FACULTY OF INFORMATION TECHNOLOGY AND ELECTRICAL ENGINEERING

Jussi Jokio

Back-end Reference Architecture for Smart Water Meter Data Gathering Service

Master's Thesis
Degree Programme in Computer Science and Engineering
February 2020

Jokio J. (2020) Back-end Reference Architecture for Smart Water Meter Data Gathering Service. University of Oulu, Degree Programme in Computer Science and Engineering. Master's Thesis, 48p.

ABSTRACT

The Finnish waterworks industry is on the brink of digitalization. Currently, many of them have started to convert their water meters to smart water meters. However, there is yet no suitable solution for gathering the IoT data from these smart water meters.

To answer their arising needs, many pilots and workshops have been conducted. Those pilots have yielded some basic ground rules for their use cases. In this study, those ground rules have been gathered as a set of requirement categories. The categories are studied and analyzed in order to establish a reference architecture for IoT data-gathering systems suitable for waterworks.

Using the requirements and the reference architecture, an information system, Dataservice, was implemented by Vesitieto Oy. The system gathers the IoT data and visualizes it to waterworks' employees. The System was deployed in Microsoft's cloud service, but other cloud vendors were examined as well. The system has a two-folded database system, the data required by the system, like users and user groups, are held in the SQL database. The IoT-data is held in a NoSQL database. The selected NoSQL database provider was MongoDB as it could be integrated with the cloud provider.

Keywords: IoT, Big data, Smart water meter, software architecture, software requirements

Jokio J. (2020) Etäluettavien vesimittareiden datapalvelun viitearkkitehtuuri. Oulun yliopisto, tietotekniikan tutkinto-ohjelma. Diplomityö, 48 s.

TIIVISTELMÄ

Suomen vesihuolto on digitalisaation partaalla. Tällä hetkellä monet vesilaitokset ovat alkaneet vaihtaa vanhoja analogisia vesimittareitaan älykkäiksi vesimittareiksi. Vesihuoltolaitokset eivät kuitenkaan ole löytäneet kaikille sopivaa ratkaisua IoT-tiedon keräämiseen älykkäistä vesimittareista. Vastatakseen vesilaitosten tarpeisiin, monia pilotteja ja työpajoja on järjestetty eri yhteistyökumppaneiden kanssa. Näistä eri piloteista on muodostunut käsitys siitä, kuinka vesimittareiden digitalisaatio voidaan ratkaista vesilaitoksilla. Tässä tutkimuksessa eri laitosten väliset perussäännöt on koottu ohjelmistovaatimusluokiksi. Näitä luokkia tutkitaan ja analysoidaan vesilaitoksille sopivan IoT-tiedonkeruujärjestelmän viitearkkitehtuurin luomiseksi.

Vaatimuksia ja viitearkkitehtuuria hyödyntäen Vesitieto Oy toteutti tietojärjestelmän nimeltään ”Dataservice”. Järjestelmä kerää IoT-tiedot ja visualisoi ne vesilaitosten työntekijöille. Järjestelmä otettiin käyttöön Microsoftin pilvipalvelussa, mutta myös muita pilvipalvelun palveluntarjoajia tutkittiin. Järjestelmässä on kaksiportainen tietokantajärjestelmä. Järjestelmän tarvitsemat tiedot kuten käyttäjät sekä käyttäjäryhmät pidetään SQLtietokannassa ja IoT-tiedot pidetään NoSQL-tietokannassa. Valittu NoSQL tietokantajärjestelmä oli MongoDB, koska se voitiin integroida pilvipalveluntarjoajan kanssa.

Avainsanat: Esineiden internet, IoT, big data, älykkäät vesimittarit, arkkitehtuuri, vaatimusmäärittely

TABLE OF CONTENTS

FOREWORD

The author would like to thank the following supervisors; Dr. Tech. Satu Tamminen and PhD Markus Harju for their guidance during the realization and writing process of this study.

The author would also like to thank the peer member Markus Heino as well as Suomen Vesitieto Oy for their valuable feedback and support.

Acknowledgments are also due to the mentally supportive individuals of the research team, through whose support work and its importance have gained more passion and encouragement.

Oulu, February 6th, 2020

Jussi Jokio

ABBREVIATIONS

AI	artificial intelligence
API	application programming interface
AWS	amazon web services
BLL	business logic layer
CEO	chief executive officer
CRUD	create, read, update & delete
EEA	European economic area
EU	European Union
FIWA	Finnish water utilities association
GDPR	general data protection regulation
HTTP	hypertext transfer protocol
IoT	internet of things
IS	information system
ISS	information system success
LoRa	long range
LoRaWan	long range low power wide area network protocol
LPWAN	low power wide area network
ML	machine learning
MQTT	message queuing telemetry transport
NoSQL	not only structured query language
PaaS	platform as a service
QoS	quality of service
SaaS	software as a service
SQL	structured query language
UI	user interface
UX	user experience

1. INTRODUCTION

In the modern-day industrial economy, information creates revenue and revenue creates business. With information, a business is able to create more informed decisions and with more informed decisions, the business is able to thrive. Even though waterworks are usually more non-profit organizations, they cannot operate without revenue. In Finland, waterworks are usually government-funded and creating revenue is not the most critical aspect of their business. The main function of the Finnish waterworks is to provide service and maintain their financial and operational status quo.

The field in which Finnish waterworks operates can be seen as ancient if one considers digitalization. On a survey of one Finnish waterworks, a classic pen and paper methods were uncovered along with an individual excel sheet. Mainly non-digital methods can be witnessed by the number of different digitization projects currently in progress by the Finnish Water Utilities Association (FIWA) [1]. Recently, FIWA has funded many multiorganizational digitization projects all over Finland. Alas, it seems painfully clear that Finnish waterworks are usually among the last organizations where new innovations or behavior models are adopted, especially regarding information technology. One could argue that, in Finland, the lack of digitalization is not as critical an issue as in other countries where freshwater is not an expendable asset [2, 3].

1.1. Motivation

It can be argued that the reason behind waterworks outdated methods lies in the lack of competition. Finnish waterworks are usually owned by the cities or are government-funded. This creates a non-competitive atmosphere. Even if some of the smaller waterworks are co-operational, digitalization has not yet fully found its way in this field. One of the other underlying reasons might be the excessive freshwater supplies in Finland [4]. There is no real motivation to hunt down leaks since it does not critically affect their business. There is no value in optimizing their process as their supplies are seen as a renewable natural resource. However, the amount and quality of fresh water is something that Finland takes serious interest in as a nation.

Even if smart water meters have been on the market starting from 2009 [5], only a handful of Finnish waterworks have started obtaining them. Only one of the waterworks had started a survey of smart water meters in 2015 [6]. In Finland, it is mandatory, by a law from 2013, to have a water meter in each household, but it was estimated in 2014 that, only in 300,000 households, water meters were actually installed. From these 300,000, only a half were smart water meters. This estimation was based on the data gathered from different water meter manufacturers. [7]. Since 2014, the numbers have certainly raised, but there is still a problem. There are not any reasonably priced information systems that could have collected the data from these water meters. There are many individual smart water meter data gathering systems implemented by manufacturers. But the problem with them is that they only support their own meters. A systematic and integrative solution is needed.

In addition to FIWA, there are many different organizations and enterprises that aim to bring Finnish waterworks to the 21st century. One of those organizations is Vesitieto Oy. It is a software company that is currently supplying many Finnish

waterworks with a new customer information system [8]. In 2018, they already had supplied multiple waterworks with their information systems (IS). Their goal is to provide unified information systems for as many Finnish waterworks as possible and to decrease the number of different systems needed to manage a waterworks as a business. Because of their connection to many waterworks, Vesitieto Oy was able to initialize an ambitious project for a unified and standardized data gathering system.

1.2. Description

The aim of this study is to create a foundation for Dataservice software for waterworks data management and gathering. Data will be gathered from smart water meters via the internet. This data-gathering software will be implemented for Suomen Vesitieto Oy and it will be part of, or more accurately, working adjacent with, their customer and invoicing software. Even when the systems can cooperate, the goal is to create a stand-alone system without the Vesitieto IS. Thus, creating an information system for waterworks with or without Vesitieto IS. The designed and implemented data gathering service software is later referenced as Dataservice.

Dataservice itself is a web-service application that gathers data from different sources like smart water meters, different grid sensors, and open-source data providers. It includes the user interface, application interface, and a storage system. The goal of Dataservice is to be a Software-as-a-Service (SaaS) platform, where data is distributed to different vendors, like customer portals, invoicing systems and customer information management systems. These systems are able to retrieve gathered data from Dataservice, and Dataservice acts as a data center for each system.

Dataservice will be a distributed system, operating within cloud storage. Important aspects of this study are to conclude the best cloud vendor for Dataservice, create a suitable requirement set and provide a reference architecture.

Data gathering software will collect data over the internet sent by smart water meters or other smart devices connected to the Internet of Things (IoT). The sent arithmetic data is collected and stored in a cloud service where the system has been deployed. From the cloud service, the data can be processed and analyzed. The analyzed data can then be shown to waterworks employees to help them manage their jobs. The overlay of the whole system has been displayed in Figure 1 below.

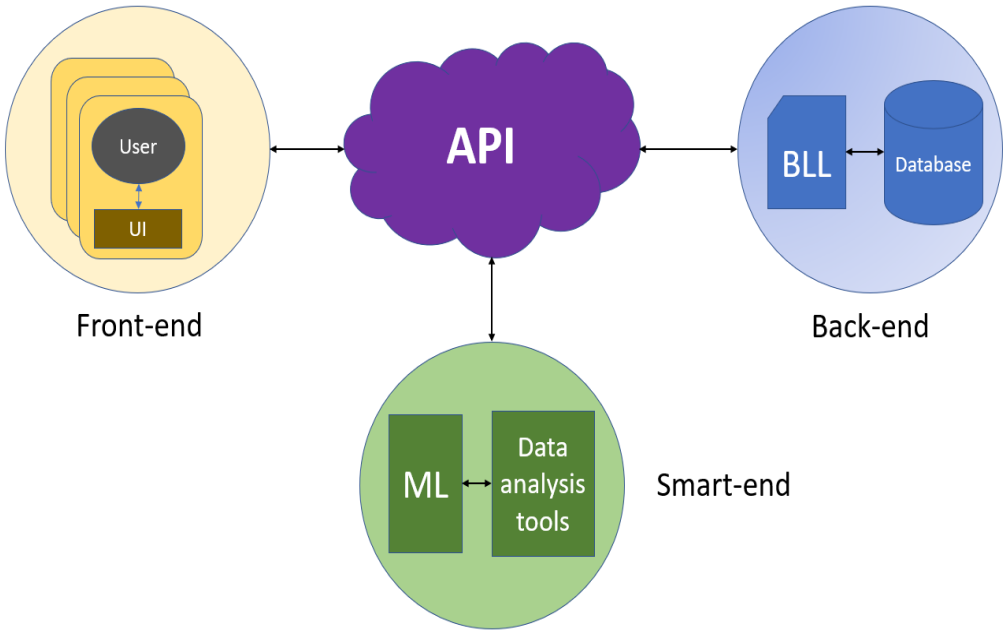

Figure 1: Web-service overlay for Dataservice.

The SaaS web-service consists of three different areas, the back-end, front-end, and the Smart-end. The front end means basically the user interface (UI) of the system as well as some minor client-sided data manipulations and visualizations. The back-end is the database, business layer logic (BLL) and the application program interface (API). The API is the main hub for the system, and it handles all the requests coming to the system. It is a vital point of integration between the system and any external or internal software. The BLL-component holds some data validation and processing functions and is considered to be the main data handler. The database is only storage for data, but still a crucial point for IS success. Although there are many arithmetic operations done inside the BLL, the main data manipulations and analysis are done within the smart-end by machine learning (ML) algorithms, Artificial intelligence (AI) agents and data analysis tools. These tools could be external applications like MATLAB or internal code libraries.

As seen in the system overlay, all these areas need to be connected together. The web-service itself is also connected to the IoT network and is able to receive data from IoT devices. Together, the devices and the web-service form a symbiosis. None of these areas can fully function without the other. When symbiosis between each different component is completed, Dataservice can be called an information system (IS) [9, 10].

The important thing to notice about this study is that the study is limited to clean water, smart water meters. However, the real Dataservice will also include other sensory data and IoT devices as well. In the scope of this study, it is adequate to limit the research to only smart water meters and clean water, as the concept of IoT is the same for other devices. Of course, there are different challenges to overcome when dealing with wastewater, for example. In this study, the devices are seen as entities that send validated data successfully. In addition to receiving data, Dataservice needs to register and identify each broadcasting device. The process of registering devices and gathering data to Dataservice will follow relatively standardized methods, thus, the processes are studied with smart water meters.

1.3. Disclaimer for cooperation

This study lays the foundation for Dataservice software. The goal is to initialize a working software that can be implemented further. There is currently available another study that will display the user interface and front-end of the software. This study is done by Mr. Markus Heino [11]. In the future, the current project will feature more sophisticated methods for data analysis. As the implementation for the project has only begun, it is more reasonable to invest in software development. When development project reaches adequate state. A study will be conducted to find suitable machine learning (ML) algorithms and artificial intelligence (AI) tools.

2. STATE OF ART

Smart water meters have been available for Finnish waterworks for a few years now, even though not many of waterworks have utilized them fully. Smart water meters introduce a new digitalization opportunity for waterworks, namely, the Internet of Things (IoT). Finnish waterworks are not highly digitalized and the concept of IoT is just surfacing to their business field. Waterworks have just scratched the surface of the possibilities of IoT and digitalization.

2.1. IoT

The internet is coming more and more accessible around the world. Only the most remote places lack reception these days. This growing robustness of the internet network layer has been the basis for more devices to be connected to the internet, creating a phenomenon called Internet of Things (IoT).

2.1.1. Description

Internet of things (IoT) is a concept introduced by Kevin Ashton in 1999 [12]. IoT network means a network where different devices are connected and are communicating with each other. The devices are connected to the internet by embedding measurement sensors with network connection technology. The term IoT is fairly commonly linked to ubiquitous computing (ubicomp), as there are many inventions that require data traffic and other communications. It has been argued that IoT and cloud computing are the two main actuators of ubicomp [13]. IoT communications happen over the internet layer of communications, hence it has been called the internet of everything or industrial internet [14, 15].

2.1.2. Connections and protocols

Different IoT devices have many different use cases, so there are many different options on how they need to function, other devices use Bluetooth connections, still others require more reliable and long-distance communication methods [16, 17]. Probably, the most common form of IoT data transfer protocol for any pedestrian is Bluetooth, as it is widely used in modern devices. It is used in short distance wireless communications and it has grown a more common technology with smartphones [18, 19]. Even if Bluetooth is a versatile technology, it has its limits. Bluetooth has a short-range and is not the most stable connection. Also, the current rising trend in security awareness has shed light on many Bluetooth security risks [33].

An ISO standard for data transfer protocol that is currently used in many IoT devices and different networks is Message Queuing Telemetry Transport (MQTT). The MQTT works over the existing TCP/IP layer. The system has multiple nodes, which connect to a single broker. For Bluetooth connections, there has been created a variant for MQTT called MQTT-S. [20, 21]

Because water meters are placed inside or even underground, the communication medium must be strong enough to penetrate through walls and soil. Currently, many water meter manufacturers are using LoRaWan connections with smart water meters.

LoRaWan is a type of LPWAN networks, LPWAN stands for Low Power Wide area networking protocol. It has almost a 10km broadcasting range and inside the Europe region, the LoRaWan operates between 433-915 MHz**.** The LoRaWan has been explicitly developed for battery-powered devices to connect to the Internet of Things. Much like IEEE is managing the World Wide Web, the LoRa Alliance is managing the LoRaWan [22, 23].

It is possible to configure LoRaWan devices to use a MQTT-protocol. LoRaWan devices will need a common gateway that is using the MQTT broker. Devices are connected to a single gateway and the gateway will use MQTT to transfer data beyond the gateway. Using both MQTT and LoRaWan will isolate devices because they cannot see beyond the gateway. [24]

Digita Oy is one of the Finnish companies that provides a LoRaWan network [25]. Currently, Digita Oy is the biggest service provider in Finland, but most likely, other service providers will emerge in the future.

2.2. Big data

A database of the size of 1Tb cannot be described as a big data database. Even if the database has 100Tb, it may not be a big data database. Big data is not only defined by the sheer volume of the bits. Big data is a highly abstract concept with four identified characteristics: Volume, Variety, Velocity, and Value. These characteristics are known as the 4 V's [26]. The data is in massive volumes and it is not possible to process all of it at the same time within a reasonable time limit. When dealing with smart water meters, Dataservice does not need to handle all of the water meter readings and water meters at the same time.

The data is of a different variety. The different devices and sensors send different measurements, so the data is heterogeneous. Different data objects have different sizes and attributes. This creates a major dilemma for database designing.

The amount of data is increasing at a certain velocity. Smart IoT devices send requests periodically and as needed. The amount of data received raises with each IoT device connected to the system, so does the velocity of the data, as requests are piling up.

The ability to create value from the gathered data is the main goal of Dataservice. The value lies not only in the single data entry, but also in the bigger picture. With a large enough data pool, it is possible to analyze the data and possibly make minor predictions or projections of the real world. Some of these methods have been utilized already in smart farming, for example [27]. Another method of utilizing big data is to create a real-time model of the real world, this method has been studied in several cities [28].

2.3. Cloud services

When dealing with large quantities of data, it is not justified to use personal computers for computing. Nor is it cost-efficient to build and manage one's own server room to reach sufficient computing power. There are many international companies that provide Platform-as-a-Service (PaaS) or Infrastructure-as-a-Service (IaaS) options. Many international companies have developed a PaaS and IaaS cloud computing services, and the three most used ones are, Amazon web services (AWS), Microsoft Azure and Google Cloud. Their business model allows anyone to build

their software on their cloud servers and they provide the needed computing power for the software. This method is called cloud computing. [29, 30, 31]

Cloud computing also provides an easy way to ensure scalability and enables the software to become a distributed system. However, it is not just computing power that is needed from these vendors. Dataservice will require an increasing amount of disk space as well. As the databases grow larger and larger, it is mandatory to invest in the scalability of the database, as well as the software distribution. While using cloud vendors, it is just a trivial question of money and disk space can be scaled up [32]. Another benefit from cloud vendors is that the largest ones, Google, Amazon, and Microsoft, also provide a Machine Learning (ML) support on their respective platforms. [30, 34]

Vesitieto Oy has concluded that this Dataservice system will be uploaded to Microsoft Azure. The reason for choosing Azure was that Vesitieto Oy has already done software on Azure. Also, Azure allows the usage of MongoDB. MongoDB is a modern distributed document database that utilizes JSON-like documents as data sets [35]. It can be argued that there are better options for NoSQL database engines, like TimescaleDB, etc. which have faster performance. But in Dataservice, it is more benefiting to prioritize the database query speed over writing speed [36].

2.4. Meter manufacturers

With smart water meters, it will be possible to display even real-time data. But not just water meter reading data, modern smart water meters are also able to measure even more aspects, like pressure, flow speed and temperature [37, 38]. The diversity of sensors and models provides few challenges for Dataservice developers.

One of the biggest challenges for Dataservice is the heterogeneity of the smart water meters. There is no real standardization between different manufacturers, and as a consequence, each has its own designs. The heterogeneity provides a challenge because the receiving end needs to be able to handle every water meter manufacturer's data and import that data to the database. Handling different data structures is an elementary function for Dataservice, as the data needs to be stored in somewhat homogeneous data models. In the future, one optimistic goal for Dataservice could be the standardization of smart water meter models. The challenge of heterogeneity does not limit only to smart water meters, when other sensor types are added to the network, the challenge grows exponentially burdensome.

Currently, in Finland, there are not so many installed water meters which are connected to any network. There are traditional analog water meters, or the connected meters are not transferring data. Therefore, the data is not accessible to waterworks employees. Other than waterworks employees, the data can be useful to consumers, water meter manufacturers, and mechanics. The lack of data provides another challenge; Dataservice is not much of service when there is no data to analyze, of course. Luckily, the trend today is that many of new installed or changed water meters are capable of data transfer. Currently, Finnish waterworks are purchasing smart water meters from Axioma, Kamstrup, and other manufacturers and distributors.

2.5. Waterworks currently

Waterworks aim to be more productive, but more importantly, waterworks value their quality and reliability. The Waterworks' main goal is to provide clean water for the city and the people around them. This means their business is quality-critical more than quantity-critical. Waterworks want to manage their product quality. The aim of Dataservice is to ease managing the waterworks quality of service (QoS).

In Finland, waterworks are part of the public sector and are seen as the country's infrastructure. Thus, there is no competition between waterworks. The lack of competition leads to a lack of innovation. Currently, many Finnish waterworks are analyzing their yearly water consumptions and other values using well-composed excel sheets or even by paper. For example, a local waterworks of Oulu, Oulun Vesi, has begun converting their water meters to smart water meters in 2018. In contrast to Oulun Vesi, Oulun Energia had converted all of their water meters to smart water meters already in 2012 [39, 40].

It is uncertain why the waterworks industry is dragging on, instead of updating their tools. It certainly is not because of a lack of choice or tools. For example, a company called TaKaDu is providing a highly flexible grid visualization tool and it is already in use in some Finnish heating plants [41].

2.6. Waterworks regulations

The government has settled some laws for the waterworks quality of service (QoS). The laws ensure that every person has access to sustainable water sources and that the environmental ideologies are successful. The business model for waterworks is regulated with local laws and regulations. In Finland, public sector institutions are mandated with local regulations as well as country laws. These regulations could vary based on the city's local politics. These regulations deal with QoS, availability and other qualities for waterworks and pipe grids. QoS for waterworks roughly includes the actual quality of the water, as it needs to be constantly monitored for impurities or bacteria. The Finnish legislative administrator states that each Finnish person has a right to clean tap water. The water needs to be reasonably accessible with reasonable expenses. The first mandate of the water management act also states that the sanitation must be adequately implemented. [42]

2.6.1. European Union's regulations

Other than local governments, regulations can come from higher instances. The EU has enacted many different directives and regulations that need to be followed. The EU has not yet directed any mandates that precisely consider waterworks, but it has given a guideline or a blueprint for safeguarding Europe's water resources. [43]

Another directive, implemented by the EU, for information systems is recently actualized general data protection regulation (GDPR). GDPR was created by the EU in 2018 and all European or European localized web-applications or storage systems needed to be finalized accordingly by 25.5.2018. The GDPR was implemented for better internet data safety. The goal was to prevent unnecessary security breaches inside databases that hold personal data. Personal data is any piece of data that can be used to single out any real person or their family. In Finland, the GDPR did not cause

as many actions as in other countries, since our internet regulations were already covering this clausal. Also, the GDPR obligated the personal database owners to provide a statement of usage and reason for the stored personal data. [44]

Because Dataservice holds and processes personal data of natural persons, it is legally seen as a personal information database. Thus, Dataservice must uphold the relatable new GDPR. The GDPR dictates that the data gathered within the EU or European Economic Area (EEA) must stay within that geological area. Another thing Dataservice needs to fulfill is the data gathering agreement with the users. The data of any natural person must be pseudo-anonymized and any person has a right of access and a right of erasure for their data. Therefore, the system must be able to delete and retrieve data for a single water meter or a single person. [44].

3. REQUIREMENTS

Dataservice must be designed as a stand-alone service. One major function is to support Waterworks using the Vesitieto system, but the system is designed as a service. At first, the primary users for Dataservice are waterworks office employees, like customer support or invoicing. However, in the future, Dataservice will be accessible for many different organizations like water meter manufacturers, different data transfer service providers and even for government officials.

At this point in time, supporting the Vesitieto system is the main function for Dataservice from the business point of view. In the future, things will need to evolve further, as it is not financially sustainable to design a whole system just as a support for another system. Especially when both of the systems are designed in the same company.

3.1. Use cases

Similar to most large ISs, there are many different use cases, hence there should be implemented many user groups and user roles. However, a user viewing visualized data on their device is a typical use case for the data gathering applications. The aim of Dataservice as an application is to gather all of the consumption information currently received by the Vesitieto web-applications in one place. This centralized data gathering allows performing more complex data analytics, as there is more data available. And in most cases, more data means more predictability. Other use cases for Dataservice are data providing for invoicing, data storing and data gathering.

The most important use case for Dataservice is the gathering of device messages, this use case is displayed in Figure 2. An IoT device, a smart water meter, in this case, is sending its current reading through LoRaWan. The LoRaWan service provider carries the message to Dataservice API. The device manufacturer is identified from the message, as the manufacturer can use different message encodings in their device messages. With the manufacturer's decoder, it is possible to decode the message. The water meter needs to be identified from the message, to ensure data integrity. When the manufacturer or water meter could not be identified, the message must still be saved in the database. Therefore, the system features an unidentified messages table, where all problematic messages are stored.

This unidentified message storage ensures better data integrity, as the messages are not discarded in unexpected situations. These situations could happen when manufacturers or LoRaWan service providers change their data encoding or when the system suffers from bad internet quality. These unidentified messages must be identified somehow. With advanced machine learning algorithms, it is possible to automate the problem solving, but for now, the identifying is made by hand.

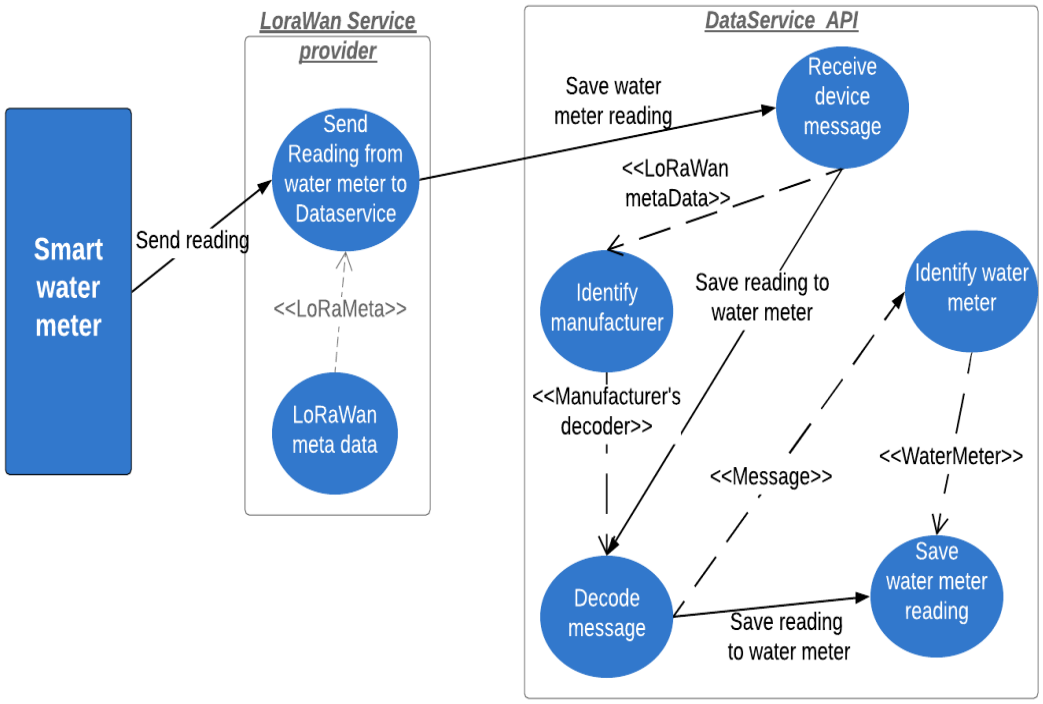

Figure 2: Receiving a device message.

Dataservice creates business value for stakeholders by helping the invoicing process, this use case is displayed in Figure 3 below. Dataservice will be integrated into Vesitieto invoicing system as a data provider. When an employee is invoicing a consumption place, he will need the installed water meter identification number, in order to identify the water meter. With the water meter number, the Vesitieto system can request water meters consumption data. This data is then utilized in the invoicing process. The invoicing itself is made in the Vesitieto system, but the consumption history for each water meter will be provided from Dataservice.

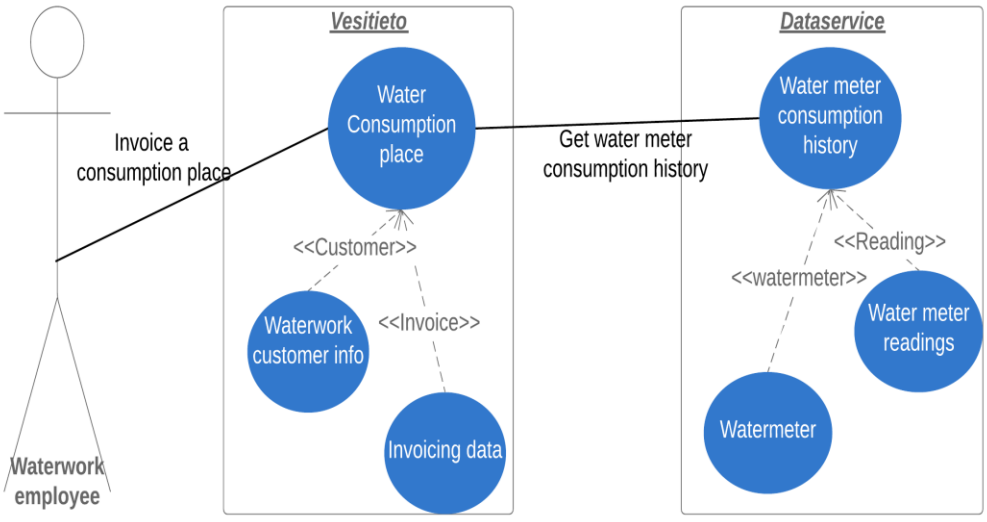

Figure 3: Invoicing use case.

Active management of numerous water meters requires easily accessible information of geological, historical and financial data. Geological data means that there is some sort of map interface for the user or at least some knowledge of the water meters' region. Historical data holds the water meter installation dates, the next scheduled meter change date, and different accuracy data. Financial data, on the other hand, tells a slightly different tale such as who has paid the bill for certain meters and who are the stakeholders for different water meters.

These measurements help the waterworks workers to deduce and plan the next area, where they need to change multiple water meters. The ideal scenario for Dataservice is that the user could inspect and plan the water meter changes by region. This regional change scheduling is a common procedure for many waterworks.

Most use cases described above are just for waterworks management purposes. However, the business model does not need to limit to waterworks. It can be argued that water meter manufacturers are also interested in their product's performance data. Thus, Dataservice must support data availability for manufacturers, as well as waterworks.

Dataservice has many different use cases for different actors. The use cases vary from the waterworks management to informing the local authorities if something is wrong in the water supplies. It is safe to assume that this many functionalities are not something that could be implemented overnight. Keeping this in mind, the scope of this study has been narrowed down to just one group of actors, the waterworks employees. The functionalities of the use cases have been limited to water meters. However, it is imperative to plan Dataservice according to future development.

3.2. Functionalities

As a part of digitalization Finnish waterworks, Allied ICT Finland (AIF) has started an undertaking with the title "Water Ecosystem - Vesilaitosten digitalisointia". In one of their workshops, the agenda was to identify different goals and hindrances for digitalization. These goals can be further generalized as four major requirement categories:

1. Business
2. Data
3. Data transfer and integration
4. Security

Each category has been identified and validated in the AIF workshop. These are the most critical aspects of any IS supplied to waterworks. By following these points, supplied IS has more ground to become successful. [45]

3.3. Business requirements

What makes Dataservice application successful from a business point of view? The system must obviously provide some profit or other financial value for the waterworks or organization. This is not always easy to pinpoint as there are many different ways to measure financial value. Financial value can be indirect, like positive headlines on newspapers, or physical investments in infrastructure.

Managing the waterworks infrastructure is just one way of creating business value, but other stakeholders have different needs. Waterworks human resources management has very different requirements and interests compared to grid management personnel. By creating business value for other stakeholders as well, acquiring Dataservice could be justified by waterworks. However, there are some overlapping interests as well. For example, the grid management and customer service workers might want to see the consumption places over a map. It is wise to design these overlapping interest points for both user groups.

The one quantitative attribute is the amount of money saved. Money can be saved by optimizing work methods or cut the amount of losses. When speaking of waterworks, leakages create losses. Dataservice should help the users to prevent or at least localize leakages inside their pipe grids. Managing the water grid can be seen as the primary object for waterworks, as it is the only way to keep their QoS at a mandatory level. Easing the grid management creates indirect value for the business, in the form of time and more productive work hours. Management easing could include something like visualizing the grid in real-time and highlighting the pipes that are behaving unnaturally. In the future, it could be possible to even display 3D-models or VR-models of the grids.

The most fatal error for the waterworks business model is when the quality of their service drops. A QoS drop can mean situations when there is no more water for consumption places, or the water is not usable for households. In the future, Dataservice aims to prevent these critical errors. Preventions can be done with risk assessments and periodical management. Dataservice will need highly sophisticated machine learning algorithms in order to be able to do these predictions. Making these predictions and assessments more accurate will allow Dataservice to have business value for any stakeholder. It is cheaper to prevent a catastrophe than fixing one.

3.4. Data requirements

Without data, Dataservice is nothing, but when there is data, the system must ensure that the data is correct and intact. Even correct and validated data is not always enough, as in some cases, data is time-sensitive. It makes a huge difference if the information about a leakage comes after 4 hours instead of 4 seconds. Dataservice needs multiple validation mechanisms to provide the most accurate and validated data with enough certainty.

For Dataservice application, different database engines have been studied [46, 47]. The studies revealed that for large heterogeneous datasets, an unstructured database schema is a more viable option [48, 49, 50]. However, it can be argued, that structured databases are quicker when dealing with smaller (not big data) sized data sets. Thus, the system architecture will be two-folded. The short time storage will be implemented with each data point as is, and the long-term storage will be created as NoSQL time-series data buckets. Time-series data means that instead of saving many single data points of an item, the system gathers one large “bucket” of individual points. This bucket is then saved on the database as a single item.

For example, the system needs to save a water meter reading once a day for a month. When using a normal database schema, at the end of a month, they would have 30 data points on their database. But if they would use a time-series data schema, they would have a single data point with 30 smaller data objects inside that

point. An example of time-series bucketing is presented in Figures 4 and 5, Figure 4 as non-bucketed data and Figure 5 as bucketed data.

The benefit of time-series bucketing lies in scalability in both ways. It is possible to scale the time-series as hourly or even minutely measured data. The same principles are true in every bucketing intervals. It is encouraged to include all smaller scale buckets inside the larger bucket. For example, if there is a bucket with 1-minute intervals, those buckets should be stored inside a bucket for the respective hour. Thus, the database has a continuum of linear data that is possible to unpack with highly specific timestamps.

However, time-series bucketing does have an issue, if a device is sending requests on deviating intervals, instead of predetermined intervals. In this case, the time buckets could have an excessive amount of empty data points. Consider the following example:

let ”b” be the time-series bucketing with time interval “t” and B is the bucket one layer above the first bucket with time interval T. If the first request comes at the start of the bucketing interval t=0, it can be stored inside the mentioned bucket. Now, if the next request comes at, t >= 2, the time bucket has no values inside the t=1 This has two different solutions. One is to just leave an empty value in t=1, but this fills the database with many empty values that are not usable. The other solution is to not add the t=1 bucket at all. By not adding the t=1 bucket, we only store actual data inside the database.

Neither of these solutions, however, can guarantee that the device is functioning properly. If the device has stopped sending requests, there is no data on when the device has sent the latest un-successful request. Because of this, the database must provide a feature where all of the income device requests are stored. This way, the data integrity will be held up. The request database only stores the incoming byte-array with some metadata to identify perhaps the sender and a timestamp for the request.

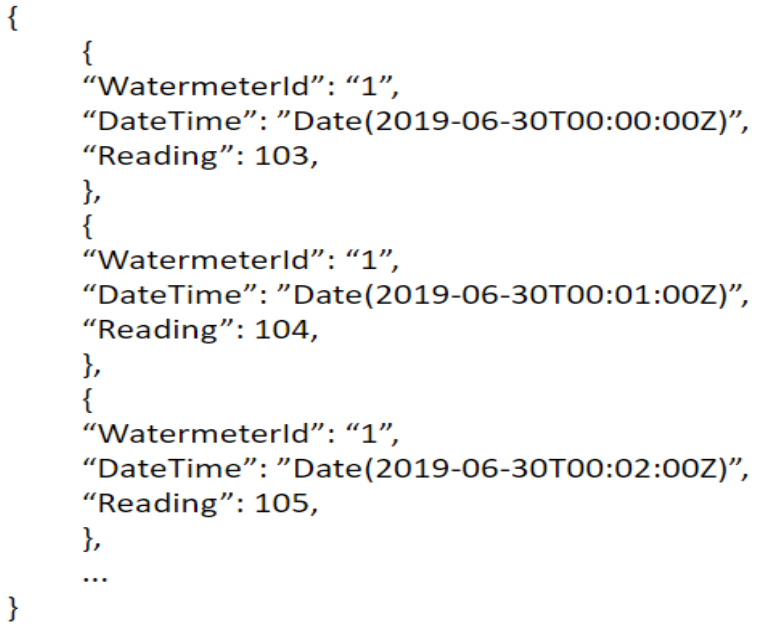

Figure 4: Each received request has its own data point.

```
{
    {
    "WatermeterId": "1",
    "DateTime": "Date(2019-06-30T00:00:00Z)",
    "Reading":{
        "0":103,
        "1":104,
            "2":105,
        ...
    }
}
```

Figure 5: Each hour has a single data point inside one bucket.

3.4.1. Long Term storage database and short storage database

After some time, it is inevitable that the gathered data is no longer relevant to see on a daily basis. Dataservice will implement long-term storage functionality as a part of the 4th category, data management. The point of this long-term storage is that the data is no longer used but is still stored and therefore it is not lost. However, the data will be stored in a bit different form. The long-term storage will implement time-series transformation. The transformation is done for each meter and on each month and day. This bucketing will decrease the amount of individual data points inside the table; thus, it also decreases the retrieve times [51, 52, 36] Along with decreasing retrieval times, the system will also implement long-term and short-term storage, division. The short -term storage will store new data and the long-term storage will store the rest of the data in time-series buckets. The division between the new and older data allows managing the costs of the storage. Long-term storage will be written on slower Hard Disk Drive (HDD) and the short-term storage will be stored on faster Solid-State Drive (SSD) [53]. In general, SSD costs more than HDD, thus it will be cheaper to store large amounts of old data in HDD. Short-term storage will be used to retrieve almost real-time and real-time data. In order to be able to provide real-time data, the data retrievals need to be as fast as possible.

3.4.2. Data analysis point-of-view

Just saving and holding the data is one of the requirements, but it does not yet yield anything of use. Dataservice must be able to do something useful with the data, otherwise, it is only a glorified file cabinet. Gathered quantitative data needs to be accessible for many different statistical or mathematical tools, coding languages, software, and engines, like Python, MATLAB or R. From these external tools, it is easier to draw conclusions or models, as these tools are designed for that purpose unlike the back-end of Dataservice. [54]

Using the latest techniques, it is possible to implement external and internal interfaces from Dataservice to data analysis tools and software. For example, it is possible to export certain data to excel sheets or other file formats, and then convert that file again to MATLAB or other similar software designed for analysis. If

necessary, it is even possible to implement one's own data analysis tools with external components. However, implementing own tools inside the system is not the most cost-efficient solution, as it requires multiple work hours depending on the project. Hence, it is wise to choose to implement interfaces cautiously.

3.5. Data transfer requirements

Our system must be able to handle multiple overlapping HTTP (Hypertext transfer protocol) requests at any time. Because of this, the API should be implemented using a stateless Representational State Transfer (REST) architecture. As a stateless service, the system does not save current session information and all relevant data comes with each request. By using a stateless REST API, Dataservice allows its API to serve multiple clients at the same time. [55]

Furthermore, as each request will be handled by itself, it prevents the data to be corrupted on mixed up. Still, it is imperative that the system handles the critical section, namely the database, smoothly and securely. Critical section locking can be handled through the Operating System (OS) or other underlying software. Usually, the database provider has already implemented these locks, but it is safe to implement the critical section carefully.

3.5.1. Receiving data

The biggest issue with data transfer is that the system needs to be able to receive even an unknown message inside the database. The system needs to be able to recover from any malfunctioning IoT device or requests. Handling any unknown message is not something that needs to function with all of the messages that comes knocking on the API's door. Filtering out unauthorized requests is the easiest way to handle junk from the web. After successful authorization, the system needs to decode the request message.

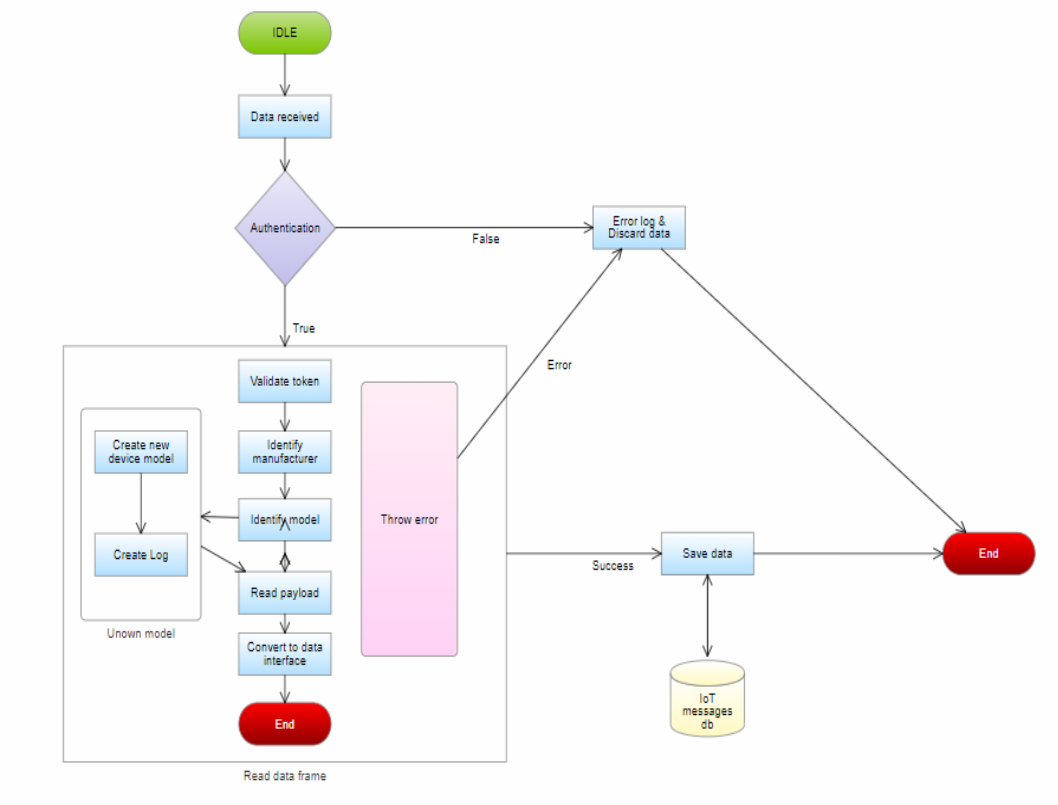

Figure 6: Flow chart of receiving an IoT device message.

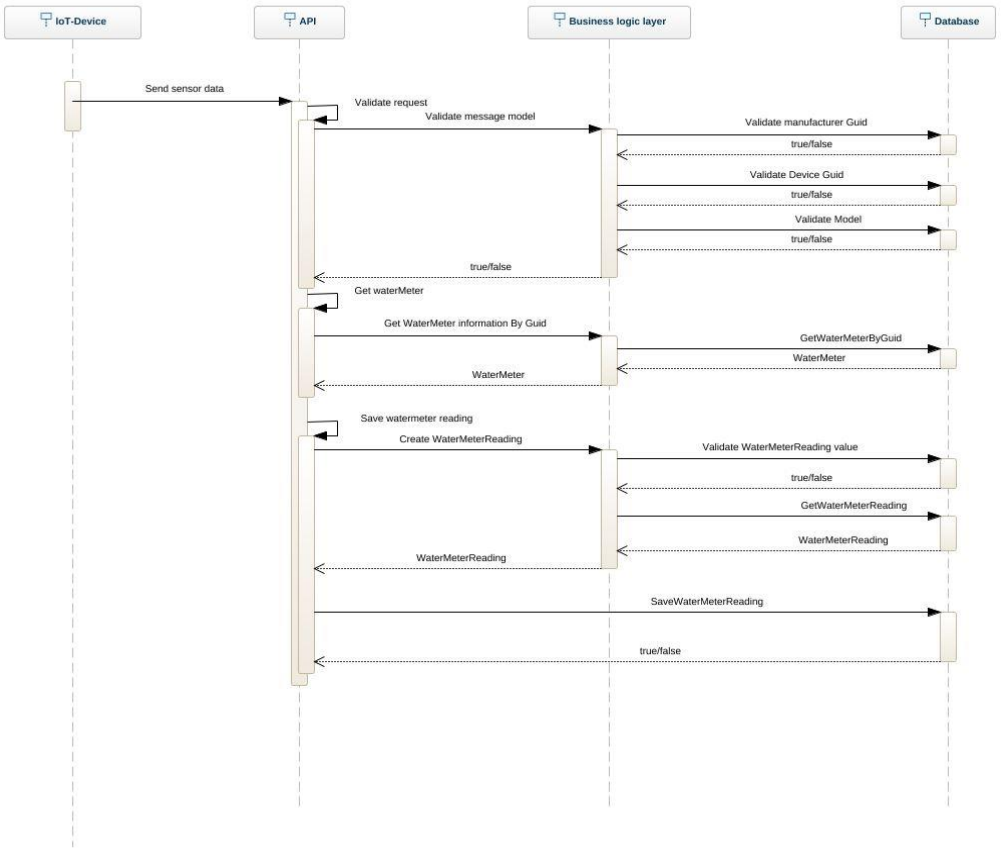

Figure 7: A sequence diagram of receiving an IoT device message.

A specified use case, receiving water meter reading, is displayed in Figure 2 above in Chapter 2. Additionally, a flow chart representing that use case is displayed in Figure 6 and a sequence diagram in Figure 7. All other IoT device messages follow more or less the same route.

Dataservice needs to be able to dynamically validate the connections and inputs. Connecting the message with the correct IoT device is mandatory to process. If the message cannot be connected with the correct device, the data cannot be ensured to be intact anymore. This will drastically decrease Dataservice QoS. As stated below, ensuring data integrity is one of the main requirements for the system.

3.6. Error handling

Nowadays, the internet is a highly reliable network with multiple error prevention protocols, like checksums, cyclic redundancy checks, etc. [79]. Even with these protocols, errors could occur in data transfers. Even if the error percentage is 0.01%, it means 100kb when transferring 1Gb of data. This is something to consider when dealing with big data data-transferring systems, as 1Gb of transferred data could potentially happen under reasonable short time constraints [21].

One of the problems handling many different class structures is that the data types and units can vary based on the sending end. For example, if one device uses a timestamp in milliseconds, and the other one a timestamp in ticks, the constructed date-object has different values, even if the timestamp is actually the same. This needs to be accounted for when creating API interfaces for different IoT devices.

These problem areas can be solved with adequate project planning and by designing and implementing a sufficient number of interfaces between systems. While implementing interfaces, it should be kept in mind that interfaces are not dynamic solutions. Any changes done to either the receiving or the requesting system requires changes in the interfaces as well.

The importance of error handling can be seen in Figure 6. It is apparent that error handling should be done excessively detailed. Wholesome error prevention and recording allow the system to preserve data integrity even when all messages are not saved correctly. An error could occur when there is too much data inside a message or when there is not enough data inside a message. When an error occurs, or the system does not recognize the device, the system must not discard the message. Instead, the system must be able to logically deduce what is the unknown part of the request and add a suitable fix. If no suitable fix is found, at least the system must save the received message for further inspecting and alert users that an error has occurred.

If message decoding cannot be done, the system can argue, with reasonable certainty, that the request has been corrupted on transfer or the received message source could not be authenticated. The system has no need for corrupted or unauthorized messages, and these can be discarded. However, it might be beneficial for maintenance to get a hold of these corrupted messages if they are recurring.

Arguably, even more important than error handling is the error logging. Suitable and descriptive log messages should always be created in every error situation. Excessive logging is the basis of updating and bug fixing of any system. As the web services are deployed on some remote server, there are not many straightforward methods for getting debug information out of the service. Arguably, the easiest way

is to generate error logs as errors occur. Logs can be stored inside the database or even text files, whatever works best with the current technology and architecture. Well-constructed logs help to identify any error-prone sections in the system and provide insight into the errors. As time goes by, the logging can help the developers to identify different error-prone sections and implement suitable fixes. However, there are some issues to be concerned about, implementing logging can take multiple work hours and finding a suitable logging method can be burdening. Also, as time goes by, the number of logs increases. It should be noted within the implementing that older log records can be removed or destroyed when they are outdated. This can be done manually, but it is less boresome to implement some form of a digital solution.

If, and when errors occur, the system should notify the corresponding user or users. The notifying should be instant as the error has not had the time to multiply or escalate yet. In the future, it is possible to implement suitable data validation algorithms and data correcting tools based on the previous data. The mathematical error-correcting could be utilized by an AI.

3.7. Integration and interface requirements

Dataservice's two most important integration interfaces are the one with waterworks current existing customer information system and the IoT input interface. It is elementary that our system must be able to handle the evolving needs of both of these techniques. The application programming interface (API) must be able to handle different inputs from many different IoT sensors and it must provide and support an unstructured dataset.

One of the major categories was integration into existing systems. It is imperative that different systems are working together instead of having multiple adjacent systems working independently. In a worst-case scenario, there are multiple systems gathering the same data multiple times. The implemented system must provide an API for waterworks current systems. This was one of the major categories identified by the AIF workshop [45]. This means that all implemented systems and functionalities should be able to exploit the existing systems and databases. The importance of integration rises from the volume of different systems that are currently overflooding waterworks employees. In a worst-case scenario, employees might have an individual system for each of their tasks and then there is also an application for monitoring working hours. Because of this surplus of applications and systems, Dataservice must provide a means to lessen the number of needed systems and not to become one on top of the pile.

If Dataservice only handles the IoT devices and their messages, the data analysis will be arguably insufficient. As there are many different users using Dataservice, they have different use cases. For example, a waterworks chief executive officer (CEO) does not need to see the most accurate volumes for each different water meter specified by date.

CEO wants to see the bigger picture and probably they want to use currency as units instead of cubic meters or liters. A customer service employee instead wants to see the customer's previous invoices, current consumption and maybe any future scheduled tasks going for a single customer or few customers living geologically close to each other. These kinds of specified needs raise a growing need for specified

data analysis, but it is not possible to provide the needed data if there is not enough data to analyze.

Data analysis can be improved by adding some existing data from other systems. These can be anything from customer data or invoicing data to work management data. With additional data, the analysis can be done for many different user groups and many different use cases. Successful integration with existing systems also benefits Dataservice as a business.

It is much more easily accepted if the system does not require any data-conversions before the system can be utilized fully. Currently, migrating from a legacy IS to newer IS is burdensome because of the data conversions. According to Vesitieto, each waterworks client has their own scheduled implementation project. And in that schedule, the conversions take at least 50% of the complete schedule. Even if the new IS has been already completed and tested, the conversions take a long time in order to function properly.

Integrating with other systems can be time-consuming and stressful for a developing point of view. Existing system providers could be less than enthusiastic to open their projects. Even with open relationships, the integrations usually take multiple work hours of planning, designing and implementing specific interfaces.

3.8. Security requirements

The system must be adequately secured as the database is classified as a personal information database and Dataservice will be used for invoicing. Thus, it is imperative to ensure data integrity and validation. It is crucial for the operation of Dataservice to continuously be able to provide reliable and truthful data. Because Dataservice is used for invoicing, the system must prevent any tampering of the water meter readings or any modifications in water consumption. This roughly means that maleficent outside force may not be able to tamper with saved data inside or outside the system.

The Dataservice IS consists of the web-service and the IoT devices connected to it. Thus, IoT devices are a relevant part of the system and therefore part of the security designing [56]. However, in the scope of this study, the IoT device and network provider security could be argued to be seen as adequate or at least sufficient. Many taxonomies and approaches for IoT devices have been researched more comprehensively in other studies [57, 58], 59]. Even with embedded security functions, Dataservice should never just blindly accept the incoming requests as valid and secured data.

3.8.1. Client authentication

As far as Dataservice is concerned, the IS security designing needs to be designed to prevent any unauthorized users to gain access to the system or the data stored in the database. Dataservice needs to fulfill all of the basic security requirements for any web-applications may have, like user identification, client privacy, secure data communications, and identity management. Additional to user handling, the system needs to be resilient to outside attacks, both from the API and from the UI. Because Dataservice will be hosted from Microsoft Azure hosting service, the communication, and network security as well as system availability can be argued to

be fulfilled by the service provider. One reason for hosting Dataservice from Microsoft Azure, or another cloud system provider, for that matter, is that they ensure a secure execution environment for the system. Any tampering of IoT devices or other physical security risks will be tackled through authentications and embedded data integrity protocols. [60]

The most critical point of security is database securing. The system needs to be able to provide successful data authentication, secure content and tamper resistance. In order to provide a secured database, the system adopts client authorization filters on all API endpoints. Authorization filtering will be implemented on both IoT devices and end-users.

User authentication ensures that only registered and allowed users can receive or modify the stored data. Requests that only receive the data can be noticeable more liberal but when someone or something is adding, modifying or deleting data, the requests need to be thoroughly inspected and verified. In order to filter out the unverified or unauthorized requests, the system should also implement some sort of user grouping policy. Generating user groups with access rights or licenses will be the key aspect of Dataservice's user filtering.

After successfully filtering unauthorized clients and clients without adequate access-level, the data modification request can be inspected further. The first thing to verify on each modify is that the data is valid. With IoT devices, the requests include a hashed token string which can be used in message validation. Token strings effectively hinder the man-in-the-middle type of attacks and package-loss errors [61]. The token is generated in the IoT device using the data inside the request and the receiving end can verify the message payload as a valid message. Of course, token strings are just one method to achieve validation, but as long as the secret hashing seed is stored securely, the token is an effective validation method. [62]

4. IMPLEMENTATION

Implementation of Dataservice started adjacent to this study. The project development started with designing as the needed state of the art study had concluded. The first designing and requirements gathering was done by Vesitieto collaborating with the AIF workshop and local waterworks [45]. The implementation phase was done following the Agile software development method [63].

4.1. Architecture

Many IoT technology providers have created their own reference architectures for IoT-systems [64]. The Dataservice architecture is greatly derived from Microsoft's and MongoDB's reference architectures. The studied references did not limit to these two, but the structure has been based on the two most similar architectures. Microsoft's architecture is presented in Figure 8 and MongoDB's version can be seen in Figure 9. When studied, these two architecture options were seen as similar and the most useful in this case, as the implementation of this Vesitieto Dataservice will be done using these technologies. Studying these architectures, it is obvious that there are many common denominators in both reference architectures. [50, 65]

Data services architecture follows both of these reference architectures because the architecture has been designed based on these two. The architecture was designed by studying reference architectures and then identifying common denominators on each architecture. These common aspects were gathered in Figure 10 as the Dataservice reference architecture.

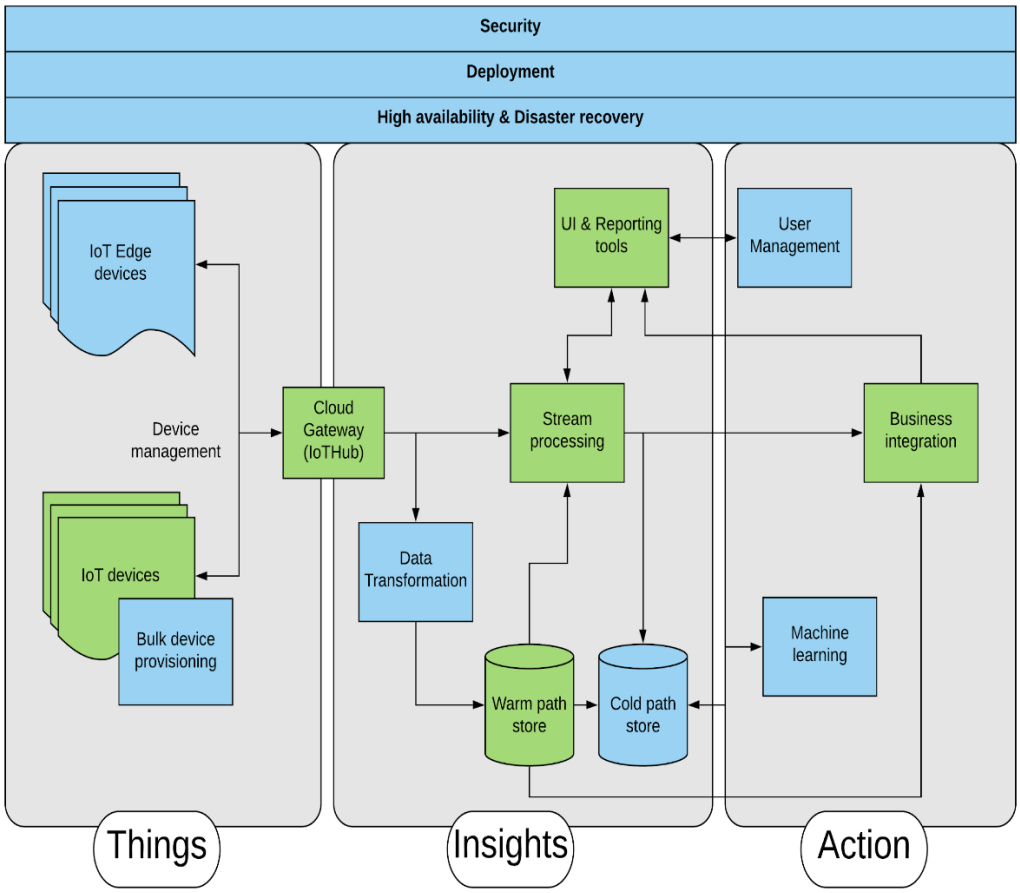

Figure 8: Microsoft Azure IoT Reference architecture.

In Microsoft's presentation, there are three entities present, "Things", "Insights" and "Action". These entities are encased within the Microsoft Azure. As the system has been deployed to the Azure cloud, the system can be seen as a distributed system. This provides the system the enhanced availability and a high error recovery rate. Arguably distributing the system leads to many different problems like timing, acknowledgments and data integrity [66]. It can be argued that these things will be handled by the cloud provider, but it is still something to keep in mind. Furthermore, this argument can be extended to security as well. If a system is deployed to the cloud, the cloud provider needs to provide adequate security for their platform. Even with platform security, it is still elementary to design security mechanisms for the system. Especially in Dataservice, where data integrity and validity are business-critical.

As stated above, the architecture consists of three entities. The first entity, "Things", is the IoT devices, software and hardware included. The IoT data transfer can be implemented by using any applicable methods and is not the actual focus point for the study. Because of that, IoT devices can be seen as a black-box entity, where data is measured and then transferred correctly. The reason for this black-box ideology comes from the amount of comprehensive, extensive and definite studies already conducted in this field. [67, 68, 69]

For systems other than Dataservice, the data transfer is more heterogeneous as the Vesitieto Dataservice relies on a third-party data transfer provider. This third-party data transfer is a purely optional method for data transfer, but in this specific case, it

is more beneficial to rely on others than implementing own data transfer methods. This enhances the system's data transfer security and data integrity because the gateway only takes the IoT devices messages and transfers them directly to the Dataservice API. Dataservice will treat all IoT devices the same as the data transfer comes through the same data transfer provider cloud gateway, ensuring more consistent data handling.

Because of the third-party data transferring and network, the data traffic is restricted as there are fewer clients connected to the network of choice. This, in its turn, improves security and data integrity. Security improvement can be justified as there are fewer potential threats or maleficent clients connected to the network and therefore there are fewer potential threats available. On the other hand, if there is a maleficent client connected to the restricted network, the client probably possesses more threats or skills than an average client in an open network. Additionally, the same security concerns as in cloud deployment are important here as well: the third-party network has, or should have, implemented its own security layers in their service.

The second entity, "Insights" holds the key elements for this architecture. As the data arrives through gateways, the system should analyze and standardize the data into suitable models. These models can be then used in reporting, processing, and in UI.

As described above, the Dataservice data storage is two-folded SQL and NoSQL storage. The NoSQL storage is also divided into two, the long-term storage, or cold path storage, and the short-term storage, or warm path storage. The long-term storage holds the bucketed time-series and the short-term storage the more recent and live datasets. This method of implementation allows live data inspecting with more accuracy. The benefits of the architecture model lie in warm and cold storage. The smaller size of the warm storage enables high-speed transactions to warm storage, thus making the web service more user-friendly and the data retrievals take less time. The separation between warm and cold storage creates more cost-efficient services. The warm storage needs a costly high-speed disk space, but the cold storage can be stored in inexpensive slower disk space. This decreases the system's upkeep costs.

The third entity "Action" holds the actual data analysis and reporting. Actions utilize the data gathered by the insight entity and yields a human-readable, analyzed data instead of raw data. This processed data is the main business enabler of Dataservice. It is elementary that the analysis, validation, and processing of the raw data is kept under control and implemented with vigorous testing. Integrating different analysis software, like excel-sheets or MATLAB, or even data analysis libraries must be allowed. Possible integration interfaces are not necessary before the need arises, but the system must be able to provide the needed interfaces when the time comes.

Other than external data analysis tools and methods, the system could be integrated with ML algorithms, Implementing and integrating the ML is also something that is not necessary, nor crucial, from the beginning, but it is always wise to plan ahead.

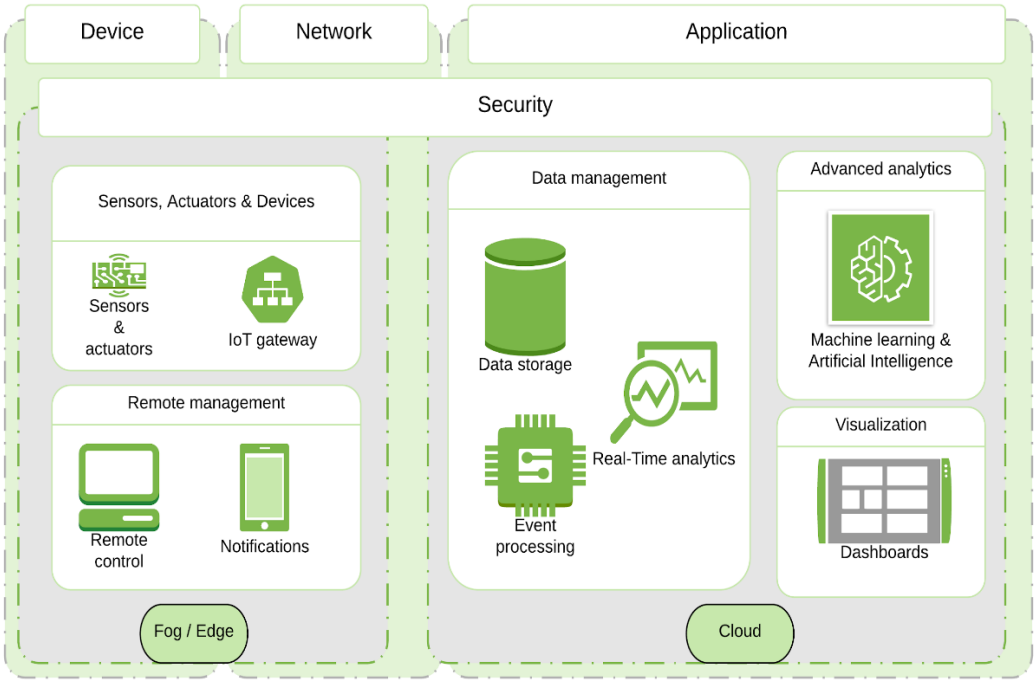

Figure 9: MongoDB IoT reference architecture.

MongoDB's option has almost identical parts as the Microsoft. This arguably helped the process of identifying common denominators. It has the same three entities encapsulated within the application security methods. But unlike the Microsoft reference architecture, MongoDB does not implement warm and cold storage separation. Another difference between the two references lies in integrations. The MongoDB architecture does not explicitly display any integrations, but studying the Mongo cloud service, the system was able to integrate with their own data analysis tools.

In the MongoDB's solution, the IoT device management is included in the "Things" entity as well. This is a noteworthy insight as the devices also need physical management and not just digital configuration. Planning the device management should be also noted in the Dataservice architecture as well. For example, the management can be eased by adding GPS coordinates for each device to locate the devices. Other methods can be estimated service intervals, photos of the installation and notes of the device.

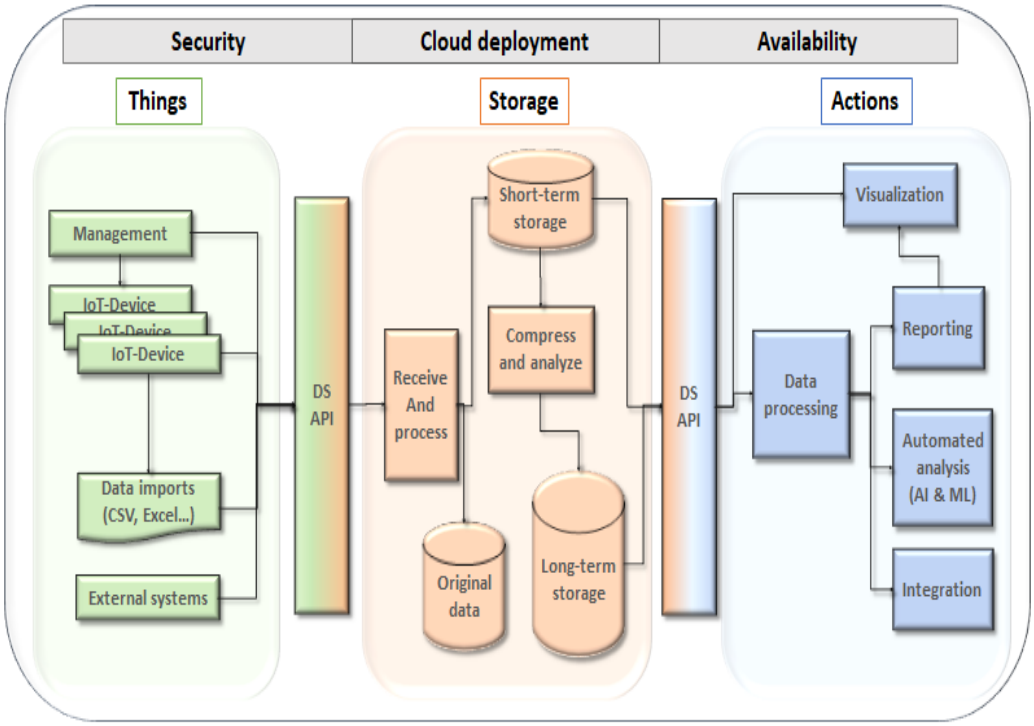

Figure 10: Dataservice reference architecture.

In Figure 10, is presented the Dataservice reference architecture included with the same three entities as both of the previous architectures. The “Things” entity holds all of the data gathering methods and device management. Devices, smart water meters, are connected to the system through IoT network, LoRaWan, thus able to transfer data to the Dataservice API (DS API). The “Storage” system receives, validates and processes the incoming data streams and stores the validated data to short-term storage. Additionally, the data streams can be stored in the original data storage as is in byte format, so that the data can be validated even after some time. Because the short-term storage is designed to hold only the most recent data and it uses only fast disk space, the Short-term storage can also be seen as live data storage. Live data streaming is arguably overkilling in most cases when speaking of smart water meters and water consumption, but in the future, the use case for live data could arise. In order to save the more costly short-term storage space, the data must be transferred to long-term storage. Allocating to the long-term storage happens through time-series bucketing functions described above.

Dataservice API yields an open interface for the Dataservice web-application. Through the client application, the data can be visualized and analyzed further. Client app will be the main portal for Dataservice users. Through the client application, users can generate reports and even operate automated analysis tools. In the future, it could be possible even to automatically generate simple AIs to specific tasks through the client application. In addition to reporting and visualization, the Dataservice API should provide an interface for data exports. Exporting via different formats is one of the integration requirements described, but it is not yet apparent what are the used mediums or formats. Possibly, in the future, the data can be retrieved automatically through the API itself and no export mediums are needed.

4.2. Database

Implementing and designing a database is the most crucial aspect of the system. The first decided attribute for Dataservice was the database schemas. As the Dataservice operation time is aimed to be many years, we needed to consider the long-term storage and implement the database accordingly. It is mandatory for the system to be vertically scalable, and it needs to be able to adapt to the real-world changing requirements.

4.2.1. SQL or NoSQL

Structured Query Language (SQL) is a database query language, or more accurately, it is the syntax that has taken over. It relies on well-structured database relations and interacts through those related schemas. SQL can be used in most common database engines as is. [70]

NoSQL means, as the name implies, quite literally not only SQL database schema. The basic idea is that the management system or the database engine does not care what is inside the tables. NoSQL databases have few key features, but for this system, arguably the most important is the ability to dynamically add new attributes to datasets. [71] Dynamic adding allows the system to evolve, adapt and overcome the different requirements and payloads from different devices. This ensures the continuous evolvement of the system [50].

Because Dataservice has both static classes, like users and user groups, it benefits from using the traditionally structured data relations. However, the unstructured and possible evolving structure for IoT devices dictates that the system utilizes a non-relative database schema. This dilemma can be compromised by using a two-folded database. In this case, the more static and permanent data structures are stored inside the SQL database and the more dynamic structures inside the NoSQL database. This two-fold schema allows pinpointing the best methods and functionalities from both, SQL and NoSQL schemas. The system could have been implemented using only the other SQL or NoSQL database schemas, but it was deemed more beneficial to store the sensor models inside the NoSQL and the more static models inside the SQL. [72]

Using SQL for the static models enables the system to use more configurable database queries and SQL standards. Using a relational database allows the database to be normalized and distributed if necessary. Also, the relational database schema ensures there is no duplicate data, thus increasing the data accuracy and integrity. The relational database supports complex queries and provides multiple users to access the database at the same time. [72]

Arguably two-folding the database also doubles the upkeep for the system. The development team needs to be able to design, implement and update both databases. It is also noted that developing a SQL database and a NoSQL database are not similar processes, this could mean that the developers need to invest more time over the other implementation. Developers might have a better knowledge of other structures than the other and this could lead potentially to some avoidable security issues. Two databases mean that the maintenance workload doubles as well. Backups need to be made on both databases and any integrity tests need to be converted to others as well.

4.2.2. Database design

As stated above, the development project follows an agile spiral methodology. This can be witnessed, especially in database designing. The designing begins by selecting the database schema. In this case, Dataservice will have a two-fold database system; the system data will be inside the SQL database and the IoT data will be inside the NoSQL database. System data includes any objects that have mostly constant relations with each other, like users, user groups and users' rights inside the system. IoT data, on the other hand, will include all of the different sensors, like water meters or temperature sensors, as well as the IoT devices communication messages. These heterogeneous sensor models are more suitable for the NoSQL database.

With SQL relational databases, the key aspect of the database implementation and design is to plan ahead. The database needs to provide an unhindered base for future development. It is possible to change database schemas after the first design, but it could be burdensome work, especially if the system is in use and needs migrations. Thus, it is beneficial to plan ahead while designing the database schema and be prepared for future development. For example, Dataservice will implement a dynamic language option. Because of this, the language option has been already designed into the database schema even though the functionality has not been developed. With SQL databases, it is also beneficial to optimize the database. Optimizing can be done manually or with external tools [73, 74 p.501-536].

With NoSQL databases, the designing is less time-consuming, as there are no relations that need optimizing. However, it is possible to optimize the queries and the models inside the database. More important than optimizing, is choosing the right tool for the system. Different NoSQL database engines have different use cases. Different database engines are more suitable for different datasets. [75]

4.3. API

As stated above, the goal of this Dataservice is to function as a data distribution center for many different systems. Therefore, Dataservice must provide a stable, secure and straightforward application programming interface (API). An API usually provides, as the name implies, an interface for data management over the Internet. The main functions for API are: GET, POST, PUT, DELETE and PATCH. These are also known as the HTTP (Hypertext transfer protocol) request headers or CRUD functions (Create, Read, Update, and Delete). The most important objective of the API is to be an interface between the client and the storage. The API will provide CRUD functions for all of the needed objects. [55]

Another important aspect to implement and the document is the HTTP response status codes. The status codes are described in Table 1 below. These status codes are defined in HTTP standard RFC2116 [76]. The standards allow the API and the client to work together and recover from errors. The status codes allow the server to send its response to the correct path for the client to receive it. For example, if an error occurs in the client, the response status code will be 400 or any other number below 499. Perhaps, the most known, and irritating, status code is the error code 404, "Not Found". It means that the requested resource was not found on the server.

Categorizing the status codes benefits the systems to be prepared for different situations. A well-implemented system should be able to handle all of these scenarios. Usually, it is not efficient to implement a handler for every exact status

code. The error codes are the most used ones and need the most handling, but the informative codes are not as necessary for the client to recognize.

Table 1: HTTP response status codes [55]

Code	Category	Description
1xx	Information	Transfer layer information
2xx	Success	Successful request
3xx	Redirection	The client must take action to complete the request
4xx	Error on the client-side	An error has occurred on the client
5xx	Error on the server-side	An error has occurred on the server

One of the most important aspects of API implementation and objectives is that the API acts as a gateway for the client and the data storage. The API should be open for all authorized users using the client application, but no other user may receive data. However, the API should be able to send and receive requests from different existing systems and devices as well as users. This versatility of different requesting endpoint creates an inconvenience with the authorization and security. It is possible to reject all but authenticated or predetermined requests, but every time some new entity wants to join Dataservice, the entity, the API or in worst cases both need more implementing and testing.

The challenges in API implementation lies in balancing with security and functionality: the more open request routes, the more potential security threats, the more allowed requests, the more potential security threats. Balancing these threats and risks could be the key element for the success of the system.

4.4. Testing

Testing an IS can be done with three different testing techniques, i.e. white box, grey box and black box testing [77]. White box testing focuses on the internal testing of the system. In white-box testing, the tester does not care if the output is correct or not and has no knowledge of the bigger picture of the IS. Black box testing, on the other hand only, sees them as one unit that magically generates output from inputs. Grey box testing, in its turn, tries to adapt the best of both worlds. The tester has limited knowledge of the internal logic and the IS as a whole. Dataservice will most likely utilize both white box and grey box testing techniques.

One thing that needs to be excessively tested is the API and IoT message gathering. For this purpose, a virtual pseudo meter was created. It sends a genuine packet over the internet in the same packet as any smart water meter would send, but

it generates pseudo data. This testing meter was created to test two things, first, the actual data receiving, and second, to test live data streaming.

As Dataservice implementation is not completed while this study has been concluded, the testing results are not visible here. Even so, the system will be implemented as a code-first system and each functionality will include testing functions as well. Currently, there are many external libraries suitable for excessive functional testing, as well as usability tests [78].

4.5. Evaluation

As described earlier in this study, the main requirement for Dataservice is that it can function independently as well as a support for Vesitieto invoicing IS. The system was evaluated as a whole based on both the front-end and back-end testing. Those tests resulted in some different qualitative results. Analysis of the results yielded a confirmation for the requirements of Dataservice.

A comparative analysis and evaluation should include the same four aspects that the requirements; business, data, data transfer, and security. These evaluation aspects should be handled individually as there are different goals to fulfill in each aspect. However, the evaluation could be accomplished as a single proof of concept demonstration to client waterworks. Waterworks usually, like most companies and organizations, publish their annual reports. One way for evaluating and verifying Dataservice could be generating this annual report.

Even if the direct measurements are not easy to conduct, the amount of saved money can be witnessed by comparing the current business outcome with previous annual outcomes. Generating the business results for the annual report demonstrates the business value for the system, as well as the data integration aspect. Calculating the business outcome can be seen as a quantitative evaluation milestone.

Validating stored data can be done either with manual testing or through implemented testing services. The goal is to proof within reason that the stored data is valid. The stored data can be deemed valid if any new or old data set can be read, added, updated and deleted correctly.

Evaluating data transferring and integrations is easily demonstrated by testing the connections between external systems and interfaces. One evaluation goal could be achieved by generating an extensive consumption report for the waterworks. The report could include the geological breakdown for consumption, as well as whole consumption.

Evaluating the security aspect is generally challenging as the concept of a secure system is abstract. The security evaluation can be done by creating a risk assessment table and to predetermine a suitable course of action for the most critical risks.

Dataservice will be presented for the target audience when the first iteration of the software is completed. This early access will allow the actual users to influence on the functionalities of the system. Engaging users early on the development allows the users to develop certain connectivity with the software even before the completed product. This enhances Dataservice change of success. Arguably, there is a danger of the project to grow unexpectedly large and complex if the user requirements are not controlled. Every functionality should be justified instead of just blindly accepted.

Further analyzing Dataservice as an information system, it is too early to tell if Dataservice has been successful or not. ISS is not something that can be quantitatively recorded or measured [9]. However, it is apparent that this kind of

service is needed as more and more smart water meters are installed and connected to the IoT network. As the IS is developed and the first users are adopting the system, it is important to enhance the user experience (UX). User experience can be described as the will to use the system and the feeling the user is experiencing while interacting with the system. UX can be improved with a better understanding of the user and designing the user interface (UI) to match the user's expectations. Even though UX is a concept more targeted to the client and UI, it is beneficial to design the API and data retrieval times to suit the UX. User experience and UI could be evaluated using either qualitative or quantitative methods.

5. DISCUSSION

As Dataservice is still under the implementation phase, it is challenging to evaluate its success as an information system. However, in the scope of this study, the current stage of implementation looks promising. It is difficult to foresee the users' attitudes towards the system. In order to ensure the successful adaption for the system, the project included multiple personnel from different waterworks.

5.1. Requirements

Gathering the four main requirements categories was done in the AIF workshop and by interviewing two employees of Oulun Vesi. This insightful interaction with local waterworks was deemed necessary as waterworks employees are the main user group for Dataservice. The workshop yielded the four categories, and these were later verified with the interview.

It can be argued that collecting the requirements in such a narrow user pool is hindering, but the waterworks industry can be generalized in a few key aspects. Keep the water clean, transfer the water to consumption places and transfer the wastewater to purification plants. This generalized way of thinking enables the requirements to be globalized to suit many waterworks. However, it is imperative to keep in mind that even the same work processes may have different sub-processes included. These sub-processes may vary between the waterworks and the implementation should not hinder or prevent these sub-processes.

5.2. Implementation

The implementation process yielded the reference architecture displayed in Figure 10. The architecture was divided into three; Things, Storage and Action. Things meant the IoT systems and data gateways, Storage is the actual data receiving, storing and providing. Storage means the analysis of stored data. In the implementation, the main focus was on developing a well-structured two-folded data storage system.

The storage was divided into two, the relational database holds the more static system management data, like users or system logs. The IoT data will be stored in a NoSQL non-relational database. This allows the system to store deviating data structures in the same database table. The NoSQL database has been divided into two sections as well, the cold data storage and the warm data storage. Warm, or short-term storage, holds more recent data or even the real-time data. Long-term storage on the other hand stores the rest of the data. Short-term storage will be utilizing a faster disk-speed than long-term storage. The conversion from short- to long-term storage will be fully utilized in the future.

5.3. Evaluation

The evaluation of Dataservice's first iteration did not yet yield any concrete evidence. The evaluation was done by showing a demo for different waterworks and smart water meter distributors. The audience consisted of users from different

designed user groups. One of the target audiences was the local waterworks employees. The software was presented to them briefly followed by an interview. The interview yielded a few improvements and ideas for future development. The first iteration was then improved and modified based on the customers' feedback. Even though no specific result was not found, nor expected, for that matter, the general mood for the system was accepting and enthusiastic.

5.4. Improvements

The second iteration of Dataservice will be presented to many different organizations dealing with smart water meters. Different meter manufacturers, network service providers, waterworks and maybe even some government agencies will be interviewed, and their insights are taken into consideration. The most valuable insight, however, arguably comes from waterworks employees, as they are the main user group for Dataservice.

In the future, Dataservice might grow to even more versatile at data management. And it might be distributed abroad and for many other business fields besides smart water meters, like smart heating and smart electricity.

As for the study of the chosen database engines, the previous researches on the subject was deemed adequate. In the scope of this study, the database engines were only studied through state of the art. It can be argued that it would be more precise to conduct one’s own research on the subject. However, inspecting the state of the art ended in a reasonably justified ending. The same conclusion can be drawn on the cloud vendors and state of the art.

5.5. Future development

As the actual implementation of Dataservice first iteration is under development, there are still many open questions. The study cannot yet predict the volume, variety, value or velocity of the incoming data. This creates an interesting dilemma for future research. How to estimate the incoming data traffic and how to prepare for it from the beginning. Another point of view for this problem could be how to configure ones’s cloud service for big data traffic.

This study did yield a reference architecture and high-level requirements for the smart water meter data gathering system. Future research could include more precise implementation and requirements for smart water meter data gathering services designed for waterworks.

As stated above, the state-of-the-art review on IoT devices and networks was deemed adequate in the scope of this study as such Future research could include the IoT device manufacturers and network providers in the implementation as well. An interdisciplinary approach for the problem could provide more comprehensive insights as well as new research proposals.

6. CONCLUSION

Finnish waterworks are significantly lacking digitized work methods. Although FIWA has organized many multiorganizational projects to correct this deficiency, the waterworks industry has not yet adopted the solutions available. One of the reasons for this slow pace arguably lies in non-profit structures. Since many of the Finnish waterworks are funded by the government, their business is not focused on creating revenue or maximizing profits. Their main goal is to keep producing the freshwater. Arguably, the lack of competition and demand is hindering the waterworks digitalization process.

This study aimed to provide keen insight into the IoT data gathering system, Dataservice. The study includes a set of requirements for the back-end of the system, not including the user interface nor the IoT devices.

In this study are represented the key requirements for a smart water meter data gathering service. These requirements were gathered by studying the state of art and with qualitative analysis of the AIF water ecosystems workshop. Different reference architectures, cloud service providers and database engines were inspected to provide an efficient combination. The AIF workshop provided the needed insight for the system requirements. The found requirements were categorized as the following:

1. Business requirements
2. Data and data integrity requirements
3. Data transfer and integration requirements
4. Security requirements

Following these categorized requirements, a reference architecture was designed. The designed architecture was then refined as a software project by Suomen Vesitieto Oy. The project, Dataservice, started in summer 2019 and it was continuously implemented during and after this study. The implemented Dataservice will be a SaaS web-service and it will be sold to different Finnish waterworks. Dataservice version 1.0 will be completed somewhere in the late 2020. It will feature a smart water meter data gathering and minor data analysis tools.

In the future, Dataservice could be provided even for local authorities for legal and management purposes, or smart water meter manufacturers as a tool to manage and inspect the performance of their products. Further improvements also include opening communications for other sensors and IoT devices, as well as smart water meters.

7. REFERENCES

[1] Vesilaitosyhdistys. URL: https://www.vvy.fi/ Accessed 5.8.2019.

[2] Goldstein H. (2010) Malta's Smart Grid Solution. URL: https://spectrum.ieee.org/energy/environment/maltas-smart-grid-solution Accessed 5.8.2019.

[3] Beal C. D. & Flynn J. (2015) Toward the digital water age: Survey and case studies of Australian water utility smart-metering programs. Utilities Policy, 32, 29-37.

[4] Gassert F., Luck M., Landis M., Reig P. & Shiao T. (2014) Aqueduct global maps 2.1: Constructing decision-relevant global water risk indicators. World Resources Institute, 31.

[5] Oracle. (2009) Smart Metering for Water Utilities. URL: http://www.oracle.com/us/industries/utilities/046596.pdf Accessed 5.8.2019.

[6] Saarelainen A. (2016) Vesimittarit liittyvät esineiden internetiin Helsingissä. URL: https://www.tivi.fi/uutiset/vesimittarit-liittyvat-esineiden-internetiin-helsingissa/d054ba46-40ab-3f6c-bc89-1e362b4676cf Accessed 5.8.2019.

[7] Katja Solla. (2014) Tuhannet asuntokohtaiset vesimittarit raksuttavat tyhjää. Yle. URL: https://yle.fi/aihe/artikkeli/2014/01/29/vesimittarien-kulut-syovat-rahalliset-hyodyt Accessed 5.8.2019.

[8] Suomen Vesitieto Oy. URL: https://vesitieto.fi/ Accessed 5.8.2019.

[9] DeLone W. H. & McLean E. R. (1992) Information systems success: The quest for the dependent variable. Information systems research, 3(1): 60-95.

[10] Alter S. (2008). Defining information systems as work systems: implications for the IS field. European Journal of Information Systems, 17(5): 448-469.

[11] Heino, M. (2019) Tiedon hyödyntäminen ja visualisointi Datapalvelussa. (Draft manuscript). Master's thesis, University of Oulu, Department of Computer Science and Engineering.

[12] Ashton K. (2009) That 'internet of things' thing. RFID journal, 22(7): 97-114.

[13] Gubbi J., Buyya R., Marusic S. & Palaniswami M. (2013) Internet of Things (IoT): A vision, architectural elements, and future directions. Future generation computer systems, 29(7), 1645-1660.

[14] Lee I. & Lee K. (2015) The Internet of Things (IoT): Applications, investments, and challenges for enterprises. Business Horizons, 58(4): 431-440.

[15] Gilchrist A. (2016) Industry 4.0: the industrial internet of things. Apress.

[16] Chen M., Miao Y., Hao Y. & Hwang K. (2017) Narrow band internet of things. IEEE access, 5: 20557-20577.

[17] Mahmoud M. S. & Mohamad A. A. (2016) A study of efficient power consumption wireless communication techniques/modules for internet of things (IoT) applications. Advances in Internet of Things, 6(2): 19–29.

[18] Suresh P., Daniel J. V., Parthasarathy V. & Aswathy R. H. (2014) A state of the art review on the Internet of Things (IoT) history, technology and fields of deployment. Proc. 2014 IEEE International conference on science engineering and management research (ICSEMR). Chennai, India, 1-8.

[19] Gomez C., Oller J. & Paradells J. (2012) Overview and evaluation of bluetooth low energy: An emerging low-power wireless technology. Sensors, 12(9): 11734-11753.

[20] Hunkeler U., Truong H. L. & Stanford-Clark A. (2008) MQTT-S—A publish/subscribe protocol for Wireless Sensor Networks. Proc. 2008 IEEE 3rd International Conference on Communication Systems Software and Middleware and Workshops (COMSWARE'08). Bangalore, India, 791-798.

[21] Chen Y. & Kunz T. (2016) Performance evaluation of IoT protocols under a constrained wireless access network. Proc. 2016 IEEE International Conference on Selected Topics in Mobile & Wireless Networking (MoWNeT). Piscataway, NJ, USA 1-7.

[22] About LoRaWAN®. URL: https://lora-alliance.org/about-lorawan. Accessed 1.10.2019.

[23] Augustin A., Yi J., Clausen T. & Townsley W. (2016) A study of LoRa: Long range & low power networks for the internet of things. Sensors, 16(9), 1466.

[24] Huang A., Huang M., Shao Z., Zhang X., Wu, D. & Cao, C. (2019) A Practical Marine Wireless Sensor Network Monitoring System Based on LoRa and MQTT. arXiv preprint arXiv:1906.09571.

[25] Mikä on LoRaWAN? URL: https://www.digita.fi/yrityksille/iot/mika_on_lorawan. Accessed 1.10.2019.

[26] Chen M., Mao S. & Liu Y. (2014) Big data: A survey. Mobile networks and applications, 19(2): 171-209.

[27] Wolfert S., Ge L., Verdouw C. & Bogaardt M. J. (2017) Big data in smart farming–a review. Agricultural Systems, 153: 69-80.

[28] Kitchin R. (2014) The real-time city? Big data and smart urbanism. GeoJournal, 79(1): 1-14.

[29] Mell P. & Grance T. (2011) The NIST definition of cloud computing. NIST Special Publication 800 (2011) 7.

[30] Sikeridis D., Papapanagiotou I., Rimal B. P. & Devetsikiotis M. (2017) A Comparative taxonomy and survey of public cloud infrastructure vendors. arXiv preprint arXiv:1710.01476.

[31] Buyya R., Yeo C. S. & Venugopal S. (2008) Market-oriented cloud computing: Vision, hype, and reality for delivering it services as computing utilities. Proc. 2008 IEEE 10th IEEE International Conference on High Performance Computing and Communications. Dalian, China, 5-13.

[32] Cáceres J., Vaquero L. M., Rodero-Merino L., Polo Á. & Hierro J. J. (2010) Service scalability over the cloud. In Handbook of Cloud Computing. Springer, Boston, MA. 357-377.

[33] Hassan S. S., Bibon S. D., Hossain M. S. & Atiquzzaman M. (2018) Security threats in Bluetooth technology. Computers & Security, 74: 308-322.

[34] Zhang Q., Cheng L. & Boutaba R. (2010) Cloud computing: state-of-the-art and research challenges. Journal of internet services and applications, 1(1): 7-18.

[35] MongoDB. URL: https://www.mongodb.com/ Accessed 9.8.2019.

[36] Kiefer R. (2019) How to store time-series data in MongoDB, and why that's a bad idea. URL: https://blog.timescale.com/blog/how-to-store-time-series-data-mongodb-vs-timescaledb-postgresql-a73939734016/ Accessed 9.8.2019.

[37] Stewart R. A., Willis R., Giurco D., Panuwatwanich K. & Capati, G. (2010) Web-based knowledge management system: linking smart metering to the future of urban water planning. Australian Planner, 47(2): 66-74.

[38] Hauber-Davidson G. & Idris E. (2006) Smart water metering. Water, 33(3), 38-41.

[39] Kilpeläinen T., Alanärä S. (2019) Oulun Vesi. Corporation meeting with Oulun Vesi and Vesitieto Oy.

[40] Oulun Energia (2019) Corporation meeting between Oulun Energia and Vesitieto Oy.

[41] Case study: Transforming a hidden leak into excellent customer service. (2019) URL: https://adcf25c6-7658-45b3-ba1c-6f61503349e0.filesusr.com/ugd/ea34fd_0fca45f6e8cc4a97bbdd962bacd1bca8.pdf. Accessed 9.8.2019.

[42] FINLEX ® - Ajantasainen lainsäädäntö: Vesihuoltolaki 119/2001. (2001) URL: https://www.finlex.fi/fi/laki/ajantasa/2001/20010119?search[type]=pika&search[pika]=vesihuolto. Accessed 1.10.2019.

[43] European Commission - Basics - Managing water resources (2015) URL: https://ec.europa.eu/environment/basics/green-economy/water-resources/index_en.htm Accessed 1.10.2019.

[44] Voigt P. & Von dem Bussche A. (2017) The EU general data protection regulation (gdpr). A Practical Guide, 1st Ed., Cham: Springer International Publishing.

[45] Allied ICT Finland (2019) Water ecosystems. Workshops with waterworks and organizations. Oulu, Finland.

[46] Grolinger K., Higashino W. A., Tiwari A. & Capretz M. A. (2013) Data management in cloud environments: NoSQL and NewSQL data stores. Journal of Cloud Computing: advances, systems, and applications, 2(1): 22

[47] Pokorny J. (2013) NoSQL databases: a step to database scalability in web environment. International Journal of Web Information Systems, 9(1): 69-82.

[48] Vicknair C., Macias M., Zhao Z., Nan X., Chen Y. & Wilkins D. (2010) A comparison of a graph database and a relational database: a data provenance perspective. Proc. 2010 ACM 48th annual Southeast regional conference. New York, NY, USA, 42.

[49] Han J., Haihong E., Le G. & Du J. (2011) Survey on the NoSQL database. Proc. 2011 IEEE 6th international conference on pervasive computing and applications. Sydney, Australia 363-366.

[50] IoT Reference Architecture. (2019) URL: https://webassets.mongodb.com/iot_reference_architecture.pdf. Accessed 18.12.2019.

[51] Kvalheim C. (2015) The Little Mongo DB Schema Design Book. The Blue Print Series.

[52] Walters R. (2018) Time Series Data and MongoDB: Part 2 – Schema Design Best Practices: MongoDB Blog. URL: https://www.mongodb.com/blog/post/time-series-data-and-mongodb-part-2-schema-design-best-practices Accessed 18.12.2019.

[53] Rizvi S. S. & Chung T. S. (2010) Flash SSD vs HDD: High performance oriented modern embedded and multimedia storage systems. Proc. 2010 IEEE 2nd International Conference on Computer Engineering and Technology. Chengdu, China. V7-297-V7-299.

[54] Tamminen S., Tiensuu H., Ferreira E., Helaakoski H., Kyllönen V., Jokisaari J. & Puukko E. (2018) From Measurements to Knowledge-Online Quality Monitoring and Smart Manufacturing. Proc. 2018 Springer Industrial Conference on Data Mining. Cham, Germany. 17-28.

[55] Masse M. (2011) REST API Design Rulebook: Designing Consistent RESTful Web Service Interfaces. O'Reilly Media, Inc..

[56] Riahi A., Challal Y., Natalizio E., Chtourou Z. & Bouabdallah A. (2013) A systemic approach for IoT security. Proc. 2013 IEEE international conference on distributed computing in sensor systems. Cambridge, Massachusetts, USA, 351-355.

[57] Wurm J., Hoang K., Arias O., Sadeghi A. R. & Jin Y. (2016) Security analysis on consumer and industrial IoT devices. Proc. 2016 IEEE 21st Asia and South Pacific Design Automation Conference (ASP-DAC). Macao SAR, China, 519-524.

[58] Zhang Z. K., Cho M. C. Y., Wang C. W., Hsu C. W., Chen C. K. & Shieh S. (2014) IoT security: ongoing challenges and research opportunities. Proc. 2014 IEEE 7th international conference on service-oriented computing and applications. Matsue, Japan, 230-234.

[59] Babar S., Stango A., Prasad N., Sen J. & Prasad R. (2011) Proposed embedded security framework for internet of things (iot). Proc. 2011 IEEE 2nd International Conference on Wireless Communication, Vehicular Technology, Information Theory and Aerospace & Electronic Systems Technology (Wireless VITAE). Chennai, India, 1-5.

[60] Babar S., Mahalle P., Stango A., Prasad N. & Prasad R. (2010) Proposed security model and threat taxonomy for the Internet of Things (IoT). Proc. 2010 International Conference on Network Security and Applications. Chennai, India, 420-429.

[61] Asokan N., Niemi V. & Nyberg K. (2003) Man-in-the-middle in tunnelled authentication protocols. Proc. 2003 International Workshop on Security Protocols. Springer, Berlin, Heidelberg, 28-41. DOI: https://doi.org/10.1007/11542322_6.

[62] Babar S., Mahalle P., Stango A., Prasad N. & Prasad R. (2010) Proposed security model and threat taxonomy for the Internet of Things (IoT). Proc. 2010 International Conference on Network Security and Applications. Springer, Berlin, Heidelberg. 420-429. DOI: https://doi.org/10.1007/978-3-642-14478-3_42.

[63] Boehm B. (2002) Get ready for agile methods, with care. Computer, (1): 64-69.

[64] Weyrich M. & Ebert C. (2015) Reference architectures for the internet of things. IEEE Software, 33(1): 112-116.

[65] Microsoft Azure IoT reference architecture - Azure Reference Architectures. (2018) URL: https://aka.ms/iotrefarchitecture Accessed 18.12.2019.

[66] Mok A. K. L. (1983) Fundamental design problems of distributed systems for the hard-real-time environment. Doctoral dissertation, Massachusetts Institute of Technology.

[67] Paschou M., Sakkopoulos E., Sourla E. & Tsakalidis A. (2013) Health Internet of Things: Metrics and methods for efficient data transfer. Simulation Modelling Practice and Theory, 34: 186-199.

[68] Zhu Q., Wang R., Chen Q., Liu Y. & Qin W. (2010) Iot gateway: Bridgingwireless sensor networks into internet of things. Proc. 2010 IEEE/IFIP International Conference on Embedded and Ubiquitous Computing. Hong Kong SAR, China, 347-352.

[69] Montella R., Ruggieri M. & Kosta S. (2018) A fast, secure, reliable, and resilient data transfer framework for pervasive IoT applications. Proc. 2018 IEEE INFOCOM 2018-IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS). Honolulu, HI, USA, 710-715.

[70] Li Y. & Manoharan S. (2013) A performance comparison of SQL and NoSQL databases. Proc. 2013 IEEE Pacific Rim Conference on Communications, Computers and Signal Processing (PACRIM). Victoria, B.C., Canada, 15-19.

[71] Cattell R. (2011) Scalable SQL and NoSQL data stores. ACM Sigmod Record, 39(4): 12-27.

[72] Parker Z., Poe S. & Vrbsky S. V. (2013) Comparing nosql mongodb to an sql db. Proc. 2013 ACM 51st ACM Southeast Conference. Savanah, Georgia, USA, 1-6. DOI: https://doi.org/10.1145/2498328.2500047.

[73] Chaudhuri S. & Narasayya V. R. (1997) An efficient, cost-driven index selection tool for Microsoft SQL server. Proc. 1997 23rd International Conference on Very Large Data Bases, VLDB. Athens, Greece, 146-155.

[74] Elmasri R. & Navathe S. B. (2011) Fundamentals of database systems (sixth edition). Boston, MA: Pearson Education.

[75] Lourenço J. R., Cabral B., Carreiro P., Vieira M. & Bernardino J. (2015) Choosing the right NoSQL database for the job: a quality attribute evaluation. Journal of Big Data, 2(1): 18.

[76] Fielding R., Gettys J., Mogul J., Frystyk H., Masinter L., Leach, P. & Berners-Lee, T. (1999) Hypertext transfer protocol–HTTP/1.1. Internet RFC 2616.

[77] Khan M. E. & Khan F. (2012) A comparative study of white box, black box, and grey box testing techniques. International Journal of Advanced Computer Science and Applications, 3(6): 12-15.

[78] Dustin E. (2002) Effective Software Testing: 50 Ways to Improve Your Software Testing. Addison-Wesley Longman Publishing Co., Inc..

[79] Gupta V. & Verma C. (2012) Error detection and correction: An introduction. International journal of advanced research in computer science and software engineering, 2(11):193-234.