University of Zurich^UZH

SOFTWARE PROJECT – Communication Systems Group, Prof. Dr. Burkhard Stiller

Design and Implementation of a Smart Contract Creator Framework for IoT Devices

Sandro Luck
Zurich, Switzerland
Student ID: 13-927-769

Supervisor: Sina Rafati Niya, Thomas Bocek
Date of Submission: 14. August 2017

University of Zurich
Department of Informatics (IFI)
Binzmülestrasse 14, CH-8050 Zürich, Switzerland

ifi

Software Project
Communication Systems Group (CSG)
Department of Informatics (IFI)
University of Zürich
Binzmühlestrasse 14, CH-8050 Zürich, Switzerland
URL: http://www.csg.uzh.ch/

Abstract

Blockchains have made tremendouse changes during the last years on financial and non financial applications. A promising new usage of the blockchain technology have been Smart Contracts, Turing complete programms globally evaluated on the blockchain. Generating vast amounts of Smart Contract for Sensor based use cases has been a time consuming and difficult task. The Smart Contract Generator Framework allows the user to automate the creation of Smart Contracts for Internet of Things related use cases and speed up the creation and monitoring process. The Django based graphical web application aids the user at generating and writing these Smart Contracts. This document describes the implementation and usage of the Smart Contract Generator Framework `https://github.com/SandroLuck/SCGenerator` .

Acknowledgments

First of all, I would like to express my sincere gratitude to my supervisors Sina Rafati Niya and Dr. Thomas Bocek for their support, help, expertise and time they dedicated to me. I thank Prof. Dr. Burkhard Stiller, the head of the Communication Systems Group at the University of Zurich, for making this project possible.

Contents

Chapter 1

German summary

1.1 Einleitung

Blockchain ist ein dezentraler public ledger, diese Technologie hat sich in den letzten Jahren sehr stark entwickelt und in verschiedensten Branchen Anwendungsfälle gefunden. Blockchains ermöglichen es den Benutzern global Daten zu teilen, zu evaluieren und zu überwachen. Sensoren, welche auf der ganzen Welt verteilt sind, sammeln stetig mehr Daten und könnten, unter Benutzung der Blockchain Technologie, diese auch öffentlich und unverfälscht zugänglich machen. In letzter Zeit wurde auch vermehrt die Kombination von Internet of Things (IoT) Geräten und der Blockchain Technologie vorangetrieben. Der Blockchain Technologie sowie IoT Geräten wird auch nachgesagt, dass sie eine entscheidende Rolle in der Industrie 4.0 spielen werden. Da das Interesse an den beiden genannten Technologie gross ist, wäre es von Vorteil ein funktionales und vertrauenswürdiges Framework zu erstellen, welches den Zugang zu diesen Technologien erleichtert. Deshalb hat diese Arbeit zum Ziel ein Framework zur erstellen, welches das Aufsetzen sogenannter Smart Contracts erleichtert und beschleunigt. Smart Contracts sind Programme, welche in der Programmiersprache Solididty geschrieben werden, und in der Ethereum Blockchain von vielen Akteuren evaluiert werden, um dessen Vertrauenswürdigkeit / Unverfälschtheit zu garantieren. Weitere Informationen zu dieser Technologie können in der beiliegenden Englischen Version des Softwareprojektdokumentes gefunden werden oder auch auf der Webseite der Ethereum Stiftung (https://www.ethereum.org/).

[Referenzen] Da es sich bei disem Deutschen Text um eine Zusammenfassung handelt, wurden die Referenzen und Begründungen weggelassen. In der Englischen Version werden die Referenzen und Begründungen allerdings ausführlich aufgeführt.

1.2 Ziele

Das Ziel dieses Softwareprojektes ist es, eine voll funktionsfähig Webapplication zu designen und zu implementieren. Diese Webapplikation erleichtert und automatisiert das erstellen von Smart Contracts für IoT und Blockcahin basierte Anwendungsfälle. Bei der Entwicklung der Applikation soll darauf geachtet werden, dass die Applikation sowie die produzierten Smart Contracts einfach erweitert und geändert werden können. Es sollen diverse neue Smart Contracts, aufgrund der vorhandenen Templates mit unterschiedlichen Thresholds und/oder anderen neuen Standards oder Geräten erstellbar sein. Die daraus entstehende Applikation wird in Zukunft dafür verwendet, die Quellcode Produktion zukünftiger Projekte zu beschleunigen und zu vereinfachen.

[Orginal Text der Aufgabenstellung] Die Aufgabenstellung wurde auf Englisch verfasst. Der Abschnitt Ziele ist eine Übersetzung des Autors.

1.3 Resultate

Entstanden ist eine Django basierte Webapplikation, welche eine einfach zu bedienenende Grafische Oberfläche bietet. Die Datenbank dieser Applikation kann über diese Grafische Oberfläche, mithilfe von Formularen, mit Daten gefüllt werden, welche die Erstellung von Smart Contracts automatisiert. So erstellte Smart Contracts können dann über die Webapplikation heruntergeladen und mithilfe der Ethereum Blockchain ausgeführt werden. Auch besteht die Möglichkeit die aufgeführten Smart Contracts mithilfe des Smart Contract Monitors zu überwachen und deren Event Verlauf nachzuschlagen. Die Applikation funktioniert sowohl auf Windows und Linux, als auch au OSX, wurde aber hauptsächlich auf Linux und Windows entwickelt. Es werden die Datenbanken SQLite, MySQL und PostrgresSQL unterstützt.

Chapter 2

Introduction

2.1 Motivation

Blockchain is a decentralized and public ledger that has made tremendous changes during last years on financial and non financial applications. Blockchains offer their users multiple advantages such as sharing their data, evaluating it and monitoring it.
Data collected by sensors, scattered all around the world with limitless types and regardless of functionality or use case, could be accessed by blockchain clients with distributed public accessibility with an almost undeniable credibility [22], thus delivering publicly accessible data for everyone including industries, researchers or motivated individuals and improving transparency and insight in various aspects of life.
Use cases for blockchain-based systems that have been focused recently in the industry and academia are integrating Internet of Things (IoT) and blockchain based approaches to extend there usability. Also, both blockchain and IoT devices are said to play a major role in Industry 4.0 [20]. Industry 4.0 describes factories widely automated by automatic machines which are interconnected in the IoT and are able to communicate, produce and monitor in an automated way [21].
As the interest in IoT applications and blockchain is increasing, a functional, trustable framework to provide Smart Contracts (SC) with high efficiency and least possible costs is needed. Most of the use cases are similar and can be rewritten with the SC Generator framework, with less time and error susceptibility. This project aims at reducing the installation time and error susceptibility and solving some of the most common sensor and blockchain related use cases by providing templates, which can be extended easily.

2.2 Blockchains

A blockchain is a decentralized and public ledger which allows for the decentralized storage and evaluation of data [6]. Blockchain-based systems provide the possibility for their users to insert their data into this distributed ledger. These so called blocks are inserted in a chronological order and a so called block number is constantly incremented and can be

used to assure global order [6]. The entire user base of a blockchain is guaranteed to have the same view of the state of the blockchain. Each member of the blockchain is identified by one or multiple unique identities called addresses. With each address a set of transactions is stored in the blockchain, such that one knows how much "money" each individual address can use.
While in the beginning blockchain where mainly a tool for transferring money, without relaying on a centralized organization, the focus is currently shifting towards a broader usage. Some of the newer uses of blockchains include digital identity [12], e-voting and governance [13].

2.2.1 Cryptocurrencies

Associated with most blockchains is a so called cryptocurrency. A cryptocurrency is a collection of digital entries in the blockchain, which can be considered to some extend as money, but cryptocurrencies until now have not been issued nor guaranteed by states. In this work a cryptocurrency, also called an electronic coin, is defined as a chain of digital signatures where one specific coin can only be hold by one individual address [7]. Cryptocurrencies can be transferred through the blockchain from one address to another address. The processes of transferring said coins costs a small amount of the cryptocurrency.
These cryptocurrencies, in most countries, are freely tradeable but do not have a guaranteed value in dollars, CHF or any other real currency. But due to the value assigned to the cryptocurrency by the market, the blockchain can be described as A Peer-to-Peer Electronic Cash System as Satoshi Nakamoto (the creator of Bitcoin) described it in 2008 [7]. Usually the upper limit of cryptocurrencies is defined or converges to a value, but this is not guaranteed in all cases.

2.2.2 Mining cryptocurrencies

The initial generation of these cryptocurrencies is generated through a process called mining. While the mining algorithm varies depending on which cryptocurrency one is mining, generally one can say that the person mining has to dedicate computer hardware and electricity to be awarded a unit or denomination of the cryptocurrency. It is assumed that miners only mine when the return for mining, mainly defined by the electricity cost and mining difficulty, is profitable. The mining difficulty increases over time, how and how much this happens depends on the specific implementation of the various cryptocurrencies. The people or institutions which participate in this process are called miners.
Each transaction is evaluated by a group of miners, and the transaction fee associated with each transaction is usually given to those miners in order to evaluate the transactions, for this purpose each miner has a copy of the blockchain on his device. A miner can also be defined as a person which provides the blockchain ecosystem with computation power and hardware.

2.2.3 Ethereum

A new and increasingly popular implementation of this concept is Ethereum [8]. Ethereum, at the time writing has a market capitalization, of above $20,000,000,000 [9]. It is worth noting that the price for Ethereum is fluctuating heavily (between January 2017 and June 2017, Ethereums market capitalization fluctuated between $800,000,000 and $30,000,000,000) [9]. From here on all information regarding blockchains are specifically meant for Ethereum.
Ethereum is maintained and extended constantly, through a group called the Ethereum Foundation [8]. The cryptocurrency associated with Ethereum is called Ether, one Ether can be subdived into multiple smaller parts as shown in table 2.1.

Table 2.1: Ether denomination [14]

	1 Ether is:
wei	10^{18}
Kwei	10^{15}
Mwei	10^{12}
Gwei	10^{9}
szabo	10^{6}
finney	10^{3}
ether	1

Ethereum offers all capabilities described in the previous sections, but adds some new features. Ethereum additionally offers the users the possibility to save Turing-complete code called Smart Contracts (SCs) into the blockchain. These SCs are simultaneously evaluated by multiple blockchain users to ensure everyone has the same state of the contracts. As with the transactions evaluating and executing the SC costs a small amount of Ether for the owner of the SC.

2.2.4 Smart Contracts and Solidity

Real contracts (juristic meaning) are a key legal instrument for private operators as they execute changes in their legal relations or try to prepare for future turns of events [1]. Such traditional contracts are often formulated in natural languages and have to be interpreted. A so called Smart Contract (SC) is a digital program which is evaluated in the blockchain and should produce the same result no matter by which computer it is evaluated. It is worth noting that SCs do not only offer the functionality of some traditional contracts but can theoretically be used for everything traditional programs can be used [1]. SCs are written in a programming Language called Solidity [10]. Solidity has been described as a contract-oriented and high-level language whose syntax and usage feels similar to JavaScript and runs on the Ethereum Virtual Machine (EVM) [10]. This virtual machine is a consensus-based globally executed virtual machine [11]. The specific part of the Ethereum protocol, which handles internal state and computation, is referred to as the Ethereum Virtual Machine [11]. This EVM is what executes the SCs and guarantees that its code is evaluated correctly. It is worth noting that all the guarantees given by the blockchain may only hold as long as the majority of the network is operating honestly.

2.2.5 Smart Contract Costs

While executing on the EVM the SC cost an amount of Ether [15]. This amount is a execution fee and usually called gas [15]. This gas is bought from the miners executing the contracts on the blockchain [15]. Ether and gas are two decoupled currencies, where gas is used to pay EVM execution costs and is usually bought via the Ethereum-Wallet while deploying or using a contract. The price of gas is defined by the miner community, since they define at which gasprice they execute a contract or transaction. The Total costs of executing can be calculated by

$$\text{Total cost} = \text{gasUsed} * \text{gasPrice}$$

Where gasUsed is the cumulative gas costs of each operation in the SC. The gas costs for each operation can be seen at the official Ethereum spreadsheet [16]. A reduce version is shown in table 2.2.

Table 2.2: Simplified table describing gas costs per operation [16]

Operation Name	Gas Cost	Additional Infos
step	1	default per execution
stop	0	stop contract execution
suicide	0	delete a contract
sha3	20	hashing
sload	20	get from storage
sstore	100	put into storage
balance	20	
call	20	read-only call
memory	1	for every word expanding memory
txdata	5	every byte of data
transaction	500	base fee transaction
contract creation	53000	cost for deploying a contract

The gasPrice is defined by the miners offering their computational power for Ether. The current price can always be found at `https://etherscan.io/chart/gasprice`. Since the gas price changes and each miner might offer a different gasPrice, when refering to the gas price the average gasPrice is ment. A typical gasPrice is around 23000000000 wei in July 2017.

2.3 IoT and blockchains

Internet of Things (IoT) devices are devices with an internet connection which are usually monitored or controlled by a variety of other IoT devices. IoT devices are often used in decentralized systems.
Combining the ability of blockchains and IoT devices to save or monitor data is desirable, since in a big network of devices a decentralized consensus finding is crucial. It ensures all devices have the same state of the collected data and outsources the computational power necessary to implement certain IoT use cases to the blockchain. Currently setting up bigger networks of sensors and serving all of the them with a SC is time consuming, this is why implementing a SC Generator framework to simplify and speed up this process is desirable.

2.4 Code Generation

Generating code in an automatic way has been done for a variety of use cases and programming languages. There is a variety of names used to describe said processes including automatic programming, generative programming or source code generators . There are two fundamentally different approaches to produce source codes [25, 26]. One is using templates, a reduced version of the code to produce, into which the application inject variables and the other approach would be using some advanced technics related to machine learning to produce source code [25, 26]. The application follows the template based approach, since most of the desired contracts to be produced are very similar.

2.5 Description of Work

This work describes how to produce a SC generating web-application. The source code of this application can be found at `https://github.com/SandroLuck/SCGenerator` .
The application Smart Contract Generator is a web based solution for SC creation mainly focusing on IoT devices.
It has be designed to be extendable and could feature more templates in the future. But as of the time writing it features two templates both written for the Ethereum-based programming language Solidity and optimized for pragma solidity 0.4.11. Since in the area of IoT devices often a big amount of devices have to be served with a relatively similar SC, a template based source code creation approach has been chosen to serve the IoT use cases which will be discussed in chapter 4.3.

2.6 Thesis Outline

In Chapter 1 a German summary of this project is given. In Chapter 2, the basic concepts used in this work such as blockchains, Ethereum, SCs, Code generation and IoT devices are explained and set into context as well as our approach on how to automated some of the time consuming aspects of their usage. In Chapter 3 related works are discussed. In Chapter 4 various aspects of the implementation of this application are discussed such as, the user interface, the database, the generated SCs and certain restriction that have influenced this application. In Chapter 5 various tests for all templates are featured as well as possible ways to optimize these even further. In Chapter 6 the Summary and Conclusions of this work are given. In Chapter 6 all bibliographic references are listed. In chapter 6 the abbreviations are listed. In the appendix the installation instructions and contents of the CD are listed.

Chapter 3

Related Work

While there are projects and publications discussing Blockchains and its usage for IoT devices [2]. They mainly focus on saving data into the blockchain, often to increase the security such systems offer. The usage of Smart Contracts for IoT use cases has been evaluated by K. Christidis [3] and has been described as a potential game changer for several industries and use cases.
Also, there have been several approaches for different languages and use cases to automate the production of certain code snippets or entire code bases [4].
But to our best knowledge there has until now been no approach to generate Smart Contracts for IoT devices in an automated manner. This might be due to the relatively new technologies involved or the current restrictions of these systems.

Chapter 4

Development

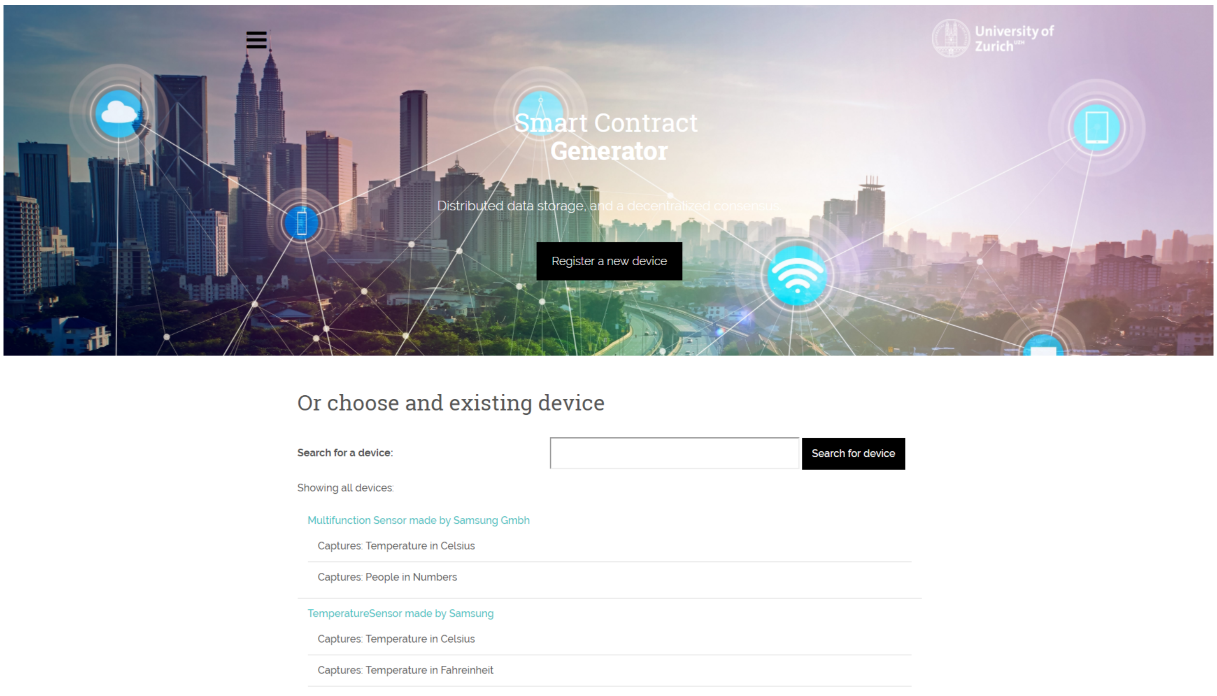
4.1 User interface

One of the first questions that arise when designing a tool to increase development speed for IoT related Smart Contracts (SC), is how should the values be specified by the user. Since most environments, using blockchains and IoT devices, should be connected to a intranet or the internet a web-based application seems to offer all necessary interactions for the user.
Therefore a Django based web application [23] has been designed to offer users the capability to easily integrate this application into their existing infrastructure. The entire application was designed to scale up to a size useful for small and intermediate companies. All web-based links and templates have been designed to be accessible by anyone able to access the described web-based solution. This design choice has been made due to the often very limited capabilities of IoT sensors such that after a template has been created, the produced SC can be easily download and deployed, even from devices without a graphical user interface.
While it is possible to access the page via handheld devices (e.g. mobile phones and tablets), it has been optimized for computer monitors. This decision has been made, since deploying SCs from handheld devices is currently not recommendable due to the lack of available software. All of the following screenshots have been made at a resolution of 1920 x 1080 and using the Google Chrome web browser, the appearance of the web-application might differ slightly depending on the resolution of the browser view and the browser you are using.

4.1.1 Starting page

The starting page, as shown in figure 4.1, is the first view a new user will come in contact with. Due to the application being optimized for sensors, the first option is to ”Register a new device”. Alternatively, the user can choose a device already registered (written in light blue).
For each device all values captured by the device are shown.

To easily search for existing devices a searching option is available, which will look for devices which contain the search string in either the devices name or the manufacturing company.
In the top left corner the navigation menu can be opened to simplify the navigation process.



Or choose and existing device

Search for a device:

Showing all devices:

Multifunction Sensor made by Samsung Gmbh
Captures: Temperature in Celsius
Captures: People in Numbers

TemperatureSensor made by Samsung
Captures: Temperature in Celsius
Captures: Temperature in Fahreinheit

Figure 4.1: SC Code-generator web application - starting page.

4.1.2 Navigation Menu

After clicking the navigation button, always located in the top left of the web application, a navigation menu is presented to the user. Via the navigation menu the user can easily find the most useful aspects of this application, as illustrated in figure 4.2. The navigation item currently selected by the user is highlighted in blue.

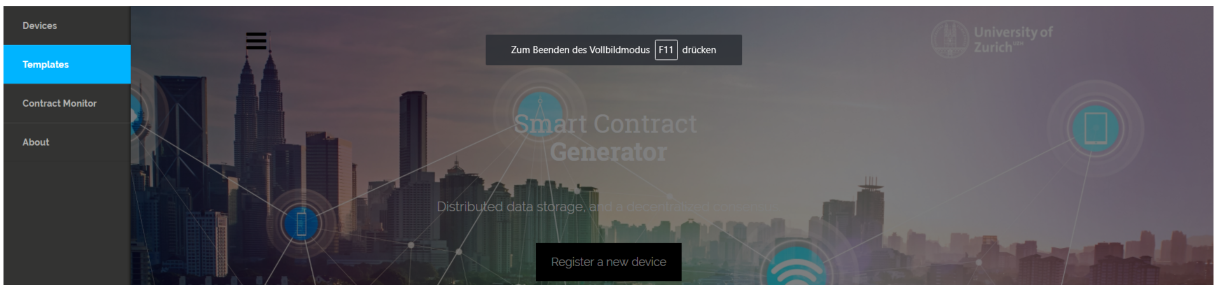


Figure 4.2: SC Code-generator web application - navigation menu.

In the current version of the application views for devices, templates, the about page and a Smart Contract Monitor can be found. These will be discussed and shown in the following sections.

4.1.3 Header

As shown in figure 4.3 the header is present on every page and shows the navigation menu (circled red) and the logo and buttons for various interactions.

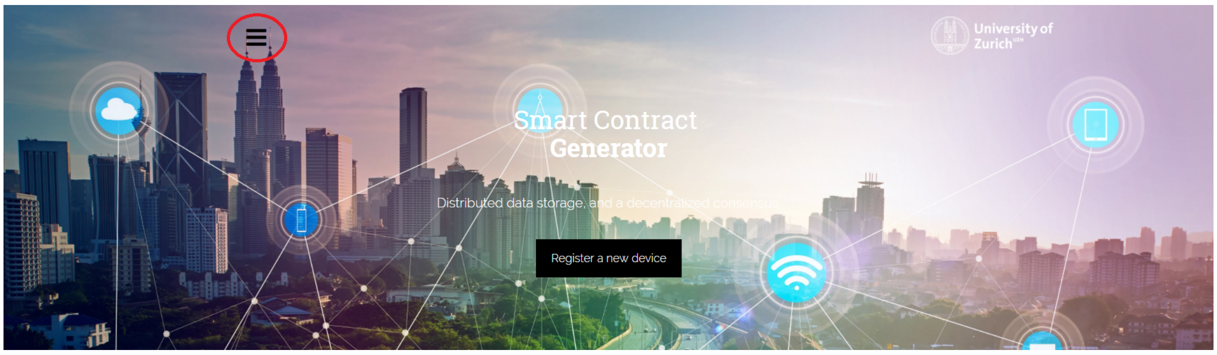
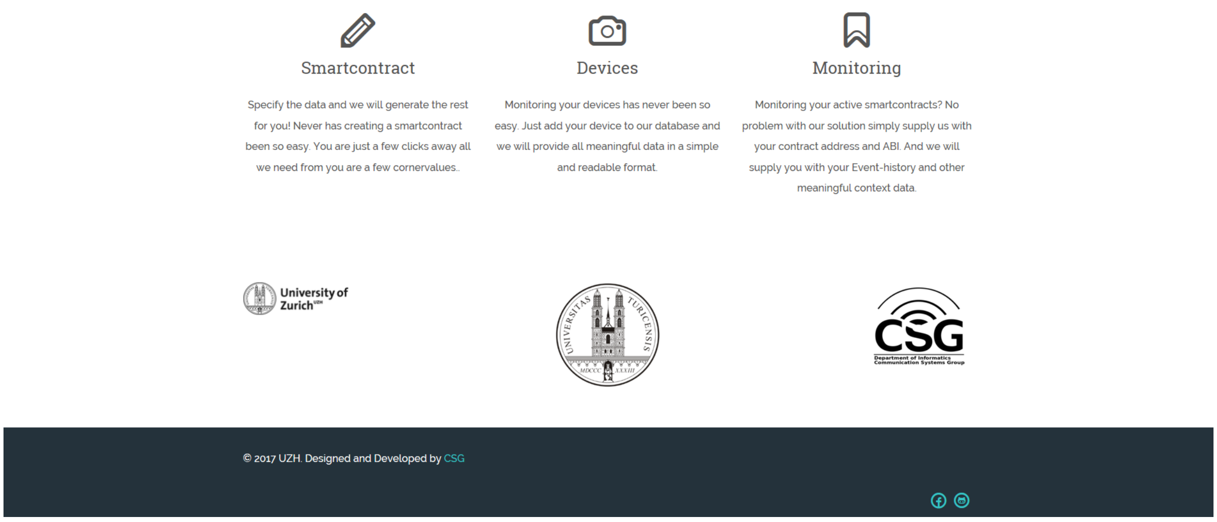


Figure 4.3: SC Code-generator web application - page header.

4.1.4 Footer

Each page features the page footer, as shown in figure 4.4, which informs the user about the features of this application, as well as links to various owner specific links.



Smartcontract

Specify the data and we will generate the rest for you! Never has creating a smartcontract been so easy. You are just a few clicks away all we need from you are a few cornervalues.

Devices

Monitoring your devices has never been so easy. Just add your device to our database and we will provide all meaningful data in a simple and readable format.

Monitoring

Monitoring your active smartcontracts? No problem with our solution simply supply us with your contract address and ABI. And we will supply you with your Event-history and other meaningful context data.

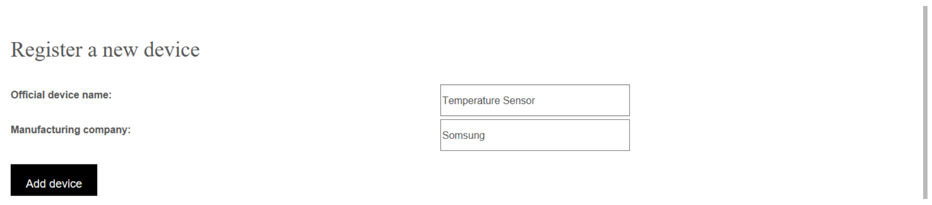
University of Zurich^UZH
UNIVERSITAS TURICENSIS MDCCC XXXIII
CSG
Department of Informatics Communication Systems Group

© 2017 UZH. Designed and Developed by CSG

Figure 4.4: SC Code-generator web application - page footer.

4.1.5 Register a new device

Registering a new device is the starting point for creating a new Smart Contract. This is illustrated in figure 4.5.
To uniquely identify a device the user has to enter the device name and the manufacturing company, since certain device names are used by multiple companies. Once a device is registered the user can add metrics it captures and reuse it for multiple templates.



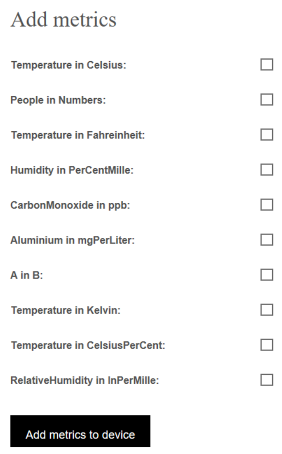
Register a new device

Official device name:	Temperature Sensor
Manufacturing company:	Somsung

Figure 4.5: SC Code-generator web application - ”register device” form.

4.1.6 Adding metrics to the device

A Sensor captures multiple metrics, which have to be added to the sensor or sensor network. All metrics can simply be checked and after accepting are added to the device, as illustrated in figure 4.6.

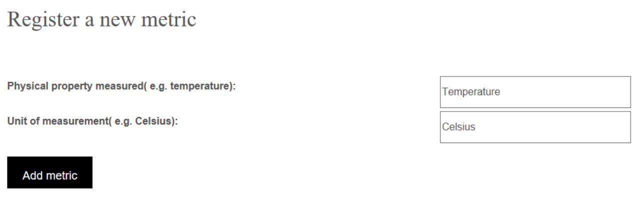


Add metrics

Temperature in Celsius:	☐
People in Numbers:	☐
Temperature in Fahreinheit:	☐
Humidity in PerCentMille:	☐
CarbonMonoxide in ppb:	☐
Aluminium in mgPerLiter:	☐
A in B:	☐
Temperature in Kelvin:	☐
Temperature in CelsiusPerCent:	☐
RelativeHumidity in InPerMille:	☐

Figure 4.6: SC Code-generator web application - ”add metrics to the device” form.

If the metric to be used is not yet in the data base the user can simply define a new metric such that all use cases can be included. This process is shown in figure 4.7

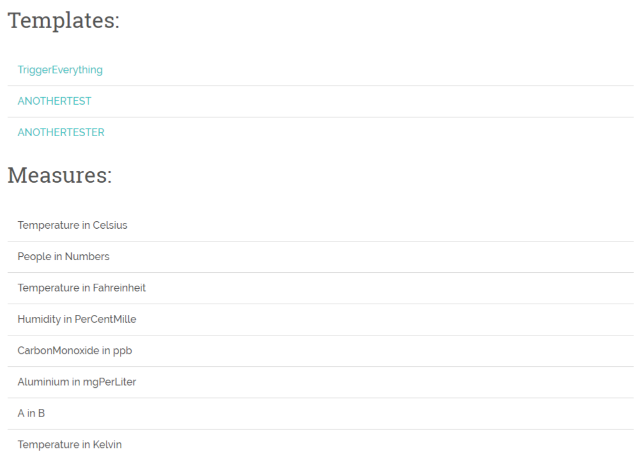

Register a new metric

Physical property measured(e.g. temperature):	Temperature
Unit of measurement(e.g. Celsius):	Celsius

Figure 4.7: SC Code-generator web application - ”register a new metric” form.

4.1.7 Device Overview

To display all information available for the device there is a device overview page. This page also offers the possibility to create new templates or add more metrics to the devices. This view is shown in figure 4.8.


Templates:

- TriggerEverything
- ANOTHERTEST
- ANOTHERTESTER

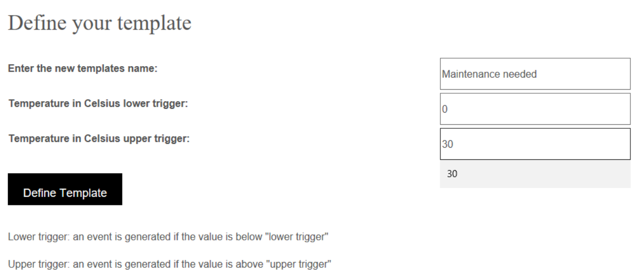
Measures:

- Temperature in Celsius
- People in Numbers
- Temperature in Fahreinheit
- Humidity in PerCentMille
- CarbonMonoxide in ppb
- Aluminium in mgPerLiter
- A in B
- Temperature in Kelvin

Figure 4.8: SC Code-generator web application - device Overview.

4.1.8 Template Creation

A template is always related to one specific device and features all metrics the device captures. To create a template the user needs the name, which will later be used for the Smart Contract name, and the trigger values to be evaluated by the Smart Contract as shown in figure 4.9. For each device there can be multiple templates which are stored under a similar URL, such that upgrading existing contracts or replacing them is simplified.


Define your template

Enter the new templates name:	Maintenance needed
Temperature in Celsius lower trigger:	0
Temperature in Celsius upper trigger:	30
	30

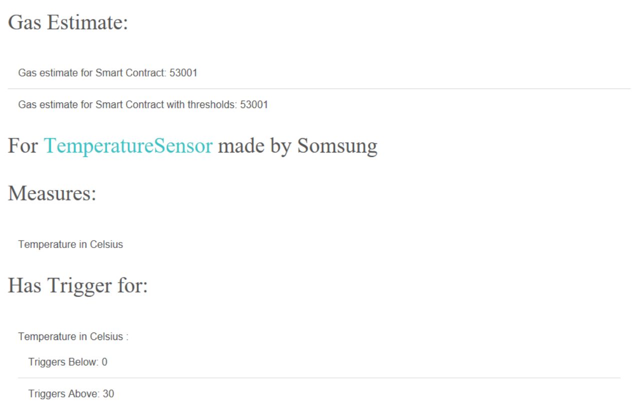
Lower trigger: an event is generated if the value is below "lower trigger"

Upper trigger: an event is generated if the value is above "upper trigger"

Figure 4.9: SC Code-generator web application - template creation process.

4.1.9 Template Overview

To display all information available for the template there is a template overview page shown in figure 4.10. This page shows all trigger values associated with the template, as well as the contract deploying costs which are further explained in 4.4.


Gas Estimate:

Gas estimate for Smart Contract: 53001

Gas estimate for Smart Contract with thresholds: 53001

For TemperatureSensor made by Somsung

Measures:

Temperature in Celsius

Has Trigger for:

Temperature in Celsius :

Triggers Below: 0

Triggers Above: 30

Figure 4.10: SC Code-generator web application - template overview.

4.1.10 Downloading Smart Contracts

After defining all necessary information to create a Smart Contract, the available Smart Contracts can be downloaded from the template overview shown in figure 4.11. The currently available Smart Contracts are described in the section 4.3. After downloading the Smart Contract, the user can deploy them through the Ethereum-Wallet [17] or similar tools like the geth-tool [18].

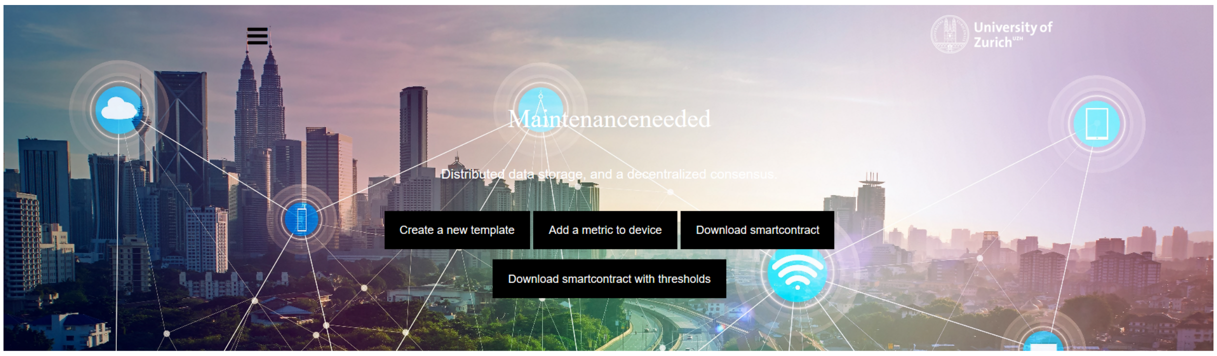
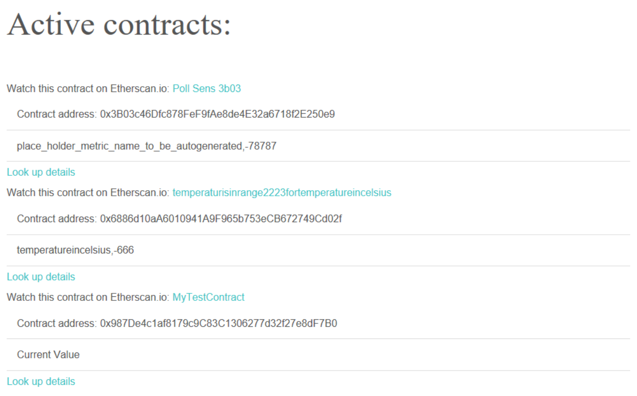


Figure 4.11: SC Code-generator web application - download section.

4.1.11 Contract Monitor

After deploying the SC the user can register his contract, using the contract address and the contract ABI (Application Binary Interface), both can be found in the Ethereum-Wallet. After registering a Smart Contract it will be added to the Contract Monitor as shown in figure 4.12. The Contract Monitor feature for each SC includes the name (defined by the user and only used for displaying purposes), its address, the last Event (e.g. current temperature) and a link to Etherscan.org which has some more advanced Monitoring qualities. It is worth noteing that the application listens to the Ethereum blockchain that has an open rpc interface on the default geth port (8545), such that it can be used for testnets as well as the mainnet or private blockchains.



Active contracts:

Watch this contract on Etherscan.io: Poll Sens 3b03

Contract address: 0x3B03c46Dfc878FeF9fAe8de4E32a6718f2E250e9

place_holder_metric_name_to_be_autogenerated,-78787

Look up details

Watch this contract on Etherscan.io: temperaturisinrange2223fortemperatureincelsius

Contract address: 0x6886d10aA6010941A9F965b753eCB672749Cd02f

temperatureincelsius,-666

Look up details

Watch this contract on Etherscan.io: MyTestContract

Contract address: 0x987De4c1af8179c9C83C1306277d32f27e8dF7B0

Current Value

Look up details

Figure 4.12: SC Code-generator web application - contract monitor.

Also, there is a detailed view for all Events fired by the Smart Contract. This view is mainly useful for debugging purposes and shows all information associated with the Events. The presentation form of this data has been kept simple to support all Smart Contracts, also those possibly customized by the user.

4.2 Database

Since the main use case of this framework is to provide SC code templates for different IoT use cases, the application focused on only saving properties to the database which are relevant to every IoT device.

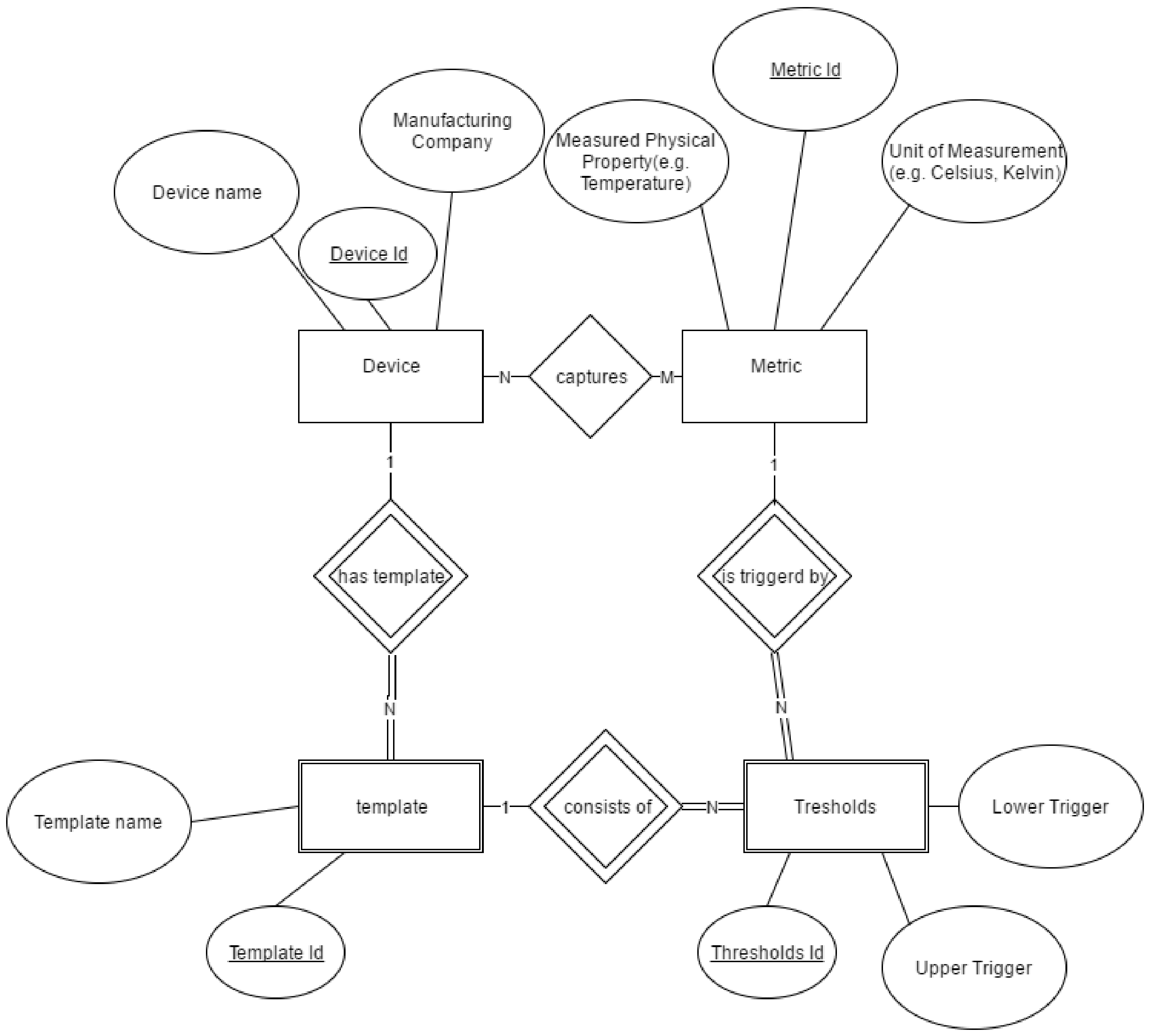


Figure 4.13: ERM of the database for this application.

As Shown in figure 4.2 the application saves different properties relating to devices. The application assumes that every device can be uniquely identified by its official name and its manufacturing company. Further the application assumes that each device (mostly thought of as a sensor) captures n $\in \mathbb{N}$ metrics.
One problem of defining the data base was to find a data model to describes all metrics a device might capture. While there seems to be no ideal solution, describing each metric with it's physical property (e.g. Temperature or Aluminium) and it's unit of measurement

(e.g. Celsius or mg/L) seems to offer potential querying possibilities in the future. The user could look for all temperature related data or only for the data in Fahrenheit.
For each of the defined devices the user can create multiple templates which all have one threshold for each of the metrics associated with the device. Each Threshold is for a specific metric and features two numbers, lower trigger and upper Trigger. While upper trigger fires when the number measured by the device is above the trigger the lower trigger fires when the number measured by the device is below said value.
All properties discussed are strings except for lower trigger and upper trigger which are integers, the reasoning behind this will be discussed in section 4.4. It is worth noting that device, metric, template, and thresholds each are identified by an unique id which makes the accessing logic consistent. For example all templates existing for a device X can be found under a URL logic associated with the id of a device, which simplifies upgrading and changing contracts for a specific device.

4.3 Generated Smart Contracts

The main purpose of our application is to generated SCs for IoT use cases. Since SCs cost money to execute on the blockchain, finding an appropriate highly optimized SC for the specific use case is desirable. Therefore the application focuses on devices which measure n $\in \mathbb{N}$ metrics and have some trigger values associated with it. This work defines metrics as every physically measurable unit a senor might capture.

4.3.1 Template with Trigger evaluation

A simple use case might be a sensor which measures the relative Humidity and the Temperature in Celsius and should trigger an alarm if the current values measured by the device are outside of the range specified by the user. Note that the unit of measurement (e.g. permille or percent) are to be specified by the user see section 4.4 for the reasoning behind this.

Table 4.1: Room Temperature according to National Asthma Council Australia [5]

	Lower Trigger	Upper Trigger
Temperature, Celsius in percent	1800	2500
Relative humidity in permille	350	550

An example Contract generated for this use case might look like the code shown in figure 4.1.

```
pragma solidity ^0.4.11;
contract owned {
  //This is an official solidity snippet from
  //http://solidity.readthedocs.io/en/develop/contracts.html
    function owned() { owner = msg.sender; }
    address owner;
    // This contract only defines a modifier but does not use
    // it - it will be used in derived contracts.
    // The function body is inserted where the special symbol
    // This means that if the owner calls this function, the
    // function is executed and otherwise, an exception is
    // thrown.
    modifier onlyOwner {
        require(msg.sender == owner);
        _;
    }
}
contract roomtemperatureisok is owned {
    int16 constant TemperatureCelsiusPerCentLT = 1800;

    int16 constant TemperatureCelsiusPerCentUT = 2500;

    int16 constant RelativeHumidityInPerMilleLT = 350;

    int16 constant RelativeHumidityInPerMilleUT = 550;

    event AlarmTemperatureCelsiusPerCent(
    int16 _valTemperatureCelsiusPerCent,
    string _id
    );

    event AlarmRelativeHumidityInPerMille(
    int16 _valRelativeHumidityInPerMille,
    string _id
    );

    event AlarmAll(
    int16 _valTemperatureCelsiusPerCent,
    int16 _valRelativeHumidityInPerMille,
    string _id
    );

    function getTriggers() onlyOwner
    returns (
    int16 TemperatureCelsiusPerCentLT,
    int16 TemperatureCelsiusPerCentUT,
    int16 RelativeHumidityInPerMilleLT,
    int16 RelativeHumidityInPerMilleUT
    ){
        return (
        TemperatureCelsiusPerCentLT,
        TemperatureCelsiusPerCentUT,
        RelativeHumidityInPerMilleLT,
        RelativeHumidityInPerMilleUT
        );
    }

    function updateTemperatureCelsiusPerCent(
    int16 _valTemperatureCelsiusPerCent,
    string _id
    ) onlyOwner {
        if (TemperatureCelsiusPerCentLT > _valTemperatureCelsiusPerCent ||
        TemperatureCelsiusPerCentUT < _valTemperatureCelsiusPerCent
        ) {
            AlarmTemperatureCelsiusPerCent(_valTemperatureCelsiusPerCent, _id);
        }
    }

    function updateRelativeHumidityInPerMille(
    int16 _valRelativeHumidityInPerMille,
    string _id
    ) onlyOwner {
        if (RelativeHumidityInPerMilleLT > _valRelativeHumidityInPerMille ||
        RelativeHumidityInPerMilleUT < _valRelativeHumidityInPerMille
        ) {
            AlarmRelativeHumidityInPerMille(_valRelativeHumidityInPerMille, _id);
        }
    }
    function updateAll(
    int16 _valTemperatureCelsiusPerCent,
    int16 _valRelativeHumidityInPerMille,
    string _id
    ) onlyOwner
    {
        if ((TemperatureCelsiusPerCentLT > _valTemperatureCelsiusPerCent ||
        TemperatureCelsiusPerCentUT < _valTemperatureCelsiusPerCent)
        &&
        (RelativeHumidityInPerMilleLT > _valRelativeHumidityInPerMille ||
        RelativeHumidityInPerMilleUT < _valRelativeHumidityInPerMille)
        ) {
            AlarmAll(
            _valTemperatureCelsiusPerCent,
            _valRelativeHumidityInPerMille,
            _id);
        }
        else {
            if (TemperatureCelsiusPerCentLT > _valTemperatureCelsiusPerCent ||
            TemperatureCelsiusPerCentUT < _valTemperatureCelsiusPerCent
            ) {
                AlarmTemperatureCelsiusPerCent(
                _valTemperatureCelsiusPerCent,
                _id);
            }
            if (RelativeHumidityInPerMilleLT > _valRelativeHumidityInPerMille ||
            RelativeHumidityInPerMilleUT < _valRelativeHumidityInPerMille
            ) {
                AlarmRelativeHumidityInPerMille(
                _valRelativeHumidityInPerMille,
                _id);
            }
        }
    }
}
```

Listing 4.1: Example of an Autogenerated SC for a sensor capturing two metrics

The SC shown above can be deployed for around 550000 gas a more detailed insight is given in section 5.
The contract owned as shown in 4.1 ensures that all functions annotated with the modifier onlyOwner are only callable by the contract deploying address. This is a security measure such that only sensors controlled by the owner are able to trigger Events.
If the allowed addresses are known prior to the deploying process, the contract can be modified easily to allow a specific set of addresses, as shown in 4.2. Using this mechanism a collection of similar sensors could communicated through the same contract, which potentially eases monitoring.

```
pragma solidity ^0.4.11;
contract owned {
  //This is an official solidity snippet from
  //http://solidity.readthedocs.io/en/develop/contracts.html
    function owned() { owner = msg.sender; }
    address owner;
    address secondDevice = 0xdeadbeefEB80aFb9972f510dA72c39de0B505752;
    address thirdDevice = 0xbeefdeadEB80aFb9972f510dA72c39de0B505752;
    modifier onlyOwner {
        require(
            msg.sender == owner ||
            msg.sender == secondDevice ||
            msg.sender == thirdDevice
            );
        _;
    }
}
```

Listing 4.2: Example of a extended owned contract

The user defines the variables shown in table 4.2.

Table 4.2: Values generated and specified by the user

Values given by the user	Generates values in SC 4.1
Template name: roomtemperatureisok	contract roomtemperatureisok is owned
Physical Property: Temperature Unit of Measurement: Celsius	int16 TemperatureCelsiusPercentLT int16 TemperatureCelsiusPercentUT event: AlarmTemperatureCelsiusPercent function: updateTemperatureCelsiusPercent
Physical Property: Relative Humidity Unit of Measurement: PerMille	int16 RelativeHumidityInPermilleLT int16 RelativeHumidityInPermilleUT event: AlarmRelativeHumidityInPerMille function: updateRelativeHumidityInPerMille
Lower trigger Temperature: 1800	int16 TemperatureCelsiusPerCentLT = 1800
Upper trigger Temperature: 2500	int16 TemperatureCelsiusPerCentUT = 2500;
Lower trigger Humidity: 350	int16 RelativeHumidityInPermilleLT = 350;
Lower trigger Humidity: 550	int16 RelativeHumidityInPermilleUT = 550;

The functions updateAll(), getTriggers() and the event AlarmAll() are generated and influenced by all metrics set by the user.
All functions starting with the name update, in this template, generate an Event including the sensor identifier and the current value, if the value violates the inequality constraint lowerTrigger < currentValue < upperTrigger. The function updateAll() triggers an Event AlarmAll, containing all current values and the device identifier, if all constraints are violated simultaneously. If only a subset of all metrics are violated only the respective Events are triggered.
For example in our ”Room temperature is ok” setting, if the Temperature in Celsius rises above 60 degrees the user could assume there was a fire in the room, but if the Humidity barely changes, it is likely that it is only a sensor malfunction.

4.3.2 Lightweight Template

Since the amount of different use cases and possibilities in the SC area are potentially endless and the SC Generator Framework might be expended in the future. Additionally to the template described in section 4.3.1, the application offers a lightweight contract, which can easily be updated and modified.

```
pragma solidity ^0.4.11;


contract owned {
    //This is an official solidity snippet from
    //http://solidity.readthedocs.io/en/develop/contracts.html
    function owned() {owner = msg.sender;}

    address owner;
    // This contract only defines a modifier but does not use
    // it - it will be used in derived contracts.
    // The function body is inserted where the special symbol
    // This means that if the owner calls this function, the
    // function is executed and otherwise, an exception is
    // thrown.
    modifier onlyOwner {
        require(msg.sender == owner);
        _;
    }
}


contract roomtemperatureisok is owned {
    event TemperatureCelsiusPerCent(
    int16 _valTemperatureCelsiusPerCent,
    string _id);

    event RelativeHumidityInPerMille(
    int16 _valRelativeHumidityInPerMille,
    string _id);

    event AlarmAll(
    int16 _valTemperatureCelsiusPerCent,
    int16 _valRelativeHumidityInPerMille,
    string _id
    );

    function alarmTemperatureCelsiusPerCent(
    int16 _valTemperatureCelsiusPerCent,
    string _id
    ) onlyOwner {
        TemperatureCelsiusPerCent(_valTemperatureCelsiusPerCent, _id);
    }

    function alarmRelativeHumidityInPerMille(
    int16 _valRelativeHumidityInPerMille,
    string _id) onlyOwner {
        RelativeHumidityInPerMille(_valRelativeHumidityInPerMille, _id);
    }

    function alarmAll(
    int16 _valTemperatureCelsiusPerCent,
    int16 _valRelativeHumidityInPerMille,
    string _id
    ) onlyOwner
    {
        AlarmAll(_valTemperatureCelsiusPerCent, _valRelativeHumidityInPerMille, _id);
    }
}
```

Listing 4.3: Example of a lightweight template

The SC shown in 4.3 is the lightweight version of a SC for Sensors. In this form it basically stores the values sent to it by the user, in the form of an Event in the blockchain. Due to its limited capabilities it is quite cheap and can be deployed from 370000 gas, this is approximately 30% cheaper than the SC shown in 4.1. The values specified by the user are shown in table 4.3. Such a contract can be used if multiple sensors are reporting their metrics every x hours, and the recorded values are then publicly available in the blockchain, for blockchain applications to process or humans to monitor the state of the sensor network. Of course it can also be useful for debugging purposes since it is lightweight and reports all values sent to it, such that when developing a new SC this SC

Table 4.3: Values specified by the user for the lightweight contract

Values given by user	Generates values in 4.3
Template name: roomtemperatureisok	contract roomtemperatureisok is owned
Physical Property: Temperature Unit of Measurement: Celsius	event: AlarmTemperatureCelsiusPercent function: updateTemperatureCelsiusPercent
Physical Property: Relative Humidity Unit of Measurement: PerMille	event: AlarmRelativeHumidityInPerMille function: updateRelativeHumidityInPerMille

can be used simultaneously .
To stay with the example already used previously, if the user has a building with 100 rooms, and every sensor saves the values accumulated throughout the day (let's say a daily average) and pushes them to the SC. A blockchain application, which listens to the Events generated by the SC could do the costly operations, such as evaluating trigger violations, calculating averages or finding suboptimal room conditions. This would also further decrease the costs of the SC evaluation.
Also, this lightweight contract can be easily extended to meet the specific requirements useful for the sensors environment required by the user. Additionally modification which lower the cost of ownership can be found in the section 5.2.

4.4 Restrictions of the pragma solidity 0.4.11 version

Since Solidity has only been around for 2 years certain restrictions, which can be expected to change in the future, exist. As of july 2017 the following restriction mainly influenced the development of this framework.

4.4.1 Usage of float/double

While using floats and double would be very useful for capturing metrics in their correct representations, as of july 2017 this is not possible. The Solidity documentation mentions that this feature should be coming soon, but do not mention an explicit date this should be completed. It would be possible to work with bytes instead, but since this would make the application harder to use it has not been used.
The best practice approach to work with this restriction is to multiply every value with a suitable number, such that the information is preserved but fits into the range of an integer. Since there is no ideal solution it has been decided that the user should handle the input format, such that all values are converted to an integer which is accurately enough for the users use case. To illustrate this some examples are shown in table 4.4.

Table 4.4: Convert values to the integer range

Real value:	Multiplier	integer representation:
-18.67 degrees in Celsius	100	-1867
0.005 Endrin $\mu g/l$	1000	5
1.0009%	10000	10009

4.4.2 Estimating the gas costs

To indicate to the users of our SCs how much their individual contracts will cost, is a desirable feature. Therefor the application estimates the gas costs of the contract via the function web3.estimateGas [10]. This functionality and its limitations are further discussed in section 4.4.
As of july 2017 the most reliable way of determining the costs of a SC is deploying it on one of the various Etherum-testnets and see the actual gas usage. This procedure is also used to determine the costs for a function calls.

4.4.3 ASCII

ASCII(American Standard Code for Information Interchange) is an encoding with 128 (this might vary slightly) different Symbols [19]. Due to Solidity only being able to work with ASCII symbols the entire application is limited to be ASCII only. This means Symbols such as ü, ä, ö etc., which might be quite common in German or other languages, can not be used.

Chapter 5

Evaluation

5.1 Smart Contract Cost Evaluation

This section testes the SCs shown in section 4.3 for their cost usage. Both, the SCs without trigger evaluation and the ones with trigger evaluation have been tested. For testing the ropsten testnet [24] has been used. Since all functions allow a parameter _id of type String, this parameter can be left empty to reduce costs to a minimum, the test String "Hello World" has been used, since it is quite common in the IT community.
The Price for 1'000'000 Gas was between 0.007 - 0.008 Ether while testing. It is worth noting that the costs for a function call in Gas can change slightly depending on the Solidity version, version pragma 0.4.11 has been used for all tests. The time of execution can also influence the Gas costs slightly.
Empirically it has been found that the function call costs increase the bigger the byte code of the SC is. This means more code in the SC increases its execution costs, therefore deleting all unnecessary functions of the SC is desirable.

5.1.1 Lightweight SC cost evaluation

The following numbers and explanation refer to the SC template shown in section 4.3.2. The costs for this template increase for each additional metric captured. The function call costs for this template are relatively stable and do not vary depending on additional logic, except for the usage of the _id string used which varies depending on the users needs. The result of our tests are illustrated in table 5.1.

Table 5.1: Lightweight template costs

Nr. of Metrics	Deploy costs in Gas	Single Alarm costs in Gas	Alarm all costs in Gas
1	274 622	27 394	27 416
2	369 049	27 394	29 915
3	462 597	27 416	32 446
4	555 685	27 416	34 930
5	630 001	27 525	37 463

It has been found that deployment costs can be approximated using

$$89740 * x + 189173 \tag{5.1}$$

where x is the amount of metrics used. The cost for a single metric update is almost constant, but increases slightly depending on the contracts size. The costs for updating all values simultaneously can be approximated using

$$2510 * x + 24901 \tag{5.2}$$

where x is the amount of metrics used.

5.1.2 Threshold Smart Contract cost evaluation

The following numbers and explanation refer to the SC template shown in section 4.3.1. The costs for this template increase for each additional metric captured. The functions cost of this template vary depending on whether an Event is fired or not, as well as if the trigger violation is an upper or lower trigger violation (upper trigger violations are cheaper, due to design choices made).
All tests have been made using a lower trigger of 0 and an upper trigger of 100, the value to violate the triggers has been chosen to be -1 and the value to not violate the trigger has been chosen to be 0 (the choice of the integer send to the function should not influence the costs). The function getTriggers() is constant and therefore free to call. The result of our tests are illustrated in table 5.2.

Table 5.2: Threshold template costs

Nr. of Metrics	Deploy costs in Gas	Single Alarm without Event costs in Gas	Single Alarm with Event costs in Gas	Alarm all with Event costs in Gas
1	378 575	23 272	27 470	27 503
2	549 987	23 272	27 470	29 998
3	722 064	23 294	27 492	32 652
4	874 578	23 403	27 601	35 282
5	1 050 324	23 447	27 645	37 743

It has been found the deployment costs can be approximated using

$$166809 * x + 214679 \tag{5.3}$$

where x is the amount of metrics used. The cost for a single metric update with and without Event is almost constant, but increases slightly depending on the contracts size. The costs for updating all values simultaneously and violating all constraints simultaneously can be approximated using

$$2576 * x + 24906 \tag{5.4}$$

where x is the amount of metrics used.

5.2 Optimization

Due to the overwhelming amount of different use cases no perfectly optimized generic contract can be generated. But optimizing the generated SCs further can significantly improve the cost efficiency of a SC.

5.2.1 Choosing the right integer

By default the generated integers are of type int16, which can be optimized or extended. Using smaller integers can reduce the SC execution costs, therefore choosing the smallest possible integer range is desirable.
If the application only uses positive numbers one could use unsigned integers to improve costs and range of the SC. Generally speaking, using unsigned integers

$$0 \quad to \quad 2^k - 1 \tag{5.5}$$

different values can be represented, where k is the amount of bits used. In Solidity unsigned integers in the range of uint8 to uint256 are allowed, also allowing all uints in between in steps of 8. This leads to the values shown in table 5.3 for unsigned integers. For signed integers as well the range int8 to int256 (again in steps of 8) is allowed. The

Table 5.3: Example values for unsigned integers

Unsigned integer	Range
uint8	0 to 255
uint16	0 to 65535
uint24	0 to 16777215
uint128	0 to 2^{128}-1
uint256	0 to 2^{256}-1

range this signed integers are able to represent can be calculated using

$$-2^{k-1} \quad to \quad 2^{k-1} - 1 \tag{5.6}$$

where k is the amount of bits used. This leads to the example values shown in table 5.4 for signed integers.

Table 5.4: Example values for signed integers

Signed integer	Range
int8	-128 to 127
int16	-32768 to 32767
int24	-16777216 to 16777215
int128	2^{127} to 2^{127}-1
int256	2^{255} to 2^{255}-1

Chapter 6

Summary and Conclusions

The implementation and design of the Smart Contract Generator Framework has been an educational and challenging task.
The projects most demanding aspect was the usage of the relatively new Solidity programming language and the combination of various different aspects of Computer Science, such as databases, web design and code generation.
Creating and designing a system from scratch showed the author the importance of use cases and various early design choices. Since there was no code base to build upon the author had the possibility to design and implement important aspects of the application himself.
All aspects of the Ethereum blockchain technology that had to be understood to implement this solution showed the author many possibilities and limitations associated with this new technology. While the fundamental usage and concepts of the blockchain was known prior, the author had no practical experience in developing such a system.
While the languages python, javascript, html and css where used by the author before he surely gained a lot of new experience and knowledge, most notably with the Django framework.
During the 3 months of this project, developing and experimenting with these different technologies was always interesting and educational. The author thoroughly enjoyed developing and writing this Softwareproject and had a lot of fun doing so.

Bibliography

[1] Kristian Lauslahti, Juri Mattila, Timo Seppälä. Smart Contracts How will Blockchain Technology Affect Contractual Practices?, https://www.etla.fi/wp-content/uploads/ETLA-Raportit-Reports-68.pdf, Last visited 09.01.2017.

[2] Postcapes, Blockchains and the Internet of Things, https://www.postscapes.com/blockchains-and-the-internet-of-things/, Last visited 29.06.2017

[3] K. Christidis and M. Devetsikiotis, Blockchains and Smart Contracts for the Internet of Things, http://ieeexplore.ieee.org/document/7467408/, Last visited 29.06.2017

[4] Comparison of code generation tools, https://en.wikipedia.org/wiki/Comparison_of_code_generation_tools, Last visited 29.06.2017

[5] Indoor Humidity Levels, National Asthma Council Australia, https://www.nationalasthma.org.au/news/2016/indoor-humidity, Last visited 30.06.2017

[6] M. Swan, Blockchain: Blueprint for a New Economy, http://w2.blockchain-tec.net/blockchain/blockchain-by-melanie-swan.pdf, Last visited 10.07.2017

[7] S. Nakamoto, Bitcoin: A Peer-to-Peer Electronic Cash System, https://bitcoin.org/bitcoin.pdf, Last visited 10.07.2017

[8] Ethereum Foundation, "Official Website", https://www.ethereum.org/ , Last visited 10.07.2017

[9] CoinMarketCap, CryptoCurrency Market Capitalizations Ethereum, https://coinmarketcap.com/currencies/ethereum/, Last visited 10.07.2017

[10] Ethereum Foundation, Solidity, https://solidity.readthedocs.io/en/develop/, Last visited 10.07.2017

[11] Ethereum Foundation, Ethereum Development Tutorial, https://github.com/ethereum/wiki/wiki/Ethereum-Development-Tutorial, Last visited 10.07.2017

[12] Stewart Bond (IBM), It Was Only a Matter of Time Digital Identity on Blockchain, https://www-01.ibm.com/common/ssi/cgi-bin/ssialias?htmlfid=GIL12346USEN&, 24.03.2017

[13] City Zug (CH), Blockchain-Identität für alle Einwohner http://www.stadtzug.ch/de/ueberzug/ueberzugrubrik/aktuelles/aktuellesinformationen/?action=showinfo&info_id=383355, Last visited 03.08.2017

[14] jrbedard, What is wei? http://forum.ethereum.org/discussion/304/what-is-wei, 01.10.2014

[15] Ethereum Community, Account Types, Gas, and Transactions http://ethdocs.org/en/latest/contracts-and-transactions/account-types-gas-and-transactions.html#what-is-gas, Last visited 17.07.2017

[16] Ethereum Foundation, 1.0 gas costs, https://docs.google.com/spreadsheets/d/1m89CVujrQe5LAFJ8-YAUCcNK950dUzMQPMJBxRtGCqs/edit#gid=0, Last visited 17.07.2017

[17] Ethereum Foundation, Ethereum-Wallet, https://www.ethereum.org/, Last visited 25.07.2017

[18] Ethereum Foundation, geth, https://geth.ethereum.org/, Last visited 25.07.2017

[19] ASCIItable.com, ASCII table, http://www.asciitable.com/, Last visited 27.07.2017

[20] Shivkumar Kalyanaraman, Industry 4.0 meets Cognitive IoT, https://www.ibm.com/blogs/internet-of-things/industry-4-0-meets-cognitive-iot/, 28.10.2016

[21] F. Shrouf, J. Ordieres, G. Miragliotta,Smart factories in Industry 4.0: A review of the concept and of energy management approached in production based on the Internet of Things paradigm, http://ieeexplore.ieee.org/document/7058728/#, 12.03.2015

[22] DataFloq, Privacy, https://datafloq.com/read/securing-internet-of-things-iot-with-blockchain/2228, Last visited April 20, 2017

[23] Django Software Foundation, Django The web framework for perfectionists with deadlines , https://www.djangoproject.com/, Last visited 27.07.2017

[24] Ethereum Foundation, Ropsten testnet PoW chain, https://github.com/ethereum/ropsten, Last visited 29.07.2017

[25] John R. Koza, a source of information about the field of genetic programming and the field of genetic and evolutionary computation, http://www.genetic-programming.org/, Last visited 05.08.2017

[26] CodeSmith Tools, CodeSmith Tools, http://www.codesmithtools.com/, Last visited 05.08.2017

Abbreviations

ASCII	American Standard Code for Information Interchange
ERM	Entityrelationship model
EVM	Ethereum Virtual Machine
IoT	Internet of Things
SC	Smart Contract
UI	User interface

List of Figures

List of Tables

Appendix A

Installation Guidelines

Additional documentation might be found at `https://www.djangoproject.com/` and `https://github.com/SandroLuck/SCGenerator`. Note that all the instructions starting with a bulletpoint are ment to be executed in the terminal.

A.1 Install and start the application

Install Python (Versions >=3 should work, yet we have developed this application using 3.5.2) and pip.
Install via pip django, psycopg2 and naked

- pip install django naked psycopg2

cd to /"Path to this app"/mysite/ and locate the manage.py file (you have to be in the folder where manage.py is).

- cd /"Path to this app"/mysite/

run the django migrations in cases something went wrong, it should return nothing to change.

- python manage.py makemigrations
- python manage.py migrate

To run the application

- python manage.py runserver

Open your browser and go to `localhost:8000/smartGenerator/`

A.2 Switching Database

Since setting up the application is easiest with SQLite it is the default database. However if you wish to use a different database choose one from `https://docs.djangoproject.com/en/1.11/ref/databases/` and follow the django documentation.

A.3 Create an admin user

To create a Django admin user.

- python manage.py createsuperuser

You can now go to `localhost:8000/admin/` and modify the database directly.

A.4 Add blockchain support

To get the full blockchain support (Mainly the Smart Contract Monitor):
Install nodejs `https://nodejs.org/en/download/` and npm (or any package manager)

- npm install web3 solc fs

Install geth `https://github.com/ethereum/go-ethereum/wiki/geth` and run the geth command.

- geth –rpc –testnet –rpcapi=”db,eth,net,web3,personal” –rpcport ”8545” –rpcaddr ”127.0.0.1” –rpccorsdomain ”localhost”

Note that this will start the testnet ropsten. The application will listen to every eth-blockchain running on the port 8545, so you can switch it with the mainnet or any other Ethereum net.

Appendix B

Contents of the CD

The CD contains the the following:

- The source code of the application.
- The latex files to generate this report.
- The German summary.
- The .pdf and .ps file of this document.