Project no. 826278

SERUMS

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

Report on Refined Specification of Smart Patient Health Record Format
D2.3

Due date of deliverable: 30th April 2020

Start date of project: January 1st, 2019

Type: Deliverable
WP number: WP2

Responsible institution: Sopra-Steria Ltd.
Editor and editor's address: Euan Blackledge, Andreas Francois Vermeulen, 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	27/04/20	E Blackledge	SOPRA	Version 001.000
2	27/04/20	A Vermeulen	SOPRA	Version 001.000

Executive Summary

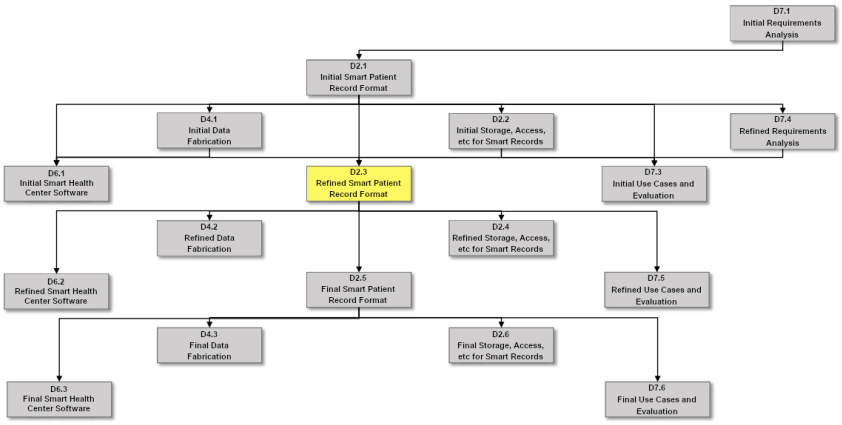


Figure 1: Dependencies between D2.3 and other deliverables

This deliverable describes the refined version of the Smart Patient Health Record format that is being used throughout the **Serums** project for encapsulating the patient data. It is a part of WP2, the objectives of which are to:

1. Define a format for smart patient records that can contain data distributed over multiple devices
2. Develop mechanisms to track the lineage of accesses to the smart patient records
3. Develop mechanisms to control storage and access rights for smart patient records
4. Build on existing and develop new machine learning methods for extracting metadata from unstructured data

The format of the data as described in this deliverable will be used throughout the project. Its relationship to other deliverables can be seen in Figure 1.
In the subsequent deliverable related to the Smart Patient Health Record format, we will finalise the format presented in this deliverable to accommodate data that can reside on different sources collected/exchanged outside of the local environment, e.g. in scenarios of trans-national exchange of data (D2.5).

Contents

Executive Summary 2

1 Introduction **5**

2 Project Use Case Descriptions **7**
2.1 Edinburgh Cancer Data Gateway with integrated Patient Reported Outcome Measures (USTAN) 7
2.2 Fundació Clìnic per a la Recerca Biomèdica (FCRB) 8
2.3 Zuyderland Medisch Centrum (ZMC) 8
2.4 Serums' Three Use Cases 8

3 Smart Patient Health Record Format **9**
3.1 Data Lake 9
3.2 Data Vault 9
3.3 API 11

4 Technical Implementation **12**
4.1 Construction of the data lake 12
4.1.1 What is a data lake? 12
4.1.2 The structure of the Serums data lake 12
4.1.3 The technology the Serums data lake is built with 13
4.1.4 Changes to the data lake since the initial specification . . . 14
4.2 Construction of the Smart Patient Health Record 14
4.2.1 The data lake, the data vault, and the JSON object 14
4.2.2 The structure of the Smart Patient Health Record 18
4.2.3 The technology the Smart Patient Health Record is built with 18
4.2.4 Changes to the Smart Patient Health Record since the initial specification 20
4.3 Construction of the API 20
4.3.1 What is an API? 20
4.3.2 The structure of the API 21
4.3.3 The technology of the API 22
4.3.4 Changes to the API since the initial specification 24

Bibliography **26**

Appendices **27**
A End-to-end creation of a Smart Patient Health Record for a use case partner 28
B The data lake directory tree 32
C Interactive documentation for the API 34

Chapter 1

Introduction

Version D2.3 is the *refined specification* of the *Smart Patient Health Record* (SPHR) 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. The main challenge for the **Serums** project is to deal with *decentralisation* of the information related to a particular patient. In modern health-care systems, the data about a single patient may reside on different subsystems in the same health-care centre, or even, in some scenarios, scattered across different health-care 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 health-care 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. 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 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 an intermediate step towards deriving a uniform **Serums** Smart Patient Health Record format. Here, we are focusing specifically on the *centralised* data, where all of the patient data is available at the same place. In the subsequent deliverable D2.5 we will publish the final format, extending this format further to support distributed data.
We propose the *data vault* as an appropriate generic format in which the data will be kept (these are described in Section 4.2.1.2). Data vaults are recognised in data science as a universal format that allows easier and more automatic data analytics. Our ultimate goal is to be able to represent the data of all use cases (described in Section 2 in the form of data vaults). For transmission, we are choosing JSON (this is described in Section 4.2.1.3). This is for the convenience of the **Serums** project, as it allows for easy integration between the different work packages.

Chapter 2

Project Use Case Descriptions

The core use of the Smart Patient Health Record is to create a singular electronic health record that the patient has autonomous control over as per General Data Protection Regulation (GDPR) [1].
One of the objectives of the **Serums** project is to understand what is required to derive a *universal* Smart Patient Health Record format that will be applicable to a wide variety of different use cases that manipulate patient data, as well as what techniques and methodologies are required for a practical implementation of this format. We are especially concerned with providing patients the possibility to grant specific granular-level access to the record to the approved service providers and be able to revoke the access if required.
In this section, we describe all three project use cases used to derive the refined version of the Smart Patient Health Record format. Our (D2.3) focus was to add the final two use cases to the existing **USTAN** use case. Our (D2.3) format is shown in Section 3.
Next we will discus a summary of the three use cases.

2.1 Edinburgh Cancer Data Gateway with integrated Patient Reported Outcome Measures (USTAN)

We aim to build a predictor within NHS Lothian for toxicity levels from treatment regimens for cancer patients with or without comorbidities. Giving patients the opportunity to give more accurate information on their symptoms throughout the treatment (possibly daily whilst at home), and combining that information with patient characteristics, cancer information and treatment regimen, will allow clinicians to adapt treatments better to individual patients with better patient outcomes and controlled toxicity levels. Further data from hospitalisations and comorbidities will contribute to a more accurate prediction [5].

2.2 Fundació Clìnic per a la Recerca Biomèdica (FCRB)

We aim to build a Chronic Disease Management System to handle several chronic diseases via the Primary Care Centre while ensuring the patient remains independent for as long as possible. For that reason, Doctors, from the Hospital Clìnic de Barcelona, specialists in Diabetes, can supply wearable medical devices.
Under the GDPR, the patient in Spain can invoke their right to porting [1] to enforce the transfer of this data to any EU state based healthcare provider that is signed-up to **Serums**.

2.3 Zuyderland Medisch Centrum (ZMC)

As recovery in the hospitals is shifting from long stay to short stay, the need for monitoring the condition of our patients at home is growing. Therefor we aim to create a platform where patients share their medical information and home measurements with any medical professional of choice. This platform must be able to communicate and share data between multiple parties, such as hospitals, physicians/surgeons at hospitals, general practitioners, various therapists, home measurements and E-coaches within **Serums**. The patients are able to receive personalized feedback and warning signals based on the collected data from the system. To demonstrate this, we follow a patient whose hip has been replaced and whose recovery is monitored at home with the Zuyderland Activity Monitor. Documentation about the surgery and the results of the Zuyderland Activity Monitor are shared with the surgeon and the physiotherapist by the patient. The **Serums** system must comply with the GDPR and the Dutch MedMij standards.

2.4 Serums' Three Use Cases

Serums now (D2.3) supports all three use case studies for the first time with the Smart Patient Health Record able to extend with ease to cover other data sources. This is achieved via the use of the extendable data vault and data processing framework that uses the Smart Patient Health Record ecosystem to share data between approved medical providers as instructed by the GDPR contracts setup by the patient.

Chapter 3

Smart Patient Health Record Format

The Smart Patient Health Record (SPHR) for D2.3 is a continuation of the development seen in D2.1 and D2.2. Following are the building blocks which make up the SPHR. These are very high-level overviews of each component, to be used as a quick reference for the architecture of the SPHR. See Chapter 4 for a much greater level of detail on each component.

3.1 Data Lake

The data lake is where the data for the SPHR is stored. It contains six (6) zones, which can be seen in Figure 3.1. The benefits of this structure were detailed in our paper *The SERUMS tool-chain: Ensuring Security and Privacy of Medical Data in Smart Patient-Centric Healthcare Systems* published by the IEEE in 2019 [2].

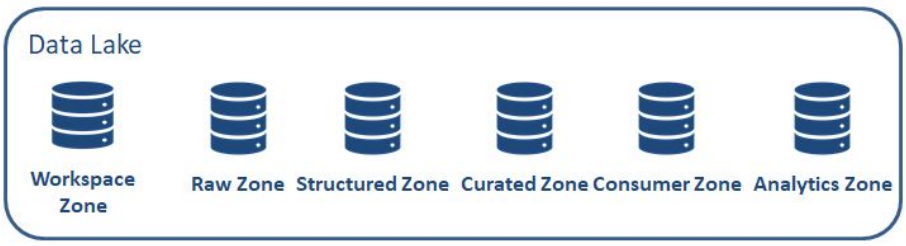


Figure 3.1: The six zones of the data lakes

3.2 Data Vault

A data vault [4] is the storage format we have chosen as the base for the SPHR. It allows for new sources of data to be added without the need for complex redesigns

of the existing data structures. 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.
The structure relies on three (3) base type of tables: the Hubs, the Links, and the Satellites. The Hubs contain the business keys, the Links join the Hubs, and the Satellites contain the actual data. These three table types can be seen in Figure 3.2, with the top row showing two Hubs joined by a Link, and two Satellites on the bottom row, each connected to its parent Hub.
Our Hubs have been chosen as Time, Person, Object, Location, and Event (TPOLE). With these five (5) categories, we are able to classify any incoming data. These Hubs can be seen illustrated in Figure 3.3.
The data vault concept is by Dan Linstedt and can be found detailed in *Building a Scalable Data Warehouse with Data Vault 2.0* [3].

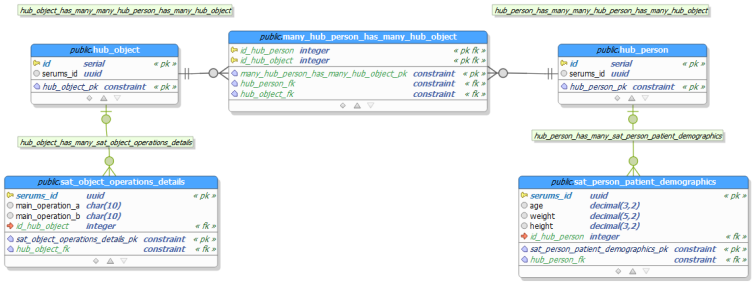


Figure 3.2: Depiction of the relationships between Hubs, Links, and Satellites

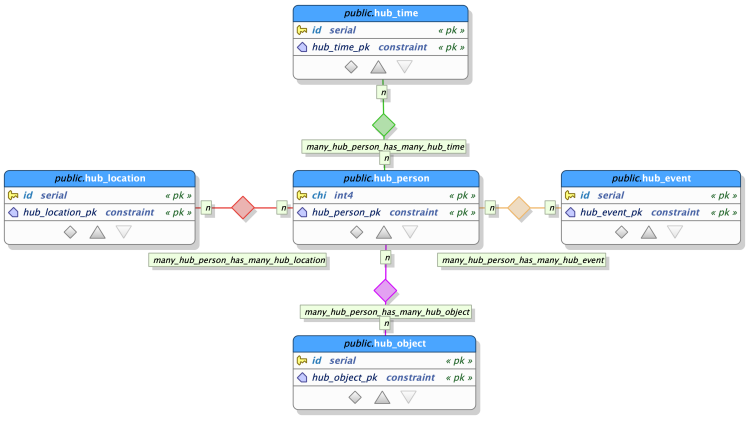


Figure 3.3: The TPOLE Hubs of the Serums data vault

3.3 API

New to D2.3 is the API. An API is a programmatical interface where a system can retrieve data from or execute commands. In the context of D2.3 the API is used for external systems to retrieve data from the data lake as a SPHR. This takes the form of an event called a *request*, which requires a *body*. The *body* contains text fields which are required to successfully initiate the *request*. The text fields of the *body* will be used as part of the code execution undertake by the API. The *response* of a *request* will be returned back to the system that initiated the *request*. This interaction can be seen in Figure 3.4. Details of both the *request* and *response* can be found in Section 4.3.3.

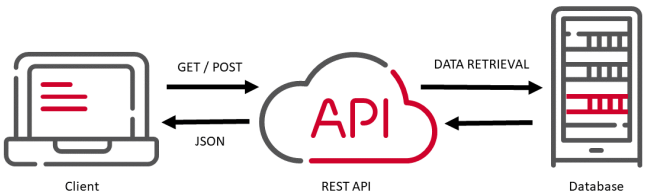


Figure 3.4: A basic API interaction

Chapter 4

Technical Implementation

In this section we describe how the new (D2.3) Smart Patient Health Record (SPHR) is created from a technical viewpoint. This will cover a brief outline of the construction of the existing (D2.2) data lake from which the SPHR pulls its data, the new (D2.3) construction of the SPHR format, and finally the new (D2.3) construction of the API for generating and delivering the SPHR.

4.1 Construction of the data lake

4.1.1 What is a data lake?

While a traditional database is normally used for storing curated data, a data lake accepts data in any form. This allows enterprises to centralise the majority of their data, no matter how raw that data is. As such we can expect to find a range of formats and usefulness within the data lake as users can and will store everything from customer enquiries to images of old receipts to the core operational data the enterprise relies on to operate.

4.1.2 The structure of the Serums data lake

Each use case partner (ZMC, FCRB, and USTAN) has their own data lake. These are stored separetely to better replicate real world conditions, where each data lake would be directly tied to the health care provider, allowing for either real-time or batch processing of their patient data.
A key feature of the **Serums** data lake is that all instances follow the same mapping. An example of this is the directory "*100-DL/100-Raw-Zone/200-Internal/100-CSV/*". This exists in all of the data lakes, and is where the healthcare provider would store any CSVs they wished to be part of the **Serums** infrastructure. With all data lakes following this common format, it allows us to programmatically access data across all users without the need to manually search for the location of files or write custom code to comb through each data lake. A complete directory tree for the data lake can be found in Appendix B.

The data lakes are broken up into six (6) zones (Figure 3.1), each serving specific uses. For instance, when a new file is added to the data lake, it will be added into the raw zone. As different forms of processing are applied to the file (adding structure, metadata, classification, etc) copies, or copies of specific elements, of the file will be created in the structured zone, followed by the curated zone, before finally being placed in the consumer zone. It is this consumer zone from which the SPHR will be served.
There are an additional two zones, the analytics zone and the workspace zone. The workspace zone is simply a test bed for new code, which allows the **Serums** developers to work with copies of data, without running the risk of damaging the original versions. The analytics zone will feature copies of the curated zone's data, and will allow any authorised parties to run analytics from, again without the risk of source data being corrupted.

4.1.3 The technology the Serums data lake is built with

Each data lake comprises of a file system within a Linux drive, combined with a PostgreSQL database. This is a highly flexible and powerful combination of technologies. The relationship between these two systems in the lifecycle of the creation of the SPHR is demonstrated in Figure 4.1.

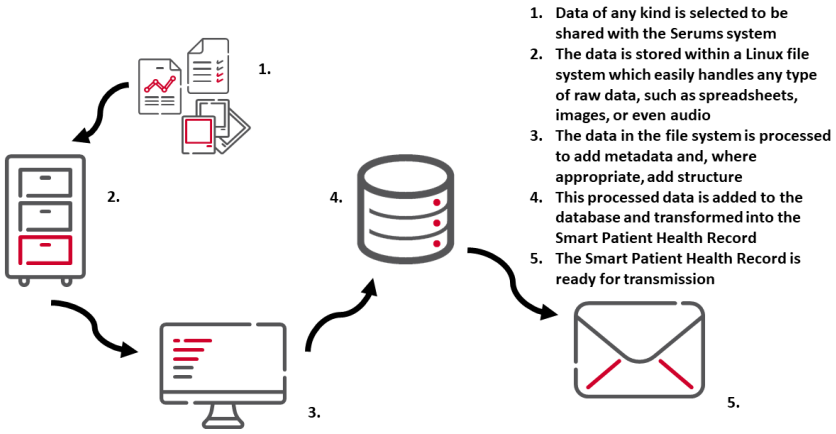


Figure 4.1: Depiction of how the Linux File system and PostgreSQL database are used in conjunction with each other during the creation of the Smart Patient Health Record

The Linux file system allows us store files of any conceivable format. It is also an ideal environment to deploy code in.
PostgreSQL is a full featured database that is perfect for our needs. It plays a key role in bringing structure to the raw data, as well as helping to generate some of

the most basic metadata.
Both Linux and PostgreSQL are open source. By focusing on open source technologies, we not only keep costs down, we also benefit from large developer communities already built up around them.

4.1.4 Changes to the data lake since the initial specification

We had originally used Hadoop as the file store for the data lake. This too is an open source technology, however, it does not have as rich a developer community surrounding it. With its focus on *Big Data*, it tends to be used by large corporations, who are able to hire specialists to set up and maintain the Hadoop cluster. With the lack of a large developer community, it proved difficult to implement correctly and, ultimately, brought no real benefit to the project.
The Hadoop instance we were running was a prebuilt trial version supplied by Cloudera as a Virtual Machine (VM). Since it was prebuilt, it was limited in adjustments we could make to it, as such we used the included relational database which was MySQL. The differences between MySQL and PostgreSQL are minimal at the levels we are using them, however this too would have limited our ability to radically alter the default behavior of the database itself.

4.2 Construction of the Smart Patient Health Record

4.2.1 The data lake, the data vault, and the JSON object

The Smart Patient Health Record (SPHR) can be thought of in many different forms. At its most basic, it is simply the selected records of a patient from a single healthcare provider. **Serums**, however, is combining the data from multiple source systems within a single format that is capable of being transmitted in a secure way. In order to achieve this we take the raw data from each of the data lakes, transform it into a data structure known as a data vault, and finally transmit it as a JSON object to the **Serums'** frontend for use by the patient or healthcare provider.

4.2.1.1 The data lake

The structure of the data lakes has been covered in detail in the previous section. An important caveat to mention here however is that the data we are using has been fabricated by IBM as part of WP4. As such, all of it is provided to us as CSVs. In the real world, it is likely that each data lake would be connected directly to the hospitals' systems. The data that enters their **Serums** data lake would take many more formats than this. The outline of the process would not change however, we would simply apply different forms of processing to the data in order to reach the same end goal.

4.2.1.2 The data vault

With the goal for **Serums** being that the technology is one day rolled out across healthcare providers, it was clear that we needed to choose data modelling that allows for constant changing of the underlying data, in terms of structure, volume, and detail. We have chosen data vault to solve this issue. By separating business keys (the patient id in the case of **Serums**) from the descriptive attributes, we are able to avoid having to worry about making drastic changes to the overall schema. An example of this can be seen in the Figures 4.2, 4.3, 4.4, and 4.5.
Figures 4.2 and 4.3 show a traditional style database, and how introducing a new data source requires multiple new joins to be made between the existing tables and the new one. Scale this issue up to a brand new department within a hospital entering the **Serums** ecosystem, or even multiple regions of a country's healthcare services linking together and this standard data model becomes untenable.
Figures 4.4 and 4.5 show a data vault containing exactly the same detail, however now only a single change is required to be made to the existing data model. This offers incredible flexibility as new sources can easily be added or redundant sources removed, without the need to refresh the entire database. The trade-off for this flexibility is the initial setup requires understanding of the underlying data, especially if it is being converted from an existing data source.
The key to the data vault structure is its use of Hub, Links, and Satellites (Figure 3.2).

- The Hubs form the backbone of the data model, containing only the business keys, as well as its own internal id used for linking together the Hubs. We have decided on five (5) categories for our Hubs. These are Time, Person, Object, Location, and Event (TPOLE). With these five categories, we are able to classify any incoming data under one of these headings

- The Links are just that, they link together the Hubs. They form many-to-many relationships between the Hubs, which gives the freedom for complex relationships to form between the Satellites across numerous Hubs.

- The Satellites are where the majority of the useful data resides. These are purposely designed to be as atomic as possible, each containing only very closely related data. This serves two purposes: first, it allows us even finer control over access to the data. Second, should we find ourselves with two sets of very similar data from multiple end users, we can evaluate more easily whether these can be combined into a single table or kept as two

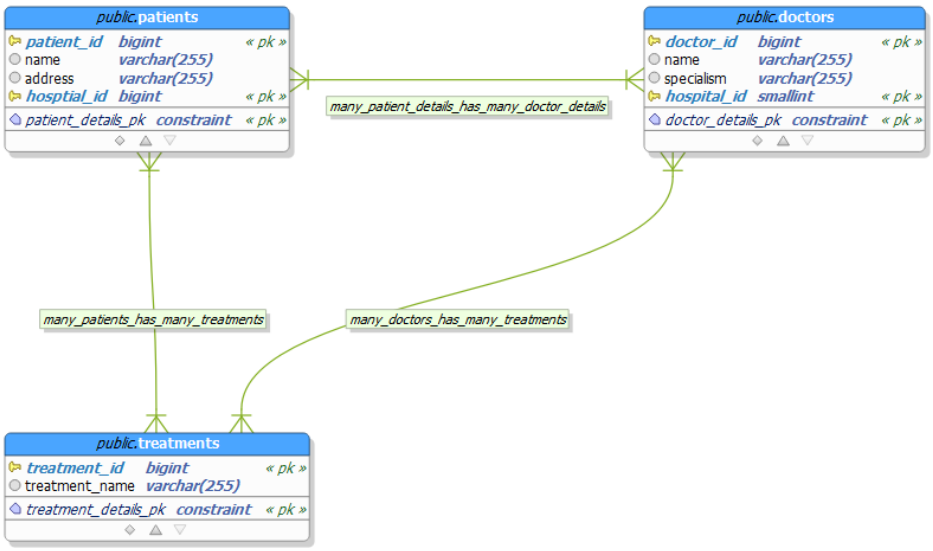


Figure 4.2: Starting point of traditional style database

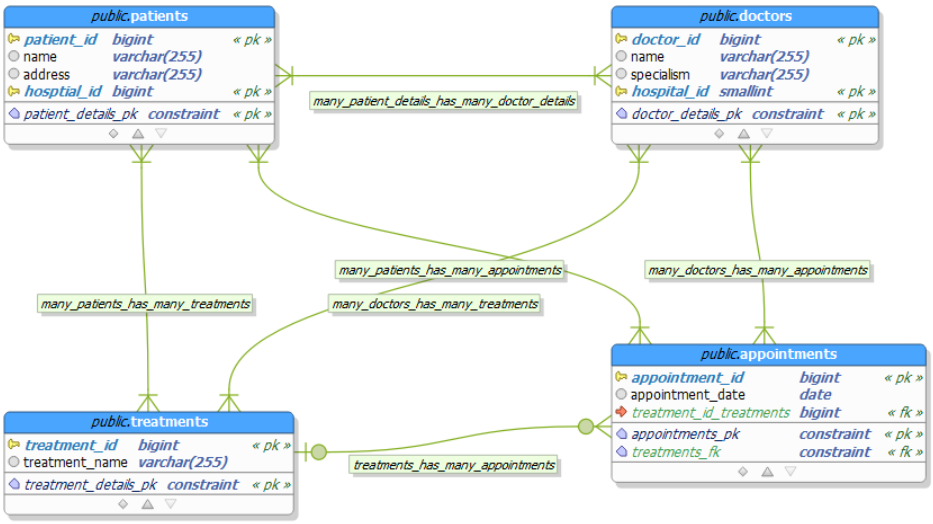


Figure 4.3: End point of traditional style database

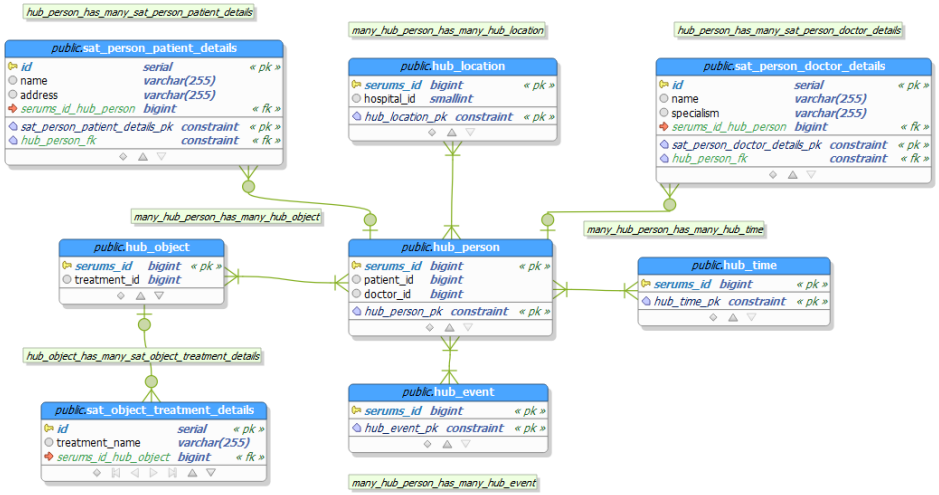


Figure 4.4: Starting point of a data vault style database

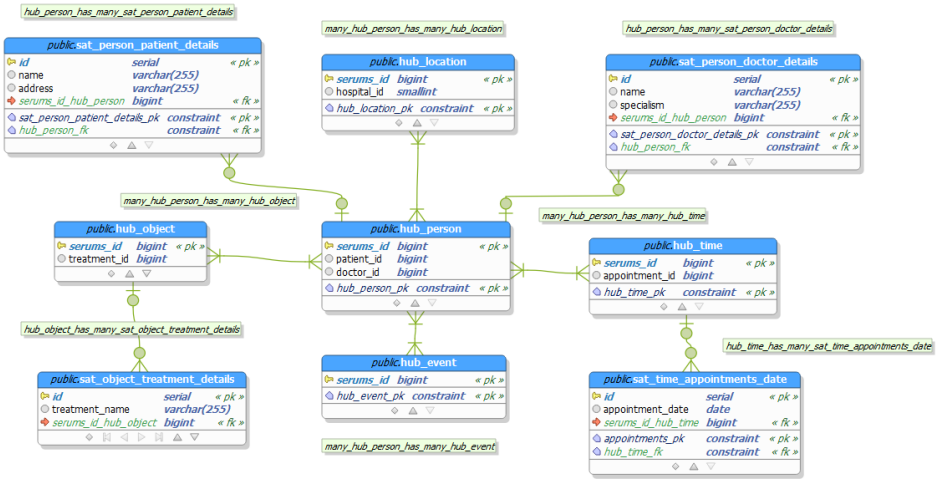


Figure 4.5: End point of a data vault style database

4.2.1.3 The JSON object

Our chosen output for the SPHR is as a JSON object (JSON stands for JavaScript Object Notation). Since the **Serums** project will be demonstrated via a web interface, transporting the data in this method allows easy integration between the backend server and the frontend website. JSON has the advantage of being recognised by a range of programming languages beyond JavaScript, including Python, Perl, Java, C, C++, C#, etc. As such, a number of different applications and platforms would be able to integrate the **Serums** data, without the need to develop tools to parse it first.

The object itself contains encrypted data, specifically an encrypted key and the

encrypted record. Exact details of this encryption will be covered in Section 4.3.3 that describes the construction of the API. The important thing to note is that the data is encrypted before it leaves the backend server, and must be decrypted by the frontend. This mitigates the risk of a data breach should the network traffic be intercepted.

4.2.2 The structure of the Smart Patient Health Record

The SPHR is the result of the application of granular access controls placed upon any and all of the data a patient has within the **Serums** system. As such, it could be the entire patient record if the patient so wished, or it could be a very specific subset of the data, given to a specialist who is external to the patient's primary care center.
Since the patient could be a patient of multiple healthcare providers within the **Serums** system, we must first pull together and classify the data from all potential sources into a single source. This is where our data vault comes into play. The schemas for each of the use case partners' data sets have been manually generated in data vault format, allowing data to be rapidly pulled from the source data sets and converted.
At this point, rules can be applied to the data which will control the outputs. These rules have been designed by the work package, and the current set, as well as a detailed description of how they are used can be found in Section 4.3.3.2. For the final version of the deliverable (D2.5) the patient themselves will be able to define these rules and store them on the blockchain (T2.2). Once the rules have been applied, the SPHR is ready for transmission.

4.2.3 The technology the Smart Patient Health Record is built with

As was covered in Section 4.1, each patient's data is stored within the data lake for the healthcare provider at which they have received care. As a reminder, these data lakes are the combination of a Linux file system and PostgreSQL database in a prescribed structure.
Preprocessing takes place in order to transform the source data structures into the data vault models. This is handled by a series of Python functions written as Jupyter Lab notebooks. This allows the entire preprocessing to be handled automatically, thus being scalable, repeatable, and reliable. Specifically the process takes the CSVs provided by IBM and stored in the corresponding data lakes, creates exact copies of these with the PostgreSQL database, before abstracting them out into their data vault forms.
The data set that is the easiest to visualise this with is from the wearable device found in the ZMC use case. Figure 4.6 shows a sample of the synthetic data in CSV form as provided by IBM. This is then converted into a table with the structure as seen in Figure 4.7. Finally it can be found in its data vault model in Figure 4.8. No data has been lost at any point of these transformations, with the result of the

previous step being kept to ensure we can track the lineage of any data within the **Serums** system.

patient_nr	day_nr	time_total	time_passive	time_active	time_sit	time_stand	time_walk	time_cycle	time_hi	nr_sst	steps_total	cadence	cyc_rot	cyc_rpm
1	1	29287	9919	19368	7581	2338	19368	0	0	30	20080	62	0	
2	1	49328	30105	19223	30104	1	19223	0	0	30	20181	62	0	
3	1	69855	57994	11861	53103	4891	11861	0	0	21	11255	56	0	
4	1	62178	54860	7318	43671	11189	7318	0	0	15	6159	50	0	
5	1	62214	51522	10692	41985	9537	5975	0	4717	13	5105	51	0	
6	1	39775	26635	13140	25396	1239	13140	0	0	21	11554	52	0	
7	1	66680	54885	11795	52056	2829	11795	0	0	21	11006	55	0	
8	1	71115	58274	12841	49278	8996	12841	0	0	21	11207	52	0	
9	1	71165	62930	8235	53069	9861	7289	946	0	16	6188	50	666	42

Figure 4.6: Sample of the synthetic data for ZMC's use case in CSV format

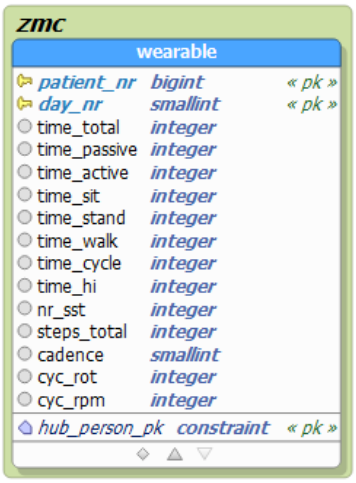


Figure 4.7: Representation of ZMC's use case data in PostgreSQL

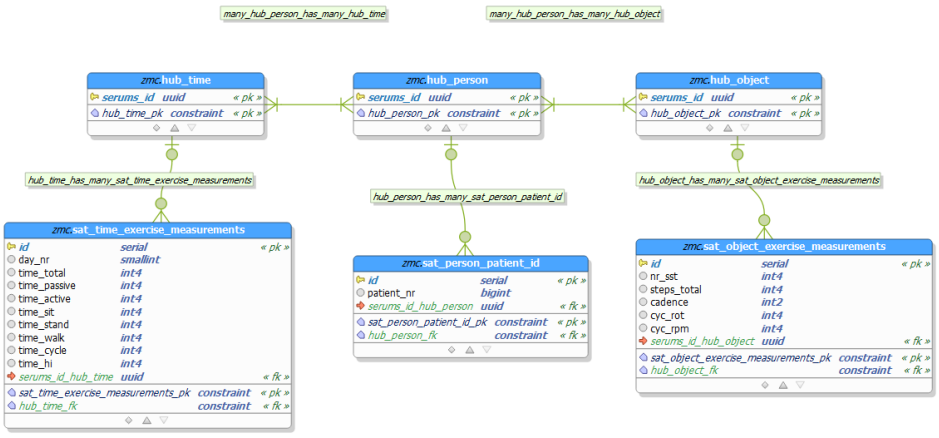


Figure 4.8: Representation of ZMC's use case data as a data vault in PostgreSQL

Should the patient wish to share the details about *how long* they took part in the

various forms of exercise being measured in this example, they could invoke the "Time" rule (a full list of the current rules can be found in Section 4.3.3.2) which would only permit the table **zmc.sat_time_exercise_measurements** from Figure 4.8 to be shared with the health professional. Of course they could choose to share all of the details, we have simply given the patient this granular level of control.
The SPHR is now ready for transmission.

4.2.4 Changes to the Smart Patient Health Record since the initial specification

We had originally planned for a data vault structure being developed within PostgreSQL to form the basis of the SPHR. As such this has remained constant between versions.
The largest change has come from the move from HDF5 (Hierarchical Data Format 5) to JSON as the actual output. This is simply down to how the **Serums** technical partners will be integrating our work packages. Since we are using a web browser interface, JSON is a more natural fit. Were we to replicate an actual hospital's hardware system, we would likely add HDF5 as an *additional* output option as this matches the standard format used across the world by healthcare providers.
A further change has come in the fact that we can now begin to apply rules to the output, enabling the patient to control levels of access to their data. These rules (Section 4.3.3.2) are still in an early stage of development and will change before the final iteration of this deliverable, however they do give a sense of how the final system will behave, as well as giving the blockchain a set of rules to test against as part of their next deliverable (D2.4).

4.3 Construction of the API

4.3.1 What is an API?

An Application Program Interface (API) is essentially a middleman between two applications. It allows these applications to interact without necessarily knowing what is going on behind the scenes of the other one. This means that a developer from Company A can use features of Company B's product within their own application, without the need to integrate their two systems.
For this deliverable, the API has the ability to accept a request for a patient's data, and return said data in an encrypted form. The request in this case will come from our system's frontend which is being developed as part of the integration work package (WP6). The frontend will be making numerous API requests (it will speak to the authentication layer, the blockchain layer, and the data lake) as part of the interaction between patient, healthcare provider, and patient data, however the user of the system will be unable to see that the web browser is not really doing any of the work other than visually representing the results of the various API requests.

4.3.2 The structure of the API

The API itself has one (1) end point at this stage. For reference, an end point in this context is the web address for the request. The end point in question is "**api/get_sphr**". This section will take a high level look at the API before Section 4.3.3 goes into specifics about the actions taking place.
This end point allows the frontend to make a request for data from the various data lakes. In order to return the patient record, we require that the frontend passes the API a packet of data in JSON format. Specifically the API requires a patient id, a key for encryption (this will be covered in more detail in Section 4.3.3), and a rule to invoke (Section 4.3.3.2).
An example of what the request looks like is as follows:


```
{
        "public_key": "——BEGIN PUBLIC KEY——
        MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAzaSIE9H
        TGxw8vVCc0k0VDZ+xj2CcCy0IfcGR0qKviRDLTaT0t+zuOX9ACn
        +wd+3SFa+OL4c6teAu/NIzfGxUkhBp2H4hCY8XwRiGCuycv/FpD
        DnaPP4ICZgzN8IOrHvbPyBTd273+DfqL+IR6xO8KCuhnOq+wBIF
        fAJmFopPFQaTsHrW1jrgGZhJYJO7IFwawbec7dhYvLURjxzQqx1
        yyUkJ7AUocQMsKlQS+kPlR7Qgj3URRJUvMYrBWxxAorm/ygN0Rw
        jwIDAQAB
        ——END PUBLIC KEY——",
        "rule":"person",
        "serums_id":"cb944c1c-6560-4ccb-875b-113a826e10df"
}
```


These values are used by the API to generate and return the SPHR. The rule is used to select the specific subset of tables, the serums id is used to select the records for one specific patient, and the public key is used for a form of encryption. In return, the API hands the frontend back a new JSON object, this time however containing the SPHR and the key to unlock it. An example of what the response looks like is as follows:


```
{
        "key": "gAAAAABej3aSK55NbKf8PdkKFMnhFGlrAXyMwhFR0HKue
        R8gclhEeAdz0007fuHtpYt1Npee3S2aCGbvedQJvrpvH1C1vEErvw==",
        "data": "sYTaaUd63cfboIHm9gIlpwB9yVRpip5KYjMDclkjMOf5LY
        fx5qleZjxlPdnlTDgn/cLfOFXVTRReSdp/3uqbG8vKsKRaXmJ/GHAO
        ep/mypSTSp/0L8XLdGvEj5fGW8O3FwZNEjYP1jATPCFm93JbplgqAc
        PYn+hkyPpUSYyRYjLJNGWBcCBS5YFsvsQ+lJ9o2BjWpYTT+S/RlAf5
        HKB8/IdcjIZ52HF1QuGbvfcBb61HxSjg15qFTIaI1+EmMSr8dnQ9ZY"
}
```

With these, the frontend is now capable of decrypting the record, and displaying the results for the patient or health professional.

4.3.3 The technology of the API

4.3.3.1 Encrypting the Smart Patient Health Record

The security of the API comes from its use of dual encryption. The frontend using asymmetric encryption, and the API using symmetric encryption.
Every time the frontend makes a request for data to the data lake, it must generate an RSA keypair. This keypair is made up of a public key and a private key. The private key, as its name suggests, is kept secret. It is the key that is used to unlock data that has been encrypted with the public key. The public key, as *its* name suggests, can be made public and shared. The public key's function in this instance is to act as a lock.
It is statistically improbable that the private key could be derived from the public key. These are generated through the multiplication of incredibly large prime numbers, and modulus mathematics ensuring that, even by brute force attacks, it is highly unlikely that an attacker would be able to break the encryption.
A draw back with this type of encryption, however, is the volume of data which can be encrypted by the public key. As such, in order to encrypt a potentially large volume of data, we rely on a different type of encryption. For the API, we have chosen Fernet encryption for its high level of security, as well as the length of data it is capable of handling. Fernet encryption works by generating a key, which in turn is used to prime an encryption algorithm, which itself is used to encrypt the data.
Within the API, once the SPHR has been generated, it is transformed into a JSON string, before being encrypted by a Fernet algorithm. The key that was used to generate the Fernet algorithm is itself encrypted using the public key that was provided as part of the API request. With both elements encrypted, they are bound to a JSON object and returned to the frontend.
The frontend is now capable of: firstly decrypting the Fernet key, using the private key which has not been revealed to the outside world, before the newly decrypted Fernet key is used to generate a decryption algorithm for use on the SPHR. At this point the request is completed and the SPHR can now be viewed. This lifecycle can be viewed in Figure 4.9.

4.3.3.2 Selecting the data to return

The key to the selection of data is the rule which is passed as part of the API call. In the final iteration of the software, the patient will be able to design these themselves, giving them granular control over who can see what. As such, this is a core tenet of **Serums**. For this interim stage, however, a basic set of groupings have been chosen to give a sense of this control, as well provide us a set of values

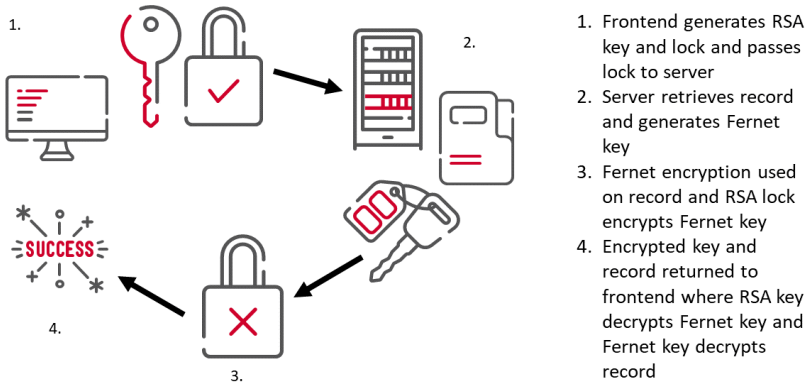


Figure 4.9: Lifecycle of the ecryption process

to test the integration of the WP2 elements, including the SPHR, the data lake, and the blockchain.
These rules are as follows:

- The five Hub categories of our data vault format:
  - Time, Person, Object, Location, Event
- Each of the three use case partners' entire data collection
  - ZMC, USTAN, FCRB
- Two more general rules, one which captures just some basic details about the patient, and one which returns all of the tables from all of the use case partners in which the patient appears
  - General, All

When the rule is received by the API, it is first looked up to determine the tables the query is allowed access to. At this point, the tables are searched using the **Serums** ID provided by the request. Once all results have been found, these are then encrypted using the method described above, before being returned to the frontend for decryption.
If we refer to Figure 4.8, and invoked the "Time" rule we would return only the Satellite **zmc.sat_time_exercise_measurements**, if we invoked the "Object" rule we would return only the Satellite **zmc.sat_object_exercise_measurements**, whereas if we invoked the "ZMC" rule we would return the Satellites **zmc.sat_time_exercise_measurements**, **zmc.sat_person_patient_id**, and **sat_object_exercise_measurements**.

4.3.4 Changes to the API since the initial specification

In the initial specification of the SPHR we did not have an API to return the data. As such, both the ability to select specific subsets of data for a patient, and the ability to encrypt the data are new for this iteration.

5. Conclusion

The current *Smart Patient Health Record* (Chapter 3) is version 001.002 and will evolve over the period of the research of **Serums** project. The *Smart Patient Health Record* version 001.002 included all other research use cases identified by the consortium. The *Smart Patient Health Record* will evolve as the researchers add machine learning processing but WP2 do not foresee any major change to the current Smart Patient Health Record's core structure, other than adding more granular controls to the data selection via the creation of rules by the patient.

Version 001.003 of *Smart Patient Health Record* (Chapter 3) will be the final formatted version for this research project and will be included as part of deliverable 2.5.

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.

[5] Agastya Silvina, Juliana Bowles, and Peter Hall. On predicting the outcomes of chemotherapy treatments in breast cancer. In *Conference on Artificial Intelligence in Medicine in Europe*, pages 180–190. Springer, 2019.

Appendices

Appendix A

End-to-end creation of a Smart Patient Health Record for a use case partner

The creation of the Smart Patient Health Record (SPHR) and the data contained therein has required collaboration between SOPRA, each of the use case partners, and IBM. In this section, we will walk through the creation of the SPHR tables for FCRB.

As with all of the use case partners, we first required the extraction of metadata from the tables FCRB wished to use for its use case. It is important to understand that the metadata does not contain any actual data that might identify someone, it merely describes what typical data for a particular field should look like. An example of one of the tables provided by FCRB can be found in Figure A.1. This is the metadata for a table named Diagnostic, and this table will be referenced throughout. A complete overview of the database they have provided can be found in Figure A.2.

With the metadata collected, a model was generated for each of the tables. This was achieved with the use of pgModeler, a database modelling tool. These tables can be seen in Figure A.3. pgModeler generates SQL files which are used by both SOPRA and IBM. The SQL for the Diagnostic table is as follows:

```
CREATE TABLE fcrb.diagnostic (
        einri char(4),
        patnr char(10) NOT NULL,
        falnr char(10),
        lfdnr varchar(255),
        dkey1 char(30),
        erusr char(12),
        CONSTRAINT diagnostic_pk PRIMARY KEY (patnr)
);
```

Diagnostic

Field name	Description	Type of variable
EINRI	Medical Centre. Can be 'HCPB' (p=0.75), 'BCL' (p=0.15) or 'HCM' (p=0.10).	CHAR 4
PATNR	Unique patient identifier. Number starting by '1'.	CHAR 10
FALNR	Unique episode identifier. Number starting by '1000000001' if it is BCL , starting by "2000000001" if it is from HCPB and "3000000001" if it is HCM.	CHAR 10
LFDNR	Number of actual diagnostic. Number from 1-10.	NUMC 3
DKEY1	Code of a diagnostic. Number with the format XXX.YY. Being XXX a number from 001 – 999 and YY a number from inexistent to 99. See ANNEX 9 for all possibilities.	CHAR 30
ERUSR	Name of the person that created the registry.	CHAR 12

Figure A.1: Metadata provided by FCRB for a table with the name Diagnostic

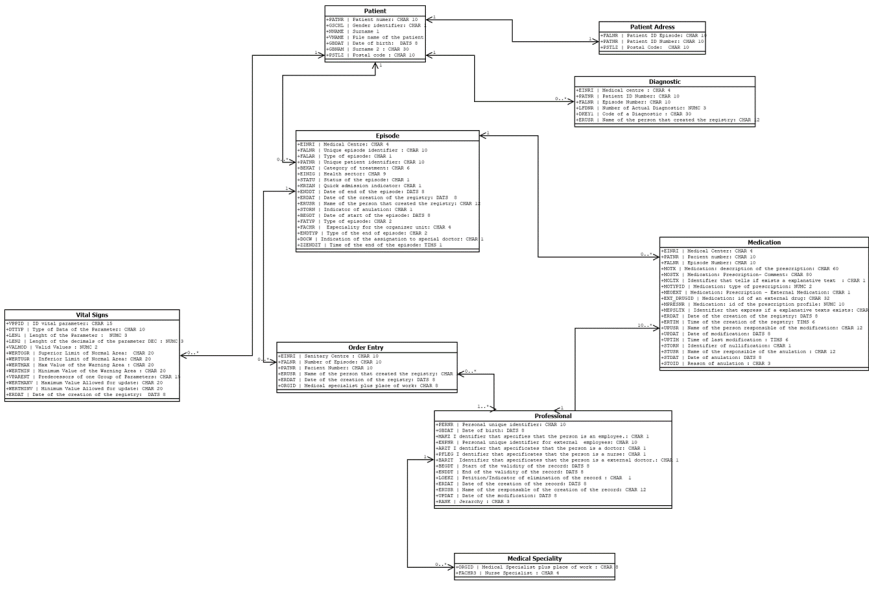


Figure A.2: Entity Relationship Diagram (ERD) provided by FCRB alonside their metadata documentation

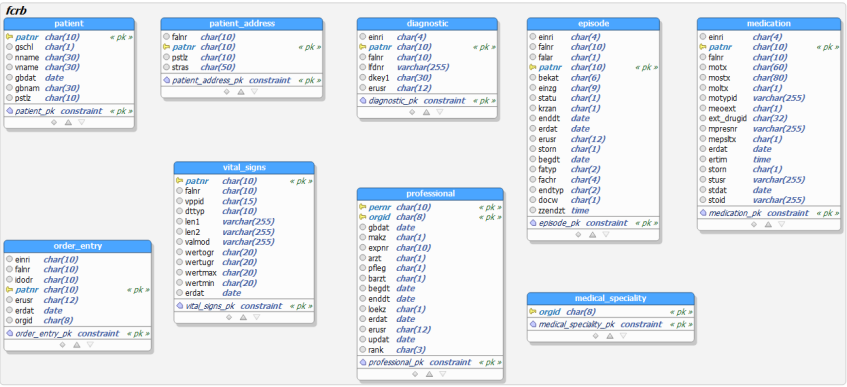


Figure A.3: All of FCRB's original tables modelled in pgModeler

IBM was supplied all of the SQL files, as well as any accompanying documentation for the metadata. In combination, these were used to create the initial version of the fabricated data as part of Deliverable 4.1 (Figure 1). This fabricated data was then passed to both the relevant use case partner and SOPRA, the use case partner to evaluate its accuracy and to begin the process of producing more realistic data, and SOPRA to continue the development of the SPHR. A small sample of some of the fabricated data for the Diagnostic table can be found in Figure A.4.
Key to the Smart Patient Health Record is the use of the data vault structure (Section 4.2.1.2). These were generated for each of the use case partners, again using

einri	patnr	falnr	lfdnr	dkey1	erusr
BCL	1	1000000001	1		
BCL	2	1000000002	1	E78.1	Ditti
BCL	3	1000000003	1		
BCL	4	1000000004	1		Blancas
BCL	5	1000000005	1	I27.2	
BCL	6	1000000006	1	I10	Leyva
BCL	7	1000000007	1	E26.02	Hedrix

Figure A.4: Example of IBM's data fabrication for the Diagnostic table of FCRB's use case

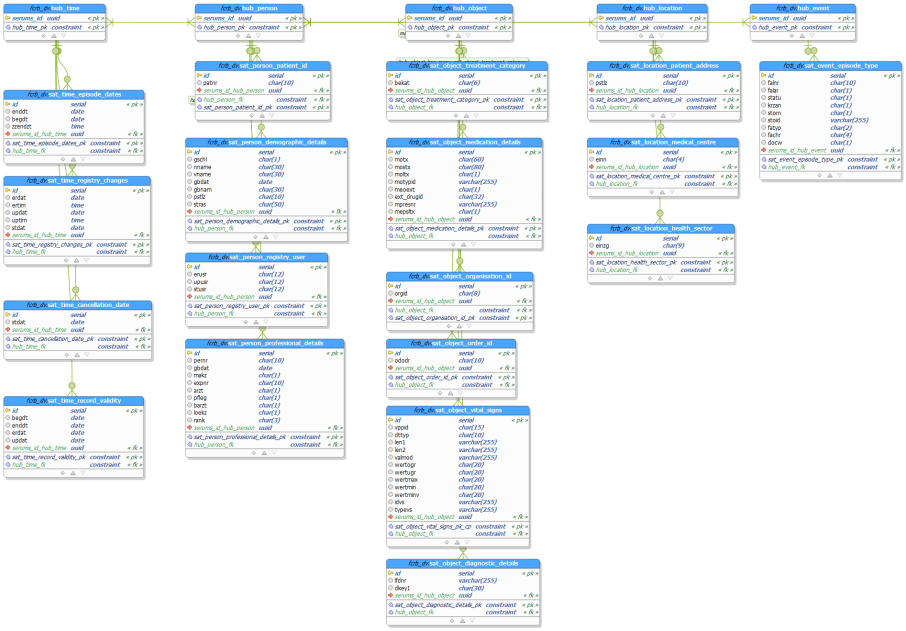


Figure A.5: The data vault model of the FCRB tables

pgModeler. In the case of FCRB it resulted in a construction which can be found in Figure A.5. The top most row shows the Hubs of the data vault with Satellites below each. The source tables are no longer recognisable in this new construction, however none of the data has been lost and the data lineage has been tracked, allowing the user to see from where the data has originated. It is important to note here that the *Satellites do not have an active relationship with any of the other Satellites*, despite what this image may appear to show. They have simply been arranged in this manner to show their relationship to their respective Hub.

At this point, it is now possible to build and fill the data vaults for the use case partners. Firstly, the source tables are created within the **Serums** data lake using the SQL files SOPRA provided to IBM and the fabricated data SOPRA received in return. Next, the SQL file for the data vault structure is ran to create an empty database within the data lake. Finally, a piece of code is executed which takes each of the source tables and automatically maps them onto the data vault structure, taking each row from each table and inserting it correctly into the waiting Satellites, all the while filling in the Hubs and Links to ensure the data maintains its original relationships.

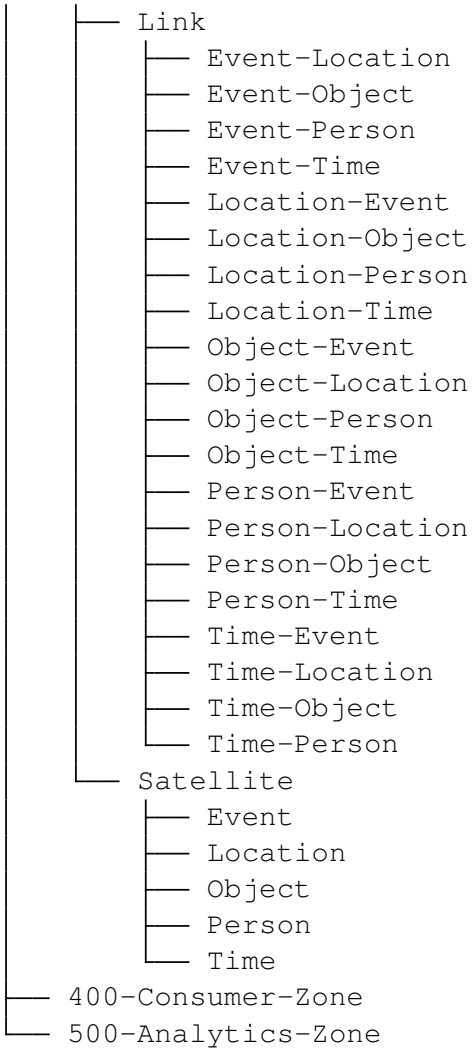
With the data vault complete for each of the use case partners, it is now possible for the Smart Patient Health Record to be generated and safely transmitted via SOPRA's API (Section 4.3.3).

Appendix B

The data lake directory tree

Below shows the complete structure of the Linux file system partition of the **Serums** data lake:

```
100-DL
├── 000-Workspace-Zone
├── 100-Raw-Zone
│   ├── 100-External
│   │   ├── 100-University-of-St-Andrews
│   │   ├── 200-Zuyderland-Medisch-Centrum
│   │   ├── 300-Fundacio-Clinic
│   │   └── 900-Other
│   ├── 200-Internal
│   │   ├── 100-CSV
│   │   ├── 200-TEXT
│   │   ├── 300-JSON
│   │   ├── 400-XML
│   │   └── 900-Human-in-the-Loop
│   └── 300-Archive
│       ├── 100-CSV
│       ├── 200-TEXT
│       ├── 300-JSON
│       ├── 400-XML
│       └── 900-Human-in-the-Loop
├── 200-Structured-Zone
├── 300-Curated-Zone
│   ├── Hub
│   │   ├── Event
│   │   ├── Location
│   │   ├── Object
│   │   ├── Person
│   │   └── Time
```


Link
Event-Location
Event-Object
Event-Person
Event-Time
Location-Event
Location-Object
Location-Person
Location-Time
Object-Event
Object-Location
Object-Person
Object-Time
Person-Event
Person-Location
Person-Object
Person-Time
Time-Event
Time-Location
Time-Object
Time-Person
Satellite
Event
Location
Object
Person
Time
400-Consumer-Zone
500-Analytics-Zone

Appendix C

Interactive documentation for the API

The interactive documentation for this deliverable is available at http://melonlander.co.uk:8080/. It is possible to test out the **/api/get_sphr** endpoint and see real encrypted data being returned from a duplicated version of the data lake.