Project no. 826278

SERUMS

Research & Innovation Action (RIA)
SECURING MEDICAL DATA IN SMART PATIENT-CENTRIC HEALTHCARE SYSTEMS

Report on Final Specification of Smart Patient Health Record Format D2.5

Due date of deliverable: 31st December 2020

Start date of project: January 1^{st}, 2019

Type: Deliverable
WP number: WP2

Responsible institution: Sopra-Steria Ltd.
Editor and editor's address: Euan Blackledge, Sopra-Steria Ltd.

Version 1.0

Project co-funded by the European Commission within the Horizon 2020 Programme		
Dissemination Level		
PU	Public	√
PP	Restricted to other programme participants (including the Commission Services)	
RE	Restricted to a group specified by the consortium (including the Commission Services)	
CO	Confidential, only for members of the consortium (including the Commission Services)	

Change Log

Rev.	Date	Who	Site	What
1	31/12/20	E Blackledge	SOPRA	Version 001.000

Executive Summary

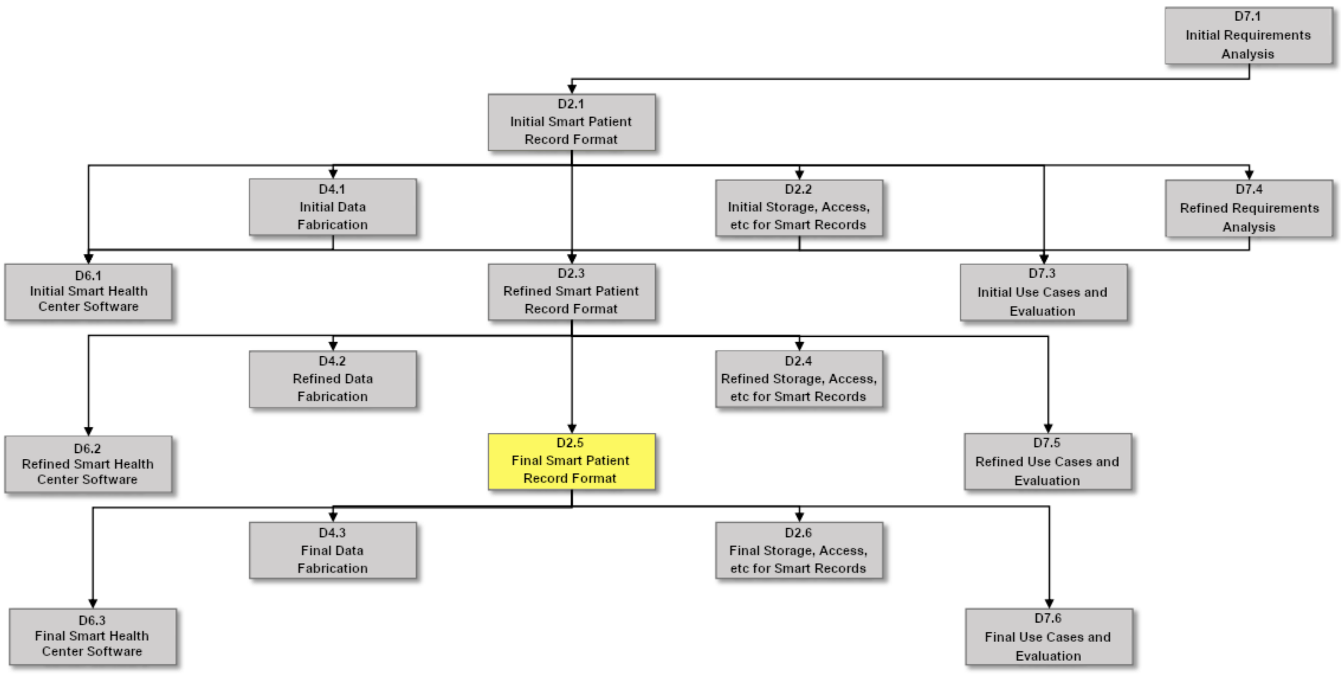

D7.1
Initial Requirements Analysis
D2.1
Initial Smart Patient Record Format
D4.1
Initial Data Fabrication
D2.2
Initial Storage, Access, etc for Smart Records
D7.4
Refined Requirements Analysis
D6.1
Initial Smart Health Center Software
D2.3
Refined Smart Patient Record Format
D7.3
Initial Use Cases and Evaluation
D4.2
Refined Data Fabrication
D2.4
Refined Storage, Access, etc for Smart Records
D6.2
Refined Smart Health Center Software
D2.5
Final Smart Patient Record Format
D7.5
Refined Use Cases and Evaluation
D4.3
Final Data Fabrication
D2.6
Final Storage, Access, etc for Smart Records
D6.3
Final Smart Health Center Software
D7.6
Final Use Cases and Evaluation

Figure 1: Dependencies between D2.5 and other deliverables

Serums proposes a unified **Smart Patient Health Record** that is capable of both securely storing data from any healthcare provider in a consistent manner, as well as the secure transmission of this data to only approved healthcare providers. Underlying this is the need for the patient to control who has access to their data, whilst still complying with all relevant legislation.

The **Serums** solution enables a patient to apply their consent to their own healthcare data using a range of role-base data sharing agreements via smart contracts stored on an identity blockchain. Any interactions with the system is stored for audit on the blockchain.

The system supports the future requirements for European wide healthcare by enabling the citizens to control their own healthcare data.

Contents

List of Abbreviations

D	Deliverable
FCRB	Fundació Clínic per a la Recerca Biomèdica
GDPR	General Data Protection Regulation [1]
M	Month
SPHR	Smart Patient Health Record
TPOLE	Time, Person, Object, Location, Event
USTAN	University of St. Andrews
WP	Work Package
ZMC	Zuyderland Medisch Centrum

Chapter 1

Introduction

Deliverable 2.5 is the *final specification* of the **Smart Patient Health Record** that represents a central core for the information about a particular patient in the smart health centre system, including both *static* personal data such as age, height, weight, date of birth etc. that is unlikely to change, and *dynamic* personal data such as the treatments the patient is undertaking, screenings, and prescribed medications[1]. The main challenge for the **Serums** project is to deal with *decentralisation* of the information related to a particular patient. In modern healthcare systems, the data about a single patient may reside on different subsystems in the same healthcare centre, or even, in some scenarios, scattered across different healthcare institutions. It will also be collected from a variety of devices, some of which (e.g. personal monitoring devices) will need to communicate with the healthcare systems over open networks. We need to be able to represent all this data in a standardised way, taking into account the possible geographical distribution of both where the data is stored and where is it collected from.

It is essential to derive a standard, precise, and machine readable format of the data related to a single patient. This also includes all the metadata associated with the data (e.g. whether the data is local or remote, how it is accessed etc). The SPHR also needs to provide the possibility of different types of access rights for different entities, such as patients, general practitioners, specialists, etc. who will be accessing the data. Furthermore, the SPHR obeys the fundamental data protection governance set out by international, national, and institutional guidelines.

Ideally, we want to derive a format that will cover all of the use cases used in the **Serums** project. This would allow generic distributed data analytics mechanism (an objective of WP3) to be performed on the medical data. Additionally, it would allow successful data fabrication (an objective of WP4), giving us access to a vast amount of *realistic* data that will have the same format as the real data (in

[1]While we talk about *static* and *dynamic* data being contained within the SPHR, it is worth clarifying that all of the data that goes into a SPHR is selected in real-time from the source systems. As such, a SPHR itself could be thought of as being made up entirely of *dynamic* data since any changes that are made by the healthcare provider to a patient's data are immediately reflected the next time a SPHR is requested.

terms of the fields used, ranges and distribution of values in the fields and correlation between different fields), but will be synthetic, without the possibility of being associated with any real data, and therefore not being subject of privacy and ownership concerns. This data will be used throughout the project both for developing new technologies and for stress-testing them on large volumes of data. Finally, the precise format of the data is essential for storage and access mechanisms.

This deliverable represents the final step in deriving a uniform **Smart Patient Health Record** format. We are now presenting a solution that is capable of supporting distributed data across multiple healthcare providers and capturing the metadata associated with the collection, transformation, transmission, and destruction of the data during the life cycle of a SPHR.

Chapter 2

Smart Patient Health Record Format

2.1 Overview of the Smart Patient Health Record Format

The **Smart Patient Health Record** format that **Serums** proposes is just that: smart. Rather than a one-size-fits-all approach to data management, we provide a format that can continuously adjust to the needs of the patient and/or healthcare providers.

This is achieved by allowing the generation of unique SPHRs that contain only the relevant data for a given circumstance. What this looks like in a practical sense is that the SPHR that is generated for a patient's physiotherapist is likely to be very different to the SPHR that is generated for the same patient's dermatologist. Essentially, these two healthcare specialisations require different subsets of the available data to provide their care effectively and **Serums** has enabled an easy way to design and deliver these in a secure, transparent way.

The way this is achieved is twofold: the application of *rules* to a patient's data (Section 2.2) and the way we structure the resulting data set (Section 2.3). In very broad strokes, the *rules* act as filters for the data and the *structuring* technique allows for multiple data sets to be joined together, often from varying sources.

The final element that goes into making up the SPHR is the method by which we track the lineage and provenance of the data that it contains (Section 2.4). This *metadata* is stored on a blockchain external to the record itself, allowing for an immutable record of all of the processing that each element of a particular SPHR has undergone.

2.2 Giving Patients Control Over Their Data

We have been tasked with designing a health record format that not only respects the privacy desires of patients but also complies with international, national, *and* institutional legislation for data protection. To discuss how our solution accomplishes these requirements we will look at how we ensure regulatory compliance as well as discuss how our solution provides patients with the tools to control variable privacy levels for their medical data. However, before we look at the specifics of the **Serums** solution, it is worth quickly covering what the end results of these goals looks like. In particular, how the patient interacts with the tools that we provide.

2.2.1 How a Patient Designs Rules

The internal language within **Serums** around how a patient controls the privacy levels of their data is to say that they *design rules* for their data. This is essentially who can see what and for how long. In order to achieve this, we have designed a user-friendly method for allowing patients the ability to select subsets of their data and to either grant or deny access for it to a particular health professional or group of health professionals[1].

Early in development it was envisioned that this control could be given to the patient in such a way that would allow atomic granularity over the data that they would be able to select for sharing. For example, a patient could make a rule that allowed one doctor access to a single field of a single table of their data for 5 minutes.

Whilst this would be possible, it was decided that this would be of little use to either the patients or the healthcare providers. As such, a new approach was chosen that breaks down the health data into sensible subsets[2] from which the patients can pick and mix when designing their *rules*. These subsets are known internally as *tags*. An example of how a patient interacts with these *tags* can be found in Figure 2.1. This depicts the rule creation screen of the **Serums Smart Health Centre System** (`https://shcs.serums.cs.st-andrews.ac.uk/`[3]).

2.2.2 How Tags are Designed

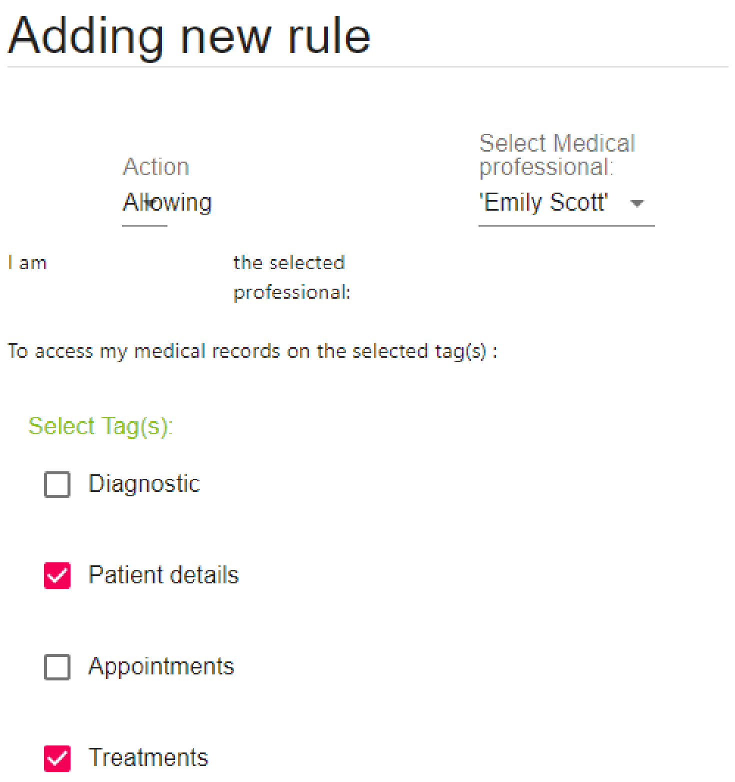
Our solution views the different regulatory requirements as a hierarchical framework that is applied to each SPHR. Since each layer of framework must work with and in no way contradict the underlying layers, we have aimed to keep this process as simple as possible with scalability and maintainability in mind.

To help visualize our approach, Figures 2.2, 2.3, and 2.4 should be referenced throughout this section. The foundation of the framework is GDPR. Since this

[1]While the current **Serums Smart Health Centre System** frontend has not implemented group control settings at the time of writing, the backend systems for controlling them have already been designed and implemented.

[2]The *sensible subsets* that have been chosen for **Serums** are very broad. If the technology were to be developed for use by healthcare providers, we would need to alter these subsets to be far more granular. For instance, a patient might want to only share details of a single treatment which they have received whereas the current subsets would return all of the treatments they have ever received.

[3]Link correct and working at time of writing.


Adding new rule

Action
Allowing

Select Medical professional:
'Emily Scott'

I am [Allowing] the selected professional:

To access my medical records on the selected tag(s) :

Select Tag(s):

- [ ] Diagnostic
- [✓] Patient details
- [ ] Appointments
- [✓] Treatments

Figure 2.1: The rule creation screen on the Serums website showing that available tags for the USTAN use case

law covers the rights of every EU citizen's personal data, it is vital that whatever we build must comply with this. Appendix A covers how WP2 addresses each of the seven principles of GDPR. In each of the Figures referenced above, GDPR makes up the largest outer box of the framework and is what everything else must fit within.

The next two layers of the framework cover both the regional laws regarding health data and each hospital's own guidelines for their patients' data. It is from these framework layers that the data content for each of the *tags* is derived.

We have designated the names for a set of 15 *tags*[4]. Working with the use case partners, we have created subsets of their data and categorised these subsets under the available *tag names*.

An example of the output of this process can be seen in Listing 1. Here we can see the various aspects that make up a *tag*. They are:

- tag: This is the *tag* under which the following data will be classified
- table: This is the name of the table within the source system that the data can be found
- fields: This is a list of one or more fields within the table that are to be classified under the *tag*
- key_lookup: This allows for instances where the source table does not contain the patient's id. Since this is the field we use to filter all data by, we must use a secondary table that contains both the patient's id and the primary key for the table we are currently trying to select. This is very much an edge case in the data sets we have been provided, however, with one such case existing in our data sets we have been sure to allow for this type of behavior.

There are three important things to note about the design process:

1. Each field of the available data has been captured under one or more of the *tags*
2. Multiple sources can be combined under a single *tag*
3. Not all *tags* have been used by all of the use case partners

If we use the Listing 1 as example of the *diagnostic tag* for **FCRB**, we can see that it falls under the categories points 1 and 2 from the above list. This is because this tag captures all of the fields from their source table "diagnostic" as well as combining the relevant fields from a second source table "episode". **FCRB**, however, does not have any data related to allergies in their current data set. As such, they have not modelled the *allergies tag* which addresses point 3.

[4]The complete list can be along with their descriptions can be found in Appendix B.

By using this common language to define the *tags* it enables the same *tag name* to be used by each of the use case partners to uniquely describe their data. Therefor, a patient can use the same *tag* across multiple different hospitals and know roughly what the underlying data is likely to contain. It also ensures that the hospitals themselves are in control over exactly what this data is. This prevents the system from being able to share data in a way that has not been authorised by the hospitals[5]. Should internal guidelines or data protection laws change, the hospitals would be able to adjust their *tag* definitions to capture these changes without the patient experiencing any change in how they interacted with the system.

A difference of how the same *tag* can be used in different ways can be found by comparing the Listings 1 and 2. As can be seen, both describe the *diagnostic tag*, however, they are made up of different tables and fields. This is due to the way that each of the use case partners stores their data. The results of these definitions are very similar data sets, however, a patient does not require prerequisite knowledge of how the data is held by each medical centre to make an informed decision when selecting data to share.

2.3 Data Vaults and How Serums Uses Them

As has been shown in Section 2.2.2, the hospitals themselves hold medical data in different ways. This is due in large part to the choice of database they use to store and organise their data in. Since there is no guarantee of consistency across these databases, it was important that we choose a common format for the SPHR. For this, we have chosen to use a **methodology for building databases** called **data vault**.

A data vault allows for new sources of data to be added without the need for complex redesigns of the existing data structures [4]. This is of massive benefit for **Serums**, and specifically the SPHR, as it allows more healthcare systems to be added, as well as the development of new use cases. An example of this benefit can be found in Appendix C. Our **Smart Patient Health Record** format can be seen as the **application of access rules to the construction of a data vault**.

2.3.1 Data Vault Basics

A data vault structure relies on three base type of tables: the Hubs, the Links, and the Satellites. The Hubs contain the business keys, the Links join the Hubs, and

[5]An issue that is beyond the scope for **Serums** but which is worth noting is that sometimes healthcare workers misuse or misunderstand the intention for fields within databases. This can lead to sensitive information being held in locations that were never intended to hold such information. It may even be the case that everyone within a healthcare centre knows that this is where that information is stored, however, unless this is communicated to the person(s) designing the *tags*, then it is likely that this data would be shared without the express knowledge of the patients. Clear communication would be vital between the healthcare workers and the system admins to ensure that misused fields like this are known and classified appropriately by their actual purpose.

Listing 1: An example of the information **FRCB's** *diagnostic tag* represents

```
[
    {
        "tag": "diagnostic",
        "table": "fcrb.diagnostic",
        "fields": [
            "einri", "patnr", "falnr", "pernr",
            "lfdnr", "dkey1"
        ],
        "key_lookup": {}
    },
    {
        "tag": "diagnostic",
        "table": "fcrb.episode",
        "fields": [
            "einri", "falnr", "patnr", "pernr", "bekat",
            "falar", "statu", "krzan", "storn", "casetx",
            "enddtx", "einzg", "fatnx"
        ],
        "key_lookup": {}
    }
]
```

Listing 2: An example of the information **ZMC's** *diagnostic tag* represents for comparison

```
[
    {
        "tag": "diagnostic",
        "table": "zmc.patient_diagnostic",
        "fields": [
            "patnr", "type", "name", "anatomical_location",
            "laterality", "begin_date", "end_date"
        ],
        "key_lookup": {}
    },
    {
       "tag": "diagnostic",
       "table": "zmc.patient_documents",
       "fields": [
            "patnr", "report_title", "department",
            "date", "content"
        ],
        "key_lookup": {}
    }
]
```

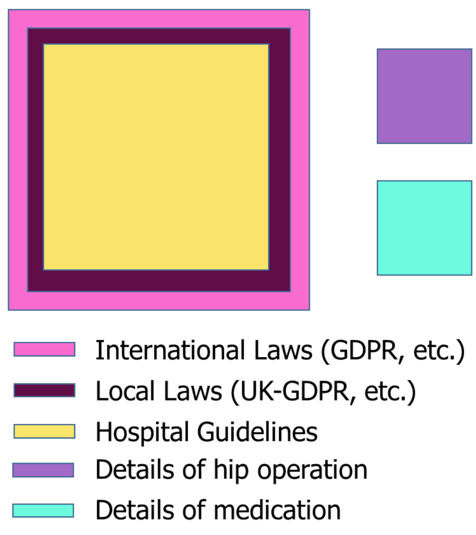


Figure 2.2: Depiction of the legal framework (left) and the two access rules a patient is going create (right)

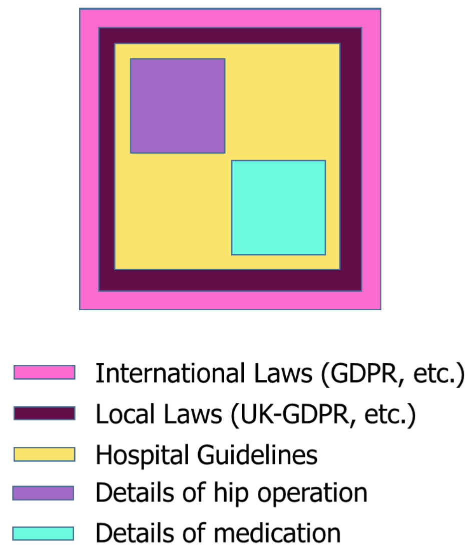


Figure 2.3: Depiction of a patient's access rules fitting within a legal framework

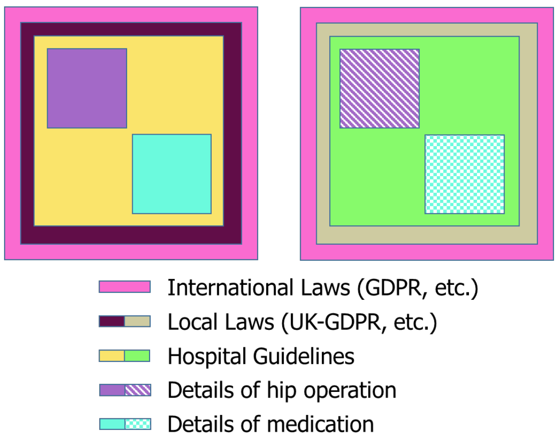


Figure 2.4: Depiction of the differing interpretations of the same access rules as generated within different legal frameworks

the Satellites contain the descriptive data. These three table types can be seen in Figure 2.5.

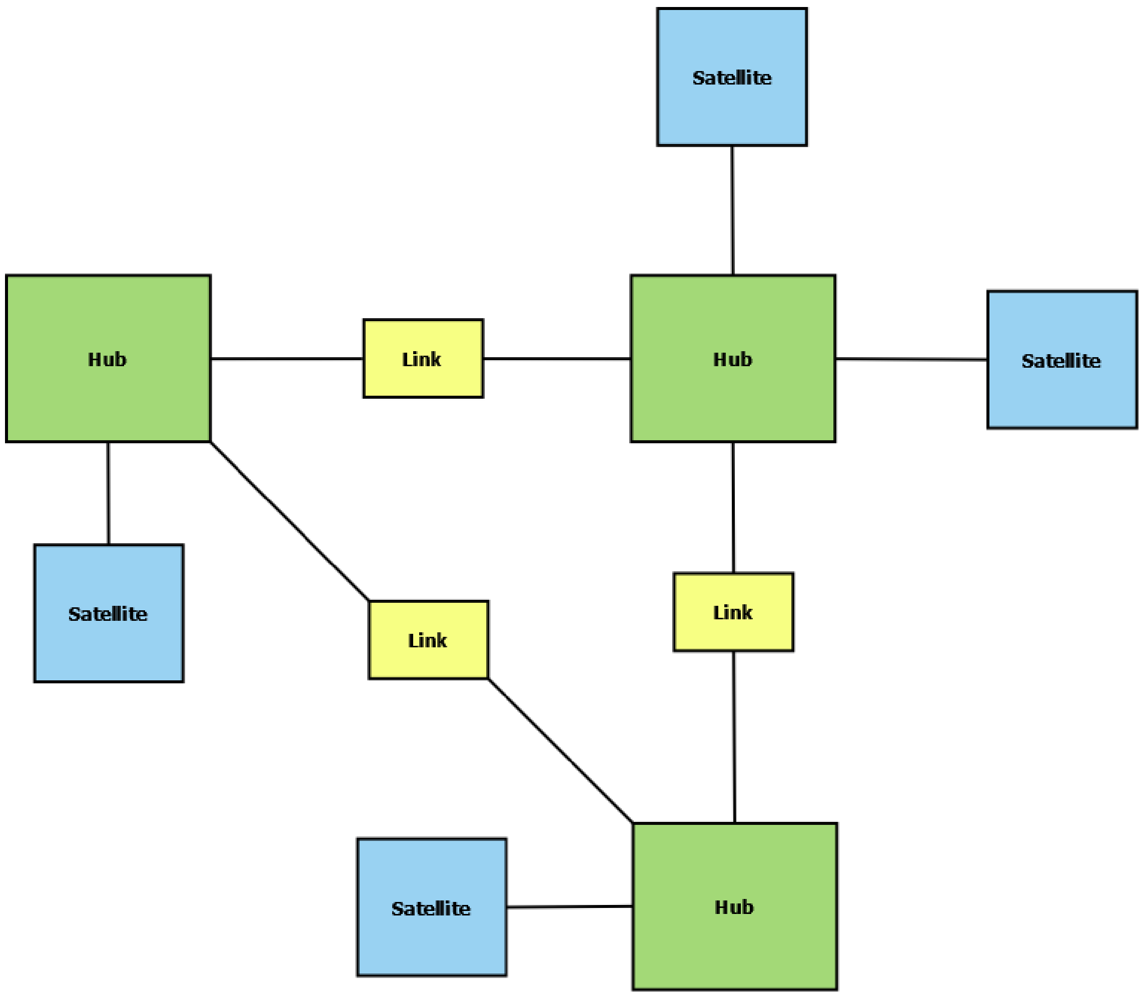
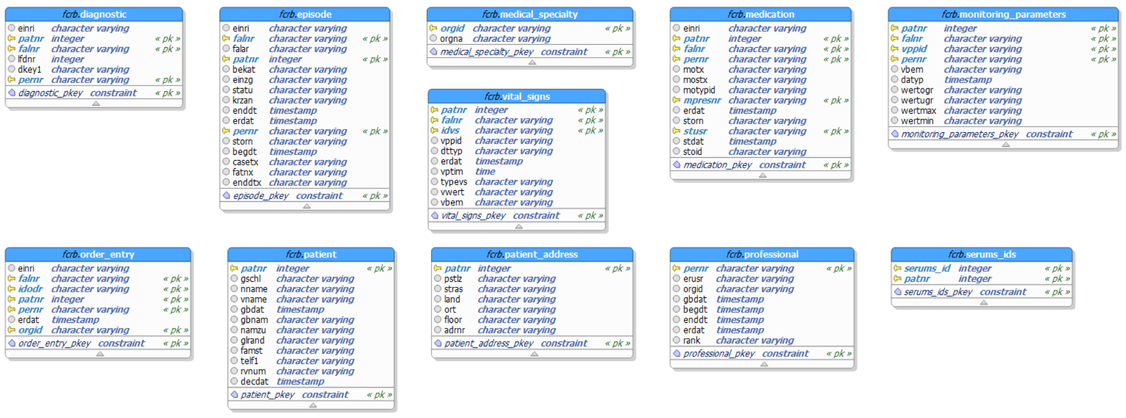


Figure 2.5: Depiction of the relationships between Hubs, Links, and Satellites

Our Hubs have been chosen as Time, Person, Object, Location, and Event. With these five categories, we are able to classify any incoming data (currently a manual process is required to determine which categories each piece of incoming data falls under, however, we are hopeful that some of this process can be automated by the delivery of D2.6 in month 34). Once the incoming data has been mapped to the data vault format we are able to automatically generate the SPHR on request, applying rules to the selection of data in the process (Section 2.2.1).

2.3.2 The construction of a Smart Patient Health Record

The following Figures demonstrate how the incoming data from one of our use case partners undergoes transformation from the source structure into the data vault form. It is worth mentioning here that we could select data from multiple sources which were never intended to be joined together. We would simply be storing



fcrb.diagnostic		
einri	character varying	
patnr	integer	« pk »
falnr	character varying	« pk »
lfdnr	integer	
dkey1	character varying	
pernr	character varying	« pk »
diagnostic_pkey	constraint	« pk »

fcrb.episode		
einri	character varying	
falnr	character varying	« pk »
falar	character varying	
patnr	integer	« pk »
bekat	character varying	
einzg	character varying	
statu	character varying	
krzan	character varying	
enddt	timestamp	
erdat	timestamp	
pernr	character varying	« pk »
storn	character varying	
begdt	timestamp	
casetx	character varying	
fatnx	character varying	
enddtx	character varying	
episode_pkey	constraint	« pk »

fcrb.medical_specialty		
orgid	character varying	« pk »
orgna	character varying	
medical_specialty_pkey	constraint	« pk »

fcrb.vital_signs		
patnr	integer	« pk »
falnr	character varying	« pk »
idvs	character varying	« pk »
vppid	character varying	
dttyp	character varying	
erdat	timestamp	
vptim	time	
typevs	character varying	
vwert	character varying	
vbem	character varying	
vital_signs_pkey	constraint	« pk »

fcrb.medication		
einri	character varying	
patnr	integer	« pk »
falnr	character varying	« pk »
pernr	character varying	« pk »
motx	character varying	
mostx	character varying	
motypid	character varying	
mpresnr	character varying	« pk »
erdat	timestamp	
storn	character varying	
stusr	character varying	« pk »
stdat	timestamp	
stoid	character varying	
medication_pkey	constraint	« pk »

fcrb.monitoring_parameters		
patnr	integer	« pk »
falnr	character varying	« pk »
vppid	character varying	« pk »
pernr	character varying	« pk »
vbem	character varying	
datyp	timestamp	
wertogr	character varying	
wertugr	character varying	
wertmax	character varying	
wertmin	character varying	
monitoring_parameters_pkey	constraint	« pk »

fcrb.order_entry		
einri	character varying	
falnr	character varying	« pk »
idodr	character varying	« pk »
patnr	integer	« pk »
pernr	character varying	« pk »
erdat	timestamp	
orgid	character varying	« pk »
order_entry_pkey	constraint	« pk »

fcrb.patient		
patnr	integer	« pk »
gschl	character varying	
nname	character varying	
vname	character varying	
gbdat	timestamp	
gbnam	character varying	
namzu	character varying	
glrand	character varying	
famst	character varying	
telf1	character varying	
rvnum	character varying	
decdat	timestamp	
patient_pkey	constraint	« pk »

fcrb.patient_address		
patnr	integer	« pk »
pstlz	character varying	
stras	character varying	
land	character varying	
ort	character varying	
floor	character varying	
adrnr	character varying	
patient_address_pkey	constraint	« pk »

fcrb.professional		
pernr	character varying	« pk »
erusr	character varying	
orgid	character varying	
gbdat	timestamp	
begdt	timestamp	
enddt	timestamp	
erdat	timestamp	
rank	character varying	
professional_pkey	constraint	« pk »

fcrb.serums_ids		
serums_id	integer	« pk »
patnr	integer	« pk »
serums_ids_pkey	constraint	« pk »

Figure 2.6: The source tables for the FCRB use case

more business keys in the Hubs and attaching additional Satellites. The rest of the process appears identical.

Starting with the source data for FCRB (Figure 2.6) we can see how the data is structured when we make a request to access it. As was covered in Section 2.2.2, the actual data that is selected from the source tables is controlled by a set of tags which are designed by the hospitals and selected by the patients. For the following examples we will be using all of the data from these source tables.

In Figure 2.7 we see the start of every one of our data vaults. Here we have generated all of the Hub and Link tables and implemented the relationships between them. Internally, this structure is called the data vault boilerplate.

Figure 2.8 shows the next step of data vault generation. Here the business keys have been added to each of the corresponding Hubs.

The final step of data vault creation is the addition of the Satellites. These are simply attached to their corresponding Hubs. Figure 2.9 shows the complete data vault structure for FCRB's use case. Should a patient wish to share all of the data from FCRB's use case, this is what their SPHR would look like at the time of delivery.

2.4 Tracking Lineage

Since we will be selecting, filtering, transforming, and combining data from multiple sources during the creation of an SPHR, it is important that we record exactly which processing has taken place to a given piece of data and when. We could

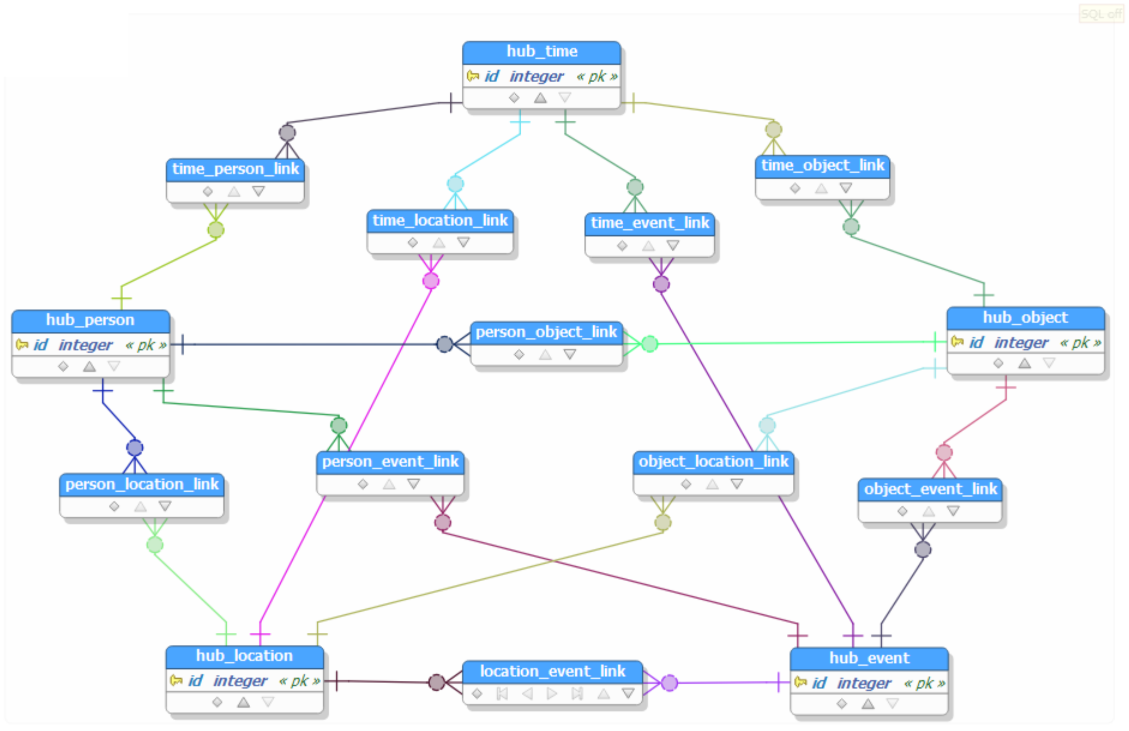


Figure 2.7: The boilerplate structure for our TPOLE data vault

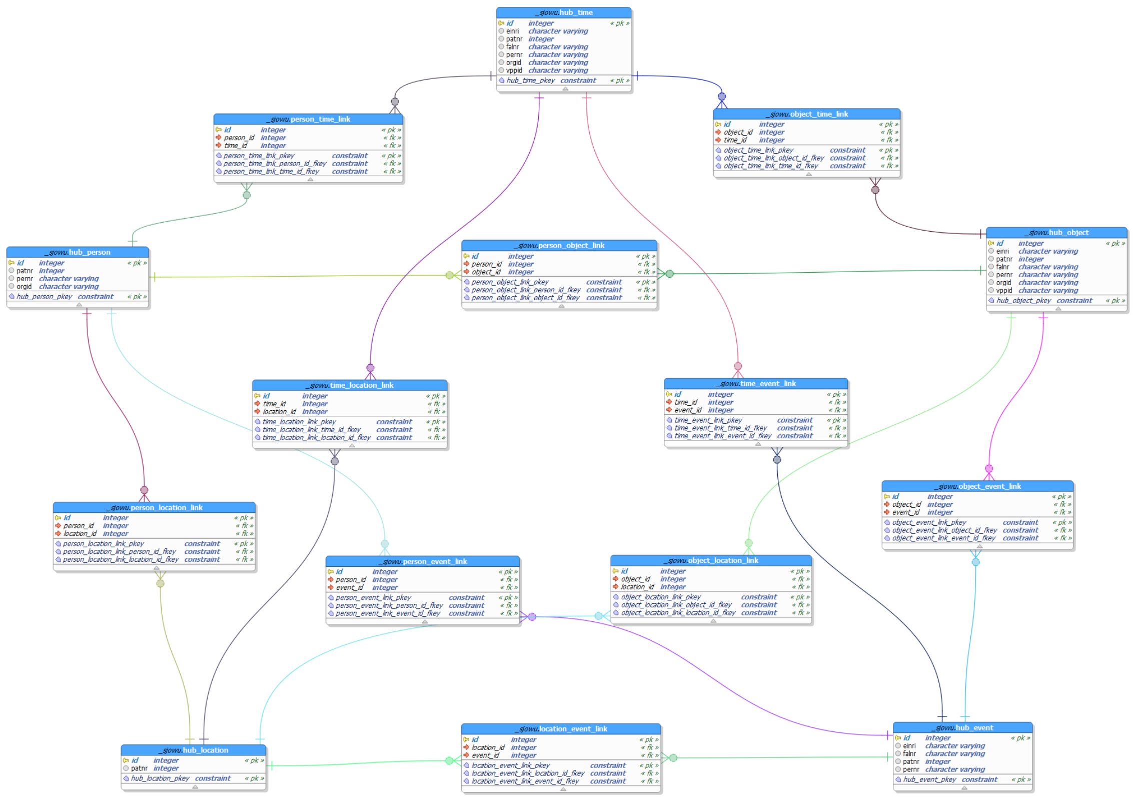


Figure 2.8: The Hubs filled with FCRB's business keys

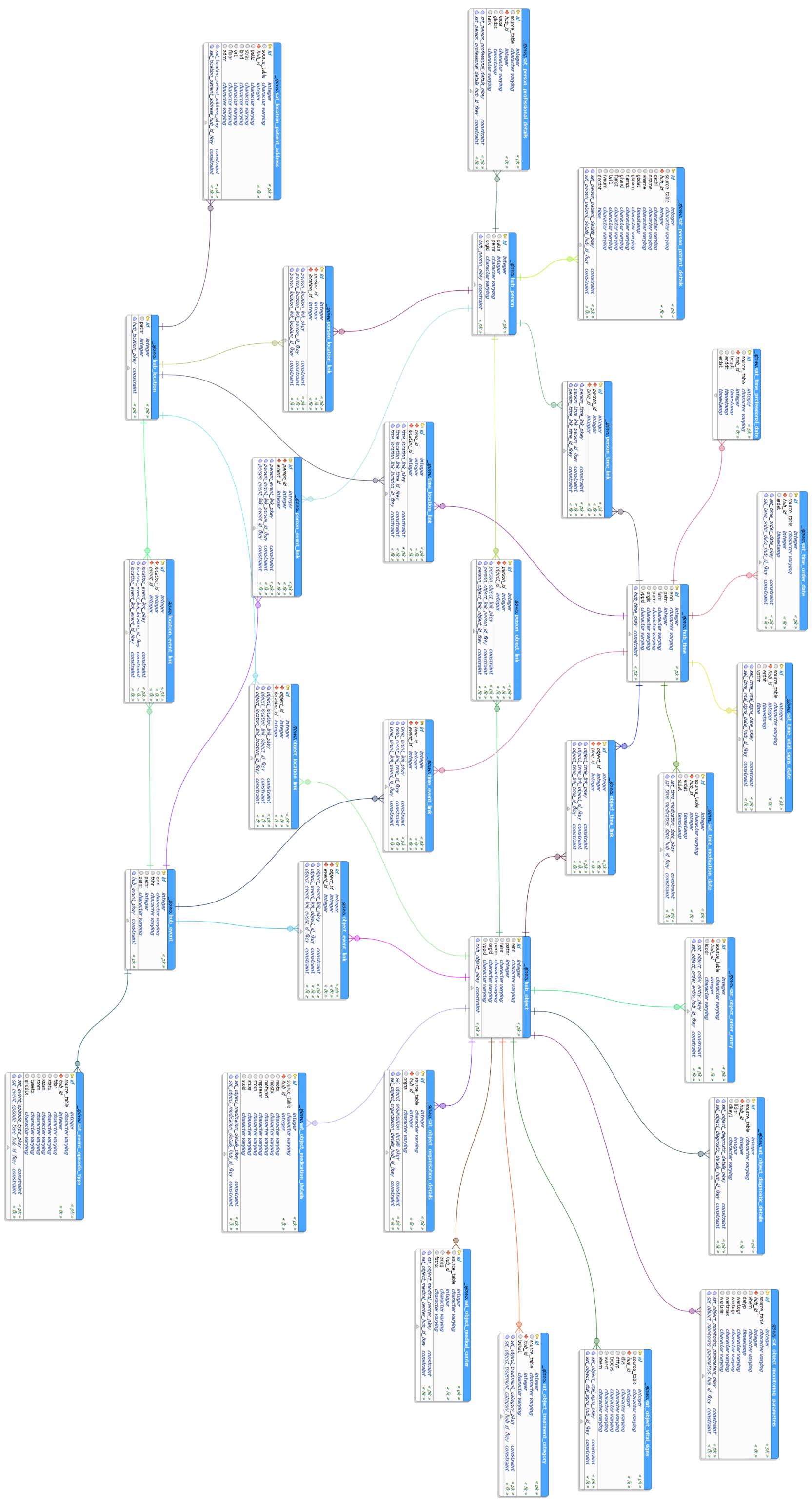


Figure 2.9: The complete data vault structure for FCRB's data

store this *metadata* within the records themselves, adding it to the Hubs of the data vault structure. This would allow each Satellite to have a clear record of where the data it contains has come from and what has happened to it along the journey.

The issue with this approach, however, is that the SPHRs themselves have a finite life. Since they are created in real-time at the time of request and are deleted from the **Serums** backend system shortly after transmission, it is highly unlikely that the *metadata* for a particular SPHR would be available for audit at a later date. As such, we have chosen to use a blockchain to store this information.

2.4.1 A (Very) High Level Overview of Blockchain

As a little bit of background, a blockchain is a special type of database that distributes copies of itself across a network of *nodes*. With each *node* holding a copy of the data, responsibility for the accuracy of the data the blockchain holds is shared. As such, there is no one single owner of the data or indeed one single point of failure[6].

2.4.2 The Provenance and Lineage Blockchain

As detailed in Section 2.3.2, data is processed in multiple different ways before it is transmitted as an SPHR. Since we want to record what these processes are and their results in a way that can be audited, our blockchain has been designed in such a way that each SPHR has a single entry on the blockchain that is updated as each process reaches its conclusion.

An important thing to note is that no medical data will be stored on the blockchain itself. As such, care has been taken in deciding what should be stored that would allow a clear understanding of the construction of the SPHR without revealing anything about the contents of the record itself.

The way that the process works is as follows:

1. When a request for a SPHR is made, the blockchain is notified and a new record is started
2. The blockchain will return an ID for the newly created record. It is this ID that acts as a fingerprint for the SPHR and will be used throughout the process to update the same record. This ID will be stored in the Hubs of the SPHR
3. Every time a process is undertaken, the action and the results are added to the record on the blockchain. Examples of what is stored include:
    - The rule (Section 2.2.1) that is being applied to the patient's data

[6]While an in depth look at the inner workings of a blockchain is beyond the scope of this report, an excellent graphical explanation of this entire process can be found at `http://thesecretlivesofdata.com/raft/`.

    - The time a process was initiated and completed
    - The source system that a piece of data was selected from
    - The type of transformations that occurred
    - A hash of the data vault schema that was generated. Since the same schema should be generated every time that a particular rule is applied to a particular patient's data, the hashes can be replicated without the need to access the data
    - The location(s) of any copies of data that has been made during the processing

4. Once the SPHR has been transmitted, a cleanup process is triggered that deletes anything that has been generated in the process of making it. The confirmation of this deletion is the final entry that will be recorded to the blockchain

With this, we have an immutable record of everything that took place during the creation of the SPHR. The blockchain is able to be searched by system admins for all of the records of a particular user. This information can then be used to check that the correct data was selected from the correct sources, the rules were applied appropriately, and that the resulting data has been properly disposed of.

3. Conclusion

The current **Smart Patient Health Record** is version 001.003 and is the final format for the **Serums** project.

It is capable of transporting all of the data for each of the use cases. Furthermore, it is capable of adapting to the privacy desires of users of the system whilst ensuring that it remains compliant with all data governance laws and guidelines. We have also ensured that we maintain a permanent record of the provenance and lineage for the data each record contains, without exposing any of the sensitive data within it.

Bibliography

[1] Information Commissioner's Office. Guide to the General Data Protection Regulation (GDPR). Technical report, Information Commissioners Office, 2018.

[2] V Janjic, J K F Bowles, A F Vermeulen, A Silvina, and E Blackledge. The SERUMS tool-chain : Ensuring Security and Privacy of Medical Data in Smart Patient-Centric Healthcare Systems. In *2019 IEEE International Conference on Big Data (Big Data)*, pages 2726–2735. IEEE, 2019.

[3] Dan Linstedt and Michael Olschimke. *Building a Scalable Data Warehouse with Data Vault 2.0.* 2015.

[4] Daniel Linstedt and Michael Olschimke. Introduction to Data Warehousing. In *Data Vault 2.0*. 2016.

Appendices

Appendix A

The Seven Principles of GDPR and how Serums Addresses them

Lawfulness, Fairness, and Transparency

We achieve this through the use of rules (Section 2.2.1). By working with GDPR by design and ensuring that the data is compliant with any needs of each use case partner, we guarantee that the data meets legal requirements. Furthermore, by giving control to the patients over who can see what, we ensure they are aware of what the data will be used for. Finally, by tracking exactly how a SPHR is created, we allow a fully transparent record for each instance of the SPHR without revealing any of the sensitive information held within.

Purpose Limitation

Since each SPHR is generated for a specific purpose, we allow the users to set exactly what this purpose is. They may generate multiple different SPHRs for multiple different purposes.

Data Minimisation

The data that makes up the SPHR is filtered at the point of selection based on the rules. This way, only relevant data can possibly be included in the final SPHR.

Accuracy

A SPHR is generated in real-time at the time of request. The source for the data is the use case partner's own data store. As such, the SPHR is as accurate as the data in the hospital's own system. If there is an issue with the data, the patient would inform their hospital and as soon as the data is updated there, any new request for a SPHR would reflect the change in data.

Storage Limitation

Currently each SPHR is very short lived. It is created at time of request and any remnants of it are cleared up after transmission. The record of its life is stored on a purpose built blockchain that records everything that took place in the creation and deletion of the record, without retaining any identifiable data beyond their user id. This user id allows audits to be carried out on SPHRs that have been generated with their data.

Integrity and Confidentiality (Security)

We have worked hard across multiple work packages to reduce the risk of unauthorised access to a patient's data. Specific to the SPHR, before transmission, it is encrypted with a one time set of keys. This means that even upon a successful interception of multiple SPHRs, a bad actor would have to break the encryption for each one separately and the result would typically be a small subset of data. This greatly minimises the risk to each patient.

Accountability

We have worked hard to develop tools that allow us to work in as transparent way as possible. This enables almost every aspect of the project to be audited in a way that respects the privacy of users.

Appendix B

The complete list of tags

The following is the current list of tags which have been prescribed to the use case partners of **Serums**. These were designed to cover all of the data for all of the use cases, with every field of every source table falling into at least one of them. With this list being a relatively new aspect of the project, there is a chance that it will be altered in some way before the final submission (D2.6 - M34).

Personal

A very limited subset of data that usually only contains the patient's name, sex, and key measurements

Patient Address

Limited to only the patient's address

Patient Details

An expanded set of data to that of the *personal* tag that also contains the details found in *patient address*. Additional fields might also contain things such as telephone number, marital status, nationality, etc.

Patient Appointments

Any data related to appointments such as the date, the reason for the appointment, and the name or id of the healthcare professional

Wearable

Any data which is generated by a wearable medical device. Both ZMC and FCRB have examples of this technology in their use cases

Diagnostic

Any data related to a patient's diagnosis. This covers a range of sources for all three use case partners that includes medical reports, specific data related to the location of the condition, the dates related to the condition's start and end, etc.

Medication

Any data related to medication the patient has been prescribed as part of their treatment

Operations

Any data related to an operation the patient has undertaken such as the date, the reason, and the outcome

Documents

Any documents connected to a patient such as doctors' notes or referrals

Treatments

Any data related to the treatments the patient is under going. This can include elements found in the *medication*, *operations*, and *documents* tags

Healthcare Providers

Details of the healthcare providers. This includes elements such as name or id, department, and speciality

Drugs and Alcohol

Any data related to substance usage

Allergies

Any data related to allergies

Additional Information

Currently used as a catch all to cover edge cases which do not yet fit into another tag. Currently the data which is covered under this tag includes additional details about living conditions and notes about hearing difficulty

All

Mostly used as a debug tool as it returns all of the data for a patient

Appendix C

Problems Data Vaults Solve

One of the key issues we are aiming to solve with the Smart Patient Health Record is the combination of sets of data into a single source that can be easily and securely transmitted. For healthcare providers who share the same systems, such as those who use SAP, this would be a relatively painless exercise. However, even within our small set of use case partners, there are key differences between the systems which we are looking to model. The following set of figures is an illustrative example of the benefits of using data vault as the backbone for our SPHR.

For our example, we start with three tables as depicted in Figure C.1. These tables are from a fictitious hospital and are for the patients, the doctors, and the treatments of the patients.

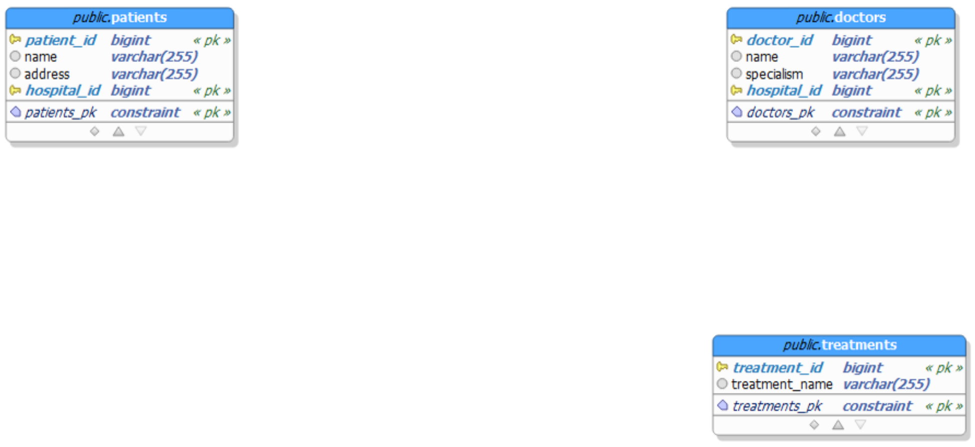
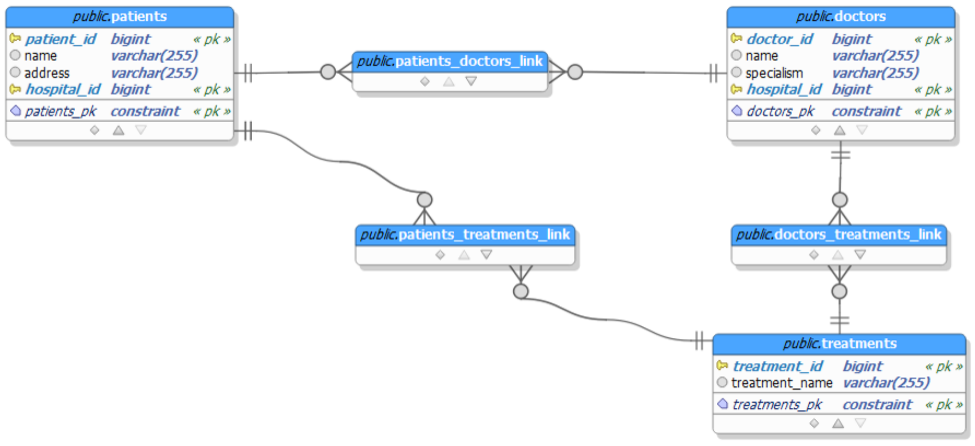

public.patients
patient_id bigint « pk »
name varchar(255)
address varchar(255)
hospital_id bigint « pk »
patients_pk constraint « pk »
public.doctors
doctor_id bigint « pk »
name varchar(255)
specialism varchar(255)
hospital_id bigint « pk »
doctors_pk constraint « pk »
public.treatments
treatment_id bigint « pk »
treatment_name varchar(255)
treatments_pk constraint « pk »

Figure C.1: Three tables which we intend to have relationships between

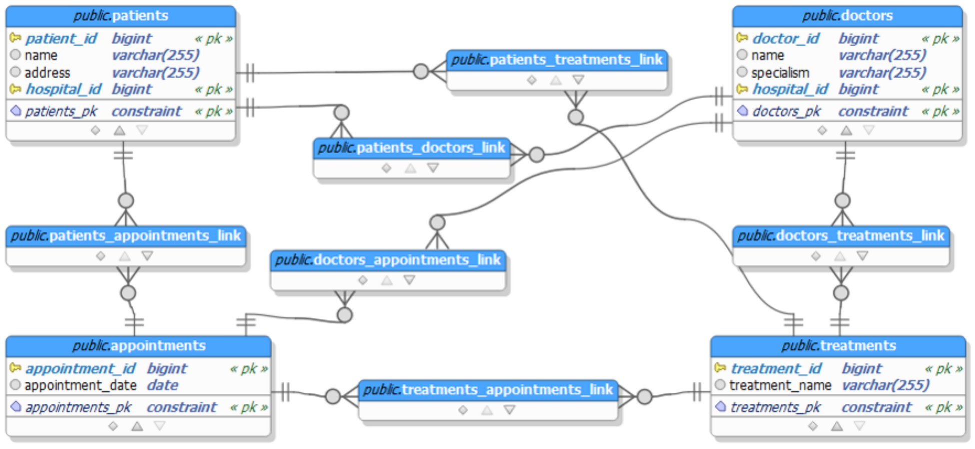
Our example hospital links these tables using the many-to-many relationship model. Specifically a patient can see many different doctors and the doctors can see many different patients. Additionally a doctor may prescribe different treatments to each of their different patients and a patient may receive many different treatments as prescribed by their many different doctors. Figure C.2 demonstrates the join tables that can be used to facilitate this type of relationship.



public.patients
patient_id bigint « pk »
name varchar(255)
address varchar(255)
hospital_id bigint « pk »
patients_pk constraint « pk »
public.patients_doctors_link
public.doctors
doctor_id bigint « pk »
name varchar(255)
specialism varchar(255)
hospital_id bigint « pk »
doctors_pk constraint « pk »
public.patients_treatments_link
public.doctors_treatments_link
public.treatments
treatment_id bigint « pk »
treatment_name varchar(255)
treatments_pk constraint « pk »

Figure C.2: The addition of join tables between our three start tables

As our example hospital continues to expand its IT capabilities it adds in the appointments booking system. There is a desire for the appointments system to maintain many-to-many relationships with all the existing tables to ensure flexibility in how the system can be used. This single new data source results in the need for three new join tables to be created and maintained as can be seen in Figure C.3 and is a demonstration of how rapidly scaling issues can be created.



public.patients
patient_id bigint « pk »
name varchar(255)
address varchar(255)
hospital_id bigint « pk »
patients_pk constraint « pk »
public.patients_treatments_link
public.doctors
doctor_id bigint « pk »
name varchar(255)
specialism varchar(255)
hospital_id bigint « pk »
doctors_pk constraint « pk »
public.patients_doctors_link
public.patients_appointments_link
public.doctors_appointments_link
public.doctors_treatments_link
public.appointments
appointment_id bigint « pk »
appointment_date date
appointments_pk constraint « pk »
public.treatments_appointments_link
public.treatments
treatment_id bigint « pk »
treatment_name varchar(255)
treatments_pk constraint « pk »

Figure C.3: A fourth table and its relationships added to the existing schema

The example here is extreme, however it provides insight into one of the core issues that we would encounter if we were to use standard database modelling techniques. What follows is the same example data set but this time it is modelled using our data vault technique.

An important step that is not covered here is the mapping of incoming data to its new structure. For each field of each source table we currently must know where it is going to end up. In other words, which Satellite it is going to form part of. This is covered in more depth in Appendix D. Simply put, we categorise each field of each incoming table under one of the Hub categories. Fields which are similar enough in scope are grouped together in Satellites. This does allow for single source tables to be split across multiple Hubs and in turn multiple Satellites.

As covered in Section 2.3.2 we always start with a boilerplate structure for our data vault, with five Hubs (Time, Person, Object, Location, and Event) with many-to-many relationships formed between each of them in the form of Links. Figure C.4 shows a simplified version of this, reducing the numbers of Links between the Hubs for readability. A complete version of this boilerplate structure can be found in Figure 2.7.

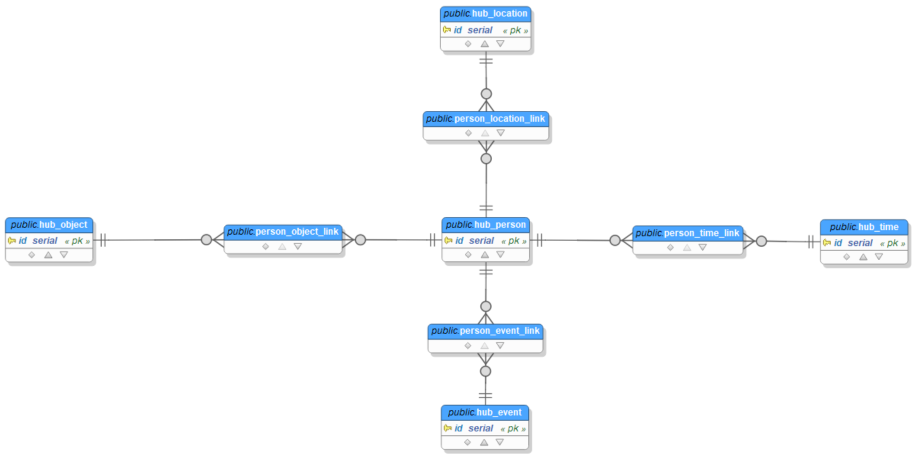


Figure C.4: A simplified data vault boilerplate

With the boilerplate ready, we can extract the business keys from the incoming data and insert them into the appropriate Hub(s) based on the destination Satellite(s) for each source table. Figure C.5 shows the results of this process as applied to our original three tables: patients, doctors, and treatments.

Note here that both the patient and doctor Satellites are attached to the Person Hub and that the treatments Satellite is attached to the Object Hub. The many to many relationships are still maintained in all of these circumstances.

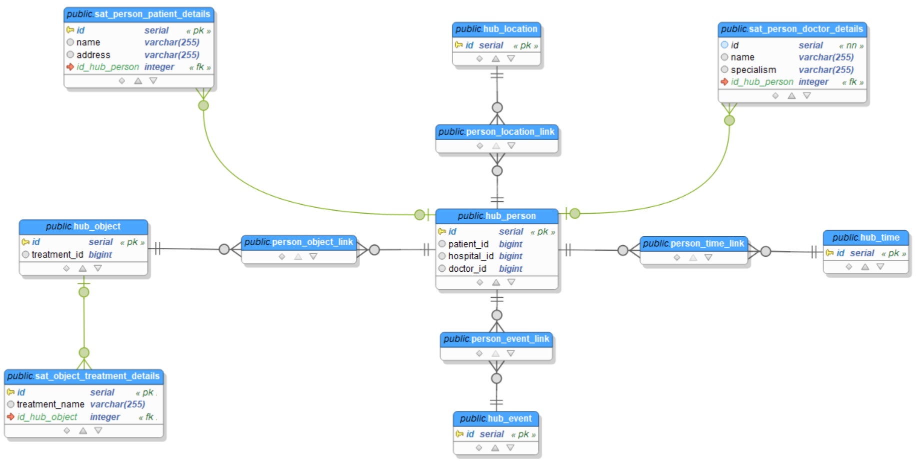


Figure C.5: Our original three tables attached to the data vault boilerplate

For the final step of this example we will add in the appointments table. This time it is worth noting that rather than requiring multiple changes to the overall schema of the database, we simply insert the business key (appointment id) into the Time Hub and attach the rest of the appointments table's data as a Satellite to the Time Hub. This can be seen in Figure C.6.

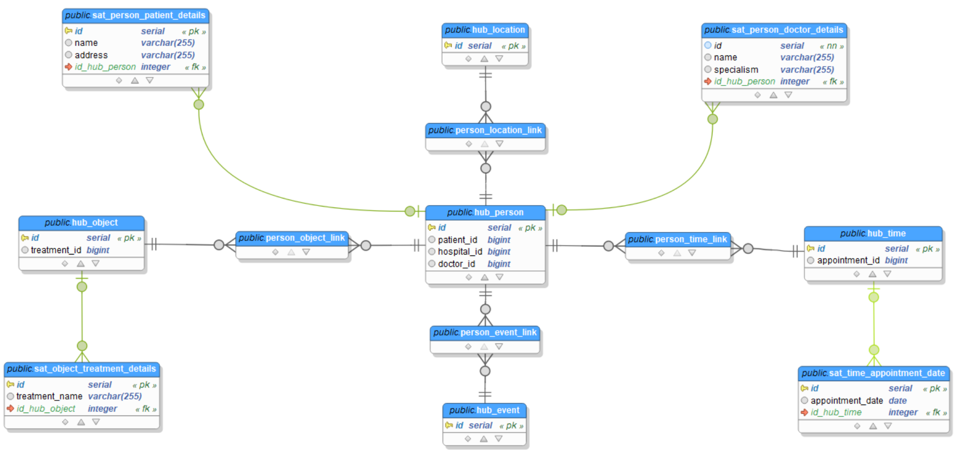


Figure C.6: The fourth table now in place on the data vault

By maintaining a consistent core structure to the data vault, we are able to rapidly implement changes to the incoming data as well as easily join together disparate data sets that were never intended to be joined in the first place.

Appendix D

Data Vault Design - Vault Bot

WP2's Smart Patient Health Records are generated in real time as a result of rules (Section 2.2.1) being applied to the source data which is then transformed in a data vault (Section 2.3.2). For this to run smoothly, we currently have to manually set the mapping of every source field to their destination Hub or Satellite (Appendix C).

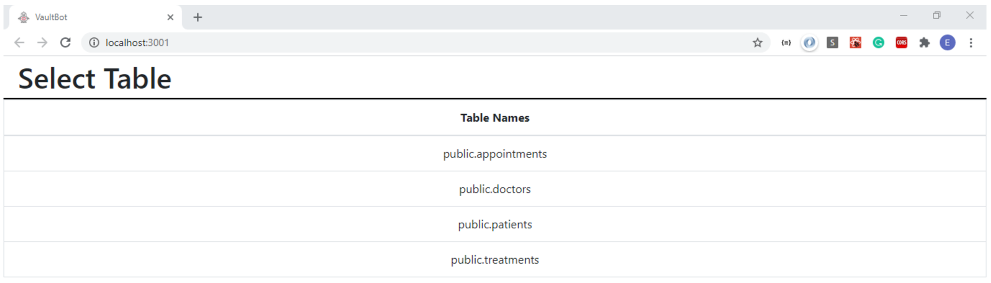
While the use cases for the **Serums** have largely stayed the same since the initial proposal, the data which the use case partners have wanted to include has changed repeatedly. With each data change comes the need to manually edit or create both the new code to reflect these changes in the data vault structure and the control files which are used to map the fields from the source to their destination. This was a time consuming effort that was rife with room for error when working with some of the larger data sets.

To improve this situation, **SOPRA** has developed tooling known as **Vault Bot** to increase both the speed and accuracy of any changes to the source data that are requested. This chapter will cover some of the functionality that it offers. It is still a very early build, however, it has already improved the speed of development and further updates to the tool will continue to be added.

The first major improvement is that there is a graphical interface which links directly to the source database. This can be seen in Figures D.1, D.2, and D.3 (for the purposes of this chapter we are continuing to use the example hospital as used in Appendix C). The advantage this gives is that the source table, the field names, and the field types are captured automatically so there is no room for human error.

A further example of improvement can be found on the table design screen as shown in Figures D.2 and D.3. Whilst not required for our example here, it is entirely possible to design multiple Satellites at the same time from the same table, such as when subsets of the table are categorised under different Hub types or broken down to be more atomic. This ensures that all the correct business keys are correct across the Hubs as well as guaranteeing that the Link tables are correctly utilized during the transfer of data from source to the data vault.

The final improvement is the auto-generation of the data vault schema and the



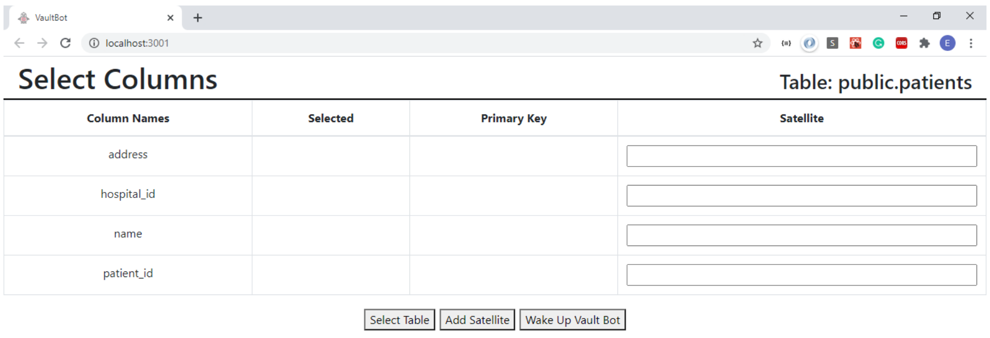
Select Table

Table Names
public.appointments
public.doctors
public.patients
public.treatments

Figure D.1: The table select screen for our fictitious hospital from Appendix C

control files which maps the source data onto its destination Hub or Satellite. The new schema is written as a series of SQLAlchemy classes and delivered as a file which can be ran in Python3 to both create the data vault and, if necessary, be used as a declarative class file for forming an Object-Relational Mapping (ORM). This allows interaction between the database and the outside world without the need for SQL, removing an otherwise potential avenue for misuse. Examples for the code for the Hubs, Links, and Satellites can be found in the Listings 3, 4, and 5 respectively.

The relevant control files are read by the API at the point at which the data is being copied from the source tables into the newly generated data vault. An example of the control file that is generated for the patients table can be found in the Listing 6.

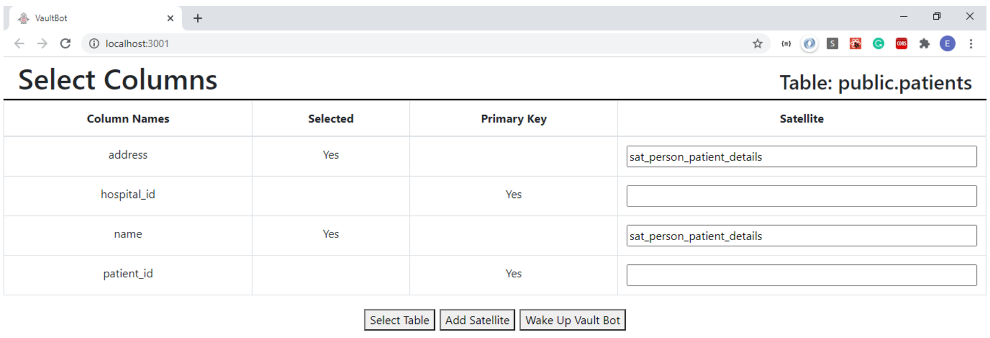


Select Columns

Table: public.patients

Column Names	Selected	Primary Key	Satellite
address			
hospital_id			
name			
patient_id			

Figure D.2: The data vault design screen for the patients table



Select Columns

Table: public.patients

Column Names	Selected	Primary Key	Satellite
address	Yes		sat_person_patient_details
hospital_id		Yes	
name	Yes		sat_person_patient_details
patient_id		Yes	

Figure D.3: The completed data vault design screen for the patients table

Listing 3: An example of the auto-generated code for a Hub

```
hub_person={'__tablename__': 'hub_person',
'__table_args__':{'schema':'public'},
'id': Column(column_types['integer'], primary_key=True)}
primary_keys = []
for keys, values in {
    'doctor_id': {'data_type': 'bigint'},
    'hospital_id': {'data_type': 'bigint'},
    'patient_id': {'data_type': 'bigint'}
}.items():
    primary_keys.append(
        {keys: Column(column_types[values['data_type']])}
    )
for key in primary_keys:
    hub_person.update(key)

person = type('PUBLIC_Hub_Person',(Base,),hub_person)
```

Listing 4: An example of the auto-generated code for a Link

```
person_object_link={'__tablename__': 'person_object_link',
'__table_args__':{'schema': 'public'},
'id': Column(column_types['integer'], primary_key=True),
'person_id': Column(column_types['integer'], ForeignKey(hub_person.id)),
'object_id': Column(column_types['integer'], ForeignKey(hub_object.id))}

person_object = type('PUBLIC_Person_Object_Link',(Base,),person_object_link)
```

Listing 5: Two examples of the auto-generated code for the Satellites

```
person_doctor_details = type(
    'PUBLIC_Sat_Person_Doctor_Details',(Base,), new_satellite
)

new_satellite={'__tablename__':'sat_person_patient_details',
'__table_args__':{'schema':  'public'},
'id': Column(column_types['integer'], primary_key=True),
'source_table': Column(column_types['string']),
'hub_id': Column(column_types['integer'], ForeignKey(hub_person.id))}

columns = []
for keys, values in {
    'name': {'data_type': 'varchar(255)'},
    'address': {'data_type': 'varchar(255)'}
}.items():
    columns.append(
        {keys: Column(column_types[values['data_type']])}
    )
for column in columns:
    new_satellite.update(column)


person_patient_details = type(
    'PUBLIC_Sat_Person_Patient_Details',(Base,), new_satellite
)

new_satellite={'__tablename__':'sat_object_treatments_details',
'__table_args__':{'schema':  'public'},
'id': Column(column_types['integer'], primary_key=True),
'source_table': Column(column_types['string']),
'hub_id': Column(column_types['integer'], ForeignKey(hub_object.id))}

columns = []
for keys, values in {
    'treatment_name': {'data_type': 'varchar(255)'}
}.items():
    columns.append({keys: Column(column_types[values['data_type']])})
for column in columns:
    new_satellite.update(column)

object_treatments_details = type(
    'PUBLIC_Sat_Object_Treatments_Details',(Base,), new_satellite
)
```

Listing 6: An example of an auto-generated control file

```
control_files = {}

public_patients_hubs = {
    'table': 'public.patients',
    'hubs': [
        {
            'hub': 'hub_person',
            'keys': [
                'patient_id',
                'hospital_id'
            ],
            'data_types': {
                'patient_id': {'data_type': 'bigint'},
                'name': {'data_type': 'varchar(255)'},
                'address': {'data_type': 'varchar(255)'},
                'hospital_id': {'data_type': 'bigint'}
            }
        }
    ]
}
public_patients_satellites = {
    'satellites': [
        {
            'satellite': 'sat_person_patient_details',
            'columns': [
                'name', 'address'
                ],
                'hub': 'hub_person',
                'hub_id': 0,
                'data_types': {
                    'name': {'data_type': 'varchar(255)'},
                    'address': {'data_type': 'varchar(255)'
                }
            },
            'source_table': 'public.patients'
        }
    ]
}
public_patients_links = {
    'links': []
}

control_files.update({'public.patients': {
    'hubs': public_patients_hubs,
    'satellites': public_patients_satellites,
    'links': public_patients_links
}})
```