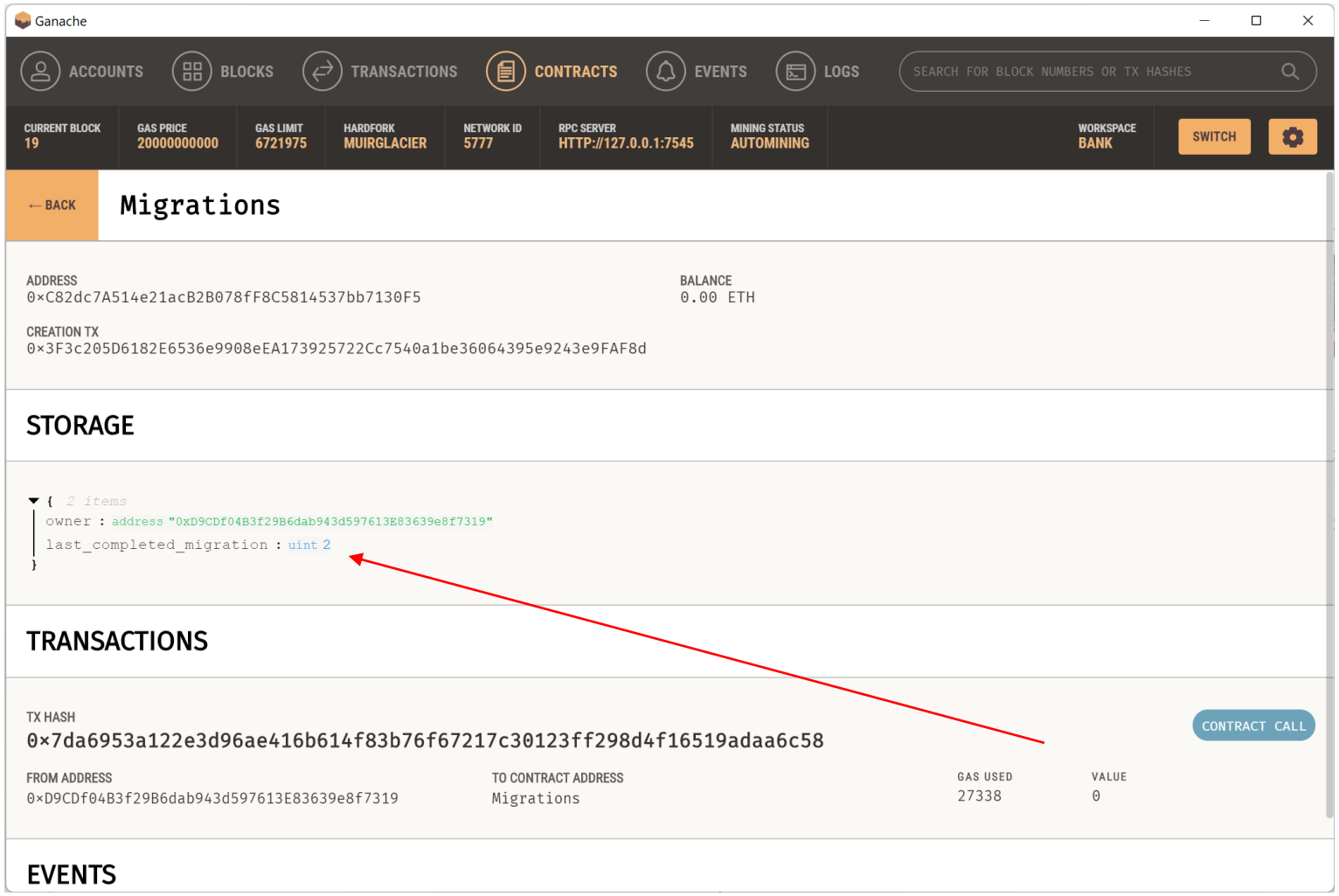
We migrate the contract onto the blockchain to make it available to end users, using the “truffle migrate” command

Verified within the Contracts tab of Ganache



Migration

The migration contract keeps track of what needs to be migrated



Testing

We chose the JS way using the Mocha framework (bundled in Truffle)

```
describe("depositing and withdrawing money", async() => {

    before(`deposit ${amountToDeposit} TNC to the first account`, async () => {
        await bank.deposit(amountToDeposit, { from: accounts[0] });
    });

    ...
})
```

Testing

We chose the JS way using the Mocha framework (bundled in Truffle)

```
describe("depositing and withdrawing money", async() => {

    ...

    it(`can get balance in TNC`, async () => {
        const balance = (await bank.getAccount({ from: accounts[0] })).toNumber();
        assert.equal(balance, amountToDeposit, `The account balance should be
${amountToDeposit} TNC.`)
    })

    it(`can withdraw ${amountToWithdraw} TNC`, async () => {
        await bank.withdraw(amountToWithdraw, {from: accounts[0]});
        const balance = (await bank.getAccount({ from: accounts[0] })).toNumber();
        assert.equal(balance, amountToDeposit - amountToWithdraw, `The account balance should
be ${amountToDeposit - amountToWithdraw} TNC.`)
    })

    ...
})
```

Testing

We chose the JS way using the Mocha framework (bundled in Truffle)

```
describe("depositing and withdrawing money", async() => {

        ...

        it(`cannot withdraw more than current balance`, async () => {
            let fails = false;
            try
            {
                await bank.withdraw(amountToDeposit + 1, {from: accounts[0]})
            }
            catch
            {
                fails = true;
            }
            assert.equal(fails, true);
        })

        it(`cannot deposit negative amount`, async () => {
            let fails = false;
            try
            {
                await bank.deposit(-1, {from: accounts[0]})
            }
            catch
            {
                fails = true;
            }
            assert.equal(fails, true);
        })

    })
```