Information System
Lab 3

T. Back s3218147

R. Bwana s3232352

19th December, 2017

1 Introduction

This lab is split into topics. The first part covers a data aggregation proxy of a local devices, which then becomes internet-ready by sending already processed data to the cloud. Secondly, a look into the Ethereum smart-contract programming language Solidity is taken.

2 Internet of Things

The task of the Internet of Things assignment is to write a NodeJs application that reads (dummy) data and aggregates it by using the simple moving average of of the past 10 data points. If the values changes by more than a threshold, the updated value is published to the cloud service `thingspeak.com`.

The code for this is straight-forward. First a timer is created to repeatedly read a value from the device and process it.

```
/** 1s scheduler */
setInterval(function () {
    const number = numberSource();
    processReadingOnIotDevice(number);
}, 1000);
```

The data is then processed. We use as the data storage a custom implementation of an iterable queue. This simplifies the calculation of the simple moving average as well as the coordination to always push out the oldest value.

```
/** calculate the SMA of a queue structure */
function getSimpleMovingAverage(queue) {
    var val = 0;
    const iterator = new queueIterator(queue);
    while (iterator.hasNext()) {
        val += iterator.next();
    }
    val /= queue.size();
    return val;
}

const readingQueueLength = 10;
const readingQueue = new queue();
var lastMovingAverage = -1;

function processReadingOnIotDevice(number) {
    // add number to queue
    readingQueue.enqueue(number);
    if (readingQueue.size() > readingQueueLength) {
        readingQueue.dequeue();
    }

    const newMovingAverage = getSimpleMovingAverage(readingQueue);
    console.log("Using " + readingQueue.size() + " values for ma: "
         + newMovingAverage);

    if (Math.abs(lastMovingAverage - newMovingAverage) > 0.5) {
        sendToEdgeNode(newMovingAverage);

        lastMovingAverage = newMovingAverage;
    }
}
```

Finally, some code is necessary to publish the values to the cloud service.

```
/** publish a value to the cloud server */
function sendToEdgeNode(payload) {
    const API_KEY = 'VYNULCYYRUT5J6FN';
    var options = {
        host: 'api.thingspeak.com',
        port: 80,
        path: '/update?api_key='+API_KEY+'&field1=' +
            encodeURIComponent(payload)
    };

    http.get(options, function (resp) {
        resp.on('data', function (chunk) {
            console.log('sent to thingspeak');
        });
    }).on('error', function (e) {
        console.log("error sending to thingspeak: " + e.message);
    });
}
```

As it turns out, wiring up a device and make it internet-ready to become a thing of the internet is easy. The next step will be to process the data, build a pipeline and control another device with it.

3 Ethereum Blockchain

In this assignment, the scenario describes a voting system in which one person (director) asks questions about which multiple participants (shareholders) can vote. The results are only public after the director closes the question. The director can always see the result. Still, the individual votes cannot be tracked back and linked to an individual participant.

Since the assignment was not completely clear about the relation between questions and shareholders, we assume that the shareholders have to be added explicitly to each question instead of having a global pool of shareholders.

With the Ethereum Blockchain these promises about secrecy and validity can be hold. Smart Contracts pose the solution to this.

During writing the contract multiple problems arose, which all have been solved. Still they are worth mentioning as Blockchain is an uprising technology and contracts are permanent once they are deployed. When they are written and released on the public blockchain, anybody is able to interact with it, unless explicitly hindered by the smart contract itself.

1. Contracts cannot deploy own contracts but rather work within their own data structures (struct).
2. For the notification of parties outside of the contract, events are fired.
3. Mappings are a challenging implementation of associative arrays to use. Values can be indexed by a non-monotonic key. But it lacks the functionality of checking whether a specific key is valid. In Solidity all keys are always valid, independent whether it was inserted or not. If a new key is accessed for reading, the object value is created anyway and initialized to zero. This requires a additional field ($flag$) to indicate whether the access value is valid in the context of the application.

   Therefore, in this case the *Shareholder* structure has a dummy field *exists* just to verify that the shareholder was really created. Otherwise it cannot be verified if the specific shareholder was added at some point to mapping and possibly removed again.
4. Protecting the contract methods to be accessible only by the correct individual (identified by the address of the sender) is key. This can be done by `require(msg.sender == director)`. It ensures that only the director is allowed to add new questions, close questions. . . . Another advantage of the *require* method is, that an exception is thrown to indicate the forbidden action as opposed to returning successfully.
5. Hiding of individual votes is possible, by just storing the accumulated amount. Since the director is able to see the votes at any time and infinitely often, the votes are not completely private to him. This could be changed if also the scenario is changed to reflect this.

6. Returning multiple values at once is possible. This helps in this scenario with showing the result of a single question. Instead of defining the output with -1 (question denied), 0 (tie) and 1 (accepted), a combination of two booleans can be used to indicate that a question has a decision (not a tie) and secondly what the outcome is.

7. A further improvement to the above idea, is the usage of enums, which is the chosen solution to indicate the outcome of the asked questions.

Following is the code of the complete ethereum contract.

```
pragma solidity ^0.4.0;

/** @title Shareholder Voting Contract */
contract Ballot {
    struct Shareholder {
        // Whether the shareholder has already voted
        bool hasVoted;
        // Key exists flag.
        bool exists;
    }

    struct Question {
        // Mapping of the allowed shareholders for this question
        mapping(address => Shareholder) participants;
        // The actual question
        string question;
        // Can the question still be answered?
        bool isOpen;
        // The outcome counter of the question
        int8 result;
    }

    event QuestionAsked (
        address indexed _from,
        uint256 indexed _questionIndex
    );
    event QuestionVoted (
        address indexed _from,
        uint256 indexed _questionIndex
    );

    enum QuestionOutcome {Accepted, Tied, Denied}

    address director;
    Question[] questions;

    /** @dev Creates a new contract
      */
    function Ballot() public {
        director = msg.sender;
    }

    /** @dev Lets everyone see the question
      * @param index The number of the question
      * @return res The actual question
      */
    function seeQuestion(uint8 index) public constant returns (
        string) {
        return (questions[index].question);
    }

    /** @dev Lets the deployer add a new question
     * (Sends out an event after)
      */
    function addQuestion(string questionStr) public {
        require(msg.sender == director);

        questions.push(Question({question: questionStr, isOpen:
            true, result: 0}));

        // send notification event
        QuestionAsked(msg.sender, questions.length);
    }

    /** @dev Add a new shareholder to the question
      * @param index The number of the question
      * @param shareholderAdr The address of the shareholder
      */
    function addParticipant(uint8 index, address shareholderAdr)
        public {
        require(msg.sender == director);

        Question storage question = questions[index];
        require(question.isOpen);

        Shareholder storage shareholder = question.participants[
            shareholderAdr];
        shareholder.exists = true;
        shareholder.hasVoted = false;
    }

    /** @dev Remove a shareholder from a question
      * @param index The number of the question
      * @param shareholderAdr The address of the shareholder
      */
    function removeParticipant(uint8 index, address shareholderAdr)
         public {
        require(msg.sender == director);

        Question storage question = questions[index];
        require(question.isOpen);

        Shareholder storage shareholder = question.participants[
            shareholderAdr];
        require(shareholder.exists);
        require(!shareholder.hasVoted);

        delete question.participants[shareholderAdr];
    }

    /** @dev Let a shareholder vote on a question
     * (Sends out an event after)
      * @param index The number of the question
```

```
       * @param vote the actual vote
       */
    function voteQuestion(uint8 index, bool vote) public {
        Question storage question = questions[index];
        require(question.isOpen);

        Shareholder storage shareholder = question.participants[msg
            .sender];
        require(shareholder.exists);
        require(!shareholder.hasVoted);
        shareholder.hasVoted = true;
        if(vote) {
            question.result = question.result + 1;
        } else {
          question.result = question.result - 1;
        }

        // Inform that a shareholder has voted
        QuestionVoted(msg.sender, index);
    }

    /** @dev Lets the deployer close a question/ stop the voting
      * @param index The number of the question
      */
    function closeQuestion(uint8 index) public {
        require(msg.sender == director);

        Question storage question = questions[index];
        require(question.isOpen);
        question.isOpen = false;
    }

    /** @dev See the result/outcome of a question
      * @param index The number of the question
      * @return validDecision Whether the question has a clear
          outcome (not a tie)
      * @return questionAccepted The outcome (TRUE=accepted, FALSE=
          denied)
      */
    function questionResult(uint8 index) public constant returns (
        QuestionOutcome) {
        Question storage question = questions[index];
        if(msg.sender != director) {
            // Everyone except the director has to wait for the
                question to be
            // closed to see the result
            require(!question.isOpen);
        }

        if(question.result != 0) {
            if(int8(0) < question.result) {
                return QuestionOutcome.Accepted;
            }
            return QuestionOutcome.Denied;
        }
        return QuestionOutcome.Tied;
    }
```

```
}
```