FACULTEIT INGENIEURSWETENSCHAPPEN
EN ARCHITECTUUR

Techno-economische kostmodellering voor IoT-technologieën via implementatie van een online selectie- en modelleringstool

Glenny De Cock
Studentennummer: 01508683

Promotoren: prof. dr. ir. Sofie Verbrugge, prof. dr. ir. Didier Colle
Begeleiders: dr. ir. Frederic Vannieuwenborg, Thibault Degrande, ir. Pieter De Mil (allòra Factory)

Masterproef ingediend tot het behalen van de academische graad van
Master of Science in de industriële wetenschappen: informatica

Academiejaar 2018-2019

UNIVERSITEIT
GENT

FACULTEIT INGENIEURSWETENSCHAPPEN
EN ARCHITECTUUR

Techno-economische kostmodellering voor IoT-technologieën via implementatie van een online selectie- en modelleringstool

Glenny De Cock
Studentennummer: 01508683

Promotoren: prof. dr. ir. Sofie Verbrugge, prof. dr. ir. Didier Colle
Begeleiders: dr. ir. Frederic Vannieuwenborg, Thibault Degrande, ir. Pieter De Mil (allòra Factory)

Masterproef ingediend tot het behalen van de academische graad van
Master of Science in de industriële wetenschappen: informatica

Academiejaar 2018-2019

UNIVERSITEIT
GENT

Woord vooraf

Na een intensieve periode van vier maanden leg ik met het schrijven van dit dankwoord de laatste hand aan mijn masterproef. Ik blik met gemengde gevoelens terug op een intense periode waarin ik niet alleen op wetenschappelijk, maar ook op persoonlijk vlak veel heb bijgeleerd. Tijdens dit sluitstuk van mijn schoolcarrière werd eens te meer duidelijk dat ik nog niet weet welke richting ik verder wil inslaan. Ik hoop dan ook dat de nabije toekomst veel verduidelijking schept.

Dit werk was niet mogelijk geweest zonder de steun en ondersteuning van verschillende personen die ik dan ook graag via deze weg wil bedanken.

Allereerst wil ik mijn begeleiders dr. ir. Frederic Vannieuwenborg, Thibault Degrande en ir. Pieter De Mil bedanken voor de vele tips, feedback, relevante gegevens en expertise. In het bijzonder een welgemeende dank u wel aan de heren Thibault Degrande en dr. ir. Frederic Vannieuwenborg voor de tussentijdse reviews, suggesties en opbouwende kritiek. Terugblikkend op de voorbije periode, mag ik mijzelf gelukkig prijzen met jullie als begeleiders.

Tevens gaat mijn dank ook uit naar mijn promotoren prof. dr. ir. Sofie Verbrugge en prof. dr. ir. Didier Colle voor de kans om deze masterproef aan te vangen en tot een goed resultaat te brengen.

Tot slot wil ik mijn ouders en vriendin bedanken om mij, niet alleen tijdens mijn masterproef, maar gedurende mijn volledige schoolcarrière bij te staan. Bedankt aan mijn ouders die al het mogelijke hebben gedaan opdat ik een mooie toekomst tegemoet zou gaan.

Glenny De Cock
Juni, 2019

Techno-economic cost modeling for IoT technologies via implementation of an online selection and modeling tool

Glenny De Cock

Supervisors: prof. dr. ir. Sofie Verbrugge, prof. dr. ir. Didier Colle, dr. ir. Frederic Vannieuwenborg, Thibault Degrande, ir. Pieter De Mil

Abstract—**This article proposes a techno-economic model that allows the user to map the costs of an IoT use case through its life cycle. First, the different components of a generic IoT system are determined. Next, the various cost drivers for each of the components of an IoT system are identified. Based on the established cost drivers, a techno-economic model is developed that allows the user to map the costs of an IoT use case. The proposed techno-economic model is used as the basis for the design and implementation of an online web tool, and the methodology is used to model the costs of a cold chain monitoring solution. Validation through the practical use case showed that the web tool allows for a quick overview of the economic impact on different design decisions such as IoT connectivity and IoT platform. The web tool guides the user to the most cost efficient solution that also meets all the functional requirements of the use case.**

Keywords—**Internet of Things, life cycle cost, selection and modeling tool**

I. Introduction

THE digital world as we know it today is rapidly evolving. Predictions indicate that there will be 21.2 billion devices connected to the Internet in 2020 and no less than 34.2 billion by 2025 [1]. This remarkable evolution will eventually lead to almost everything being connected with the Internet, also known as the Internet of Things (IoT). Just a few of the many industries in which IoT will have a significant impact are: transport and logistics, industrial automation, home automation, healthcare...

It is therefore not surprising that different companies are curious as to whether an IoT system can offer a positive outcome for their use case. IoT use cases are, however, often technically feasible though are not economically viable. Each IoT system has different requirements and priorities, as well as a different budget and scale. When developing IoT systems the developer must choose from, among other things, the various possible IoT connectivity technologies and IoT platforms. In addition to different properties and functional characteristics, the various options also have various costs associated with them. The final choices heavily impact the economic feasibility of the use case.

The goal of this master's dissertation is to design and implement an online selection and modeling tool that allows to map the life cycle costs of an IoT use case in a quick and clear manner. After entering the required use case parameters, the tool will provide an overview of the possible design decisions and enable the user to simulate the economic impact of design decisions. The tool guides the user to the most economical solution that also meets all the functional requirements of the use case.

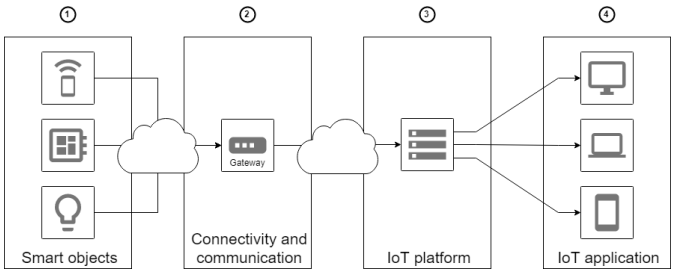


Fig. 1: The four main components of a generic IoT system.

II. Literature on IoT systems

Literature on IoT systems, such as [2], [3], [4] and [5], distinguishes the four main components of IoT systems as shown in Fig. 1. Smart objects are the building blocks of IoT and form the bottom layer of an IoT system. It are the physical devices that will detect and capture information from their environment on the basis of sensors and may also influence their environment on the basis of actuators. Next, the connectivity and communication component is essential to every IoT system. The connectivity and communication component realizes the exchange of data between the smart objects and the IoT platform over a network. It is the link between the physical world, the smart objects, and the virtual world, the Internet. The third component, the IoT platform, is the central backbone of each IoT system and can be distinguished, among other things, in terms of segment focus, technological depth and technological implementation [2]. The final component, the IoT application, presents IoT use cases to end users. IoT applications vary based on, among other things, the nature of the sensor data and business needs. The design of user interfaces is use case dependent, therefore they are usually custom developed or, in the case of IoT platform interfaces, uniquely configured. The best approach depends on the required user experience.

III. Methodology

After having gathered insights into the different components of an IoT system, a Cost Breakdown Structure (CBS) was set up. By definition, the summation of the various cost components at the lowest level of the CBS should provide a good approximation for the total cost of an IoT system. A differentiation was made between the Capital Expenditure (CAPEX), which are the upfront costs for an IoT system, and the Operational Expendi-

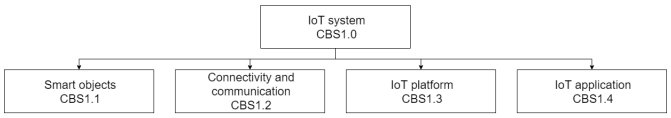


Fig. 2: Cost components of an IoT system.

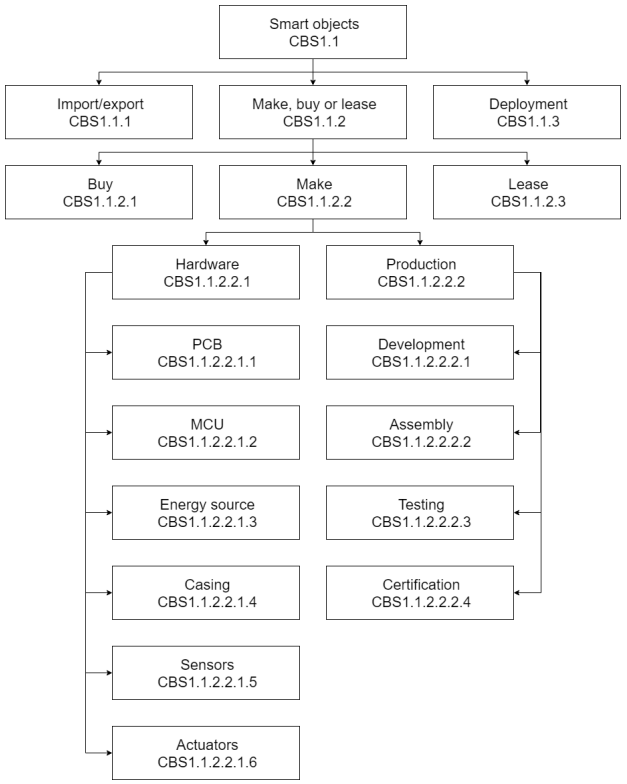


Fig. 3: Smart objects cost component breakdown.

ture (OPEX), which are the recurring costs over a certain period of time. The sum of the CAPEX and OPEX over the lifespan of an IoT system makes up the total cost. It is important to note that not every IoT system always contains all the cost components or that a choice has to be made between the various cost components. Fig. 2 shows, by analogy with the various components of an IoT system, the highest level of the CBS. The following subsections each deal in detail with a cost component and the main cost drivers are listed and clarified.

A. Smart objects

The costs associated with the cost component smart objects (CBS1.1) can be further subdivided into three cost components, namely the possible import or export cost (CBS1.1.1), the decision whether the smart objects are made in-house, purchased or rented (CBS1.1.2) and deployment costs (CBS1.1.3). The breakdown of the smart objects component is show in Fig. 3.

The first cost component, the possible import or export cost, applies if hardware is purchased or sold outside of Europe. If this is the case, an extra cost may have to be allocated on top of the hardware cost, this contributes to the CAPEX.

If the smart objects are purchased, the CAPEX consists of the product of the number of smart objects with the individual

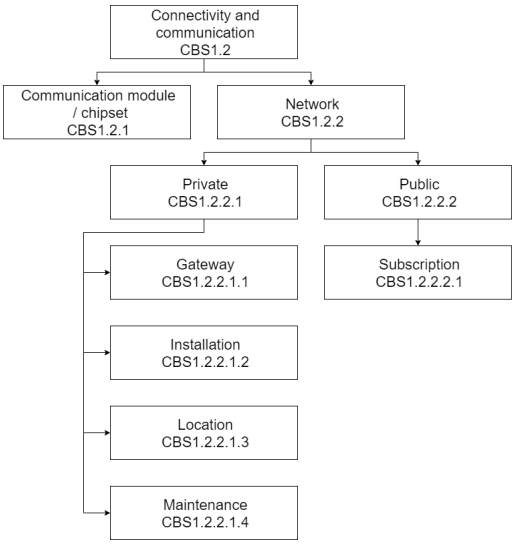


Fig. 4: Connectivity and communication cost component breakdown.

cost of a smart object. The same applies if the smart objects are rented, with the difference that a recurring rent must be paid for the smart objects. As a result, the smart objects cost is now part of the OPEX. If the smart objects are made in-house, in addition to purchasing the required individual hardware components, the production of the smart objects must take place. The production of hardware starts with a development phase in which the smart object is designed and tested. After the development, the individual hardware components are assembled, which entails a cost per smart object. After the assembly of the smart objects, these are tested. The designed smart objects must in some cases be certified, for example to realize connectivity with public network providers. Certification is a one-off cost, regardless of the number of smart objects. The production cost component of creating in-house smart objects contributes to the CAPEX.

The third cost component, the deployment cost, applies to both smart objects that were created in-house, purchased or rented. It represents the cost involved in rolling out, installing, integrating and authenticating the smart objects and contributes to the CAPEX. Notice that rented smart objects may have different or all parts of the deployment cost component included in the recurring rent.

If the smart objects have a finite energy source and the battery lifespan expires before the intended lifespan of the IoT system, the battery must be replaced. This cost contributes to the OPEX.

B. Connectivity and communication

The costs associated with the cost component connectivity and communication (CBS1.2) consist of two cost components, namely the cost of the communication module or chip set (CBS1.2.1) present on the smart object and the network cost (CBS1.2.2) as shown in Fig. 4.

The communication module belongs to the hardware of the smart object but is explicitly put under the connectivity and communication cost component, because different IoT connectivity technologies carry different costs for the communication mod-

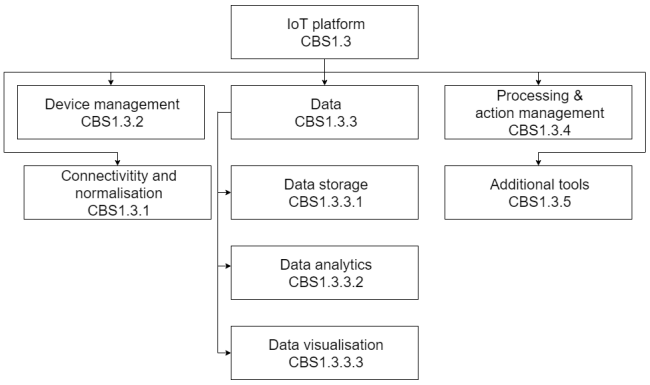


Fig. 5: IoT platform cost component breakdown

ule. The communication module cost is present for every smart object and is part of the CAPEX.

The second cost component, the network cost, depends on whether the IoT developer opts for a private or public network. In a private network, the CAPEX consist of the gateways, the installation costs of the gateways and setting up the network. If gateways are installed at public or other locations than the owner's own property, often a rental cost for the location has to be paid. The rental cost, together with the maintenance cost of the network, forms the OPEX of the connectivity and communication cost component. If a private network is not possible or if a public network is opted for, the OPEX of the connectivity and communication cost component consists of the subscription cost per smart object multiplied by the number of smart objects. Notice that some IoT connectivity technology subscriptions require a one-off cost, such as buying a Subscriber Identity Module (SIM) card for each smart object.

C. IoT platform

The costs associated with the cost component IoT platform (CBS1.3) consists mainly of five cost components: connectivity and normalisation (CBS1.3.1), device management (CBS1.3.2), data (CBS1.3.3), processing and action management (CBS1.3.4) and additional tools (CBS1.3.5). The breakdown of the IoT platform cost component is shown in Fig. 5. The possibility of designing, developing and hosting an IoT platform itself is not taken into consideration.

The connectivity and normalisation component takes care of communication with the smart objects and normalises the various communication protocols and data formats into a uniform interface. The cost is usually calculated on the data volume exchanged or the number of messages exchanged between the smart objects and the IoT platform.

Most IoT platforms do not charge any startup or termination costs, so you only pay for the individual components you need, for as long as you use them. The costs of an IoT platform therefore only contribute to the OPEX of an IoT system.

D. IoT application

IoT applications are the largest variable cost of an IoT system. The CAPEX consist of designing and developing the IoT application, the OPEX of maintaining the IoT application. The OPEX largely depend on the scale on which the IoT application is used. The requirements of an IoT application are very use case specific and therefore difficult to identify generically. The cost determination of the design, development and maintenance of an IoT application therefore falls outside the scope of this master's dissertation.

IV. Results and discussion

The developed web tool, which is built upon the proposed techno-economic model, is used to simulate a practical use case, namely monitoring the temperature in the cold chain, to consider its economic feasibility.

A. Cold chain monitoring

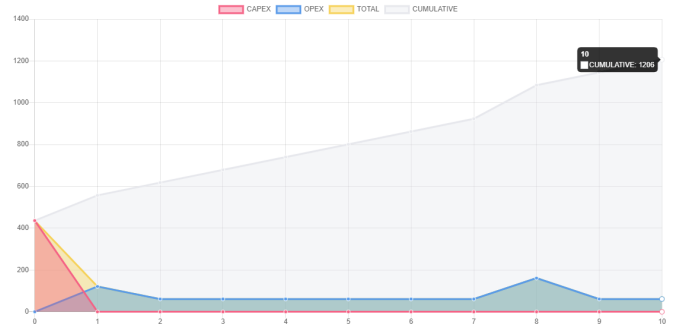
The cold chain is the continuous system of cooling in the supply chain of, among other things, fresh or frozen food and pharmaceutical products from the producer, via carriers, wholesalers and stores to the consumer. This case study deals in detail with monitoring the temperature of the cold chain of fresh or frozen food products and more specifically the link that is responsible for storing fresh or frozen food products in refrigerators. The monitoring of the temperature in refrigerators is often still done manually and not automatically. As a result, if a problem occurs, it is often not possible to intervene in time. A solution whereby the temperature is monitored automatically and wireless is certainly usable. The use of an automatic solution also has the potential to be economically profitable through the saving of labor costs, as well as by early reporting that the temperature rises above a certain threshold, thereby preventing the destruction of products. An additional element is that through the automatic monitoring of the temperature in refrigerators, the efficiency of different refrigerators can be investigated, which can lead to optimization of the cold chain.

The use case has the following main features and requirements:

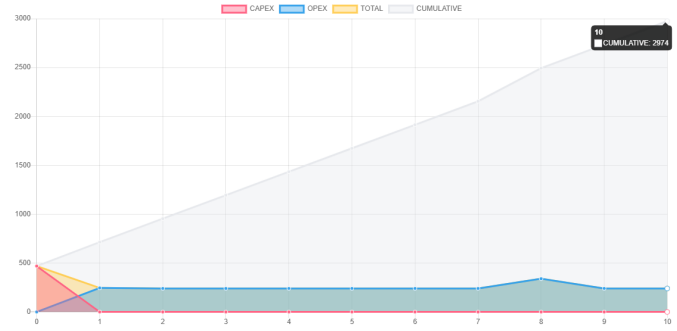
– The number of smart objects is limited to a few dozen, namely one per refrigerator;
– The connectivity is always limited to local connectivity;
– The monitoring of the temperature in refrigerators requires no real-time reporting. Frequent updates such as two or four times per hour are sufficient;
– The desired lifespan of the IoT system and in particular the battery life of the smart objects is preferably as long as possible.

Despite the fact that the use case is mainly aimed at local IoT systems where the high range of LPWAN connectivity technologies is not considered added value, it is nevertheless opted to use LPWAN connectivity technologies. This is partly because the different LPWAN connectivity technologies have a higher link budget than for example WiFi or Bluetooth, which certainly comes in handy when the connectivity has to break through the high insulation barriers of the refrigerators.

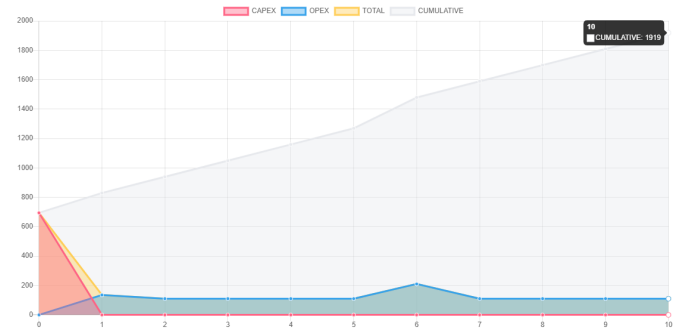
The impact of design decisions, such as LPWAN connectivity technology and IoT platform, is evaluated from an economic point of view on the basis of different simulations. The battery lifespan for Sigfox and LoRa is set to 7 years. NB-IoT is less energy efficient than LoRa and Sigfox, therefore the battery lifespan for NB-IoT is set to 5 years.



(a) Sigfox



(b) LoRa



(c) NB-IoT

Fig. 6: The impact of different LPWAN connectivity technologies on the life cycle costs of the use case.

Various simulations indicate that the use of the Sigfox connectivity technology is significantly cheaper than LoRa and NB IoT as shown in Fig. 6. The total life cycle costs for the Sigfox, LoRa and NB-IoT LPWAN connectivity technologies are respectively 1,206 EUR, 2,974 EUR and 1,919 EUR. The tool visualizes that Sigfox has the lowest CAPEX as well, since the communication modules are cheaper than LoRa and NB-IoT. Although the simulations show that the Sigfox connectivity technology is the most cost effective in this use case, the Sigfox connectivity technology has a number of important disadvantages compared to LoRa and NB-IoT. For example, only four downlink messages per day are possible, no possibility for over-the-air updates and the maximum payload of 12 bytes.

The output of the web tool shows that making use of the Google Cloud IoT Core IoT platform is free in this use case, since the exchanged data is small and thus provided by the free tier of the Google Cloud Suite. The Microsoft Azure IoT Hub IoT platform requires an annual connectivity cost of 253 EUR, which means a total extra cost of 2,530 EUR over a ten-year lifespan.

V. Conclusion

The intrinsic heterogeneous nature and the different stakeholders of IoT systems do not make the techno-economic comparison simple. Different components of an IoT system are completely use case dependent and are therefore difficult to identify generically. For this reason, among other things, the IoT application component of an IoT system has not been taken into consideration. An additional factor is the great uncertainty about certain parameters such as the battery lifespan and replacement and installation times of various components. The listed factors confirm that the various simulations performed must be viewed from a critical point of view.

The web tool allows for a quick overview of the economic impact on different design decisions, which guides the user to the most cost efficient solution that also meets all the functional requirements of the use case. In addition to the specific documented use case, the tool offers the ability to simulate a phased roll out of smart objects, cost erosion of the IoT connectivity technologies, the ability to simulate the production of your own smart objects and to set up a private Lora network.

VI. Future work

There are several ways in which the methodology can be extended, such as increasing the functionality or enhancing the proposed techno-economic model. Increased functionality such as the ability to simulate Personal Area Networks (PAN) and Local Area Networks (LAN) can generalize the methodology. Various use case parameters, such as battery lifespan and the IoT application development cost, are unknown. The possibility to predict battery lifespan would greatly enhance the methodology.

Acknowledgements

The author would like to thank dr. ir. Frederic Vannieuwenborg, Thibault Degrande and ir. Pieter De Mil for sharing their expertise and providing advice, suggestions and constructive criticism. The author would also like to thank prof. dr. ir. Sofie Verbrugge and prof. dr. ir. Didier Colle for the opportunity to start this master's dissertation and to achieve a good result.

References

[1] Knud Lasse Lueth, "State of the IoT 2018: Number of IoT devices now at 7B - Market accelerating," 2018.

[2] Knud Lasse Lueth and Jan Kotzorek, "IoT Platforms: The central backbone for the Internet of Things," Tech. Rep., IoT Analytics, 2015.

[3] Michael E Porter and James E Heppelmann, "How Smart, Connected Products Are Transforming Competition," *Harvard Business Review*, vol. 92, no. 11, pp. 64, 2014.

[4] Ibrar Yaqoob, Ejaz Ahmed, Ibrahim Abaker Targio Hashem, Abdelmuttlib Ibrahim Abdalla Ahmed, Abdullah Gani, Muhammad Imran, and Mohsen Guizani, "Internet of Things Architecture: Recent Advances, Taxonomy, Requirements, and Open Challenges," *IEEE Wireless Communications*, vol. 24, no. 3, pp. 10–16, jun 2017.

[5] Cisco, "The Internet of Things Reference Model," *Internet of Things World Forum*, pp. 1–12, 2014.

Inhoudsopgave

Inhoudsopgave **i**

Lijst van figuren **vi**

Lijst van tabellen **viii**

Lijst van codefragmenten **ix**

Lijst van afkortingen **x**

Inleiding **1**

1 Situering van de masterproef 1

2 Probleem- en doelstelling van de masterproef 1

2.1 Probleemstelling 1

2.2 Doelstelling 2

3 Methodiek 2

4 Indeling van de scriptie 3

I Literatuurstudie **4**

1 Internet Of Things **5**

1.1 Wat is IoT? 5
1.2 Geschiedenis van IoT 7
1.3 Instaatstellers van IoT 8
1.4 IoT-architectuur 8

2 De componenten van een IoT-systeem 10

2.1 Slimme objecten 10
2.2 Connectiviteit en communicatie 12
2.2.1 Netwerktoegangslaag connectiviteitstechnologieën 13
2.2.2 Applicatielaag communicatieprotocollen 16
2.3 IoT-platform 20
2.4 IoT-applicatie 21

II Techno-economisch model van een IoT-systeem 23

1 Cost Breakdown Structure 24

1.1 Slimme objecten 25
1.2 Connectiviteit en communicatie 26
1.3 IoT-platform 27
1.4 IoT-applicatie 28

2 Kostendrijvers van een IoT-systeem 29

2.1 Slimme objecten 29
2.1.1 Maken, aankopen of huren 30
2.1.2 Bill Of Material 31

2.1.3 Eindige energiebron . . . 32
2.2 Connectiviteit en communicatie . . . 33
2.2.1 Sigfox . . . 33
2.2.2 LoRa . . . 33
2.2.3 NB-IoT . . . 34
2.3 IoT-platform . . . 35
2.3.1 Google Cloud IoT Core . . . 36
2.3.2 Microsoft Azure IoT Hub . . . 36
2.3.3 Amazon Web Services IoT Core . . . 37

III Ontwerp en ontwikkeling van de webtool 39

1 Behoefteanalyse 40
1.1 Functionele vereisten . . . 40
1.2 Niet-functionele vereisten . . . 41

2 Ontwerp 42
2.1 Technologiekeuze . . . 42
2.1.1 MEAN stack . . . 43
2.1.2 Docker . . . 44
2.2 Systeemarchitectuur . . . 44
2.2.1 Front-end . . . 45
2.2.2 Back-end . . . 47

3 Realisatie 48

3.1 Realisatie van de belangrijkste kernfuncties 48
3.1.1 Angular component interactie 48
3.1.2 Kostenberekening 51
3.1.3 HTTP API calls 52
3.1.4 Aanmaken van gebruikers 53
3.1.5 Authenticatie en autorisatie 54
3.1.6 Opslaan en laden van simulaties 57

4 Reflectie **60**

4.1 Ontwerp 60
4.2 Realisatie 61
4.3 Testen 61
4.4 Aandachtspunten voor de toekomst 61
4.4.1 Map- en bestandsstructuur 61
4.4.2 JavaScript in Angular vs JavaScript in Node.js 62
4.4.3 Localhost of 127.0.0.1 63

IV Validatie van de webtool aan de hand van een use case **64**

1 Koudeketen monitoring **65**

1.1 Achtergrond 65
1.2 Eigenschappen en vereisten 66
1.3 Simulaties 67
1.4 Resultaten en discussie 69

Besluit **70**

Referentielijst **72**

Bijlage A: Installatiehandleiding **77**

1 Installatie met behulp van Docker 77

2 Installatie zonder Docker 78

Bijlage B: Gebruikershandleiding **81**

1 Registreren en inloggen 81

2 Simulaties uitvoeren 81

3 Simulaties opslaan en inladen 82

Bijlage C: Use case simulaties **83**

Lijst van figuren

1.1 De drie verschillende visies op IoT. 6

1.2 De vier verschillende componenten van een IoT-systeem. 9

2.1 Architectuur van een slim object. 11

2.2 DIY slimme objecten. 11

2.3 TCP/IP model. 12

2.4 IoT-connectiviteitstechnologieën in de netwerktoegangslaag. 13

2.5 Acht componenten van een end-to-end IoT-platform. 20

1.1 De kostcomponenten van een IoT-systeem (CBS1.0). 24

1.2 De kostcomponenten van slimme objecten (CBS1.1). 25

1.3 De kostcomponenten van connectiviteit en communicatie (CBS1.2). 27

1.4 De kostcomponenten van een IoT-platform (CBS1.3). 28

1.5 De kostcomponenten van een IoT-applicatie (CBS1.4). 28

2.1 Deployment diagram van de webapplicatie. 45

2.2 Front-end webapplicatie opsplitsing in Angular Componenten. 46

2.3 Front-end klassendiagram. 46

2.4 Back-end klassendiagram. 47

4.1 Map- en bestandsstructuur van de front-end. 62

Lijst van tabellen

2.1 Technische en functionele karakteristieken van SigFox, LoRa en NB-IoT. 17

2.2 Vergelijking tussen IoT-communicatieprotocollen CoAP en MQTT. 19

2.1 Bill Of Material slim object. 31

2.2 Sigfox Engie M2M abonnementen prijslijst. 33

2.3 LoRa Proximus abonnementen prijslijst. 34

2.4 NB-IoT Orange abonnementen prijslijst. 35

2.5 Google IoT Cloud Core prijslijst. 36

2.6 Microsoft Azure IoT Hub prijslijst. 37

2.7 AWS IoT Core prijscomponent berichten prijslijst. 38

1.1 Algemene inputparameters voor de verschillende simulaties. 67

Lijst van codefragmenten

3.1 Angular Service: Component interactie met RxJS. 50
3.2 Angular Component: Component interactie met RxJS. 51
3.3 Mogelijke inhoud van de datastructuur voor gecalculeerde simulatiekosten. 52
3.4 Angular Service: HTTP API calls met de HttpClient interface. 53
3.5 Node.js back-end RESTful API. 53
3.6 Gebruiker model met behulp van de Mongoose bibliotheek. 54
3.7 Authenticatie en autorisatie in de back-end met behulp van Passport.js. 55
3.8 Authenticatie aan de hand van een JWT. 57
3.9 Angular CanActivate interface implementatie. 57
3.10 Data doorgeven bij Angular navigatie. 58
3.11 Doorgegeven data bij Angular navigatie consumeren. 59

Lijst van afkortingen

3G	3rd Generation. 16
3GPP	3rd Generation Partnership Project. 16, 17
ACK	Acknowledgement. 18
ADC	Analog-to-Digital Converter. 11
AMQP	Advanced Message Queuing Protocol. 16, 36
API	Application Programming Interface. 21, 44, 45, 47, 52, 53, 63
AWS	Amazon Web Services. 35, 37, 38, 69
BOM	Bill Of Material. 30, 31
BSON	Binary JSON. 44
CAPEX	Capital Expenditures. 25, 26, 28, 30, 33, 34, 40, 41, 51
CBS	Cost Breakdown Structure. 24–27, 29
CE	Conformité Européenne. 30
CoAP	Constrained Application Protocol. 16, 18, 19
CoRE	Constrained RESTful Environments. 18
CSS	Chirp Spread Spectrum. 15, 17
DBPSK	Differential Binary Phase Shift Keying. 14, 17
DIY	Do It Yourself. 11

DRY	Don't Repeat Yourself. 51
DTLS	Datagram Transport Layer Security. 18, 19
EER	Europese Economische Ruimte. 30
ERP	Enterprise Resource Planning. 21
FAVV	Federaal Agentschap voor de Veiligheid van de Voedselketen. 65
FDMA	Frequency Division Multiple Access. 16
GFSK	Gaussian Frequency Shift Keying. 14, 17
GPRS	General Packet Radio Service. 16
GSM	Global System for Mobile communications. 16
HTTP	HyperText Transfer Protocol. 16, 18, 36, 37, 44, 52, 54, 55, 57, 63
IAAS	Infrastructure As A Service. 21
IETF	Internet Engineering Task Force. 18, 19
IoT	Internet of Things. 1–3, 5–14, 16, 20, 21, 24, 26–38, 40–42, 45, 66, 67, 69–71
IP	International Protection. 30, 34
IP	Internet Protocol. 8, 14, 63
IPSO	IP for Smart Objects. 8
ISM	Industrial, Scientific and Medical. 12, 17
ITU	International Telecommunications Union. 7
JSON	JavaScript Object Notation. 44
JWT	JSON Web Token. 54–56

KB	KiloByte. 36–38
LoRa	Long Range. 14, 15, 17, 31, 33, 34, 68, 69
LoRaWAN	LoRa Wide Area Network. 15
LPWAN	Low-Power Wide-Area Network. 13, 14, 29, 32, 33, 67–70
LTE	Long Term Evolution. 16, 17
M2M	Machine-to-Machine. 14, 18
MB	MegaByte. 35, 36
MCU	Micro Controller Unit. 11, 31
MEAN	MongoDB, Express, Angular en Node.js. 43, 44
MEMS	Micro-ElectroMechanical Systems. 7
MQTT	Message Queuing Telemetry Transport. 16, 18, 19, 36, 37
MVC	Model-View-Controller. 43, 44, 47
NB	NarrowBand. 16
NB-IoT	NarrowBand-IoT. 14, 16, 17, 31, 33–35, 68, 69
OFDMA	Orthogonal FDMA. 16
OPEX	Operating Expenses. 26–28, 30, 32–35, 40, 41, 51
PCB	Printed Circuit Board. 31
QoS	Quality of Service. 18, 19
QPSK	Quadrature Phase Shift Keying. 16, 17

REST	REpresentational State Transfer. 18, 44, 52, 53
RxJS	Reactive Extensions for JavaScript. 49, 51
SAAS	Software As A Service. 31
SF	Spreading Factor. 15
SIM	Subscriber Identity Module. 34, 35
SSL	Secure Sockets Layer. 18, 19, 81
TCP	Transmission Control Protocol. 18, 19
TDD	Test Driven Development. 60, 61
TDOA	Time Difference Of Arrival. 15, 17
TLS	Transport Layer Security. 18, 19
UDP	User Datagram Protocol. 18, 19
UNB	Ultra NarrowBand. 14
URI	Universal Resource Identifier. 18
URL	Uniform Resource Locator. 47
WBS	Work Breakdown Structure. 24
WWW	World Wide Web. 16

Inleiding

1 Situering van de masterproef

In 2018 waren er naar schatting 17,8 miljard actieve toestellen verbonden met het Internet. Voorspellingen wijzen er op dat er in 2020 21,2 miljard en in 2025 maar liefst 34,2 miljard toestellen verbonden zullen zijn met het Internet, wat neer komt op een jaarlijkse stijging van ongeveer 10% [1]. Deze opmerkelijke evolutie zal er uiteindelijk toe leiden dat nagenoeg alles verbonden is met het Internet, ook wel bekend als het Internet of Things (IoT). Slechts enkele van de vele sectoren waar IoT een significante impact op zal hebben, zijn: transport en logistiek, industriële automatisatie, thuis automatisatie, gezondheidssector, ... Door de snelle evolutie van Internet geconnecteerde toestellen en de vele mogelijke toepassingssectoren zal de globale marktwaarde van IoT gigantisch stijgen van 151 miljard in 2018 tot 1.567 miljard in 2025 [1].

Enkele voorbeelden van de IoT-toepassingen van (over)morgen, die een significante impact gaan hebben op het dagdagelijkse leven van Jan met de pet, zijn onder andere:

- De nieuwste generatie digitale meters voor elektriciteit, gas en water die vanaf 1 juli 2019 in Vlaanderen worden geïnstalleerd bij nieuwbouwwoningen [2].
- Amazon thuislevering met behulp van drones [3].

De mogelijkheden zijn eindeloos en nog lang niet aan hun proefstuk toe.

2 Probleem- en doelstelling van de masterproef

2.1 Probleemstelling

IoT is duidelijk een *booming business*. Het is dan ook niet verwonderlijk dat verschillende bedrijven nieuwsgierig zijn of een IoT-systeem een positieve uitkomst kan bieden voor hun use

case. Bij de ontwikkeling van IoT-systemen dient de ontwikkelaar echter een keuze te maken uit, onder andere, de verschillende mogelijke IoT-connectiviteitstechnologieën en IoT-platformen. De verschillende mogelijkheden hebben naast verschillende eigenschappen en functionele karakteristieken ook verschillende kosten die eraan verbonden zijn. De uiteindelijke keuzes beïnvloeden de economische haalbaarheid van de use case. IoT use cases zijn vaak technisch haalbaar, maar zijn echter vaak niet economisch rendabel. Een IoT-ontwikkelaar heeft idealiter a priori een overzicht van zowel de investerings- als operationele kosten van een IoT-systeem om de afweging te maken of een IoT-systeem economisch haalbaar is. De inherente heterogeniteit van de verschillende componenten van een IoT-systeem maakt dit echter niet eenvoudig.

2.2 Doelstelling

De doelstelling van de masterproef is het ontwerpen en implementeren van een online selectie- en modelleringstool, dat toelaat om op een snelle en overzichtelijke manier de kosten van een IoT use case in kaart te brengen. De tool zal, na het ingeven van de benodigde use case parameters, een overzicht geven van de mogelijke ontwerpbeslissingen en de gebruiker in staat stellen om de economische impact van ontwerpbeslissingen te simuleren. De tool leidt de gebruiker als het ware naar de meest economische oplossing, die tevens voldoet aan alle functionele vereisten van de use case.

3 Methodiek

De vooropgestelde methodiek om de beoogde doelstelling te bereiken is als volgt:

1. Allereerst wordt er een literatuurstudie verricht met als doel inzichten te verzamelen in techno-economisch werk en in de verschillende (kost)componenten van een IoT-systeem.

2. Na de literatuurstudie wordt gepoogd om de verschillende kostendrijvers te bepalen voor de verschillende componenten van een IoT-systeem. Aan de hand van de vastgestelde kostendrijvers wordt dan een techno-economisch model ontwikkeld dat de gebruiker toelaat om de kosten van een IoT use case in kaart te brengen.

3. Het ontwikkelde techno-economisch model wordt gebruikt als basis voor het ontwerp en de implementatie van de webtool.

4. De ontwikkelde webtool valideren met idealiter twee IoT use cases.

4 Indeling van de scriptie

De scriptie is, naar analogie met de vooropgestelde methodiek, ingedeeld in vier verschillende delen. Deel I reproduceert de verworven kennis van de literatuurstudie en behandelt onder andere de sleutelconcepten van IoT en de verschillende componenten die samen een IoT-systeem vormen. Deel II behandelt het vooropgestelde techno-economisch model van een IoT-systeem en beschrijft de verschillende kostendrijvers van de verschillende componenten van een IoT-systeem. Deel III bediscussieert het ontwerp en de ontwikkeling van de webtool. Er wordt van start gegaan met een behoefteanalyse, daaropvolgend het ontwerp en de realisatie van de webapplicatie en als laatste een algemene reflectie van het ontwerp en de realisatie van de webapplicatie. Deel IV valideert en illustreert de webtool aan de hand van een praktische use case, namelijk het monitoren van de temperatuur in de koudeketen. Bijlagen A, B en C bevatten respectievelijk de installatie- en gebruikershandleiding van de ontwikkelde webapplicatie en de verschillende simulaties van de use case.

Deel I

Literatuurstudie

1

Internet Of Things

Dit hoofdstuk geeft een globale introductie tot IoT. Sectie 1.1 poogt een uniforme definitie te geven voor IoT. Sectie 1.2 geeft een chronologisch overzicht weer van de geschiedenis van IoT en de belangrijkste bijdragers ervan. Vervolgens beschrijft sectie 1.3 waarom, gezien de ruime geschiedenis van IoT, IoT pas recent in opmars is. Ten slotte introduceert sectie 1.4 de verschillende IoT-componenten die samen een *end-to-end* IoT-systeem vormen.

1.1 Wat is IoT?

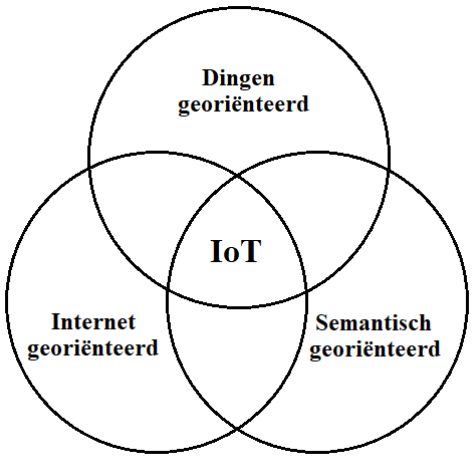
Er bestaat niet zoiets als een eenduidige definitie voor IoT. IoT betekent verschillende dingen voor verschillende mensen [4]. Hieronder zijn slechts enkele van de veel voorkomende verwoordingen van IoT weergegeven:

- IoT is een technologische revolutie die de toekomst van computers en communicatie vertegenwoordigt en de ontwikkeling ervan is afhankelijk van dynamische, technische innovatie op een aantal belangrijke gebieden, van draadloze sensoren tot nanotechnologie [5].
- IoT is een wereldwijde infrastructuur voor de informatiemaatschappij die geavanceerde diensten mogelijk maakt door onderling (fysieke en virtuele) dingen met elkaar te verbin-

den, gebaseerd op bestaande en evoluerende, interoperabele informatie- en communicatietechnologieën [6].

- IoT is de interconnectie van sensoren en actuatoren die de mogelijkheid biedt om via een uniform *framework* informatie over verschillende platformen te delen, waarbij een gemeenschappelijke operationele visie wordt ontwikkeld om innovatieve toepassingen mogelijk te maken. Dit wordt bereikt aan de hand van naadloze, alomvertegenwoordigde waarnemingen, gegevensanalyse en informatierepresentatie [7].

Het IoT paradigma is opgebouwd uit drie verschillende visies, namelijk Internet georiënteerd (connectiviteit), dingen georiënteerd (sensoren) en semantisch georiënteerd (kennis) [8]. Hoewel dit type afbakening mogelijk is vanwege het intrinsieke heterogene karakter van IoT, kan het volledige potentieel van IoT zich enkel ontplooien in een intersectie van de drie visies zoals is weergegeven in figuur 1.1.



Figuur 1.1: De drie verschillende visies op IoT.

De visie vanuit het perspectief van dingen slaat op slimme objecten. Dit zijn sensoren die niet alleen zijn uitgerust met de gebruikelijke draadloze communicatie, geheugen en verwerkingsmogelijkheden, maar ook met nieuwe mogelijkheden. Autonoom en proactief gedrag en contextbewustzijn zijn slechts enkele van de vele mogelijkheden [8]. De visie vanuit het perspectief van het Internet slaat op de connectiviteit en communicatie tussen verschillende slimme objecten over een netwerk. De visie vanuit het semantisch perspectief slaat op de grote verscheidenheid aan verschillende IoT-toepassingen, connectiviteitstechnologieën en communicatieprotocollen. Semantiek staat in voor het representeren, opslaan, interconnecteren, zoeken en organiseren van informatie die door IoT is gegenereerd.

Het is duidelijk dat er geen eenduidige definitie bestaat voor IoT. IoT staat voor verschillende

ideeën, concepten en visies. IoT is geëvolueerd van de samensmelting van draadloze connectiviteitstechnologieën, Micro-ElectroMechanical Systems (MEMS), microservices en het Internet. De samensmelting heeft geholpen de silo's tussen operationele technologie en informatietechnologie af te breken, waardoor ongestructureerde, door machine gegenereerde gegevens, kunnen worden geanalyseerd op inzichten om verbeteringen aan te brengen [9]. IoT kan dan ook eerder aangeduid worden als de vierde revolutie tot een intergeconnecteerde wereld, na telefonie, televisie en het Internet.

1.2 Geschiedenis van IoT

IoT heeft de laatste jaren een enorme opmars aan populariteit gekend. Nochtans is de term IoT of het idee om alledaagse dingen te verbinden met het Internet zeker niet nieuw. Hieronder volgt een chronologische opsomming van de meest belangrijke bijdragers aan IoT zoals het de dag van vandaag gekend is.

Begin 1980 De allereerste IoT-toepassing was te vinden in de informatica afdeling van de Carnegie Mellon universiteit in Pittsburgh, Pennsylvania. Informatici installeerden microschakelaars in de Coca-Cola-automaat en verbonden deze met de centrale PDP-10-computer. Via het netwerk kon de status van de machine gecontroleerd worden en bepaald worden of er nog koude drankjes aanwezig waren [10].

1991 Mark Weiser publiceert het artikel *The Computer for the 21st Century* in Scientific American. Het artikel gebruikt de termen *ubiquitous computing* en *embodied virtuality* om zijn visie te beschrijven over hoe gespecialiseerde elementen van hardware en software, verbonden door draden, radiogolven en infrarood, in de toekomst zo alomtegenwoordig zullen zijn dat niemand hun aanwezigheid zal opmerken [11].

1999 De term Internet of Things werd voor het eerst gebruikt door Kevin Ashton in een presentatie voor Procter & Gamble [12]. Hij gebruikte de term om aan te geven dat niet langer mensen maar dingen data zullen genereren op het Internet. MIT professor Neil Gershenfeld's boek, *When Things Start to Think*, dat ook in 1999 verscheen, maakte geen gebruik van de exacte term, maar bood een duidelijke visie over wat de mogelijkheden waren van IoT [13].

2005 De International Telecommunications Union (ITU) publiceert de zevende in haar reeks rapporten over het Internet, getiteld "The Internet of Things"[5].

2008 Een groep bedrijven lanceert samen de Internet Protocol for Smart Objects (IPSO) Alliance, om het gebruik van het Internet Protocol (IP) in netwerken van slimme objecten te promoten en IoT mogelijk te maken [14].

2009 Er zijn meer dingen verbonden met het Internet dan personen. Het overgrote deel bestaat uit smartphones, tablets en computers [15].

1.3 Instaatstellers van IoT

Zoals is toegelicht in sectie 1.2 is het idee van een intergeconnecteerde wereld zeker niet nieuw, toch is IoT pas relatief recent in opmars. Dit heeft zowel op hardware- als softwarematig vlak een aantal verschillende oorzaken.

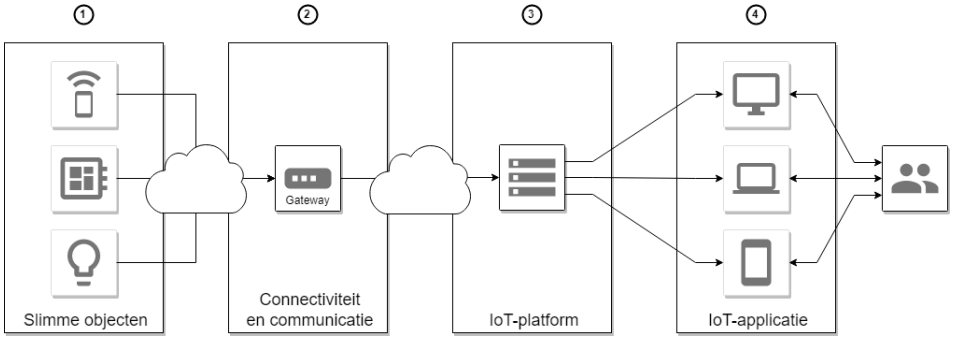
Op hardwarematig vlak bijvoorbeeld stelt de wet van Moore dat het aantal transistoren in een dicht geïntegreerd circuit ongeveer elke twee jaar verdubbelt. Dit heeft de miniaturisatie van hardware als direct gevolg, waardoor steeds meer verwerkingskracht aanwezig kan zijn in een kleiner formaat. Hierdoor is het mogelijk om in alledaagse, banale dingen hardware te integreren. Hardware wordt niet alleen kleiner maar ook steeds energie-efficiënter, wat een langere batterijlevensduur van toestellen waarborgt. Hardware wordt ook steeds goedkoper waardoor bijvoorbeeld massaproductie van sensoren mogelijk is.

Op softwarematig vlak is er, mede dankzij de IPSO Alliance, veel vooruitgang geboekt op gebied van standaardisatie van energie-efficiënte connectiviteitstechnologieën en communicatieprotocollen voor embedded toestellen met beperkte capaciteiten. Cloudservices daarentegen voorzien de mogelijkheid om grote hoeveelheden data op te slaan en te verwerken. Bijkomstig is dat de kostprijs voor draadloze connectiviteit en communicatie de laatste 15 jaar met maar liefst 97% is gedaald [16].

1.4 IoT-architectuur

De architectuur van een generisch IoT-systeem bestaat uit vier verschillende componenten zoals afgebeeld in figuur 1.2. Slimme objecten zijn de bouwstenen van IoT en vormen de onderste laag van een IoT-systeem. Het zijn de fysieke toestellen die aan de hand van sensoren informatie uit hun omgeving gaan waarnemen en capteren en die mogelijks ook zelf invloed uitoefenen op hun omgeving aan de hand van actuatoren. De component connectiviteit en communicatie is de link tussen de fysieke wereld, de slimme objecten, en de virtuele wereld, het Internet. De derde

component, het IoT-platform, is de centrale *backbone* van elk IoT-systeem. De laatste component, de IoT-applicatie vormt de interactie met eindgebruikers van een IoT-systeem. Hoofdstuk 2 gaat in detail in op de verschillende componenten van een IoT-systeem.



Figuur 1.2: De vier verschillende componenten van een IoT-systeem.

2

De componenten van een IoT-systeem

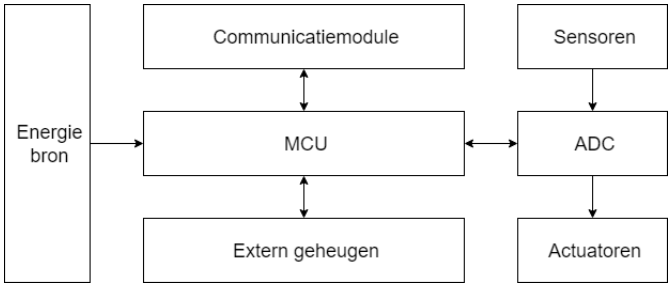
Dit hoofdstuk behandelt in detail de vier verschillende componenten van een IoT-systeem, zoals vooropgesteld in sectie 1.4. Sectie 2.1 bespreekt de bouwstenen van elk IoT-systeem, namelijk de slimme objecten. Sectie 2.2 behandelt de link tussen de fysieke wereld en de virtuele wereld. Sectie 2.3 beschrijft de centrale backbone van elk IoT-systeem, namelijk het IoT-platform. Ten slotte behandelt sectie 2.4 de interactie met eindgebruikers van het IoT-systeem aan de hand van de IoT-applicatie.

2.1 Slimme objecten

Slimme objecten zijn de bouwstenen van IoT en vormen de onderste laag van een IoT-systeem. Het zijn fysieke objecten die doorheen hun volledige levenscyclus uniek identificeerbaar zijn en onafhankelijk kunnen interageren met hun omgeving en andere slimme objecten [17].

Slimme objecten zijn er in alle vormen, kleuren en maten. De grootte van een slim object kan variëren van klein tot zeer groot, het kan een enkele, eenvoudige functie uitvoeren of net complexe *on-board* intelligentie hebben. Slimme objecten bevinden zich mogelijks op vaste, gekende of ongekende locaties of zijn net beweeglijk. Slimme objecten kunnen zowel bedraad als draadloos zijn. Doorheen de scriptie wordt uitgegaan van een draadloos standpunt. Ondanks de vele

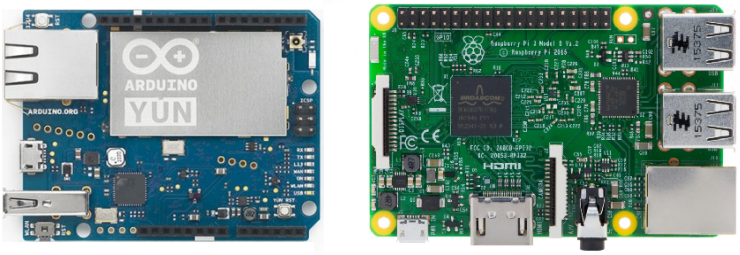
verschillen, is er één algemene vereiste voor alle draadloze slimme objecten; ze moeten namelijk beschikken over een energiebron om te kunnen werken. Omwille van de eindige energiebron zegt men dat deze objecten *constrained* zijn. Constrained objecten hebben naargelang het toepassingsgebied van het IoT-systeem (hevige) restricties op onder andere het energieverbruik, geheugenhoeveelheid, rekenkracht en mogelijks de afmetingen van het object [18].



Figuur 2.1: Architectuur van een slim object.

De architectuur van een slim object is weergegeven in figuur 2.1. De centrale component is de Micro Controller Unit (MCU), die onder andere instaat voor het verwerken van gegevens en het besturen van de functionaliteit van de andere componenten. De communicatiemodule staat in voor de connectiviteit en communicatie met de buitenwereld. De sensoren en actuatoren staan in contact met de omgeving via een Analog-to-Digital Converter (ADC). De sensoren nemen de omgeving waar, terwijl actuatoren invloed kunnen uitoefenen op de omgeving. Het is belangrijk om op te merken dat niet in alle slimme objecten de verschillende componenten aanwezig moeten zijn. Een slim object kan bijvoorbeeld enkel sensoren bevatten en geen actuatoren.

Er bestaan heel wat kant-en-klare Do It Yourself (DIY) producten om aangepaste IoT-applicaties en producten te ontwikkelen. Arduino en Raspberry Pi bijvoorbeeld, bieden een breed scala open-source boards aan die in staat zijn om simpele taken, zoals een LED laten knipperen, tot complexe netwerktaken uit te voeren.



(a) Arduino YÚN [19] (b) Raspberry Pi 3 [20]

Figuur 2.2: DIY slimme objecten.

2.2 Connectiviteit en communicatie

Connectiviteit en communicatie is een essentiële component van elk IoT-systeem. De component connectiviteit en communicatie realiseert de uitwisseling van data tussen de slimme objecten en het IoT-platform over een netwerk. Het is de link tussen de fysieke wereld, de slimme objecten, en de virtuele wereld, het Internet.

Er is een grote verscheidenheid aan technologieën om IoT-connectiviteit te realiseren, elk met hun eigen karakteristieken, voordelen, beperkingen en verbonden kosten. Een eerste indeling van IoT-connectiviteitstechnologieën kan gemaakt worden op basis van of de connectiviteit plaats vindt in het *licensed* of *unlicensed* spectrum. IoT-systemen die gebruik maken van het licensed spectrum dienen te beschikken over een licentie. De licentie waarborgt het exclusieve recht om data te zenden in een frequentieband in een geografische regio [21]. De meeste IoT-systemen maken echter gebruik van het unlicensed spectrum. Er is geen licentie vereist om de frequentieband te gebruiken maar de IoT-systemen dienen zich wel te houden aan de opgelegde technische beperkingen (bijvoorbeeld *maximum transmission power* en *duty cycle* limieten). De technische beperkingen garanderen *fair use* van de unlicensed frequentieband voor alle technologieën die er gebruik van wensen te maken. Naar de unlicensed frequentieband wordt meestal gerefereerd als de Industrial, Scientific and Medical (ISM) frequentieband.

Een tweede indeling van IoT-connectiviteitstechnologieën kan gemaakt worden op basis van de karakteristieken energieconsumptie, bandbreedte en bereik. Idealiter wordt er een zo hoog mogelijke bandbreedte en bereik behaald met een zo laag mogelijke energieconsumptie, maar het is belangrijk om op te merken dat er altijd een *tradeoff* bestaat tussen deze verschillende karakteristieken. Gebruikelijk wijzen een hogere bandbreedte en hoger bereik op een hogere energieconsumptie.

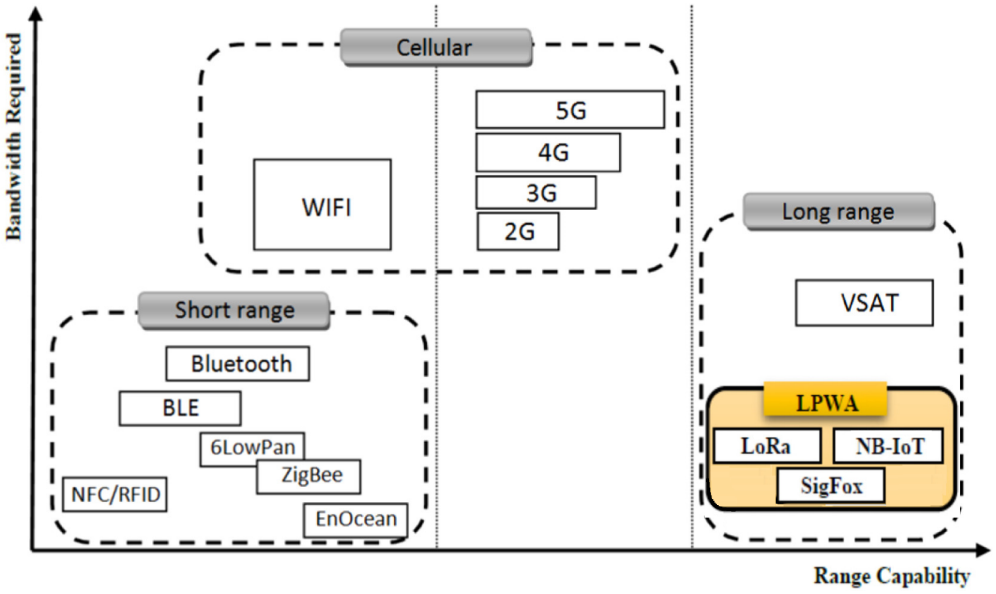
Er is een opdeling gemaakt aan de hand van het TCP/IP model: eerst worden de voornaamste vertegenwoordigers van IoT-connectiviteitstechnologieën in de netwerktoegangslaag besproken in subsectie 2.2.1, waarna de voornaamste IoT-communicatieprotocollen in de applicatielaag besproken worden in subsectie 2.2.2.

Figuur 2.3: TCP/IP model.

2.2.1 Netwerktoegangslaag connectiviteitstechnologieën

Tot voor kort waren de twee meest belangrijke IoT-connectiviteitstechnologieën in de netwerktoegangslaag *multihop mesh* netwerken die gebruik maken van *short-range* technologieën in het unlicensed spectrum en *long-range* cullulaire technologieën die gebruik maken van het licensed spectrum. Sinds kort is er ook een nieuw type IoT-connectiviteitstechnologie beschikbaar, namelijk de zogenaamde Low-Power Wide-Area Networks (LPWAN). Zo recent als begin 2013 bestond de term LPWAN nog niet eens [22]. LPWAN-technologieën worden gekenmerkt door een hoog bereik, lage bandbreedte, lage energieconsumptie en lage kostprijs. Deze karakteristieken maken LPWAN-technologieën uitermate geschikt voor IoT-systemen die kleine hoeveelheden data moeten zenden over lange afstanden over een lange periode van tijd.

Figuur 2.4 geeft een overzicht van de voornaamste vertegenwoordigers van IoT-connectiviteitstechnologieën in de netwerktoegangslaag.



Figuur 2.4: IoT-connectiviteitstechnologieën in de netwerktoegangslaag [22], [23].

LPWAN

LPWAN biedt langeafstandscommunicatie aan van 10 tot 40 km in landelijke gebieden en tot 5 km in stedelijke gebieden [24]. Bovendien is LPWAN ook zeer energie-efficiënt, met een batterij levensduur van meer dan tien jaar [25], en niet duur.

De meeste LPWAN-technologieën maken gebruik van de Sub-GHz frequentieband, wat resulteert

in een robuuste en betrouwbare connectiviteit met lage energieconsumptie. IoT-connectiviteitstechnologieën die gebruik maken van lagere frequentiebanden ervaren minder verzwakking veroorzaakt door obstakels zoals muren, gebouwen of mensen. Ook is de Sub-GHz frequentieband minder belast dan bijvoorbeeld de 2.4 GHz frequentieband, die ondermeer wordt gebruikt voor Wi-Fi, Bluetooth, ZigBee en andere huishoudelijke apparaten.

Netwerken van LPWAN-technologieën worden gestructureerd als een stertopologie of als een ster van sterren. Elk slim object is in directe verbinding met een enkel centraal punt, algemeen een *gateway* of *base station* genoemd, die ook de overbrugging naar de IP-wereld vormt [24]. *Uplink* netwerkverkeer verloopt van de slimme objecten via de gateway naar het Internet en *downlink* netwerkverkeer verloopt van het Internet via de gateway naar de slimme objecten.

LPWAN-technologieën zijn ontstaan in zowel het licensed als het unlicensed spectrum. Onder hen zijn Sigfox, Long Range (LoRa) en NarrowBand-IoT (NB-IoT) de voornaamste vertegenwoordigers. Hieronder worden, na een introductie in deze drie LPWAN-technologieën, de meest belangrijke technische en functionele karakteristieken opgesomd in tabel 2.1.

Sigfox

De Sigfox technologie werd in 2010 ontwikkeld door de start-up Sigfox in Toulouse, Frankrijk. Sigfox is zowel een bedrijf als een LPWAN-netwerkprovider. Sigfox opereert en commercialiseert zijn eigen IoT-oplossing in 60 landen en wordt nog steeds wereldwijd uitgerold in samenwerking met verschillende netwerkproviders. In België werkt Sigfox samen met de netwerkprovider Engie M2M, zij garanderen een *coverage* van 99% in België [26].

De Sigfox technologie opereert in het Sub-GHz unlicensed spectrum en maakt gebruik van Ultra NarrowBand (UNB) radio technologie in combinatie met Differential Binary Phase Shift Keying (DBPSK) en Gaussian Frequency Shift Keying (GFSK). Elk Sigfox bericht gebruikt slechts 100 Hz van de 200 kHz beschikbare bandbreedte en behaalt *datarates* van 100 tot 600 bps, afhankelijk van de regio [27]. In Europa wordt de 868-869 MHz frequentieband gebruikt en bedraagt de uplink datarate 100 bps en de downlink datarate 600 bps. Sigfox berichten zijn zeer klein, met respectievelijk een maximale payload van 12 bytes in uplink verkeer en 8 bytes in downlink verkeer.

Slimme objecten die gebruik maken van de Sigfox connectiviteitstechnologie hebben een opgelegde duty cycle van maximaal 1% uplink. Door de lage datarates kan het wel tot twee seconden duren om een bericht te zenden, wat resulteert in een maximum van 140 uplink berichten per slim object per dag. Downlink berichten kunnen enkel plaatsvinden na een uplink bericht en hebben dan weer een opgelegde duty cycle van 10% waardoor er een maximum geldt van vier downlink

berichten per slim object per dag. Dit betekent dat uplink berichten niet bevestigd worden. De betrouwbaarheid van de uplink communicatie wordt gewaarborgd door gebruik te maken van tijds- en frequentiediversiteit alsook door transmissieduplicatie [23]. Elk uplink bericht wordt standaard drie maal verstuurd over drie verschillende frequentiekanalen.

LoRa

LoRa werd in 2009 ontwikkeld door de start-up Cycleo in Grenoble, Frankrijk en werd drie jaar later gekocht door het Amerikaanse bedrijf Semtech. LoRa is momenteel actief in 100 landen. Het LoRa-netwerk in België wordt verzorgd door Proximus, The Things Network Foundation en LORIOT [28].

De LoRa-technologie opereert in het Sub-GHz unlicensed spectrum en moduleert de radiosignalen met behulp van een *spread spectrum* techniek, ontworpen en gepatenteerd door Semtech [29]. De bidirectionele communicatie wordt verzorgd door een speciale Chirp Spread Spectrum (CSS) modulatie, die een kleine frequentieband spreidt over een grotere bandbreedte. Het resulterende signaal heeft lage ruisniveaus, waardoor het een hoge storingsbestendigheid heeft en moeilijker te detecteren is of vastloopt [30].

LoRa gebruikt zes Spreading Factors (SFs), SF7 tot SF12, om de tradeoff tussen de datarate en het bereik aan te passen. Een hogere SF maakt een hoger bereik mogelijk ten koste van een lagere datarate en vice versa. Datarates liggen tussen de 300 bps en 50 kbps, afhankelijk van de SF en bandbreedte; LoRa-datarates zijn dus adaptief. Berichten die met verschillende SFs zijn verzonden kunnen gelijktijdig door LoRa-gateways worden ontvangen [31]. De maximale payload voor elk bericht bedraagt 243 bytes.

Het LoRa-communicatieprotocol, LoRa Wide Area Network (LoRaWAN) werd in 2015 gestandaardiseerd door de LoRa Alliance. Met LoRaWAN wordt elk verzonden bericht door een slim object ontvangen door alle LoRa-gateways in het bereik, waardoor de betrouwbaarheid van de uplink communicatie verbeterd wordt. De resulterende duplicate berichten worden weggefilterd in de netwerkserver, die ook instaat voor de beveiliging, het verzenden van bevestigingen en het verzenden van het bericht naar de corresponderende applicatieserver [23]. De duplicate berichtontvanging door de LoRa-gateways maakt ook localisatie van slimme objecten mogelijk aan de hand van de Time Difference Of Arrival (TDOA) techniek. Dit vereist enigszins wel een zeer accurate synchronisatie tussen de verschillende LoRa-gateways. Ook de overdracht tussen LoRa-gateways is niet nodig als de slimme objecten mobiel zijn; mobiliteit wordt bij LoRaWAN inherent ondersteund door de duplicate berichtontvanging.

NB-IoT

NB-IoT is een NarrowBand (NB) technologie gespecificeerd in de dertiende release van 3rd Generation Partnership Project (3GPP) in juni, 2016 [32]. De NB-IoT-technologie opereert in het Sub-GHz licensed spectrum. In België hebben zowel Orange [33], Proximus [34] als Telenet [35] een NB-IoT netwerk uitgerold.

De NB-IoT-technologie is gebaseerd op de Long Term Evolution (LTE) technologie. NB-IoT reduceert LTE-protocolfunctionaliteiten tot een minimum en verbetert de protocollen waar nodig voor gebruik binnen IoT [23]. NB-IoT kan beschouwd worden als een nieuwe interface bovenop de LTE-protocolstack. Hierdoor is het mogelijk om NB-IoT te ondersteunen door slechts een software-update te doen bovenop bestaande LTE-infrastructuur [29]. NB-IoT is niet compatibel met 3rd Generation (3G), maar kan wel opereren naast Global System for Mobile communications (GSM), General Packet Radio Service (GPRS) en uiteraard LTE [29].

NB-IoT gebruikt *single-carrier* Frequency Division Multiple Acces (FDMA) in de uplink en Orthogonal FDMA (OFDMA) in de downlink en gebruikt hierbij Quadrature Phase Shift Keying (QPSK) [36]. De datarate is gelimiteerd tot 20 kbps voor de uplink en 200 kbps voor de downlink. De maximum payload van de berichten is zowel 1600 bytes voor uplink als downlink communicatie. Zoals beschreven in [37] kan NB-IoT-technologie een batterij levensduur van tien jaar bereiken als er gemiddeld 200 bytes data per dag worden verstuurd.

2.2.2 Applicatielaag communicatieprotocollen

In tegenstelling tot het World Wide Web (WWW) dat berust op een enkel communicatieprotocol, namelijk het HyperText Transfer Protocol (HTTP) bestaat er geen *one size fits all* communicatieprotocol voor IoT [38]. Er zijn verschillende IoT-communicatieprotocollen ontwikkeld voor verschillende IoT-toepassingen. Voor snelle en vertrouwelijke zakelijke communicatietransacties, is er bijvoorbeeld het Advanced Message Queuing Protocol (AMQP). Voor communicatie in constrained netwerken is er dan weer het Message Queuing Telemetry Transport (MQTT) protocol en het Constrained Application Protocol (CoAP). Deze verscheidenheid toont duidelijk aan dat de toekomst van IoT op verschillende communicatieprotocollen berust en dat eender welk IoT-communicatieprotocol niet kan dienen voor elke IoT use case [39]. Hieronder wordt, na een introductie tot de twee IoT-communicatieprotocollen CoAP en MQTT, een vergelijking gegeven in tabel 2.2.

Tabel 2.1: Technische en functionele karakteristieken van SigFox, LoRa en NB-IoT.

	Sigfox	LoRa	NB-IoT
Technische karakteristieken			
Modulatie	DBPSK (UL) en GFSK (DL)	CSS	QPSK
Frequentieband	unlicensed ISM-band	unlicensed ISM-band	licensed LTE-band
Bandbreedte	100 Hz	250 kHz en 125 kHz	200 kHz
Maximum datarate	100 bps	50 kbps	200 kbps
Bidirectioneel	gelimiteerd (half-duplex)	ja (half-duplex)	ja (half-duplex)
Maximum aantal berichten per dag	140 (UL), 4 (DL)	ongelimiteerd	ongelimiteerd
Maximum payload lengte	12B (UL), 8B (DL)	243B	1600B
Bereik	10 km (stedelijk), 40 km (landelijk)	5 km (stedelijk), 20 km (landelijk)	1 km (stedelijk), 10 km (landelijk)
Handover	niet enkele gateway	niet enkele gateway	enkele gateway
Standaardisatie	onder specificatie	LoRa Alliance	3GPP
Functionele karakteristieken			
Adaptieve datarate	nee	ja	nee
Storingsbestendigheid	zeer hoog	zeer hoog	laag
Energie-efficiëntie	zeer hoog	zeer hoog	hoog
Privaat netwerk mogelijk	nee	ja	nee
Authenticatie en encryptie	niet ondersteund	ja (AES 128b)	ja (LTE-encryptie)
Localisatie	ja (RSSI)	ja (TDOA)	niet ondersteund

CoAP

CoAP is een lichtgewicht Machine-to-Machine (M2M) protocol ontwikkeld door de Internet Engineering Task Force (IETF) Constrained RESTful Environments (CoRE) werkgroep. CoAP ondersteunt zowel *request/response* als *resource/observe* (een variant van *publish/subscribe*) architecturen [40]. CoAP is voornamelijk ontwikkeld om via eenvoudige proxy's samen te werken met HTTP en RESTful webservices in constrained netwerken [39]. CoAP gebruikt de HTTP commando's GET, POST, PUT en DELETE om resource-georiënteerde interacties aan te bieden in een client-serverarchitectuur [41]. In tegenstelling tot MQTT gebruikt CoAP Universal Resource Identifiers (URIs) in plaats van *topics* [42].

CoAP gebruikt het User Datagram Protocol (UDP) als transport protocol en het Datagram Transport Layer Security (DTLS) protocol voor beveiliging om de protocolimplementatie en overhead licht te houden. De communicatie is dus verbindingsloos en minder betrouwbaar. CoAP gebruikt echter bevestigbare en niet-bevestigbare berichten om twee verschillende niveaus van Quality Of Service (QoS) te bieden. Bevestigbare berichten moeten worden bevestigd door de ontvanger door middel van het zenden van een Acknowledgement (ACK), niet-bevestigbare berichten moeten door de ontvanger niet bevestigd worden.

MQTT

MQTT is een van de oudste M2M-communicatieprotocollen en is ontworpen in 1999 door IBM [39]. MQTT is een publish/subscribe messaging protocol ontworpen voor lichtgewicht M2M-communicatie in constrained netwerken [40]. Het is een zeer basic messaging protocol en bevat slechts enkele controle opties [39]. MQTT clients publiceren berichten naar een MQTT *broker*. Elk bericht wordt gepubliceerd naar een adres dat bekend staat als een topic [43]. Clients kunnen zich abonneren op meerdere topics en ontvangen elk bericht dat voor elke topic is gepubliceerd. Het MQTT-protocol is ontworpen om spaarzaam gebruik te maken van bandbreedte en batterijgebruik en wordt hierom bijvoorbeeld momenteel gebruikt door Facebook Messenger [44].

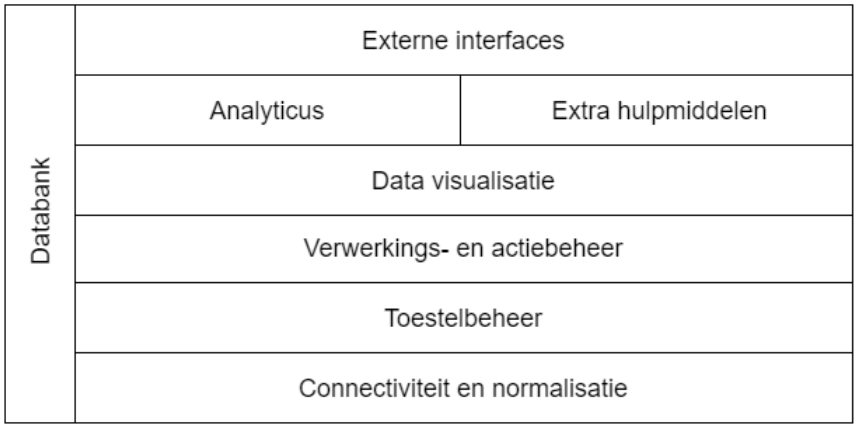
MQTT gebruikt het Transmission Control Protocol (TCP) als transportprotocol en Transport Layer Security/Secure Sockets Layer (TLS/SSL) voor beveiliging. De communicatie is verbindingsgericht en ondanks het gebruik maakt van TCP, is de implementatie en overhead licht. MQTT voorziet drie niveaus van QoS voor betrouwbare aflevering van berichten [40]. MQTT is het meest geschikt voor grote netwerken van slimme objecten die moeten worden gemonitord of bestuurd vanop afstand [39]. Volgens een recente studie [42] ondervindt MQTT lagere vertragingen dan CoAP bij lage pakketverliezen, maar genereert CoAP minder extra netwerkverkeer voor het garanderen van betrouwbaarheid.

Tabel 2.2: Vergelijking tussen IoT-communicatieprotocollen CoAP en MQTT.

	CoAP	MQTT
Ontworpen in	2010	1999
Architectuur	client/server of client/broker	client/broker
Abstractie	request/response of publish/subscribe	publish/subscribe
Header lengte	4 Byte	2 Byte
Semantics/methoden	GET, POST, PUT, DELETE	connect, disconnect, publish, subscribe, unsubscribe, close
QoS	bevestigbare berichten of niet-bevestigbare berichten	QoS - 0 maximaal eenmaal, QoS - 1 minimaal eenmaal, QoS - 2 precies eenmaal
Standaardisatie	IETF en Eclipse Foundation	OASIS en Eclipse Foundation
Transportprotocol	UDP, SCTP	TCP
Beveiliging	DTLS en IPSec	TLS/SSL
Standaard poort	5683 (UDP) en 5684 (DTLS)	1883 en 8883 (TLS/SSL)
Encodering	binair	binair
Licensing model	open-source	open-source

2.3 IoT-platform

IoT-platformen zijn de centrale backbone van elk IoT-systeem en zijn onder andere te onderscheiden op vlak van segment-focus, technologische diepte en technologische implementatie [45]. Het is belangrijk om op te merken dat er een enorme verscheidenheid bestaat aan IoT-platformen, met elk hun eigen specificaties en mogelijkheden. Wat doorgaans wordt beschouwd als een end-to-end IoT-platform bestaat uit acht bouwblokken zoals weergegeven in figuur 2.5 [45].



Figuur 2.5: Acht componenten van een end-to-end IoT-platform [45].

De minimale vereiste waaraan een IoT-platform moet voldoen, is het voorzien van connectiviteit en normalisatie. De connectiviteit- en normalisatielaag verzorgt de communicatie met de slimme objecten en normaliseert de verschillende communicatieprotocollen en dataformaten tot een uniforme interface. Het hebben van alle data van de slimme objecten in een uniform dataformaat is de basisnoodzaak om slimme objecten efficiënt te monitoren, beheren en analyseren. Het tweede bouwblok, de databank, is eveneens als de connectiviteit- en normalisatielaag aanwezig in elk IoT-platform. IoT-data opslaan tilt de vereisten van elke databank naar een nieuw niveau in termen van *volume, variety, velocity* en *veracity*. De databank moet in staat zijn om mogelijks gigantische hoeveelheden ongestructureerde data met grote snelheid te verwerken. De toestelbeheerlaag van een IoT-platform zorgt ervoor dat de verbonden slimme objecten correct functioneren en dat de software en applicaties zijn bijgewerkt. Indien er duizenden of zelfs miljoenen slimme objecten actief verbonden zijn met een IoT-platform dient de toestelbeheerlaag van het IoT-platform in te staan voor automatische configuratie van de slimme objecten. Waar mogelijk, voorziet de toestelbeheerlaag de mogelijkheid tot *over-the-air* updates van de slimme objecten of gateways om zowel kosten te drukken als manuele arbeid te verlichten. De verwerkings- en actiebeheerlaag brengt data tot leven met op regels gebaseerde *triggers* voor gebeurtenissen en acties, gebaseerd op specifieke sensorgegevens. De data visualisatielaag visualiseert de verwerkte sensordata om de gebruiker de mogelijkheid te bieden trends of patronen te

herkennen. Visualisatie wordt meestal gerealiseerd aan de hand van grafieken, diagrammen en modellen. De analyticuslaag staat in voor complexe analyses op de datastroom afkomstig van de slimme objecten. De analyse-engine omvat alle dynamische berekeningen op sensorgegevens, van basis gegevensclustering tot *deep machine learning*. Geavanceerde IoT-platformen bieden vaak, naast de analyticuslaag, een extra set van tools aan voor de IoT-ontwikkelaars en managers van het IoT-systeem. De extra tools laten de IoT-ontwikkelaars bijvoorbeeld toe om use cases te testen aan de hand van prototyping of stellen de managers in staat om rapporten te genereren. Als laatste laat de externe interfaceslaag eenvoudige integratie met externe systemen toe aan de hand van Application Programming Interfaces (APIs). Hierme kan het IoT-systeem bijvoorbeeld geïntegreerd worden met een bestaand Enterprise Resource Planning (ERP) systeem.

Naast wat doorgaans wordt beschouwd als een end-to-end IoT-platform zijn er ook IoT-platformen die slechts gedeeltelijke functionaliteit aanbieden. Deze IoT-platformen kunnen in vier groepen geclassificeerd worden [46]. De eerste groep focust vooral op connectiviteit en communicatie van de slimme objecten. De tweede groep biedt vooral verwerkingskracht en hostingruimte voor IoT-applicaties aan en valt onder de noemer van Infrastructure As A Service (IAAS). De derde groep bestaat uit hardware-specifieke softwareplatformen die gemaakt zijn voor specifieke IoT-hardware. Ten slotte bestaat de laatste groep van IoT-platformen uit IoT-platformen die gericht zijn op integratie van verschillen IoT-applicaties.

2.4 IoT-applicatie

IoT-applicaties presenteren IoT use cases aan de eindgebruikers. IoT-applicaties variëren onder andere op basis van de aard van de sensorgegevens en bedrijfsbehoeften. Sommige IoT-applicaties zullen zich bijvoorbeeld richten op het monitoren van slimme objecten, andere dan weer op het besturen van slimme objecten. Het monitoren en besturen van slimme objecten vergt veel verschillende applicatiemodellen, programmeerpatronen en softwarestacks [47]. Het generisch in kaart brengen van de vereisten en beoogde doeleinden van IoT-applicaties is dan ook moeilijk.

Over het algemeen hebben IoT-applicaties geen directe interactie met de slimme objecten, in plaats daarvan gebeurt de communicatie via het IoT-platform. IoT-platformen vormen de brug tussen IoT-applicaties en de slimme objecten. Volgens [48] voorzien IoT-applicaties in drie kritieke functies: het voorzien van een gebruikersinterface, data-analyse en data visualisatie. Merk op dat deze drie functies ook aanwezig zijn in het vooropgestelde end-to-end IoT-platform maar dat dit in de IoT-applicatie op een hoger niveau uitgewerkt wordt. De gebruikersinterface bijvoorbeeld kan verschillende IoT-systemen, die dezelfde use case uitwerken, onderscheiden. Het ontwerpen van gebruikersinterfaces zijn use case afhankelijk, daarom zijn ze meestal op maat ontwikkeld of in het geval van IoT-platform interfaces, uniek geconfigureerd. De beste aanpak

is afhankelijk van de vereiste gebruikerservaring.

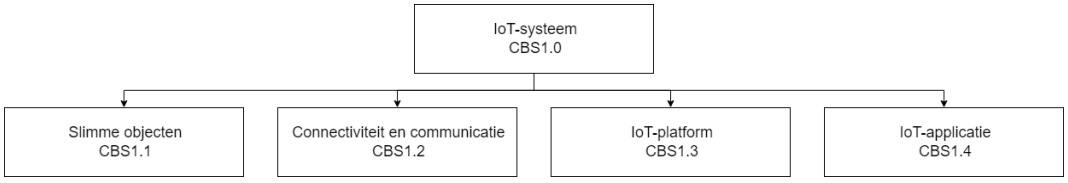
Deel II

Techno-economisch model van een IoT-systeem

1

Cost Breakdown Structure

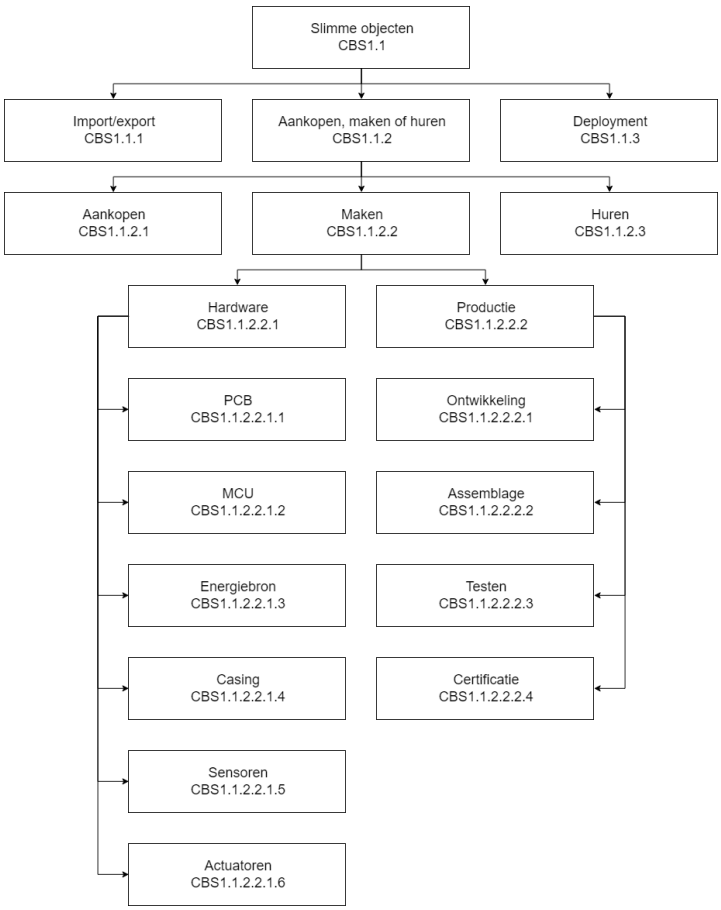
Na inzichten te hebben verzameld in de verschillende componenten van een IoT-systeem is, naar analogie met een Work Breakdown Structure (WBS), een Cost Breakdown Structure (CBS) opgesteld. Per definitie zou de sommatie van de verschillende kostcomponenten op het laagste niveau van het CBS een goede benadering moeten geven voor de totale kostprijs van een IoT-systeem. Belangrijk om op te merken is dat niet elk IoT-systeem steeds alle kostcomponenten bevat of dat er tussen de verschillende kostcomponenten een keuze dient gemaakt te worden. Figuur 1.1 geeft, naar analogie met de verschillende componenten van een IoT-systeem, het hoogste niveau weer van het CBS. De volgende secties van dit hoofdstuk behandelen telkens in detail een kostcomponent en wordt, waar mogelijk, het CBS op meerdere niveau's uitgebreid. Hoofdstuk 2 behandelt voor de verschillende kostcomponenten van het CBS de voornaamste kostendrijvers.



Figuur 1.1: De kostcomponenten van een IoT-systeem (CBS1.0).

1.1 Slimme objecten

De kosten die gepaard gaan met de kostcomponent slimme objecten (CBS1.1) kan verder worden onderverdeeld in drie kostcomponenten, namelijk de mogelijke import of export kost (CBS1.1.1), de beslissing of de slimme objecten *in-house* gemaakt, aangekocht of gehuurd worden (CBS1.1.2) en de deployment kost (CBS1.1.3). De verschillende kostcomponenten, met elk hun eigen verdere onderverdelingen, is weergegeven in figuur 1.2.



Figuur 1.2: De kostcomponenten van slimme objecten (CBS1.1).

De eerste kostcomponent, de mogelijke import of export kost, is van toepassing als hardware aangekocht of verkocht wordt buiten Europa. Indien dit het geval is moet er mogelijks een extra kost toegerekend worden en behoort dit tot de CAPEX van de slimme objecten kostcomponent.

De tweede kostcomponent is afhankelijk van of de slimme objecten in-house worden aangemaakt, gekocht of gehuurd. Indien er gekozen wordt om de slimme objecten aan te kopen, bestaat de CAPEX uit het product van het aantal slimme objecten met de individuele kost van een slim object. Hetzelfde geldt als de slimme objecten gehuurd worden, met als verschil dat voor

de slimme objecten een wederkerende huurprijs dient betaald te worden. Hierdoor vervalt de slimme objecten CAPEX kostcomponent en behoort de slimme objecten kostcomponent nu tot de OPEX.

Indien de slimme objecten in-house worden gemaakt dient, naast uiteraard het aankopen van de benodigde afzonderlijke hardwarecomponenten, de productie van de slimme objecten te gebeuren. De productie van hardware start met een ontwikkelingsfase, waarin het slim object wordt ontworpen en getest. Na de ontwikkeling worden de afzonderlijke hardware componenten geassembleerd, wat een kost per slim object met zich meebrengt. Na de assemblage van de slimme objecten worden deze getest. De ontworpen slimme objecten moeten in sommige gevallen gecertificeerd worden, om bijvoorbeeld connectiviteit te realiseren met publieke netwerkproviders. Certificatie is een eenmalige kost, ongeacht het aantal slimme objecten. De productie kostcomponent van het in-house aanmaken van slimme objecten draagt bij tot de CAPEX van de slimme objecten kostcomponent.

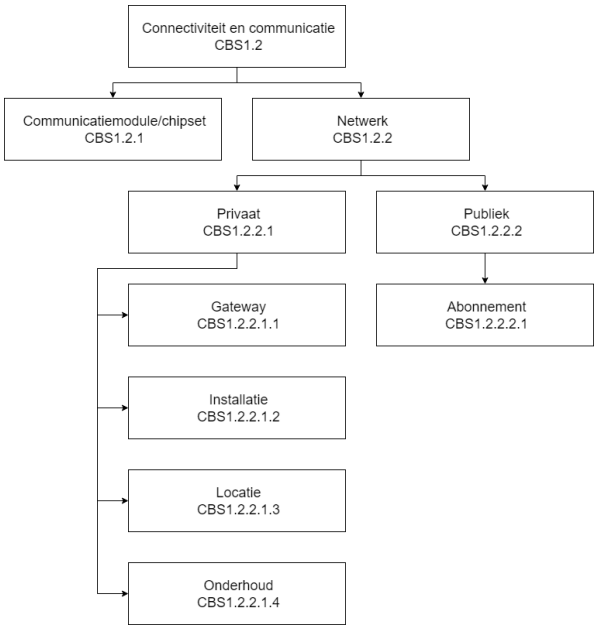
De derde kostcomponent, de deployment kost, is van toepassing op zowel slimme objecten die in-house werden aangemaakt, gekocht of gehuurd. Het vertegenwoordigt de kost die het uitrollen en installeren van de slimme objecten met zich meebrengt en draagt bij tot de CAPEX van de slimme objecten kostcomponent.

1.2 Connectiviteit en communicatie

De kosten die gepaard gaan met de kostcomponent connectiviteit en communicatie (CBS1.2) bestaan uit twee kostcomponenten, namelijk de kost van de communicatiemodule of chipset (CBS1.2.1) aanwezig op het slim object en de netwerkkost (CBS1.2.2) zoals is weergegeven in figuur 1.3. De communicatiemodule/chipset behoort tot de hardware van het slim object maar is expliciet geplaatst onder de component connectiviteit en communicatie. Dit omdat verschillende IoT-connectiviteitstechnologieën een verschillende kost dragen voor de communicatiemodule of chipset. De communicatiemodule/chipset kost is aanwezig voor elk slim object en maakt deel uit van de CAPEX.

De tweede kostcomponent, de netwerkkost, is afhankelijk van of er geopteerd wordt voor een privaat of publiek netwerk. Indien geopteerd wordt voor een privaat netwerk staat de IoT-ontwikkelaar zelf in voor het opzetten van het IoT-connectiviteitsnetwerk. De CAPEX bestaan uit de gateways, de installatiekost van de gateways en het opzetten van het netwerk. Indien gateways geïnstalleerd worden op publieke of op andere locaties dan de eigen eigendom dient er vaak een huurkost voor de locatie betaald te worden. De huurkost vormt samen met de onderhoudskost van het netwerk de OPEX van de connectiviteit en communicatie kostcomponent. Merk op dat som-

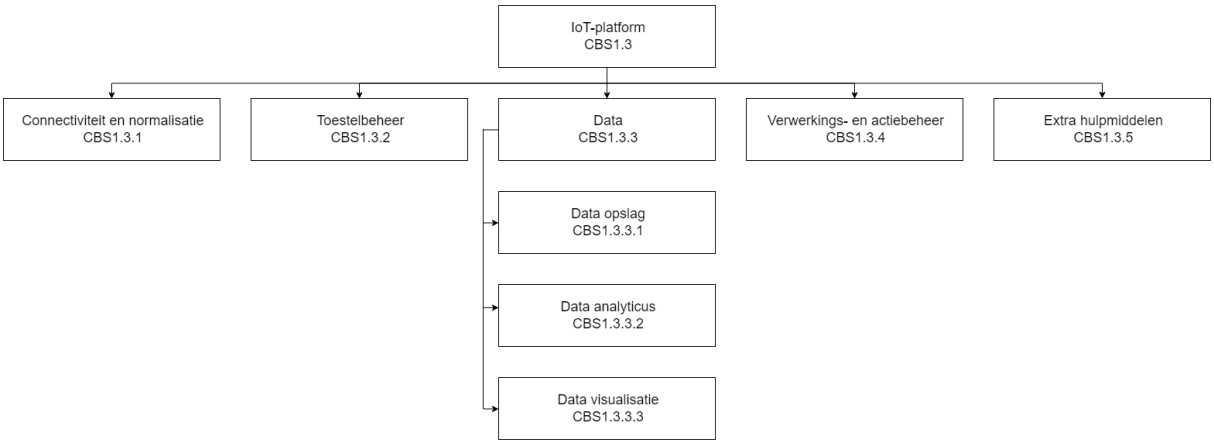
mige IoT-connectiviteitstechnologieën geen privaat netwerk ondersteunen, hierbij dient gebruik gemaakt te worden van een publieke netwerkprovider. Indien er geen privaat netwerk mogelijk is of er geopteerd wordt voor een publiek netwerk bestaat de OPEX van de connectiviteit en communicatie kostcomponent uit een abbonementskost per slim object. De publieke netwerkprovider staat dan in voor de installatie- en onderhoudskosten van het IoT-connectiviteitsnetwerk. Merk op dat in dit model de keuze tussen een privaat of publiek netwerk gemaakt dient te worden. Beide kunnen dus niet naast elkaar bestaan.



Figuur 1.3: De kostcomponenten van connectiviteit en communicatie (CBS1.2).

1.3 IoT-platform

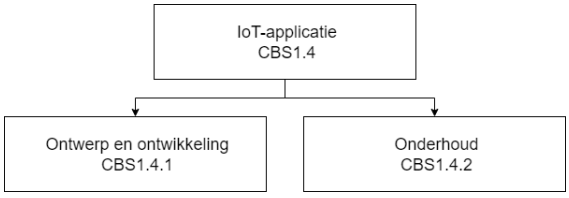
De verschillende kostcomponenten van de IoT-platform kostcomponent (CBS1.3) vormen zo goed als een één op één overeenkomst met het vooropgestelde end-to-end IoT-platform model in sectie 2.3 van deel I. Merk op dat, in tegenstelling tot de andere besproken kostcomponenten, de kostcomponenten van een IoT-platform kunnen overlappen. Het ene IoT-platform kan bijvoorbeeld de kostcomponent connectiviteit en normalisatie samennemen met de kostcomponent toestelbeheer, waarvoor eenzelfde wederkerende prijs dient betaald te worden. Andere IoT-platformen bieden dan weer niet alle diensten aan, waardoor de verantwoordelijkheid wordt verschoven naar de IoT-applicatie. De mogelijkheid om zelf een IoT-platform te ontwerpen, ontwikkelen en hosten wordt buiten beschouwing gelaten. Alle kostcomponenten van het IoT-platform behoren dus tot de OPEX van een IoT-systeem.



Figuur 1.4: De kostcomponenten van een IoT-platform (CBS1.3).

1.4 IoT-applicatie

IoT-applicaties zijn de grootste variabele kost van een IoT-systeem. De CAPEX bestaat uit het ontwerpen en ontwikkelen van de IoT-applicatie. De OPEX bestaat uit het onderhouden van de IoT-applicatie. Het onderhouden van een IoT-applicatie bestaat uit onder andere het oplossen van problemen, refractoring van code, bestaande features uitbreiden of nieuwe features toevoegen. De OPEX is grotendeels afhankelijk van op welke schaal de IoT-applicatie gebruikt wordt. De kostprijs van het ontwerpen en ontwikkelen van een IoT-applicatie hangt voornamelijk af van de ontwikkelingstijd, wat op zijn beurt afhangt van de vereisten van de IoT-applicatie. De vereisten van een IoT-applicatie zijn zeer use case specifiek en dan ook moeilijk om generisch in kaart te brengen. De kostenbepaling van het ontwerpen, ontwikkelen en onderhouden van een IoT-applicatie valt dan ook buiten de scope van deze masterproef.



Figuur 1.5: De kostcomponenten van een IoT-applicatie (CBS1.4).

2

Kostendrijvers van een IoT-systeem

Dit hoofdstuk behandelt, in analogie met het vooropgestelde CBS in hoofdstuk 1, de verschillende kostendrijvers van de componenten in een IoT-systeem. Voor de verschillende kostendrijvers is, waar mogelijk, de kostprijs weergegeven. Dit is ondermeer het geval bij de besproken voornaamste vertegenwoordigers van LPWAN-connectiviteitstechnologieën en de populairste grootschalige end-to-end IoT-platformen. Bij andere kostendrijvers uit de verschillende componenten van het CBS is een grootte-orde van de kosten weergegeven. Merk op dat sommige prijzen onderhevig zijn een de wisselkoers tussen de Euro en de Amerikaanse Dollar.

2.1 Slimme objecten

De kostendrijvers van een slim object hangen, naast de vereisten van het slim object, voornamelijk af van of het slim object *in-house* gemaakt wordt, aangekocht of gehuurd wordt.

2.1.1 Maken, aankopen of huren

Maken

De ontwikkelingskost is een grote initiële investering, zeer use case specifiek en dan ook moeilijk te becijferen. De ontwikkelingskost hangt ondermeer af of er genoeg in-house expertise is en van de vereisten van de slimme objecten, met andere woorden de moeilijkheidsgraad. Een expert in het ontwikkelen van slimme objecten, specifiek voor IoT-applicaties informeerde dat de ontwikkelingskost snel kan oplopen tot een grootte-orde van € 100.000 en enkele maanden tot jaren in beslag kan nemen.

De assemblagekost hangt ondermeer af van de beoogde International Protection (IP) waarde van het slim object. Indien een slim object bijvoorbeeld outdoor dient te worden gebruikt, vereist het slim object een hogere IP waarde. De expert informeerde dat de hogere eisen op vlak van IP van het slim object zich voornamelijk uiten in een hogere assemblagekost en niet in een hogere hardwarekost.

De kostendrijvers van certificatie zijn onder andere de benodigde tijd, kennis van certificatie en het huren van een testlabo. Sigfox hanteert bijvoorbeeld een eenmalige certificatiekost van € 1500 en daaropvolgende certificatiekosten van € 500 indien de slimme objecten een upgrade ondergaan [49]. Indien de slimme objecten verkocht worden in de Europese Economische Ruimte (EER) dienen deze te beschikken over de Conformité Européenne (CE) markering, om aan te geven dat de elektronische apparatuur voldoet aan alle Europese eisen voor veiligheid, gezondheid en milieubescherming. De expert informeerde dat het verkrijgen van een CE-keuring een kost heeft van ongeveer € 10.000, ongeacht het aantal slimme objecten.

Kopen

Indien er gekozen wordt om de slimme objecten aan te kopen, bestaat de CAPEX uit het product van het aantal slimme objecten met de individuele kost van een slim object. De individuele kost om een slim object op te zetten en uit te rollen behoort ook tot de CAPEX. Bij het aankopen van slimme objecten wordt uiteraard een meerprijs betaald op de hardwarecomponenten. De expert informeerde dat dit varieert tussen 1,6 tot 2,5 maal de prijs van de Bill Of Material (BOM).

Huren

Een model dat meer en meer zijn intrede doet is het huren van slimme objecten. Hierdoor vervalt de CAPEX en behoort de slimme objecten component tot OPEX. In de verhuur van slimme

objecten kunnen bijkomstige kosten ingebakken zijn zoals bijvoorbeeld het plaatsen en uitrollen van de slimme objecten of het onderhoud ervan.

2.1.2 Bill Of Material

De hardware kost wordt beschouwd als de sommatie van de BOM en de communicatiechip component kost. De afzonderlijke hardware componenten zoals PCB, MCU, casing, sensoren en actuatoren worden samengenomen omdat deze afzonderlijke componenten vaak dezelfde kost dragen en om tevens het aantal inputparameters te beperken. De communicatiechip component kost wordt specifiek buiten de BOM geplaatst zodat duidelijk het onderscheid kan gemaakt worden tussen verschillende IoT-connectiviteitstechnologieën. Tabel 2.1 geeft de typische elementen van de BOM van een slim object weer. Het valt op dat de communicatiechips van LoRa en Sigfox beduidend goedkoper zijn dan NB-IoT.

Tabel 2.1: Bill Of Material slim object.

Hardware component	Fabrikant (type)	Eenheidsprijs (bulk 100 stuks)	Eenheidsprijs (bulk 5000 stuks)
Temperatuur-sensor	Texas Instruments (LMT84LP)	€ 0,374	€ 0,252
MCU	STMicroelectronics (STM32L4)	€ 2,67	€ 1,78
Casing	Hammond (RZ250)	€ 3,86	€ 3,59
LoRa chip	Semtech (SX1272)	€ 4,94	€ 3,47
Sigfox chip	Microchip (ATA8520E)	€ 1,79	€ 1,79
NB-IoT	Nordic Semiconductor (nRF91)	€ 15,83	€ 14,71 (1000)

Schaalvoordelen en prijserosie

De kostprijs van hardware en in het bijzonder van elektronische componenten is onderhevig aan schaalvoordelen. Schaalvoordelen slaan op het feit dat indien er grote oplages van producten gekocht of gehuurd worden, de prijs per product daalt. Dit is van toepassing op de individuele componenten van de hardware van slimme objecten, zoals te zien is in tabel 2.1, maar eveneens als slimme objecten *off-the-shelf* aangekocht worden. Schaalvoordelen zijn ook niet enkel van toepassing op hardware maar bijvoorbeeld ook op SAAS diensten zoals het IoT-platform. Dit

wordt verder toegelicht in sectie 2.3.

Indien slimme objecten of hardware gefaseerd worden aangekocht geniet men in mindere mate van schaalvoordelen, al kunnen opeenvolgende aankopen van slimme objecten of hardware nog steeds goedkoper uitvallen door prijserosie. Prijserosie is het fenomeen dat een gegeven product in de toekomst minder duur zal zijn dan op dit moment. Dit is zeker van toepassing bij de relatief nieuwe LPWAN-connectiviteitstechnologieën met de bijhorende communicatiechip component. Voorspellingen zijn dat op termijn, de kosten van een communicatiemodule/chipset ongeveer € 2 en de operationele kost ongeveer € 1 per slim object per jaar zal bedragen [29].

2.1.3 Eindige energiebron

Als de slimme objecten beschikken over een eindige energiebron en met andere woorden niet aangesloten zijn op het elektriciteitsnet of geen gebruik maken van een vorm van *energy harvesting* zoals bijvoorbeeld zonne-energie, dan dienen de slimme objecten te beschikken over een batterij. Indien de batterijlevensduur verstrijkt voor de beoogde levensduur van het IoT-systeem dienen de batterijen vervangen te worden. Indien de kost van het vervangen van batterijen de kost van een nieuw slim object overstijgt kan er geopteerd worden om het slim object volledig te vervangen, al is dit in de praktijk eerder gering het geval. Het voorspellen van de levensduur van batterijen is een complexe en moeilijke taak. Indien de verschillende slimme objecten zich variabel gedragen, bijvoorbeeld in het aantal berichten die zij uitwisselen, zal de levensduur van de verschillende batterijen ook variabel zijn. In de praktijk worden de batterijen meestal reeds enige tijd voor het verstrijken van de levensduur vervangen. Het vervangen van batterijen draagt een groot deel bij tot de OPEX van een IoT-systeem en hangt ondermeer af van het aantal slimme objecten, de levensduur van de batterijen, de individuele batterijkost en de individuele batterijvervangingskost. De individuele batterijkost hangt ondermeer af van het type batterij en de batterijcapaciteit, uitgedrukt in mAh. De batterijen voor slimme objecten zijn typisch lithium batterijen aangezien deze gekenmerkt worden door een hoge batterijcapaciteit en lage zelfontlading. Bij het nagaan van verschillende prijzen voor verschillende types lithium batterijen werd al snel duidelijk dat batterijen slechts in beperkte oplage beschikbaar waren omwille van de grote vraag en het beperkte aanbod, wat ook bevestigd werd door de expert. De expert informeerde dat lithium batterijen, die typisch gebruikt worden in slimme objecten voor langdurige IoT-systemen, ongeveer € 5 kosten. Een Tadiran SL750/S batterij kost bijvoorbeeld € 4,46 per stuk indien aangekocht in bulk van 50 stuks. De batterij heeft een batterijcapaciteit van 1100 mAh en een uitvoerspanning van 3,6 V.

2.2 Connectiviteit en communicatie

Hieronder worden de verschillende prijsmodellen voor de voornaamste vertegenwoordigers van LPWAN-connectiviteitstechnologieën, namelijk Sigfox, LoRa en NB-IoT besproken. Zoals reeds eerder vermeld in subsectie 2.2.1 van deel I maken zowel Sigfox als LoRa gebruik van het unlicensed spectrum, terwijl NB-IoT gebruik maakt van het licensed spectrum. Tevens is er bij LoRa de mogelijkheid om een privaat IoT-connectiviteitsnetwerk op te zetten waardoor terugkerende abonnementskosten bij netwerkproviders vermeden worden. Zoals hieronder besproken wordt, treden er uiteraard dan wel andere terugkerende kosten op.

2.2.1 Sigfox

Het Sigfox netwerk wordt in België verzorgd door Engie M2M. Engie M2M biedt verschillende jaarlijkse abonnementen aan, naargelang het aantal berichten die de slimme objecten maximaal kunnen uitwisselen. De jaarlijkse abonnementen gaan van € 3,63 tot € 8,47 per slim object per jaar. In het eerste jaar van connectiviteit wordt er een eenmalige activatiekost aangerekend gelijk aan de jaarlijkse abonnementskost. Tabel 2.2 geeft een overzicht van de mogelijke abonnementen die Engie M2M aanbiedt. Daarnaast biedt Engie M2M ook nog abonnementen op maat aan die zullen resulteren in een mindere kost voor grote aantallen slimme objecten. De CAPEX bestaat dus uit de eenmalige activatiekost en de OPEX uit de terugkerende abonnementskosten.

Tabel 2.2: Sigfox Engie M2M abonnementen prijslijst.

Abonnement	Aantal uplink/downlink berichten per dag	Eenmalige startup kost	Jaarlijkse kost
One	2/0	€ 3,63	€ 3,63
Silver	50/1	€ 6,05	€ 6,05
Gold	100/2	€ 7,26	€ 7,26
Platinum	140/4	€ 8,47	€ 8,47

2.2.2 LoRa

Zoals reeds eerder vermeld heeft de LoRa-connectiviteitstechnologie de mogelijkheid om zowel privaat als publiek gebruikt te worden. De twee verschillende mogelijkheden met elk hun kostendrijvers worden hieronder toegelicht.

Publiek LoRa netwerk

In België stellen momenteel verschillende netwerkproviders zoals Proximus, The Things Network Foundation en LORIOT een publiek LoRa-netwerk beschikbaar. De prijzen die zij hanteren worden niet publiek gemaakt. De expert informeerde dat Proximus abonnementen heeft die varieren van € 1,30 per maand per slim object voor 24 uplink en 2 downlink berichten per dag tot € 2 per maand per slim object voor 288 uplink en 48 downlink berichten per dag. De initiële startup kost bedraagt ongeveer € 0,50 per slim object, ongeacht het gekozen abonnement.

Tabel 2.3: LoRa Proximus abonnementen prijslijst.

Aantal uplink/downlink berichten per dag	Eenmalige startup kost	Jaarlijkse kost
24/2	€ 0,50	€ 15,6
288/48	€ 0,50	€ 24

Privaat LoRa netwerk

Bij een privaat LoRa netwerk vervallen de abonnementskosten van een publieke LoRa-netwerk-provider. Het verkiezen van een privaat LoRa-netwerk brengt anderzijds wel andere kosten met zich mee. De CAPEX bestaat uit het opzetten van het privaat LoRa-netwerk, de OPEX uit het onderhouden ervan. De kostendrijvers van de CAPEX zijn het aantal LoRa-gateways, de individuele LoRa-gateway kost en de installatiekost. Industriële LoRa-gateways zijn beschikbaar in een prijsrange van € 300 tot € 1500, afhankelijk van ondermeer de IP-waarde. De Advantech WISE-3610 LoRa-gateway bijvoorbeeld kost 916,97€. De OPEX hangt nauw samen met de schaal waarop het privaat LoRa-netwerk wordt gebruikt. Gebruikelijk kan gesteld worden dat de jaarlijkse onderhoudskost 10 % van de initiële netwerkinstallatiekost is. Door de initiële grote investeringskost voor het opzetten van een privaat LoRa-netwerk is deze optie voornamelijk te overwegen bij grote IoT-systemen waarbij er veel slimme objecten worden verbonden.

2.2.3 NB-IoT

In België stellen onder andere Orange, Proximus en Telenet een NB-IoT netwerk beschikbaar. Orange maakt in tegenstelling tot Proximus en Telenet zijn beschikbare abonnementen publiek. Deze zijn opgelijst in tabel 2.4. De CAPEX bestaat uit de aankoop van de Subscriber Identity Module (SIM) kaart en bedraagt € 10 voor de eerste SIM-kaart en daaropvolgende aankopen € 2,50 per SIM-kaart. Bij het aankopen van tien SIM-kaarten is de gemiddelde prijs per SIM-kaart € 3,25 en bij 100 € 2,575. Bij het overschakelen naar een ander abonnement dient de SIM-kaart

niet vervangen te worden. De OPEX bestaat uit de jaarlijkse abonnementskost en is afhankelijk van het uitgewisselde datavolume per maand. Indien het datavolume overschreden wordt, wordt de connectiviteit van het slim object gedeactiveerd.

Tabel 2.4: NB-IoT Orange abonnementen prijslijst.

Abonnement	Datavolume per maand	Eenmalige SIM kost	Jaarlijkse kost
Light	1 MB	€ ≈2,50	€ 11
Plus	10 MB	€ ≈2,50	€ 18

1NCE biedt een opmerkelijk ander prijsmodel aan voor NB-IoT connectiviteit. Zij bieden een all-inclusive NB-IoT abonnement aan voor een eenmalige kost van € 10. De eenmalige kost is zowel voor de SIM-kaart als 500 MB inbegrepen datavolume. Er is geen extra kost voor activatie en geen wederkerende jaarlijkse kost. De gebruiker kiest zelf wanneer en hoeveel data verbruikt wordt. Indien de 500 MB datavolume is opgebruikt kan de gebruiker een nieuw tariefplan afsluiten voor opnieuw € 10. Roaming is inherent en gratis ondersteund in landen waarin 1NCE actief is. Jammer genoeg is 1NCE NB-IoT in Europa momenteel enkel actief in Duitsland. Ter vergelijking: een Orange Plus abonnement van vier jaar geeft maximaal 480 MB datavolume aan een kost van totaal € 82 (SIM-kaart (€ 10) + 4 * jaarlijkse abonnementskost (€ 18)).

2.3 IoT-platform

Verschillende IoT-platformen hanteren verschillende prijsmodellen. Gelukkig publiceren de meeste IoT-platformen publiekelijk de prijzen die zij hanteren. Een vergelijking tussen verschillende IoT-platformen is echter niet eenvoudig aangezien de verschillende IoT-platformen een verschillende terminologie gebruiken, of het ene IoT-platform de verschillende componenten van een Cloud oplossing opdeelt in verschillende pakketten met elk hun eigen specifiek prijsmodel. De enige overeenkomst tussen vrijwel alle IoT-platformen is dat zij geen startup kost aanrekenen, je betaalt dus enkel voor wat je gebruikt. De kosten van een IoT-platform bestaan dus enkel uit OPEX. Om inzicht te krijgen in de kostendrijvers van een IoT-platform worden hieronder de verschillende prijsmodellen van de grootste en populairste end-to-end IoT-platformen onderzocht, namelijk Google Cloud, Microsoft Azure en Amazon Web Services (AWS). Enkel de connectiviteit en communicatie wordt in rekening gebracht, met andere woorden het IoT-platform vervult enkel de functie van connectiviteit en normalisatie. Het opslaan van sensordata, verwerken en analyseren van sensordata of het ondernemen van acties als resultaat van geanalyseerde sensordata, komt niet aan bod aangezien dit volledig use case afhankelijk is. De aanvullende parameters die nodig zijn om de kost van bijkomstige IoT-platform functies te berekenen zou het model te uitgebreid maken.

2.3.1 Google Cloud IoT Core

Google Cloud IoT Core is de component van de Google Cloud Suite die instaat voor de connectiviteit en configuratie van slimme objecten. Google Cloud IoT Core maakt het mogelijk om snel en veilig een groot aantal slimme objecten aan te sluiten en te configureren. Google Cloud IoT Core ondersteunt de MQTT en HTTP connectiviteitsprotocollen. De service werkt op de Google infrastructuur die automatisch in realtime wordt geschaald en waar, indien nodig, miljoenen slimme objecten kunnen worden aangesloten.

Google Cloud IoT Core hanteert een prijsmodel gebaseerd op het datavolume dat in een kalendermaand verbruikt wordt. Datavolume is gebaseerd op gegevens die worden uitgewisseld door slimme objecten die zijn verbonden met Google Cloud IoT Core. Het minimum datavolume per uitgewisseld bericht bedraagt 1 KB, wat wil zeggen dat berichten kleiner dan 1 KB steeds gerekend worden als 1 KB, omdat 1 KB het minimale datavolume is dat voor facturering wordt gebruikt. Elke maand is er een gratis datavolume beschikbaar van 250 MB, wat dus wil zeggen dat er elke maand 256.000 berichten van telkens het minimale datavolume van 1 KB gratis zijn. Tabel 2.5 geeft een overzicht van de gehanteerde prijzen voor connectiviteit met Google Cloud IoT Core.

Tabel 2.5: Google IoT Cloud Core prijslijst.

Maandelijkse datavolume	Prijs per MB
tot 250 MB	€ 0,00
250 MB tot 250 GB	€ 0,004034249
250 GB tot 5 TB	€ 0,001792999
vanaf 5 TB	€ 0,000403424

2.3.2 Microsoft Azure IoT Hub

Microsoft Azure IoT Hub is de component van Microsoft Azure die instaat voor de connectiviteit en configuratie van slimme objecten. Microsoft Azure IoT Hub ondersteunt zowel de MQTT en HTTP connectiviteitsprotocollen, alsook het AMQP protocol.

Microsoft Azure IoT Hub hanteert een prijsmodel gebaseerd op het dagelijkse verbruik van IoT Hubs. Het dagelijkse verbruik van IoT Hubs hangt volledig af van het totale aantal uitgewisselde berichten per dag. Een bericht is gedefinieerd als 4 KB, wat betekent dat een uitgewisseld bericht van 5 KB gerekend wordt als twee berichten. Microsoft Azure IoT Hub biedt twee verschillende lagen van functionaliteit aan, namelijk de basislaag en de standaardlaag. De twee lagen zijn

verschillend in welke functies zij ondersteunen: de basislaag ondersteunt slechts een subset van de functies die de standaardlaag ondersteunt. De basislaag ondersteunt bijvoorbeeld geen downlink berichten (van de Cloud naar de slimme objecten) of complexe apparaatconfiguratie. In elke laag zijn er drie verschillende edities van IoT Hubs. De verschillende edities onderscheiden zich in het maximale totale aantal berichten per dag, aanhoudende doorvoer van berichten en verzendsnelheid van berichten. Het verbruik van IoT Hub eenheden wordt dagelijks gemeten en de facturering wordt maandelijks gegenereerd. Klanten worden gefactureerd op basis van het aantal IoT Hub eenheden dat gedurende de maand is verbruikt. Tabel 2.6 geeft een overzicht van de gehanteerde prijzen voor connectiviteit met Microsoft Azure IoT Hub.

Tabel 2.6: Microsoft Azure IoT Hub prijslijst.

Type editie	Prijs per IoT Hub per maand	Totale aantal berichten per dag
B1	€ 8,433	400.000
B2	€ 42,165	6.000.000
B3	€ 421,650	300.000.000
S1	€ 21,083	400.000
S2	€ 210,825	6.000.000
S3	€ 2.108,25	300.000.000

Microsoft Azure IoT Hub voorziet ook een gratis testeditie van de standaardlaag. De testeditie kan tot 8.000 berichten per dag verwerken van maximaal 500 slimme objecten. In de testeditie wordt, in tegenstelling tot de andere edities van de standaardlaag, een bericht gedefinieerd als 0,5 KB. De testeditie is zoals de naam al doet vermoeden voor testdoeleinden en wordt in tegenstelling tot Google IoT Cloud Core niet in vermindering gebracht bij de facturatie. Het is ook niet mogelijk om van de testeditie te opwaarderen naar een andere editie.

2.3.3 Amazon Web Services IoT Core

AWS IoT Core is de component van AWS die instaat voor de connectiviteit en configuratie van slimme objecten. AWS IoT Core ondersteunt de HTTP, WebSockets en MQTT connectiviteitstechnologieën *out-of-the-box* maar stelt de gebruiker ook in staat om eigen, ongestandaardiseerde protocollen te definiëren. Prijzen voor AWS IoT Core zijn net zoals alle andere prijzen van AWS regio-afhankelijk. Tabel 2.7 geeft de vergelijking tussen de drie EU-regios Londen, Ierland en Frankfurt.

AWS IoT Core hanteert een prijsmodel gebaseerd op vier prijscomponenten, namelijk connec-

tiviteit, berichten, apparaatschaduw en register operaties en het gebruik van de *rules engine*. Het gehanteerde prijsmodel van AWS IoT Core maakt het beduidend moeilijker om a priori de kosten te berekenen in vergelijking met Google Cloud IoT Core of Microsoft Azure IoT Hub. De kosten voor de prijscomponent connectiviteit worden gefactureerd per minuut dat een slim object verbonden is met AWS IoT Core. De minimum connectiviteitstijd per connectie is een minuut. Het prijsmodel voor de prijscomponent connectiviteit hangt af van hoeveel minuten alle slimme objecten geconnecteerd zijn met AWS IoT Core. Connectiviteit wordt uitgedrukt in een vaste prijs per miljoen minuten van connectiviteit. De prijs bedraagt in Londen en Frankfurt € 0,0860 per miljoen minuten van connectiviteit, terwijl in Ierland de prijs € 0,0716 per miljoen minuten van connectiviteit bedraagt. Het prijsmodel voor de prijscomponent berichten hangt af van hoeveel berichten er uitgewisseld worden tussen de slimme objecten en AWS IoT Core. Een bericht is gedefinieerd als 5 KB, wat betekent dat een uitgewisseld bericht van 6 KB gerekend wordt als twee berichten. In dit opzicht is AWS IoT Core iets genereuzer dan Microsoft Azure IoT Hub. De derde component, apparaatschaduw & register operaties, is volledig use case afhankelijk en dan ook zeer moeilijk om generisch in kaart te brengen. De vierde prijscomponent, het gebruik van de rules engine, wordt opnieuw niet in rekening gebracht omdat de andere besproken IoT-platformen deze kost afzonderlijk in rekening brengen.

Tabel 2.7: AWS IoT Core prijscomponent berichten prijslijst.

Maandelijks aantal berichten	Prijs per miljoen berichten (Londen en Frankfurt)	Prijs per miljoen berichten (Ierland)
tot 1 miljard berichten	€ 1,0744	€ 0,8954
volgende 4 miljard berichten	€ 0,8596	€ 0,7163
meer dan 5 miljard berichten	€ 0,7521	€ 0,6268

De prijscomponent berichten is doorgaans een factor tien groter dan de andere drie prijscomponenten. Er kan dus gesteld worden dat bij AWS IoT Core de grootste kost voor connectiviteit met het IoT-platform afhangt van het aantal uitgewisselde berichten, net zoals bij Microsoft Azure IoT Hub.

AWS IoT Core voorziet ook eenmalig een gratis proefversie. De proefversie heeft een limiet van 2.250.000 minuten connectiviteit, 500.000 berichten, 225.000 apparaatschaduw & register operaties en 250.000 acties door de rules engine. Eenmaal de limiet overschreden, worden bovenstaande kosten in rekening gebracht.

Deel III

Ontwerp en ontwikkeling van de webtool

1

Behoefteanalyse

Dit hoofdstuk bespreekt de eisen waaraan de webapplicatie moet voldoen. Dit is de verwerking van de input verkregen uit de eerste thesismeetings. Sectie 1.1 beschrijft de functionele vereisten, met andere woorden wat de webapplicatie allemaal zou moeten kunnen doen. Sectie 1.2 beschrijft de niet-functionele vereisten, met andere woorden hoe de functionele vereisten gerealiseerd worden en hoe de webapplicatie zich softwarematig gedraagt.

1.1 Functionele vereisten

De webapplicatie geeft, na invoer van de gebruiker, een betrouwbare en realistische weergave van het kostenplaatje van een IoT-systeem, gebaseerd op het kostenmodel ontwikkeld in deel 2. Gebruikers van de webapplicatie kunnen zowel technisch als niet-technisch zijn. Het resultaat moet gepresenteerd worden aan de hand van een summiere kostentabel, die de opsomming weergeeft van de verschillende kostencomponenten. Er wordt een duidelijk onderscheid gemaakt tussen CAPEX en OPEX. Het resultaat wordt ook visueel voorgesteld als kostprijs in functie van de simulatietijd op een grafiek.

De webapplicatie moet de gebruiker in staat stellen om simulaties op te slaan en op een later tijdstip weer in te laden. Na het inladen van een opgeslagen simulatie moet de gebruiker de

simulatie kunnen voortzetten en ingeladen inputparameters kunnen aanpassen. De gebruiker dient hierbij de mogelijkheid te hebben om zich te registreren en in te loggen. Na het inloggen kan de gebruiker de opgeslagen simulaties raadplegen in een overzicht. Het is belangrijk om op te merken dat indien een gebruiker zich niet geregistreerd heeft en dus ook niet ingelogd is dat hij/zij wel nog steeds de mogelijkheid heeft om de kostprijs van een IoT-systeem te simuleren.

Indien de tijd het toelaat zijn er nog extra uitbreidingen mogelijk. De voornaamste is bijvoorbeeld meerdere opgeslagen simulaties met elkaar kunnen vergelijken, waarbij duidelijk de verschillen in CAPEX en OPEX gepresenteerd worden. Ook het mogelijk maken om opgeslagen simulaties af te drukken of te e-mailen zijn mogelijke uitbreidingen.

1.2 Niet-functionele vereisten

Aangezien het IoT-landschap enorm snel evolueert staan de niet-functionele vereisten uitbreidbaarheid, flexibiliteit en onderhoudbaarheid centraal. Uitbreidbaarheid en flexibiliteit betekenen dat de webapplicatie zo moet opgebouwd zijn dat het eenvoudig is om bestaande logica aan te passen of om nieuwe logica toe te voegen. Voorbeelden hiervan zijn bijvoorbeeld het aanpassen van kostencomponenten van reeds geïmplementeerde IoT-connectiviteitstechnologieën of het toevoegen van nieuwe IoT-connectiviteitstechnologieën. De webapplicatie moet ook onderhoudbaar zijn, wat betekent dat de implementatie van de webapplicatie duidelijk gestructureerd moet zijn en dat de geïmplementeerde logica goed gedocumenteerd is. Er moet duidelijk beschreven zijn hoe de webapplicatie kan aangepast worden en hoe er nieuwe logica kan toegevoegd worden.

De webapplicatie moet naast deze primaire vereisten ook eenvoudig overdraagbaar zijn naar verschillende hardware systemen en moet uiteraard werken op de meest vertegenwoordigde webbrowsers. De webapplicatie moet gebruiksvriendelijk zijn voor niet-technische gebruikers en moet een betrouwbaar resultaat weergeven. Het verzenden van data van gebruikers zoals inloggegevens of simulatie inputparameters moet veilig en geëncrypteerd gebeuren. De webapplicatie moet grondig testbaar zijn.

2

Ontwerp

Dit hoofdstuk bespreekt het vooropgestelde ontwerp van de webapplicatie. Sectie 2.1 verwoordt de gemaakte technologiekeuzes en geeft een korte introductie tot deze gebruikte technologieën. Sectie 2.2 beschrijft de vooropgestelde architectuur van de webapplicatie.

2.1 Technologiekeuze

Aan de hand van de vooropgestelde niet-functionele vereisten uit de behoefteanalyse werd duidelijk dat de webapplicatie duidelijk gestructureerd en modulair moest zijn, met oog op uitbreidbaarheid en flexibiliteit. De populairste modulaire front-end frameworks zijn Angular, React en de nieuwkomer Vue.js. Aangezien Angular een zeer uitgebreide documentatie heeft en er al reeds enige ervaring verworven was met het front-end framework viel de eerste keuze hierop terug. Als back-end werd er geopteerd voor een JavaScript omgeving in plaats van de klassieke traditionele back-end omgevingen zoals Java, PHP of ASP.NET. Een JavaScript back-end geeft snelle ontwikkelingsmogelijkheden mits de juiste structuur gehanteerd wordt. Node.js was dan ook een evidente keuze als back-end. Als back-end middleware werd geopteerd voor Express, het populairste Node.js framework, dat toelaat om een duidelijke modulaire en gestructureerde back-end omgeving te ontwikkelen. De inherente heterogene aard van IoT vraagt duidelijk om een niet-gestructureerde databank en eerder een NoSQL, document-georiënteerde databank om

karakteristieken op te slaan. MongoDB is een populaire keuze van NoSQL-databank en is ook zeer goed gedocumenteerd. Ook de integratie van MongoDB in een JavaScript back-end framework is uitstekend zoals hieronder verder zal worden toegelicht. Als primaire technologiekeuze voor de webapplicatie werd dan ook gekozen voor de MongoDB, Express, Angular en Node.js (MEAN) stack.

2.1.1 MEAN stack

MEAN is een verzameling van gratis *open source* technologieën die samen een end-to-end framework vormen voor het implementeren van dynamische, modulaire webapplicaties. De verschillende technologieën ondersteunen implementatie in JavaScript, waardoor zowel de front-end of *clientside* als de back-end of *serverside* in dezelfde programmeertaal geïmplementeerd kunnen worden. Hieronder wordt een introductie gegeven tot de verschillende technologieën die samen de MEAN stack vormen.

Angular

Angular is samen met React een van de populairste front-end frameworks voor het implementeren van schaalbare, modulaire en dynamische *single-page* webapplicaties. Angular wordt vaak benoemd als Angular 2+ om duidelijk het onderscheid te maken met AngularJS, dat conceptueel volledig verschilt van Angular. Angular is berust op het Model-View-Controller (MVC) concept, wat inhoudt dat de presentatie volledig gescheiden wordt van de logica. Ontwikkelen van webapplicaties in Angular gebeurt in de programmeertaal TypeScript. TypeScript is een *superset* van JavaScript, dat ondermeer *static typing*, *interfaces* en *decorators* aanbiedt.

Node.js

Node.js is een JavaScript *runtime environment* dat toelaat om programma's geschreven in JavaScript uit te voeren buiten de browser om. Hierdoor wordt het mogelijk om programma's geschreven in JavaScript te beschouwen als *standalone* applicaties. Intern maakt Node.js, net zoals de webbrowser Chrome, gebruik van Google's V8 JavaScript *engine* om JavaScript te compileren naar C++ code en uitvoerbaar te maken. Node.js maakt gebruik van een *event-driven non-blocking I/O* model, dat resulteert in een enorm snelle uitvoering, dat tevens ook lichtgewicht is.

Express

Express is een lichtgewicht webapplicatie framework bovenop de Node.js runtime. In essentie is Express een *wrapper* rond de Node.js runtime dat toelaat om eenvoudiger webapplicaties te ontwikkelen in de back-end. Express stelt de ontwikkelaar in staat om een MVC-architectuur te ontwikkelen, maar dit is geheel de keuze van de ontwikkelaar. Express kan ondermeer instaan voor het afhandelen van HTTP requests en routing.

MongoDB

MongoDB is een document-georiënteerde NoSQL-databank, die gebruik maakt van schema's. Schema's worden opgesteld in Binary JavaScript Object Notation (BSON), een formaat afgeleid van JavaScript Object Notation (JSON). Het voordeel van een NoSQL-databank is dat de opgeslagen documenten van structuur kunnen veranderen, zonder ingrijpende globale veranderingen. Om vanuit de Node.js runtime een connectie te maken met MongoDB dient er gebruik gemaakt te worden van een MongoDB *driver*.

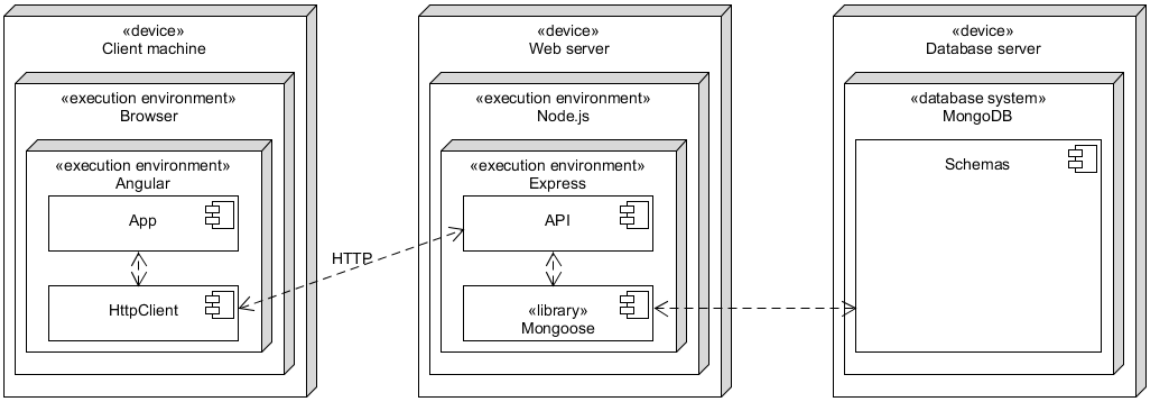
2.1.2 Docker

Om de overdraagbaarheid van de webapplicatie te vereenvoudigen werd er geopteerd om de verschillende componenten te virtualiseren met behulp van Docker. Docker maakt het mogelijk om de webapplicatie eenvoudig te deployen, uit te voeren en te testen op verschillende hardware systemen. Ook meer en meer hostingproviders voorzien de mogelijkheid om gevirtualiseerde applicaties out-of-the-box te deployen.

2.2 Systeemarchitectuur

Een MEAN webapplicatie is duidelijk opgedeeld in een front-end en een back-end, de back-end is verder opgedeeld in een *business logic* laag en een datalaag. De front-end wordt verzorgd door het Angular framework, de back-end door de Node.js runtime, het Express framework en de MongoDB databank. Figuur 2.1 geeft een *high-level* overzicht van de webapplicatie, ook wel bekend als het *deployment* diagram. De communicatie tussen de front-end en de back-end wordt verzorgd door HTTP REST API *calls*. Sinds Angular 4.3.x is de HttpClient interface van de HttpClientModule beschikbaar, dat toelaat op een eenvoudige manier HTTP calls uit te voeren. In de back-end wordt de connectiviteit en communicatie met de databank verzorgd door de Mongoose bibliotheek. Mongoose is een populaire MongoDB driver, die onder andere instaat

voor vereenvoudiging van de connectiviteit en communicatie met de MongoDB databank en validatie van data.

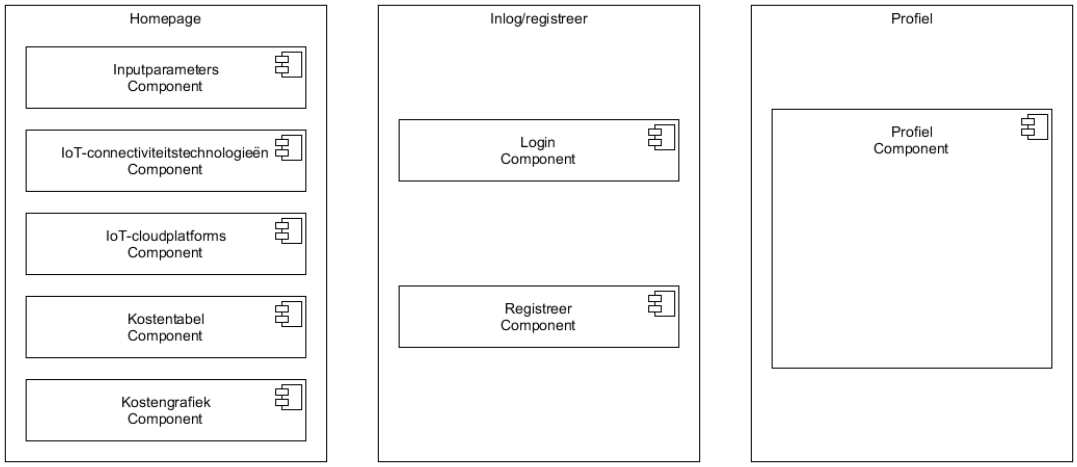


Figuur 2.1: Deployment diagram van de webapplicatie.

2.2.1 Front-end

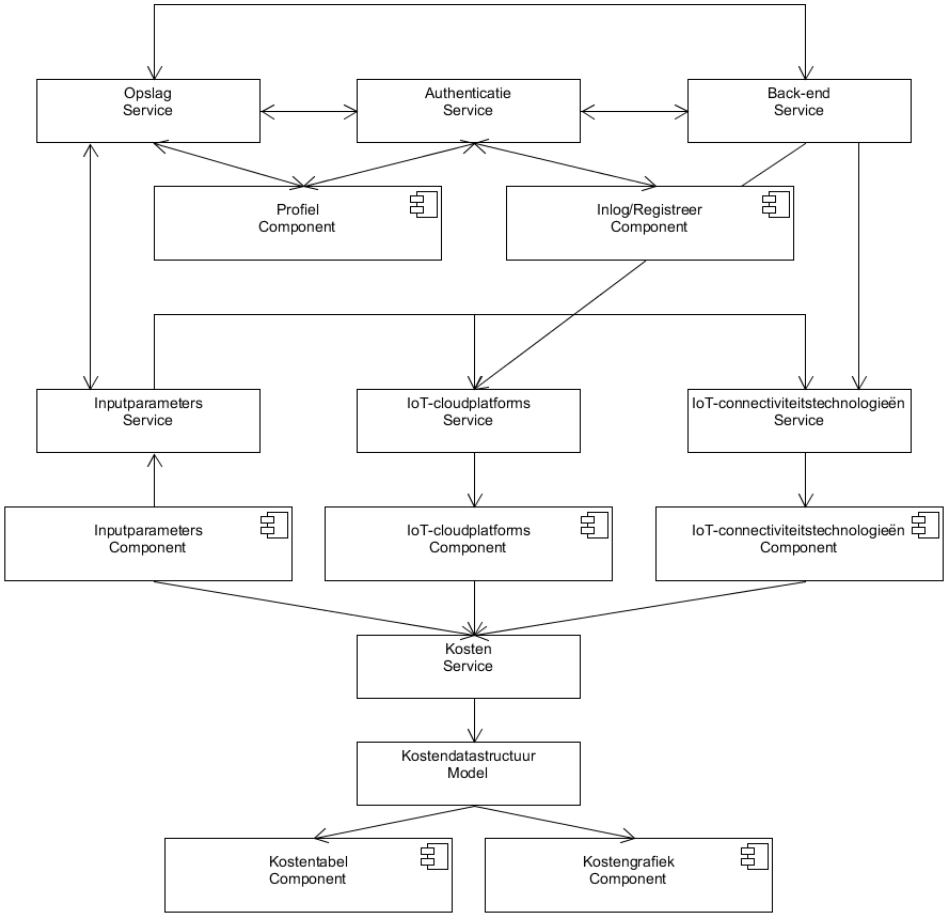
De front-end wordt opgesplitst in verschillende Angular Componenten zoals weergegeven in figuur 2.2. Het klassendiagram van de front-end, dat ondermeer de relaties tussen de verschillende Componenten en de communicatieflow weergeeft, wordt weergegeven in figuur 2.3.

Allereerst worden de IoT-connectiviteitstechnologieën en IoT-cloudplatformen ingeladen vanuit de back-end via de Back-end Service. Deze worden weergegeven in respectievelijk de IoT-connectiviteitstechnologieën Component en de IoT-cloudplatformen Component. Indien de gebruiker invoer ingeeft in de Inputparameters Component wordt dit via Services gecommuniceerd naar de andere betrokken Componenten, ondermeer naar de IoT-connectiviteitstechnologieën Service om IoT-connectiviteitstechnologieën te filteren en naar de kosten Service om kostcomponenten te berekenen. De Kosten Service maakt gebruik van een zelf gedefinieerde datastructuur om gecalculeerde kosten op te slaan en weer te geven in zowel de Kostentabel Component als de Kostengrafiek Component. Authenticatie en autorisatie wordt verzorgd door de Authenticatie Service. De gebruiker kan inloggen, of indien de gebruiker nog niet geregistreerd is, registreren via de Inlog/registreer Component. De gebruiker kan de opgeslagen inloggegevens raadplegen via de Profiel Component, die enkel bereikbaar is indien de gebruiker geauthenticeerd is. Het opslaan en inladen van simulaties gebeurt door de Opslag Service, die opnieuw enkel bruikbaar is indien de gebruiker geauthenticeerd is. Alle verkeer naar de API in de back-end verloopt via de centrale Back-end Service.



(a) Homepage. (b) Login/registreer. (c) Profiel.

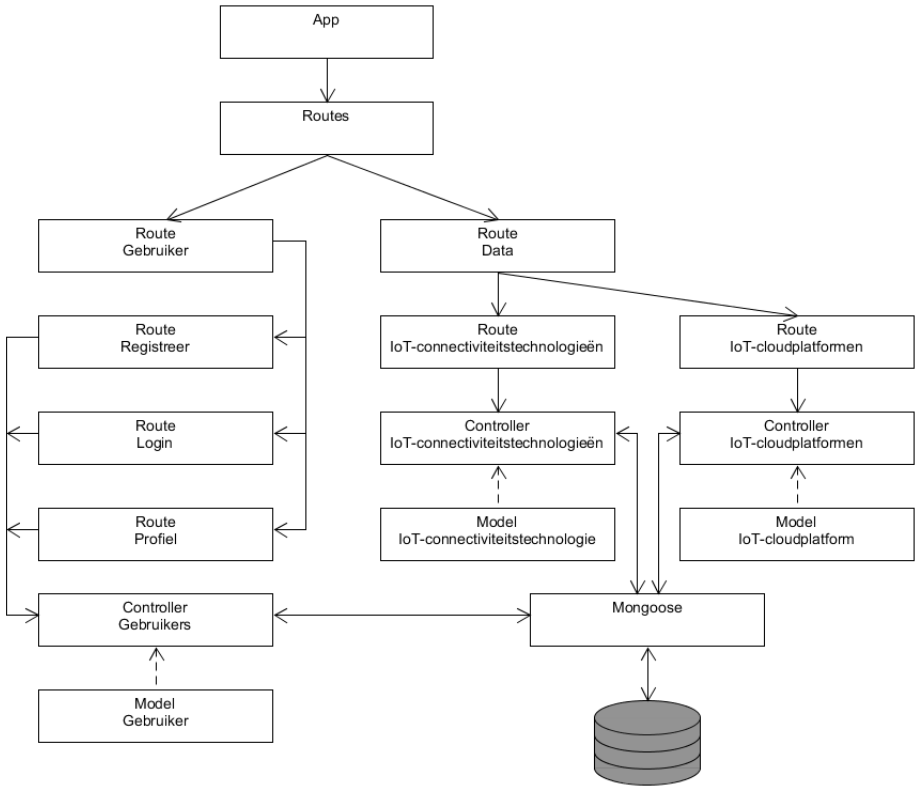
Figuur 2.2: Front-end webapplicatie opsplitsing in Angular Componenten.



Figuur 2.3: Front-end klassendiagram.

2.2.2 Back-end

Het Express framework biedt de mogelijkheid om net als in de front-end een MVC-architectuur op te bouwen in de back-end. Concreet betekent dit dat de back-end API publieke Uniform Resource Locators (URLs) open stelt aan de front-end en dat Controllers de gevraagde inhoud verschaffen. De Controllers staan ook in voor het aanvragen of wijzigen van de inhoud van de databank. De communicatie en connectiviteit met de databank verloopt via Mongoose. Een duidelijke MVC-architectuur zorgt er dus voor dat enkel de Controllers in verbinding staan met de datalaag. Aangezien in de webapplicatie de front-end zorgt voor de presentatie is er in de back-end uiteraard geen *View* component.



Figuur 2.4: Back-end klassendiagram.

3

Realisatie

Dit hoofdstuk bespreekt de implementatie van de belangrijkste kernfuncties van de webapplicatie, en meer specifiek hoe het vooropgestelde ontwerp in de praktijk geïmplementeerd werd. Ondervonden moeilijkheden tijdens de implementatie of tekortkomingen van het ontwerp worden toegelicht en gemaakte implementatiekeuzes worden bediscussieerd.

3.1 Realisatie van de belangrijkste kernfuncties

3.1.1 Angular component interactie

Idealiter staan Angular Componenten enkel in voor het presenteren van data en het reageren op gebruikersinteracties. Het opslaan van data gebeurt in Angular Services. Indien verschillende componenten met elkaar willen interageren om bijvoorbeeld data uit te wisselen of data synchroon te houden, zijn er twee verschillende mogelijkheden, afhankelijk van de relaties tussen de beide Componenten. Indien beide Componenten een ouder-kind relatie hebben, is het mogelijk om data te delen via Angular *input binding* aan de hand van de *@Input* en *@Output* decorators. Indien er geen ouder-kind relatie tussen beide Componenten bestaat, dient er gebruik gemaakt te worden van een gedeelde Angular Service, die beide Componenten injecteren.

Tijdens het ontwerp van de webapplicatie werd er voorop gesteld dat Component interactie zou gebeuren aan de hand van gedeelde Services, desondanks er toch ouder-kind relaties zijn. Dit opdat de verschillende interagerende Componenten zoveel mogelijk *loosely coupled* zouden zijn. Er werd echter niet duidelijk gespecificeerd hoe de communicatie via een gedeelde Service zou gebeuren. Dit leverde al snel tijdens de realisatie onnauwkeurig- en onduidelijkheden op die idealiter in de ontwerpfase geadresseerd hadden moeten zijn.

Tijdens de realisatie is er geopteerd om gebruik te maken van *Reactive programming* aan de hand van de Reactive Extensions for JavaScript (RxJS) bibliotheek, een relatief nieuwe JavaScript bibliotheek aanwezig in Angular Core. RxJS is een uitgebreide, krachtige bibliotheek die asynchrone communicatie tussen afhankelijke modules loosely coupled maakt, aan de hand van het *publish/subscribe* model. Reactive programming is eerder een *mindset* verschuiving dan een bibliotheek, het is dan ook niet verwonderlijk dat Reactive programming een arsenaal aan nieuwe concepten en tools introduceert. Hieronder wordt kort geschetst hoe Reactive programming aan de hand van de RxJS bibliotheek in de webapplicatie werd verwezenlijkt.

In de Reactive programming mindset is er steeds een bron van informatie, namelijk de *source* en een of meerdere verbruikers van informatie, namelijk de *sinks*. De source wordt in RxJS voorgesteld door een object van de klasse *Subject*, of een van de afgeleide klassen van *Subject*. Informatie kan aan de source worden toegevoegd via de *next* methode van *Subject*. Een *Subject* object kan voorgesteld worden als een *Observable* object aan de hand van de methode *asObservable* van *Subject*, *Observable* variabelen worden typisch aangeduid met een $-teken op het einde van de naam, maar dit is eerder een conventie en geen vereiste. Het *Observable* object laat verschillende sinks toe om asynchroon de informatie aanwezig in de source te raadplegen. Dit is de kern van Reactive programming. Er is geen één op één verband tussen source en sinks. Listing 3.1 geeft een vereenvoudigde implementatie van een Angular Service die gebruik maakt van de RxJS bibliotheek.

```
import { Injectable } from '@angular/core';
import { BehaviorSubject } from 'rxjs';

@Injectable()
export class CommunicationService {
  private messageSource: BehaviorSubject<string>;
  public message$: Observable<string>;

  constructor() {
    this.messageSource = new BehaviorSubject<string>("initial message");
    this.message$ = this.messageSource.asObservable();
```

```
  }

  public changeMessage(message: string): void {
    this.messageSource.next(message);
  }
}
```

Codefragment 3.1: Angular Service: Component interactie met RxJS.

Listing 3.2 geeft een vereenvoudigde implementatie van een Angular Component die gebruik maakt van de gedefinieerde Angular Service om data te delen en te ontvangen. Sinks kunnen dynamisch abonneren op *Observable* objecten van sources wanneer ze interesse hebben in de opgeslagen informatie en ook weer uitschrijven als ze niet langer geïnteresseerd zijn. De *lifecycle hooks* van Angular Componenten komen hierbij in aanmerking om te abonneren en uit te schrijven op *Observable* objecten.

```
import { Component, OnInit } from '@angular/core';
import { Subscription } from 'rxjs';
import { CommunicationService } from "./communication.service";

@Component({...})
export class CommunicationComponent implements OnInit, OnDestroy {
  private message: string;
  private subscription: Subscription;

  constructor(private communicationStream: CommunicationService) {}

  private newMessage(message: string): void {
      this.communicationStream.changeMessage(message);
  }

  ngOnInit() {
    this.subscription = this.communicationStream.message$
      .subscribe(message => this.message = message);
  }

  ngOnDestroy() {
    this.subscription.unsubscribe();
```

```
  }
}
```

Codefragment 3.2: Angular Component: Component interactie met RxJS.

3.1.2 Kostenberekening

Kostenberekeningen gebeuren onafhankelijk van Angular Componenten in een Angular Service. Interacties tussen de verschillende Componenten en Services gebeuren aan de hand van de RxJS bibliotheek zoals beschreven in subsectie 3.1.1. Gecalculeerde kosten worden opgeslagen in een zelf gedefinieerde datastructuur.

Tijdens het ontwerp van de webapplicatie werd vooropgesteld dat de gecalculeerde kosten opgeslagen zouden worden in een zelf gedefinieerde datastructuur. Zo zou het eenvoudig zijn om het kostenplaatje in kaart te brengen in verschillende Componenten, onafhankelijk van elkaar, met het Don't Repeat Yourself (DRY) principe in gedachten. Opnieuw werd er echter niet gedetailleerd gespecificeerd hoe de gecalculeerde kosten zouden opgeslagen worden, of hoe deze opgeslagen data kon gewijzigd of geraadpleegd worden.

Tijdens de realisatie is ervoor gekozen om de gecalculeerde kosten op te slaan als *key/value* paren in een JavaScript *Map*. De datastructuur is niet verantwoordelijk voor het berekenen van de kosten, dit is de verantwoordelijkheid van een externe Service, die de datastructuur zal aanspreken. De keys van de *Map* zijn *strings* en beschrijven over welke kostcomponent het gaat, bijvoorbeeld "connectivitySubscriptionCost"of "smartDeviceHardwareCost". De bijbehorende values zijn *arrays* van getallen, die de kosten weergeven. De arrays zijn dan ook zo lang als de gesimuleerde levensduur van de applicatie plus één om de kosten in het jaar nul aan te kunnen duiden.

Het lijkt heel triviaal maar enkel kosten die effectief van toepassing zijn op de simulatie worden opgeslagen in de datastructuur. Kostcomponenten die niet van toepassing zijn op de simulatie dienen als gevolg niet aangeduid te worden als *onbestaand* of *leeg* omdat ze niet bestaan. Een andere mogelijkheid was om alle mogelijke kosten op te slaan, ook al zijn ze niet van toepassing op de simulatie. Er werd geopteerd voor de eerste keuze om op deze onnodige *overhead* te besparen. Ook het totaal van sommaties zoals CAPEX, OPEX, totale en cumulatieve kosten worden in de datastructuur opgeslagen. Strikt gezien is dit niet nodig en kunnen deze sommaties berekend worden op de aanwezige kosten in de datastructuur. Vanwege performantieredenen werd echter gekozen om deze sommaties ook op te slaan. Listing 3.3 geeft de voorstelling van een mogelijke inhoud van de datastructuur voor een simulatie van vijf jaar.

```
[
```

```
 { "connectivitySubscriptionCost": [0,10000,20000,20000,20000,20000] },
 { "smartDeviceHardwareCost": [0,8000,16000,0,0,0] },
 { "CAPEXCost": [0,8000,16000,0,0,0] },
 { "OPEXCost": [0,10000,20000,20000,20000,20000] },
 { "totalCost": [0,18000,36000,20000,20000,20000] },
 { "cumulativeCost": [0,18000,54000,74000,94000,114000] }
]
```

Codefragment 3.3: Mogelijke inhoud van de datastructuur voor gecalculeerde simulatiekosten.

3.1.3 HTTP API calls

Zoals reeds vermeld in hoofdstuk 2, sectie 2.2 wordt de communicatie tussen de front-end en de back-end verzorgd door HTTP REST API calls met behulp van de HttpClient interface in Angular. Listing 3.4 geeft een vereenvoudigde implementatie van een Angular Service die gebruik maakt van de HttpClient interface om HTTP REST API calls uit te voeren. De teruggeefwaarde van de HTTP calls zijn *Observable* objecten, omdat HTTP calls asynchroon gebeuren. Componenten die de verzonden en/of verkregen data willen raadplegen dienen te abonneren op de gedeclareerde methoden, zoals beschreven in subsectie 3.1.1.

```
import { Injectable } from '@angular/core';
import { HttpClient } from '@angular/common/http';
import { Observable } from 'rxjs';

@Injectable()
export class BackendService {
  private backendAPIURL: string = "...";

  constructor(private http: HttpClient) {}

  public getHTTPCall(): Observable<any> {
      return this.http.get<any>(this.backendAPIURL);
  }

  public postHTTPCall(message: any): Observable<any> {
      return this.http.post<any>(this.backendAPIURL, message);
  }
}
```

Codefragment 3.4: Angular Service: HTTP API calls met de HttpClient interface.

Listing 3.5 geeft een vereenvoudigde implementatie van een RESTful API gebouwd op Node.js en Express. Dit is een voorbeeld van een eindpunt van API calls naar de back-end. De afhandeling van GET en POST requests worden gedelegeerd naar een *Controller* die antwoord zal verschaffen op de aanvragen. Concrete invulling van de controller wordt besproken in subsectie 3.1.5.

```
const express = require('express');
const router = express.Router();

const userController = require('./user-controller');

router.post('/login', userController.login);
router.post('/register', userController.register);

module.exports = router;
```

Codefragment 3.5: Node.js back-end RESTful API.

3.1.4 Aanmaken van gebruikers

Het aanmaken van gebruikers wordt verzorgd door een POST request te doen naar de back-end API route /register met als parameters het e-mailadres en het wachtwoord. De Gebruiker Controller handelt het request af en staat in voor het aanmaken van een nieuwe Gebruiker. De Controller doet hiervoor een beroep op de Mongoose bibliotheek, wat de connectiviteit en communicatie met de MongoDB databank erg vereenvoudigt. Listing 3.6 geeft een vereenvoudigde implementatie van het Gebruiker model. Merk op dat naast het e-mailadres, het wachtwoord niet wordt opgeslagen in de databank. Het wachtwoord wordt aan de hand van de methode *setPassword* gehashed met behulp van de builtin Node.js Crypto bibliotheek. De hash wordt samen met de gebruikte salt opgeslagen in de databank. Indien later de gebruiker opnieuw wil inloggen wordt het doorgezonden wachtwoord opnieuw gehashed in de methode *validPassword* en vergeleken met de opgeslagen hash. Indien beide overeenkomen is het doorgezonden wachtwoord correct en kan de gebruiker inloggen.

```
const mongoose = require('mongoose');
const crypto = require('crypto');
```

```
const userSchema = new mongoose.Schema({
  email: {
    type: String,
    unique: true,
    required: true
  },
  hash: String,
  salt: String
});

userSchema.methods.setPassword = function (password) {
  this.salt = crypto.randomBytes(16).toString('hex');
  this.hash = crypto.pbkdf2Sync(password, this.salt, 1000, 64, 'sha512')
    .toString('hex');
};

userSchema.methods.validPassword = function (password) {
  var hash = crypto.pbkdf2Sync(password, this.salt, 1000, 64, 'sha512')
    .toString('hex');
  return this.hash === hash;
};

module.exports = mongoose.model('User', userSchema);
```

Codefragment 3.6: Gebruiker model met behulp van de Mongoose bibliotheek.

3.1.5 Authenticatie en autorisatie

Authenticatie en autorisatie worden gebaseerd op de uitwisseling van een JSON Web Token (JWT), uitgesproken als *jot*, tussen de front-end en de back-end. Elke HTTP call bevat het JWT waardoor de back-end de gebruiker kan identificeren. JWTs authenticeren geregistreerde gebruikers aan de hand van een unieke code. Een JWT bestaat uit drie delen, namelijk een *header*, *payload* en *signature*, gescheiden door komma's.

Authenticatie en autorisatie is iets wat typisch gebeurt in de back-end van een webapplicatie. De back-end doet een beroep op de bibliotheek Passport.js, een eenvoudige bibliotheek om gebruikers te authenticeren en te autoriseren. Met Passport.js is het mogelijk om gebruikers te

authenticeren aan de hand van verschillende strategieën, bijvoorbeeld via Facebook, Twitter of Google. Hier is geopteerd om gebruik te maken van de lokale strategie, wat betekent dat gebruikers zich authenticeren aan de hand van een uniek e-mailadres en een zelfgekozen wachtwoord. Listing 3.7 geeft een vereenvoudigde implementatie van de Gebruikers Controller in de back-end. Indien de *callback* functie een error bevat wordt een *500 Internal Server Error* HTTP response teruggegeven naar de front-end. Indien er een gebruiker gevonden wordt, wordt het gegenereerde JWT teruggegeven naar de front-end in de body van een *200 OK* HTTP response. Indien er geen error wordt opgegooid en er ook geen gebruiker gevonden wordt dan bestaat de gebruiker niet of zijn de inloggegevens incorrect. De HTTP response is dan een *401 Unauthorized*.

```
const passport = require('passport');
const User = require('./User');

module.exports.login = (req, res) => {
  passport.authenticate('local', (err, user, info) => {
    let JWT;
    if (err) {
      res.status(500).json(err);
      return;
    }
    if (user) {
      JWT = user.generateJwt();
      res.status(200).json({
        "JWT": JWT
      });
    } else {
      res.status(401).json(info);
    }
  })(req, res);
};
```

Codefragment 3.7: Authenticatie en autorisatie in de back-end met behulp van Passport.js.

De front-end dient zich enkel te bekommeren over het opslaan van het gegenereerde JWT, wat als antwoord wordt teruggestuurd op een succesvol login request. Het uitloggen van een persoon is eenvoudigweg het opgeslagen JWT verwijderen, deze actie hoeft dus zeker niet gecommuniceerd te worden naar de back-end. Een JWT opslaan kan gebeuren in de *localstorage* of *sessionstorage* van de browser. Hier is geopteerd voor de localstorage, zodat als een gebruiker de browser sluit, deze zich niet opnieuw moet aanmelden. Dit houdt echter wel in dat de gebruiker ingelogd blijft

zolang de geldigheidsduur van het JWT niet verstreken is. Er is gekozen voor een geldigheidsduur van het JWT van zeven dagen.

```
import { Injectable } from '@angular/core';

@Injectable()
export class AuthenticationService {
  private backendAPIURL: string = "...";
  private JWT: string;
  private localStorageKeyName: string = "...";
  private loginCredentials: any = {email: "...", password: "..."};

  constructor(private http: HttpClient) {}

  private saveJWT(JWT: string): void {
    window.localStorage.setItem(this.localStorageKeyName, JWT);
    this.JWT = JWT;
  }

  public login(): void {
    this.http.post<any>("${this.backendAPIURL}/login", this.loginCredentials)
      .pipe((JWT) => {this.saveJWT(JWT)});
  }

  public logout(): void {
    this.JWT = '';
    window.localStorage.removeItem(this.localStorageKeyName);
  }

  public isLoggedIn(): boolean {
      if(this.JWT) {
          return true;
      } else {
          return false;
      }
  }

}
```

Codefragment 3.8: Authenticatie aan de hand van een JWT.

Indien een gebruiker niet ingelogd is, heeft hij/zij minder rechten dan ingelogde gebruikers. Zo is het bijvoorbeeld niet mogelijk om naar de profielpagina te navigeren. Deze restrictie kan in Angular in de front-end geïmplementeerd worden aan de hand van een zogenaamde *route guard*. Een route guard is een Angular Service dat de *canActivate* methode van de*CanActivate* interface implementeert. De methode heeft als teruggeefwaarde een boolean, dat *true* is als de navigatie mag doorgaan.

```
import { Injectable } from '@angular/core';
import { Router, CanActivate } from '@angular/router';
import { AuthenticationService } from './authentication.service';

@Injectable()
export class AuthGuardService implements CanActivate {

  constructor(private auth: AuthenticationService, private router: Router) {}

  canActivate(): boolean {
    if (!this.auth.isLoggedIn()) {
      this.router.navigateByUrl('/');
      return false;
    }
    return true;
  }
}
```

Codefragment 3.9: Angular CanActivate interface implementatie.

3.1.6 Opslaan en laden van simulaties

Simulaties worden opgeslagen door de inputparameters van de formulieren te capteren en te converteren naar een string. De string wordt naar de back-end verzonden via een HTTP POST bericht, waar de string in zijn geheel wordt opgeslagen in de databank bij de correcte ingelogde gebruiker. Eenmaal een gebruiker een simulatie opnieuw wil inladen selecteert hij de gewenste simulatie waardoor de methode *loadSimulation* wordt opgeroepen. De methode maakt gebruik

van een relatief nieuwe feature van Angular waardoor het mogelijk wordt om arbitraire data bij navigatie mee te geven zoals wordt aangegeven in listing 3.10. Angular navigeert naar de homepage en geeft de string van inputparameters mee als attribuut van het *state* object.

```
import { Component } from '@angular/core';
import { Router } from '@angular/router';

@Component({...})
export class ProfileComponent {
  private userDetails: UserDetails;

  constructor(private router: Router) { }

  public loadSimulation(this.userDetails.simulation: string): void {
    this.router.navigateByUrl('/',
      { state: { simulation: this.userDetails.simulation } }
    );
  }
}
```

Codefragment 3.10: Data doorgeven bij Angular navigatie.

Eenmaal de navigatie compleet is wordt de homepage opnieuw opgebouwd en worden de Componenten opnieuw geconstrueerd. Bij het construeren van de *FormComponent* bevat het *history.state* object nu de doorgegeven data. De inputparameters van het formulier kunnen nu eenvoudig opnieuw ingevuld worden aan de hand van de *patchValue* methode van het formulier. De methode *patchValue* vult de inputparameters in, wel moet uiteraard de doorgegeven string nog geconverteerd worden naar een object met behulp van *JSON.parse*.

```
import { Component, OnInit } from '@angular/core';
import { FormGroup } from '@angular/forms';

@Component({...})
export class FormComponent implements OnInit {
  private form: FormGroup;

  constructor(){}
```

```
  ngOnInit() {
    if(history.state.simulation) {
      this.form.patchValue(JSON.parse(history.state.simulation));
    }
  }
}
```

Codefragment 3.11: Doorgegeven data bij Angular navigatie consumeren.

4

Reflectie

Dit hoofdstuk geeft een algemene reflectie over het ontwerp en de realisatie van de webapplicatie. Meer specifiek, het ontwerp en de realisatie van de webapplicatie wordt vanuit een kritisch oogpunt bekeken. Daarnaast worden enkele belangrijke aandachtspunten voor de toekomst verwoord.

4.1 Ontwerp

Idealiter zijn alle ontwerpbeslissingen en implementatiekeuzes vastgelegd tijdens het ontwerp van een softwareproject. Het ontwerp van deze webapplicatie legde echter enkel in grote lijnen het klassendiagram vast, waardoor er nooit een finale klassendefinitie was: er was geen duidelijkheid over de klassenmethoden of klassenvariabelen. Het ontwerp van een softwareproject mag niet langer gezien worden als de vervelende fase van programmeren, maar als een noodzakelijke fase van elk softwareproject, groot of klein. Het is tijdens het ontwerp van software dat een goed resultaat gevormd wordt en niet tijdens de realisatie ervan. Het ontwerp moet bijgevolg de noodzakelijke aandacht krijgen dat het verdient. Extreme maatregelen kunnen getroffen worden opdat eenieder niet dezelfde fout opnieuw zou blijven maken. Test Driven Development (TDD) vereist dat voor de start van de realisatie van een softwareproject het volledige ontwerp op punt staat. Bijkomstig zijn voor elk deel van het ontwerp testen uitgeschreven. Indien eenieder slaagt

op alle testen heeft men een werkend, volledig conform het ontwerp, werkend geheel.

4.2 Realisatie

Het ontwerp van de webapplicatie stond nog niet volledig op punt, waardoor er geen goede fundering was om te starten met de realisatie. Daarnaast, is er voor de start van de realisatie van de webapplicatie te weinig studie verricht over de gebruikte technologieën. Dit uitte zich ondermeer in het programmeren van code, die op een later tijdstip herschreven zou moeten worden aangezien de implementatie niet correct of onnauwkeurig was. Dit leverde niet alleen een initiële vertraging op omdat de implementatie niet correct functioneerde, het resultaat van een verkeerde werkwijze maakte de implementatie bovendien onnodig complex.

4.3 Testen

Aangezien zowel tijdens het ontwerp als de realisatie de klassendefinities, inclusief de klassenmethoden en klassenvariabelen, niet finaal waren, werden dan ook geen tussentijdse testen verricht. De testen werden opgespaard voor een later tijdstip, eens de klassendefinities finaal waren. Jammer genoeg is dit punt er nooit gekomen waardoor er dan ook geen unit testen of integratie testen zijn verricht of gedocumenteerd. De webapplicatie werd bijgevolg enkel manueel getest door de opeenvolgende features een voor een te overlopen. De vermelding van TDD is ook hier op zijn plaats.

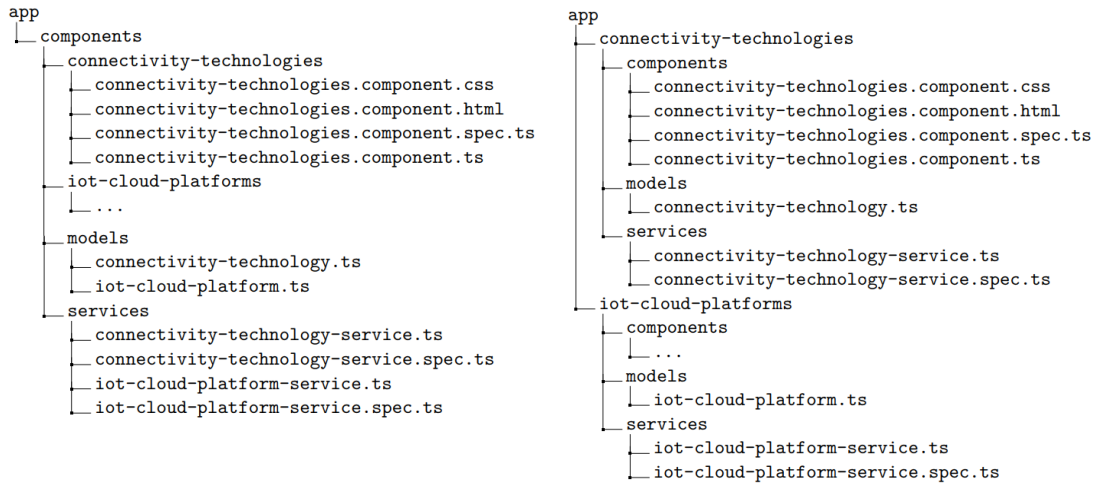
4.4 Aandachtspunten voor de toekomst

Deze sectie behandelt enkele van de getrokken lessen tijdens de praktische realisatie van de webapplicatie. Idealiter worden deze aandachtspunten in de toekomst in het achterhoofd gehouden.

4.4.1 Map- en bestandsstructuur

Tijdens het ontwerp en de realisatie van de webapplicatie werd er geen aandacht besteed aan de map- en bestandsstructuur van zowel de front-end als de back-end. Tijdens de realisatie werd al snel duidelijk dat de vele verschillende Componenten en Services in de front-end onduidelijk gestructureerd waren. Er werd geopteerd om de verschillende bestanden van de front-end te structuren naar type, zo is er bijvoorbeeld een map *components* voor Componenten, een map

models voor klassedefinities en een map *services* voor Services zoals afgebeeld in figuur 4.1a. Deze structuur wordt al snel onoverzichtelijk indien er meer Componenten worden toegevoegd. Een betere map- en bestandsstructuur zou een indeling per functie geweest zijn, zoals afgebeeld in figuur 4.1b. Deze vorm van map- en bestandsstructuur is schaalbaarder en steeds overzichtelijk. Componenten en Services die in meerdere functies gebruikt worden kunnen bijvoorbeeld in een map *shared* terechtkomen.



(a) Indeling volgens type.

(b) Indeling volgens functie.

Figuur 4.1: Map- en bestandsstructuur van de front-end.

4.4.2 JavaScript in Angular vs JavaScript in Node.js

Zowel Angular als Node.js gebruiken JavaScript als programmeertaal. Angular gebruikt de superset Typescript, wat in essentie gecompileerd wordt naar JavaScript. Doorheen de documentatie van Angular en Node.js treden echter steeds dezelfde syntaxverschillen op. Zo gebruikt Node.js nog steeds het verouderde *require* keyword in plaats van het *import* statement en zijn *arrow functions* nauwelijks vertegenwoordigd in Node.js, terwijl in Angular er net hevig gebruik van wordt gemaakt. Dit komt omdat Angular en Node.js gebruik maken van een verschillende ECMAScript versie. ECMAScript standaardiseert de JavaScript programmeertaal, met andere woorden JavaScript is een implementatie van een ECMAScript versie. De correcte verwoording is dus eigenlijk dat zowel Angular als Node.js ECMAScript gebruiken als programmeertaal maar jammer genoeg niet altijd dezelfde versie. Het is dus zinloos om consistentie proberen aan te brengen tussen de twee.

4.4.3 Localhost of 127.0.0.1

In computernetwerken is localhost de hostnaam van de eigen computer. Localhost wordt gebruikt voor toegang tot de netwerkdiensten die op de eigen computer worden uitgevoerd via de loopback netwerkinterface. Indien bijvoorbeeld een webapplicatie lokaal wordt opgestart op poort 80, de standaard poort voor HTTP, kan de webapplicatie bezocht worden door te surfen naar http://localhost. Achter de schermen wordt er gesurft naar het IPv4-adres 127.0.0.1 of het IPv6-adres ::1. Enige voorzichtigheid is echter aangewezen indien de webapplicatie in een later stadium gevirtualiseerd wordt. Indien de webapplicatie gevirtualiseerd wordt met bijvoorbeeld Docker, duidt localhost niet meer de eigen computer aan maar de container waarin de webapplicatie uitgevoerd wordt. Mits hier geen aandacht aan besteed wordt, zorgt dit voor verrassende resultaten. API-calls tussen de front-end en de back-end bijvoorbeeld, zullen nooit meer succesvol zijn, aangezien de localhost van beide containers een ander toestel is. Dit probleem is eenvoudig op te lossen door de webapplicatie in zowel de front-end als de back-end niet expliciet te laten verwijzen naar localhost maar naar een omgevingsvariabele. Indien de webapplicatie lokaal wordt geïnstalleerd en uitgevoerd bevat de omgevingsvariabele alsnog het IP-adres van de localhost (127.0.0.1). Indien de webapplicatie gevirtualiseerd wordt, wordt in beide containers de omgevingsvariabele statisch ingesteld op een gekend IP-adres, waardoor de communicatie verzekerd wordt, buiten de containers om.

Deel IV

Validatie van de webtool aan de hand van een use case

1

Koudeketen monitoring

Deze casestudie behandelt in detail de monitoring van de koudeketen van verse of diepgevroren voedingsproducten en meer specifiek de schakel die instaat voor de opslag van verse of diepgevroren voedingsproducten in koelcellen.

1.1 Achtergrond

De koudeketen is het doorlopende systeem van koeling bij de supply chain van ondermeer verse of diepgevroren voedings- en farmaceutische producten vanaf de producent, via transporteurs, groothandels en winkels tot de consument. In de supply chain van dergelijke producten is het belangrijk en wettelijk verplicht om de temperatuur in de verschillende schakels van de koudeketen te registreren. In België staat het Federaal Agentschap voor de veiligheid van de voedselketen (FAVV) in voor de controle van de veiligheid van de voedselketen en de kwaliteit van het voedsel.

Het monitoren van de temperatuur in koelcellen is van toepassing op alle winkels die gekoelde producten behandelen of verkopen zoals bakkerijen, beenhouwerijen, tankstations, broodjeszaken, ... De monitoring van de temperatuur in koelcellen gebeurt vaak nog manueel en niet geautomatiseerd. Hierdoor kan, als er zich een probleem voordoet, vaak niet tijdig ingegrepen worden. Een oplossing waarbij de temperatuur automatisch en draadloos wordt gemonitord is

dan ook zeker bruikbaar. Het toepassen van een automatische oplossing heeft ook het potentieel om economisch rendabel te zijn door de uitbesparing van arbeid, alsook door vroegtijdig te melden dat de temperatuur boven een bepaalde grenswaarde komt waardoor verderf van producten kan vermeden worden. Een bijkomend element is dat door de automatische monitoring van de temperatuur in koelcellen de efficiëntie van verschillende koelcellen kan onderzocht worden, wat een optimalisatie van de koudeketen tot gevolg kan hebben.

1.2 Eigenschappen en vereisten

De use case heeft volgende eigenschappen:

- Het aantal slimme objecten is in de grootte orde van enkele tientallen, namelijk een per koelcel.
- De connectiviteit beperkt zich steeds tot lokale connectiviteit. De use case behandelt dus niet de monitoring van koelcellen bij grootschalige voedselproducenten of winkelketens met meerdere vestigingen op verschillende locaties. De use case kan eveneens wel hier van toepassing zijn indien elke vestiging de koelcellen lokaal, afzonderlijk monitort.
- De slimme objecten wisselen gemeten temperatuurwaarden uit.
- De monitoring van de temperatuur in koelcellen vereist geen real-time berichtgeving. Frequente updates zoals twee- of viermaal per uur volstaan.
- De slimme objecten komen niet in aanraking met de verse of diepgevroren voedingsproducten, er dienen dus geen extra maatregelen getroffen te worden om de casing van de slimme objecten voedselveilig te maken.
- De slimme objecten worden niet geïntegreerd in de koelcellen en kunnen dus ook geen actuator functie vervullen, downlink communicatie is dus niet noodzakelijk.
- De mogelijkheid om te beschikken over over-the-air firmware updates is niet noodzakelijk.
- De communicatie is het liefst geëncrypteerd, maar dit is geen noodzakelijke vereiste.

De IoT-oplossing dient te voldoen aan volgende vereisten:

- De slimme objecten dienen te werken in temperaturen onder het vriespunt.
- De monitoring van de temperatuur dient een nauwkeurigheid te hebben in de grootte orde van 0,1 °C.

- De gewenste levensduur van het IoT-systeem en in het bijzonder de batterijlevensduur van de slimme objecten is het liefst zo lang mogelijk.

1.3 Simulaties

Tabel 1.1 geeft een overzicht van de algemene inputparameters van deze use case in de webtool. Bij de simulatie wordt de afweging gemaakt tussen de verschillende LPWAN-connectiviteitstechnologieën en IoT-platformen. De output van de verschillende simulaties, uitgevoerd in de webtool, is weergegeven in bijlage C.

Tabel 1.1: Algemene inputparameters voor de verschillende simulaties.

Inputparameter	Eenheid	waarde
Aantal slimme objecten	/	10
Gesimuleerde levensduur	[jaar]	10
Gefaseerde uitrol	/	nee
Maken, kopen of huren	/	kopen
Kostprijs	[€/slim object]	40
Deployment kost	[€/slim object]	0
Eindige energiebron	/	ja
Batterijlevensduur	[jaar]	7
Batterijvervangingskost	[€/slim object]	10
Bedraad of draadloos	/	draadloos
Over-the-air firmware updates	/	nee
Aantal uplink berichten	[slim object/dag]	48
Uplink payload grootte	[bytes]	2
Zijn downlink berichten vereist?	/	nee

Ondanks het feit dat de use case vooral gericht is op lokale IoT-systemen waarbij het hoog bereik van LPWAN-connectiviteitstechnologieën niet nodig wordt geacht, wordt er toch geopteerd om LPWAN-connectiviteitstechnologieën te gebruiken. Dit ondermeer omdat de verschillende LPWAN-connectiviteitstechnologieën een hoger link budget hebben dan bijvoorbeeld WiFi of Bluetooth, dat zeker van pas komt als de connectiviteit bijvoorbeeld de hoge isolatiebarrières van koelcellen moet doorbreken. Aangezien er slechts een beperkt aantal slimme objecten zullen

geïnstalleerd worden, wordt er geopteerd om gebruik te maken van een publieke netwerkprovider en wordt de optie om een privaat LoRa netwerk te installeren buiten beschouwing gelaten, omdat de hoge initiële investeringskost geen haalbare kaart zou zijn. Ook de optie om de slimme objecten zelf te ontwerpen en produceren wordt buiten beschouwing gelaten.

De slimme objecten worden aangekocht. De Celsius Air Industrial van Allóra Factory bijvoorbeeld kan zowel Sigfox, LoRa als NB-IoT connectiviteit voorzien. Aangezien de verschillen LP-WAN-connectiviteitstechnologieën een verschillende, niet te verwaarlozen, kost hebben voor de communicatiemodule of chipset wordt dit doorgerekend naar de klant. Er wordt uitgegaan van een basisprijs van € 40 voor een slim object, daarbovenop wordt de kostprijs van de communicatiemodule of chipset toegevoegd, vermenigvuldigd met 1,6 tot 2,5 zoals geïnformeerd door de expert. Zoals vermeld in subsectie 2.1.2 van deel II, is de aankoopprijs van een Sigfox, LoRa en NB-IoT communicatiemodule respectievelijk € 1,79, € 3,47 en € 14,71 per stuk, indien aangekocht in oplages van 5000. Dit resulteert in uiteindelijk een extra kost, op de basis hardwareprijs, van respectievelijk € 3,58, € 6,94 € en € 29,42 indien de aankoopprijs vermenigvuldigd wordt met 2. De slimme objecten kosten uiteindelijk voor Sigfox, LoRa en NB-IoT respectievelijk € 43,58, € 46,94 en € 69,42 per slim object. De deployment kost is verwaarloosbaar omdat de slimme objecten gewoon in de koelcellen worden geplaatst en niet geïntegreerd. Alle slimme objecten worden samen aangekocht en geïnstalleerd, er is dus geen sprake van een gefaseerde uitrol. Aangezien het gaat om slechts een beperkt aantal slimme objecten zijn schaalvoordelen niet van toepassing.

De batterijlevensduur is moeilijk te voorspellen maar in een simulatietijd van tien jaar is de kans groot dat ze minstens eenmaal gaan moeten vervangen worden. Er wordt geopteerd om de batterijen allemaal samen te vervangen, dit om de vervangingstijd en bijkomstig de individuele kost om de batterijen te vervangen laag te houden. De vervangingstijd van batterijen is afhankelijk van, onder andere, waar de slimme objecten zijn geïnstalleerd, hoeveel batterijen er in elk slim object aanwezig zijn en hoe ver de verschillende slimme objecten van elkaar verwijderd zijn. De slimme objecten in deze use case zijn eenvoudig bereikbaar, niet ver van elkaar verwijderd en bestaan uit slechts 1 batterij. De auteur is van mening dat in een tijdspanne van een uur, de batterijen van 10 slimme objecten vervangen kunnen worden. Indien personeelskosten gerekend worden aan € 50 per uur levert dit een personeelskost van 5 € per slim object op. Aangezien de slimme objecten werken in een omgeving waarbij de omgevingstemperatuur rond of onder het vriespunt ligt, wordt er gebruik gemaakt van lithium batterijen. Lithium batterijen hebben tevens ook een langere levensduur dan de goedkopere alkaline batterijen. De kostprijs van een enkele lithium batterij is rond de € 5. De totale kostprijs per slim object om de batterij te vervangen is de som van de personeelskost en de aankoopprijs van de batterij. Dit geeft als resultaat € 10 per slim object.

De slimme objecten zenden tweemaal per uur een bericht met als payload de waargenomen

temperatuurwaarde, wat neer komt op 48 uplink berichten per slim object per dag. Een (negatief) kommagetal in het verwachte interval van een temperatuurmeting kan voorgesteld worden door 2 bytes. Er worden geen downlink berichten uitgewisseld.

1.4 Resultaten en discussie

De verschillende simulaties met respectievelijk de keuze tussen de drie LPWAN-connectiviteitstechnologieën Sigfox, LoRa, NB-IoT en de drie verschillende IoT-platformen Google Cloud IoT Core, Microsoft Azure IoT Hub en AWS IoT Core wijzen er op dat de combinatie Sigfox en Google Cloud IoT Core de meest economische keuze is. Het maandelijkse gratis datavolume dat Google Cloud IoT Core ter beschikking stelt heeft als gevolg dat er geen kosten moeten betaald worden voor connectiviteit met het IoT-platform. Bij Microsoft Azure IoT Hub bedraagt de jaarlijkse connectiviteitskost € 253, wat neerkomt op een totale extra kost van € 2.530 over een levensduur van 10 jaar.

Verschillende simulaties wijzen er op dat de Sigfox connectiviteitstechnologie beduidend goedkoper is dan LoRa en NB-IoT. De goedkope connectiviteit uit zich ook in de goedkopere communicatiemodule of chipset kost. Sigfox connectiviteit gaat wel gepaard met een aantal belangrijke nadelen ten opzichte van LoRa en NB-IoT. Zo zijn er slechts vier downlink berichten per dag mogelijk, geen mogelijkheid tot over-the-air updates en de maximale payload van 12 bytes.

Besluit

Steeds meer alledaagse toestellen worden verbonden met het Internet waardoor ze 'slim' worden, een evolutie die ook wel gekend staat als het Internet of Things (IoT). Na telefonie, televisie en het Internet kan IoT beschouwd worden als de vierde revolutie tot een intergeconnecteerde wereld, wat een significante impact zal uitoefenen op het dagdagelijkse leven. Het is dan ook niet verwonderlijk dat bedrijven nieuwsgierig zijn of de verschillende IoT-geïnduceerde investeringsopportuniteiten een positieve uitkomst kunnen bieden voor hun use case. IoT use cases zijn vaak technisch haalbaar, maar zijn echter vaak niet economisch rendabel. Hierom is het voor bedrijven belangrijk om a priori de kosten van een IoT use case doorheen zijn levenscyclus in kaart te brengen.

Deze masterproef had als doel het ontwerpen en ontwikkelen van een online selectie- en modelleringstool, dat IoT-ontwikkelaars toelaat, op een snelle en overzichtelijke manier, a priori de kosten van een IoT use case in kaart te brengen. De verschillende componenten van een IoT-systeem, zijnde de slimme objecten, connectiviteit en communicatie, het IoT-platform en de IoT-applicatie, werden zo generisch mogelijk gemodelleerd. Specifiek werden de verschillende voornaamste LPWAN-connectiviteitstechnologieën en IoT-platformen onderling vergeleken op zowel technisch als economisch vlak. Het intrinsieke heterogene karakter en de verschillende stakeholders van IoT-systemen maken de techno-economisch vergelijking echter niet eenvoudig. Verschillende componenten van een IoT-systeem zijn volledig use case afhankelijk en zijn dan ook moeilijk generisch in kaart te brengen. Omwille van deze reden is onder andere de IoT-applicatie component van een IoT-systeem buiten beschouwing gelaten. Bijkomstig is er ook de grote onzekerheid over bepaalde parameters zoals bijvoorbeeld de batterijlevensduur en vervangings- en installatietijden. De opgesomde factoren bevestigen dat de verschillende uitgevoerde simulaties vanuit een kritisch oogpunt bekeken moeten worden.

De belangrijkste persoonlijke les die uit de masterproef kan getrokken worden, komt neer op het spreekwoord "een goed begin is het halve werk". Ondermeer exact weten wat de verwachtingen en eisen zijn is cruciaal om tot een goed eindresultaat te komen. Het maken van een duidelijk en gedetailleerd ontwerp voorkomt dat men verzeild geraakt in een doolhof van gemaakte implementatiekeuzes, waarin later niet of met slechts heel veel moeilijkheden opnieuw de weg kan

gevonden worden.

Mogelijke uitbreidingen op het techno-economische model zijn:

- De overige kostcomponenten van een IoT-platform modelleren. Dit is echter niet eenvoudig vanuit een theoretisch perspectief. De aangewezen methode zou dan ook zijn om de verschillende kosten van IoT-platformen experimenteel te bestuderen, waarbij een opstelling connectiviteit realiseert met verschillende IoT-platformen. Aangezien de meeste IoT-platformen een gratis testperiode ter beschikking stellen is dit kostenloos te realiseren.
- De verschillende headerformaten van de IoT-connectiviteitsprotocollen en IoT-communicatieprotocollen mee in rekening brengen.
- De batterijlevensduur van slimme objecten pogen te voorspellen. Zoals eerder al vermeld is dit een complexe taak, wat ook de expert in het ontwikkelen van slimme objecten bevestigt. De batterijlevensduur van slimme objecten is onder andere afhankelijk van: het type batterij, het aantal batterijen, de omgevingstemperatuur van het slimme object, het aantal uplink en voornamelijk het aantal downlink berichten, transmissiekracht, energie-efficiëntie van de hardware, energie-efficiëntie van de firmware, ...

Referentielijst

[1] K. L. Lueth, "State of the IoT 2018: Number of IoT devices now at 7B - Market accelerating," 2018. [Online]. Available: https://iot-analytics.com/state-of-the-iot-update-q1-q2-2018-number-of-iot-devices-now-7b/

[2] Vlaamse Overheid, "De digitale energiemeter," 2017. [Online]. Available: https://www.vlaanderen.be/bouwen-wonen-en-energie/zelf-energie-produceren/zonnepanelen-en-zonneboilers/de-digitale-energiemeter

[3] W. Jeff, "A drone program taking flight." [Online]. Available: https://blog.aboutamazon.com/transportation/a-drone-program-taking-flight

[4] J. Hoebeke and E. De Poorter, *Internet Of Things*. Ghent University, 2018.

[5] ITU, "The Internet of Things," ITU Internet, Tech. Rep., 2005. [Online]. Available: https://www.itu.int/net/wsis/tunis/newsroom/stats/The-Internet-of-Things-2005.pdf

[6] P. P. Ray, "A survey of IoT cloud platforms," *Future Computing and Informatics Journal*, vol. 1, no. 1-2, pp. 35–46, dec 2017. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2314728816300149

[7] J. Gubbi, R. Buyya, S. Marusic, and M. Palaniswami, "Internet of Things (IoT): A vision, architectural elements, and future directions," *Future Generation Computer Systems*, vol. 29, no. 7, pp. 1645–1660, sep 2013. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167739X13000241

[8] L. Atzori, A. Iera, and G. Morabito, "The Internet of Things: A survey," *Computer Networks*, vol. 54, no. 15, pp. 2787–2805, oct 2010. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1389128610001568

[9] M. Rouse, "What is internet of things (IoT)?" 2019. [Online]. Available: https://internetofthingsagenda.techtarget.com/definition/Internet-of-Things-IoT

[10] J. Teicher, "The little-known story of the first IoT device," 2018. [Online]. Available: https://www.ibm.com/blogs/industries/little-known-story-first-iot-device/

[11] M. Weiser, "The Computer for the 21st Century," *Scientific American*, 1991. [Online]. Available: https://www.ics.uci.edu/~corps/phaseii/Weiser-Computer21stCentury-SciAm.pdf

[12] K. Ashton, "That 'Internet of Things' Thing," 2009. [Online]. Available: https://www.rfidjournal.com/articles/view?4986

[13] N. Gershenfeld, *When Things Start to Think*. New York, NY, USA: Henry Holt and Co., Inc., 1999.

[14] IPSO Alliance, "IPSO Alliance - OMA SpecWorks." [Online]. Available: https://www.omaspecworks.org/ipso-alliance/

[15] Cisco Internet Business Solutions Group (IBSG), "The Internet of Things - How the Next Evolution of the Internet is Changing Everything," *CISCO white paper*, no. April, pp. 1–11, 2011. [Online]. Available: https://www.cisco.com/c/dam/en_us/about/ac79/docs/innov/IoT_IBSG_0411FINAL.pdf

[16] G. Sachs, "The Internet of Things: Making sense of the next mega-trend," *Goldman Sachs Global Investment Research*, pp. 1–15, 2014. [Online]. Available: https://www.goldmansachs.com/insights/pages/internet-of-things/iot-report.pdf

[17] C. González García, D. Meana Llorián, C. Pelayo G-Bustelo, and J. M. Cueva-Lovelle, "A review about Smart Objects, Sensors, and Actuators," *International Journal of Interactive Multimedia and Artificial Intelligence*, vol. 4, no. 3, p. 7, 2016. [Online]. Available: https://www.researchgate.net/publication/307638707

[18] C. Bormann, M. Ersue, and A. Keranen, "Terminology for Constrained-Node Networks," Internet Engineering Task Force, Tech. Rep., may 2014. [Online]. Available: https://www.rfc-editor.org/rfc/rfc7228.txt

[19] ARDUINO, "Arduino YÚN," 2014. [Online]. Available: https://store.arduino.cc/arduino-yun

[20] Raspberry Pi Foundation, "Raspberry Pi 3 Model B," 2016. [Online]. Available: https://www.raspberrypi.org/products/raspberry-pi-3-model-b/

[21] Z. Khan, J. J. Lehtomaki, S. I. Iellamo, R. Vuohtoniemi, E. Hossain, and Z. Han, "IoT Connectivity in Radar Bands: A Shared Access Model Based on Spectrum Measurements," *IEEE Communications Magazine*, vol. 55, no. 2, pp. 88–96, feb 2017. [Online]. Available: http://ieeexplore.ieee.org/document/7842417/

[22] R. S. Sinha, Y. Wei, and S. H. Hwang, "A survey on LPWA technology: LoRa and NB-IoT," pp. 14–21, mar 2017. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2405959517300061

[23] K. Mekki, E. Bajic, F. Chaxel, and F. Meyer, "A comparative study of LPWAN technologies for large-scale IoT deployment," pp. 1–7, mar 2018. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2405959517302953

[24] M. Centenaro, L. Vangelista, A. Zanella, and M. Zorzi, "Long-range communications in unlicensed bands: The rising stars in the IoT and smart city scenarios," *IEEE Wireless Communications*, vol. 23, no. 5, pp. 60–67, oct 2016. [Online]. Available: http://ieeexplore.ieee.org/document/7721743/

[25] D. Patel and M. Won, "Experimental Study on Low Power Wide Area Networks (LPWAN) for Mobile Internet of Things," in *IEEE Vehicular Technology Conference*, vol. 2017-June. IEEE, jun 2017, pp. 1–5. [Online]. Available: http://ieeexplore.ieee.org/document/8108501/

[26] Engie M2M, "Engie M2M - Official Sigfox provider in Belgium." [Online]. Available: https://www.engiem2m.be/

[27] Sigfox, "Radio technology keypoints," 2017. [Online]. Available: https://www.sigfox.com/en/sigfox-iot-radio-technology

[28] LoRa Alliance™, "Home page | LoRa Alliance™," 2018. [Online]. Available: https://lora-alliance.org/

[29] U. Raza, P. Kulkarni, and M. Sooriyabandara, "Low Power Wide Area Networks: An Overview," *IEEE Communications Surveys and Tutorials*, vol. 19, no. 2, pp. 855–873, 2017. [Online]. Available: http://ieeexplore.ieee.org/document/7815384/

[30] B. Reynders, W. Meert, and S. Pollin, "Range and coexistence analysis of long range unlicensed communication," in *2016 23rd International Conference on Telecommunications, ICT 2016*. IEEE, may 2016, pp. 1–6. [Online]. Available: http://ieeexplore.ieee.org/document/7500415/

[31] K. Mikhaylov and J. Petäjäjärvi, "Analysis of Capacity and Scalability of the LoRa Low Power Wide Area Network Technology," *University of Oulu, Centre for Wireless Communications, Finland*, 2016.

[32] 3GPP, "Standardization of NB-IOT completed." [Online]. Available: https://www.3gpp.org/news-events/1785-nb_iot_complete

[33] Orange, "Orange, partner voor uw Internet of Things-projecten." [Online]. Available: https://business.orange.be/nl/business-oplossingen/internet-things

[34] Proximus, "Proximus lanceert NB-IoT-netwerk om digitale meters te ondersteunen." [Online]. Available: https://www.proximus.com/nl/news/proximus-launches-nb-iot-network-support-digital-meters

[35] Telenet, "Telenet Tinx van Telenet Business: Internet of Things voor uw bedrijf." [Online]. Available: https://www2.telenet.be/nl/business/producten-diensten/internet-of-things/

[36] Y. P. Wang, X. Lin, A. Adhikary, A. Grövlen, Y. Sui, Y. Blankenship, J. Bergman, and H. S. Razaghi, "A Primer on 3GPP Narrowband Internet of Things," *IEEE Communications Magazine*, vol. 55, no. 3, pp. 117–123, 2017. [Online]. Available: https://arxiv.org/ftp/arxiv/papers/1606/1606.04171.pdf

[37] A. Adhikary, X. Lin, and Y. P. Eric Wang, "Performance evaluation of NB-IoT coverage," in *IEEE Vehicular Technology Conference*. IEEE, sep 2017, pp. 1–5. [Online]. Available: http://ieeexplore.ieee.org/document/7881160/

[38] N. Naik and P. Jenkins, "Web protocols and challenges of Web latency in the Web of Things," in *International Conference on Ubiquitous and Future Networks, ICUFN*, vol. 2016-Augus. IEEE, jul 2016, pp. 845–850. [Online]. Available: http://ieeexplore.ieee.org/document/7537156/

[39] N. Naik, "Choice of effective messaging protocols for IoT systems: MQTT, CoAP, AMQP and HTTP," in *2017 IEEE International Symposium on Systems Engineering, ISSE 2017 - Proceedings*. IEEE, oct 2017, pp. 1–7. [Online]. Available: http://ieeexplore.ieee.org/document/8088251/

[40] S. Bandyopadhyay and A. Bhattacharyya, "Lightweight Internet protocols for web enablement of sensors using constrained gateway devices," in *2013 International Conference on Computing, Networking and Communications, ICNC 2013*. IEEE, jan 2013, pp. 334–340. [Online]. Available: http://ieeexplore.ieee.org/document/6504105/

[41] V. Karagiannis, P. Chatzimisios, F. Vazquez-gallego, and J. Alonso-zarate, "Application Layer Protocols for the Internet of Things," *Transaction on IoT and Cloud Computing*, pp. 1–10, 2015. [Online]. Available: https://www.researchgate.net/publication/303192188

[42] D. Thangavel, X. Ma, A. Valera, H. X. Tan, and C. K. Y. Tan, "Performance evaluation of MQTT and CoAP via a common middleware," in *IEEE ISSNIP 2014 - 2014 IEEE 9th International Conference on Intelligent Sensors, Sensor Networks and Information Processing, Conference Proceedings*. IEEE, apr 2014, pp. 1–6. [Online]. Available: http://ieeexplore.ieee.org/document/6827678/

[43] T. Jaffey, "MQTT and CoAP, IoT Protocols," *Eclipse*, 2014. [Online]. Available: https://www.eclipse.org/community/eclipse_newsletter/2014/february/article2.php

[44] MQTT.org, "MQTT used by Facebook Messenger," 2011. [Online]. Available: http://mqtt.org/2011/08/mqtt-used-by-facebook-messenger

[45] K. Lasse Lueth and J. Kotzorek, “IoT Platforms: The central backbone for the Internet of Things,” IoT Analytics, Tech. Rep., 2015.

[46] P. Scully, “5 Things To Know About The IoT Platform Ecosystem,” 2016. [Online]. Available: https://iot-analytics.com/5-things-know-about-iot-platform/

[47] Cisco Systems Inc., “The Internet of Things Reference Model,” in *Internet of Things World Forum*, 2014, pp. 1–12. [Online]. Available: http://cdn.iotwf.com/resources/71/IoT_Reference_Model_White_Paper_June_4_2014.pdf

[48] G. Dunko, J. Misra, J. Robertson, and T. Snyder, *A Reference Guide to the Internet of Things*. Bridgera, 2017.

[49] Sigfox, “Sigfox Certification Handbook,” Sigfox, Tech. Rep., 2019. [Online]. Available: https://storage.sbg1.cloud.ovh.net/v1/AUTH_669d7dfced0b44518cb186841d7cbd75/staging_docs/att30344767-SigfoxCertificationHandbook.pdf

Bijlage A: Installatiehandleiding

Deze bijlage beschrijft hoe de webapplicatie lokaal geïnstalleerd en uitgevoerd kan worden. Dit kan op twee manieren gebeuren, de eenvoudigste manier is met behulp van Docker, waardoor de webapplicatie gevirtualiseerd wordt. Het voordeel hierbij is dat enkel Docker geïnstalleerd dient te worden en niet de afzonderlijke softwarecomponenten waarop de webapplicatie gebouwd is. Het nadeel is dan weer dat aanpassingen aan de code of databank niet mogelijk zijn. De tweede manier, zonder Docker, vereist de installatie van meerdere softwarecomponenten, maar de gebruiker kan dan wel de code aanpassen. Merk op dat een lokale installatie niet vereist is indien enkel gebruik van de webapplicatie gewenst is. De webapplicatie is publiek beschikbaar via: http://193.190.127.227:4200/ tot 31 juli 2019.

1 Installatie met behulp van Docker

Docker virtualiseert de benodigde software om de webapplicatie lokaal te installeren en uit te voeren; er dient dus geen additionele software geïnstalleerd te worden. De enige vereiste is uiteraard dat Docker geïnstalleerd is. De gratis Docker Community Edition (CE) volstaat, het is dus niet nodig om de Docker Enterprise Edition (EE) te installeren. Merk op dat de webapplicatie enkel uitgevoerd kan worden met Docker CE of EE, het is dus niet mogelijk om Docker Toolbox te gebruiken. Indien u niet beschikt over een van de vereiste besturingssystemen dient de webapplicatie zonder Docker geïnstalleerd en uitgevoerd te worden, zoals beschreven in sectie 2.

Voor de installatie van Docker CE wordt doorverwezen naar de officiële Docker CE installatiehandleidingen (enkel beschikbaar in het Engels):

- Microsoft Windows 10 Professional of Enterprise 64 bit:
  https://docs.docker.com/docker-for-windows/install/
- Apple MacOS Sierra 10.12 of hoger:
  https://docs.docker.com/docker-for-mac/install/

- Linux CentOS:
  https://docs.docker.com/install/linux/docker-ce/centos/
- Linux Debian:
  https://docs.docker.com/install/linux/docker-ce/debian/
- Linux Fedora:
  https://docs.docker.com/install/linux/docker-ce/fedora/
- Linux Ubuntu:
  https://docs.docker.com/install/linux/docker-ce/ubuntu/

Nadat Docker CE (of EE) geïnstalleerd is, dient enkel nog Docker Compose geïnstalleerd te worden. Docker Compose vereenvoudigt het uitvoeren van de webapplicatie. Indien u Docker CE (of EE) heeft geïnstalleerd op Windows 10 of MacOS is Docker Compose reeds geïnstalleerd, deze stap kan in dat geval overgeslagen worden. Voor de verschillende Linux versies wordt doorverwezen naar de officiële Docker Compose installatiehandleiding (enkel beschikbaar in het Engels): https://docs.docker.com/compose/install/.

Eenmaal zowel Docker CE (of EE) en Docker Compose geïnstalleerd zijn, kan de webapplicatie uitgevoerd worden door volgende stappen te ondernemen:

1. Een Terminal (MacOS of Linux) of Command Prompt (Windows) openen in de submap *docker*.
2. Het commando *docker-compose up --build* uitvoeren in de Terminal of Command Prompt.
3. Na enige tijd heeft Docker de webapplicatie opgestart en kan deze bezocht worden door te surfen naar localhost:4200 in een webbrowser naar keuze.

2 Installatie zonder Docker

De installatie zonder Docker laat aanpassingen aan de code toe waardoor de webapplicatie naar eigen hand kan gezet worden. Hiervoor dienen wel de extra softwarecomponenten, waarop de webapplicatie gebouwd is, geïnstalleerd te worden.

Allereerst dient de MongoDB Community Server databank gedownload en geïnstalleerd te worden. De MongoDB Community Server databank kan gedownload worden via https://www.mongodb.com/download-center/community. Eenmaal de databank gedownload is, dient de databank nog geïnstalleerd te worden. Hiervoor wordt doorverwezen naar de officiële MongoDB Community Server installatiehandleidingen (enkel beschikbaar in het Engels):

- Microsoft Windows:
  https://docs.mongodb.com/manual/tutorial/install-mongodb-on-windows/
- Apple MacOS:
  https://docs.mongodb.com/manual/tutorial/install-mongodb-on-os-x/
- Linux:
  https://docs.mongodb.com/manual/administration/install-on-linux/

Enkele opmerkingen omtrent de MongoDB Community Server databank:

- Eenmaal de databank gedownload en geïnstalleerd is, dient deze nog manueel opgestart te worden. Dit is afhankelijk van de gevolgde installatieprocedure en hiervoor wordt dan ook doorverwezen naar de officiële MongoDB Community Server installatiehandleiding. Bij succes verschijnt volgende bevestiging: *waiting for connections on port 27017*.
- De webapplicatie maakt gebruik van de standaard 27017 poort.

Eenmaal de databank geïnstalleerd en gestart is dient deze nog te beschikken over de benodigde data. De databank opvullen met de benodigde data dient uiteraard maar eenmalig te gebeuren, en kan gebeuren door volgende stappen te ondernemen:

1. Een Terminal (MacOS of Linux) of Command Prompt (Windows) openen in de submap *database*.
2. Volgende commando's uitvoeren in de Terminal of Command Prompt:
   mongoimport --db iot_tool --collection connectivityTechnologies --type json --file connectivityTechnologies.json --jsonArray
   mongoimport --db iot_tool --collection IoTCloudPlatforms --type json --file IoTCloudPlatforms.json --jsonArray

Nadat de databank geïnstalleerd en opgevuld is, dient de *runtime environment* Node.js geïnstalleerd te worden. Node.js kan gedownload worden via: https://nodejs.org/en/download/. De installatie is eenvoudig, maar controleer zeker of de optie om de Node Package Manager (NPM) te downloaden en te installeren ook aangevinkt is.

Als laatste dient de webapplicatie te beschikken over alle benodigde pakketten om de webapplicatie te kunnen uitvoeren. Gelukkig biedt, de eerder geïnstalleerde, NPM hiervoor een eenvoudige oplossing aan. Om de laatste softwarecomponenten te downloaden en installeren, dienen volgende stappen ondernomen te worden:

1. Een Terminal (MacOS of Linux) of Command Prompt (Windows) openen in de submap *back-end*.
2. Het commando *npm install* uitvoeren in de Terminal of Command Prompt.
3. Een Terminal (MacOS of Linux) of Command Prompt (Windows) openen in de submap *front-end*.
4. Het commando *npm install* uitvoeren in de Terminal of Command Prompt.

Eenmaal alle benodigde softwarecomponenten geïnstalleerd zijn, kan de webapplicatie uitgevoerd worden door volgende stappen te ondernemen:

1. De databank opstarten zoals hierboven beschreven.
2. Een Terminal (MacOS of Linux) of Command Prompt (Windows) openen in de submap *back-end*.
3. Het commando *npm start* uitvoeren in de Terminal of Command Prompt.
4. Een Terminal (MacOS of Linux) of Command Prompt (Windows) openen in de submap *front-end*.
5. Het commando *npm start* uitvoeren in de Terminal of Command Prompt.
6. Na enige tijd is de webapplicatie opgestart en kan deze bezocht worden door te surfen naar localhost:4200 in een webbrowser naar keuze.

Bijlage B: Gebruikershandleiding

Deze bijlage beschrijft hoe de webapplicatie gebruikt dient te worden. De webapplicatie is publiek beschikbaar via: http://193.190.127.227:4200/ tot 31 juli 2019. Het is ook mogelijk om de webapplicatie lokaal te installeren en uit te voeren. De handleiding om de webapplicatie lokaal te installeren en uit te voeren is beschreven in bijlage A.

1 Registreren en inloggen

De gebruiker kan een account aanmaken of inloggen door te surfen naar respectievelijk de /register en /login webpagina's. Authenticatie gebeurt op basis van het verschafte e-mailadres en het zelf gekozen wachtwoord. Merk op dat de publieke webserver (193.190.127.227) niet beschikt over een SSL-certificaat. De communicatie is dus niet versleuteld en het wordt dan ook ten zeerste aangeraden om een eenmalig wachtwoord te gebruiken, een enkel inputkarakter volstaat. Eenmaal de gebruiker succesvol is ingelogd kan hij/zij de profielpagina raadplegen door te surfen naar de /profile webpagina. De gebruiker beschikt niet over de mogelijkheid om geregistreerde accounts te verwijderen. Indien een gebruiker toch wenst dat zijn/haar account verwijderd wordt, dient de gebruiker de administrator van de webapplicatie te contacteren. De gebruiker verschaft dan het gebruikte e-mailadres, waardoor de administrator eenvoudig het account kan verwijderen door middel van de databank interface. De gebruiker, noch de administrator, beschikken over de mogelijkheid om een verloren wachtwoord te achterhalen of te wijzigen.

2 Simulaties uitvoeren

De gebruiker kan simulaties uitvoeren door de verschillende inputparameters in te geven op de thuispagina. Het is aangewezen om elke inputparameter in te vullen, indien de inputparameter geen waarde heeft volstaat een 0 als input. Simulaties uitvoeren is mogelijk zonder dat de gebruiker geregistreerd en ingelogd is. Merk wel op dat de gebruiker enkel simulaties kan opslaan indien hij/zij succesvol ingelogd is.

3 Simulaties opslaan en inladen

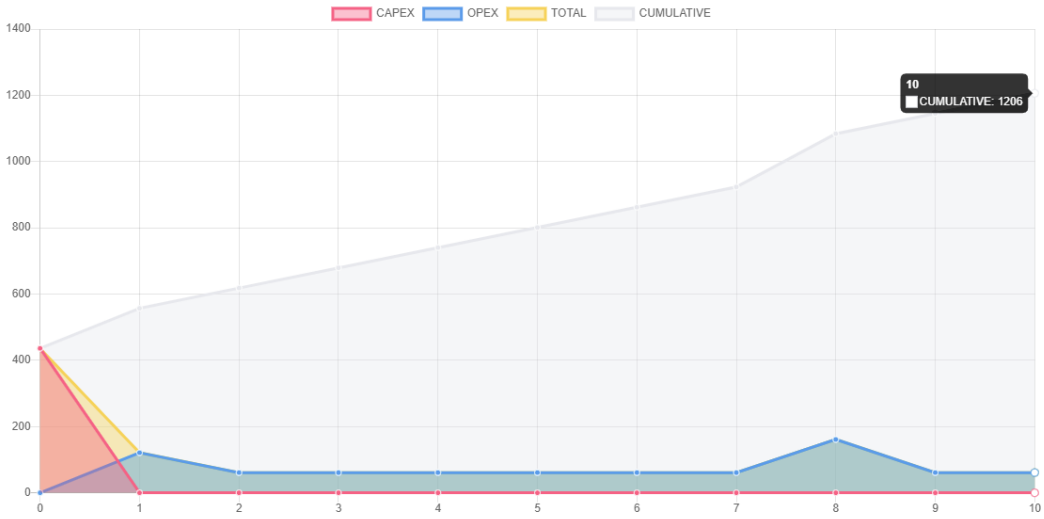
De gebruiker kan enkel simulaties opslaan en inladen indien hij/zij succesvol is ingelogd. Om simulaties op te slaan dient de gebruiker de gewenste inputparameters in te vullen en daarna de knop *Save simulation* aan te klikken. Het pop-up venster vraagt om de simulatie een naam en een beschrijving te geven. Een enkel karakter volstaat voor beide inputvelden. Eenmaal de gebruiker een of meerdere simulaties succesvol heeft opgeslagen kan hij/zij deze opnieuw inladen door de knop *load* van de gewenste simulatie aan te klikken. De gebruiker wordt genavigeerd naar de thuispagina en de opgeslagen inputparameters worden ingeladen waardoor de gebruiker de simulatie verder kan zetten. De gebruiker beschikt niet over de mogelijkheid om opgeslagen simulaties te verwijderen. Indien een gebruiker toch wenst dat een of meerdere van zijn/haar simulaties verwijderd wordt, dient de gebruiker de administrator van de webapplicatie te contacteren. De gebruiker verschaft dan het gebruikte e-mailadres en de verschillende simulatienamen, waardoor de administrator eenvoudig de gewenste simulaties kan verwijderen door middel van de databank interface. Indien de gebruiker de naam of beschrijving van een simulatie wil wijzigen, dient deze eveneens de administrator te contacteren.

Bijlage C: Use case simulaties

Lifecycle costs summary

Description	Year										
	0	1	2	3	4	5	6	7	8	9	10
smart device hardware cost	436	0	0	0	0	0	0	0	0	0	0
smart device deployment cost	0	0	0	0	0	0	0	0	0	0	0
CAPEX	436	0	0	0	0	0	0	0	0	0	0
battery replacements cost	0	0	0	0	0	0	0	0	100	0	0
communication and connectivity cost	0	121	61	61	61	61	61	61	61	61	61
IoT platform cost	0	0	0	0	0	0	0	0	0	0	0
OPEX	0	121	61	61	61	61	61	61	161	61	61
TOTAL	436	121	61	61	61	61	61	61	161	61	61
CUMULATIVE	436	557	618	679	740	801	862	923	1084	1145	1206

Lifecycle costs graph

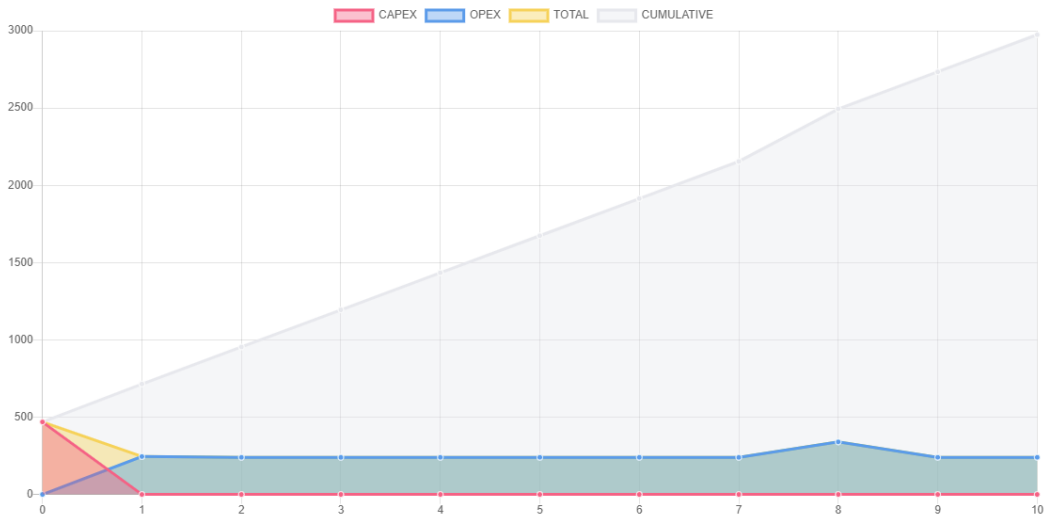


Figuur C.1: Simulatie met Sigfox en Google Cloud IoT Core.

Lifecycle costs summary

Description	Year										
	0	1	2	3	4	5	6	7	8	9	10
smart device hardware cost	469	0	0	0	0	0	0	0	0	0	0
smart device deployment cost	0	0	0	0	0	0	0	0	0	0	0
CAPEX	469	0	0	0	0	0	0	0	0	0	0
battery replacements cost	0	0	0	0	0	0	0	0	100	0	0
communication and connectivity cost	0	245	240	240	240	240	240	240	240	240	240
IoT platform cost	0	0	0	0	0	0	0	0	0	0	0
OPEX	0	245	240	240	240	240	240	240	340	240	240
TOTAL	469	245	240	240	240	240	240	240	340	240	240
CUMULATIVE	469	714	954	1194	1434	1674	1914	2154	2494	2734	2974

Lifecycle costs graph

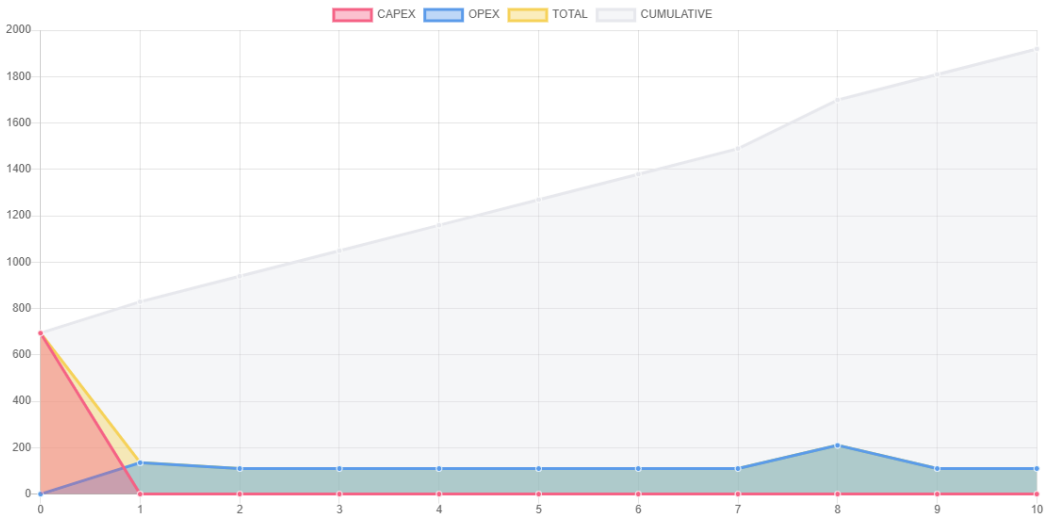


Figuur C.2: Simulatie met LoRa en Google Cloud IoT Core.

Lifecycle costs summary

Description	Year										
	0	1	2	3	4	5	6	7	8	9	10
smart device hardware cost	694	0	0	0	0	0	0	0	0	0	0
smart device deployment cost	0	0	0	0	0	0	0	0	0	0	0
CAPEX	694	0	0	0	0	0	0	0	0	0	0
battery replacements cost	0	0	0	0	0	0	0	0	100	0	0
communication and connectivity cost	0	135	110	110	110	110	110	110	110	110	110
IoT platform cost	0	0	0	0	0	0	0	0	0	0	0
OPEX	0	135	110	110	110	110	110	110	210	110	110
TOTAL	694	135	110	110	110	110	110	110	210	110	110
CUMULATIVE	694	829	939	1049	1159	1269	1379	1489	1699	1809	1919

Lifecycle costs graph



Figuur C.3: Simulatie met NB-IoT en Google Cloud IoT Core.