Writing Interaction Layer

Write Interaction Layer

```
pragma solidity ^0.4.23;


contract Counter {

    event CounterIncrementedEvent(int count);
    event CounterDecrementedEvent(int count);


    int private count = 0;

    function incrementCounter() public {
        count += 1;
        emit CounterIncrementedEvent(count);
    }


    function decrementCounter() public {
        count -= 1;
        emit CounterDecrementedEvent(count);
    }


    function getCount() public constant returns (int) {
        return count;
    }
}
```

More setup

---

1. **Setup Codebase:** If you have followed part-1, you can skip this step. Otherwise checkout code from github.

```
git clone https://github.com/sarveshgs/firstBlockchainApp.git
git checkout tutorial-part-1
```

2. **Initialise node project:** Use npm to initialise node project.

```
npm init
```

3.**Web3:** Web3 is a library which helps you to interact with a local or remote ethereum node, using a HTTP or IPC connection. You can use npm to install it.

```
npm install web3 – save
```

Deploy Contract

1.Run ganache-cli: Setup ethereum node by running ganache-cli.

ganache-cli

2.Unlock Account: Unlock first account of created by ganache-cli.a.
Open truffle console:

```
truffle console
```

b. Unlock Account: Use first account address and private key printed on console when ganache-cli is ran to unlock account. This unlock account is used to deploy contract.

```
web3.personal.unlockAccount(firstAccountAdress,firstPrivateKey,15000)
```

3.Deploy contract: You can use truffle to deploy contract. Note down contract address underlined with red line in the image.

```
truffle migrate
```

Interact with Contract

I. Create a folder in your project structure named as '**interaction**' and '**interaction.js**' in interaction folder.

II. Make web3 connection with local ethereum node: ganache-cli is running at *http://localhost:8545*

```
const Web3 = require('web3');
const web3 = new Web3(new Web3.providers.HttpProvider("http://localhost:8545"));
```

Interact with Contract

III. Initialise contract: To initialise contract, we need two things

- Contract address: You must be having a contract address after deploying contract successfully in step-1
- ABI: ABI stands for *application binary interface*. Basically it's metadata of your contract in json format. You can find abi in '***build***' folder of your project. It is generated after compilation of contracts.

```
const metadata = require('../build/contracts/Counter');
const abi = metadata.abi;
const contractAddress = "0x42f18096c12db0b85ce999dbe5cceb4316f1e0be";
const counter = new web3.eth.Contract(abi, contractAddress);
```

Interact with Contract

IV. Calling contract methods: Last thing, which needs to be done is to call contract method. Calling a contract method will form a transaction in ethereum blockchain. Once the transaction is executed, it will return a transaction receipt. This receipt can be used to retrieve return value from method and events.
In below code snippet, we are calling ‘incrementCounter’ method and listening for ‘on receipt receive event’

```
const Web3 = require('web3');

const web3 = new Web3(new Web3.providers.HttpProvider("http://localhost:8545"));

const metadata = require('../build/contracts/Counter');
const abi = metadata.abi;
const contractAddress = "0x42f18096c12db0b85ce999dbe5cceb4316f1e0be";

async function getMainAccount() {
  let accounts = await web3.eth.getAccounts();
  return accounts[0];
}

async function main() {
  const mainAccount = await getMainAccount();
  const counter = new web3.eth.Contract(abi, contractAddress);

  counter.methods.incrementCounter().send({from: mainAccount})
    .on('receipt', function (receipt) {
      if (receipt.status) {
        let events = Object.keys(receipt.events);
        if (events.length > 0) {
          let event = receipt.events[events[0]];
          console.log("event Name  ", event.event);
          let returnValues = event.returnValues;
          let returnedCount = returnValues.count;
          console.log("count ", returnedCount);
        }
      }
    })
    .on('error', console.error);
}

main();
```

Interact with Contract

V. Run interaction.js: Run interaction.js using node.
node interaction/interaction.js
Great you now know, how to write smart contract and interaction layer to interact with contract methods using web3.