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

G. Jahn

February 1, 2017

Abstract

Data is the currency of tomorrow. Organizations, whether in the private or public sector, are gathering enormous amounts of personal (big) data. This data is harvested and incorporated by these third parties, but were created by individuals and should, therefore, belong to them. People are depending on their data. Their identity as well as their personality are defined by their personal data. Meanwhile data silo operators are hammering onto these haystacks eagerly trying to find any correlations worth interpreting, thereby almost inevitably discriminating against the rightful owners. To reduce the possibility of discrimination only bare minimum of data required should be handed over to a third party. Thus the individual has to be in charge of the whole process. A personal data service will empower its user to regain full control over her data and facilitates detailed information on every data flow. To be able to trust such a tool, the user should be able to look inside. Therefore a personal data service has to be open source and developed transparently, which would then also encourage self-hosting.

Contents

1 Introduction **6**

1.1 Motivation 6
1.2 Purpose & Outcome 8
1.3 Scenarios 9
1.4 Terminologies 15

2 Fundamentals **17**

2.1 Digital Identity, Personal Data and Ownership 17
2.2 Personal Data in the context of the Big Data Movement . . 24
2.3 Personal Data as a Product 28
2.4 Related Work 30
2.5 Standards, Specifications and related Technologies 34

3 Core Principles **42**

3.1 Data Ownership 42
3.2 Identity Verification 43
3.3 Reliable Data 43
3.4 Authorisation 44
3.5 Supervised Data Access 44
3.6 Containerization 45
3.7 Open Development 45

4 Requirements **47**

5 Design Discussion **54**

5.1 Authentication 54
5.2 Data Reliability 60
5.3 Access Management 65
5.4 Data 73

5.5 Architecture 85
5.6 Environment and Setup 94
5.7 User Interfaces 98

6 Specification *(Draft)* **103**
6.1 Overview 103
6.2 Components 104
6.3 Data 104
6.4 Protocols 104
6.5 APIs 107
6.6 Security 108
6.7 Recommendations 109

7 Conclusion **110**
7.1 Ethical & Social Impact (TODO: or “Relevance”) 110
7.2 Business Models & Monetisation 110
7.3 Target group perspectives 111
7.4 Challenges 111
7.5 Solutions 112
7.6 Attack Scenarios 112
7.7 Future Work 113
7.8 Summary 113

Bibliography **114**

List of Tables

5.1 selection of characteristics that a database system has to feature in order to be suitable for either of the defined purposes 82
5.2 All platform types where components of the *PDaaS* architecture can be placed 86
5.3 Features that should be supported by the given user interfaces 99

List of Figures

5.1 PDaaS Architecture, monolithic composition 102
5.2 PDaaS Architecture, distributed composition 102

This page intentionally left blank

1 Introduction

1.1 Motivation

Nowadays it is rare to find someone that does not collect data about some kind of thing; particularly humans are the targets of choice for the *Big Data Movement* [1]. Since humans are all individuals, they are - more or less - distinct from each other. However, subsets of individuals might share a minor set of attributes, but the bulk is still very unique to an individual, given that the overall variety of attributes is fairly complex. That small amount of shared attributes 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 arn't. Stereotypical patterns are applied to these subsets and thus to all relating individuals. Thus enriched information are then used to help predicting outcomes of problems or questions regarding these individuals. In other words, searching for causation where in best the case one might find correlations - or so called *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 still a serious issue in our society, but also when humans leverage computers and algorithms to uncover formerly unnoticed information in order to include them in their decision making. For example when qualifying for a loan, hiring employees, investigating crimes or renting flats. Approval or denial, the decision is based on computed data about the individuals in question [3], which is simply 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 in machine executable language and made 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 above, humans have the habit to create more and more data on a daily basis - pro-actively (e.g by writing a tweet) and passively (e.g by allowing the twitter app accessing their current location while submitting the tweet). As a result, already tremendous amounts of data keep growing bigger and bigger, waiting to be harvested, collected, aggregated, analyzed and finally interpreted. The crux here is, the more data being made available [7] to *mine*, the higher the chances to isolate data sets, that differ from each other but are coherent in themselves. Then it is just a matter of how to distinguish the data set and thereby the related individuals from each other.

In order to lower potential discrimination we either need to erase responsible parts from the machines, thereby it's crucial raising awareness and teaching people about the issue of discrimination, or we try to prevent our data from falling into these data silos. The latter will be addressed in this work.

1.2 Purpose & Outcome

From an individual's perspective providing data to third parties might not seem harmful at all. Instead eventually one get improved services in return, e.g. more adequate recommendations and fitting advertisement, or more helpful therapies and more secure environments. That said, though it is a matter of perception what's good and bad, what's harmful and what's an advantage. Computing data to leverage decision making is essentially just science and technology and it's up to the humans how such tools are getting utilized and what purposes they are serving. Hence it should be decided by the data creators, how their data get processed and what parts of them are used.

To tackle the described issue the initial idea here is (1) to equip individuals with the ability to control and maintain their entire data distribution and (2) thus reducing the amount of *potentially discriminatory* [2] attributes leaking into arbitrary calculations. To do so people need a reliable and trustworthy tool, which assists them in managing all their *personal data* and making them accessible for 3rd parties but under their own conditions. After getting permission granted these data consumers might have the most accurate and reliable one-stop resource to an individuals's data at hand, while urged to respect their privacy at the same time. However this also comes with downsides in terms of security and potential data loss. Elaborating on that and discussing different solutions will be part of the [design process][Design].

The way how to solve the described dilemma is not new. Early days of work done in this field can be dated back to the Mid-2000s where studies were made e.g. about recent developments in the industry or user's concerning about privacy, and the term *Vendor Relationship Management (VRM)* were used initially within the context of user-centric personal data management, which also led into the *ProjectVRM* [8] started by the *Berkman Klein Center for Internet & Society at Hardvard University.* Since then a great amount of effort went into this research area until today, while also commercial products and business models trying to solve certain problems. For instance concepts such as the *Personal Data Store (PDS)* [9] or a *MyData* [10] implementation called *Meeco* [11], which will all be covered in a more detailed way within the following chapter.

The work and research done for this thesis will be the foundation for an *Open Specification*, which by itself is a manual to implement a concept called *Personal Data as a Service*. Important topics like how the architecture will look like, where the actual data can be stored, how to obtain data from the external API or what requirements a user interface for data management need to satisfy, will be examined. After the thesis will be finished, the majority of core issues should already be addressed and can then get outlined in the specification document. Only then the task to actual implement certain components can begin. The reason for that is, when sensitive subjects especially like people's privacy is at risk, all aspects in question deserve a careful considerations and then get addressed properly. Thus it is indispensable to put adequate effort primarily into the theoretical work. To be clear though, that doesn't mean writing code to test out theories and ideas can't be done during research and specification development. It might even help to spot some flaws and eventually trigger evolvement.

To ensure a great level of trust to this project and the resulting software, it is vital to make the development process fully transparent and encourage people to get involved. Therefore it is required to open source all related software and documents [12] from day one on.

In summary, this document is meant to be the initial step in a development process fabricating a tool to manage all data defining a data subject's identity, that is controlled and administrated by that individual, so that maybe she is giving a more precise understanding about where her personal information flows and how this might effect her privacy.

1.3 Scenarios

The following use cases shall depict different situations and possible ways such emerging software might be applicable or useful, while providing it's user with more control over her personal data. Some of them are more practical and realistic, like ordering and purchasing online a product, others might have no current usage, but showing a certain potential to become more relevant when new technologies and business models emerge, followed by new demands of data.

Ordering a product online

The data subject searches through the web to find a new toaster, since her old one recently broke. After some clicks and reviews, she found 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, caught her attention. She decides to have a deeper look into the toasters and thus has heading towards the original web shop entry. Finally she came around and put the item onto her card, despite tha fact, that she has never bought something from that online shop before. Then she proceeded to checkout to place her order. The shop-interface is asking her to either insert her credentials, proceed without registration or sign-in, or insert s URI to an endpoint of her *Personal Data as a Service*. TODO: the following description might need some adjustments according data flow / process description She opens up the management panel of her *PDaaS* and creates a new entry in a list of data consumers, that already have access to characteristics of her personal data. As a result, she receives a URI, which she inserts according, as mentioned before; after she assures herself that the data exchange with the shop through the browser is based on a secure connection (HTTPS). Under this URI, the shop-system can then request data, that is required for a successful transaction. Moving on to the next step after submitting the URI, the data subject is ask to decide how she would like to pay. The choices are: credit card, invoice, online payment provider of choice or bank transfer. She chooses the last one, submits her selection and thereby completes her order. After a moment, a push notification pops up on her mobile device, which is a permission request from her *PDaaS*, asking for granting the shop-system, she just places the order, access to her full name, address and email. Additionally she can decide between three states of how long the permission wil be valid: *only one time*, *expires on date* and *granted, until further notice*. Since she never ordered at this shop before and might never again, she decided to grant access only for this specific occasion. After the shop-system receives the data, it sends an email to the data subject, containing some information about her order, including the shop's bank details. which then enables her to actually pay the amount due. After the system recognizes the payment has coming in, it triggers the shipment of the toaster. In order to get a full impression

of how the whole process might have look like when the data subject had chosen one of the other payment methods, the differences will be describes in the following. If the data subject would have wanted to pay with her credit card, the only difference would have been, that the shop-system had requested also to access the credit card number and it's belonging secret, and when sending the email the system would have omitted the information about the shop's bank details. Being able to choose paying with invoice where possible only because the *PDaaS* response has indicated, that it's containing *profile data* is certified and therefore trustworthy. Which reduces the shop owner's risk and would have enabled him in case of fraud or misuse to take action. Choosing to involve paypal as a *middleman* to process the payment, requires the data subject to had already granted paypal certain access to her *PDaaS*. If that's the case, then the shop-system would have ask also for her paypal-ID, which then the system will use to request the payment directly from paypal. This on the other side will cause paypal to consult the *PDaaS*, which results in a second notification, asking the data subject for permission to proceed. After the payment transfer was successful, the shipment will gets initiated. And with the package arriving at the data subject's doorstep the whole transaction has finished.

Interacting with a social network

Entering a social network for the first time, only take the URI to the data subject's *PDaaS* and a password. The data subject receives a notification on her mobile device asking for permission to access certain data about her. If her mobile device is currently not at hand, she can also use the administration panel provided by her *PDaaS* and reachable with a web browser on every internet-enabled device. Within that panel pending permission reviews will be indicated. Whether the data subject has already reviewed the request or not, she should be able to login to the social network. After doing so, she should not be able to see any of her information. After granting permissions to the social network to accessing certain data *until-further-notice* and reloading the session, she then should see all her So every time, someone on that network tries to access her information, whom she has allowed to see that information (which is managed by the user only from within the

network), the network pulls the data from the data subjects's *PDaaS*, if it's still permitted to do so. It's also imaginable, that the social network and a *PDaaS* are establishing a backward channel. This channel could be used to send all the content she would create over time while interacting with the social network and it's participants back to her *PDaaS*. The network itself only stores a reference to all content object, whether it's for example an image, a post or comment on somebody else's post and if it's needed the actual content will be fetched from the data subject's *PDaaS*.

Applying for a loan and checking creditworthiness

The data subject would like to buy an apartment. In order to finance such a acquisition, she needs a funding, which in her case, will be based on a loan. During a conversation in a credit institute of her choice, an account consultant describes to her what data will be required in order to decide about her creditworthiness. While giving a consensual nod, she takes out her smartphone and brings up the management panel of her *PDaaS*. With a few taps she has just created a new *data consumer*. The panel then shows a QR-Code, that holds a URI to a dedicated endpoint of the data subject's *PDaaS*. She shows that code to her consultant, who then scans it. While handling some more formalities and talking about several issues and possible products she might be interested it, she gets a notification on her phone, informing her about a permission request the institute just made. It lists all the different data points the institute would like to access in order to calculate her scoring, 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 decided to just partially allow access to the requested data and just for this time and purpose. The consultant kindly pointed out, that these decisions might have an impact on the scoring and thereby on the lending and it's terms. After the consultant got a signal from the computer system, the two then finishing up their meeting and the consultants informed the data data subject about the next steps, which includes a note, that the institute will contact her within the next few days, when they have come to a conclusion. In case of a positive outcome a new appointment need to be made, for doing all the paperwork

and signing the contract. From a technical point of view, two different ways of computing the score are imaginable. The first one would be, transferring only the plain data - request, containing the query and response containing the data - including the expire date and information regarding the signature state. But the actual computations and analytics to obtain the score, will happen within the infrastructure of the credit institute. When this process is over, all transferred personal data has to be deleted. An alternative could prevent the data from leaving the *PDaaS*, in which the institute's request won't consists of a data query. Instead it would came along with an chunk of software and some information on how to run it. The *PDaaS* server will provide an isolated runtime in which the software then gets executed. After the process has finished, the result will be send back to the credit institute's infrastructure.

Maintain and provide it's own medical record

Some time ago on a hiking trip in a moment of carelessness the data subject has accidently broke her leg. She came into a hospital and went straight into surgery, where the physicians could fix the injury. Time went by and the leg has healed completely. After she woke up today she felt some pain coming from that area where her leg was broken. She decided to call in sick and went straight to a doctor nearby. During her recovery she visited that doctor regularly. At the reception desk, she opens up the *PDaaS*'s management panel on her smartphone and searched through the list of data consumers. After she found the entry for this clinic, she flipped her phone to show the receptionist the corresponding QR-Code, which she started to scans immediately. However the receptionist couldn'd see any data on the screen, because the access has already expired. The data subject only had permitted access for the estimated time of recovery, which was over some time ago. That's why she got a notification, to re-grant some access. Going through the data points the clinic-system has requested, she noticed that her address is incorrect. Last month she moved out and into a bigger apartment just down the street. She must have forgot to change that data, which she corrects immediately right before submitting the access configurations for the clinic-system. She also included the access to all the data originated

from that time after her accident. A moment later the receptionist confirms to now being able to see all necessary data. The data subject takes a seat in the waiting room. While passing some time, she had a deeper look into her list of data consumers; some of them she couldn't even remember and for others she was surprised to what data she has granted access to and started to reduce certain permissions, if it was appropriate in her eyes. She even removed some of the entries. The appointment with her doctor went great. He even had to review the x-ray images in order to make a adequate differential diagnosis. After the visit, she had to make a quick stop at a pharmacy along the way to pickup the drugs her doctor had prescribed for her to reduce the pain. She had to wait in the queue with two other customers being in front of her. She realized, that it's the first time she has been here. So she prepared a new entry in her data consumer list, including all information about her prescriptions. So by the time she get served, she just let the person behind the register scan her code. In the next seconds the data subject gets a quick confirmation notification about the request that just happened. A moment later the pharmacist come back with her drugs, which she then pays in cash and the transaction is done.

Vehicle data and mobility

Assuming a car itself has no hardware on board in order to establish a wireless wide area connection to an outside access node. Only from the inside one can connect to the car (wired or wireless). After entering a car, on the data subject's mobile device pops up a notification asking for permission to connect to that device. In addition to the expiration date, the data subject can choose to en- or disable two more options. First, a wifi network with an uplink to the internet can be provided to everyone inside the car. Secondly, connections, the car might want to establish, in order to emit data via internet - which, regardless, have to go through the currently linked mobile device. Thus the device owner gains full control over any external data transfer that might happen. This again would allow two things: (A) permission management for all outgoing data and (B) funnel all data generated and provided by the car into the *PDaaS* associated with that linked device. It might also be feasible to deny any connection the car is trying to make. Thus the data

will only be stored in the *PDaaS*. If somebody is interested in such then have to ask for access permission. That same concept about movement tracking and vehicle data could also be applied to driving (motor) bicycle.

1.4 Terminologies

Web Service TODO

Open Specification TODO

Big Data deep learning, neural networks

Profile Data individual's inherent data; TODO

Personal Data (TODO) Personal Information predominantly static data points related to an individual

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

Personal Data Store TODO

Vendor Relationship Manager The *ProjectVRM* defines the a VRM as follows: "TODO" [13]

Personal Information Management Systems (PIMS) TODO [14]

serverless TODO https://auth0.com/blog/2016/06/09/what-is-serverless/

Digital Footprints TODO

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* using a PDaaS to control (and probably host) her personal data; sometimes referred to as *data controller*

(Data) Consumer Third party, external entity requesting data, authorized by the data subject to do so; sometimes referred to as *(data) collector*

Data Broker(s) entities with commercial interests, that collect, aggregate and analyze information/data of any kind - in this case about human beings - from different sources in order to enrich the data sets, to finally license the resulting corpora to other organisations. [15]

Permission Request initial attempt to request permissions for accessing

certain data from the *PDaaS*; third party registers as *data consumer*

Access Request obtain/request actual data from the system (requires a third party to be registered as a *data consumer*)

Permission Profile a data set about a third party that already made a permission request. The set contains additional information and access rules

Data Access after a third party's *permission request* got reviewed and saved, that entity is then able to make an attempt to access data.

Endpoint an endpoint is defined as the part of the URI that is unique to every *data consumer*, or to be more precise, unique to every *permission profile*. Usually it is the first part of a URI, whereas following parts stand for different resources that might be available within that endpoint It can also be seen as group of resources that all can be accessed by under specific circumstances

2

Fundamentals

The following chapter shall provide the foundational knowledge about concepts like *Personal Identity* or *Big Data* and therefore ensures a common understanding on their relation to the problem this work tries to solve. Additionally it is given a brief overview on what existing standards and technologies might be used, and summarizes the research already been made as well as it's current state.

2.1 Digital Identity, Personal Data and Ownership

- *Digital Identity*
  - what is a *DI*? and in comparison to *Personal Data*?
  - what is required to make the PDaaS used or seen as a *DI*?
- *Personal Data* definition
  - general - freely spoken
  - as of EU law (incl citation)
  - as of US law (incl citation)
  - is it just policy/guideline or enforceable too (law/rule)? what

relevance/impact have companies *terms and conditions*?
  - EU and USA (since server might be located outside the state or effective range)
- *Ownership* of personal data
  - who is the owner in what situation or under what circumstances?
  - am I the owner when I was the one who was collecting them? Does it depend on whether the resource was public or somewhat private?
  - what will happen with her data service after a person died?

- A **Digital Identity** is a non-physical abstraction of an entity, such as an organisation, an individual, a device or even software, which allows bidirectional association. In the context of this document, it only refers to human beings. Therefore a *digital identity* is the individual's representation in digital systems, consisting of identity-defining data, such as *personal information* and it's history and preferences [16]. *Personal information*, in this case, refers to inherent (date of birth) and imposed (credit card number) characteristics.

- From a technical perspective a DI is essentially a collection of characteristics, attributes and time series data (e.g. interaction logs or bank account history). A subset of those attributes combined can form unique fingerprint, like certain single data points (e.g. social security number) in their own context might be, too. Thus it might not be necessary to know the values of all attributes in order to identify a person as the rightful owner and physical counterpart. It can also be seen as an avatar in the digital world or as the digital part of a human's identity. Therefore it's important to not view the *DI* as a reduction of a living individual to some bits and bytes, but rather as a appropriate representation for certain purposes and contexts.

- It is also possible to provide an additional level of authenticity insurance for data related to an entity. Therefor an unrelated third party, which needs to be approved not only by the related individual, but also by all entities participating in a context, which might be relevant e.g. for some administration purposes.

- But the concept would also impose a new level of attacking vectors to

the identity owner, such as identity theft. The attacker is no longer required to be physically present to be able to steal certain unique identifiers from a person. It is sufficient to gain access to area where the sensitive data is stored.

- In the context of this document and all related work, **Personal Data** is specified as a combination of an individual's *Digital Identity* and all of it's ever created intellectual property [17] (e.g. posts, images, tweets or comments). This includes all sorts of tracking data and interaction monitoring, as well as metadata manually or automated enriching content (e.g.geo-location attached to a tweet as meta information). Data, captured by someone ore something on or about the individual's private living space and property. Simply every data point reflecting the individual's personality - partly or as a whole - is seen as *personal data*.

- The european *Data Protection Regulations* defining *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; > *[18]*

- The U.S.A. has little legislation on defining and protecting consumer's privacy. At least they have no explicit bills addressing such area [19]. Though some of the existing sectoral laws consist of partially applicable policies and guidelines [20]; most of them addressing specific types of data. In 2015 the White House made an attempt to fill the gap with the *Consumer Privacy Bill of Rights Act*, but to this date it didn't passes the draft state. According to the critics, it lags of concrete enforceable rules consumers can rely on [21]. 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 [...] > *[22]*

- followed by a list of concrete data points, e.g. email or postal address, name, social security number and alike. Aside from the legislation with bills, a few third-party organisation can also participate by and add new or overwriting existing rules and policies. Namely for example the *Federal Communications Commission* (FCC), recently releasing *Rules to Protect Broadband Consumer Privacy* including a list of categories of sensitive information [23], which 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. > *[24]*

- Despite minor difference in detail, they all have similar ideas of personal data and their belonging. Even though, the version proposed by EU is almost identical with the definition introduced for the context of this work. Although the FCC's statutory authorities might be somewhat debatable regarding certain topics, the *Communications Act* as a U.S. federal law equips the FCC with power to regulate and legislate.

- Having a common opinion on what data points are belonging to person is the foundation to define a set of rules on how deal with *Personal Data* accordingly. Every business, operating within the EU, is required[1] to provide it's users with a *Privacy Policy*, while e.g. in the U.S. - as mentioned above - only partially and depending on context and data type users must be informed about which and how their data get processed [25].

- A user commonly agrees on the privacy policy, by starting to interact with the author's business, thus every *Privacy Policy* is required to be publicly accessible; e.g. before creating an account. > By clicking Create an account, you agree to our Terms[2] > and that you have read our Data Policy[3], including > our Cookie Use[4]. > *[web_2016_facebooks-*

[1] according to article 12-14 of the *EU General Data Protection Regulation 2016/679*
[2] https://www.facebook.com/legal/terms
[3] https://www.facebook.com/about/privacy
[4] https://www.facebook.com/policies/cookies/

landing-page_policy-acknowledgement]

- It can be seen more likely an information notice, that translates and specifies general given law, rather then a contract.
- With such knowledge at hand, it is up to each individual, if the service's benefits are worth sharing some personal data, while simultaneously acquiescing potential downsides concerning the privacy of such data.
- Every entity who is doing so, muss process Personal data according to the law and their *Privacy Policy*. If they policies are violating existing law or the entity effectively goes against the law with their actual doing, penalties might follow. Depending on the level and impact of their infringement in addition the law itself, aside from revising their wrongdoings the entity might have to compensate the affected individuals, pay a fine or get revoked their license.
- Not only privacy laws, but every legal jurisdiction has it's limitations - concerning their territorial nature - which makes legislation not exactly an appropriate tool when it comes to fixing existing issues and strengthen the individual's privacy and rights in a global context like the *world wide web*. If no international agreement is in place [26], only those laws are considered valid and enforcible where the organisation is registered, and maybe the fact where (meaning in which area of jurisdiction) the their servers are located or the data is processed and stored.

Whereas **Ownership** of *Personal Data* has no legal ground foundation what so ever. The concepts of intellectual property protection and copyright might intuitively be applicable, because the data, created by the data subject, seems to be her *intellectual property*. Such property implies to be a result of a creative process though, but unfortunately there is no *threshold of originality* in facts, like *personal information* is [27].

- Ownership in the sense of having exclusive control over it's personal data and how they get processed at any given point in time; this not only comes with high costs, but is also very inconvenient for both parties - data subject and data consumer. It consists of two concepts: (A) the right to do what every is desired with their property and (B) in

which rules and mechanisms the ownership can be assigned to someone [28].

- The european DPR[5] contains only one occurrence of the word *ownership*, which is not even related to the context of *personal data* or the *data subject*. It only stats, that *"Natural persons should have control of their own personal data."* [29]. Whereas Commissioner J. Rosenworcel of the FCC wants *"consumers [...] to [...] take some ownership of what is done with their personal information."* [30]

- Typically the question of data ownership is addressed in data consumer's *Terms of Service* (ToS), which an individual might have to accept in order to establish a (legal) relationship with it's author. I should be kept in mind, that *ToS* might change over time; not necessarily to the users advantage. All addressed issues (by the ToS) must not violate any applicable or related law, otherwise the *ToS* might not be legally recognized. Taking the following excerpts from different *ToS*:

  > 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 [31])*
  >
  > 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 [32])*
  >
  > 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 [33])*
  >
  > 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 [34])*

[5]EU Data Protection Regulation

All these statements are followed by the same term, stating that the user grants the author a worldwide license to do almost any imaginable thing with her data. This even applies to Apple, 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"* [34].

- It is worth noticing, that in every *ToS* it is only referred to the data subject's content, not all her personal data. As mentioned above, personal information are no intellectual property, but playing an important role in data analytics though. Which is why *privacy policies* are in place, to ensure at least some user enlightenment, even though it doesn't compensate the lack of control.

- In addition to that, the meaning of *ownership* used in the quoted *ToS* is missing a clear outline and thus causing ambiguity and leaving room for interpretation. Nor the actual definition of *ownership*, as described earlier, is applicable for these kind of cases, since the user losing all its control is by design. Handing over data to the consumer annihilates the exclusive control over the data and revokes the ability of assigning such control. There is no (legislation based) way to establish a feasible concept of *ownership*, if the data consumer has no motivation to promote the user the a comprehensive owner of her data.

- Leaving all the legal layer aside for a moment and switching the perspectives a bit; Data consumers might argue, that they had invested in enabling themselves to collect, process and store personal data, so it belongs to them. But from the data subject's point of view it might only be the case as long as as she would benefit as well somehow, e.g. using products, services or features, offered by consumers, which quality depends on personal data. If the data subject chooses to move to a competitor might what to bring her personal data with her. But then again the former data consumer would object, competitors would benefit from all investments the consumer has made, but without any effort. Though, not entirely wrong, two aspects need to be emphasize. (A) In order to archive a high level of quality for their analytics and therefore in making right decisions to gain improvement, it's vital to huge amount of effort in developing these underlying technologies,

not only in acquiring personal data. Which again only constitutes (B) the foundation of various subsequential computations followed by an ongoing collecting, aggregation and analytics of actively and passively created data and metadata (e.g. food deliver history or platform interactions and tracking). Given the initially introduced definition of *personal data* it appears to only be a fraction of the involved data belonging to its owner. The larger part consists of highly valuable metadata [35] [36] and therefore should remain to the data collector and either be deleted or sufficiently anonymized, if the owner cancels the relationship. The data subject should not depend on the collector's willingness when it comes to handing over her personal data (e.g. list of favorites or delivery history). Instead, using her own tool to provide the consumer with required data (e.g. list of favorites) or tap into her data creating interactions (e.g. food deliveries) on her own.

- Whether an individual dies or a user deletes her account, as long as certain data point are shared with / connected to other users, the data will remain. At least when it comes to facebook.

- Generally speaking, all data solely associating with an individual, is in the ownership of the same. But since it doesn't exist any legal concepts on *personal data* ownership, a technical solution could help to regain some control.

2.2 Personal Data in the context of the Big Data Movement

- big data itself initially can be seen as a *huge blob of data* containing more or less structured data sets [37], whose size might have exceeded the capabilities of retrieving certain information almost only by hand. Such high data haystacks usually come along with new challenges in logistic and resource management, when information retrieval needs to get automated on a large scale [38]. Theses practices are commonly referred to *Big Data (Analysis)* including distributed computing and machine learning.

- Big Data, or to be more precise, collecting and analyzing big data, serves the prior purpose to extract useful information, which on the other hand depends on what was the opening question about, but also what data sets the corpus is containing.

- At first, (A) formalizing question(s) that the results have to answer. Secondly, (B) deciding what data is needed and appropriate and then start collecting. Third, (C) designing data models accordingly and correlate with the data (D) next, analyse and interpret the results. (E) last but not least, make business decisions based und the analyses ([39] Fig. 3).

- machine learning/data mining –> computers trained to find corona-tions

- since quite a few businesses (in terms of purpose or intention) are based around the concept of customers, which are generally somewhat entities consisting of at least one human being, personal data takes a major part in what *Big Data* can be about. In the context of this thesis, these entities are individuals with a unique identity. And to understand the behaviour, decision making and needs of her customers a vendor, who owns the business, needs to know as much as possible about them, when she wants to know what changes she needs to address in order to move towards the most lucrative business.

- personal data and information are reflecting all this knowledge. It starts with profile (or sensitive) data, such as gender, age, residency or income, goes on with time series events like geo-location changes, or web search history and goes all the way up to health data and self-created content like *Tweets*[6] or videos.

- all these classes of personal data hold a major share[7] in the field

[6] public massages published by an account on twitter.com, which will be displayed in the timeline of all her subscribers and also might contain additional types of content like images, links or video

[7] it doesn't matter whether an individual or just someone on behalf of an organisation spend money for something. at the end of the day, they are all humans on this planet and in a capitalistic oriented world money needs to flow and profits needs to be maximized. So to know where it will flow or why it will flow in a certain direction it is crucial to know everything about it's decision maker - the humans on this planet.

of data analytics (TODO: find statistics showing shares of data types/classes/categories, [40] [41])

- but, depending on the specific attributes, they might be not that easy to acquire. in general most businesses obtain data from within their own platforms. some data might be in the user's rang of control (e.g. customer or profile data), but most of the data comes from interacting directly (content creation, inputs) or indirectly (transactions, meta information). the level of sensitivity is mainly based on the purpose of the platform (benefit for the user) and what is the provider's demand from the users commitment (e.g. required inputs or usage requires access to location)

- from a technical perspective collecting passively created data is as simple as integrating logging mechanisms in the program logic. since the industry moved towards the cloud[8] most scenarios utilized server-client architectures. Furthermore the *always-on* philosophy evolved to an imperative state. standalone software is starting to call the author's servers from time to time, just to make sure the user behaves properly. For browsers it was already a common narrative to make here and then requests to the server - still preventable though, but when it comes to native mobile apps it is almost impossible [42] to notice such behaviour and therefore preventing apps from doing so.

- these architectural developments were inducing the gathering of potentially useful information from all over the system on a large scale [43]. Logging events, caused by the user's interactions, on the client, which then get forwarded to backend servers. Or keeping track of all kinds of transactions, which is done directly in the backend. Before running together in a designated place, all these collected chucks of data (TODO or "data points") are getting enriched with meta information. Finally get stored and probably never removed again - all for later analyses.

- The mindset in the *Big Data Community* is grounded on the basic assumption of *more data is more helpful*, which already is emphasised by

[8] side note - one might come to the conclusion, that only the trend towards the *cloud* made it actually possible to collect to such an extent we are all observing these days, because standalone software should not necessarily require internet connection and therefore the vendors had no way to gather information whatsoever

the often-cited concept of the three *Vs* (Volume, Velocity, Variety) [44]. which is not entirely wrong, because it lies in the nature of pattern and correlation discovery, to provide increasing quality results [45], while enriching the overall data with more precise data sets. But when new technologies are emerging, questioning the downsides and possible negative mid- or long-term impacts are typically not very likely to be a high priority. The focus lies on e.g. trying to to reach and eventually breach boundaries while beginning to evolve. So non-technical aspects such as privacy and security awareness doesn't come in naturally, instead a wider range of research needs to be done alongside the evolution process and the increasing adoption rate in order to uncover such issues. Only then they can addressed properly on different levels - technical, political as well as social. So that the *Big Data Community* itself is able to evolve, too. All in all it's a balancing act between respecting the user's privacy and having enough data at hand to satisfy the initial questioning with the computed results. Therefore people working in such contexts need to have advanced domain knowledge, be aware of any downsides or pitfalls and need to be sensible about the ramifications of their approaches and doings. Such improvements are already happening, not only originating from the field's forward thinkers [46], but also advocated by governments, consumer rights organisations and even leading Tech-Companies start trying to do better [47] [48] [49] - as discussed in the section [TODO see personal data as of the law],

- earlier in the text a difference was made between actively created and passively created data
- based on that one could say *profile/account data* is actively created, because it got into the system by the user's actively made decision to insert these information into a form and submit it - for whatever reason. whereas detecting the user's current location and adding this information to the submitted form is *meta data*
- of cause, it is debatable whether these kind of data belongs, in the sense of being the rightful owner, to the user or to the author or owner of the software containing the code that effectively created the data.
- maybe personal data is every data/information whose creation (or dig-

ital existence) is a direct result of user interaction/engagement?

- lets have a look into what the rule book says about that –> next topic (law)

2.3 Personal Data as a Product

- *Big Data Analytics* by itself just comprises a structured and technical-aided procedure, serving the purpose of finding invisible information, that might be helpful to make (right) (business) decisions. Though, if one would ask data collectors about their motivation, most likely the answer would be something along the lines of PR phrasing like *"We want to have a better understanding of our customers"*. But to do what exactly? To predict what might be the next thing I am supposed to buy Or what things I probably would like to consume but most certainly not yet know of?

- Let's take a look at some examples. An advertising service uses tracking data for targeted advertising. The more information they have about an individual, the more accurate decisions they are able to make about what ads are the ones the individual most likely will click on and disclose with a successful purchase. As a result this makes the placed advertisement more valuable for ad service and therefore more expensive to the advertisers, because of a high precision. Or a streaming provider's content recommendation is also based on heavy user profiling done by looking at her consumption history, tracked platform interactions and probably many more vectors. Another example is *Google Traffic* [50] [51], a service, integrated as a feature in *Google Maps*, which is Google's web mapping service. *Google Traffic* visualises real-time traffic conditions, when using *Maps* as a navigation assistant, to provide the user with a selection of possible paths, but enriched with duration, that takes such conditions into account. The data, required to offer these information, is supplied by mobile devices, constantly sending GPS coordinates with a timestamp into Google's infrastructure. This, however, only is made possible, because Google's services are widely used in addition to the fact that the majority of

mobile devices [52] is driven by Android, an mobile operating system developed by Google, that deeply integrates with it's services. For this case the same assertion can be made - the more constantly streaming geo-location data, the more precise the information are about traffic conditions. Since this information demands the real-time aspect, adding time to the equation, add a other dimension of complexity to problem.

- while the impact on our society of this first example group might be doubtable, a change of perspective opens up a different range of application areas. Such as
  - planing and managing human resources for situations, like e.g. big events or emergency situations where attendees might need some help [53]
  - predicting infrastructure workloads [TODO http://ieeexplore.ieee.org/document/7336
  - making more accurate diagnostics to improve their therapy [54]
  - finding patters in climate changes, which otherwise wouldn't be detected [55].

- Through all these examples, some of them might not necessarily founded on personal data, whereas others primarily depend on them and yet others only implicitly rely on data collected from individuals. As always, it depends on the purpose - also known as *business model* - but it seems to be consensual, that it all comes down to improving and enhancing the collector's product in order to satisfy the customers - and that on the other hand depends on what is meant to be the product and who is seen as customers.

- Putting a top 10 list of industries using utilizing *Big Data* [56] right next to visualization showing categories of personal data targeted by data collectors
  [57], at least 7[9] of these industries can be identified as data collectors, whereas less then a half[10] are taking part of being a *Data Broker*, but almost all of them are using people's personal data, whether collected

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

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

by themselves or acquired from *Data Broker*.

- At this point it's save to say, that *Personal Data* is either seen directly as a product, especially from a Dater Broker's point of view, or indirectly due to it's essential part in *Big Data* practices. The former generates direct revenue by selling these data and the latter might affect a business's product quality in a positive manner and thereby increasing revenue as well.
- 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.
- individuals then get in role of selling/offering it's own data to those who were previously collecting them

2.4 Related Work

The idea of a digital vault, controlled and maintained by the data subject, the individual, isn't that new. Holding her most sensitive and valuable collections of bits and bytes, protected from all these data brokers and authorities, while interacting with the digital and physical world, opening and closing it's door from time to time, to either put something important for her inside or retrieving an information important for someone else. While in the mid and late 2000s the growth of computer performance and capacity were crossing it's zenith (see Moore's Law [58]), at the same time the internet was starting 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, causing the shift towards parallel distributed processing and patterns alike. Thereby making it possible to rethink solutions from the past and trying to go new ways, namely the breakthrough 2007 in *neuronal networks* cutesy of G. Hinton [59]. As a result, fields like *deep machine learning*, *big data analytics* and most recently *data mining*, were gaining a wide range of

attention. In almost any industry a greater amount of resources is invested in these areas [60].

The initial research motivation can be seen as a counter-movement away from the *cloud*, starting to focus again on privacy, the individual and it's digital alter ego.

From simple middleware-solutions, via full-fledged software-based platforms, through embedded hardware devices, a great variety of approaches were starting to appear in the mid 2000s until this day. A side effect was, that over time various research teams and projects have invented and coined different terms, all referring to the same concept. The following list shows some examples *(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 research projects is *ProjectVRM*, which originated from *Berkman Center for Internet & Society* at *Harvard University*. As it's name implies, it was inspired by the idea of turning the concepts of a *Customer Relationship Management* (CRM) upside down. This puts the vendor's customers back in charge of their data priorly managed by the vendors. It also solves the problem of unintended data redundancy. Over time the project has growing to the largest and most influential in this research field. It transformed into an umbrella and hub for all kinds of projects and research related to that topic [61], 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 [62], including 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 ways across a lot of research done within this area.

Another research that is worth mentioning, because of the foundational work it has been done, is the european funded project called *Trusted Architecture for Securely Shared Service* (TAS3). The project led to a open source reference implementation called *ZXID*.[11] The major goal was, to develop an architecture, that takes all involved parties into account, whether it's commercial businesses (vendors) or it's users (customers), in order to fit into more sophisticated and dynamic processes, but at the same time demanding a high level of user-centric security facilitate i.a. by a developed policy framework. Due to these requirements the architecture ended up being rather complex [63]. *ZXID* as it's implementation incorporates several standards like SAML 2.0[12] and XACML,[13] has only three third-party dependencies which are *OpenSSL*, *cURL (libcurl)* and *zlib* and as of now it supports Java, PHP and Perl. The project lasted for a period of 4 years, but after it ended in 2011, the research work has pursued i.a. by the *Liberty Alliance Project*, which is now part of the *Kantara Initiative* [64], including all documents and results. These results were taken up occasionally, recently from the IEEE [65].

A research project, which is probably the closest to what this document aims to create, bears the name *openPDS* [66] and is done by *Humans Dynamics Lab* [67], which is part of *MIT Media Laboratories*. Despite the usual concepts of a *PDS*, it introduces multi-platform components and user interfaces including a mobile devices and separating the persistence layer physically at the same time. This facilitates administrative tasks regardless of the data subject's position and time. Moreover, with their idea of *SafeAnswers* [68], the team even goes a step further. The concept behind that, is based around *remote code execution*, briefly described in one of the user stories during 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 representation: (A) a short explanation of what the data consumer wants to know and which data might be involved and thus what information a data consumer actually will receive, instead of raw data the consumer could

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

[12] Security Assertion Markup Language 2.0

[13] eXtensible Access Control Markup Language

then use for all kinds of purposes e.g. data aggregation or mining. Aside from that, the request payload also includes (B) a code-based representation, which gets executed in a sandbox on the data subjects's *PDS* system with the necessary data as arguments. The resulting output is answer and response all in once.

Aside from all the research projects done within the scientific context, applications with a commercial interest were starting to occur in a variety of sectors, too. Microsoft's HealthVault [69], for example, which aims to replace all the paper-based patient medical record and combine them in one digital version. This results in a patient-centered medical data and documents archive, helping doctors to make the most accurate decisions on medical treatment.

Meeco [70] [71], based on the MyData-Project [whitepaper_2014_mydata-a-nordic-model-for-human-centered-personal-data-management-and-processing], which essentially just cuts out the advertisement service provider as a middle man inherits that role by itself. The platform does provide the data subjects with more control over what information they reveal, but it doesn't go the so eagerly demanded next step, which would means real uncoupling from the advertisement market and finding a suitable business model that focuses on the data subject, instead of 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* [72]. The project unifies both, research and commercial interests and runs over time period of three years until the third quarter of 2018. It aims to *"[…] to facilitate the secure exchange and easy linkage of data in business ecosystems"*, where at the same time *"[…] ensuring digital sovereignty of data owners"* [73]. It will be interesting to see how these two, yet rather distinct objectives, will 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.

Hereafter a selective list can be found of further research projects, work and commercial products regarding the issue around *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*[14])
- Polis (greek research project from 2008) [http://polis.ee.duth.gr/Polis/index.php]

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]

2.5 Standards, Specifications and related Technologies

The overall attempt is to involve as much standards as possible, because it increases the chances of interoperability and thereby it lowers the effort, that might be needed, in order to integrate with third parties or other APIs. Hereinafter, some of these possible technologies will be touched on just briefly, why they might be a reasonable choice and what purposes they might going to service.

HTTP [74], well known as the stateless *"transport layer"* for the *World Wide*

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

Web, is most likely going to fulfill the same purpose in the context of this work, because it implements a server-client pattern in it very core. Whether internal components (local or as part of a distributed system) talk to each other or data consumers interact with the system, this protocol transfers the data hat need to be exchanged. Features introduced with Version 2 [75] of the protocol are yet to be known of their relevance of use cases within this project. *WebSockes* [76] might also be a possibility to communicate between components or even with external parties, which has the advantage of high efficient ongoing bidirectional connections using for real-time data exchange or remotely pending process responses, while at the same time avoiding HTTP's long-polling abilities.

JSON[15] 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.

The open standard **OAuth** defines a process flow for authorizing third parties to access externally hosted resources, such as the user's profile image from a social media platform. The authorisation validation is done by the help of a previously generated token. However, generating and supplying such token can be initiated in a variety of ways depending on the already existing architecture and design, e.g. with the user entering her credentials (`grant_type=authorization_code`). This design tends [79] to integrate *OAuth* mistakenly but intentionally
as an authentication service rather then a authorization service; regardless if 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. While *version 1.0a* [80], commonly seen as a protocol, provides integrity for transferred data by using signatures and confidentiality by encrypting data ahead of transfer. *Version 2.0* [81], labeled as a framework, on the other side requires *TLS* and thus hands off the the responsibility to confidentiality to the transport layer below. It also includes certain process flows for specific platforms, such as *"web applications, desktop applications, mobile phones, and living room devices"* [82].

[15]The JavaScript Object Notation (JSON) Data Interchange Format; ECMA Standard [77] and Internet Engineering Task Force RFC 7159 [78]

With **OpenID** on the other side, the authenticity of a requesting user gets verified, which is by design. An in-depth description of the whole process can be found in the protocol's same-titled open standard. With decentralisation kept in mind, the protocols's nature encourages to design a distributed application architecture, similar to the idea behind *microservices*, but without owning all services involved, *decentralized authentication as a service* so to speak. An application owner doesn't have to write or implement it's own user management system, instead it is sufficient to just integrate these parts from the standard need to support signing in with *OpenID*. Equally the user is not required to register a new account whenever it is necessary, instead she can use her *OpenID*, already created by another identity provider, to authenticate with the application. The extension *OpenID Attribute Exchange* allows to import additional profile data. *OpenID Connect* [83] is the third iteration of the OpenID technology. *Connect* is to OpenID what *facebook connect* is to *facebook*, except for the additional authentication layer, which is build upon *OAuth2.0* and JWT. It therefore enables, aside from authorisation mechanisms, third parties to authenticate an OpenID-user and makes certain data available about that account via REST interface.

If it's necessary for certain components, as part of a distributed software, to make them stateless, apart from changing the architecture so that the state at that point is not needed anymore, the only other option would be to carry the state along (TODO: or "passing the state around"). This is a common use case for a **JSON Web Token** *(JWT)* [84]. A *JWT*, as it's name implies, is syntactically speaking formatted as *JSON*, but URI-safe into *Base64* encoded, before it gets transferred. The token itself holds 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 certain data could be readout and therewith get verified. Such a token typically consists of three parts: information about itself, a payload, which can be arbitrary data such as user or state information, and a signature; all separated with a period. Additional standards define encryption *(JWE*[16]*)* to ensure confidentiality and signatures *(JWS*[17]*)* to preserve integrity of it's contents. Using a *JWT* for authentication purposes is described as *stateless authentication*, because the

[16]JSON Web Encryption, Internet Engineering Task Force RFC 7516 [85]
[17]JSON Web Signature, Internet Engineering Task Force RFC 7515 [86]

verifying entity doesn't need to be aware of session IDs nor any information about a state. So instead of the backend interface being constrained to check a state (`isLoggedIn(sessionId)` or `isAuthorized(sessionId)`) on every incoming request in order to verify permissions, it just needs

When transferring data over a potential non-private channel several properties 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. To achieve this, **Symmetrical Cryptography** is used for. It states that the sender encrypts the data with the help of a key and the receiver decrypts that data also with that key. That is, sender and receiver, both need to know that one key, but everyone else should not . To agree on a key without compromising the key during that process, both entities either change the medium (e.g meet physically and exchange) or have to use a procedure, in which at any point in time the entire key is not exposed to others then sender and receiver. This procedure is called **Diffie-Hellman-Key-Exchange** [87] and is based on rules for modulo operations 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 can't be used to deduce the secret. Such behaviour is similar to the concepts of **Asymmetrical Cryptography** *(or public-key cryptography)* [88], 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.

Wrapping *HTTP* in the *Transport Layer Security* [89] (*TLS*) results in **HTTPS**. TLS provides encryption during the data transport, which reduces the vulnerability to *man-in-the-middle* attacks and thus ensures not only confidentiality but data integrity as well. *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. But relying on those cryptographical concepts requires additional infrastructure. Such an infrastructure is known as *Public Key Infrastructure* (or *PKI*) [90].

It manages and provides public keys in a directory, including related information to the owners of these 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 the largest. It is based on the widely used IETF[18] standard *X.509* [91].

REST(ful)[19] is a common set of principles to design web resources communication, primarily server-client relations, in a more generic and thereby interoperable way. Aside from hierarchically structured URIs, which reflect certain semantic content, it involves a group of rudimentary vocabulary[20] to provide basic Create-Read-Update-Delete operations across distributed systems. The entire request need to contain everything that is required to get proceeded, e.g. state data and possibly authentication. These operation normally wont get applied directly to the responsible component. Instead the whole system (or certain services) exposes a restful API, with which a third party can then interact.

The *QL* in **GraphQL** [94] stands for *query language*. Developed by Facebook Inc., it's 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 a JSON format, which naturally supports graph-like data structures. This is utilized in GraphQL and implicitly embraced through it's purpose of abstraction. Data points that might be somehow related but stored in different locations, can be obtained so that both 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 not only is an abstraction layer by itself, it also moved almost any operations and flow controls into an additional layer. This so called *GraphQL* server is then responsible for resolving and executing queries.

The term **Semantic Web** bundles a conglomerate of standards released by

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

[19] *Representational State Transfer*, introduces by Roy Fielding in his doctoral dissertation [92]

[20] knows as HTTP Methods or Verbs [93] (e.g. GET, OPTIONS, PUT, DELETE)

the W3C,[21] that is based around an idea called *web of data*, which aims to *"allow data being shared and reused across"* [web_2016_w3c_semantic-web-activity]. Those 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[22] basically defines the syntax. OWL[23] provides the guidelines on how the semantics and schemas should be defined and with SPARQL [97], the query language, data can be retrieved. One could not help but picture the web as a database, queryable data with URIs that is embedded in arbitrary websites. For example, 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) [98]* - provided in a certain syntax *(RDF)* and 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 URI that works as a unique identifier, 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 and embedding or linking data points rather then creating instances, which might only be valid at that point in time, in short preventing redundant work. Related to the *Semantic Web* is the a project called **Solid**.[24] Based on the *Linked Data* principles and backed the *WebAccessControl* [100] system and the standards just mentioned, that project focuses on decentralization and personal data. A reference implementation called *databox* [101] combines all these technologies and is build on top of the.

The concept of application (or software) **container** is about encapsulating runtime environments by introducing an additional layer of abstraction. A container bundles just the software dependencies (e.g. binaries) that are absolutely necessary so that the enclosed program is able to run properly. The actual container separation is done, aside from others, with the help of two features provided by the Linux kernel. *Cgroups*,[25] which define or restrict how much of the existing resources a group of processes (e.g. CPU, memory

[21] World Wide Web Consortium; international community that develops standards for the web

[22] Resource Description Framework [95]

[23] Web Ontology Language [96]

[24] social linked data [99]

[25] control groups [102]

or network) can use. Whereas *namespaces* [103] define or restrict 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, KVM, virtualbox) have been used for years, but were usually too cumbersome and never reached a great level of convenience, so that only people with a certain expertise were able to handle systems build upon virtualization, but people with other backgrounds couldn't and weren't that much interested. Until *Docker* and *rkt* emerged. After some years of separated work, both authors, and others, recently joined forces in the *Open Container Initiative* [104], which aims to harmonize the diverged landscape and start building common ground to ensure a higher interoperability, and that in turn is requisite for orchestration. It also marks the initial draft of the specifications for runtime [105] and image [106] definition, on which the work is still ongoing. This concept of *containerization* also inherits the a ability known from *emulation*, because it allows a certain set of software to run on a system that otherwise is not supported, e.g. mobile devices. It only requires the runtime to be working.

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, for example changing residence information or filing tax report, which is summarized under the term *E-government*.[26] One of those developments is the so called **Electronic ID Card**{#link_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 also introduced in Germany in 2010. The so called *nPA*[27] was an important step towards an operational *e-government*. Aside from minor flaws [107] and disadvantages [108] an *eID card* can come with, the question here is, how can such technology be usefully integrated in this project and does it even makes sense. As an official document the card has one major advantage over selfconfigured or generated authentication mechanisms

[26]Electronic government

[27]in german so called *elektronische Personalausweis (nPA)*

like passwords, fingerprints or TANs.[28] 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 the government - has verified the authenticity of that individual.

When communication via email it is 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 trust that this key actually belongs to the user that it says it does. That public key is then simply uploaded by the owner to public servers where another user, who wants to write that key owner an email can obtain keys. Related to that topic, another technology emerging as part of the *e-government* development, is the german **De-Mail** [109]. 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 all the way from sender through to the recipient [110], 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 enough to prove the author's identity if the provider signs the document on the email server and that this implementation results in a legally binding document by definition of that law.

[28] Transaction authentication number

3

Core Principles

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

3.1 Data Ownership

Depending on the standpoint, the question about ownership of certain data might not that trivial to answer. As stated in the 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 her

identity and her as an individual. So instead of defining those rules meant to protect data subjects, but demanding 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 accessor is allowed to do with it. This would make the *data subject*, per definition and effectively to the owner of those data. Although, it is to be noted, that the legal rulebook for data consumers mentioned before, remains a highly important, since this project is not able to cover every use case, that might occur.

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

3.2 Identity Verification

When an instance of this system is going to be the digital counterpart of an individuals identity or it's *personal agent* [111], then everyone who relies on the information that agent is providing, must as well be able to trust the source from where that data is coming from and vice versa; the *operator* too must be able to verify the authenticity of the requesting source; regardless if it's the initial *permission request* or further *access attempts*. Based on these mechanisms, the system can also provide an authentication services to all sorts of generic or restricted platforms for the associated identity, including second factor abilities.

3.3 Reliable Data

Being able to verify the authenticity of a communication partner means only to be half-way through. Data consumers also need to trust the data itself, which is attributed to the following properties.

(A) *integrity* - which means the recipient can verify, that the data, sent to

her, 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 belongs to 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 the recipient about the lack of authenticity, so that she, herself, 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 want to obtain certain information about her in a semi-automated way. 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 this plain *query and response data* approach could be replaced by a more supervised 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 that was initiated by the *data consumer* beforehand; similar to a distributed Map-Reduce concept [112]. The opposite approach, to provide some software to the *data consumer* that is necessary to access the contents of a response or provides a runtime environment querying the system by itself, would be conceivable as well. 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 who is able of cancel the *access permissions* or applying appropriate changes. Supervising methods provide an appropriate

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 a virtualization results into an environment that can be, depending on the configuration, fully encapsulate it's 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 to a minimum.

(B) Thereby gaining higher flexibility in placing components, through which advantages can be made out of other devices characteristics. while not necessarily increasing the overall complexity of the system

(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 such aspects. Furthermore, it allows to suit more versatile and diverse scenarios, like storing data about a using data, providing sensitive profile data or getting used as a medical record. The convenience of a precise resource assignment might also become relevant for case where device's hardware specification might be somewhat low. Building a system upon a container-based philosophy and enclosing components in their own environment brings 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 understand as *open standards* and *open source*; *open* in the sense of *unrestricted accessible by everybody* and not to be confused with free - as in *freedom* - software. Advocating such a philosophy permits not only to develop implementations in a collaborative way, but en-

ables
also 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 looking into the source code and getting a picture of what the program actually does and how it works. Thus, source code reviews become possible as well. Those might reveal certain security flaws, which then are able to get fixed very quickly. Furthermore, this approach allows data subjects to setup their own infrastructure and host such a system, 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 another provider. It also encourages any kind of adjustments or customization to the system in order to serve the own's needs. Enabling an open development allows users and contributors working together and thus improve the outcome 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 about how the open specification and thus the resulting software might look like, and to give an overview about priorities (can/could, may/might, should, must/have to) on the other hand. Other chapters may contain specific references to the 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 should implement a web service, who provides supervised access to personal data.

S.A.02 **- Portability**
All major components should be designed and communicate between each

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

S.A.03 **- Roles**
The system has to define two types of roles. The first one is the operator, who is in control of the system and, depending on the architecture, must be at least on individual but can be more. The operator takes care of all the data that then get's provided and decides about which third party get's access to what data. The second type are the consumers. These are external third parties that desire certain data about or from the operator. (see Terminologies)

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 to ensure the authenticity of their identity and the data they are sending to the opponent. It should be possible to opt out to that level of reliability, if is not necessary, or to opt-in for certain aspects. However, if one of the parties demanding the other one of providing 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 is 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 to be available all the time. Even if the request won't get proceeded completely, the *data subject* can still be informed about that event; a bit 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*.

Persistence:

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

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.

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 data set and data point has to relate to a corresponding and existing 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 only one who has the permissions to add, change or remove data. This is done either by using the appropriate forms provided by graphical user interface or import mechanisms. The latter could be enabled through (A) support for file upload containing supported formats, (B) data API restricted to the operator or (C) defining an external source reachable via http (e.g. *RESTful URI*) in order to (semi-)automate additional an ongoing data import from multiple data sources (e.g. IoT, browser plugin). Additionally, it might be possible in the future to allow *data consumers* letting some data to flow back into the operator's system, after she is certain about it's validity and usefulness.

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 tool 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*.

Interfaces:

S.I.01 **- Documentation**
The interfaces of all components have to be documented; in a way that the

components themselves can be 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). After checking if the access request is permitted, the system first parses and validates the query and eventually proceeds to actually execute the included query. When querying data from the system, the *data consumer* might be required tp provide a schema, which should force him to be as precise as possible about what data is exactly needed. 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 data points in 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 communicating 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, because of the diversity of screen sizes nowadays.

***P.VIU.02* - Platform support**
The user interface must be at least implemented based on web technologies, that is provided by a server and is thus available on any platform that comes with a modern browser. To enable additional features and behavior, at least for mobile devices it is recommended to build a user interface upon native 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 the list of *access profiles*; for a better administration experience and to easily find certain entries while the overall amount increases over time.

***P.VIU.04* - Access History** The operator must be provided with a list of all past permission requests and data accesses, 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. This 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 processes, like changing *profile data*, importing data sets or manage *permission request* 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 these circumstances, the *permission request review* and *permission profile creation* might become a special challenge.

***P.I.02* - Design**
The graphical user interface must be designed and structured in such a way that is is highly intuitive for the user to operate. Thus, it is important e.g. to use meaningful icons and appropriate labels. It also means a flat and not crammed menu navigation. Context related interaction elements should be positioned within the area designated for that context. TODO: maybe emphasize more UI aspects (or not)

***P.I.03* - Notifications**
The user should be notified about every interaction with the *PDaaS* originated by a third party immediately after it's occurrence, but she must get 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 the notifications themselves must show up and pending user interactions must be indicated in the user interface.

***P.I.04* - Permission Request & Review**
A process involving data transaction must always be initiated by the data subjects. So before a *data consumer* is able to access data, first the *operator* need to *invite* him and tell him whereto address his requests. This has to be done by sending him a URI leading to an endpoint, that needs to be unique among all *data consumers* interacting with the same instance of the system. When a *data consumer* makes the first attempt to connect to the system, it must be a well formed *permission request*, which has to include information about the *consumer*, what data he wants to get access to, for what purpose and how log or how often the data need to be requested. The operator then reviews these information and creates an *permission profile* based on that information. A key configuration in such a profile has to be what defines 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 permission type set by the operator.

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

(B) reduce recurring redundant configurations.

Behaviour:

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

***P.B.02* - Real time**
Real time communication might be essential for time-critical data transac-

tion. Hence at least one user interfaces should be connected to the server through an ongoing connection to enable real time support (example scenario: permission request got reviewed on mobile device, but notification indicator reflects “still pending”). But if just one client is associated to the system, real time (in the sense of keeping UI state up to date) would not be necessary. (see P.VIU.02)

5

Design Discussion

The following chapter documents the processes of some 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 will get it's
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, which is based on the prior discussions related to that topic.

5.1 Authentication

First of all, the system has to support two roles. Any entity can be assigned to either one of them, hence entities that are trying to authenticate to the system might have different intentions. The *operator* for example wants to review *permission requests* in real time, so accessing the system from different devices is a common scenario. 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 do 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*. Furthermore, to preserve confidentiality on every in- and outgoing data (S.P.01) the most convenient solution 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 seasonable 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 [113] 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 the protocol. Within the scope of this work, those technologies are categorized in the following types (TODO: maybe find other labels): (A) stateful and (B) stateless authentication. The first one (A) includes vor example *Basic access Authentication* (or *Basic Auth*) and authentication 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 it's 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 get provided with all the 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* [114]) is done based on files containing keys and certificates, which are typically not very fluctuant in it's 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* [115], thus they are left aside, because they serve the sole purpose of performance optimization. Similar to the *Session Ticket Resumption* [116] 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 obtain data from it and reacting accordingly.

Keeping track of a state (or multiple states) on the server and keeping data that is involved
synchronized between server and client is expensive and by fare trivial. Expensive in the sense of additional resources a server would require to remember all the data for those states, that otherwise won't be needed. And it's not trivial, because this pattern requires the server to be aware of all current states (sessions) and has to have them accessible at all time. This also means, that the contents responses for certain requests might depend on preceding requests and their incoming order. Furthermore those session data has to be 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 without any possibility to get recovered. To stateless authentications non of those aspects apply. Certificates and keys as well as web tokens are both carry the information that might be necessary with them. Thus, considering those disadvantages, *public key cryptography* and web tokens are the preferred technologies for all authentication processes.

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

gin or *sign in* and requires the authorizing entity to provide some credentials consisting at least of two parts. One part, that uniquely relates to the entity but doesn't have to be private, and another part 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 and provided by a USB stick). As another possible secret (or unique object) an *eID card* could be used. Conceivable applications would be (1) to let the *operator* login to the *PDaaS* Management Tool or

(2) to approve or authorize *access requests* or *data access* attempts. How the actual login process (1) 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)*, accessing the management tool via desktop requires also a card reader, preferably with an integrated hardware keypad. Instead accessing the tool on 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 (1+2) need 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 [117] starting with applying for a permission to send a certificate signing request to a BerCA.[1] This request is send from an *eID-Server* [118] in order to get signed a public key,
which previously has been generated on a dedicated and certified hardware. This hardware is required 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

[1] (german) Berechtigungszertifikate-Anbieter

dependency, but would also weaken the system. 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 it's simplicity the concept of web tokens are fairly straightforward to implement into the *PDaaS*. But since web tokens ensure integrity and the optional confidentiality only of their own carriage but not the entire HTTP payload, both requirements need to be addressed separately. Serving HTTP over *TLS* solves this issue. 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*, because all information required to authenticate is provided by the token itself. Though, the situation is different if instead *two-way authentication* is used. For this, the system has to provide it's 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 without any external role in order to function independently so that only invited parties can get involved. With referring to the statement mentioned above, *data consumer* have to be able as well to verify the identity of the *PDaaS*, in order to prevent man-in-the-middle attacks. Address 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 *consumer* uses 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.

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 token 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. So if it's likely that a *operator* has several clients, regardless for what purposes, then the private key has to be on those clients. Though, a private key typically does not leave it's current system or 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 for associating with entities that need to authenticated, similar to approach proposed in this work. But since it underpinned by a very distinct scenario it is also very related and therefore restricted to that. Trying to adopt the standard might result in various changes 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 authors identity, by embedding a legal foundation into email-based communication. But instead of providing technically valid authenticity by 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 [119], 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 it's 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 also worth mentioning, that a a JSON Web Token implementation is feasible as well to fully replace the approach consisting of *two-way authentication* on TLS level and a private *PKI*. The disadvantage here would be, that 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 doable on the TLS-level as well; by restricting this possibility to only trusted environments like native mobile applications, because browser-based applications are not considered trusted and they lack of certain capabilities. Addressing the need of *consumers* verifying whether the *PDaaS* provided certificate 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. Even though 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 make an attack successful. 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 ensured. In this case, authenticity 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 related responses leaves 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 somehow 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 so that it is selfcontained. If instead it's not required to provide information on the data being reliable or not, 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 it's reliability or not. Depending on what data is requested, the *PDaaS* decides whether it's necessary to test for reliability or not. 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 data set. 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 are distinguish 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 data sets.

(1) **Local Verification by matching**
The probably simplest and at the same time least reliable method is

to just look at the existing data 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. 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 she does, if the matching procedure has a positive outcome. It is required to sign the whole response or at least a replicable data set 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). In this method involved no matching procedure. The external third party, verifies if the data in question are correct and if 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 *Certificate Authority*. This certificate is then served as part of a response, which enabled the *data consumer* to verify the data 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. If data has changed in the *PDaaS* and data points are affected, that are also included in this verification process, then a new certificate created, 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 method that has been used. 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 when the *PDaaS* verifies data reliability that method has to runs. Thus the *operator* is forced to interact wit the *PDaaS*. Otherwise the operators private key need to be stored somewhere within the *PDaaS*. No matter where or when, 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 vulnerable to certain attacks. Aside from that, it's highly unlikely that an *eID card* would allow to extract it's containing private keys. That is, increasing inconvenience is inevitable for this proposed method. The *Local Verification and signing* method also has the same dependencies mentioned in the discussion about the 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, this verification method eventually will not being supported in the specification.

The *Remote Verification and signing* 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. They are commonly trusted. The same goes for The *Recurring Certification*, but while the *Remote Verification and signing* method introduces a very strong

dependency to that external party, the *Recurring Certification* offers a simple loosely linking dependency. Whose design would make it even possible to obtain such a certificate manually but automate it on the other side. Nevertheless, both provide a trustworthy certification.

Finally the first method, which does just a matching of two data sets against each other. Those data sets are obtained from the same *PDaaS* storage, but at a different time; right before the request finally gets proceeded, though. The method is not very useful - in general and specifically to this issue, because not much happens within the system during that time (case were data in the storage changes during request processing are discussed in the section on *Access Management*). Even if the whole system would be compromised, this method has no use in that case, because a) if that's the situation, other issues might need more urgent addressing then ensuring data reliability for *data consumers* and b) even then, the chances are insignificant that this method result is negative. 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 that can be provide. Determined by legislation or other rules, violation might prevent the *PDaaS* from being used. Others instead - depending on their guidelines and business model - don't rely on any kind of confidence. In general, *data consumers* are expected to already have a basic confidence in a *PDaaS* and the data coming from there. Regardless of that, providing an indication about the data's authenticity 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.

For the *PDaaS* a primary goal 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 proceed with the obtained data.

5.3 Access Management

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

Below it is proposed what a general design might look like for the process of how *data consumers* get authorized and thereby access the *data subject's* personal data and how previous mentioned technologies can be assembled in order to meet the specified requirements. OR Based on the outlined technologies and specified requirements the general design for a process in which a *data consumer* gets authorized and thereby access the *operator's* personal data is proposed as followed.

Part One: consumer registration
0) The *operator* creates a new unique URI in the system

1) **Prepare registration**; the *operator* has to tell third party were and how to register as a *data consumer* by handing over a URI that is unique 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.*

NOTICE: the two initial steps could also be made from the opposite direction. Third parties put all information and data required for a registration together and present them to the operator *in form of a QR-Code, so that the* operator* can obtain it and whereby is able to proceed. This approach would short cut and hence simplify the process.*

2) **Send permission request**; 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.

3) **Review permission request**; the operator gets notified about new registration attempts, which she then has to review and decide whether to grant or refuse the requested data access.

4) **Create permission profile**; if access has been granted a new *permission profile* is going to be created. Optionally, a new *permission profile* could also be created if the access has been 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 that decision 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**; *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 the related *permission profile*. 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 that would look like mainly depends on what the *access request* contains and also what the *permission profile* determines (see *access request types* below).

4) **Respond to consumer**; handover the computed response back to the requester. There are two ways of responding to an *access request*. Either the system respond with a state 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 data in demand.

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 gets 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 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 has to be applied. If that's the case, the confidentiality of all personal data is already 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 personal data is fairly simple. During 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 to access and how long those permissions last. The later is defined as *permission type* and can be one of the following:

[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)

one-time-only access permissions are hereby granted for just a single *access request* (with respect to 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 also enabled in the *profile*. 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 *data 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* 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 that 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 send to t *data consumer*. Furthermore, it needs to be ensured that only after the *permission requests* are being reviewed and the *permission profiles* are being created, the *data consumers* receive their credentials or a notification to get started.

It is up to the *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, 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 it can be processed automatically by *data consumers*. It is also feasible to restrict that access to only registered *consumers*, in order to prevent those information from being crawled. They might be valued as meta data and therefore used in unwanted computations that could raise privacy concerns. It is imaginable to let the *operator* restrict the access to these availability information on the level of individual *data consumers* or system-wide, and to set a default configuration for that behaviour. Depending on that configurations request might fail, thus requester need to be provided with meaningful errors. Http error codes [120] might be a a sufficient fit for that purpose.

An already standardized way to implement authorization would be 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 process design introduced above. The types are `password` and `client_credentials`, which basically require identifier(s) and secret or credentials to directly request the `token`. The other two types define additional steps and interactions involving client *(consumer)* and user *(operator)* before getting the`token`. This would make the proposed process undesirably more complicated. Although the proposal includes user interactions like selecting and confirming requested permissions as well. According to the docu-

mentations [121] [122], both OAuth versions (1.0a and 2) require the client *(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 is to prevent unwanted applications for data access, because they all have to get 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 step either. Instead, it is not needed process at all, because according to the former proposed process, identifying the client happens implicitly as a result of how the resource owner *(operator)* obtains the registration request from the client (Part One: consumer registration, step 0 and 1). Further investigations show that the `access_token` semantic as from the perspective of a resource server, which are a) authentication (does this token exist?) and b) authorization (is this token valid and what does it permit?), have in part already been provided by the proposed way of using the TLS layer. Because every *data consumer* has it's own endpoint to connect with the *PDaaS* and the certificate used by the *consumer* is singed by a signature that is only used for that endpoint. This means, the *consumer* is already authenticated, when the TLS connection has successfully established. And since the endpoints relates to the *permission profiles* 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 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 were *data consumers* would

register themselves at the *PDaaS*. This already needs a secure channel to prevent man-in-the-middle attacks. Furthermore does option a) obtain a symmetric key for creating signatures used to ensue confidentiality and integrity in the subsequent steps. All those cryptographic procedures need to be adopted when implementing the specification emerging from this work and when interacting with those implementations. While this can cause much more harm, is is proposed to leave as much of these sensitive parts as possible to existing implementations that already have 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*. And combining b) and c) would result in significant redundancy, since both solutions have much overlap in the features they are providing, even though b) aims to be a framework for authorization. The process description from the beginning of this section will be 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 for it's permissions. And even these identifiers are obsolete when combining TLS with dedicated *consumer*-specific endpoint, 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 the specification 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 it's implementers [123] [124]. In the context of this project OAuth doesn't really match with the rest of the

design aspects.

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

When obtaining personal data, at the same time preventing those data from leakage is almost impossible, because of 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 mechanism. At the same time it requires also 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, thus introduces new attack scenarios. For now the only approach being considered, is the *Supervised Code Execution*, aside from the default 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*, gains 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 with 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 anonymize certain types of data somehow, if it's capable of doing so all, because the *consumer* at least supposedly knows what individual is behind that *PDaaS* it currently interacts with. The field for *data anonymization* is a large research area on its own, which recently started to gains 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* as well as for previously introduces scenarios.

First of all, to address the need of portability, which has to be satisfied by those components, that 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 into other platforms and environments. Thereby the *personal data storage* itself becomes platform-agnostic from an outside view, 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 it's 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 example below gives 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.

Code 01: Example query in SPARQL:

```
# query 1: obtain the data subject's first and last name
```

```
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"
                }
            }
```

```
        ]
    }
}
```

Without going into much details here, the syntax of this example (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 supported as well. The reason for performing two queries instead of just one is, because otherwise the result might have returned multiple "rows" with redundant data, if more then one bank account would have supported online payment; varying in three columns containing data about bank accounts though, but being identical in the fields related to the profile information.

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"
        }
    ]
}
```

Whereas comparing the *GraphQL* query syntax (Code 03) with it's 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 it's functions and logic was 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. 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 it's syntax. *SPARQL* is able to notice implicit relations between data points, though it's query language is not capable of grabbing and presenting them. Thus the result 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 develop and provide schemas for their requests as well. 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 have the ability to model 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, which are based on common standards, for example *medical record* [125], *point of interest* [126] or *bank transfer* [127]. With this approach those data types can be viewed as something like a plugin or add-on for the *PDaaS*.

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* include a *data subject's* profile (Code 05), a contact (Code 06) and bare position (Code 07). All those examples and lists are only meant as a starting point that should cover basic scenarios as well 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
- Organisation
- Date
- TimeRange
- Language
- Diseases

NOTICE: schema notation is based on the rules underpinning the schema definition provided by the SimpleSchema project [128].

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
}
```

List 02: relevant (sub-)categories of data

- Finance
  - Income
  - Bank transfers
- Shopping history
- 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* [129] and *GraphQL* [130]. 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 point 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, therefore they should be enforced. Whereas another know concept in data modelling, 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 persist 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 the question 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), which is oddly not marked in the Table 5.1. That's because of the priorly introduced concept of abstracting the *personal data storage* with a additional query language. This makes the access to the *PDS* 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 differently accessed in multiple ways.

It's only relates to the query language abstraction. Data and especially it's 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 used for horizontal scaling, federation and backups (S.P.05). Here it is focused 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 ensuring 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 could be to not only store the actual data written to the *PDS* but also to save all queries used to write that data in a chronological order. Therefore the current state can be restored just by running those writing queries against the *PDS*. It is reasonable to store the the queries from the abstracted query language not the ones the query language is transformed into. If a mobile device is part of the *PDaaS*, another approach would be for the *operator* could be to perform regular device backups. These are all just initial thoughts which might be sufficient only as a starting point. Other solutions are imaginable, but elaborating on those is beyond the scope of this work. Depending on what technologies 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. This might be no alternative, when it comes to configuration files certain technologies or key files, which are typically accessed as files. In any case, both have to be able to store files of any kind, which is required for instants to support the use case of medical records. File size restriction should be mandatory though. The *PDaaS* has no intention to replace existing *file hosting* solutions.

Being able to revert 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 those features. Therefore and because database systems with no alternative might not be able to provide this capability, it's not required by either the *PDS* or the *PL*. If it not natively supported but still desired, it needs to be considered if for example high frequently backups would already suffice or if a implementation on the application is required.

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

1. the *data subject* is provided with forms by the graphical user interface, which she is using 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.
2. the *data subject* is in possession of file(s) or string(s) that contain 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 elaborate on the context of those data.
3. Third party software, for example a browser plugin, is used to provide the *PDaaS* with data; in this case it's a browser history. This software uses a restricted API which is provided by the *PDaaS*, to let data flow into the system.

These three concepts, especially 2. and 3. are required to be inspected for malicious content and extensively validated against existing *structs*. Only if these are matching, the submitted data can be stored. In the second version the *data subject* need to be ask 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* clients, but without permissions to obtain data.

Conclusions: In order to gain flexibility in choosing technology and location for the *personal data storage*, the logical consequence is to abstract the interface to to 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 in a distributed system; *SPARQL* with it concepts of URIs as identifiers and resources, and *GraphQL* with it's separation of query definition and execution. This also has an effect on the process of query validation, which is much harder to do for *SPARQL*, because its syntax is more flexible and allows some shorthands. 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 which are embedded in the query syntax, in the output as well. Therefore *GraphQL* (and its implementations) is the query language of choice for this project.

Engaging a user community when it comes to creating new structs can compensate the lack of certain types. Examples for a potential start point of *PDaaS* supported data types were showed before. Data Modelling in general is a large research field for it's own. With regards to the *PDaaS* it needs much more work, though tt's beyond the scope of this document. 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 previous sections, their discussions and the requirements as well into account, this section serves the sole purpose of figuring out how all the different concepts and conclusions discussed before can fit together in an overall system architecture that is organized in either a distributed or a monolithic manner. This type of changes should not impact how the system's interfaces behave from a user perspective.

The foundation of this project is a server-client Architecture, which is chosen for a) providing availability (S.A.05) and b) separating some concerns [131]. 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 it's internals are taken into account. This results in the following three types of *platforms*:

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

Type	trusted	private	controlled by	Layer	Purpose
Mobile	no	conditionally	data subject	front end	• typically mobiles devices • based on host-specific native technologies • data storage

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

Web server *Platform:* Server

Purpose:

- 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)
- web client notification

Technologies:

- HTTP
- TLS
- WebSockets

Permission Manager *Platform:* Server

Purpose:

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

Technologies:

PKI *Platform:* Server

Purpose:

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

Technologies:

- X.509
- ACME [132] (Let's Encrypt)

Storage Connector *Platform:* Server

Purpose:

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

Technologies:

- driver for used database

Operator API *Platform:* Server

Purpose:

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

Technologies:

- JWT

Code Execution Environment *Platform:* Server

Purpose:

- 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

Technologies:

- Virtualization
- Container (OCI)

Tracker *Platform:* Server

Purpose:

- log all changes made with *Storage Connector*

- tracks states for ongoing consumer requests
- log all *access requests*

Technologies:

Personal Data Storage *Platform:* Server, Mobile

Purpose:

- stores the *operator's* personal data

Technologies:

- non relational database
- depending on host environment

Persistence Layer *Platform:* Server

Purpose:

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

Technologies:

- non relational database
- Filesystem

Notification Infrastructure *Platform:* Server

Purpose:

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

Technologies:

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

User Interface *Platform:* Desktop, Mobile

Purpose:

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

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 needs to be figured out which arrangements make sense and what variations might be possible. The result are two, more or less, distinct designs that are proposed. As stated above, one is a more monolithic approach and the other involve more platform types and demonstrates the flexibility.

The main difference between the two compositions is the lack of the mobile platform in the more monolithic approach (Figure 5.1). Although *monolithic* refers only to the components arrangement on the *server* platform. It is also imaginable that all server components not necessarily have to be placed into one server environment, but being distribute over several virtual machines or containers, so that they can scale and run more independently. This can improve *redundancy* as well.

In theory, a possible version of the arrangement would be to move all components to either the client or the mobile platform. This comes along with some downsides and major issues that are anything but trivial to solve. Aside from ensuring a nearly 100% uptime and localization in a landscape where NAT[3] and dynamic IPs are still common practice, not only on the mobile platform but on the client platform as well, all component, but the user

[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

interface, needs to be implemented with native technologies. Nevertheless, from a *operator's* perspective it would mean having all components at hand and therefore full control over the *PDaaS*, it still would lack of major requirements, though.

Aside from providing the *operator* with a non-stationary and instantly accessible interface to her *PDaaS*, involving a *mobile platform* 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, 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 *data* section. A mobile platform being part of the system makes it more easier for the *data subject* to establish a security concept, in which the relation the 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, or put it into the *personal data storage* as well or define it as an own storage component, in order to be flexible regarding it's location.

Authenticating *consumers* is performed based on TLS by the web server and it's 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 client on a mobile platform is considered *private*, it is reasonable to change the *operator* authentication from JWT-based to TLS-based *two-way authentication*, which would otherwise be inconvenient when using web-based clients.

If there are components that are only placed on the server and that have 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 be unidirectional only. 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 the other components via *HTTPS*.

The architecture implicitly distinguishes between two different groups of endpoints. These endpoints 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-services in a bidirectional manner. The group of endpoints for *data consumers* mainly consists of those were *access requests* and *permission requests* are coming in and the public one, that is used for *consumer* registrations. The other one is a small group of endpoints required for all the tools the *operator* might need; from data API or notification through 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 outsource that sensitive topic of user management by integrating third party authentication services, which not only makes is 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 her apps - a universal key so to say, but only one exemplar. So the downside here is, only a handful of third parties [133] 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 those addressed by the *PDaaS*; at least, when it comes to *data consumer* interactions. Although, 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 an external parties.

Conclusions: Considering amount of effort a single-platform version, namely client 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 environment with proper security measures, static IP and high availability. Even if that server is a local machine connected to the *data subject'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 comes with a higher level , because all the sensitive personal data somewhere on a computer machine. As of the proposed architecture all components (or group of components) are portable and therefore relocatable among the suggested platforms; and with the introduced authentication method for *operators* using multiple clients the for the same *PDaaS* are thereby supports and can be implemented with almost no effort, which covers more use cases. As a supplement, an *identity provider* based on the OpenID standard would fit nicely into the existing arrangement and not interfering with the other components. However, it is beyond the scope of this work to make 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 get taken and used as it suits everyone's needs; of cause, while respecting their licenses. Furthermore, enforcing *portability* 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 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

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

in *Java* (for Google's Android) or in Swift (for Apple's iOS). Whereas the environment systems has nearly no relevance for the *client*, other then the screen size and maybe which browser and version the environment system runs. But that's probably something the user can change.

As a result, being able to use certain components on a *server* platform depends on what *server* environment is provided. And vice versa, in order to decide on what implementation of a component is suitable, it's crucial in what environment that component has to run in. Either way, not to forget all the dependencies a component 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 technologies used, this concept is commonly known as *virtualization* or *containerization*. It isolates software by putting them into 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 also emulates a whole network infrastructure (e.g. DNS, TCP/IP, routing). Thereby, it is used to determine how certain container (and its containing component) are allowed to communicate and what resource are accessible from inside (e.g. filesystem). This complete abstraction to the surrounding environment means it quasi is 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* [106]. This concept can be utilized for the *supervised code execution* (S.A.01) mentioned before without any restraints.

Migrating from a server-located *personal data storage* to the *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* client application is installed.*

1. After starting the app, the *operator* needs to establish a connection between the server and mobile application. Therefor the *operator* either has to scans a QR-Code with the help of that app. THe QR-Code

is presented to the *operator* within her personal management interface of the *PDaaS* running in a browser. Or the *operator* inserts her credentials in to 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 just has been associated with the *PDaaS*.

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

    a) *[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. 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 the *[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 dump to the target environment and triggers the import process. Otherwise, the software not only needs to be aware of both database systems and their native query, it also has to have a comprehensive understanding of how their data structuring concepts work, in order to reliably transform one into the other. So to be more specific, at first the software has to analyse the structure of source database. Based on this result it might need to perform some configuration on the target database, before actually obtaining the data from the source database. The received data then need to be transformed into queries that are supported by the target system. Those transformed queries are transferred to the target environment, where those incoming queries get executed until all data is migrated.

4. After thr 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).

So to conclude, a migration process of 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 used 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 are 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 for that 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 accordingly. Aside from hardware on which the *PDaaS* needs to run, the only other requirement is owning a 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]Domain Name System; decentralized open directory that associates readable names with IP addresses

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 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 their appearance. In this way the user can 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 provides 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 lower level. Even though they provide more functionality and can 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, used to enable software developers program automated requests against those interfaces.

Table 5.3 provides a list of features with the associated user interfaces that should support them. It is notable that the *GUI* provides the *operator* with a powerful tool, which therefore needs reliable protection mechanisms (see Authentication). Whereas *API* capabilities are very limited, because it's the

one interface exposed 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 *client*	**X**		
authenticate *operator*	**X**		
migrate *personal data*	**X**	**X**	
review *permission requests* (P.I.04)	**X**		
create & maintain templates (P.I.05)	**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 *client* and *mobile* platforms. While prioritizing a web-based *GUI*, the management tool for the *operator* also need to be implemented natively 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 WebSocket-based connections add the same feature to *client* platforms. 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). Given the capabilities of, a inaccurate or error-prone rendered *GUI* can quickly cause unintended incidents. The main focus though has to

Known challenges for the *GUI* design is primarily to make very efficient but also fun ro review review *consumer registrations* and *permission requests*. Especially the latter

- when a data point is a Date or location, some kind of accuracy should be able to defined (in user interface)
- list all different features (w/ respect to the requirements) and give a suggestion on how a solution would look like; e.g. graph data structure with accordion menus/dropdowns

Conclusions:

Those are all vital characteristics that need to be addressed by the *specification*.

GUIs need to be provided for all *client* 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). *Client* platforms might use those provided *APIs* as well. In any case *APIs* must be separated according to the *roles*.

The most important aspect when interacting with something or someone is being provided with some kind of feedback. An action typically causes, and is therefore expected, a reaction. When combining both, the result is interaction.

The details of their implementations are not the concern of this specification, as long a every containing requirement is acknowledged.

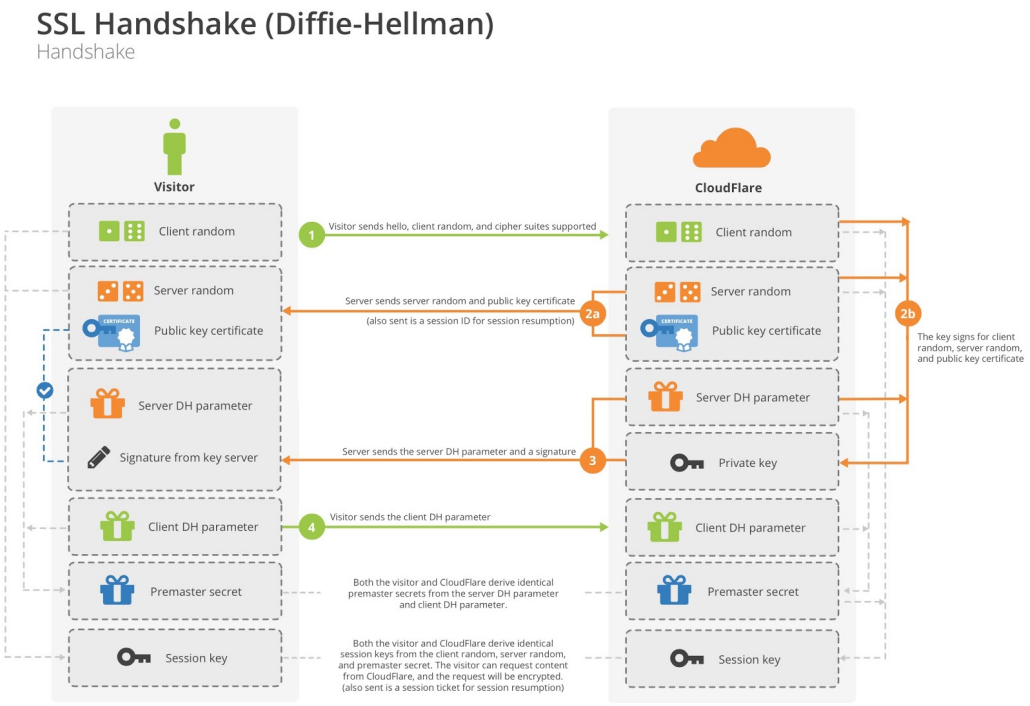

Figure 5.1: PDaaS Architecture, monolithic composition

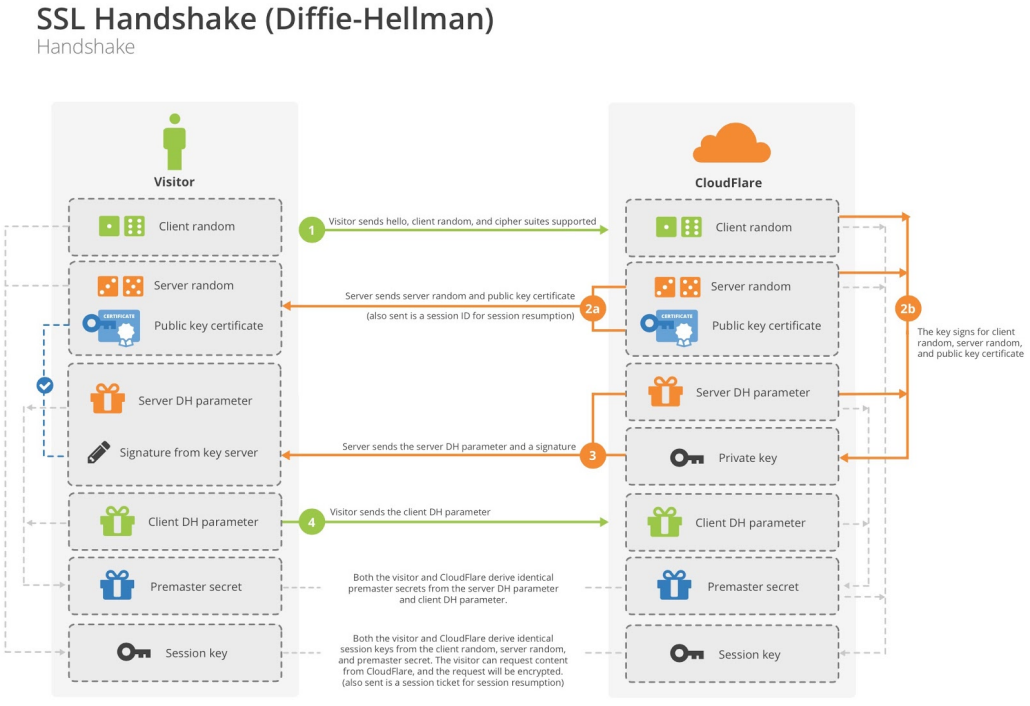

Figure 5.2: PDaaS Architecture, distributed composition

6

Specification *(Draft)*

This chapter hold the first draft of what might become a specification. As for now it has therefore no claim of completeness, continuity or accuracy. The contents is based on and a result of all previous discussions and developed solutions.

TODO: should might must n stuff in table (see dark mail spec)

6.1 Overview

- purpose
- architectural overview
- short description of the whole process

6.2 Components

6.2.1 Webserver

6.2.2 User Interface

6.2.3 Storage/Persistence

6.2.4 Notification Infrastructure

6.2.5 Data API

6.3 Data

6.3.1 Structure & Types

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

6.3.2 Read

6.3.3 Write

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

6.4 Protocols

Consumer registration

0) [OPTIONAL] *data subject* provides URI to *data consumers*

1) *data consumers* create *permission request* that includes

- X.509 based CSR[1]
- callback URI via HTTPS as feedback channel
- [OPTIONAL] information about what data points wanted to be accessed

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

Data Access

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

[1]Certificate signing request

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 TODO: 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)

Add or Change Personal Data

6.4.1 Data Management

- one third party access (consumer) relates to one access *endpoint*, that also authenticates that third party by TLS based *two-way auth*
- zero or more *permission profiles* are associated to one *endpoint*

6.5 APIs

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*

TODO: basic structure of a *permission profile*

How do the APIs involved with the protocols look like?

6.6 Security

- the downside of having not just parts of the personal data in different places (which is currently the common way to store), is in case of security breach, it would increase the possible damage by an exponential rate Thereby all data is exposed at once, instead of not just the parts which a single service has stored
- 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
- is the consumer able to call the access request URI repeatedly and any time? (meaning will this be stateless or stateful?)
- 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`

6.6.1 Environment

6.6.2 Transport

- communication between internal components *must* be done in https only, but which ciphers? eventually even http/2?

6.6.3 Storage

- documents based DB instead of Relational DBS, because of structure/model flexibility
- graphql because of it's nature to abstract a storage engine, which comes in handy when the actual storage gets relocated (e.g. from a server to a mobile device)

6.6.4 Authentication

- how should consumer authenticate?

6.7 Recommendations

6.7.1 Software Dependencies

6.7.2 Host Environment(s)

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.

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:

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 im-

peded. 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)

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.

Bibliography

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

[9] 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

[10] 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

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

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

[13] *ProjectVRM wiki - about VRM*. URL https://cyber.harvard.edu/projectvrm/Main_Page#About_VRM. - retrieved 2016-11-11

[14] *ProjectVRM wiki - list of personal information management systems*. URL https://cyber.harvard.edu/projectvrm/VRM_Development_Work#Personal_Information_Management_Systems_.28PIMS.29. - retrieved 2016-11-11

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

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

[17] *Outline of intellectual property*. URL https://en.wikipedia.org/w/index.php?title=Outline_of_intellectual_property&oldid=743830160. - retrieved 2016-12-25. — Page Version ID: 743830160

[18] Regulation (EU) 2016/679 — General data protection regulation, 2016

– scale

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

[20] Loeb), Ieuan Jolly (Loeb &: *PLC - data protection in the united states: Overview*. URL http://us.practicallaw.com/6-502-0467. - retrieved 2016-11-20

[21] 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

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

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

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

[25] *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

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

[27] Rosner, Gilad: Who owns your data? In: : ACM Press, 2014 — ISBN 978-1-4503-3047-3, pp. 623–628

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

[29] Regulation (EU) 2016/679 — General data protection regulation, 2016

– scale

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

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

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

[33] 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

[34] 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

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

[36] 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

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

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

[39] Tsai, Chun-Wei; Lai, Chin-Feng; Chao, Han-Chieh; Vasilakos, Athanasios V.: Big data analytics: A survey. In: *Journal of Big Data* vol. 2 (2015), Nr. 1, p. 21

[40] Zaïane, Osmar R: *Principles of knowledge discovery in databases*, 1999

– scale

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

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

[43] 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

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

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

[46] 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

[47] CEO), Tim Cook (Apple's: *A message to our customers. Customer letter*. URL http://www.apple.com/customer-letter/. - retrieved 2016-11-18

[48] 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

[49] 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

[50] *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

[51] Wikipedia: *Google traffic*. URL https://en.wikipedia.org/w/index.

php?title=Google_Traffic&oldid=746200591. - retrieved 2016-11-18. — Page Version ID: 746200591

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

[53] 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

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

[55] 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)

[56] Gaitho, Maryanne: *Applications of big data in 10 industry verticals*. URL https://www.simplilearn.com/big-data-applications-in-industries-article. - retrieved 2016-11-19

[57] 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

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

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

[60] 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

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

09

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

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

[64] *Kantara initiative – join. innovate. trust.* URL https://kantarainitiative.org/. - retrieved 2016-12-14

[65] 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

[66] 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

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

[68] 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

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

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

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

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

[73] 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.

[74] 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

[75] Belshe, Mike; Thomson, Martin; Peon, Roberto: *Hypertext transfer protocol version 2 (HTTP/2).* URL https://tools.ietf.org/html/rfc7540. - retrieved 2016-12-17

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

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

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

[79] 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

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

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

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

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

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

[85] Hildebrand, Joe; Jones, Michael: *JSON web encryption (JWE).*

URL https://tools.ietf.org/html/rfc7516. - retrieved 2016-12-17

[86] Bradley, John ; Sakimura, Nat ; Jones, Michael: *JSON web signature (JWS)*. URL https://tools.ietf.org/html/rfc7515. - retrieved 2016-12-17

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

[88] 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

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

[90] 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

[91] 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

[92] 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

[93] 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

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

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

12-19

[96] 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

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

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

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

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

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

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

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

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

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

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

[107] *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

[108] 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

[109] 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

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

[111] 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

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

[113] 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

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

[115] *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

[116] 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

[117] *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

[118] *Personalausweisportal - eID-server.* URL https://personalausweisportal.

de/DE/Wirtschaft/Technik/eID-Server/eID-Server_node.html. - retrieved 2017-01-06

[119] 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

[120] 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

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

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

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

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

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

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

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

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

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

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

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

[131] *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

[132] 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

[133] 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