Open Specification of a user-controlled Web Service for Personal Data

G. Jahn

February 18, 2017

Abstract

Often data is referred to as *the oil of the 21st century*. But recovering petroleum or coal is governed by certain rules and legislation, while vast amounts of personal data are harvested from all sorts of resources with no working restrictions or asking for permission. Well air-conditioned data centers are mining around the clock via dozens of CPUs. Eager to discover even the tiniest correlations worth interpreting in enormous haystacks that are filled with data belonging to millions of individuals, who have no knowledge of those computations. But these very computations, and thus their developers, almost inevitably and without restraints discriminate against these so called *data subjects*. Such practices cannot be accepted because these haystacks contain actual identities of human beings including their personalities but still getting recklessly monetized. To address this issue and reduce the possibility of discrimination, third parties have to be supplied with the least amount of data in order to stay functional. That is, data owners must be in charge of deciding which entity has access to what personal data of theirs. This project aims to specify a personal data service that empowers its user to regain full control over her personal data. It facilitates the regulation of data flows and provides detailed information on where the data goes so that a *data subject* can become her own data broker. In order to trust such a service, the user must be able to look inside and see for herself if it behaves in unexpected ways. Therefore the specification and all implementations are being open sourced and developed transparently in public, which also encourages self-hosting.

Acknowledgement

I'm thanking my Haekelschmein and my internet provider. TODO My other/better half (significant other) for having my back through the whole time My parents for letting/allowing me to get this fare.

Table of Contents

List of Tables

List of Figures

This page intentionally left blank

1 Introduction

1.1 Motivation

Nowadays, it is difficult to find a business that does not collect data about something; humans are particular targets of choice for the *Big Data Movement* [1]. Since humans are all individuals, they are distinct from each other. While subsets of individuals might share a minor set of attributes, the majority is still very unique to an individual, given that the overall variety of attributes is complex. That small amount of similarity might seem to be less important, due to the nature of inflationary occurrence, but the opposite turns out to be true. These similarities allow to determine the individuals who are part of a subset and the ones who aren't. Stereotypical patterns are applied to these subsets and thus to all related individuals. This enriched information is then used to help predict outcomes of problems or questions related to these individuals. In other words, searching for causation where in best the case one might find correlations. This is also known as *discrimination*, which

> *[…] refers to unfair or unequal treatment of people based on membership to a category or a minority, without regard to individual merit. [2]*

When interacting directly with each other, discrimination of human beings is a serious issue in our society, but also when humans leverage computers and algorithms to uncover formerly unnoticed information in order to inform their decision making. For example, when qualifying for a loan, hiring employees, investigating crimes or renting flats. The decision to approve or deny is based on computed data about the individuals in question [3], which is merely discrimination on a much larger scale and with less effort (almost parenthetically). The described phenomenon is originally referred to as *Bias in computer systems* [4]. What at first seems like machines going rouge on humans is, in fact, the *cognitive bias* [5] of human nature, modeled into machine executable language and built to reveal the patterns their creators were looking for. The *"Inheritance of humanness"* [6], so to say.

In addition to the identity-defining data mentioned before, humans have the habit to create more and more data on a daily basis, both pro-actively (e.g by writing a post) and passively (e.g by allowing the app to access their current location while submitting the post). As a result, already gigantic databases grow ever larger, waiting to be harvested, collected, aggregated, analyzed and finally interpreted. The crux here is, the more data being made available [7] to *mine* on, the higher the chances to isolate datasets (clusters) that differ from each other but are coherent within themselves. By defining those datasets, instead of distinguishing on an individual level, humans are being reduced to these set-defining characteristics in order to fit in these clusters.

In order to lower potential discrimination, either the responsible parts in these machines need to be erased while simultaneously raising awareness and teaching people about this issue of discrimination, or all the personal data needs to be prevented from falling into these data silos in the fist place. Although both approaches are valid and should be pursued simultaneously, the latter will be addressed in this work.

1.2 Purpose & Outcome

At first glance, it might not be considered harmful to provide one's own personal data to third parties, at least from an individual's perspective, because free or improved services are eventually offered in return. For example, more adequate recommendations and fitting advertisement, or more helpful therapies and more secure environments. Gathering and processing data is essentially just mathematics and computer technologies. How those tools are utilized and what purposed they serve is within the decision of their developers. However, what data points are used and how they get processed should be determined by the data creators. Thereby allowing them an influence on the results of these processes and thus on decisions made upon them which impact their lives.

To address the described issue, the initial idea here is to (1) equip individuals with the ability to control and maintain their entire data distribution, in order to (2) reduce the amount of *potentially discriminatory* [2] attributes that could leak into arbitrary computations. For that, people need a reliable and trustworthy tool, which helps them to manage all their *personal data* and provides an interface for third parties to access their data, but on their own terms. The parties that would be responsible for such a tool would likely have the most accurate and reliable one-stop resource to an individuals' *personal data* at hand, while simultaneously being urged to respect their privacy. This approach comes along also with some downsides related to security and potential data loss. Elaborating on these issues and discussing potential solutions is part of the design process.

This way of addressing the described dilemma about personal data analysis is not new (see Related Work). Early work done in this field can be dated back to the Mid-2000s when studies were conducted, for example, about recent developments in the industry (e.g targeted ads) and the user's concerns about privacy [8]. At that time, the term *Vendor Relationship Management(VRM)* was first used within the context of user-centric personal data management, which then also led into the *ProjectVRM* [9] started by the *Berkman Klein Center for Internet & Society at Harvard University.* A great amount of effort has gone into this area of research since then. While commercial products and business models try to solve some of the problems related to

this. For instance, with concepts like the *Personal Data Store (PDS)* [10] or an implementation of the *MyData* concept [11] called *Meeco* [12], which are all be covered in detail in the following chapter.

The research work done for this thesis constitutes the foundation for an *Open Specification*, which is a manual for implementing a concept called *Personal Data as a Service*, henceforth called *PDaaS*. Examining important topics like the architecture, where data can be stored, how to obtain data from the exposed API or what requirements have to be met by a user interface for personal data management, is part of this work. By the time this document has been submitted, most of the core issues should have already been addressed and can thus be outlined in a first draft of the specification. Only then can the task of actually implementing certain components begin. The reason for that is, when sensitive subjects, especially things like people's privacy, are at risk, all aspects in question deserve careful consideration, so they can be addressed properly. That is, adequate effort must be put primarily into the theoretical work. However, that does not mean writing code to test theories and ideas isn't allowed during the development and specification process. It is encouraged and might even help to spot flaws or perhaps trigger improvements of the specification.

To create confidence in this project and in the software built upon it, it's vital to make all development processes fully transparent and encourage people to get involved. For this reason all related software and documents [13] are open source from day one.

In summary, this document lays the ground work and is intended to be the initial step in a development process fabricating a tool to manage personal data. The tool is controlled and administrated by the individual to whom the personal data belongs. It enables her to get a more precise understanding of what data is accessed by whom and how this might affect her privacy.

1.3 Terminologies

Web Service: A service, that is accessible by electronic devices over the internet. This makes it almost effortless to use a service which would

otherwise be out of reach. Interactions with a service usually happen through enriched websites or other web-compatible applications and interfaces.

Open Specification: A specification is a formal and very detailed way of describing a technology, its internals, and behaviour from external perspectives. It provides guidance for implementations to ensure a minimum level of interoperability. Structured in a formalized document it might become a *technical standard*. *Open* in this case means at first nothing, but it's accessible for anyone without restrictions. When it comes to the intellectual value itself, that might be handled differently, for example with an enclosed license.

Profile Data: A collection of data points reflecting an individual's inherent information and other basic predominantly static data points (no sets), which, in conjunction, uniquely relate to that individual.

Digital Footprint: Refers to data that is related to an individual. It is distinguished between an active footprint, which involves data and information about an individual who chose to share them publicly, and a passive footprint, which includes all data about an individual collected by third parties without the individual's knowledge.

Personal Data as a Service (PDaaS): A web service that is controlled, owned, and maybe even hosted by an individual. It provides access to the data subject's personal data and offers maintainability as well as permission management for those data. It can be seen as her personal agent; sometimes also referred to as *the system*.

Data Subject: An individual who first and foremost is the owner of all of her personal data; sometimes referred to as *owner*.

Operator: A *data subject* that uses a *PDaaS* to control (and probably host) her personal data; sometimes referred to as *data controller* or *data owner*.

(Data) Consumer: Third party who requests permission or is already allowed to access the *operator's personal data* through her *PDaaS*; sometimes referred to as *(data) collector* or vendor.

Data Broker: Third party with commercial interests in collecting, aggregating and analyzing information/data about humans from any possible resource in order to combine and enrich the data, to finally license those corpora to other organisations. [14]

Permission Request: A formalized attempt made by a third party to request permissions in order to access certain data points on the *PDaaS*. The request has to include all the data points to which access is being demanded, as well as sufficient information about the purpose. It requires the third party to already being registered as *data consumer*.

Access Request: An attempt to actually access data provided by a *PDaaS*. The request primarily consists of a query, that defines what data points are tried to be accessed. The access is only permitted if the query matches against the *permission profiles*.

Permission Profile: A set of access rules and configuration tied to a *data consumer*. It determines what data is accessible by the related *data consumer* and for how long. The profile is the result of a reviewed and granted *permission request*.

Endpoint: An endpoint is defined as part of a URI that is uniquely associated with a single *data consumer*. Usually it's the first part of a URI (e.g. domain incl. subdomains), whereas following parts indicating different resources that might be available within that endpoint. It can also be viewed as group of resources whose access is restricted.

1.4 Scenarios

The following use cases portray different situations and possible ways in which the tool in question might be applicable, and shows in several ways that it can be helpful to be in charge of its own personal data. Some are more practical and realistic, like ordering and purchasing a product on the internet, while others might at the moment not seem to be very useful, but show a certain potential to become more relevant when new technologies and business models will occur, that are followed by new desires for data.

Ordering a product on the web

The data subject searches through the web to find a new toaster because her old one recently broke. After some clicks and reviews, she finds her soon-to-become latest member of the household's kitchenware. After putting the

model name in a price search engine, hoping to save some money, the first entry, offering a 23% discount, catches her attention. She decides to have a deeper look into the toasters, so she heads towards the original web shop entry. Finally she comes around and puts the item onto her shopping cart, despite the fact that she never bought something from that online shop before. Then she proceeds to checkout so that she can place her order. The shop-interface asks her to either insert her credentials, proceed without registration or sign-in, or allow the shop to obtain all required data on its own by either scanning a QR-Code displayed below or insert a URI to her *Personal Data as a Service*. She opens up the management panel of her *PDaaS* in a new browser window and authenticates herself to the system. Afterwards she creates a new entry in a list of *data consumers* who already get permitted to access certain characteristics of her personal data. As a result, she gets prompted with a URI, which she inserts as the shop interface requests her to do, only after she has convinced herself that the data exchange with the shop is based on a secure connection (HTTPS). Moving on to the next step after submitting the URI, the data subject is asked to decide how she would like to pay. The choices are: credit card, invoice, online payment or bank transfer. She chooses the last one, submits her selection and thereby completes the order process. She goes back to the kitchen. After some time, a push notification appears on her mobile device. The notification is about a *permission request* which has just arrived at her *PDaaS*, asking her to grant permissions to the shop-system, where she earlier placed the order. The shop wants to access her full name, address and email, which are required to proceed with the order. Based on the information given in the request, she creates a new *permission profile* for the shop. Additionally, for the profile she can decide between three states of how long the permission is going to last: *one time only*, *expires on date* and *granted, until further notice*. Since she has never ordered at this shop before and probably won't do it again, she decides to grant access only for this specific occasion. The shop-system is then notified of the decision result. If the result is positive - which is the case here - the data can be obtained and the order can be further processed. As a result, the data subject receives an email, containing information regarding her order, including the shop owner's bank details, which enables her to pay the due amount. After the shop-system receives the payment, the toaster is shipped.

In order to get a full impression of how the whole process would look like if the data subject would have chosen another payment method, the differences are described below. If the data subject had wanted to pay with her credit card, the shop-system would have asked to also access her credit card as well as its associated secret. When sending the email, the system would have omitted the information about the shop's bank details. Paying with invoice, would have been possible only if the *PDaaS* initially had been able to provide certified profile data, which necessarily would have been rated trustworthy. That again would have reduced the risk taken by shop owner and would have enabled her to take action in cases of fraud or misuse. Choosing to involve an online payment service provider as a *middleman* for processing the payment would have required the data subject to have granted the provider access to her *PDaaS* upfront. In that case, the shop-system would have asked for her payment provider account identifier, so that the system could have requested the payment directly from that payment service provider. This would have caused the service provider to consult the *PDaaS*, which would have resulted in a second notification asking the data subject for permission to proceed. After a successful payment transfer, the shipment would have been initiated.

Interacting with a social network

The first entry to a social network requires either a URI to the data subject's *PDaaS*, which has uniquely been generated for that purpose, or a QR-Code provided by the social network. The data subject receives a notification on her mobile device send from her *PDaaS*, revealing what data that network wants to access and maybe even why. If her mobile device is currently not at hand, she can also use the management panel provided by her *PDaaS*, which is accessible with a web browser on every internet-enabled device. Within that panel pending permission reviews are indicated. Regardless of whether the data subject has already reviewed the request, she should still be able to login. After doing so, she would see all her information, unless she has not yet granted permissions to the social network to access her data *until-further-notice*. If this is done, after waiting a moment and then reloading the browser session, all her data should then show up. So every time someone on that network tries to access her information, with whom she has allowed to

see that information, managed by each user within the network, the network pulls the required data from her *PDaaS* as long as it is permitted to do so. It is also conceivable that the social network does provide a back-channel to the *PDaaS*, so that all content she creates within that network, including all interactions with other users, can be stored in her *PDaaS*, allowing it to be provided to other *data consumers*. The network itself only stores references to all these content objects. Whether it is an image, a post, or comment on somebody else's post, the actual content to be displayed is fetched from the corresponding *PDaaS*.

Applying for a loan and checking creditworthiness

The data subject would like to buy an apartment. In order to finance the acquisition, she needs funding which, in her case, is based on a loan. During a conversation in a credit institute of her choice, an account consultant describes to her what data is required in order to decide her creditworthiness. While nodding consensually, she takes out her smartphone and brings up the management panel of her *PDaaS*. The consultant flips his screen showing a QR-Code and the *data subject* scans it. The tool displays some information about the institute including a reference to this assignment and a list of all data points the institute would like to access in order to calculate her score, such as address, monthly income, relationship status and family, history of banking or other current loans. After some back and forth and solving some misunderstandings with the help of her consultant, she decides to allow only partial access to the requested data, and only for this time and purpose. The consultant kindly points out, that these decisions might impact her score and thus on the loan and its terms. While handling some more formalities and talking about other possible products she might be interested in, the consultant gets notified by his computer, confirming the access permission. Thereupon the two finish their meeting and the consultant informs the data subject of the next steps, which include a note about contacting her within the next few days, when the institute has come to a conclusion. In the case of a positive outcome, a new appointment for handling all the paperwork and signing the contract needs to be made. From a technical point of view, two different ways of computing the score are imaginable. The first would be

to just transfer the plain data including expiry date and information about how reliable the data is. However, the actual computations and analytics to obtain the score happen within the infrastructure of the credit institute. When this process has finished, all the personal data that have been transferred must be erased. An alternative, though, could prevent the data from leaving the *PDaaS*. Therefore, the institute's request won't contain a data query. Instead it comes along with some software and information on how to run it. The *PDaaS* server will provide an isolated runtime in which the software can be executed. After that process is finished, the result is sent back to the credit institute's infrastructure.

Maintain and provide its own medical record

Some time ago on a hiking trip, in a moment of carelessness the data subject accidently broke her leg. She went to a hospital and straight into surgery, where the surgeon where able to fix the injury. Time went by and the leg healed completely. When she woke up today, she felt pain coming from the area where her leg was broken. She decided to call in sick and went straight to a doctor nearby. During her recovery she had been visiting that doctor regularly. At the reception desk, she opens up the *PDaaS*'s management panel on her smartphone and searches through the list of data consumers. After she finds the entry for this clinic, she flips her phone to show the receptionist the corresponding QR-Code, which the receptionist scans immediately. However the receptionist was not able to see any data on the screen, because the access has already expired. The data subject only permitted access for the estimated time of recovery, which is already over. That's why she now gets a notification asking to re-grant some access. While going through the data points the clinic-system has requested, she notices that her address is incorrect. Last month she moved into a bigger apartment across the street. She must have forgotten to change that data. She immediately corrects the address, right before saving the *permission profile* for the clinic-system. She also includes access to all the data originating from the time after her accident. A moment later the receptionist confirms now being able to see all necessary data. The data subject takes a seat in the waiting room. While passing time, she decides to take a deeper look into her list of data

consumers. Some of them she couldn't even remember and for others she was surprised by which data points she had granted access to. She starts to restrict certain permissions, as she desires. She even removes some of the entries entirely. The appointment with her doctor went great. The doctor even had to review the x-ray images in order to make an adequate differential diagnosis. After the visit, she makes a quick stop at a pharmacy along the way to pickup the drugs her doctor had prescribed to reduce the pain. She has to wait in the queue since two other customers are in front of her. She realizes that it's the first time she's in this store. So she prepares a new entry in her list of data consumers, including all information about her prescriptions. By the time she gets served, she simply lets the person behind the register scan her QR-code. In the next seconds the data subject gets a notification about a *permission request* from this store, which she quickly reviews and confirms by making sure that the permissions in that profile are the ones she prepared minutes ago. A moment later the pharmacist comes back with her drugs, which she then pays in cash. The transaction is done and the data subject leaves the store.

Vehicle data and mobility

Let's assume a car has no hardware on board to establish a wireless wide area connection to an outside access node. One is only able to connect to the car from the inside (wired or wireless). After entering a car, on the data subject's mobile device a notification comes up from the car asking for permission to connect that mobile device. In addition to the expiration date, the data subject is provided with two additional options which she can enable or disable. First, a wifi network provided to everyone in the car can be enabled, which utilizes the uplink from the mobile device to the internet. Secondly, the car is allowed to use the uplink to open up connections so it can emit or receive data from the internet. As a result the device owner gains full control over any data the car might want to transfer. And this again would allow two things: (A) permission management for all outgoing data and (B) funnel all data generated and provided by the car towards the *PDaaS* that is associated with the linked device. It might be feasible as well to deny certain connections the car tries to open. Data will then be stored

only in the *PDaaS*. If a third party is interested in that data they have to ask for access permission. That same concept of movement tracking and vehicle data aggregation could be applied to driving motorcycles and bicycles as well.

2

Fundamentals

The following chapter shall provide basic knowledge of concepts like *Personal Identity*, *Big Data* and *Data Ownership*. It also explains what *Personal Data* is from a legal standpoint and covers some of the issues caused by different legislation colliding through the one global internet. This again requires an elaboration on how *Personal data* impacts our society as well as the economy. Overall, this chapter is meant to facilitate a common understanding of the stated issue and why it could be addressed as described later on. Furthermore, there is a summary of what research has already been done its current state. Finally, it gives a brief overview of standards and technologies that might be utilized for this project.

2.1 Digital Identity, Personal Data and Ownership

A **Digital Identity** is viewed as a non-physical abstraction of an entity, such as an organisation, an individual, a device or even some software. It is bidirectionally associated to its physical counterpart. In the context of this work, it only refers to human beings. Therefore a Digital Identity is

the representation of an individual in digital systems, consisting of identity-defining data, such as *personal information*, its own history and preferences [15]. *Personal information*, in this case, refers to inherent (e.g. date of birth) as well as imposed (e.g. credit card number) characteristics. The individual to whom those data relates to, is the owner of that *Digital Identity*. From a technical standpoint, a *Digital Identity* is essentially a collection of characteristics, attributes and time series data (e.g. interaction logs or bank transfer history). Based on a subset of those attribute values, a unique fingerprint can be easily generated. Depending on the data point and complexity of its value, such a fingerprint could require either a single, unique identifier on its own (e.g. social security number) or only a few. Hence, it doesn't take a complete set of attributes including all its values, but rather just a fraction of a *Digital Identity* in order to determine its rightful owner and physical counterpart. The *Digital Identity* can be viewed as an avatar in digital environments or even as the digital part of a persons's identity. That is, a *Digital Identity* of a living individual cannot simply be reduced to bits and bytes. Instead, it should be valued as an appropriate, and perhaps legal, representation in certain contexts and for a variety of purposes. In some of those situations it might be required (e.g. administrative purposes) to ensure a certain level of authenticity for a *Digital Identity* or for particular attributes of it. This means, to provide reliable confirmation that the attribute values are really the ones that belong to exactly they individual they supposedly belong to. An independent third party, who is trusted by all entities participating in such a construct, could somehow verify (or vouch for) the subject in question. On the other hand, this concept opens up at least one class of attack scenarios. The risk of identity theft, for example, increases dramatically when assigning such value to a *Digital Identity* because the attacker is no longer required to be physically present in order to impersonate that identity or "steal" certain unique identifiers from that person. Instead, it is sufficient to gain access to the areas where those sensitive data are stored. It is noted, that different technical solutions to these issues exist and will be discussed later on.

In the context of this project, and all related work, **Personal Data** is defined as the total amount of data that is part of either an individual's *Digital Identity* or her *Digital Footprint*. On the one hand, this includes all intellec-

tual property (e.g. posts, images, videos or comments) ever created, and all kinds of tracking data, interaction monitoring and metadata, that is used to manually or automatically enrich content (e.g.geo-location attached to a tweet as meta information). Moreover, it refers to data that is captured by someone or something from within the individual's private living space or property. *Personal data* is basically understood as every data point reflecting the individual as such and her personality - partially or as a whole.

The european "Data Protection Regulations", on the other hand, defines *Personal Data* as follows:

> *'personal data' means any information relating to an identified or identifiable natural person ('data subject'); an identifiable natural person is one who can be identified, directly or indirectly, in particular by reference to an identifier such as a name, an identification number, location data, an online identifier or to one or more factors specific to the physical, physiological, genetic, mental, economic, cultural or social identity of that natural person; [16]*

In contrast, the U.S.A. has little to no legislation defining *personal data* and thereby protecting the individual's privacy. There is at least no explicit federal law addressing such areas [17]. Though, some of the existing sectoral laws contain partially applicable policies and guidelines [18]; most of them addressing only data related to specific topics (e.g. health insurance, financial record or lending). In 2015 the White House has attempted to fill that gap with the "Consumer Privacy Bill of Rights Act", but to this date it has not left the draft state. According to the critics, the bill lacks concrete, enforceable rules which consumers can rely on [19]. The draft contains a general definition of *Personal Data*:

> *"Personal data" means any data that are under the control of a covered entity, not otherwise generally available to the public through lawful means, and are linked, or as a practical matter linkable by the covered entity, to a specific individual, or linked to a device that is associated with or routinely used by an individual, including but not limited to [...] [20]*

At the end a list of concrete data points follows. Examples include email or postal address, name, social security number and so on.

Aside from government legislation, several third-party organisations in the U.S. are also allowed to define rules and policies that can overwrite existing laws, namely the *Federal Communications Commission (FCC)*, which recently released "Rules to Protect Broadband Consumer Privacy" for ISPs[1] including a list of categories of sensitive information [21]. Thereby the FCC wants *Personally Identifiable Information* (alias *Personal Data*) to be understood as:

> *[…] any information that is linked or linkable to an individual. […] information is "linked" or "linkable" to an individual if it can be used on its own, in context, or in combination to identify an individual or to logically associate with other information about a specific individual. [22]*

Despite minor difference in details, the FCC has a serious idea of what is included in *personal data* and to whom they belong. Although the FCC's legal participation might be somewhat debatable regarding limitation to certain topics, the "Communications Act" as a U.S. federal law qualifies the FCC to regulate and legislate within its boundaries.

Having a common understanding on what data points belong to a person is the foundation of defining a set rules on how to handle *Personal Data* appropriately. Hence, every business operating within the EU is required[2] to provide its users with a *Privacy Policy*, while in the U.S. for example, as mentioned above, only infrequently and depending on the class of data or context users, must inform its users about how their data get processed and what data points are involved [23]. A user typically agrees on a *Privacy Policy* by starting to interact with the author's business or platform. Thus every *Privacy Policy* is required to be publicly accessible. For instance, not in a restricted area after logging in, but before creating an account. The following example is taken from facebook's current landing page.

> *By clicking Create an account, you agree to our Terms[3] and that*

[1] Internet Service Provider
[2] according to article 12-14 of the "EU General Data Protection Regulation 2016/679"
[3] https://www.facebook.com/legal/terms

you have read our Data Policy[4], including our Cookie Use[5]. [24]

It can be viewed more like an information notice, which translates and specifies the prevailing legal situation, rather then a contract, which the user would be forced to read and accept before revealing her data; otherwise known from procedures like software installations, where the user might have to accept terms of use or license agreements. With a *Privacy Policy* at hand, it's up to each individual to decide if the benefits the service offers are worth sharing part(s) of her *personal data*, while at the same time reluctantly tolerating potential downsides concerning the privacy of that data. When the vendor considers its policies accepted by a user, her personal data must be processed as stated in those policies but also according to the law. If the policies violate existing law or the vendor effectively goes against the law with its actual doings, penalties might follow. Depending on level and impact of the infringement in addition to what the law itself says, the vendor, while revisiting its wrong-doings in order to improve, might have to compensate affected individuals, pay a fine, or have its license revoked.

Not only privacy laws, but every legal jurisdiction has it's limitations - such as its territorial nature - which makes it an inappropriate tool for addressing existing issues and strengthening individuals' privacy and rights in a global context like the *world wide web*. Whereas the EU has approved extensive regulations, mentioned above, that are supposed to provide privacy protection and defines the handling of personal data, the U.S., on the other hand, has only subject-specific rules which merely apply to its own citizens. Even though the definition of personal data included in the EU regulation is almost identical to the one introduced for the context of this work, it only applies to vendors and individuals who are part of the EU. Even privacy policies won't help if the vendor is registered in a different area of jurisdiction than the user's location. For those circumstances, international agreements might be established [25], but this approach might still be useless if it either fails to provide proper tools for users to enforce their rights or it is simply ignored by contract partners with or without legal foundation [26] [27].

While the legislation mentioned above is in place, *Ownership* of *Personal*

[4]https://www.facebook.com/about/privacy
[5]https://www.facebook.com/policies/cookies/

Data has no legal basis whatsoever. The concepts of intellectual property protection and copyright might intuitively be applicable, because the data that is defined through the sole existence of the *data subject* (Digital Identity) and the data that is created by her seems to be her *intellectual property* as well. Such property implies that it is the result of a creative process, but unfortunately facts, which most *personal data* are, don't show a *threshold of originality* [28]. Thus, the legal concept of *intellectual property* does not apply. However, Ownership in the context of this work, is understood as a concept of having exclusive control over its personal data and how those data get processed at any given point in time. The exclusive control is emphasized as (A) the right to do what ever is desired with its property and (B) by which rules and mechanisms the ownership can be assigned to someone [29]. This might result in a logistical challenge in which the data subject has to allow data access without losing the exclusive control over that data. In any case some effort might be required in order to preserve ownership as described, caused by the general characteristics that data has.

The european "Data Protection Regulations" mentioned before contain only one occurrence of the word *ownership*, and it's not even related to the context of *personal data* or the *data subject*. It merely states that *"Natural persons should have control of their own personal data"* [30]. Whereas Commissioner J. Rosenworcel of the FCC wants *"consumers […] to […] take some ownership of what is done with their personal information."* [31] Despite those two exceptions, elaborations on *data ownership* are almost non-existant in current legislation. Instead, the question is typically addressed in *Terms of Service (ToS)* provided by *data consumers*, which an individual might have to accept in order to establish a (legal) relationship with its author. The individual should keep in mind, that *Terms of Services* can change over time; not necessarily to the users advantage. The contents of a *ToS* must not violate any applicable or related law, otherwise the terms might not be legally recognized. Taking for example the following excerpts from different *Terms of Services*:

> *You own all of the content and information you post on Facebook, and you can control how it is shared […]. (under "2. Sharing Your Content and Information", by Facebook [32])*

> *You retain your rights to any Content you submit, post or display on or through the Services. What's yours is yours — you own your Content. (under "3. Content on the Services", by Twitter [33])*
>
> *Some of our Services allow you to upload, submit, store, send or receive content. You retain ownership of any intellectual property rights that you hold in that content. In short, what belongs to you stays yours. (under "Your Content in our Services", by Google [34])*
>
> *Except for material we may license to you, Apple does not claim ownership of the materials and/or Content you submit or make available on the Service (under "H. Content Submitted or Made Available by You on the Service", by Apple [35])*

All those statements are essentially superceded by a subsequent statement within each *ToS*, stating that the user grants the author a worldwide license to do almost any imaginable thing with her data. In case of Apple for example, if the user is *"submitting or posting […] Content on areas of the Service that are accessible by the public or other users with whom [the user] consent to share […] Content"* [35]. It is worth to be noted though, that every *ToS* only refers to the content created by the *data subject* instead of all her personal data. As mentioned above, personal information are not the subject of intellectual property, but still play an important role in data analytics. Which is why *privacy policies* are in place, to ensure at least some enlightenment on the whereabouts of the user's personal data, even though it doesn't compensate the absence of control. Additionally, neither the meaning of *ownership*, to which the quoted terms refer to, is sufficiently outlined, which results in ambiguity, and thus leaves room for interpretation, nor the proposed definition of *ownership*, as described earlier, is applicable to these *Terms of services*, since the *data subject* loses all its control by design. Handing over data to a consumer effectively disables the exclusive control over the data and eliminates the ability to assign such control. Hence, the *ownership* to that data doesn't exist any further. That is, no (legislation based) way exists to establish a feasible concept of *ownership*, unless the data consumer has a motivation to promote the *data subject* to the sole

owner of her data and to honour these privileges.

Leaving the legislation aside for a moment to move away from the top view; data consumers might argue that they have invested a lot in order to enable themselves to collect, process and store personal data, hence it belongs to them. Whereas from the data subject's point of view this might only be acceptable as long as she herself benefits from that arrangement somehow. Which would be the case if, for example, the data subject uses products, services or features offered by consumers whose quality depends on personal data. If the data subject then chooses to move to a competitor, she would want to bring her personal data with her. Again, the former data consumer would object that competitors would then benefit from all the investment the consumer has already made, but without any effort. While not entirely wrong, two aspects need to be emphasized here. In order to get high quality analytics and therefore be able to make accurate decisions to gain overall improvement, it is (A) vital to put a huge amount of effort in developing the underlying technologies, rather than primarily collecting masses of personal data. But this effort is just the precondition and applies to all of its customers, which again weakens its argumentation. It is followed by (B) an ongoing and recurring process of collecting, aggregating and analyzing actively and passively created data and metadata (e.g. food delivery history or platform interactions and tracking). According to the definition of *Personal Data* through legislation, it appears to be only a fraction of the data, namely *personal information*, that is involved in these kind of processes. The larger part, which is defined as the subject's *digital footprint*, consists of highly valuable metadata [36] [37], but is not covered by law. If the *data subject* ends the relationship with a collector, at the very least, the personal information should be erased and all remaining data sufficiently anonymized or even handed back. Which again can currently be enforced only by legislation because the *data subject* has no access to the collector's infrastructure where the data about her is stored.

In summary, approaching the issues from a legislative angle has shown to be fraught with problems on a variety of levels. Not least because it can hardly be proven that vendors behave accordingly. Thus *data subjects* should not depend on the collector's willingness to respect *Ownership* of *Personal Data* as stated here. Instead a technical approach is proposed to embrace the *data*

subject as the origin of her personal data and to proactively regain control.

2.2 Personal Data in the context of the Big Data Movement

The term **Big Data** initially has been referred to as a *huge amount of data*, containing more or less structured datasets [38], whose size, over time, has started to limit its usability because it exceeded the capabilities of state-of-the-art standalone computer systems and storage-capacity. Instead of reducing the overall size, the issues are addressed by utilizing distributed storing and parallel computing. Aside from challenges in logistic and resource management when for example information retrieval needs to get automated on a large scale [39], this strategy still doesn't answer the question of how to extract useful information from such deep "data lakes". What questions need to be asked to get answers whose usefulness has yet to be known of? To discover knowledge in order to back decision-making processes, technologies from the fields of *data mining*, *artificial intelligence* and *machine learning* (e.g. neural networks) have been adapted. Taken together, this is nowadays known as *Big Data (Analytics)*. Additionally it is a collective term for the practice or approach, described here, as well as the attitude of massively collecting data while tending to neglect people's privacy.

Big Data analytics serve the prior purpose of extracting useful information, whose results depend on the question initially being asked as well as what datasets the corpus contains. General steps involved in such a knowledge discovery process, can be outlined as follows [40] [41],

1. find and understand problems; formalize question(s) which the results have to answer
2. design data models accordingly and test/correlate them against sample data
3. collect and prepare data
4. process data (data mining)
5. analyse and interpret results
6. use discovered knowledge (e.g. make appropriate business decisions)

In general, the majority of businesses are required to have customer relationships. Such relations are based on the transfer of valued goods (e.g. services, products, etc.) in exchange for compensation (e.g. money). In order to process such a transfer, the vendor requires certain information about the involved customer. Since all entities related to this concept, including the customers, are considered to be human beings, such information most likely includes *Personal Data*. A business normally is eager to grow and, if it has commercial interests as well, it aims for profit maximization. So the business needs to improve and, therefore, it requires the knowledge of what and where its flaws are and how it can improve. To gain such knowledge, analytics based on Big Data approaches are part of various business strategies. But this also means that a lot of Personal Data get collected as part of those analytics, since that data is part of many business processes.

As a result of humans being primarily responsible for all money flow in this globalized environment, they also decide on the success of business. This basically means, every process analysis with an underlying commercial dependency somehow involves personal data [42] [43] [44] [45], whether this data is mandatory in that process or additionally obtained to,for example, measure and analyze customer behaviour. Common data points involved in big data analytics start with gender, age, residency or income, goes on with time series events like changing current geo-location or web search history and continues all the way up to health data and self-created content like posts, images, or videos. Depending on the data point though, those data might not be easy to collect. In general, most businesses obtain data from within their own platforms. Some data might even be in the customer's range of control (e.g. customer or profile data), but most of the data is created during direct (content creation, inputs) or indirect (transactions, meta information) interaction with the business. The sensitivity level of involved personal data is determined by how big the benefit is for the customer in comparison to what the vendor's demand is from the customer's commitment (e.g. required inputs, or usage requires access to location information).

From a technical perspective, collecting indirectly created data is as simple as integrating logging or debugging statements in the program logic. Since most vendors and organisations nowadays have a business that is partially happening through the internet or is even completely based on it, most sce-

narios of transactions utilize server-client architectures. Furthermore, the *always-on* philosophy has evolved to an imperative and implicit state of devices. Standalone software, installed on a personal computer, calls the vendor's infrastructure that is located in the cloud on a regular basis, just to make sure that its user behaves properly while, en passant, the vendor is provided with detailed user statistics. The web-browser already invokes a common narrative that requests happen here and there and are preventable. But when it comes to native mobile applications, it's almost impossible [46] to notice such behaviour or even prevent them from exposing potentially sensitive information. Those developments in architecture design have enabled a system-wide collection of potentially useful information on a large scale [47]. Logging transactions triggered by the user on the client and forwarding the resulting logs to the back end infrastructure, or keeping track of all sorts of transactions directly in the back end; all these collected chunks of data are then enriched with meta information before running together in a designated place where they are finally stored and probably never removed, waiting to be analyzed some day.

Within *Big Data Communities* sometimes the *big* is misinterpreted as, regardless of what the problem is that needs to be solved, speculatively gathering as many data points as possible with the hope that, in the future, those data might become valuable. This mindset is reflected by the oft-cited concept of the three *Vs* (Volume, Velocity, Variety) [48]. It is not entirely wrong though, because it's the nature of pattern and correlation discovery to provide increasing quality results [49], when the overall data corpus is enriched with larger and more precise datasets. When new technologies emerge and are hyped, questioning their downsides and potential negative mid- or long-term impacts is typically not a high priority. The focus is instead to experiment and try to reach and perhaps breach boundaries while beginning to evolve. Non-technical aspects such as privacy and security awareness don't come in naturally, instead the adoption rate has to increase before more and diverse research occurs that might uncover such issues. Only then can they be addressed properly and on different levels - technical, political, and social. Hence, the *Big Data Community* itself is able to evolve, too.

Big Data and Knowledge Discovery is a balancing act between respecting the user's privacy and having enough data at hand so that initial questioning

can be satisfied by the computed results. Aside from having specific domain knowledge of the used technologies, people working in these fields need to be aware of downsides or pitfalls and also have to be sensible about the ramifications of their approaches and doings. Such improvements are already happening, not only originating from the community's forward thinkers [50], but also advocated by governments and consumer rights organisations, as stated in the previous section (see *legislation*. Even leading Tech-Companies have begun trying to do better [51] [52] [53].

In various science and research areas it's also very common to gather and store enormous amounts of data. In these contexts the data and its analytical results are used to, for example, run complex simulations (e.g. weather, population, diseases, hardware, physics), monitor and analyse complex proceedings (e.g. nature, infrastructure, behaviour), explore the unknown (e.g. universe), or even to imitate a famous painter [54]. But unlike in the academic sector, the commercial interest within the private sector is much larger, therefore Personal Data are in great demand, because forecasting customer behaviour rather than global warming is much more valuable in that context.

It is the logical conclusion to distribute and scale horizontal when the data demand exceeds latest hardware capacity and capability, but it's no justification for thoughtless data collecting. Ultimately, it's not the amount of data that counts, but how it is handled (to not loose its usefulness); what data flows into those data lakes is up to the questioner, which might not act in the interest of everyone. Therefore *data subjects* need to regain control and actively participate in formulating these questions.

2.3 Personal Data as a Product

The *Big Data* paradigm itself, as mentioned before, merely provides a structured and technically-based method to uncover non-obvious or non-visible information from self-made data silos in order to assist in making (correct) (business) decisions. Though, when asking data collectors about their actual motivation, most likely the answer would be something along the lines of typical PR-phrasing like *"we want to have a better understanding of our customers"*. In the long run, this means to increase revenue, but in the

short term to do what exactly? Maybe to predict what might be the next thing people are supposed to buy, or what things they would probably like to consume but do not yet know of?

In order to try comprehending such perspective, lets take a look at some examples.

(A) An advertising service utilizes tracking data for targeted advertising. The more information the service has on an individual, the more accurate decisions it is able to make about what ads are the ones that the individual most likely will click on and disclose with a successful purchase. As a result the placed advertisement becomes more valuable to the advertiser, because of the high precision. This again causes the service to increase the charges for serving and presenting the ad, because the overall quality of its service - the product - has improved.

(B) Content recommendation engines of large streaming provider's, regardless of the content, serve as another example. This feature is also underpinned by extensive data aggregation of customer information (user profiling), such as consumption histories (e.g. watch list), favoured content, friend's consumption or any kind of trackable platform interaction. This means, the more information the service has on the user, the more precise are the assumption on current mood, taste and interests, which leads to more suitable recommendations. At the end the user feels well taken care of and therefor values the service.

(C) Another example is *Google Traffic* [55] [56], a service that is integrated as a feature in *Google Maps*, a web-based mapping service from Google. *Google Traffic* visualises real-time traffic conditions, while using *Maps* as a navigation assistant in order to provide the user with a selection of possible paths. These paths are enriched with a duration, which takes the traffic conditions into account. The data, required to offer such information, is supplied by mobile devices, constantly sending its current position including a timestamp to Google's infrastructure. This however, is only possible, because Google's services are widely used in addition to the significant market share of mobile devices [57] that are driven by Android - a mobile operating system developed by Google that deeply integrates with its services. The same assumptions can be

made as stated in the previous examples. The more geo-location data simultaneously is obtained, the more precise the information about traffic conditions are. Since this scenario demands real-time information, it adds the *time* component as an additional level of complexity to the problem.

Whereas the impact on the society of the first group of examples might be questionable, yet it seems to be in doubt if proper applications even exist, but adjusting the perspective reveals additional categories of scenarios, for example

(D) Planing and managing human resources for special occasions with big croweds, such as huge events or emergency situations where attendees might get in danger and need some help [58]

(E) Predicting infrastructure workloads (e.g. power grid) [59]

(F) Making more accurate diagnostics to improve patient's therapy [60] (

G) Finding patterns in climate changes, which otherwise would not have been detected [61]

Even though all described examples require a large corpus of data each and utilize Knowledge Discovery, some of them might not necessarily depend on Personal Data, whereas for other scenarios they are indispensable and yet others only implicitly rely on data collected from individuals. As noted in the previous section on Big Data[6], it depends on the purpose, which can be defined i.a. through a existing *business model*. But at least in those examples it seems to be common motivation to primarily improve and enhance the collector's product in order to satisfy its customers - yet again that depends on what is considered as product and who are the customers, generally indicated by the money flow.

Generalizing businesses based on its affiliation to an industry sector is hardly an for utilizing Personal Data, but empirically a rough attribution often suffices in order to get a picture of possible business models. With that said, the following observation can be made. When putting a Top 10 List of industries utilizing *Big Data* [62] and a visualization showing categories of

[6]personal-data-in-the-context-of-the-big-data-movement

personal data targeted by data collectors [63] side by side, it appears to be that at least seven[7] of these industries can be identified as data collectors, whereas less then a half[8] are participating in being a Data Broker, but almost all of them are suspected to target people's personal data, whether obtained by themselves or acquired from *Data Broker* (for more examples, see [64]). Therefore, it's save to say, that *Personal Data* is considered either as the actual product, especially from a Dater Broker's point of view, or indirectly due to its essential part in *Big Data* approaches. The former generates direct revenue by selling these data and the latter might affect a business in a positive matter.

To conclude, the possession of more and precise information on individuals leads to increased revenue for the possessor. Though,using personal data to improve a product becomes not necessarily an issue, unless the data owner is not the customer. Taking a closer look at the business model often reveals the role of personal data. Thus Personal Data becomes the currency and its owner becomes the product, whereas the possessor becomes controller and profiteer. This rather unsatisfying situation is going to be addressed further on. For example by shifting the individual's role to offering its own data to those who have previously collected and sold them.

2.4 Related Work

NOTICE: All research, projects, studies and work mentioned in this section represent only a fraction of what's already been done in this field and should be therefore seen as an excerpt containing a selection of the most related and relevant approaches.

The idea of a digital vault, controlled and maintained by the data subject, is not new. It holds her most sensitive and valuable collections of bits and bytes, protected from all the data brokers, collectors and authorities, while interacting with the digital and physical world, opening and closing its door

[7] Banking and Securities; Communication, Media & Entertainment; Healthcare Providers; Government; Insurance; Retail & Wholesale Trade; Energy & Utilities

[8] Banking and Securities; Communication, Media & Entertainment; Insurance; Energy & Utilities

from time to time to either put something important to her inside or releases important information for someone else. In the mid and late 2000s the growth of computer performance and capacity crossed its zenith (see Moore's Law [65]). At the same time, the internet started to become a key part in many people's lives and in society as a whole. Facilitated by these circumstances, *cloud computing* has been on the rise ever since, causing the shift towards distributed processing and patterns alike, thereby making it possible to rethink solutions from the past and try to go new ways, namely a breakthrough 2007 in *neuronal networks* courtesy of G. Hinton [66]. As a result, fields like *data mining*, *machine learning*, *artificial intelligence* and most recently combined under the collective term called *Big Data*, have gained a wide range of attention as tools for knowledge discovery. In almost any industry a greater amount of resources is invested in these areas [67].

The initial motivation for this project can be understood as a countermovement away from all the data silos in the cloud, returning to a focus on privacy, the individual, and her digital alter ego.

From simple middleware-solutions to full-fledged software-based platforms through to embedded hardware devices, a great variety of approaches have started to appear in the mid 2000s, continuing to this day. However a side effect was, over time, various research teams and projects have invented and coined different terms, which, in the end, all refer to the same (or similar) concept. The following list shows some of them *(alphabetical order)*:

- Databox
- Identity Manager
- Personal …
  - Agent
  - Container
  - Data Store/Service/Stream (PDS)
  - Data Vault
  - Information Hub
  - Information Management System (PIMS)
- Vendor Relationship Management (VRM)

One of the first occurring research projects was *ProjectVRM*, which originated from *Berkman Center for Internet & Society* at *Harvard University*.

As its name suggests, it was inspired by the idea of turning the concept of a *Customer Relationship Management* (CRM) upside down. This puts the vendor's customers back in charge of their data previously managed by vendors. It also solves the problem of unintended data redundancy - from the customers perspective. Over time the project has grown to the largest and most influential one in this research field. It has transformed into an umbrella and hub for all kinds of projects and research related to that topic [68], whether it's frameworks or standards, services offering (e.g. privacy protection), reference implementations, applications, software or hardware components. *VRM* became more and more a synonym for a set of principles [69], for example *"Customers must have control of data they generate and gather. [They] must be able to assert their own terms of engagement."* These principles can be found in various forms across much work done in this area of research.

Another work of research worth mentioning, because of the foundational work it has done, is the european funded project called *Trusted Architecture for Securely Shared Service* (TAS3) [70]. This project has led to an open source reference implementation called *ZXID*.[9] The major goal was to develop an architecture, that generalizes various approaches towards a non context-agnostic solution fitting into more sophisticated and dynamic scenarios, while still respecting commercial businesses (vendors) and users (customers), but at the same time facilitating a high level of user-centric security and privacy i.a. by a developed policy framework. Due to these requirements the architecture ended up being rather complex [71]. The implementation, called *ZXID*, involves several standards, for example SAML 2.0[10] and XACML,[11] and has just three third-party dependencies, *OpenSSL*, *cURL (libcurl)* and *zlib*. As of now it even supports Java, PHP and Perl. The project has lasted for a period of 4 years. After completion in 2011, research has continued i.a. by the *Liberty Alliance Project*. It is now part of the *Kantara Initiative* [72], including all documents and results. These results are occasionally referenced, i.a. by the IEEE [73].

A research project, which is probably the closest to what this work aims to

[9] more information on the project, the code and the author, Sampo Kellomäki, can be found under *zxid.org*

[10] Security Assertion Markup Language 2.0

[11] eXtensible Access Control Markup Language

create, bears the name *openPDS* [74] and is done by *Humans Dynamics Lab* [75], which again is part of *MIT Media Laboratories*. Despite the usual concepts of a *PDS*, it introduces multi-platform components and user interfaces including mobile devices, and also separates the persistence layer physically. This approach enables place- and time-independent administrative access for the data subject. Moreover, with their idea of *SafeAnswers* [76], the team goes a step further. The concept behind that idea is based around *remote code execution*, briefly described in one of the user stories in the first chapter. It abstracts the concept of a data request to a more human-understandable level: a simple question. This question consists of two representations: (A) a human-readable question of a third party, and

(B) a code-based representation of that question, which gets executed in a sandbox on the data subjects's *PDS* system with the required data as arguments. The data, used as arguments, is implicitly defined trough (A). The output of that execution represents both answer and response.

Aside from all the research projects done within the academic context, applications with a commercial interest have also started to occur in a variety of sectors. Microsoft's HealthVault [77], for example, which aims to replace the patient's paper-based medical records and combine them in one digital version. This results in a patient-centered medical data and document archive, helping doctors to make more accurate decisions on medical treatment, because they have more knowledge obtained, for example, from a personal medical history.

Meeco [78] [79], based on the MyData-Project [whitepaper_2014_mydata-a-nordic-model-for-human-centered-personal-data-management-and-processing], essentially just replaces platform-agnostic advertisement service providers with a closed environment and serves ads on its own. Although data subjects in this environment are provided with more control over what information they reveal, this approach doesn't take the eagerly demanded next step of getting rid of the advertisement market as a revenue stream and, instead, finding a suitable business model that focuses on the data subject, not surrounding them with just another walled garden.

A recently announced project, sponsored by Germany's *Federal Ministry of Education and Research*, but developed and maintained primarily by

Fraunhofer-Gesellschaft in cooperation with several private companies like *PricewaterhouseCoopers AG*, *Volkswagen AG*, *thyssenkrupp AG* or *REWE Systems GmbH*, is the so called *Industrial Data Space* [80]. The project unifies both, research and commercial interests and runs over a time period of three years until the third quarter of 2018. It aims *"[…] to facilitate the secure exchange and easy linkage of data in business ecosystems"*, where at the same time *"[…] ensuring digital sovereignty of data owners"* [81]. It will be interesting to see how these two related but distinct objectives come together in the future. Based on the white paper, the project's focus mainly seems to lie in enabling and standardizing the way companies collect, exchange and aggregate data with each other across process chains to ensure high interoperability and accessibility.

The following is a list of further research projects, work and commercial products regarding the topic of *personal data*:

Research

- Higgins [https://www.eclipse.org/higgins/]
- Hub-of-All-Things [http://hubofallthings.com/what-is-the-hat/]
- ownyourinfo [http://www.ownyourinfo.com]
- PAGORA [http://www.paoga.com]
- PRIME/PrimeLife [https://www.prime-project.eu, http://primelife.ercim.eu/]
- databox.me (reference implementation of the *Solid framework*[12])
- Polis (greek research project from 2008) [http://polis.ee.duth.gr/Polis]

Organisations

- Open Identity Exchange [http://openidentityexchange.org/resources/white-papers/]
- Qiy Foundation [https://www.qiyfoundation.org/]

Commercial Products

- MyData [https://mydatafi.wordpress.com/]
- RESPECT network [https://www.respectnetwork.com/]
- aWise AEGIS [http://www.ewise.com/aegis]

[12]https://github.com/solid/solid

2.5 Standards, Specifications and related Technologies

The overall attempt for this project is to involve as much standards as possible, because it increases the chances for interoperability and thereby it lowers the effort, that might be required to integrate with third parties or other APIs. Hereinafter, some of the possible technologies will be described briefly, and outlined what purposes they might be going service in addition to why they might be a reasonable choice.

HTTP [82], the well known stateless 'transport layer' for, among others, the *World Wide Web*. It most likely will fulfill the same purpose in the context of this work, because it implements a server-client pattern in its very core. Whether internal components, locally on the same host or as part of a distributed system, talk to each other or data consumers interact with the system, this protocol transfers the data that needs to be exchanged. Features that were introduced with Version 2 [83] of the protocol are yet to be known of their relevance for use cases within this project.
Unlike *HTTP*, **WebSockes** [84] provide the concept of ongoing bidirectional connections on top of TCP, though connection establishing utilizes the same principles known for HTTP(S). This combination still allows TLS while benefiting from high efficient real-time data exchange or from supporting remotely pending process responses, while at the same time avoiding HTTP's long-polling abilities. It is conceivable to use *WebSockets* for communicate between components or even with external parties.

JSON[13] is an alternative data serialization format to XML, heavily used in web contexts to transfer data via *HTTP*, whose syntax is inspired by the JavaScript object-literal notation. Like XML, its structuring mechanisms allow i.a. type preservation and nesting, which enable to represent more complex data structures including relations.

The open standard **OAuth** defines a process flow for authorizing third parties to access externally hosted resources, such as the user's profile image on a social media platform. The authorisation validation is done by the help of a previously generated token. However, generating and supplying

[13]The JavaScript Object Notation (JSON) Data Interchange Format; ECMA Standard [85] and Internet Engineering Task Force RFC 7159 [86]

such token can be initiated in a variety of ways depending on the underlying architecture and design, for example with the user entering her credentials (`grant_type=authorization_code`). This design seems to cause that *OAuth* is being misleadingly - but intentionally - integrated as an authentication service [87] rather then a authorization service; whether as an alternative or as an addition to existing in-house solutions. Therewith the application authors pass the responsibility on to the OAuth-supporting data providers. OAuth *version 1.0a* [88], which is rather considered a protocol, provides confidentiality by encrypting data before it gets transferred and integrity of transferred data by using signatures. Whereas *Version 2.0* [89], labeled as a framework, requires *TLS* and thus passes on the responsibility for confidentiality to the transport layer below. It also supports additional process flows for scenarios involving specific platforms, such as *"web applications, desktop applications, mobile phones, and living room devices"* [90].

With **OpenID** on the other side, the authenticity of a requesting user gets verified by design. An in-depth description of the whole process can be found in the protocol's same-titled specifications [91]. With decentralisation in mind, the protocols's nature encourages to design a distributed application architecture, similar to the idea behind a *microservice*, but without owning all services involved - *decentralized authentication as a service* so to speak. An application owner doesn't have to write or implement its own user authentication and management system, instead it is sufficient to just integrate those parts that are need to support signing in with *OpenID*, which is typically a client interacting with the Identity Provider. Equally the user is not required to register an account for every new app, instead she can use her *OpenID*, already created with another identity provider, to authenticate with the application. The extension *OpenID Attribute Exchange* allows to import additional profile data, similar to what OAuth is meant to be used for. *OpenID Connect* [92] is the third iteration of the OpenID technology. While for example facebook uses OAuth also for authentication (known as pseudo-authentication [93]) instead of just authorising entities, *OpenID connect* on the other hand, provides authentication in an additional layer build upon *OAuth2.0* and *JWT*. Previous versions of OpenID have provided the concept of extension in order add functionality such as accessing profile information. This is ability is no part of the core facilitated by OAuth, so that a user's

identity can share certain data with third parties via REST interface.

If it's required for certain components, while being part of a distributed software, to not maintain any state, either the architecture need to be changed so that the state at is no longer needed in that component, or embed the state into the process communication so that it is passed from one component to the other. This is a common use case for a **JSON Web Token** *(JWT)* [94]. A *JWT*, as it's name implies, is, syntactically speaking, formatted as *JSON*, but URI-safe encoded into *Base64*, before it gets transferred. The token itself contains the state. Here is where the use of *HTTP* comes in handy, because the token can be stored within the HTTP header and therefore can be passed through all communication points, where then data can be extracted and thereby verified. Such a token typically consists of three parts: (A) information about itself, (B) a payload, which can be arbitrary data such as user or state information, and (C) a signature; all separated with a period. Additional standards define encryption *(JWE*[14]*)* to ensure confidentiality, and signatures *(JWS*[15]*)* to preserve integrity of it's contents. Using a *JWT* for authentication purposes is described as *stateless authentication*, because the verifying entity doesn't need to be aware of session IDs nor any other information about a session. So, instead of the backend interface being burdened to check a state (e.g. `isLoggedIn(sessionId)` or `isAuthorized(sessionId)`) on every incoming request, in order to verify permissions, which required to maintain a state in the first place, it just needs decrypt the token and proceed according to the contained information.

When transferring data over a potential non-private channel, several features might be desired, which eventually provide an overall trust to that data. One important aspect might me, that no one else expect sender and receiver are able to know and see what the actual data is. **Symmetrical Cryptography** is used to achieve this. It states that the sender encrypts the data with the help of a key and the receiver decrypts the data with that same key. That is, sender and receiver, both need to know that particular key, but everyone who is not allowed to access that information must not be in possession of that key. To agree on a key without compromising the key during that process, both entities either need to switch to a private medium (e.g meet

[14]JSON Web Encryption, Internet Engineering Task Force RFC 7516 [95]
[15]JSON Web Signature, Internet Engineering Task Force RFC 7515 [96]

physically and exchange) or have to use a procedure, in which at any point in time the entire key is not exposed to others than sender and receiver. This procedure is called **Diffie-Hellman-Key-Exchange** [97] and is based on mathematical laws of modulo operations for when prime numbers are involved. It is designed with the goal to agree on a *secret* while at the same time using a non-private channel. The data, exchanged during the process, alone cannot be used to exploit the secret. Such behaviour is similar to the concepts of **Asymmetrical Cryptography** *(or public-key cryptography)* [98], which is underpinned by a *key-pair*; one key is *public* and the other one is *private*. Depending on which of keys is used to *encrypt* the data, only the other one can be used for *decrypting* the cipher. If then this technology gets combined with the concept of digital signatures (encrypted fingerprints from data), together it would provide integrity and authentication.

Putting *HTTP* on top of the *Transport Layer Security (TLS)* [99] results in **[HTTPS]{#}**. TLS provides encryption during data transport, which reduces the vulnerability to *man-in-the-middle* attacks and thus ensures not only confidentiality but data integrity too. *Asymmetric cryptography* is the foundation for the connection establishment, hence *TLS* also allows to verify integrity of the entity on the the connection's counterside, and, depending on the integration, it could even be used for authentication purposes. Though, relying on those cryptographic concepts requires additional infrastructure. Such an infrastructure is known as *Public Key Infrastructure (PKI)* [100]. It manages and provides keys and certificates for a dedicated scope of entities in a directory, including related information to the owners of these key and certificates. A Certificate Authority (or *CA*), as part of that infrastructure, issues, maintains and revokes digital certificates. The infrastructure that is needed to provide secure HTTP connections for the internet is one of those *PKI*s - a public one and probably one of the largest. It is based on the widely used IETF[16] standard *X.509* [101].

REST(ful)[17] is a common set of principles to design web resources and its interaction. It primarily defines server-client communication, in a more generic and thereby interoperable way. Aside from hierarchically structured

[16]Internet Engineering Task Force; non-profit organisation that develops and releases standards mainly related to the Internet protocol suite

[17]*Representational State Transfer*, introduces by Roy Fielding in his doctoral dissertation [102]

URIs, which can reflect semantic relations or hierarchical order between data points, it involves a group of rudimentary vocabulary[18] for HTTP requests to provide basic CRUD-operations[19] across distributed systems. The entire request needs to contain everything that is required to be processed on the REST-server, e.g. state information and possibly authentication parameter. A restful API typically has the purpose of exposing certain features provided through the platform or service to third parties in order to synergistically integrate with them. But utilizing these principals for all internal server-client interaction is also very common. This concept can also be understood as a proxy to the actual business logic in the back end.

The *QL* in **GraphQL** [104] stands for *query language*. Developed by Facebook Inc., its goal is to abstract multiple data sources into a unified API or resource, so that different storage technologies are seamlessly queryable without using it's native *QL*. The result is provided in JSON format, which naturally supports graph-like data structures. This is utilized in GraphQL and implicitly embraced through its purpose of abstraction. Data points that might be somehow related but stored in different locations, can be obtained so that both are end up in the same object through which they are related or indirectly linked to each other. The shape of a query is later mirrored by the result. GraphQL is not only an abstraction towards a more generic query language. It also separates almost any operation and the flow control from the the *QL* and moves it into a second layer. The so called *GraphQL* server is responsible for resolving and executing queries.

The term **Semantic Web** bundles a conglomerate of standards released by the W3C.[20] It is based around an idea called *web of linked data*, which aims to *"enable people to create data stores on the Web, build vocabularies, and write rules for handling data"* [web_2016_w3c_semantic-web-activity]. The standards address syntax, schemas, formats, access control and integrations for several scopes and contexts. Among others, the following three technologies are essential for the *Semantic Web*. RDF[21] basically defines the syntax.

[18]HTTP Methods or Verbs [103] (e.g. GET, OPTIONS, PUT, DELETE)
[19]Create, Read, Update, Delete
[20]World Wide Web Consortium; international community that develops standards for the web
[21]Resource Description Framework [105]

OWL[22] provides the guidelines on how the semantics and schemas (vocabulary) should be defined and with SPARQL [107], the query language, data can be retrieved. One can not help it, but picture the web as a database - queryable data with URIs that are embedded in arbitrary websites. Imagine a person's email address, which is available under a specific domain (preferably owned by that person) - or to be more precise, a URI *(WebID) [108]* - and provided in a certain syntax *(RDF)*, tagged with the semantic *(OWL)* of a email address, all together embedded in an imprint of a website. This information can then be queried *(SPARQL)*, if the unique identifier of that person *(URI)* is known. While defining the standards, a main focus was to design a syntax that is at the same time valid markup. The vision behind this: embracing the concept of a single source of truth (per entity) and embedding or linking data points rather then creating new instances of the same data that might only
valid at that point in time - in short, preventing redundant work and outdated data. Related to the *Semantic Web* is the a project called **Solid**.[23] Backed by the *WebAccessControl* [110] system and based on the *Linked Data* principles including the standards just mentioned, the project focuses on decentralization and personal data management while developing a specification around this. A reference implementation called *databox* [111] follows the specification process.

The concept of an application (or software) **container** is about encapsulating runtime environments by introducing an additional layer of abstraction. A container bundles just those software dependencies (e.g. binaries) that are absolutely necessary so that the enclosed program is able to run properly. The actual separation from the host system is done, aside from other technologies, with the help of two features provided by the Linux kernel. *Cgroups*,[24] which defines or restricts how much of the existing resources can be used by a group of processes (e.g. CPU, memory or network). Whereas *namespaces* [113] defines or restricts what parts of the system can be accessed or seen by a process (e.g. filesystem, user, other processes). The idea of encapsulating programs from the operating system-level is not new. Technologies, such as *libvirt*, *systemd-nspawn*, *jails*, or *hypervisors* (e.g. VMware,

[22] Web Ontology Language [106]
[23] social linked data [109]
[24] control groups [112]

KVM, virtualbox) have been used for years, but some of them were usually too cumbersome and never reached a great level of convenience, so that only people with a certain expertise have been able to handle systems build upon virtualization, whereas people with other backgrounds couldn't and weren't that much interested. Until *Docker* and *rkt* have emerged. After some years of separated work, both authors, and others, recently joined forces in the *Open Container Initiative* [114], which promises to harmonize the diverged landscape and start building common ground to ensure a higher interoperability. That in turn started to raise a demand for sophisticated orchestration. It also marks the initial draft of specifications for runtime [115] and image [116] definition for *application container*, which are still under heavy development. This concept of *containerization* also has the side-effect of making the application platform-agnostic, because it allows a certain set of software to run on a system which might otherwise not support that software (e.g. mobile devices); it just requires the runtime to be supported.

In the past years different countries around the world started to introduce *information technology* to the day-to-day processes, interactions and communications between public services and their citizens. Processes like changing residence information or filing tax report, are all summarized since then under the name *E-government*.[25] One of those developments is the so called **Electronic ID Card**{#def–eid-card}, hereinafter called *eID card*. Equipped with storage, logic and interfaces for wireless communication, those *eID cards* can be used to store certain information and digital keys, or to authenticate the owner electronically to a third party without being physically present. Such an *eID card* was introduced also in Germany in 2010. The so called *nPA*[26] has been an important step towards an operational *e-government*. Aside from minor flaws [117] and disadvantages [118] an *eID card* might come along with, the question here is, how can such technology be usefully integrated in this project and does it somewhat plausible to do so. As an official document, the card has one major advantage over inherent, self-configured or generated secrets like passwords, fingerprints or TANs.[27] It is *signed* by design, which means, by creating this document and handing it over to the related citizen, the third party (or *'authority'*) - in this case

[25] Electronic government
[26] in german so called *elektronische Personalausweis (nPA)*
[27] Transaction authentication number

the government - has verified the authenticity of that individual.

When communicating via email it's already common to encrypted the transport channel, but using *asymmetric cryptography* for encrypting emails end-to-end is rather unusual. The equivalent to a *PKI* would be basically a *public key server* that follows a concept called *web of trust*. In which all entities (user; senders and recipients) are signing each other's public keys. The more users have signed a public key, the higher the level of trust is on that key and therefore on the entity who the signed key-pair belongs to. Public keys are simply uploaded by its owners to those key servers mentioned before. If someone want to write an email to others, all public keys relating the recipients must be obtained from these public server, so that the email can then be encrypted with those keys and are therefore only readable by the owner of the keys's private part.

Related to that topic, another technology emerging as part of the *e-government* development, is the german **De-Mail** [119]. It's an eMail-Service that is meant to provide infrastructure and mechanisms to exchange legally binding electronic documents. One would expect a *public key cryptography*-based implementation from sender all the way over to the recipient [120]; maybe even with taking advantage of the *nPA's* capability to create *QES*, which refers to the ability of using the *nPA* to sign arbitrary data. Instead, the creators of the corresponding law decided that it's sufficient to prove the author's identity based on its credentials when handing over the email to the server, and that it's reasonable to let the provider signs the document on the email server, and finally that this described implementation results in a legally binding document by definition of that law. The different levels of mistakes made in conception and legislation are outlined in a statement from CCC[28]-members [120], who have been consulted as official experts during the development of that law.

[28]Chaos Computer Club e.V.

3

Core Principles

Right from the start a set of principles have built the cornerstones and orientation marks of the idea behind the *PDaaS*. Those, who are meant to be reflected also by the upcoming *Open Specification*, will be explained further within the following sections.

3.1 Data Ownership

Depending on the perspective, the question about ownership of certain data might not that trivial to answer. As stated in a previous section, ownership requires a certain amount of originality to become intellectual property, which is not the case for personal data - at least for all the non-creative content. Thus there is no legal ground for an individual to license those data that obviously belongs to her. Switching the perspective from the *data subject* to the data *consumer*; for them, several laws exist addressing conditions and rules regarding data acquisition, processing and usage. Leaving aside the absence of any legislation regarding data ownership, it cannot be denied, that is seems unnatural not being the owner of all the data that

reflects a person's identity and the person as an individual. So instead of defining those rules meant to protect data subjects but actually demands data consumers to comply with, the proposal here is to put the entity, to whom the data is related to, in control of defining, who can access her data and what are accessors allowed to do with it. This would make the *data subject*, per definition (see Ownership) and effectively to the owner of those data. Although, it is to be noted, that the legislation for data consumers mentioned before, remains a highly important, since this project is not able to cover every single use case, that might occur.

Promoted from the data subject to the data owner and therefore being the center of the *PDaaS*, the *operator* of the *PDaaS* gains abilities to have as much control as possible over all the data related to her, in order to determine in a very precise way what data of hers can be accessed by third parties at any point in time, and also to literally carry all her personal data with her.

3.2 Identity Verification

When an instance of such a system *(PDaaS)* is going to be the digital counterpart of an individuals identity or her *"personal agent"* [121], then everyone who relies on the information that agent provides, must also be able to trust the source from where that data originates and vice versa; the *operator* too need to verify the authenticity of the requesting source; regardless if it's the initial registration or further *access attempts*. Based on these mechanisms, the system can also provide an authentication services that third parties might utilize to outsource the authentication of its owner, including enabling additional security factor steps.

3.3 Reliable Data

Being able to verify the authenticity of a communication partner only means to be half-way through. Data consumers also need to trust the data itself, which is attributed by the following properties. (A) *integrity* - which means the recipient can verify that the received data is still the same, or if someone

has tampered with the obtained data. (B) *authenticity* - it is somehow ensured, or the recipient must be certain, that the received data actual belongs to the individual, meaning the data truly reflects attributes of the individual, from whom the data comes from. A negative result of that check should not cause a termination of the process, but instead should warn data consumers unverified authenticity, so that they themselves can decide if and how to proceed.

3.4 Authorisation

Controlling it's own data might probably be the most important ability of such a system, because the data owner gets enabled to grant permission to any entity who wants to obtain certain information in a potentially automated way about her. She can authorise as precise as desired how long and what data (sets, points or fields) is accessible by a single entity. Thereby, the data owner is able to change the *access permissions* for any entity at any point in time, for example motivated by a noticed incident.

3.5 Supervised Data Access

Rules and constraints might be one way to handle personal data demands of third parties. But a plain *query-and-respond-data* approach could be replaced by a more goal-driven concept, that prevents data from leaving the system. It allows to execute a small program within a locally defined environment, computing only a fraction of a larger computation initiated by the *data consumer* beforehand; similar to a distributed Map-Reduce concept [122]. The opposite but also conceivable approach would be to provides either some software to the *data consumer*, which is required in order to access the contents of a response, or a runtime environment that queries the system by itself. In general, it is not very likely that data consumers, who already got granted certain access, would renounce their privileges. Thus it is vital that the data owner is the one in charge of canceling access permissions or applying appropriate changes to them. Supervising methods provide an ap-

propriate ways to make data available to those who are eager to consume them.

3.6 Containerization

Abstracting an operating system by moving the bare minimum of required parts into virtualizations results in an environment setup that can, depending on the configuration, fully encapsulate its internals from the host environment. This approach yields to some valuable features, such as

(A) Effortless portability, which reduces the requirements on environment and hardware
(B) Higher flexibility in placing components, through which advantages can be made out of characteristics that other devices might bring with
(C) Isolation and reduction of shared spaces and scopes, which for example can prevent side effects

All these in conjunction lead also to an overall security improvement, or at least it enables new patterns to improve those aspects. Furthermore, it opens up for more versatile and diverse scenarios, like storing data decentralized, scaling during unexpected workloads or getting used as a medical record due to higher security precautions. The convenience of precisely assign environment resources might also become relevant for cases where device's hardware specification might be somewhat low. Building a system upon a container-based philosophy and enclosing components in their own environment offers a variety of design and architectural possibilities without the necessity of increasing the overall system complexity.

3.7 Open Development

When developing an *Open Specification* it only comes natural to build upon open technologies, which are recognized as *open standards* and *open source*; *open* in the sense of *unrestricted accessible by everybody*, not to be confused with free - as in *freedom* - software. Advocating such a philosophy allows

not only to develop implementations in a collaborative way, but also enables to work fully transparent on the specification itself. Such an open environment makes it possible for anyone who is interested, to participate or even to contribute to the project. Thus, to lower the barrier, usable and meaningful documentation is vital. Such an openness ensures the possibility of people looking into the source code and getting a picture of what the program actually does and how it works. Even source code reviews therefore become feasible and any security flaw thus uncovered can be fixed immediately. Furthermore, this approach allows data subjects to setup their own infrastructure and host such a system by themselves, which gains even more control over the data and increases the level of trust, instead of using a *SaaS*[1] solution that is host by possibly untrusted entity. It also encourages any kind of adjustments or customization to the software in order to serve operator's own needs. Enabling an open development enables users and contributors working together and collectively improve the project in a variety of ways.

[1] Software as a Service

4

Requirements

Derived from the Core Principles, the subsequent requirements shall be served as a list of features on the one hand, to get an idea of how the open specification and thus the resulting software might look like, and on the other hand to give an overview about priorities (can/could, may/might, should, must/have to). Other chapters may contain references to specific requirements listed below.

Architecture & Design:

S.A.01 **- Accessibility & Compatibility**
Since the internet is one of the most widely used infrastructure for data transfer and communication, it is assumed that all common platforms support underlying technologies, such as HTTP and TLS. Thus the emerging system must implement a service based on web technologies, that provides supervised access to personal data.

S.A.02 **- Portability**
All major components should be designed and therefore communicate with

each other in a way so that individual components are able to get relocated while the system has to remain fully functional. It must be possible to build a distributed system, that may require to place certain components into different environments/devices/platforms.

***S.A.03* - Roles**
The system has to define two types of roles (see also Terminologies). The first one is the operator, who is in control of the system and, depending on the architecture, must be at least one individual but could be more. The operator maintains all the data that gets provided trough the system, and decides which third party gets access to what data. The second type is a consumers. These consumers are external third parties that desire to access certain data about or from the operator. While the consumer has to interact with the system only via plain HTTP requests, the operator must be provided with graphical user interfaces for possibly multiple platforms.

***S.A.04* - Authenticity**
Since they have to rely on the data, both entities - everyone who belongs to one of the *roles* - have to be able able to verify the authenticity of the opposite' identity as well as of the received data, in order to be certain that this data is real and belongs to the sender. It should be possible to opt out to that level of reliability if it's not necessary, or to opt-in selectively for certain types of data. However, if one of the parties demands the other one to provide such level, but the other doesn't, then the access attempt has to fail.

***S.A.05* - Availability**
When third parties are requesting data, it's very likely that those procedures are triggered automatically or at least machine-supported. Hence those requests can arrive at the *PDaaS* at any point in time. Therefore the *PDaaS*, or at least parts of it, should be available all the time. Even if the request can't be processed completely, the system is still able to inform the *data subject* about that event; somewhat like an answering machine. This also enhances the *PDaaS* as a serious and reliable data source. It also relates to the topic of *failure safety and redundancy*.

***S.A.06* - Communication**
In order to elaborate on S.A.01 and S.A.02, to ensure internal interoperabil-

ity, all communication between components has to happen on HTTPS, if they don't run in the same environment.

Persistence:

S.P.01 **- Data Outflow**
Data may leave the system only if it's absolutely necessary and no other option exists to retain the goal of the relating process. But if data still has to get transferred, no other than the data consumer must be able to access that data. Confidentiality has to be preserved at all cost.

S.P.02 **- Data Relationship**
Data structures and data models must show high flexibility and may not consist of strong relations and serration. If the syntax of the data representation also provides structure elements then it should be utilized to also embed semantics.

S.P.03 **- Schema and Structure**
The operator can create new data types (based on a schema) in order to extend the capabilities of the data API. Structures and schemas can change over time (S.P.04). Every dataset and data point has to relate to a corresponding type, whether it's a simple type (string, integer, boolean, etc.) or a structured composition based on a schema.

S.P.04 **- Write**
Primarily the operator is the one who has permissions to add, change or remove data. This is done either by using the appropriate forms provided by a graphical user interface or with the help of other import mechanisms. The latter could be enabled by (A) supporting the upload of files that containing supported formats, (B) data APIs restricted to the operator or (C) defining an external resource reachable via http (e.g. *RESTful URI*) in order to (semi-)automate further ongoing data imports from multiple data resources (e.g. IoT, browser plugin). Additionally, it might be possible in the future to allow *data consumers* making some data to flow back into the operator's system, after she is certain that this data is valid and useful for her.

S.P.05 **- Data Redundancy**

Providing and managing data is the core task here. Hence the system needs to make backups or at least provide mechanisms and tools for the *operator* to do that. Different strategies are conceivable but have to respect related requirements (S.P.01, S.A.03) and specific environment conditions though. The least feasible solution would be a manual backup only allowed by the *operator*.

***S.P.06* - Data Precision**

As stated in the previous chapter about *Data as a Product*, the level of precision of data has a significant impact on the *data subject's* privacy. Therefore the *data subjects* must be provided with the ability to configure how precise certain data should be when it is provided to *data consumers* somehow. Those adjustments must have no impact on the actual stored data. That is, adjustments have to be made after the data is fetched but before it is further processed.

Interfaces:

***S.I.01* - Documentation**

The interfaces from all components have to be documented and well specified so that the components themselves could be individually replaced without any impact to the rest of the system. This also involves comprehensive information on how to communicate and what endpoints are provided, including required arguments and result structure.

***S.I.02* - External Data Query**

Data consumer can request a schema, in order to know how the response data will actually look like, since certain parts of the data structure might change over time (see S.P.03, S.P.04). In order to check if the access request is permitted, the system first parses and validates the query, and eventually proceeds to execution. When querying data from the system, the *data consumer* might be required to provide a schema, which should force him to be as precise as possible about what data exactly needs to be accessed. In addition to that, the consuming entity must provide some *meaningful* text, describing the purpose of the requested data. He should not be allowed to place wildcard selectors for sets of data into the query. Instead he must

always define a more specific filter or a maximum number of items, if the query retrieves more then one element.

***S.I.03* - Formats**
When components communicate between each other or interactions with the system from the outside take place, all data send back and forth should be serialized/structured in a JSON or JSON-like structure.

Graphical User Interface:

***P.VIU.01* - Responsive user interface**
The graphical user interface has to be responsive to the available space, in order to acknowledge the divers market of available screen sizes.

***P.VIU.02* - Platform support**
All graphical user interfaces must be implemented at least based on web technologies, which are then provided by a server and available on any system that comes with a modern browser. To enable additional features and deeper integration with the surrounding environment, it is recommended - at least for mobile devices - to build user interfaces upon natively supported technologies, such as *Swift* and *Java*. The operator would benefit from capabilities such as *push notifications* and storing data on that device.

***P.VIU.03* - Permission Profiles**
The operator should be capable of filtering, sorting and searching through a list of *permission profiles*; for a better administration experience and to easily find certain entries in a continuously growing list of profiles.

***P.VIU.04* - Access History** The operator must be provided with a list of all past *permission and access requests*, in order to monitor who is accessing what data and when, and thus being capable of evaluating and eventually stopping certain access and data usage. Such tool should have filter, search and sort capabilities. It is build upon and therefore requires the access logging functionality.

Interactions:

P.I.01 **- Effort**
Common interactions for use cases, such as changing *profile data*, importing datasets or managing *permission requests* have to require as little effort as possible. This means short UI response time on the one hand and as less single input and interaction steps as possible to complete a task. Given the circumstances (e.g. distributed architecture, involvement of mobile devices), the *permission request review* and *permission profile creation* might become a special challenge to hide certain complexity.

P.I.02 **- Design**
Graphical user interfaces must be designed and structured in such a way that is is highly intuitive for the user to operate. Thus, it is important for example to use meaningful icons and appropriate labels. This also refers to a flat and not crammed menu navigation. Context related interaction elements should be positioned within the area related to that context.

P.I.03 **- Notifications**
The user should be notified about every interaction with the system originated by a third party immediately after its occurrence, but she has to be notified at least about every *permission request*. This behaviour should be configurable; depending on the *permission type* and on every *permission profile*. Regardless of the configuration, notifications themselves must show up decently and pending user interactions must be indicated in the user interface. Notifications do not necessarily require a reaction by the operator.

P.I.04 **- Permission Request & Review**
A process involving data transaction must be always initiated by the data subjects. So, before a *data consumer* is able to access data, first the *operator* needs to *invite* that specific third party by either telling him (per URI) where to register or asking him to provide all information required for a registration process upfront encoded in QR-Code. Both solutions need a secure channel for transport, which refers to *TLS*. The latter has to be ensured by the applying third party alone, whereas the first option requires the *PDaaS* to provide a TLS-supporting endpoint as well. After a successful registration, the consumer can submit a *request permission*, which has to

include information about the *consumer*, what data he wants to access, for what purpose and how long or how often the data needs to be accessed. The operator then reviews these information and creates an *permission profile* based on that information. A verz important attribute in such a profile has to be the definition of when this permission expires. The operator should be able to decide between three *permission types*:

- *one-time-only*
- *expires-on-date*
- *until-further-notice* After creating the profile, a response must be send to the *data consumer*, which should contain the review result and the determined permission type.

***P.I.05* - Templating**
The operator should be able to create templates for *permission profiles* and *permission rules* in order to (A) apply a set of configurations in advance before a *permission request* arrives and

(B) reduce recurring redundant configurations and interactions.

Behaviour:

***P.B.01* - Access Logging**
All interactions and changes in the persistence layer should be logged. At data request must be logged. Such log is the foundation of the *access history*, so that the user is able to keep track of occurring accesses and look up past ones.

***P.B.02* - Real time**
Real time communication might be essential for time-critical data transaction. Hence graphical user interfaces should be communicating with the server through an ongoing connection to enable real time support. Assuming the following example scenario: permission request got reviewed on mobile device, but notification indicator in the desktop browser still reflects 'pending'. If just one of the user interfaces has no real-time capabilities but all the others do, the user interface might get into an undefined state presenting the user with wrong information, which will decrease user experience

dramatically. This mean, either all user interfaces have to provide real-time feedback or non.

5

Design Discussion

The following chapter documents the processes of a variety of design decision makings, examines possible issues emerging alongside and discuses different solutions obtained from several perspectives in order to evaluate their advantages and disadvantages. Probably not every issue gets its deserved room, but major aspects will be addressed. In short, the majority of the project's conceptual work is done below.

The end of every subchapter includes a section containing a summary of conclusions based on the prior discussions related to that topic of that subchapter.

5.1 Authentication

First of all, the system has to support two roles. Either one or non of these roles is assigned to an entity, hence entities that are trying to authenticate to the system might have different intentions depending on the characteristics of those roles. The *operator* for example wants to review *permission requests* in real time, so accessing the system from different devices is a common sce-

nario. When inheriting the *operator role* an entity gains further capabilities to interact with the system, such as data manipulation. Whereas a *data consumer* always uses just one origin and processes requests sequentially. Those very distinct groups of scenarios would make it possible to apply different authentication mechanisms that not necessarily have a lot in common.

With respect to the requirements (S.A.01), the most appropriate way to communicate with the *PDaaS* over the internet would be by using *HTTP*. Thus, to preserve confidentiality on all in- and outgoing data (S.P.01) the most convenient solution in this case is to use *HTTP* on top of *TLS*. *TLS* relies i.a. on asymmetric cryptography. During the connection establishment the initial handshake requires a certificate, issued and signed by a CA, which has to be provided by the server. This ensures at the same time a reasonable level of identity authentication, almost effortless. If the certificate is not installed, it can be installed manually on the client. If the certificate is not trusted (e.g. it is self-signed), it can either be ignored or the process fails to establish a connection, depending on the server configurations. The identity verification in TLS works in both directions, which means not only the client has to verify the server's identity by checking the certificate. If the server insists on, the client has to provide a certificate as well, which then the server tries to verify. Only if the outcome is positive, the connection establishing succeeds. According to the specification [123] it is still optional though.

HTTP as a comprehensive and flexible protocol enables to use several technologies for server-client authentication purposes. Some of them are build-in, others can simply be implemented on top of it. Within the scope of this work, those technologies are categorized in the following types: (A) stateful and (B) stateless authentication. The first one (A) includes vor example *Basic access Authentication* (or *Basic Auth*) and authentications based on *Cookies*. Whereas the *two-way authentication* in TLS mentioned above and authentication based on web-token are associated with the latter (B). *Basic Auth* is natively provided by the *http-agent* and requires in its original form *(user:password)* some sort of state on the server; at least when the system has to provide multitenancy. If instead just a general access restriction for certain requests would suffice, no state is required. One of the most common implementations of user-specific states is a *session* on the server, that contains one or more values representing the state and a unique identifier, by

which an entity can be associated with. A client has to provide that session ID in order to be provided with all session-related data hold by the server. This is typically done in a HTTP header, whether as *Basic Auth* value, a *Cookie*, which is domain-specific, or in some other custom header. Since the *two-way authentication* (or *mutual authentication* [124]) is performed based on files containing keys and certificates, which are typically not very fluctuant in its contents or state, this procedure is categorized as stateless. Order or origin of incoming requests have no impact on the result of the actual authentication process. The same applies to TLS features such as *Session [ID, Ticket] Resumption* [125], therefore they are left aside, because they serve the sole purpose of performance optimization. Similar to the *Session Ticket Resumption* [126] a web token, namely the JSON Web Token, also moves the state towards the client, but that's about all they have in common. A *JWT* carries everything with it that's worth knowing, including possible states, and if necessary the token is symmetrical encrypted by the server. That is, only the server is able to see the actual data contained by JWT and therefore reacting accordingly.

Keeping track of one or multiple states on the server and maintain the synchronization of involved data between server and client is expensive and by fare trivial, because this pattern requires the server to be aware of all current states (sessions) and has to have them accessible at all time. The additional resources required for such an approach were referred to as being expensive. It also means, that requests, ultimately resulting in responses that contain contents, might depend on preceding requests and their incoming order. Furthermore those session data has to be in a consistent state in order to ge safely stored from time to time. Otherwise, if the server fails to run at some point, data only existing in the memory would be gone with no chance to get recovered. To a stateless authentication non of those aspects applies. Certificates and keys as well as web tokens are both carry the information with them that might be necessary. Thus, considering those disadvantages, *public key cryptography* and web tokens are the preferred technologies for all authentication processes.

Except for *two-way authentication* all authentication technologies mentioned above require an initial step from the client to obtain some sort of token that is used to authenticate all subsequent requests. This step is commonly

known as *login* or *sign in* and requires the authorizing entity to provide some credentials consisting of at least two parts. One part, that uniquely relates to the entity but doesn't have to be private, and another part that only the entity knows or has. Typically that's a username or email address and a password or some other secret bit sequence (e.g. stored on and provided by a USB stick). An *eID card* could possibly be used as secret (or unique object) as well. Suitable use cases therefor are (A) to let the *operator* login to the *PDaaS* management tool or

(B) to approve or authorize *access requests*. How the actual login process (A) would look like partially depends on the *eID card*'s implementation, but in general this use case would make sense. When considering the german implementation *(nPA)* for example, accessing the management tool via desktop requires also a card reader, preferably with an integrated hardware keypad. Whereas authenticating to the tool with a mobile device could be achieved with the card's RFID-capabilities, as long as the used device is able to communicate with the RFID-chip. Both scenarios (A+B) required the *nPA* to have the *eID* feature enabled. If a service wants to provide *nPA*-based online authentication *(eID-Service)*, which is defined as a non-sovereign *("nicht hoheitlich")* feature, it has to comply with several requirements [127] starting with applying for a permission to send a certificate signing request to a BerCA.[1] This request is send from an *eID-Server* [128] in order to get a public key signed, which priorly has been generated on a dedicated and certified hardware. This hardware is requested by the officials as part of a *eID-Server*. The key pair - re-generated and re-signed every three days - is needed to establish a connection to the *nPA*, which is then used to authenticate the owner of that *eID card*. The described procedure appears to be highly expensive (regarding effort, hardware costs etc.), especially because every single *operator* would needs to go through the whole process in order to support this authentication method; not mentioning the uncertainty of the official's decision on the permission application. Another approach could be to integrate an external authentication provider supporting the *nPA*, which would not only add an additional dependency, but also weaken the system.

[1] (german) Berechtigungszertifikate-Anbieter

All two scenarios are fairly similar, insofar as they would use the same mechanism to initially authenticate to the system, but with different intentions.

Because of its simplicity the concept of web tokens are fairly straightforward to implement into the *PDaaS*. But since web tokens ensure integrity and optional confidentiality only of their own carriage but not for the entire HTTP payload, both requirements need to be addressed separately. Serving HTTP over *TLS* solves that issue though. For connections that use a web token, it should be sufficient to rely on the public PKI that drives the internet with *HTTP* over *TLS*. All information required for the actual authenticate are provided by the token itself. Though, the situation is different if *two-way authentication* is used instead. Therefor, the system has to provide its own *PKI* including a Certificate Authority that issues certificates for *data consumers*, because not only the *endpoints* on the *PDaaS* (server) need to be certified, all *data consumers* (clients) need to present a certificate as well. Only the *PDaaS* verifies and thus determines (supervised by the *operator*) who is authorized to get access to the system. Hence the *PKI* needs to be selfcontained and private in order to function independently so that only invited parties can get involved. Referring to the statement mentioned above, *data consumer* have as well to be able to verify the identity of the *PDaaS*, in order to prevent man-in-the-middle attacks. Addressing this issue basically means, *data consumers* have to verify the certificate presented by the *PDaaS*. This can be done in two ways. One is, a certificate has been installed on the *PDaaS* that is certified and therefore trusted by a trustworthy public CA, as mentioned above. Then *consumers* use the CA's certificate to verify the *PDaaS* certificate. The other way is, to let the *PDaaS* create and sign a public key by itself. Before *consumers* then get presented with the self-signed certificate of the *PDaaS* during the initiation of the TLS connection, they already have to be aware of that certificate. That is, *consumers* need to be provided with that certificate on a private channel upfront. Otherwise would the process be again vulnerable to man-in-the-middle-attacks.

In summary, if a public-key-based connection, performing a *two-way authentication*, establishes successfully, it implies that the identity originating the request is valid and the integrity of the containing data is given. Whereas on a token-based authentication every incoming request has to carry the to-

ken so that the system can verify and associate the request with an account. Furthermore data it not automatically encrypted and thus integrity is not preserved.

An advantage of token-based authentication over TLS-based *two-way authentication* is that the token can be used on multiple clients at the same time. Or an account, a token is associated with, can actually have more than one token. Whereas during the asymmetric cryptography-based *two-way authentication* the client's private key is required, which results in 1:1 relation. So if it's likely that a *operator* has several clients, regardless for what purposes, then the same private key has to be on those clients. Though, a private key typically does not leave its current system or at least does not exist in multiple systems at the same time in order to prevent exposure, which any action of duplication implies. To reduce those risks, it's common practice to generate a private key at that location where it is going to be used.

OpenID, a open standard for decentralized user authentication, also uses subdomains as unique identifiers to associate with entities that need to be authenticated, similar to the approach proposed in this work. But since it originates in a very distinct scenario it is also very related and therefore restricted to that. Trying to adopt the standard might result in various adjustments to this project leading to an implementation that shares not much compliance, which is not the intention of a standard.

The technology *De-Mail* tries to ensure authenticity of an author's identity by embedding a legal foundation into email-based communication. But instead of providing technically valid authenticity through end-to-end encryption so that a recipient can truly rely on that information, it only goes as far as legal definition and legislation reaches. Thus it has no relevance to this work, other then the concept of letting a server sign outgoing data, which might be the only solution to avoid an overhead in user interaction caused by recurring events.

Computations based on asymmetric cryptography usually is slower then the ones based on symmetric cryptography [129], but since there are no timing constrains when interacting with the *PDaaS*, regardless of whether it's external communication with *data consumers* or internal between components,

parameters for cryptographic procedures can chosen as costly as the system resources allow them to be, thus the level of security can be increased.

Conclusions: Based on the several requirements and distinct advantages of the two authentication mechanisms, it is preferred to use asymmetric cryptography in combination with *HTTPS* for the communication between the system and *data consumers*, where the system provides its own *PKI*. Whereas a token-based authentication on top of *HTTPS* and public CAs should be suitable for communication between the system and the *operator*, preferable based on *JSON Web Tokens*, because the session state is preserved within the token rather then having the system itself keeping track of it. Though, it is worth mentioning, that a JSON Web Token implementation is feasible as well to fully replace the approach of *two-way authentication* and private *PKI*. The disadvantage here would be, whether *data consumers* are able to authenticate themselves or not, a HTTPS connection will establish in any case. At the same point, authenticating the *operator* is as well doable on the TLS layer; but this approach is restricted only to trusted environments like native mobile applications, because browser-based applications are not considered trusted and they missing of certain capabilities. Addressing the requirement of *consumers* to verify whether the certificate, presented by the *PDaaS*, can be trusted or not, both solutions, providing self-signed certificate on a secure channel upfront or using certificates certified by publicly trusted entities are legitimate. Although, the latter requires a service or an automatism that provides a new signed certificate whenever a new *data consumer* registers, such dependencies should be kept to an absolute minimum. To hardening an authentication procedure often one or more factors are added, for example an *eID card* or one-time password. This adds complexity to the procedure and thus increases the effort that is needed to perform a successful attack. But equally it also increases the effort to support those factors in the first place. Using multi-factor authentication is generally valued and will be briefly noted as an optional security enhancement for the *operator role*. However detailed discussions regarding this topic are left to follow-up work on the specification.

5.2 Data Reliability

Within the section about authentication it was discussed how to preserve data integrity - referring to possible man-in-the-middle attacks and alike. Furthermore it was described how to authenticate the different user roles so that their identities are ensured, though, authenticity of the actual data a *PDaaS* provides has yet to be provided. Data authenticity here refers to authentic and reliable (S.A.04) data, which means (A) the data really represent the entity that is associated to the originating *PDaaS* and is thus owned by that entity, and (B) the data is true at that moment when the data, attempted to be accessed, is queried from within the system. Since the *operator* can change the data at any point in time, this property requires a process where a trustworthy third party has to be able somehow to verify the reliability of the data in question. That process on the other hand, is in direct contrast to the discussion about the authentication system and why it should be designed to be selfcontained. But if providing information on data reliability is not required, it won't be an issue anymore. The information can be defined in a response as an optional property. Within the request the *data consumer* has to indicate whether the response should contain information about its reliability or not. Depending on what data is requested, the *PDaaS* then decides to test the reliability. Based on the procedures that are available, the data reliability gets verified somehow.

But how does this reliability check exactly should look like? It comes down to two general steps. The first one is matching the actual data involved in that request against a reference dataset. The second step is optional, although important for the *data consumer* in order to evaluate the sufficiency of the provided level of reliability. It involves the party, that also runs the first step, to confirm the result of that audit. The result of that evaluation then gets included in the response.

The following proposed methods vary in the provided level of reliability as well as in the amount of effort to support them and in the possible impact to its surrounding system. Not all data points are necessary to test for reliability. Profile data for example are more likely to be tested, whereas consumption lists or location histories are more or less hard to verify, because currently there is no reliable way to verify the origin of those datasets, besides from

being not a primary focus here.

(1) **Local Verification by matching**
The probably simplest and at the same time least reliable method is to just look at the existing data tha is stored in the database and matches them against those data that is used to create a response.

(2) **Local Verification and signing**
An electronic ID card can serve as an authentication token for the *operator*, but it can also be utilized to verify the reliability of certain data. Using the german implementation *(nPA)* as an example, the *eID* feature would provide access to the owner's basic profile data, which thus can be used to match against those data points that are both, hold by that *nPA* and going to be used to create the response, originating in the *PDaaS* persistence layer. If the result of that matching procedure is positive, the related data then gets signed with a *QES* courtesy of the *data subject's nPA*. That signature also gets included in the response, so that the recipient can verify the reliability of the data.

(3) **Remote Verification and signing**
Another method involves a third party who also has the same data that needs to be tested. The idea is to hand the data in question over to that party, who then tries to match against all those data points available in that context. The party also has the ability to sign data, which is what's happening, if the matching procedure has a positive outcome. It is required to sign the whole response or at least a replicable dataset that contains the data that were initially required. The party then hands everything back to the *PDaaS* for further processing.

(4) **Recurring Certification**
The following method describes a modification of (3). This method involves no matching procedure. The external third party, verifies whether the data in question are correct and whether they relate to the *data subject* by either literally looking at the data or by automatically processing a matching against their databases. If that party is satisfied, a certificate will be issued. This certificate contains an expiration date, which implies the consequence of going through this process again in the future, much like an issuing process of a common *Certifi-*

cate Authority. This certificate is then served as part of a response, which enabled the *data consumer* to verify the data's reliability on its own. This is done by hashing the data in question, decrypting the hash embedded in the certificate and matching one against the other. If they are equal, the data has not changed since the party's review and is therefore reliable. Though, if data has changed in the *PDaaS* and data points are affected, that are also included in this verification process, then a new certificate needs to be issued, because the containing hash is now invalid.

Only one method per request should be used to verify data reliability, because every method can imply a different level of confidence. As described above the response send back to the *data consumer* has to indicate the used method. Based on that the *data consumer* is then able evaluate that level und can act accordingly (e.g. verify a signature).

Expanding those verification procedures is reasonable, but to keep it simple for now this aspect won't receive further attention, since the current requirements are sufficiently met. It will left to future work, though.

Conclusions: The signing procedure as part of local verification method involve private key and certificate stored on the operator's *eID card*. Every time the *PDaaS* verifies data reliability that method has to be went through. Thus the *operator* is forced to interact wit the *PDaaS*. Otherwise the operator's private key needs to be stored somewhere within the *PDaaS*. No matter where or how, that would potentially expose a highly confidential part of a cryptographic procedure. Not only would this reduces the overall security level of the system, it also makes every task this method is involved in vulnerable to certain attacks. Aside from that, it's highly unlikely that an *eID card* would allow to extract it's containing private keys. This all results in increased inconvenience which is inevitable for this proposed method. The *Local Verification and signing* method has the same dependencies too mentioned in the discussion about requirements for using the (german) *eID card* as an authentication token. And since it was rejected because of those dependencies and because of the inconvenience mentioned before, that verification method eventually is not going to be supported in the specification.

The *Remote Verification and signing (3)* method would require the external

party to be an official authority, because no other entity has (A) the data in question (primarily profile data), which makes them (B) legally binding; and they are commonly trusted. The same goes for The *Recurring Certification (4)*, but while the *Remote Verification and signing* method introduces a very strong dependency to that external party, the *Recurring Certification* offers a simple loosely linked dependency. Whose design would make it even possible to obtain such a certificate manually but automate it on the other side as well. Nevertheless, both provide a trustworthy certification.

Finally the first method, *Local Verification by matching (1)*, which just performs a matching of two datasets against each other. Those datasets are obtained from the same *PDaaS* storage, but at different times; right before the request finally gets proceeded, though. The primary purpose of addressing the issue of data reliability rests on the *data consumer's* concern of accessing data that is intentionally tampered with (e.g. by the *data subject*), since integrity during transport is already ensured. In this light, and even if the whole system would be compromised, which in that case might need more urgent addressing then ensuring data reliability, it won't mitigate the fact, that the *operator* is the only one able to change data. Hence it provides the lowest level of reliability.

Certain fields of application of a *PDaaS* as a data resource might already impose some constraints about the level of reliability and maybe even how it can be provide. Violation of relevant legislation or other rules might prevent the *PDaaS* from being put in use. Others instead - depending on their guidelines and business model - don't rely on a certain level of confidence. In general, *data consumers* are expected to already have a basic confidence in a *PDaaS* and in the data originating there. Regardless of that, providing an indication on the authenticity of data is valued as a first and important step towards a fully working feature. All of the proposed verification methods have some downsides, Though, the *Recurring Certification* method would be the least invasive and therewith an adequate choice.

A primary goal for the *PDaaS* is to preserve all data owned by the *data subject* and giving her control over where the data might go; not providing sufficient proof for the data authenticity.
Though, it is still important, to provide *data consumers* with an information

about the level of reliability, but it is up to them how to rate that information and how to act on that.

5.3 Access Management

In the subsequent section it will be discussed, how several processes around the topic of *data consumers accessing data on the PDaaS* can be modeled, what consequences certain variations might have and what issues need to be addressed.

Below a general design is proposed of how *data consumers* get authorized and thereby are able to access the *data subject's* personal data, and how previous mentioned technologies can be assembled in order to meet the specified requirements.

Part One: consumer registration

0) The *operator* creates a new unique URI in the system

1) **Prepare registration**; the *operator* has to tell the third party were and how to register as a *data consumer* by handing over a URI uniquely associated to the current registration process. *Several things need to be noted here. First, the operator 'pulls' consumers into the system. This is the only way for a consumer to establish a relation. If consumers were able initiate this process on their own without the operator's involvement, it would be much harder for the system to detect spam or fraudulent requests. Second, handing over that URI must be done over a secure channel.*

2) **Send registration attempt**; the third party then makes the actual attempt to register as a *data consumer* by providing required information. Those information have to be include some kind of feedback channel (e.g. URI) so that the system can get back to that third party. They also may contain a first *permission request.*

NOTICE: the two initial steps can be skipped if the third party would rather

present the information mentioned in step 2), as a QR-Code so that the operator* can obtain it and thereby is able to proceed. This approach would short cut and hence simplify the process.*

3) **Review registration attempt**; the operator gets notified about a new registration attempt, which she then has to review and decide whether to accept it or not.

4) **Create permission profile**; if a *permission request* is enclosed in the registration, it has to be reviewed as well. If it's not, step follows immediately. If permissions are granted a new *permission profile* is created. Optionally, it could also be created if the permissions were refused. It's just meant for the *operator* to keep track of her decisions.

5) **Respond to third party**; regardless of the decision, the third party get's then informed via feedback channel about th decisions and is also provided with further details required to obtain actual data.

Part Two: obtain data

0) A successful registration as a *data consumer* is required

1) **Send request**; the *data consumer* sends *access request* to the system, containing a all information about what data is needed, how to process the data and what the response should contain.

2) **Parse and check request**; after the system has received an *access request*, first it
authenticates the *data consumer* and checks related *permission profiles*. According to the defined *access rules*, the system decides how to proceed. Either it pauses, because it needs further attention from the *operator*, or it can start to process and create the response.

3) **Compute response**; how the computation would look like mainly depends on the contents of the *access request* and also on what a *permission profile* determines (see 'types of access requests' below).

4) **Respond to consumer**; handover the computed response back to the requester by proceeding with one of the two following options. Either

the system responses with a current status of the process and where the *consumer* will/can find the demanded data, or the *consumer* includes a callback URI, which the system has to invoke with demanded response.

With respect to the requirements (S.P.01), personal data should not leak into the outside. To tackle this issue, the following three types of *access requests* are defined, starting with the most sufficient solution:

a) **Supervised Code Execution**; *access requests* additionally come with an executable program - binary or source code - potentially including information about provisioning. After the required data is retrieved from the storage, the program gets invoked with the data locally on the system but within a completely separated environment *(sandbox)*. The result of that invocation gets returned to the system.
b) **Data DRM**[2]; after data is retrieved from the storage it gets encrypted. The cipher is included in the response. Upfront, *data consumers* are equipped with a small program, that can connect to the *PDaaS* and has to wrap the *consumer's* own software that is planned to proceed the requested data. Now when *consumers* receive the response, the program needs to get invoked with the cipher, so that, by priorly fetching the key from the *PDaaS*, the cipher can be decrypted from within the invocation. Thus the data is made available to the wrapped software and only during runtime. After the invocation has finished the program needs to propagate the results that are returned by the software back to the outer environment.
c) **Plain Forwarding [default]**; retrieve data from the storage, quick-checking the result and forwarding it directly into response.

So, the data won't leave, unless the *PDaaS* doesn't support any of the proposed request types or the *data consumer* provides no alternative, so that the fallback type must be applied. If that's the case, the overall confidentiality of all personal data is still preserved, because all communications from and to the *PDaaS* are generally happening over HTTPS anyway, so that the data is encrypted during the transport.

The concept of authorizing a data consumer to get the ability of accessing

[2] *Digital Rights Management* - set of technologies, that are used to control access to data or content that is restricted in certain ways (e.g. content provided by video streaming)

personal data is fairly trivial. During (or after) the *registration* consumers have to provide detailed information about their intentions, so that the operator is confident about their permissions when reviewing them. The created *permission profile* reflects the result of that review. Such a permission profile defines what data points are requested and eventually permitted to access, how often they can be accessed and how long those permissions last. The latter is defined as *permission type* and is either one of the following:

one-time-only access permissions are hereby granted for just a single *access request* (with tolerating certain errors regarding the communication layer)

expires-on-date access permissions are hereby granted until the defined point in time has arrived

until-further-notice access permissions are hereby granted until the *permission type* has changed or the *permission profile* has been deleted

NOTICE: The default permission type should be configurable. The operator can change all permission profiles at any point in time.

Among other information, an access request contains the *data query* that shows very precisely what data points are affected by that request. So if an *access request* arrives at the *PDaaS*, assuming the *data consumer* has been authenticated sufficiently, the systems (0) searches for a permission profile that correspond to the *data consumer* and the requested data points. If it fails to find one, the access request gets refused. But if it does, then it checks (1) if the permission type suffices at that moment and (2) if the query only contains data points that are enabled in the profile as well. Here the order does matter, because it is imaginable that the operation behind (1) is less complex then operation (2). So, at the end running (1) before (2) can result in a lower response-time, if operation (1) already results negative. If all operations have a positive result, access is granted.

As stated in the section about data reliability, the *data subject* is able to add, change or remove all her data or even the *permission profiles* at any point in time. This raises the question of how to solve the situation were *access requests* are being processed, while those changes are happening and might affect the result of those requests. The first and simplest approach would be to not address this issue at all, but that would be unreasonable, because

providing data to the *consumer* normally means for the *data subject* to get something in return or to somehow benefit from that. So that approach is no option. Using a failure of reliability verifications as a mechanism to re-request data won't work either in that case, because it would be based on a wrong assumption, since that failure can have multiple causes, not only the issue here in question. A stateless solution seems to be not fitting due to the time-related dependency. So the only currently perceivable way is to keep track of all momentarily processing or pending *access requests* to detect those who are affected by the changes so that each of them can be aborted and processed again. Here it is important to determine the right moment, when all changes are done, otherwise the system might end up restarting those computations repeatedly within a short amount of time. The described issue relates to both, *personal data* and *permission profiles*, because either can impact the response that is returned to the *data consumer*. Furthermore, it needs to be ensured that only after *permission requests* are being reviewed and *permission profiles* are being created, the *data consumers* receive their credentials or a notification to get started.

It is up to *data consumers* to decide which data they are requesting to access, but how do they know what data can be requested? The only option is to expose information about data availability (S.I.02), which can be done in a variety of ways. First, those information can be made publicly available via URI, providing a Machine-readable format, so that information can be processed automatically by consumers. It is also feasible to restrict that access to registered *consumers* only, in order to prevent those information from being crawled. They could be valuable as meta data and therefore used in undesired processing that could raise privacy concerns. It is imaginable to let the *operators* restrict the access to such availability information on an individual consumer-basis or system-wide, and furthermore to set default configurations for this behaviour. Depending on those configurations request might fail, thus requester need to be provided with meaningful errors. Http error codes [130] might be a sufficient fit for that purpose.

An already standardized way to implement authorization is the OAuth Specification. And since the TLS layer is already in place to handle authentication, the choice would be to use version 2 of the standard, because it relies on HTTPS. Only two of the four *grant types* provided by OAuth would

match with the process design introduced before. The types are `password` and `client_credentials`, which basically require identifier(s) and secret or credentials to directly obtain a `token`. The other two types define additional steps and interactions involving consumer and operator before getting the `token`, because these other two types are intended to be used for processes involving untrusted environments (e.g. browser, third party apps). Aside from not fitting into such scenarios it would also make the proposed process undesirably more complicated. Although the proposal includes user interactions like selecting and confirming requested permissions. According to the documentations [131] [132], both OAuth versions (1.0a and 2) require the client (here *data consumer*) to register to the authorization server upfront (to obtain a `client_id`), before initializing the authorization process. However, as stated before, the concept of the *data subject* 'pulling' a *data consumer* towards the *PDaaS* is preferred over letting *data consumers* try to 'push' themselves towards the system. The reason to prevent undesired registration attempt, is, because they all have to be reviewed by the *data subject*. Furthermore, it is not within the scope of the OAuth Specification to define how this should be accomplished. Thus, such step needs to be added in addition to an entire OAuth-Flow, which might cause otherwise avoidable overhead in user interactions. Moreover, the proposed design does not include that specific registration process either. Instead, this process is not needed at all, because according proposal, client identification happens implicitly as a result of how the resource owner (operator) obtains the registration request from the client (see *Part One: consumer registration, step 0 to 2*). Further investigations show that the semantic of an `access_token` from the perspective of an resource server consists of (A) authentication [does this token exist?] and (B) authorization [Is this token valid and what does it permit?]. Those aspects are in part already provided by the proposed way of using the TLS layer. Because every data consumer has its own endpoint to connect with the *PDaaS* and the certificate used by the *consumer* is singed by a signature only used for that endpoint. This means, the consumer is already authenticated, when the TLS connection has successfully established. And since *permission profiles* relate to a specific endpoint, it would make providing an `access_token` to become obsolete. To summarize, implementing OAuth would introduce several mechanisms that otherwise can be provided by the combination of *two-way authentication* in TLS, dedicated

endpoints and certification.

Conclusions: In the preceding text, various solutions were developed, based on which, the following three solutions are being at disposal:

a) OAuth 1.0a and HTTP
b) OAuth 2 and HTTPS (public Certification and PKI)
c) HTTP over TLS with *two-way authentication*, private PKI, subdomains as dedicated endpoints

The solutions a) and b) require an extra step in which data consumers need to register themselves at the *PDaaS*. This already must be done on a secure channel to prevent man-in-the-middle attacks. Furthermore in that step does option a) obtain a symmetric key for creating signatures used to ensue confidentiality and integrity in subsequent steps. All those cryptographic procedures need to be adopted when implementing the here proposed specification and also when interacting with those implementations. While this can cause much more harm, it is proposed to leave as much of these sensitive parts as possible to existing implementations who have already proven themselves. Thus HTTPS is mandatory, which makes b) more suitable over a), because it's also more flexible and easier to implement. Whereas solution c) moves the complete authentication procedure to a different layer. It hence results in separating authentication and authorization from each other, leaving no remains of relation. This opens the authorization design up to for example other implementations that might be more suitable for certain *data types*. In addition, it would only require little effort to support the case where multiple *data consumers* share the same *endpoint* and thereby the same *permission profiles*. Combining b) and c) would result in significant redundancy, since both solutions have much overlap in their provided features, even though b) aims to be a framework for authorization. The process description in the beginning of this section is used as the foundation of *access management* in the *PDaaS*. Implementing OAuth based on this design would leave nothing from the framework, but a simple request returning an identifier associated with permissions. And even these identifiers are obsolete when combining TLS with dedicated *consumer*-specific endpoints, as c) states. So, there is not much benefit in using OAuth, other then developers might be somewhat familiar with the API. This can be addressed by a detailed specification for

this project, hence c) is preferred over b). At the end, the only suitable use case from this work would consists of just a request that obtains a token after authenticating with the provided credentials. And since OAuth only provides a framework for how to authorize third parties to access external resources, but leaves the procedure of how to actually verify those access attempts up to its implementers [133] [134]. In the context of this project OAuth does not comply with the rest of the design aspects.

How the first steps of a registration are look like, is up to the *consumer*, even though the option involving a QR-Code might result in a nicer user experience from the perspective of a data subject. In any case, a secure channel is vital.

When accessing personal data, at the same time preventing those data from leakage is almost impossible, which originates in the nature of digital data being able to get effortlessly copied. Nevertheless it is possible to make it much more difficult, so that it becomes inefficient to bypass those mechanisms. At the same time it requires some effort to establish, run and maintain the infrastructure needed for those mechanisms. In case of the *Data DRM* proposal that effort is not proportionate, because it requires additional infrastructure, interfaces and cryptographic procedures and introduces therefore new attack scenarios. For now the only approach being considered, is the *Supervised Code Execution*, aside from defaulting to simple forwarding. When implementing this approach, two directions might need to be considered. Alongside the executable program, *data consumers* either provide all dependencies so that everything is bundled up, or don't provide any dependency at all. The latter is preferred, because it reduces the amount of potentially malicious, flawed or needless components, so that the data subject, supported by her *PDaaS*, has more supervising capabilities and thus more control over her personal data. Since the overall goal here is to prevent the data subject from loosing control over her data, it might also be conceivable, that certain categories of personal data, representing a higher level of sensitivity, also require a least sufficient *request type*. If the data consumer does not comply, access will be refused. Also, depending on which category the personal data relates to, the *PDaaS* might be able to somehow anonymize certain types of data, if it is even capable of doing so, because the *consumer* at least supposedly knows what individual is behind

the *PDaaS* it is currently interacting with. The field of *data anonymization* is a large research area on its own, which recently started to gain a lot of traction due to emerging privacy concerns about *Big Data*. Thus it will be left for future work.

5.4 Data

The core task of a *PDaaS* is providing data, *personal data*, which in conjunction is the digital manifestation of an individual, a person. One party creates the data, another one obtains and processes it. Thus, both need to agree, or at least need to know, how that data looks like, how is it structured and what are their semantics. The following section is intended to discuss different technologies, used to create queries that obtain those data points that are desired. Further on, it describes some basic data types and schemas, that might be useful in the context of *personal data* and also for previously introduces scenarios.

First of all, to address the need of portability, which has to be satisfied by those components, who are storing and providing *personal data*, it is essential to abstract the actual storage from how it gets accessed. This makes it possible to relocate those storage onto other platforms and environments. Thereby the *personal data storage* itself becomes platform-agnostic from an outside perspective, in other words portable. In order to reduce possible issues related to unsupported communication protocols it might be reasonable to enforce HTTP - over TLS, if they don't share the same environment - even if the storage therefor requires an additional driver or proxy layer, like for example a mobile app.

Possible technologies are for example *GraphQL* or the *SPARQL*, which is part of the *Semantic Web Suite*. Both are query languages underpinned by the concept of a graph. This means, relations between data points are embedded within the data structure itself. That meant, in terms of a graph, relations are *edges* and data points are *nodes*. In consequence the structure of a query itself reappears in its result, which means the originator of that query knows exactly what to expect for the response. Therefore it's not necessary to provide any additional information about how to handle and

interpret the responded data. The code examples (Code 01 and Code 02, Code 03 and Code 04) give a first impression of how it might look like, when a *consumer* obtains the name of the *data subject* and a bank account of hers, that supports online payment.

Without going into much details here, the syntax of the SPARQL query (Code 01) already shows its nature of decentralization. This aspect at the same time introduces additional external dependencies. Because the query language itself has no concept of schemas or any kind of semantic, it needs to be made aware of them. SPARQL queries typically return XML which then can be rendered into (HTML) tables. JSON and RDF are also supported. The reason for performing two queries in the example instead of just one, is, because otherwise the result would have returned multiple 'rows' with redundant data, if more then one bank account would have supported online payment; varyingly in those three columns data contain data about bank accounts, but being identical in the fields related to the profile information, though.

Whereas the GraphQL query syntax (Code 03) compared with its result (Code 04) shows of a remarkable resemblance. By defining `paymentMethod` as an argument, the resolver for `bankAccounts` then implements an instruction to match the value of that argument (`'online-service'`) against the whole set. GraphQL's server then knows from which resources the data in question need to be pulled and how they need to be aggregated. While SPARQL has a full-featured query language syntax including all sorts of controls (e.g. aggregation, operators, nested queries etc.), GraphQL's syntax instead is more rudimental, because all its functions and logic has been abandoned from the language itself and put into a server part. With this concept of separation it is straightforward to validate queries, because it essentially means matching against types. Both query languages share a comprehensive understanding of a type-system, that encourages to create all kinds of data types. While in GraphQL common schemas still need to be created, SPARQL already provides a reasonable amount of vocabulary [135]. However, when comparing the results of both languages, some distinctions appear. While in GraphQL the characteristics of graph-structured data are remain, SPARQL's output is missing a certain level of depth. The reason for that originates in the design of the query language and its syntax. SPARQL is able to

notice implicit relations between data points, though its query language is not capable of grabbing and presenting them. Thus the result (Code 02) only consists of two dimensions.

It is crucial for the *PDaaS* to provide the *data subject* with abilities to create her own data types and schemas (S.P.03). Thereby she is enabled to serve data points according to her own needs and terms. In order to interact with their customers or users, *data consumers* might as well develop and provide schemas for their requests. This can help *data subjects* to speedup the process of permission granting and to easier understand what data points are affected. Data types and schemas are the key to validate incoming and outgoing data. If data violates the underlying schema or no appropriate schema exists, the data transfer fails. Other missing data types could be developed by a community, because not every *data subject* might be capable of modeling her own data types. Thus everyone can benefit from that effort taken by a few. As a result, the ones that are widely used might then become de facto standards. Moreover, it's also possible that several data types will emerge, that are based on common standards, for example *medical record* [136], *point of interest* [137] or *bank transfer* [138]. With that approach those data types can be viewed as something like a plugin or add-on to the *PDaaS* ecosystem.

In order to avoid confusion about the differences between types and schemas and to simplify their relations, the following two definitions are henceforth being used. A (data) *type* is the superior term; hence refers to both of them.

```
*Float* or *Nil (null)*
```

```
*primitives*, but can consist of other structs as well
```

Based on these two concepts almost any imaginable data type can be modeled. A selection of such types can be found in the list of suggested structs (List 01), whereas an extract of (sub-)categories that might be useful in a *PDaaS* are specified in a list of data categories (List 02). Additional examples for *structs* are a *data subject's* profile (Code 05), a contact (Code 06) and bare position information (Code 07). All those examples and lists are only to be understood as a starting point that should cover basic scenarios as well as to give a first impression of what data types a *PDaaS* could provide.

List 01: Suggestions for useful structs

- Address
- Contact
- Location
  - Country
  - City
  - Position
- Media
  - Audio
  - Video
  - Photo
- Organisation
- Date
- TimeRange
- Language
- Diseases

List 02: relevant (sub-)categories of data

- Finance
  - Income
  - Bank transfers
- Shopping history
- Product
- Things/Objects
- Media consumption
  - Music playlist
  - Watchlist
- Favorites/Interests
  - Music genres
  - Songs
  - Movies
  - Books
  - Travel destinations
  - Topics
- Curriculum vitae (CV)

  - Educational level
  - Visited schools
- Visited …
  - points of interest
  - countries
  - websites/URLs (browser history)
- Units (measurements)
- Organisations
  - Company
  - Bank
  - …
- Medical/Health Record
  - Diseases
  - Treatments
  - Visits to the doctor
  - Medication

The available *primitives* mainly depend on those who are supported by the query language itself. In this case, all *primitives* mentioned above are supported by *SPARQL* [139] and *GraphQL* [140]. When choosing a database system it has to be ensured that either the system already supports the required *primitives* or they can be emulated somehow with a least amount of drawbacks. When modelling relations between data points one can use for example keys (or identifiers) to make reference, or additional syntactical tools like *lists* (or arrays) and maps (or objects). Those tools facilitate readability so that relations are almost intuitively observable, hence they should be enforced. Whereas another known concept in data modelling, called *inheritance*, isn't required, but could help to reason about certain *structs* and their representations. It might add complexity, though.

Aside form the subject's personal data other information and data must be persisted as well. This includes for example:

- Application data
  - Templates (P.I.05)
  - Permission profiles (incl. versioning)
  - Consumer information

  - Meta data
  - Notifications
  - States
  - Tokes
  - Access logs
- Files
  - Cryptographic keys
  - Executable program
  - Container images
  - Configurations
  - User interfaces
  - Documentations

The list revels that not only a database system is needed to satisfy the requirements, but the environments filesystem might need to be utilized as well. This leads to the question of what requirements a database system has to satisfy. But first of all it is pivotal to distinguish between the needs of a *personal data storage (PDS)* and a general *persistence layer (PL)* for the system's backend.

Table 5.1: selection of characteristics that a database system has to feature in order to be suitable for either of the defined purposes

Characteristic	Personal Data Storage	Persistence Layer
portable	-	-
advanced user & permission management	-	**X**
document-oriented	**X**	**X**
support common primitives	**X**	**X**
replication	-	**X**
efficient binary storage and serialization	**X**	**X**
high performance	-	**X**
operations and transactions	**X**	**X**
background optimization	-	**X**
version control	-	-

Although, most of the characteristics (in Table 5.1) are self explanatory, certain aspects need to be commented on. Fist, portability, an important requirement (S.A.02), that is oddly not marked in Table 5.1. This is because of the priorly introduced concept of abstracting the *personal data storage* with an additional query language. Therefore the access to the *PDS* becomes platform-agnostic. Whereas the database system storing that data can be implemented with respect to the requirements while considering the environment constraints at the same time. Basic permission management should suffice the *PDS*, since it's not accessed in multiple ways. It only relates to the query language abstraction. Data and especially its structure is expected to be highly fluctuant (S.P.02), thus advantages of relational databases (e.g. schema-oriented and -optimized) would instead harm the performance and flexibility, because they are not primarily designed to handle schema changes. Database systems, whose storage engine is build upon a document-oriented approach, would be a better choice. Replication can be utilized for horizontal scaling, federation or backups (S.P.05). The focus here is on the latter, because without *PL* the *PDaaS* wont be able to function. In case of irreversible data loss, the whole system state is gone, which then has to be reconfigured and reproduced from the ground up. Such effort can be spared by introducing a reliable backup strategy. With the *PDS* on the other hand replication is not necessary, but to ensure no data loss still needs to be addressed. Therefor every database system that might be used for the *PDS* must provide a mechanism to backup or at least to export the data, which can be triggered and obtained through the operator's management tool. Another approach imaginable could be to not only store the actual data written to the *PDS* but also to save all queries in a chronological order that have somehow changed the data. Therefore the current state can be restored just by running those queries against the *PDS*. It is reasonable to store the queries of the abstracted query language not the ones the query language is transformed into. If a mobile device is part of the *PDaaS*, another approach for the operator could be to perform regular device backups. Those strategies are all just initial thoughts which might be sufficient only as a starting point. Other solutions are imaginable. Though, elaborating on those is beyond the scope of this work. Depending on the technologies that are being used, it might be necessary from a conceptional perspective to split the *PL* into two parts. One part is a database system and the other

is represented by the environment's filesystem. It might be no alternative when it comes to key files, which are typically accessed as files, or configuration files for certain technologies. In any case, both *PL* and *PDS*, have to be able to store files of any kind, which is required for instants to support the secenario of medical records. File size restriction should be mandatory though, because the *PDaaS* has no intention to replace existing *file hosting* solutions.

Being able to undo changes of certain data points or to review the change-history of those data can be very useful; not only when those changes were persisted mistakenly. This behaviour might not be necessary for every data, especially when it comes to application configuration or logs. Also, not every *operator* might require these features. Therefore, and because database systems with no alternative might not be able to provide this capability, it's not required by either *PDS* nor *PL*. If a history is not natively supported but still desired, it has to be considered if for example high frequently backups would already suffice or if a implementation on the application-level is required.

Before serving data it needs to be at first inserted into the *PDS*. This can be done in three different ways:

a) The *data subject* uses forms provided by the graphical user interface to insert data about her, for example her profile information (Code 05). This data is then submitted into the *PDaaS* which takes care of storing it.
b) The *data subject* is in possession of file(s) or string(s) containing a data format that is supported by the system. The graphical user interface provides a mechanism to either upload the file(s) or insert the string(s). Thereby the data is then send into the system. If this raw data is not self-explaining to the system the *data subject* has to provide more information on the context of those data.
c) Third party software, for example a browser plugin, is used to provide the *PDaaS* with data; in this case a browser history. This software uses a restricted API provided by the *PDaaS* to let data flow into the system.

These three concepts, especially b) and c) are required to get inspected for

malicious content and extensively validated against existing *structs*. Only if these checks do not fail, the submitted data can be stored. For scenario b) the *data subject* needs to be asked to review the imported data to make sure everything is as expected. When enabling third party software to submit data, appropriate authentication and permission mechanisms must be in place. That software is classed like all other *operator* front ends, but without permissions to obtain data. According to the discussion on authentication, these mechanisms thus can be implemented through *JWT*.

Conclusions: In order to gain flexibility in choosing technology and location for the *personal data storage*, the logical consequence is to abstract the interface from the database system. Introducing a separate query language is proposed as a reasonable approach. It can be chosen between two suggested query languages, *GraphQL* or *SPARQL*. Both provide the necessary features required to integrate them with a distributed system; *SPARQL* with its concepts of URIs as identifiers and resources, and *GraphQL* with its separation of query definition and execution. This also effects the process of query validation, which is much harder to do for *SPARQL*, because its syntax is more flexible and allows some shorthands, therefore the possibilities are way to many. In general *SPARQL's* syntax is harder to reason about compared to *GraphQL*. And even though the result of both languages is formatted in JSON, only *GraphQL* preserves all the relations in the output, which are already embedded in the query syntax. In result, *GraphQL* (and its implementations) is the query language of choice for this project.

When it comes to creating new structs engaging a user community can compensate the lack of certain types. Examples for a potential starting point of *PDaaS*-supported data types were showed before. Data Modelling in general is a large research field to its own. With regards to the *PDaaS* it needs much more attention, though it's beyond the scope of this work. The basic approaches within this section should only be viewed as an introduction that gives an outlook of how it's imagined.

5.5 Architecture

By taking all requirements as well as previous sections, their and discussions into account, this section has the purpose of figuring out how all the different concepts and conclusions from this chapter can fit together in an overall system architecture that is organized in either a distributed or a monolithic manner. The outcome of this section should not impact results or conclusions from other topics related to the behave of the system's interfaces from a user point of view.

The foundation of this project is a server-client Architecture, which is chosen for (A) providing availability (S.A.05) and (B) separating concerns [141]. Such a distributed system provides various locations to place these concerns, which are in fact different environments with different properties. Those combinations of locations and environments are herein after called *platforms*. To further describe these *platforms*, characteristics such as architectural layer and access possibilities to its internals are taken into account. The resulting three *platform* types are shown in Table 5.2.

Table 5.2: All platform types where components of the *PDaaS* architecture can be placed

Type	trusted	private	controlled by	Layer	Purpose
Server	yes	yes	data subject	back end	- business logic - third-party interfaces - data storage
Desktop	no	no	data subject	front end	- based on web technologies
Mobile	no	cond.	data subject	front end	- typically mobiles devices - based on host-specific native technologies - data storage

The next step is to determine all the components, that are required in order to cover most of the defined use cases. The conglomeration below highlights all major components, including information in which platforms they could be positioned, in addition to further details about their major task(s), underlying technologies and relation(s) to each other.

Web server

Platform: Server

Tasks:

- serve web-based user interface(s)
- handle all in- & outgoing traffic (outmost layer)
- revers proxying certain traffic to different components
- en- & decrypt HTTPS traffic, thus authenticate *consumers*
- load balancing (if necessary)
- desktop notification
- spam protection

Relations:

- operator, consumers, third parties
- any front end platform (Desktop, Mobile)

Technologies:

- HTTP
- TLS
- WebSockets

Permission Manager

Platform: Server

Tasks:

- creating *permission profiles*
- permission validation
- examine data queries
- queue *consumer* requests

Relations:

- Storage Connector
- Notification Infrastructure
- Persistence Layer
- Tracker
- Code Execution Environment

Technologies:

- any modern language/framework capable of parallel computing

PKI

Platform: Server

Tasks:

- CA
- manage keys and certificates per *endpoint*
- obtain trusted certificates from public CAs

Relations:

- Web Server

Technologies:

- X.509
- ACME [142] (Let's Encrypt)

Storage Connector

Platform: Server

Tasks:

- abstracts to system agnostic Query Language
- queries personal data, regardless of where it's located

Relations:

- Personal Data Storage

Technologies:

- driver for used database

Operator API

Platform: Server

Tasks:

- authenticates *operator*
- writes personal data through Storage Connector
- provides relevant data, such as history
- system configuration
- automated data inflow

Relations:

- Storage Connector
- Notification Infrastructure
- PKI
- Tracker
- Permission Manager

Technologies:

- JWT

Code Execution Environment

Platform: Server

Tasks:

- isolated runtime (sandbox) for computations/programs provided by *consumers*
- restrict interaction with outer environment to absolute minimum (e.g. no shared filesystem or network)
- one-time use

- monitor sandbox during computation
- examine and test the provided software

Relations:

- Permission Manager

Technologies:

- Virtualization
- Container (OCI)

Tracker

Platform: Server

Tasks:

- log all changes made with *Storage Connector*
- tracks states for ongoing consumer requests
- log all *access requests*

Relations:

- Persistence Layer

Technologies:

- any modern language/framework capable of parallel computing

Personal Data Storage

Platform: Server, Mobile

Tasks:

- stores the *operator's* personal data

Relations:

- Storage Connector

Technologies:

- non relational database
- depending on host environment

Persistence Layer

Platform: Server

Tasks:

- stores Permission Profiles, History, Tokens, Configurations and other application data
- cache runtime data and information
- holds keys

Relations:

- Operator API
- Permission Manager

Technologies:

- non relational database
- Filesystem

Notification Infrastructure

Platform: Server

Tasks:

- notifies about everything that needs *operator's* approval (e.g. new registrations, new *permission requests*)

Relations:

- Web server

Technologies:

- WebSockets for web UIs via local web server
- mobile device manufacturer's Push Notification server for mobile apps

User Interface

Platform: Desktop, Mobile

Tasks:

- access restricted to *operator* only
- access & permission management
- data management (editor, types & import)
- history and log viewer
- system monitoring

Relations:

- Web server
- Push Notification Service

Technologies:

- HTML, CSS, Javascript
- Java
- Swift, Objective-C

After outlining all different components while keeping the aspect of portability (S.A.02) in mind, it becomes apparent which arrangements make sense and what variations might be possible. As a result, two, more or less, distinct designs are proposed. One is a rather monolithic approach and the other involve more platform types and outlines a certain flexibility.

The main difference between the two compositions is the non-existence of the mobile platform in the monolithic approach (Figure 5.1). Although *monolithic* only refers to the components arrangement on a *server* platform, originally consisting of in a single process that contains all components and is thus is responsible for every task. It is also imaginable that all server components not necessarily have to be placed into one server environment, but being distributed over several virtual machines or containers, so that they can scale and run more independently. This can improve *redundancy* as well.

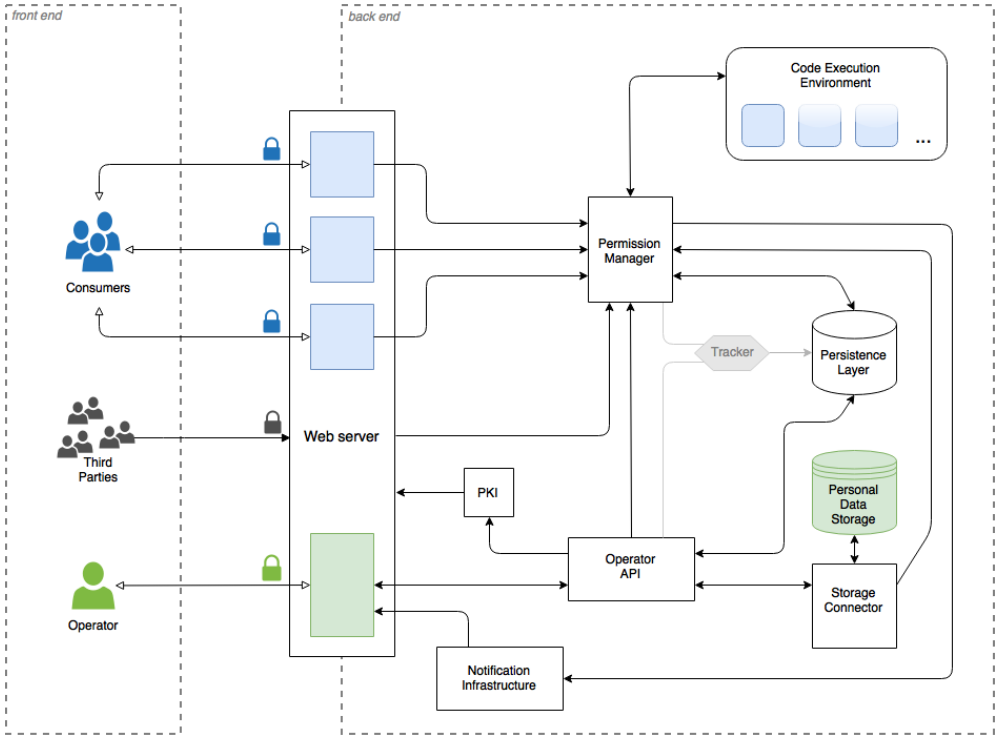

Figure 5.1: PDaaS Architecture, monolithic composition

In theory, a possible version of the arrangement would be to move all components to either the desktop or the mobile platform. This comes along with some downsides and major issues though, that are anything but trivial to solve. Nevertheless, not only to ensure nearly 100% uptime and discovery in a landscape where NAT[3] and dynamic IPs are still common practice, mobile platforms as well as on the desktop, all components but the user interface need to be implemented natively. From a *operator's* perspective that would mean, to have all components at hand and therefore full control over the *PDaaS*. It still would still raise security concerns, though.

Aside from providing the *operator* with a non-stationary and instantly accessible interface to her *PDaaS*, involving a *mobile platform* primarily has the purpose of enabling the *data subject* to carry all her sensitive data along. This is considered a major advantage over the monolithic approach, were all the personal data is located in the *'cloud'*. Depending on the perspective,

[3]Network Address Translation; practice of routing traffic between and through distinct networks address spaces by remapping IPs from those different networks onto each other (e.g. by utilizing ports)

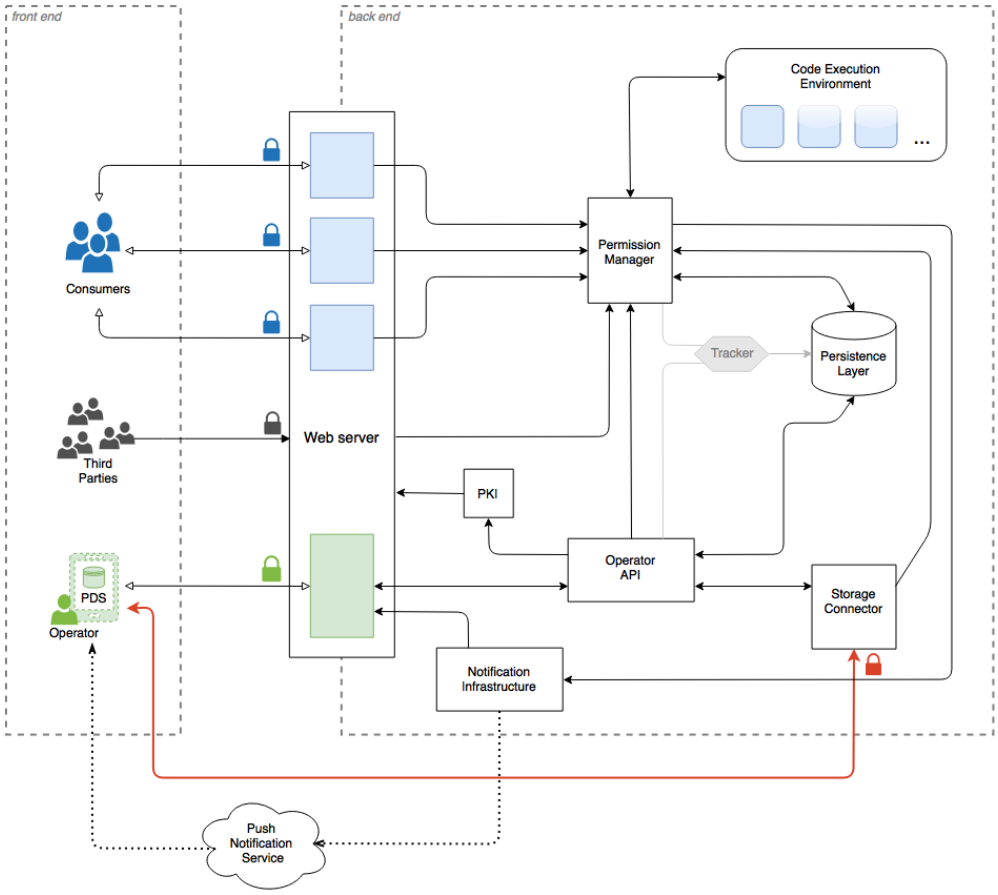

Figure 5.2: PDaaS Architecture, distributed composition

it can either be seen as a *singe source of truth* or a *single point of failure*. Regardless of that, it introduces the demand of a backup or some redundancy concept, which has briefly been touched on in the discussion about database system requirements within the section on *data*. A mobile platform as part of the system makes it more easier for the data subject to establish a security concept, in which the relation between *personal data storage* and the rest of the system is much more liberated, so that all access attempts only happen under full supervision. It is debatable whether to place the *permission profiles* in the *persistence layer* among all other domain-related information, put it into the *personal data storage* too, or define it as an own storage component, in order to be flexible in its placing.

Authenticating *consumers* is performed based on TLS by the web server and its configured subdomains including their individual keys and certificates provided by the *PKI*. The *operator* authentication is either done by the *Operator API* or by the *web server*, depending on the *web server's* capabilities. Though, it makes more sense, to entrust the *web server* with that task, because it's the outmost component and it would prevent unauthorized and potential malicious requests from getting further into the system. And since a native front end on a mobile platform is considered *private* as well, it is reasonable in that case to change the *operator* authentication from JWT-based to TLS-based *two-way authentication*, which would otherwise be inconvenient when using web-based front ends.

If components are placed only on the server and require to communicate between each other, but are separated into independent processes, then some inter-process communication need to be established (e.g. sockets). It is also conceivable that inter-communication between server components could just be unidirectional. Approaches like changing configuration files by writing to the filesystem can therefor be feasible in some cases. Components that can vary in terms of their platform, have to communicate to other components via *HTTPS*.

The architecture implicitly distinguishes between two different groups of endpoints. These endpoints that are made available by the *web server*, which reverse-proxys incoming connections to role-related (operator* or data consumer) components. Starting from that, this separation can be driven further

by simply encapsulating those components into services, that either are related to one of the roles or used by both. This basically results in the *web server* communicating with the two role-grouped services in a bidirectional manner. The group of endpoints for *data consumers* mainly consists of those through witch *access requests* and *permission requests* are coming in and the public one, that is used for when consumers apply for registration. The other one is a small group of endpoints required for all tools the *operator* might need; from data API or notification to authentication and web-based user interface.

Considering the rapid growth of emerging website and applications, which all require user registration, users are getting tired of creating new accounts. Hence they tend to reuse their password(s). Providers started outsourcing that sensitive topic of user management by integrating third party authentication services, which not only makes that feature almost effortless to implement, but also leaves the responsibility as well as the accessibility to those service owners. Whereas users get the benefit of just using one account for all their apps - a universal key so to say, but only one exemplar. So the downside here is, in reality only a handful of third parties [143] provide those authentication services.
OpenID is designed with a very specific type of scenarios in mind, namely the one just described - bringing decentralisation to the market of authentication services - which differs from the ones addressed by the *PDaaS*; at least, when it comes to *data consumer* interactions. Even though, the *PDaaS* has the ability to become the digital representation of it's *operator*. Hence it can and also should be used to authenticate that individual against external parties.

Conclusions: Considering the amount of effort a single-platform composition, namely *desktop* or *mobile*, would take to get fully operational with respect to the specification, it is not only reasonable but also more secure to involve a server platform with proper security measures, static IP and high availability. Even if that server is a local machine connected to the operator's private network. That said, it is sufficient to start with the *monolithic* approach and as suitable mobile applications emerge that are supporting major administration features, notifications and *personal data storage*, it should be possible to migrate effortlessly towards the *distributed* approach

that brings a higher level of confidence, because all the sensitive personal data is not on some computer machine somewhere on the internet, but right in the hands of its owner. By the proposed architecture, all components (or groups of components) are portable and therefore relocatable among the suggested platforms; and with the introduced authentication methods for operators multiple front ends for the same *PDaaS* are thereby supported and can be implemented with almost no effort, and that again covers more use cases. As a supplement, an *identity provider* based on the OpenID standard would fit nicely into the existing arrangement and does not interfere with the other components. However, it is beyond the scope of this work to elaborate on this topic. For now it is stated as a feasible and logical addition to the *PDaaS*.

5.6 Environment and Setup

As stated in the project's core principles, *Open Development* is vital for the project to gain trust. Interestingly, this has a significant impact on how a *PDaaS* might be deployed or installed. All its components can just be grabbed and used as it suits everyone's needs; while respecting their licenses. Furthermore, enforcing *portability* (S.A.02) leads to a more simplified and independent development process that can be organized in a way so that the primary division into components can be leveraged.

The range of environment systems for *server* platforms is highly diverse but the main shares [144] belong to either the UNIX or LINUX family, even though almost every platform is POSIX-compliant.[4] When it comes to *mobile* platforms the market is fare less divers. Native applications are either developed in *Java* (for Google's Android) or in Swift (for Apple's iOS). Whereas the environment systems has nearly no relevance for the *desktop*, other then the screen size and maybe which browser and version the environment system runs. But that's a situation the user can change if necessary.

As a result, being able to use certain components on a *server* platform depends on what *server* environment is available. And vice versa, in order

[4]Portable Operating System Interface; a collection of standards released by the IEEE Computer Society to preserve compatibility between operating systems

to decide what implementation of a component is suitable, it's crucial to know in which environment that component has to run in. Either way, not to forget all the dependencies a component itself might have. Such constraints can be avoided by abstracting the runtime of those components and either embed every required software dependency or provide them in separate runtimes, if that's possible. Depending on the used technologies, this concept is commonly known as *virtualization* or *containerization*. It isolates software by putting them into a so called *container*. But since those container-wrapped components still have to interact with each other, they need to be supervised or at least managed. This is done by an orchestration software, which not only allocates system resources but is also capable of emulating a whole network infrastructure (e.g. DNS, TCP/IP, routing). Thereby, it is utilized to determine how certain container (and its containing component) are allowed to communicate and what resource are accessible from the inside (e.g. filesystem). This complete abstraction to the surrounding environment it effectively means it's the only dependency the *PDaaS* would have, regardless of how its components are implemented. They just have to be *'containerizable'* - satisfy the *container image specification* [116]. This concept can also be utilized for the *supervised code execution* (S.A.01) mentioned before without any restraints.

Migrating from a server-located *personal data storage* to a *mobile* based version introduces another challenge. The subsequent approach is a first and more general solution to that problem.

NOTICE: it is assumed that a running instance of a PDaaS* is in place, the *operator* owns a modern mobile device and on this device a *PDaaS* mobile application is installed.*

1. After starting the app, the operator needs to establish a connection between server and mobile application. Therefor the operator either has to scan a QR-Code with the help of that app. The QR-Code is presented to the operator within the management tool of the *PDaaS* running in a browser. Or the operator inserts her credentials into a form presented by the mobile application.

2. After the connection has established, the operator can trigger a progress that duplicated all her personal data to the device that has

just been associated with the *PDaaS*.

3. At this point, one of two ways can be proceeded with, depending on whether a complete write log for the *personal data* (see discussion about backup strategies does exist or not.

    a) If *[LOG-EXISTS]*, query by query the whole log is obtained from the existing storage and is then again executed in chronologically order by the query language abstraction, starting with the oldest. The only difference here is that the target storage, on which that query is actually performed on, is located on that newly introduced platform

    b) If *[LOG-NOT-EXISTS]*, the situation is more complicated, if the database systems are not based on exactly the same technology. Hence, additional migration software is required. If both database systems provide import and export mechanisms that support at least one interoperable data format, the migration software can leverage this features simply by exporting all the data and saving it to the filesystem. The software then transfers the dump to the target environment and triggers the import process. When this si not the case, the software not only needs to be aware of both database systems and their native query language, it also has to have a comprehensive understanding of how their data structuring concepts work, in order to reliably transform one into the another. So, to be more specific, at first the software has to analyse the structure of the source database. Based on this result it might need to perform some configuration on the target database, before actually obtaining the data from there. Received data then need to be transformed into queries that are supported by the target database system. Those transformed queries are transferred to the target environment, where those incoming queries get executed until all data is migrated.

4. After the duplication process has finished, the operator can decide which PDS the *PDaaS* should use to serve *access requests* and what should happen with the other storage(s).

To conclude, a migration process like moving *personal data* from one platform

instance to another can be much more simplified and robust, if a complete query log would exist. It is also worth mentioning, that the migration process described above is not restricted to exactly this source or migration direction. As long as target and source are either a *server* or a *mobile* platform, any variant is imaginable.

Conclusions: Installing a *PDaaS* should be straightforward with the least possible effort being spend for preparations. Package manager of all popular operating systems should offer (semi-)automated installations. Additionally, components themselves and the project as whole have to provide detailed documentations for various ways of how those parts or the entire system need to be installed. Alternatively, *data subjects* might be willing to entrust external third parties with hosting a *PDaaS* instance for them. In that case the distributed approach involving a *mobile* platform might come in handy, so that the actual data is not stored somewhere beyond their reach. The *PDaaS* as an open source development encourages anybody who is interested or even wants to contribute to checkout the source code of the various implementations, get it to run and play around with it. But therefor at least the components of the *server* platform are required to have documented on what other software they depend on, so that the target environment can be prepared properly. Aside from hardware, on which the *PDaaS* needs to run, the only other requirement is owning an internet domain that is registered on a public DNS[5] server and has no subdomains configured yet.

If a component needs to get segregated from its host environment, *containerization* is the recommended technique, since it causes less overhead compared to *virtualization* and is generally a lightweight approach. Though, additional abstraction might also introduce new problems instead of solving them.

5.7 User Interfaces

Designing graphical user interfaces is beyond the scope of this work and the *PDaaS* specification as well. Nevertheless this section shell be understood as a collection of proposed ideas addressing the questions of what types of user

[5]Domain Name System; decentralized open directory that associates readable (domain) names with IP addresses

interfaces the *PDaaS* should provide and which features they might need to support.

The most notable characteristic used to distinguish user interfaces from each other are those interfaces that are visible and the ones that arn't. For example a *graphical user interface (GUI)*, composed of visually separated areas with a certain semantic and assembled with meaningful objects on which the user can physically act, for example by touching them. The interface then reacts on those actions by changing its appearance. In this way the user can recognize and comprehend her actions. Whereas non-graphical user interfaces don't provide the user with objects or surfaces to interact with. Instead, the primarily used medium is text, regardless if it's human-readable or not. But *command line interfaces (CLI)*, available mainly in command line environments or shells, still provide a certain level of interactivity. A running program can pause in order to prompt the user with an input request. If an input is made and submitted the program then proceeds. The group of interfaces whose interactions can be fully automated, are for example *application programming interfaces (API)*. Depending on the transport technologies, it's no unusual that *API* interactions are consisting of just one action causing one reaction. Non-graphical interfaces enabling interactions on a lower level. Even though they provide more functionality and can be more time efficient, they are more rudemental and often less secure. While *GUIs* are normally meant for end users to interact with applications on a more sophisticated level, *CLIs* are used during development, for automation, or for server environment administration; probably remotely, because they are typically headless. Whereas *APIs*, documented by its provider, enable software developers to program automated requests against those interfaces.

Table 5.3 provides a list of features and associates the different types of user interfaces mentioned before, which indicates if they should be supported by a certain type. It is notable that the *GUI* provides the *operator* with a powerful tool Hence it requires reliable protection mechanisms (see Authentication). Whereas *API* capabilities are very limited, because it's the one interface that the *PDaaS* exposes to third parties.

Table 5.3: Features that should be supported by the given user interfaces

Feature	GUI	CLI	API
manage permission profiles (P.VIU.03)	**X**	-	**X**
view access history (P.VIU.04)	**X**	**X**	-
register consumer	**X**	**X**	-
add new front end	**X**	-	-
authenticate operator	**X**	-	-
migrate personal data	**X**	**X**	-
review permission requests (P.I.04)	**X**	-	-
create & maintain templates (P.I.05)	**X**	-	-
adjust precision of data (P.I.06)	**X**	-	**X**
introduce new data structs	**X**	-	**X**
configure *PDaaS*	**X**	-	-
import personal data	**X**	-	**X**
read/access personal data	**X**	**X**	**X**
manipulate personal data	**X**	**X**	-
run supervised code execution	-	**X**	**X**

The architectural design includes *desktop* and *mobile* platforms. While prioritizing a web-based *GUI*, the management tool for the *operator* also needs to be natively implemented for common mobile systems (P.VIU.02); in this case Android and iOS. This again enables to provide real-time notifications (P.I.03, P.B.02) on mobile platforms, whereas the same feature is added to *desktop* platforms by providing WebSocket-based connections. Since screen sizes can vary - in particular on *mobile* platforms - the *GUI* is required to be highly responsive and has to adapt (P.VIU.01) various screen sizes. Given the capabilities of the management tool, an inaccurate or error-prone rendered *GUI* can quickly cause unintended incidents. Thus the main focus must be to ensure a very robust and low-latency interface rendering.

Known challenges for the *GUI* design are primarily to develop very efficient but also fun to use interfaces for reviewing *consumer registrations* and *permission requests*. Especially the latter can become hard to solve, because how can graph-based and nested data structures be displayed in such a way that makes reviewing and also manipulating an easy task to do - even on a

screen of a mobile phone? One approach could be to utilize the *accordion* pattern [145] for edges and start nesting them in order to represent subsequent data structure. The interaction then might look like tree-structured navigation; moving alongside relations just by expanding and folding in data points.

Since other parts of the system have to provide the mechanisms for increasing or reducing the precision of data due to privacy protection, the challenge here is to find the right design concepts for data subject to facilitate those adjustments. Precision adjustments can be achieved by either changing the sampling rate in a dataset containing a series of data points, or by rounding values to a certain extend. Example are cutting fractional digits of the latitude and longitude values in a position information, or removing all position information obtained between every quarter from a full day tracking period. Whether data subjects can choose from an abstracted precision grading (e.g. *high*, *mid*, and *low*) or they set specific type or unit related filter mechanisms, configurable defaults on a system-wide level should be provided by *GUIs* in any case. Following data types are supposedly vulnerable to compromise privacy, thus proposed to support precision adjustments: *Date* (time), any kind of absolute measurements, sets containing data series, and position information, as mentioned before.

Conclusions: The most important aspect, when interacting with something or someone, is being provided with a feedback. An action typically causes - and is therefore *expected* - a reaction. The result is an interaction, unless no reaction occurred.

The discussion above outlines the relevance of those interactions for the *PDaaS*. Thus, for users and other software to interact with the *PDaaS* interfaces is mandatory. Primary characteristics of those interfaces are complete functionality, security precautions and restrictions, as well as comprehensive documentations. And visual user interfaces in addition, needs to provide reliable and adaptive rendering, a consistent and encouraging interaction design. *GUIs* need to be provided for all *desktop* and *mobile* platforms, primarily to provide an efficient user experience for the operator. The operator is the only role with permissions to access a *GUI*. Components on the *server* platform should provide *CLIs*, at least when no other technical option exist

to interact with them. Also accessing the database from command line could be appreciated at some point. APIs are mostly meant for data consumers to interact with the *PDaaS*, and perhaps for automated data contribution (based on *operator* role; e.g. browser plugin). *Desktop* platforms might use those *APIs* as well. In any case, *APIs* must be separated according to the *roles*.

These are all vital characteristics whose details need to be addressed by the *specification*. Whose implementation details though are not the concern of this specification, as long as every stated requirement is being acknowledged.

6

Specification *(Draft)*

Version 0.1.0

This chapter holds the first draft of what might become an open specification. As for now it has therefore no claim of completeness, continuity or accuracy. Frequent changes are to be expected. The contents is based on and a result of all previous discussions and developed solutions. Hence parts of the contents might reoccurring.

Introduction

This specification describes a system with the purpose of controlling the personal data of a single entity. By 'entity' it is primarily referred to a person, an individual. This individual manages all data relating to her in an online platform, operated by her, which exposes a service making those data selectively accessible to third parties that might have an interest in them. Meanwhile the system tries to ensure that not personal data ever leaves the system. The overall flexibility and portability of the system enables the individual to store her data for example on a mobile phone so that she is

able to carry all her personal data along while the accessibility of the data is ensured at all time.

Notation and Conventions

The requirements notation used in this document originates in the RFC 2119[1] and shall be interpreted as described in there. This applies to the following keywords, recognizable through capital letters: MUST, MUST NOT, REQUIRED, SHALL, SHALL NOT, SHOULD, SHOULD NOT, RECOMMENDED, MAY and OPTIONAL.

Values and words presented in a `fixed-width font` need to be understood as names and must therefore be used as is.

Lists that are numbered outline its given order, which must therefore be followed. Lists that start with a character (upper or lower case) have no claim to its order. Their items are inclusive (logical *and*), while bullet list items are exclusive (logical *or*).

Terminology

(The) System: refers to either the overall concept or the implementation of this specification

Platform: combination of environment capabilities and common role characteristics, which can be inherited form its hardware/device properties; possible values: *desktop*, *server*, *mobile*

Component: independent piece of software that is part of the system and might be capable of running on different platforms, but not at the same time though; serves a unique purpose within the system

Operator: data subject and owner, not necessarily provider or host though; individual, represented by the system and whose personal data can be accessed through the system; also referred to as *data controller*, *data subject* or *data owner*

Third Party: external entity or vendor, who is yet to become registered as

[1] https://tools.ietf.org/html/rfc2119

a *consumer* to the system

(Data) Consumer: external entity or vendor, who has registered to the system and is therefore permitted to request access to data or even to access data hold by the system

(Data) Contributor: external entity, that is authorized by the *operator* to obtain and push personal data of hers into the system

Personal Data: data, relates to the *operator*, that is contained in the system and selectively permitted to get accessed by consumes

Data Broker: entity with commercial interests in collecting, aggregating and analyzing personal data from any possible resource in order to combine and enrich those, to license the result to other entities

Endpoint: dedicated entry for a specific *data consumer* to communicate with the system (e.g. access personal data)

Permission Profile: set of access rules and configurations tied to an *endpoint*, defining e.g. how long or often and what personal data is accessible by a related *data consumer*

Access Type: defines the method of how the access to personal data is provided

6.1 Overview

The overall purpose of this specification is to provide detailed instructions to build a web service that on the one side encourages an individual to manage and maintain all the data relating to her in one place, and on the other side that enables third parties to access such data if they are permitted by the individual to do so, preferably through supervised code execution instead of just handing over the raw data points. The result is a one-to-many relation between the data owner - the individual - and all those who might require some data to for example proceed a purchase initiated by the operator or make a proper decision on her medical treatment.

The system architecture is designed while keeping flexibility and portability in mind but still preserving simplicity and security. The result supports to run some components on different platforms in a distributed way. Therefore the operator is able to operate the system while being on the move, and can

literally carry all her personal data along.

The design of this specification tries to leverage as much existing standards and open technologies as possible for all the different aspects and components. So that by recognizing common practices its implementation and integration can be made as effortless as possible.

Key Features

Perspective: *Operator*

- full control over the flow of personal data
- maintain all personal data in one place
- real-time information and notification
- taking her personal data with her
- collect and analyse her own personal data
- trust through open development and open source
- commercialize data access on her own

Perspective: *Consumers*

- reliable single source of latest data about an individual
- access only those data that are actually required and thus reduce 'noise'
- access data that never were been collectible before
- distributed computing

Scenarios *(Excerpt)*

Online Purchase In order to proceed with the checkout, the shop requires some personal from the user, such as shipping address, email and payment information. Either the shop is already a data consumer, then it would access the system in the background to check if any data has changed, or the shop has to register as a consumer first. After the registration process has been initiated, and if not already has happened the user is then forwarded to complete the checkout. After reviewing the registration and the data that are attempted to get accessed, the use is informed via email that the order has been proceeded and the shipment has started.

Social Network Instead of storing personal data by itself, a social network, after registering and being permitted to, can request and display data about a user, whenever other users try to access them. Created content such as posts, comments, images or videos can be stored in the system as well. The social network then just holds references to all the content, so that it can obtain and forward information on how to request those data, for example with a one-time url.

Apply for a Loan The data that credit institutes take into account when deciding on creditworthiness of an individual can directly be accessed through the system; instead of gathering and acquiring them from all sorts of resources including filled out forms from applicants. The credit institute takes out the part of the computation that is responsible for those calculations and hands it over to the system, after it is permitted by the operator to do so. The system invokes that computation with the required data points and afterwards it just sends the result back to the credit institute.

Browser History A browser plugin, which is connected to the service, keeps track over every called URL. Those data, collected by the operator, could not only be reviewed and analyzed by herself, but also be made available to data consumers.

Movement Profile Instead of letting third party apps keep track of an individual's daily movements (e.g. fitness app), an app that is associated with the system, obtains and pushed those location data directly into it. If a third party is now interested in those data, it can apply and eventually get permitted to access those movement data, but with configured resolution the data subject is comfortable with.

Architectural Overview

The architecture design (see Fig. 6.1) defines three different platforms whereon components of the system might be running. While *desktop* and *mobile* platforms are primarily meant to serve as the front end of the system

and to present the operator with GUIs, the *mobile* platform especially might host the *Personal Data Storage*. The storage of the personal data can be located on every platform. This is enabled by abstracting the storage through the *Storage Connector* on the *server* platform.

That *server* platform also provides an external API for accessing personal data. Incoming requests from third parties and consumers get then processed by the *Permission Manager*, which i.a. decides if and how data can be accessed. Every consumer has its dedicated exposed endpoint consisting of a subdomain (e.g. `CONSUMER_ID.system.tld`). Further components are i.a. a *Code Execution Environment*, a *PKI* that provides and issues certificates and key-pairs to facilitate authenticating at the endpoints. The *Persistence Layer* is represented by potentially multiple technologies, such as databases or filesystem. A *Notification Infrastructure* streamlines the different ways and technologies to notify the operator about certain events (e.g. system receives new registration). Probably one of the most important component is the *Operator API*, through which the operator i.a. is able to configure the system or manage permissions, and the API is granted read/write permissions at the *Storage Connector*.

General Process Description

0. *Prerequisites: the data subject has an up-and-running system; existing third party aims to access personal data on the system*

1. Third party has to register at the system to become a data consumer, therefore it has to provide certain information to the operator either via QR-Code, which the operator can scans, or by submitting those information to a unique URI that is upfront generated and provided by the operator to the third party.

2. Consumer reviews the registration. On a positive outcome a new endpoint gets defined and, depending on the content of the registration request, a permission profile is optionally created and configured by the operator. If the outcome instead is negative, an error message is prepared. In any case, afterwards the third party is informed about the outcome via callback URI, and optionally provided with additional

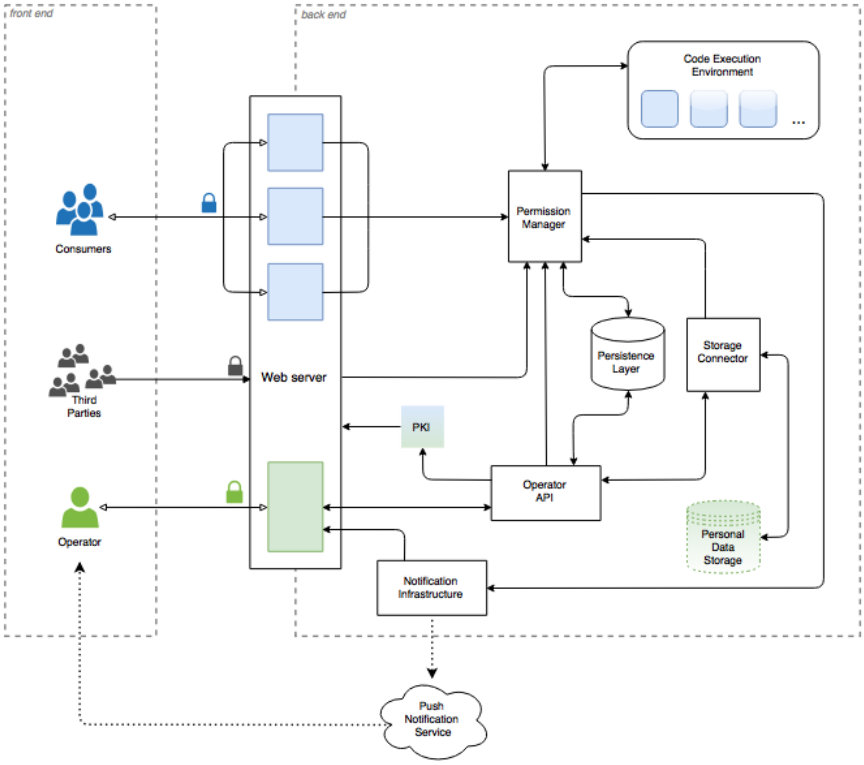

Figure 6.1: System Architecture, simplified

information that are required for further interactions.

3. After the consumer has set up a client according to the documentation and the information he has been provided with, and after he successfully authenticates to the system, he then submits a query to the dedicated endpoint.

4. The incoming query is parsed and validated against all associated *permission profiles*. If the query passes, the request is processed according to configurations and depending on the determined *access type* (e.g. supervised code execution or plain forwarding). The response to the consumer either contains information about when and where the result can be obtained, or the actual result is included directly.

5. Finally, the query result is returned to the consumer, either directly or self-obtained later on.

Relations

consumer:endpoint [1:1] one entity (consumer) relates to one access point (endpoint), exclusively accessible and authenticated by TLS-based *two-way authentication*

endpoint:permission-profile [1:n] zero or more *permission profiles* are associated to one *endpoint* and are therefore responsible and included in the query validation procedure

storage-connector:personal-data-storage [1:n] the *Storage Connector* MUST be able to access at least one or more *Personal Data Storage*, regardless of its location (platform)

Depending Technologies and Standards

a) HTTP (RFC 2616[2], RFC 7540[3])
b) TLS (RFC 5246[4]
c) WebSockets (RFC 6455[5])
d) JSON (ECMA-404[6])
e) JWT (RFC 7519[7], RFC 7515[8], RFC 7516[9])
f) GraphQL (Working Draft – October 2016[10])
g) Open Container Specifications (by Open Container Initiative)
    - image[11]
    - runtime[12]
h) X.509 (RFC 5280[13]
i) ACME (Draft 05[14])

- document-based database systems
- browser-supported technologies (HTML, CSS, JavaScript)
- Swift, Java

Prerequisites

a) DNS-registered Domain
b) publicly reachable server (e.g public IP or dynamic DNS)
c) [OPTIONAL] mobile device

[2]https://tools.ietf.org/html/rfc2616
[3]https://tools.ietf.org/html/rfc7540
[4]https://tools.ietf.org/html/rfc5246
[5]https://tools.ietf.org/html/rfc6455
[6]http://www.ecma-international.org/publications/files/ECMA-ST/ECMA-404.pdf
[7]https://tools.ietf.org/html/rfc7519
[8]https://tools.ietf.org/html/rfc7515
[9]https://tools.ietf.org/html/rfc7516%7D
[10]https://facebook.github.io/graphql/
[11]https://github.com/opencontainers/image-spec/blob/master/spec.md
[12]https://github.com/opencontainers/runtime-spec/blob/master/spec.md
[13]https://tools.ietf.org/html/rfc5280
[14]https://ietf-wg-acme.github.io/acme/

6.2 Components

Web server an interface to (or proxy for) server components

a) process incoming traffic originating by operator, consumer and third parties
b) all other server components are reachable only through this component
c) serve web-based front ends
d) perform all TLS-based authentications
e) do load balancing, if necessary
f) pass through notification for web-based Management Tools that are connected via Web Sockets
g) rule-based traffic management leveraging network characteristics and alike

Permission Manager processes incoming requests from third parties or consumers

a) parse requests and proceed accordingly
b) examine and check queries that are aiming to access personal data
c) queue requests, if further processing is blocked
d) notify operator about new registration attempts
e) create and manage permission profiles
f) read personal data through *Storage Connector*
g) delegate supervised code execution and obtain result

Key Infrastructure (PKI) provides the web server with keys and issues certificates

a) issue certificate based on CSR provided by third party
b) create and self-sign a system key-pair
c) create and sign key-pairs for every endpoint
d) maintain certificates and their validity (recurring renewal) issued by trusted public CAs
e) create new Diffie-Hellman groups

Storage Connector abstracts the *Personal Data Storage* and facilitates read, write, changelog and permissions

a) basic access management
b) connect to the PDS
c) validate query according to the abstracted query language
d) query personal data
e) keep track of all changes
f) provide migration and backup possibilities

Personal Data Storage a database system where the personal data is actually stored

a) can be located on any platform
b) capable of translating the abstract query language into domain-specific query languages required by the supported database systems
c) add, change, remove personal data
d) provide translations for common operations (filter, sort, aggregate)
e) store change history
f) store binary data

Operator API provides all functionalities an operator must be capable of

a) perform all JWT-based authentications
b) read and write personal data through *Storage Connector*
c) configure, maintain and monitor system
d) provide access history
e) manage data structs
f) can control *Permission Manager*
g) administrate restricted access for *data contributors*

Code Execution Environment provides isolated runtime (sandbox) for supervised code execution

a) invoke software provided by data consumers with required data points
b) restrict access to the host environment

c) provision runtimes
d) perform test runs and code review
e) monitor runtime during invocation
f) handover results back to *Permission Manager*
g) archive software in *Persistence Layer*

Tracker logs events and transactions occurring in the system

a) track states of ongoing access requests
b) provide data for monitoring and system analytics
c) log access history
d) pattern recognition and anomaly detection

Persistence Layer combines multiple technologies to represent and hold the current systems state

a) store:
    - system related data
    - component configuration
    - data provided by the *Tracker*
    - permission profiles
    - tokens
b) filesystem access
c) hold keys and certificates
d) cache certain runtime data

Notification Infrastructure a unified facility for server components to distribute notifications to multiple front ends

a) support native mobile notifications through connected Push Notification Service
b) notify operator about pending approval or review

Data Contributor an authorized entity, typically software, that sends personal data into the system

a) MUST be explicitly authorized by the operator
b) data structure and format MUST be known by the system

Management Tool (and other GUIs used by the operator) a graphical user interface, available for mobile and desktop platforms, accessible only by the operator to control the system

a) based on either web technologies or native technologies that are supported by mobile platforms
b) offer all relevant features provided by the Operator API
c) scan third party registrations via QR-Codes or generate URI to submit registration

6.3 Security

The following aspects are required measures in order to improve and ensure the overall sufficient security level of the system. Configurations mentioned below MAY change prospectively due to issues or vulnerabilities emerging in the future.

Transport

All communications from and towards the system as well as internal communication between components located on different platforms MUST be established with *HTTP over TLS*. Thus, external third parties are only allowed to communicate with the system on port `443`, whereas internal communication runs through port `4223`. Port `80` MUST return HTTP Error `403 Forbidden` and all other ports SHOULD be blocked.

The key-pair used in TLS to agree on a mutual symmetric key, MUST be based on either the *RSA* or *DSA* cipher suite, although *RSA* SHOULD be preferred over *DSA*. All created keys MUST have a length of at least `4096` bits. The Ciphers for TLS, that support *Perfect Forward Secrecy* SHOULD be preferred, but ciphers not supported by `TLSv1.2` MUST be avoided. Evey

endpoint and all other dedicated entry points MUST provide their own generated Diffie-Hellman groups with a minimal length of `4096` bits.

For web-based GUIs *TLS session resumption* SHOULD be activated, but for *endpoints* it MUST be deactivated. Web-based GUIs MUST NOT depend on external resources. All involved assets MUST be stored in the system and thus get served by the *web server* as well. This required behaviour is enforced by utilizing the *Content Security Policy (CSP)* in HTTP headers. The *web server* MUST facilitate a web socket connection fFor web-based GUIs. If a browser does not support those natively, a fallback SHALL be provided by the GUI. Furthermore, those GUIs SHOULD be served with HTTP/2.

The subsequent examples show two Nginx configuration for the *web server*, implementing the previous specifications.

Code 01: web server configuration for a web-based GUI (excerpt):

```
server {
    server_name gui.system.tld;

    listen 4223 ssl http2;
    listen [::]:4223 ssl http2 ipv6only=on;

    ssl_trusted_certificate /path/to/public-ca.tld.crt.pem;
    ssl_certificate         /path/to/gui.system.tld.crt.pem;
    ssl_certificate_key     /path/to/gui.system.tld.key.pem;
    ssl_dhparam             /path/to/gui.system.tld.dhp.pem;

    ssl_prefer_server_ciphers on;
    ssl_protocols TLSv1.2;
    ssl_ciphers 'ECDHE-ECDSA-AES256-GCM-SHA384:
        ECDHE-RSA-AES256-GCM-SHA384:ECDHE-ECDSA-CHACHA20-POLY1305:
        ECDHE-RSA-CHACHA20-POLY1305:ECDHE-ECDSA-AES128-GCM-SHA256:
        ECDHE-RSA-AES128-GCM-SHA256:ECDHE-ECDSA-AES256-SHA384:
        ECDHE-RSA-AES256-SHA384:ECDHE-ECDSA-AES128-SHA256:
        ECDHE-RSA-AES128-SHA256';

    ssl_session_cache shared:SSL:10m;
```

```
    ssl_session_timeout 5m;

    add_header Strict-Transport-Security "max-age=15768000" always;
    add_header Content-Security-Policy "default-src 'self'";
}
```

Code 02: web server configuration for a consumer endpoint (excerpt):

```
server {
    server_name CONSUMER_ID.system.tld;

    listen 80;
    listen [::]:80 ipv6only=on;

    return 403;
}

server {
    server_name CONSUMER_ID.system.tld;

    listen 443 ssl http2;
    listen [::]:443 ssl http2 ipv6only=on;

    ssl_trusted_certificate /path/to/system.tld.crt.pem;
    ssl_certificate         /path/to/CONSUMER_ID.system.tld.crt.pem;
    ssl_certificate_key     /path/to/CONSUMER_ID.system.tld.key.pem;
    ssl_dhparam             /path/to/CONSUMER_ID.system.tld.dhp.pem;

    ssl_verify_client on;
    ssl_client_certificate  /path/to/CONSUMER_ID.crt.pem;

    ssl_prefer_server_ciphers on;
    ssl_protocols TLSv1.2;
    ssl_ciphers 'ECDHE-ECDSA-AES256-GCM-SHA384:
        ECDHE-RSA-AES256-GCM-SHA384:ECDHE-ECDSA-CHACHA20-POLY1305:
        ECDHE-RSA-CHACHA20-POLY1305:ECDHE-ECDSA-AES128-GCM-SHA256:
```

```
29          ECDHE-RSA-AES128-GCM-SHA256:ECDHE-ECDSA-AES256-SHA384:
30          ECDHE-RSA-AES256-SHA384:ECDHE-ECDSA-AES128-SHA256:
31          ECDHE-RSA-AES128-SHA256';
32
33      ssl_session_cache off;
34
35      add_header Strict-Transport-Security "max-age=15768000" always;
36  }
```

Authentication

The following two authentication technologies MUST be supported by the system. 2-Factor authentication as an enhancement of the operator authentication procedure is OPTIONAL and can be implemented either by email or if a mobile platform is part of the system.

Transport Layer Security Before the first consumer tries to register on the system, the system MUST generate a key-pair and sign it by itself. With the resulting certificate the system becomes a private Certificate Authority primarily responsible for signing certificates that are required for every endpoint, and maybe even for connections between mobile and server platforms. The key-pair for a specific endpoint is then used to issue a certificate based on the *Certificate Signing Request (CSR)*, supplied by the consumer, who is associated to that very endpoint. The certified certificate and the endpoint's certificate MUST then be transferred to the consumer on a secure channel, which the consumer is responsible to provide (e.g. HTTP over TLS certified by trusted public CA, or by a self-signed certificate provided with the registration).

In order to use TLS for bidirectional authentication, not only the client (consumer) MUST be able to verify the server's (system) certificate, also the server MUST do the same for the client. This procedure is known as *two-way authentication*, which is part of the TLS connection establishing. If the connection failed to establish the authentications has failed, and vice versa the consumer has successfully authenticated to the system.

JSON Web Token (JWT) When using JWT, `{"alg":"none"}` as header MUST NOT be used. A JWT MAY be encrypted (JWE). If the operator fails to connect to the system before the token's expiration date has been exceeded, the operator is REQUIRED to login again. The token MUST be renewed, but at least after half of the period of validity has been reached. JWTs MUST be created and verified by the *Operator API*. The secrets and keys for that purpose are stored in the *Persistence Layer*. Signature validation MAY get performed by the *Web server*.

The following claims are REQUIRED:

- `"iss"` (Issuer) - domain from which the front end component obtains its data
- `"sub"` (Subject) - front end platform name; MUST be system-wide unique
- `"aud"` (Audience) - `"operator"` or `"contributor"`
- `"exp"` (Expiration Time)
- `"iat"` (Issued At)
- `"jti"` (JWT ID)

One of the following algorithms is REQUIRED (for the `"alg"` header):

- `"HS512"` (key length: 512 bits)
- `"RSA1_5"` (recommended key length: 2048 bits)
- `"RSA-OAEP-256"` (recommended key length: 2048 bits)
- `"A256KW"` (recommended key length: 2048 bits)

6.3.1 System Architecture

- distributed approach: location of PDS
- Containerization

6.3.2 Supervised Code Execution

- no shared filesystem, no network

6.3.3 System Monitoring

- pattern recognition & anomaly detection for access requests (e.g. check if request come constantly from the same IP)
  - does it matter from what origin the data request was made? how to check that? is the requester's server domain in the http header? eventually there is no way to check that, so me might need to go with request logging and trying to detect abnormal behaviour/occurrence with a learning artificial intelligence
- spam protection

6.4 Protocols

The following protocols reflect core procedures. They have to be understood as detailed instructions on how these procedures have to be implemented.

Consumer Registration

Description: TODO

- initial consumer registration would be done on a common and valid https:443 CA-certified connection. after transferring their cert to them as a response, all subsequent calls need to go to their own endpoint, defined as subdomains like `consumer-name.owners-notification-server.tld`

0. [OPTIONAL] *data subject* provides URI to *data consumers*
1. *data consumers* create *permission request* that includes
   - X.509 based CSR[15]
   - callback URI via HTTPS as feedback channel
   - [OPTIONAL] information about what data points wanted to be accessed

[15] Certificate signing request

2. depending on 0), *data consumer* provides *operator* with priorly created *permission request* either as QR-Code or via HTTPS by given URI

3. *operator* reviews request and decides to either refuse or grant assess; the latter results in:

   a) creating new *endpoint*
      - create new unique subdomain and a related asymmetric key pair signed by the system's root CA (self-signed)
      - issue *consumer* certificate based on it's CRS and sign it with the key pair related to this *endpoint*

   b) if information is provided, creating new *permission profile* by either applying existing draft/template or configuring *permission type* (incl. expiration date if required) and permitted data endpoints; associate to specific *endpoint*

4. *data consumers* gets informed about the decision via callback channel

   - on grant, response includes
     - *consumer's* certified certificate
     - certificate that's associated with the created endpoint
     - information on what data points are allowed to be accessed;
   - on refusal: error code/message

5. *data consumer* handles the response appropriately

   - [OPTIONAL] pin the provided *PDaaS* certificate

Permission Request

Description: applying for permission to get access to certain data

Access Request (accessing data)

Description: TODO

0. after successfully authenticated, *consumer* sends *access request*

1. request contains at least the *data query*; based on that query and the

permission profiles, access is tried to get verified

   a) on success, response gets computed
   b) on failure, error code/message is responded; process aborts
      - if the error was raised because no appropriate *permission profile* was found, then the *consumer* first needs to request permission for the *data points* that were part of the query

2. [OPTIONAL] depending on whether the `keepalive` flag was set `true`, the connection of this requests lasts until response computation has finished or timeout has reached, otherwise the response contains a URI unique to this current request including an estimation when response will be available under that URI; connection can still timeout, which is defined by the system

3. depending on the type of that *access request*,

   (A) the data get queried and the result is added to the response
   (B) based in further information provided by the request, the environment for the *supervised code execution* is getting provisioned, the program from the *consumer* will be ran against various tests
      a) in fail, error code/message get added to the response
      b) on pass, computed result gets added to the response

4. response is finalized and gets returned back to the *consumer*, either as a response to this request or provided under the unique URI as of 2)

Permission Validation

Description: detailed description of the algorithm that checks permission profiles according to an access request; including all different possible cases (multiple profiles for one consumer etc)

Adding or Changing Personal Data

Description: TODO

6.5 Data & Types

- abstraction concept (GraphQL)
- areas of applications (excerpt)
  - profile data
  - sensor data (e.g. geo-location, motions)
  - financial record (and history)
  - payment information
  - medical record
  - governmental data
  - biometric datasets (e.g. finger print, retina)
  - web search history
  - what about user-created content, like pictures, videos or blog posts?
- primitives and structs
- minimum feasible types
- structs as add-ons
- data models (PL)
  - registration
  - endpoint (relation to the key and file)
  - permission profile
    * permission type
    * max/min interval (how often)
    * block all on this endpoint until further notice

6.5.1 Structure & Types

- henceforth only two things: primitive and struct
- supported date types

6.5.2 Read

- permission profiles
  - type

  - how often
  - what data

6.5.3 Write

(!) every data or configuration change has to be reversible

precision of data: demanding lower precision than the *data subject* has approved is always possible. The other ways around not.

6.6 Interfaces

- graphical user interfaces
  - management tool: web and mobile
  - platform types
  - feature list
- registration
- Operator APIs
- for data consumers
  - permission requests
  - access request

Registration Request

- contains certificate signing request
- [OPTIONAL] contains *permission request*

```
{
    "callbackUri": "TODO",
    "csr": "TODO",
    "dataPoints": [
        "profile.lastname"
    ]
}
```

Permission Request

- creates new *permission profile*
- `https://example-consumer.pdaas.tld/pr`

```
{
    "callbackUri": "",
    "dataPoints": [
        "profile.lastname"
    ]
}
```

Access Request

- obtains actual data
- if `keepalive` is set `true`, the connections lasts until response computation has finished, otherwise the response contains a URI unique to this current request including an estimation when response will be available under that URI; connection can still timeout, which is defined by the system
- `https://example-consumer.pdaas.tld/ar`

```
{
    "query": "TODO"
}
```

Requirements:

- query has to match exactly one corresponding *permission profile*

6.7 Recommendations

Host Environment(s)

- Setup/Hardware

Software Recommendations

- letsencrypt

- nginx
- systemd
- rkt
- Kubernetes
- LibreSSL

Resources

- SSL and TLS Deployment Best Practices[16]
- Security/Server Side TLS[17]
- Strong SSL Security on nginx[18]

[16]https://github.com/ssllabs/research/wiki/SSL-and-TLS-Deployment-Best-Practices
[17]https://wiki.mozilla.org/Security/Server_Side_TLS
[18]https://raymii.org/s/tutorials/Strong_SSL_Security_On_nginx.html

7

Conclusion

7.1 Ethical & Social Impact (TODO: or "Relevance")

- Regarding involving an official party to verify data reliability: The actual question would be, is the *data subject* certain, that she really wants to hand over those capabilities to official authorities? Depending on which *data consumers*, what task their are entrusted with and what motivation the *data subject* has has in mind to do so, the *PDaaS* might become a powerful *'digital reflection'* and starts to get seen as a real and reliable representation of herself. Then the decisions made by *data consumers* might have a big impact for the *data subject's* life. For example a housing loan won't be granted or a medical treatment has been refused.
- give back the data subject to control the level of privacy she is willing to share
- (formally placed in 230) At the end it all comes down to understanding the human being and why she behaves as she does. The challenge is not only to compute certain motives but rather concluding to the right

ones. When analyzing computed results with the corresponding data models and trying to conclude, it is important to keep in mind, that correlation is by far no proof of causation.

7.2 Business Models & Monetisation

- possible resulting direct or indirect business models
- data subject might want to sell her data, only under her conditions. therefor some kind of infrastructure and process is required (such as payment transfer, data anonymization, market place to offer data)

7.3 Target group perspectives

- User: would I use this stuff? The underpinning technical details and how it works is not my concern and non of my interests. I want this stuff work and being reliable. it its simple to use. and maybe even easy to setup (server n stuff), then the hell, I would!
- Dev:
  - spec implementer
  - integrater in consumer:
- Consumer:
  - what can she do with it: adjsut precision of datasets and values to increase privacy
  - control and get an overview on where her data might flow (and for what purpose)

TODO: make a reference or involve the research mentioned at the beginning

7.4 Challenges

- adoption rate of such technology
- data reliability from the perspective of a *data consumer* Since it is

almost impossible to ensure complete reliability of all the data a *PDaaS* has stored or might me offering, and because it is operated by exactly that individual, and that individual only, all data in question is relates to and is thereby owned by her, it, of cause, makes it not easy for *data consumers* to trust *PDaaS*s as resources for their business processes, but I am certain, that the demand for all different kinds of data exceeds the partial uncertainty of their reliability.

- personal data leaking Preventing personal data from being leaked to the outside, is, especially because of the system's purpose, extremely hard to prevent, if not possible at all. Just by querying data from the storage or by physically transferring them from one location to another, it's already copied. It's the very nature of digital information technology/systems. So this cannot be defeated. It only can be impeded. Interestingly though, is the same approach the media industry for centuries is trying to make copyright infringements more difficult.
- scenario where the mobile device, or in general the data storage get lost. first of all, not much of a problem, because either device backup or since the liberal relation, the system would continue to function, but limited, until a data storage gets part of the system again (TODO: touched on in the data section at the end)
- during concept development, it appears to become necessary to define another role, for *data contributors* (plugins/clients that are authorized by the *operator* but only allowed to push data to the *PDaaS*).
- when *structs* currently in use get changes all data have to migrate accordingly and fully automated

7.5 Solutions

- even though *OAuth* don't find it's way into this project, working through the standard inspired here and there a solution, for example using a URI as a feedback channel or TODO.
- refer to the scenarios at the beginning by saying that with the *PDaaS*

one is able to implement all of them

7.6 Attack Scenarios

- single point of failure (data-wise),
  - but considering what data users already put into their social networks (or: thE social network: fb), they/it has already become a de facto data silo and is thus a single point of failure. If that service breaks or get down, the data from all users might be lost or worse (stolen). The aspect of data decentralisation achieved by individual data stores can be valued as positive.
- what about token stealing when using jwt?
- future work: add/activate an intrusion detection system

7.7 Future Work

- maybe enable the tool to play the role of an own OpenID provider?
- going one step further and train machine (predictor) by our self with our own data (https://www.technologyreview.com/s/514356/stephen-wolfram-on-personal-analytics/)
- finalize first draft of the spec with all core aspect included and outlined
- developing based on that a first prototype to find flaws in the spec. iterate/repeat
- release 1.0 (spec and example implementation)
- touch on parts that were left blank
- first supporting platforms
- full encryption of the *data storage*

7.8 Summary

- main focus
- unique features
- technology stack & standards
- resources
- the tool might be not a bulletproof vest, but

The work will be continued.

Source Code

Code 01: Example query in SPARQL:

```
# query 1: obtain the first and last name fof data subject
PREFIX person: <http://pdaas.tld/schemas/person>

SELECT $firstname $lastname
FROM <https://unique-consumer-endpoint.pdaas.tld/sparql/profile>
WHERE {
    $person person:firstname $firstname .
    $person person:lastname $lastname .
}


# query 2: obtain all bank accounts that are available for
# online payment
PREFIX bank-account: <http://pdaas.tld/schemas/bank-account>

SELECT $accountId $bankName $paymentMethod
FROM <https://unique-consumer-endpoint.pdaas.tld/sparql/finance>
WHERE {
    $bank-account bank-account:payment-method "online-service" .
    $bank-account bank-account:payment-method $paymentMethod .
    $bank-account bank-account:account-id $accountId .
    $bank-account bank-account:bank-name $bankName .
}
```

Code 02: Results of Code 01 in JSON:

```
// result 1:
{
    "head": {
        "vars": [
            "firstname",
            "lastname"
        ]
    },
    "results": {
        "bindings": [
            {
                "firstname": {
                    "type": "literal",
                    "value": "Doe"
                },
                "lastname": {
                    "type": "literal",
                    "value": "Jane"
                }
            }
        ]
    }
}

// result 2:
{
    "head": {
        "vars": [
            "accountId",
            "bankName",
            "paymentMethod"
        ]
    },
    "results": {
        "bindings": [
            {
                "accountId": {
```

```
                    "type": "integer",
                    "value": 0905553715
                },
                "bankName": {
                    "type": "literal",
                    "value": "A. W. Fritter Institute"
                },
                "paymentMethod": {
                    "type": "literal",
                    "value": "online-service"
                }
            }
        ]
    }
}
```

Code 03: Example query in GraphQL:

```
# URL: https://unique-consumer-endpoint.pdaas.tld/graphql

query {
    profile {
        firstname
        lastname
    }
    bankAccounts(paymentMethod: 'online-service') {
        accountId
        bankName
        paymentMethod
    }
}
```

Code 04: Result of Code 03 in JSON:

```
{
    "profile": {
        "firstname": "Jane",
        "lastname": "Doe"
    },
    "bankAccounts": [
        {
            "accountId": 0905553715,
            "bankName": "A. W. Fritter Institute",
            "paymentMethod": "online-service"
        }
    ]
}
```

NOTICE: schema notation is based on the rules underpinning the schema definition provided by the SimpleSchema project [146].

Code 05: Struct - Profile (example)

```
{
    firstname: String,
    lastname: String,
    pseudonym: String,
    birth: Date,
    gender: String,
    religion: String,
    motherTongue: Language
    photo: File,
    residence: Address,
    employer: Organisation
}
```

Code 06: Struct - Contact (example)

```
{
    label: String,
    type: String('phone'|'email'|'url'|'name-of-social-network'),
    prio: Integer(0-2),
    uid: String
}
```

Code 07: Struct - Position (example)

```
{
    lat: Float,
    lon: Float,
    radius: {
        value: Float,
        unit: Distance
    },
    description: String
    ts: Date
}
```

References

[1] *Big data privacy international.* URL https://www.privacyinternational.org/node/8. - retrieved 2016-11-15

[2] Pedreshi, Dino ; Ruggieri, Salvatore ; Turini, Franco: Discrimination-aware data mining. In: *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining* : ACM, 2008, pp. 560–568

[3] Spiekermann, Sarah: *Ethical IT Innovation: A Value-Based System Design Approach* : CRC Press; Taylor & Francis Group, LLC, 2015 – scale — ISBN 978-1-4822-2635-5

[4] Friedman, Batya ; Nissenbaum, Helen: Bias in computer systems. In: *ACM Transactions on Information Systems (TOIS)* vol. 14 (1996), Nr. 3, pp. 330–347

[5] *Cognitive bias.* URL https://en.wikipedia.org/w/index.php?title=Cognitive_bias&oldid=742803386. - retrieved 2016-11-08. — Wikipedia. — Page Version ID: 742803386

[6] Lemov, Rebecca: *Why big data is actually small, personal and very human. Aeon essays.* URL https://aeon.co/essays/why-big-data-is-actually-small-personal-and-very-human. - retrieved 2016-11-17

[7] Dewes, Andreas: *C3TV - Say hi to your new boss: How algorithms might soon control our lives.* URL https://media.ccc.de/v/32c3-7482-say_hi_to_your_new_boss_how_algorithms_might_soon_control_our_

lives#video&t=1538. - retrieved 2016-11-03

[8] Hong, Jason I.; Landay, James A.: An architecture for privacy-sensitive ubiquitous computing. In: *Proceedings of the 2nd international conference on mobile systems, applications, and services* : ACM, 2004, pp. 177–189

[9] *ProjectVRM - about. ProjectVRM*. URL https://blogs.harvard.edu/vrm/about/. - retrieved 2016-11-09

[10] Tom Kirkham; Sandra Winfield; Serge Ravet; Kellomaki, Sampo: The personal data store approach to personal data security. In: *IEEE Security & Privacy* vol. 11. Los Alamitos, CA, USA, IEEE Computer Society (2013), Nr. 5, pp. 12–19

[11] Poikola, Antti; Kuikkaniemi, Kai; Honko, Harri: MyData – a nordic model for human-centered personal data management and processing, Ministry of Transport; Communications (2015), pp. 1–12 — ISBN 978-952-243-455-5

[12] *Meeco how it works*. URL https://meeco.me/how-it-works.html. - retrieved 2016-11-09

[13] *Open specification of the concept called personal data as a service (pdaas). GitHub*. URL https://github.com/lucendio/pdaas_spec. - retrieved 2016-11-11

[14] USA, Federal Trade Commission: *Data brokers*, 2014 – scale

[15] Rose, John; Rehse, Olaf; Röber, Björn: The value of our digital identity. In: *Boston Cons. Gr* (2012)

[16] Regulation (EU) 2016/679 — General data protection regulation, 2016 – scale

[17] Wikipedia: *Information privacy law*. URL https://en.wikipedia.org/wiki/Information_privacy_law#United_States. - retrieved 2016-11-20. — Page Version ID: 749338152

[18] Loeb), Ieuan Jolly (Loeb &: *PLC - data protection in the united states: Overview*. URL http://us.practicallaw.com/6-502-0467. - retrieved

2016-11-20

[19] Wilhelm, Alex: *White house drops "consumer privacy bill of rights act" draft. TechCrunch.* URL http://social.techcrunch.com/2015/02/27/white-house-drops-consumer-privacy-bill-of-rights-act-draft/. - retrieved 2016-11-20

[20] Consumer privacy bill of rights act (cpbora) — Administration discussion draft: Consumer privacy bill of rights act of 2015, 2015 – scale

[21] FCC 16-148 — Report and order, 2016 – scale. — In the Matter of Protecting the Privacy of Customers of Broadband and Other Telecommunications Services

[22] FCC 16-39 — Notice of proposed rulemaking, 2016 – scale. — In the Matter of Protecting the Privacy of Customers of Broadband and Other Telecommunications Services

[23] *Privacy policies are mandatory by law.* URL https://termsfeed.com/blog/privacy-policy-mandatory-law/. - retrieved 2016-11-20. — Disclaimer: Legal information is not legal advice

[24] Facebook: *facebook - creating an account.* URL https://www.facebook.com/. - retrieved 2016-11-20

[25] *International privacy standards.* URL https://www.eff.org/issues/international-privacy-standards. - retrieved 2016-11-20

[26] Jan Philipp Albrecht, MdEP: *EU-US "privacy shield" - background and frequently asked questions (faq).* URL https://www.janalbrecht.eu/themen/datenschutz-digitalisierung-netzpolitik/eu-us-privacy-shield-2.html. - retrieved 2017-02-06

[27] Dachwitz, Ingo: *Nationale datenschutzbehörden kritisieren privacy shield und kündigen umfassende prüfung an.* URL https://netzpolitik.org/2016/nationale-datenschutzbehoerden-kritisieren-privacy-shield-und-kuendigen-umfassende-pruefung-an/. - retrieved 2017-02-06

[28] Rosner, Gilad: Who owns your data? In: : ACM Press, 2014

— ISBN 978-1-4503-3047-3, pp. 623–628

[29] Grunebaum, J.O.: *Private ownership*, *Problems of philosophy* : Routledge & Kegan Paul, 1987 – scale — ISBN 9780710207067

[30] Regulation (EU) 2016/679 — General data protection regulation, 2016 – scale

[31] FCC 16-148 — Report and order, 2016 – scale. — In the Matter of Protecting the Privacy of Customers of Broadband and Other Telecommunications Services

[32] Facebook: *Facebooks's terms of service. Statement of rights and responsibilities.* URL https://www.facebook.com/legal/terms. - retrieved 2016-12-01

[33] Twitter: *Twitters's terms of service. Twitter terms of service.* URL https://twitter.com/tos#intlTerms. - retrieved 2016-12-01

[34] Google: *Google's terms of service. Google terms of service.* URL https://www.google.com/intl/en/policies/terms/regional.html. - retrieved 2016-12-01

[35] Apple: *Apple's iClound terms and conditions. V. content and your conduct.* URL https://www.apple.com/legal/internet-services/icloud/en/terms.html. - retrieved 2016-12-01

[36] *Why metadata matters.* URL https://www.eff.org/deeplinks/2013/06/why-metadata-matters. - retrieved 2016-11-24

[37] Stevens, John P.: *Why you need metadata for big data success.* URL http://www.datasciencecentral.com/profiles/blogs/why-you-need-metadata-for-big-data-success. - retrieved 2016-11-24

[38] *Big data n.* URL http://www.oed.com/view/Entry/18833#eid301162177. - retrieved 2016-11-11

[39] Wikipedia: *Big data.* URL https://en.wikipedia.org/w/index.php?title=Big_data&oldid=748964100. - retrieved 2016-11-11. — Page Version ID: 748964100

[40] Cios, Krzysztof J. ; Swiniarski, Roman W. ; Pedrycz, Witold ;

Kurgan, Lukasz A.: The knowledge discovery process. In: *Data mining* : Springer, 2007, pp. 9–24

[41] Marbán, Óscar ; Mariscal, Gonzalo ; Segovia, Javier: A data mining & knowledge discovery process model. In: *Data mining and knowledge discovery in real life applications*. Madrid, Span : InTech, 2009

[42] Dewey, Caitlin ; Dewey, Caitlin: 98 personal data points that facebook uses to target ads to you. In: *The Washington Post* (2016)

[43] Taie, Mohammed Zuhair Al: *Big data: Types of data used in analytics. Agroknow blog*. URL http://blog.agroknow.com/?p=4690. - retrieved 2017-02-08

[44] Zaïane, Osmar R: *Principles of knowledge discovery in databases*, 1999 – scale

[45] *Big data collection collides with privacy concerns, analysts say. PCWorld*. URL http://www.pcworld.com/article/2027789/big-data-collection-collides-with-privacy-concerns-analysts-say.html. - retrieved 2016-11-15

[46] *Answers.io. Answers*. URL https://answers.io/answers. - retrieved 2016-11-14

[47] Burgelman, Author: Luc ; Burgelman, NGDATA Luc ; NGDATA: *Attention, big data enthusiasts: Here's what you shouldn't ignore. WIRED*. URL https://www.wired.com/insights/2013/02/attention-big-data-enthusiasts-heres-what-you-shouldnt-ignore/. - retrieved 2016-11-15. — Partner Content

[48] Laney, Douglas: *3D data management: Controlling data volume, velocity, and variety* : META Group, 2001 – scale

[49] Hilbert, Martin: Big data for development: A review of promises and challenges. In: *Development Policy Review* vol. 34 (2015), Nr. 1, pp. 135–174

[50] Davis Kho, Nancy: *The state of big data*. URL http://www.econtentmag.com/Articles/Editorial/Feature/The-State-of-Big-Data-108666.htm. - retrieved 2016-11-18

[51] CEO), Tim Cook (Apple's: *A message to our customers. Customer*

letter. URL http://www.apple.com/customer-letter/. - retrieved 2016-11-18

[52] Green, Matthew: *What is differential privacy? A few thoughts on cryptographic engineering*. URL https://blog.cryptographyengineering.com/2016/06/15/what-is-differential-privacy/. - retrieved 2016-11-18

[53] Budington, Bill: *WhatsApp rolls out end-to-end encryption to its over one billion users*. URL https://www.eff.org/deeplinks/2016/04/whatsapp-rolls-out-end-end-encryption-its-1bn-users. - retrieved 2016-11-18

[54] *The next rembrandt: Blurring the lines between art, technology and emotion. Microsoft news centre europe*. URL https://news.microsoft.com/europe/features/the-next-rembrandt-blurring-the-lines-between-art-technology-and-emotion-2/. - retrieved 2017-02-08

[55] *Stuck in traffic? Insights from googlers into our products, technology, and the google culture*. URL https://googleblog.blogspot.com/2007/02/stuck-in-traffic.html. - retrieved 2016-11-18

[56] Wikipedia: *Google traffic*. URL https://en.wikipedia.org/w/index.php?title=Google_Traffic&oldid=746200591. - retrieved 2016-11-18. — Page Version ID: 746200591

[57] *Global mobile OS market share*. URL https://www.statista.com/statistics/266136/global-market-share-held-by-smartphone-operating-systems/. - retrieved 2016-11-18

[58] Ao, Ji ; Zhang, Peng ; Cao, Yanan: Estimating the Locations of Emergency Events from Twitter Streams. In: *Procedia Computer Science* vol. 31 (2014), pp. 731–739

[59] Shi, Weiwei ; Zhu, Yongxin ; Zhang, Jinkui ; Tao, Xiang ; Sheng, Gehao ; Lian, Yong ; Wang, Guoxing ; Chen, Yufeng: Improving power grid monitoring data quality: An efficient machine learning framework for missing data prediction. In: : IEEE, 2015 — ISBN 978-1-4799-8937-9, pp. 417–422

[60] Palem, Gopalakrishna: The Practice of Predictive Analytics in Healthcare. In: *ResearchGate* (2013)

[61] Burger, Nicholas ; Ghosh-Dastidar, Bonnie ; Grant, Audra ;

Joseph, George ; Ruder, Teague ; Tchakeva, Olesya ; Wodon, Quentin: Data Collection for the Study on Climate Change and Migration in the MENA Region (2014)

[62] Gaitho, Maryanne: *Applications of big data in 10 industry verticals*. URL https://cfs22.simplicdn.net/ice9/free_resources_article_thumb/Applications_of_big_data_infographic.png. - retrieved 2016-11-19

[63] USA, Federal Trade Commission: *Personal data ecosystem*. URL https://www.ftc.gov/sites/default/files/documents/public_events/exploring-privacy-roundtable-series/personaldataecosystem.pdf. - retrieved 2016-11-17. — Protecting Consumer Privacy in an Era of Rapid Change - Recommendations for Business and Policymakers - FTC Report

[64] Christl, Wolfie: *Corporate surveillance, digital tracking, big data & privacy*, 2016 – scale

[65] Moore, Gordon E.: Cramming more components onto integrated circuits. In: *Electronics* vol. 38 (1965), p. 4

[66] Pritlove, Tim ; Schöneberg, Ulf: *Neuronale netze*, 2015 – scale

[67] Columbus, Louis: *51% of enterprises intend to invest more in big data*. URL http://www.forbes.com/sites/louiscolumbus/2016/05/22/51-of-enterprises-intend-to-invest-more-in-big-data/. - retrieved 2016-12-07

[68] *ProjectVRM - cDevelopment work. ProjectVRM*. URL https://cyber.harvard.edu/projectvrm/VRM_Development_Work. - retrieved 2016-12-09

[69] *ProjectVRM - principles. ProjectVRM*. URL https://cyber.harvard.edu/projectvrm/Main_Page#VRM_Principles. - retrieved 2016-12-09

[70] *TAS3 - project overview*. URL http://vds1628.sivit.org/tas3/. - retrieved 2017-02-10

[71] The TAS3 Consortium: *TAS3 architecture - figure 2.2: Major components of organization domain.*, 2011 – scale. — v 2.24

[72] *Kantara initiative – join. innovate. trust.* URL https://

kantarainitiative.org/. - retrieved 2016-12-14

[73] Kirkham, Tom ; Winfield, Sandra ; Ravet, Serge ; Kellomaki, Sampo: The personal data store approach to personal data security. In: *IEEE Security & Privacy* vol. 11 (2013), Nr. 5, pp. 12–19

[74] Montjoye, Yves-Alexandre de ; Wang, Samuel S. ; Pentland, Alex ; Anh, Dinh Tien Tuan ; Datta, Anwitaman ; others: On the trusted use of large-scale personal data. In: *IEEE Data Eng. Bull.* vol. 35 (2012), Nr. 4, pp. 5–8

[75] *openPDS/SafeAnswers - the privacy-preserving personal data store.* URL http://openpds.media.mit.edu/. - retrieved 2016-12-14

[76] Montjoye, Yves-Alexandre de ; Shmueli, Erez ; Wang, Samuel S. ; Pentland, Alex Sandy: openPDS: Protecting the privacy of metadata through SafeAnswers. In: Preis, T. (ed.) *PLoS ONE* vol. 9 (2014), Nr. 7, p. e98790

[77] *Microsoft HealthVault. Overview.* URL https://www.healthvault.com/de/en/overview. - retrieved 2016-12-14

[78] *How it works meeco.* URL https://meeco.me/how-it-works.html. - retrieved 2016-12-14

[79] Page, Mike: Online adver>sing – booming or broken?, 2015 – scale

[80] *The principles. Industrial data space e.V.* URL http://www.industrialdataspace.org/en/the-principles/. - retrieved 2016-12-14

[81] Prof. Dr.-Ing. Otto, Boris ; Prof. Dr. Auer, Sören ; Cirullies, Jan ; Prof. Dr. Jürjens, Jan ; Menz, Nadja ; Schon, Jochen ; Dr. Wenzel, Sven: Industrial data space - digital sovereignity over data.

[82] Leach, Paul J. ; Berners-Lee, Tim ; Mogul, Jeffrey C. ; Masinter, Larry ; Fielding, Roy T. ; Gettys, James: *Hypertext transfer protocol – HTTP/1.1.* URL https://tools.ietf.org/html/rfc2616. - retrieved 2016-12-17

[83] Belshe, Mike ; Thomson, Martin ; Peon, Roberto: *Hypertext transfer protocol version 2 (HTTP/2).* URL https://tools.ietf.org/html/

rfc7540. - retrieved 2016-12-17

[84] Fette, Ian ; Melnikov, A.: *The WebSocket protocol*. URL https://tools.ietf.org/html/rfc6455. - retrieved 2016-12-17

[85] Crockford, Douglas: The JSON data interchange format.

[86] Bray, T.: *The JavaScript object notation (JSON) data interchange format*. URL https://tools.ietf.org/html/rfc7159. - retrieved 2016-12-17

[87] Bradley, John: *The problem with OAuth for authentication*. URL http://www.thread-safe.com/2012/01/problem-with-oauth-for-authentication.html. - retrieved 2016-12-17

[88] *OAuth core 1.0a*. URL https://oauth.net/core/1.0a/. - retrieved 2016-12-18

[89] Hardt, Dick: *The OAuth 2.0 authorization framework*. URL https://tools.ietf.org/html/rfc6749. - retrieved 2016-12-18

[90] WG, IETF OAuth: *OAuth 2.0*. URL https://oauth.net/2/. - retrieved 2016-12-16

[91] *Specifications & developer information OpenID*. URL https://openid.net/developers/specs/. - retrieved 2017-02-10

[92] *OpenID connect core 1.0 incorporating errata set 1*. URL https://openid.net/specs/openid-connect-core-1_0.html. - retrieved 2016-12-17

[93] *OpenID - openid vs. pseudo-authentication using oauth*. URL https://en.wikipedia.org/w/index.php?title=OpenID&oldid=763047614#OpenID_vs._pseudo-authentication_using_OAuth. - retrieved 2017-02-10. — Page Version ID: 763047614

[94] Bradley, John ; Sakimura, Nat ; Jones, Michael: *JSON web token (JWT)*. URL https://tools.ietf.org/html/rfc7519. - retrieved 2016-12-17

[95] Hildebrand, Joe ; Jones, Michael: *JSON web encryption (JWE)*. URL https://tools.ietf.org/html/rfc7516. - retrieved 2016-12-17

[96] Bradley, John ; Sakimura, Nat ; Jones, Michael: *JSON web*

signature (JWS). URL https://tools.ietf.org/html/rfc7515. - retrieved 2016-12-17

[97] Diffie, Whitfield ; Hellman, Martin: New directions in cryptography. In: *IEEE transactions on Information Theory* vol. 22 (1976), Nr. 6, pp. 644–654

[98] Stallings, William: 9.1 principles of public-key cryptosystems. In: *Cryptography and network security: Principles and practice*. Seventh edition. ed. Boston : Pearson, 2014 — ISBN 978-0-13-335469-0, pp. 256–264

[99] Dierks, Tim ; Rescorla, E.: *The transport layer security (TLS) protocol version 1.2*. URL https://tools.ietf.org/html/rfc5246. - retrieved 2016-12-17

[100] Stallings, William: 10.5 pseudorandom number generation based on an asymmetric cipher. In: *Cryptography and network security: Principles and practice*. Seventh edition. ed. Boston : Pearson, 2014 — ISBN 978-0-13-335469-0, pp. 443–445

[101] Cooper, Dave ; Santesson, S. ; Farrell, S. ; Boeyen, S. ; Housley, W., R. andPolk: *Internet x.509 public key infrastructure certificate and certificate revocation list (CRL) profile*. URL https://tools.ietf.org/html/rfc5280. - retrieved 2017-01-11

[102] Fielding, Thomas: Representational state transfer (REST). In: *Architectural styles and the design of network-based software architectures* : University of California, Irvine, 2000, pp. 76–106

[103] Leach, Paul J. ; Berners-Lee, Tim ; Mogul, Jeffrey C. ; Masinter, Larry ; Fielding, Roy T. ; Gettys, James: *HTTP methods*. URL https://tools.ietf.org/html/rfc2616#section-9. - retrieved 2016-12-18

[104] *GraphQL*. URL https://facebook.github.io/graphql/. - retrieved 2016-12-17

[105] Beckett, Dave ; McBride, Brian: *RDF/XML syntax specification (revised)*. URL https://www.w3.org/TR/REC-rdf-syntax/. - retrieved 2016-12-19

[106] W3C OWL Working Group: *OWL 2 web ontology language*

document overview (second edition). URL https://www.w3.org/TR/owl2-overview/. - retrieved 2016-12-19

[107] Harris, Steve; Seaborne, Andy; Prud'hommeaux, Eric: *SPARQL 1.1 query language.* URL https://www.w3.org/TR/sparql11-query/. - retrieved 2016-12-19

[108] *WebID specifications.* URL https://www.w3.org/2005/Incubator/webid/spec/. - retrieved 2016-12-19

[109] *Solid specification.* URL https://github.com/solid/solid-spec. - retrieved 2016-12-17

[110] *WebAccessControl - w3c wiki.* URL https://www.w3.org/wiki/WebAccessControl. - retrieved 2016-12-19

[111] *Databox.me.* URL https://databox.me/. - retrieved 2016-12-19

[112] Heo, Tejun: *Control group (v2) documentation.* URL https://www.kernel.org/doc/Documentation/cgroup-v2.txt. - retrieved 2016-12-20

[113] *Overview of linux namespaces.* URL http://man7.org/linux/man-pages/man7/namespaces.7.html. - retrieved 2016-12-20

[114] *Open container initiative.* URL https://www.opencontainers.org/. - retrieved 2016-12-20

[115] *Container runtime specification (v1.0.0-rc3).* URL https://github.com/opencontainers/runtime-spec/tree/v1.0.0-rc3. - retrieved 2016-12-20

[116] *Container image specification (v1.0.0-rc3).* URL https://github.com/opencontainers/image-spec/tree/v1.0.0-rc3. - retrieved 2016-12-20

[117] *Basisleser weiterhin kritische schwachstelle des elektronischen / neuen personalausweises. Netzpolitik.org.* URL https://netzpolitik.org/2013/basisleser-weiterhin-kritische-schwachstelle-des-elektronischen-neuen-personalausweises/. - retrieved 2017-01-05

[118] Stiemerling, Oliver: *Qualifizierte elektronische signatur mit dem neuen personalausweis – oder: QES mit nPA, ein selbstversuch. CR-online.de blog.* URL http://www.cr-online.de/blog/2014/08/26/qualifizierte-elektronische-signatur-mit-dem-neuen-personalausweis-oder-

qes-mit-npa-ein-selbstversuch/. - retrieved 2017-01-05

[119] Bundesregierung für Informationstechnik, Der Bundesbeauftragte der: *IT-beauftragter der bundesregierung de-mail*. URL http://www.cio.bund.de/Web/DE/Innovative-Vorhaben/De-Mail/de_mail_node.html. - retrieved 2017-01-06

[120] Neumann, Linus: Stellungnahme zum elektronischen rechtsverkehr.

[121] Spiekermann, Sarah: *Ethical IT Innovation: A Value-Based System Design Approach* : CRC Press; Taylor & Francis Group, LLC, 2015 – scale — ISBN 978-1-4822-2635-5

[122] Dean, Eeffrey ; Ghemawat, Sanjay: MapRednce: Simplified data processing on large clusters (2004)

[123] Dierks, Tim: *The transport layer security (TLS) protocol version 1.2*. URL https://tools.ietf.org/html/rfc5246#section-7.4.6. - retrieved 2017-01-09

[124] *Mutual authentication*. URL https://en.wikipedia.org/w/index.php?title=Mutual_authentication&oldid=737409981. - retrieved 2017-01-10. — Page Version ID: 737409981

[125] *Networking 101: Transport layer security (TLS) - high performance browser networking (o'Reilly). High performance browser networking*. URL https://hpbn.co/transport-layer-security-tls/#tls-session-resumption. - retrieved 2017-01-12

[126] Joseph Salowey, P. Eronen, H. Zhou: *Transport layer security (TLS) session resumption without server-side state*. URL https://tools.ietf.org/html/rfc5077. - retrieved 2017-01-12

[127] *BSI - technische richtlinien des BSI - BSI TR-03130 eID-server*. URL https://www.bsi.bund.de/DE/Publikationen/TechnischeRichtlinien/tr03130/tr-03130.html. - retrieved 2017-01-06

[128] *Personalausweisportal - eID-server*. URL https://personalausweisportal.de/DE/Wirtschaft/Technik/eID-Server/eID-Server_node.html. - retrieved

2017-01-06

[129] Stallings, William: 9.1 public-key infrastructure. In: *Cryptography and network security: Principles and practice*. Seventh edition. ed. Boston : Pearson, 2014 — ISBN 978-0-13-335469-0, p. 307

[130] Leach, Paul J. ; Berners-Lee, Tim ; Mogul, Jeffrey C. ; Masinter, Larry ; Fielding, Roy T. ; Gettys, James: *Hypertext transfer protocol – HTTP/1.1*. URL https://tools.ietf.org/html/rfc2616#section-10. - retrieved 2017-01-20

[131] *OAuth core 1.0a*. URL https://oauth.net/core/1.0a/#rfc.section.4.2. - retrieved 2016-11-01

[132] Hardt, Dick: *The OAuth 2.0 authorization framework*. URL https://tools.ietf.org/html/rfc6749#section-2. - retrieved 2016-11-01

[133] *OAuth core 1.0a*. URL https://oauth.net/core/1.0a/#rfc.section.7. - retrieved 2016-11-01

[134] Hardt, Dick: *The OAuth 2.0 authorization framework*. URL https://tools.ietf.org/html/rfc6749#section-7. - retrieved 2016-11-01

[135] Brickley, Dan ; Guha, R.V.: *RDF schema 1.1*. URL https://www.w3.org/TR/rdf-schema/. - retrieved 2017-02-12

[136] Foundation: *OpenEHR - EHR information model*. URL http://www.openehr.org/releases/RM/latest/docs/ehr/ehr.html. - retrieved 2017-01-28

[137] W3C: *Points of interest core*. URL https://www.w3.org/TR/poi-core/. - retrieved 2017-01-28

[138] Authority, ISO 20022 Registration: *ISO 20022 - universal financial industry message scheme*. URL https://www.iso20022.org/. - retrieved 2017-01-28

[139] W3C: *XML schema part 2: Datatypes second edition*. URL https://www.w3.org/TR/xmlschema-2/#built-in-primitive-datatypes. - retrieved 2017-01-29

[140] Facebook, Inc.: *GraphQL Specification*. URL https://facebook.

github.io/graphql/#sec-Input-Values. - retrieved 2017-01-29

[141] *Separation of concerns*. URL https://en.wikipedia.org/w/index.php?title=Separation_of_concerns&oldid=747272729. - retrieved 2017-01-24. — Page Version ID: 747272729

[142] Kasten, James ; Barnes, Richard ; Hoffman-Andrews, Jacob: *Automatic certificate management environment (ACME)*. URL https://tools.ietf.org/html/draft-ietf-acme-acme-04. - retrieved 2017-01-11

[143] Carlson, Nicholas: *Facebook connect is a huge success – by the numbers*. URL http://www.businessinsider.com/six-months-in-facebook-connect-is-a-huge-success-2009-7. - retrieved 2016-12-16

[144] *Usage share of operating systems*. URL https://en.wikipedia.org/w/index.php?title=Usage_share_of_operating_systems&oldid=764383522#Public_servers_on_the_Internet. - retrieved 2017-02-12. — Page Version ID: 764383522

[145] *Accordion (GUI)*. URL https://en.wikipedia.org/w/index.php?title=Accordion_(GUI)&oldid=758084292. - retrieved 2017-02-01. — Page Version ID: 758084292

[146] *Aldeed/node-simple-schema. GitHub*. URL https://github.com/aldeed/node-simple-schema. - retrieved 2017-01-29