Managing medical data collection from multiple sources

C.C. Berg
E.A. Rietdijk
P.J.M. Verkooijen

Delft University of Technology

TU Delft

Managing medical data collection from multiple sources

by

C.C. Berg
E.A. Rietdijk
P.J.M. Verkooijen

Authors: C.C. Berg
E.A. Rietdijk
P.J.M. Verkooijen
Project duration: April 23, 2018 – July 2, 2018
TUDelft coach: Dr. Ir. A. Bozzon

TU Delft

Preface

This Bachelor Thesis is written by Chris Berg, Emiel Rietdijk and Paul Verkooijen to conclude the Bachelor Project as part of the Bachelor Computer Science Program. Over the course of three months the project was carried out for Ivido B.V..

We would like to thank everyone at Ivido B.V. for their support during the project, especially Tristan Garssen and Hans Niendieker for their coordination.

Furthermore, we would like to thank our TU Delft supervisor, Alessandro Bozzon from the Web Information Systems Research Group at Delft University of Technology, for his guidance and valuable feedback during the project.

C.C. Berg
E.A. Rietdijk
P.J.M. Verkooijen
Delft, June 2018

Contents

1

Summary

The Netherlands is working on making online health platforms where patients can access their medical data and add, modify or share it. Recently new standards have been set in the Netherlands regarding the sharing of this highly personal data, which includes a standardized structure for sharing called FHIR. These new standards need to be applied by companies that allow their users to share medical information. Ivido is such an online health platform, which aims to retrieve medical data from multiple sources to construct a complete medical overview for a patient. However, with information coming in from multiple sources, duplicate and conflicting data is bound to happen.

This report focuses on a highly erroneous sensitive aspect of medical data, medication. A high number of hospital administrations originate from medication errors. This can be prevented by doing a conflict check on data.

A solution was created, to retrieve data from multiple sources while complying to all Dutch regulations. Additional, an algorithm to find conflicts in data was constructed to help prevent medical errors.

For retrieval of data a class was implemented to make a TLS connection with a central point. By using a white list of permitted sources and a TLS connection, communication can be done safely.

The algorithm finds conflicting data by comparing getter functions in data classes. Any getters returning different values are stored and it will later be determined to be conflicting data or new data.

A user interface was created to show all medication data about a patient. It is possible to review and solve found conflicts and get a current medication overview.

Finally, all personal identifiable data stored is encrypted by AES-256-GCM. This encryption is compliant with Dutch regulations about data storage for medical data.

Testing has been an important aspect of the project, since the data used if personal medical data. For this reason, unit testing was of utmost importance for the project. Eventually a code coverage of 67% has been achieved. Furthermore, a user test has been sent out, but at the moment of writing there has no answer been received yet.

2

Glossary

In this paper a few abbreviations will be used which are not common in the English language, but do appear in the Dutch language, along some generic terms with a specific meaning in this report. Abbreviations without an English counterpart are translated by the writers of the report. This shall be mentioned explicitly.

- **PGO** - Personal Health Environment.
  Dutch: Persoonlijke Gezondheidsomgeving. Translated by the writers.
  A digital tool where personal healthcare data can be collected. From here the owner of the data can share, edit and add data. A PGO can only retrieve a patients medical information when gotten explicit consent from the patient to do so.
- **BGZ** - Basic healthcare information.
  Dutch: Basis Gegevenssetzorg. Translated by the writers.
  A collection of all medical information about a patient, structured in modules.
- **HCIM** - Health and Care Information Module.
  Dutch: Zorg Informatie Bouwsteen.
  A single module from the BGZ, which contains specific information about a patient. There are a total of 18 HCIMs, containing information about medication use, patient details, insurer details, allergies ect.
- **LSP** - National Switch Point.
  Dutch: Landelijk Schakelpunt. Translated by the writers.
  The LSP is a Dutch national infrastructure over which data can be exchanged between two sources. When connecting to another source, the LSP will make sure that connection and data sharing is secure and done correctly.
- **MedMij** - Not an abbreviation.
  MedMij is an agreement system for PGOs. For any PGO to be approved by MedMij and use the LSP, they have to follow and comply to all regulations and agreements.
- **FHIR** - Fast Healthcare Interoperability Resources, pronounced ”fire”.
  A standard to share medical information using RESTful API. FHIR also has a set data structure and all medical information shall follow this structure. When mentioning FHIR in this paper, we explicitly mean the FHIR version which is modified by Nictiz, to more closely follow Dutch healthcare.
- **Ivido/Client**
  A Dutch PGO which focusses on improving a patients health for a long-term. At this company our bachelors end project for the Delft, University of Technology was done.
- **Patients/Professionals**
  When talking about patients and professionals, we explicitly mean patients and healthcare professionals enrolled in the Ivido platform.

3

Introduction

In the Netherlands medical data is increasingly being stored in an Electronic Patient Dossier (EPD). The EPD was introduced in 2008, and since then more plans have been made to improve the Dutch health care with digital technologies. The Dutch government now aims to have Personal Health Environments (PGO: Persoonlijke Gezondheids Omgeving) in 2018 [5]. A PGO is a digital service where people can view, edit, add, or share their medical data with other people.

In the Netherlands, an agreement system was founded to which PGOs have to comply. If they do not comply, they will not be verified as PGO and will not be able to use the perks offered to PGOs that comply. This agreement system is called MedMij. MedMij is an initiative of Patientfederation Netherlands, Nictiz and the Ministry of Health, Welfare and Sports [5].

The medical data of patients is stored on different independent sources. All healthcare related institutions can become a data source. For example, when a patient goes to a hospital, the hospital creates new data of the visit. The hospital now becomes an information source. Another example would occur when a patient uses medication. He now has data about his medication usage, so the patient is an information source. These are only a few examples of sources and many more exist. Hospitals, pharmacies, caregivers, general practitioners and any other health care institution can all be sources of medical data about a patient.

For health care institutions it is possible to share data with other institutions. A general practitioner can send data about medication that a patient needs to use to a pharmacy, so that the pharmacy can give the patient what he needs. Both instances now have the same data. If the pharmacy makes any adjustment, the pharmacy and the general practitioner now have conflicting data. This is a problem when giving a patient an overview of his medical data. With duplicate and contradicting data present in the overview, this can raise medical errors and confusion.

This problem motivates the main research question of this thesis: 'How to construct a data collection algorithm which finds and handles errors in data, while complying to MedMij regulations.' To answer this question a solution was created at Ivido, a Dutch PGO.

The first part of the solution was designing a user interface. Here a patient should be able to view his medical data. After communication with different sources had to be implemented. After retrieving data, an algorithm has to compare records to find duplicate and conflicting data from different sources. This report will give an overview of the process and the development of the solution. All design choices, problems and solutions are covered in this report.

4

Problem definition

This chapter will give more information about the problem. First, an introduction will be given about the client. Second, a detailed problem description will be given. Finally we will talk about the problem analysis. This will contain the MoSCoW model for the project.

4.1. Client

Online healthcare, also known as e-health has a rising popularity. Ivido is a Personal Health Environment (PGO, Persoonlijke Gezondheidsomgeving) which aims to improve patient health on long time. Special courses to improve health have been for among others chronic pain patients, pregnancy support and positive health.

These courses connect patients and professionals in a personalized health care environment. Ivido constructs a personal health dossier, which contains medical information about the patient. The patient can choose to edit, add or share any data freely with others.

4.2. Problem description

This section will give an introduction to the problem, stating how this problem came to be a problem. Then the problem will be described, with all tasks that should be solved.

4.2.1. Background information

The Netherlands tries to go live with PGOs this year. But for companies to own personal medical data, there should be regulations. All regulations are stored in the agreement system MedMij. MedMij contains rules for sharing and storing medical data. A PGO is required to follow MedMij regulations.

One of the rules MedMij constructed is that all data sharing should be done with the Fast Healthcare Interoperability Resources (FHIR, pronounced ”fire”) standard [7]. This standard states how data should be structured when being shared. The structure is based on resources, which contain some information. A small example of a FHIR structure is given in figure 4.1. As can be seen in the figure, the FHIR standard is based on JSON. It is also possible to share the data in XML, but both languages keep the same data structure. This standard is modified by Nictiz, a Dutch company specialized in e-health. These modification were made to be more applicable to the Dutch health care system.

For a PGO to retrieve data from a patient, it will do the request for data at the National Switch Point (LSP, Landelijk SchakelPunt). Within this request the source where the information can be found and about whom the information is about are defined. The LSP will retrieve data from the requested source about the given patient and translate the data to the modified FHIR structure. Finally, it will send back the information to the PGO.

```
"medicationReference": {
  "reference": "http://vonk.test-nictiz.nl/Medication/medmij-medication-medication-09",
  "display": "Simvastatine tablet fo 20mg"
},
"subject": {
  "reference": "http://vonk.test-nictiz.nl/Patient/medmij-medication-patient-01",
  "display": "dhr. P. Jansen"
},
"authoredOn": "2014-07-03",
"requester": {
  "agent": {
    "reference": "http://vonk.test-nictiz.nl/Practitioner/medmij-medication-healthprofessional-01",
    "display": "H. Huis"
  }
},
```


Figure 4.1: An example of the FHIR structure. Contains information about medication, the patient, the professional who requested the data and the date it was requested on.

All medical information about a patient is collectively called Basic Healthcare Information (BGZ, Basis Gegevensset Zorg). The BGZ is divided in 18 different building blocks. These blocks are called Health and Care Information Modules (HCIM, in Dutch: zorginformatiebouwsteen). Each HCIM follows a predefined FHIR structure and contains information about a specific subject. To give some examples, there is a block for patient information, a block for a patients insurance company and a block for patients medication usage information (fig. 4.2). A medication information block can for example contain information about usage, but also when the medication should be used and its dosage.

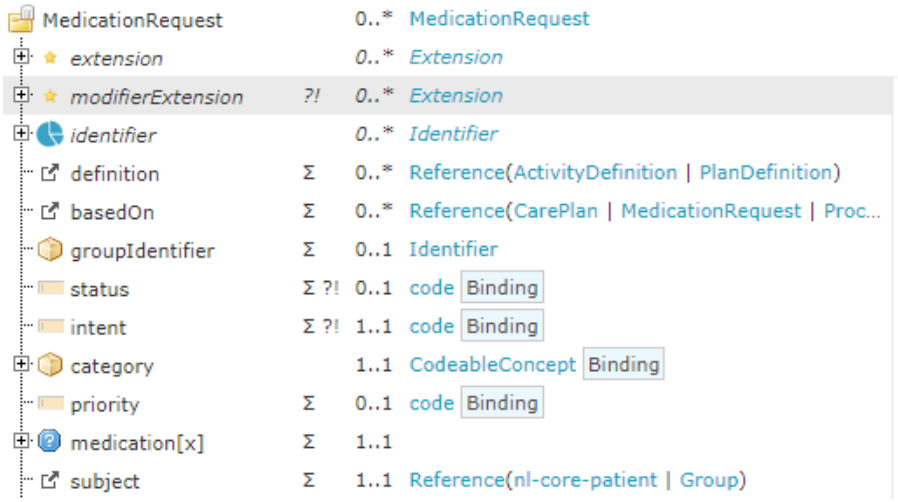


Figure 4.2: The HCIM about a medication request. The first column shows which resource is stored in the HCIM, the second column shows its flag. The third shows the cardinality of the resource. The last column shows the data type of the resource.

4.2.2. Problem

Currently data is stored at health institutes. Hospitals, therapists and pharmacies create their own data about a patient, but also share with other institutes. It is expected that data can be duplicate or be conflicting with other data when retrieving data from a source. To clarify how this can happen the following example is given. A hospital determines that a patient needs medication and shares a medication request with a pharmacy, so that the pharmacy can give the patient his medication. Both the hospital and the pharmacy store the data and

now duplicate data has been created. If the pharmacy would add additional information, a conflict is created. The hospital and the pharmacy own similar information where both institutes believe they have the correct information.

Although this problem might sound unlikely for some, this is a large problem in healthcare. The Dutch HARM (Hospital Admissions Related to Medication) report states that it is estimated that about 19.000 people yearly go to the hospital due to a preventable, medication error [8]. This source originates from 2006 and is therefore a bit old, but no newer information about medication errors is found. The Dutch Consumers Association however, states that it is estimated that 40.000 people go to yearly due to preventable medical errors [2]. However, no indication is given on how much of these errors are due to medication.

Ivido has requested a solution for this problem. This problem motivates the research question of this report. *How can a data collection algorithm which finds and handles errors in data be constructed, while complying to MedMij regulations.*

To solve this problem and answer the research question, the following five requirements have to be met.

1. It is of high importance that the solution complies with MedMij regulations. If this is not done, Ivido will not be allowed to communicate with the LSP and thus cannot retrieve any medical information.
2. Data should be able to be collected from different sources. This data follows the FHIR structure, modified by Nictiz.
3. New medical data should be compared with existing data. Conflicts should be found.
4. Data should be stored in the database of Ivido. Since it might become a requirement for PGOs to send data to other healthcare instances, the data must be stored in a way it can easily be reconstructed.
5. An user interface must be developed where all data can be viewed and conflicts can be solved.

4.3. Problem analysis

In this section our initial analyse of the problem is done. This analyse was primarily done during the research phase and resulted in a MoSCoW model. This model can be found at the end of this section.

4.3.1. Prove of concept

The technology and regulations that is used to create our solution are still in development. This has quite some implications for our product. The largest problem is that there is no knowledge of how scenarios will play out. There is no knowledge about common mistakes in the FHIR structure or how the load will be on the database. Due to these unknown factors, we tried to make all design choices with a worst case scenario in mind. This means we expect conflicts will occur and we should take responsibility for preventing mistakes, even if it becomes clear in the future that these mistakes will never happen.

4.3.2. Project boundaries

When looking at the BGZ, it becomes clear how large the required implementation is. There are 18 different HCIMs and when counting all sub-blocks of a HCIM, the total blocks to be implemented becomes larger than 30. With the limited time and programmers available, the team and client agreed to focus on a single HCIM. The HCIM that shall be implemented contains the information about medication. This HCIM is one of the more complex and larger HCIMs. It gives an overview of all medication a patient uses and must be implemented with care.

The medication block is divided in three smaller sub-blocks, namely the medication agreement, the medication dispense and the medication statement.

The medication agreement is the agreement to use a certain medication. This is often bound to a medical treatment and is active for a longer time. This sub-block contains amongst other things information about the agreement, who requested it, for whom it is, what medication and the period of use.

The medication dispense is about the dispense of medication. A dispense for prescription medication, will be part of a medication agreement in most cases. This sub-block contains amongst other things information about what medicine, who received it, how much was given to the patient and instructions for dosing the medicine.

The medication statement is about usage of medication. For medication that is bought at pharmacy, it will almost always be part of a medication dispense. This sub-block contains amongst other things information about who gave the medicine (caregiver or patient himself), if it was taken, reasons for taking or not taking the medication and a date and time on which the statement was done.

4.3.3. MedMij regulations

Following these regulations is important, due to Ivido needing acceptance of MedMij to communicate with the LSP. These regulations were a basis of some of the design choices the team had to make. MedMij restricts certain design choices by giving different options where the developers can implement the most suitable. There are regulations about encryption, communication, legal issues and more. When MedMij regulations influenced a design choice it will be mentioned explicitly.

4.3.4. Data collection

Data collection is the starting point of solving the problem. According to MedMij, a secure connection with the LSP must be made using TLS. All communication is based on the RESTful API. To communicate with the LSP, there are no other options than using TLS and using REST. When Ivido makes a request for data from a source, it must do the request to the National Switch Point (LSP, Landelijk Shakelpunt). The LSP will retrieve the data from the correct source. All data shall be in the modified FHIR structure.

4.3.5. Conflict checking

At the time of writing, there is no agreement about which party should remove duplicate data. This could be done by both the LSP and the PGO. For our project to be future proof, the team and client decided that the PGO should take this responsibility. There are several possibilities when receiving new medication data and comparing it to existing data. First, the data is new data and should be stored in the database. Second, the data is duplicate data and should not be stored. The last option is that data about the same medical treatment contains conflicting information with existing data. Conflicts could be different dosages of medication or differences in usage periods.

4.3.6. Data storage

The data should be stored within the platform of Ivido. However, in the future it should become possible for a PGO to also send data to LSP, instead of only retrieving it. This means, that data should be stored in a way it can easily be retrieved to and converted to the FHIR standard required for communication.

4.3.7. User interface

Finally, there should be an interface where patients and professionals have the opportunity to retrieve data, view retrieved data and solve any conflicts. Sometimes the conflict is solvable by the patient, but more often a professional will have to solve the conflict. Only very few conflicts should be solved automatically by an algorithm. This is due to the fact that neither Ivido nor the team should not be accountable for mistakes made during solving.

4.4. Moscow

After the analysis the team was able to create a MoSCoW model. First a use case will be given, to illustrate the problem that should be solved. Then a fitting solution will be given, which the team is capable of doing.

Must haves

M.1 **Use case** Ivido must implement the BGZ.
Solution Due to time limitations, the medication HCIM from BGZ shall be implemented. This is the most complex and should be a good basis for following HCIMs.

M.2 **Use case** Ivido must follow MedMij regulations.
Solution To follow regulations in our product, data encryption and communication with LSP the TLS protocol should be used.

M.3 **Use case** Ivido must be able to retrieve data from a source.
Solution Ivido will make connection with the LSP according to the RESTful API with TLS protocol. For now, the LSP is not live yet so an TLS connection shall be made with the test server of Nictiz.

M.4 **Use case** Two different sources have shared patient information.
Solution When Ivido retrieves medication data all new files should be checked to not be duplicate data. Duplicate data shall not be stored.

M.5 **Use case** A professional has unknowingly made an mistake while filling in medication data for a patient.
Solution When retrieving medication data all new files should be compared with existing files. Any conflicts should be found.

M.6 **Use case** A patient wants to see his currently used medication.
Solution In the my dossier page of Ivido, a new information block will be shown with medical information that is currently used.

M.7 **Use case** A patient wants to see all medication data.
Solution The information block on the my dossier page, shall have a button to redirect to an extended view. This contains all information about past, current and future medication agreements, dispenses and statements.

M.8 **Use case** A patient wants to see found conflicts.
Solution In the extended view, there shall be a tab with the existing and the new, conflicting zib next to each other. The conflicting fields in the zib are highlighted, so that the conflict can be easily determined.

M.9 **Use case** A Professional wants solve a conflict.
Solution In the conflict tab, there shall be an option to choose the correct fields. These changes shall be logged and stored.

Should haves

S.1 **Use case** One record has additional information over an other record.
Solution The additional information shall be combined with the existing data, but only in risk free situations. This risk free additional information includes notes, instructions for usage of medication and information about length or weight of the patient.

S.2 **Use case** It might become possible to not only get, but also to post data via the LSP.
Solution All data should be stored in a way it can be easily reconstructed to match the requested FHIR structure.

S.3 **Use case** A patient wants to share medication information, but not his usage.
Solution An interface to choose what sub-block of the Medication HCIM can be shared and retrieved through the LSP. This can be extended to choosing specific information within a sub-block.

S.4 **Use case** A patient wants to access more Basic Health Information.
Solution Other important HCIMs should be implemented.

Could haves

C.1 **Use case** A patient wants to access all Basic Health Information.
Solution All HCIMs must be implemented for this.

C.2 **Use case** A lot of large HCIMs are stored in the database.
Solution A document store can be used to store HCIM data.

Won't haves

W.1 **Use case** A patient wants to send or retrieve his medication data to another source not connected to the LSP.
Solution For communication that does not follow the MedMij regulations, new APIs have to be made.

W.2 **Use case** A professional wants all conflicts to be solved automatically.
Solution Only trivial problems should be solved automatically. These include the two records holding additional information in the should haves. Solving a conflict might harm the patient when done by a poorly designed algorithm. An AI might be able to decide conflicts.

5 Design

This chapter goes into more detail about the design choices made during the project. First the design of the database will be discussed, followed by the user interface.

5.1. Database

This section will give more information about the architecture of the database. First there will be given some background information about what data, in which quantity and what operations we can expect to be performed. Second the design of the database will be given and described.

5.1.1. Expected data

When Ivido launches, it may facilitate 10.000s of users, both patient and professionals, in a realistic scenario. The database should be able to support the amount of data belonging to those users. An average patient in Ivido could have in the 100s of medical records stored in the database within the first year, which results in more than 1 million Health and Care Information models (HCIM) records in the database. These numbers are an estimate determined with our client.

5.1.2. Design

Design choices

Each HCIM is always accessed a a whole and it is expected that it will be rarely updated. For this reason each HCIM is stored as an encrypted blob in a dedicated relation in the database. This also improves scalability in the future, as this mechanism supports transformation of the relation into a key value store, which could be maintained on a distributed system. Furthermore, by storing the HCIM as a whole, it becomes very easy later on to send these HCIMs to the LSP. Downsides to this approach include data redundancy, as overlapping values in a HCIM are stored multiple times this way. It also risks data corruption, as structural integrity of the HCIMs can not be ensured by the database. These two downsides would be solved by splitting the entire HCIM into relations. This would however yield a major load on the server, as every time HCIMs are requested the entire HCIM has to be reconstructed.

Every time a call is made to the Nation Switch Point (LSP, Landelijk Schakel Punt), a time stamp can be given when the last call was made about that user regarding the type of HCIMs. This saves up a lot of time determining what HCIMs already are in the database. For that reason a table is made where the last request times are saved for each user regarding the source and which HCIM.

Furthermore, each HCIM is checked on conflicts before it is entered in the database. Once such a conflict is founds and its severity is determined, as described in chapter 7.2, it is necessary to save this data into the database to solve the conflicts later on. For this reason

there is a table created that saves each conflict. The new entry in the HCIM entries table, the HCIM that caused the conflict, refers to the representing conflict entry, which holds all necessary information about the conflict, including a severity id. This id refers to the severity table that holds the information what the level is of the severity and what user should solve the conflict.

UML

Below a UML diagram is given of our database (fig. 5.1). Within this figure all used tables are shown. Any tables that created by the team during the project are inside a box called 'BGZ medication'.

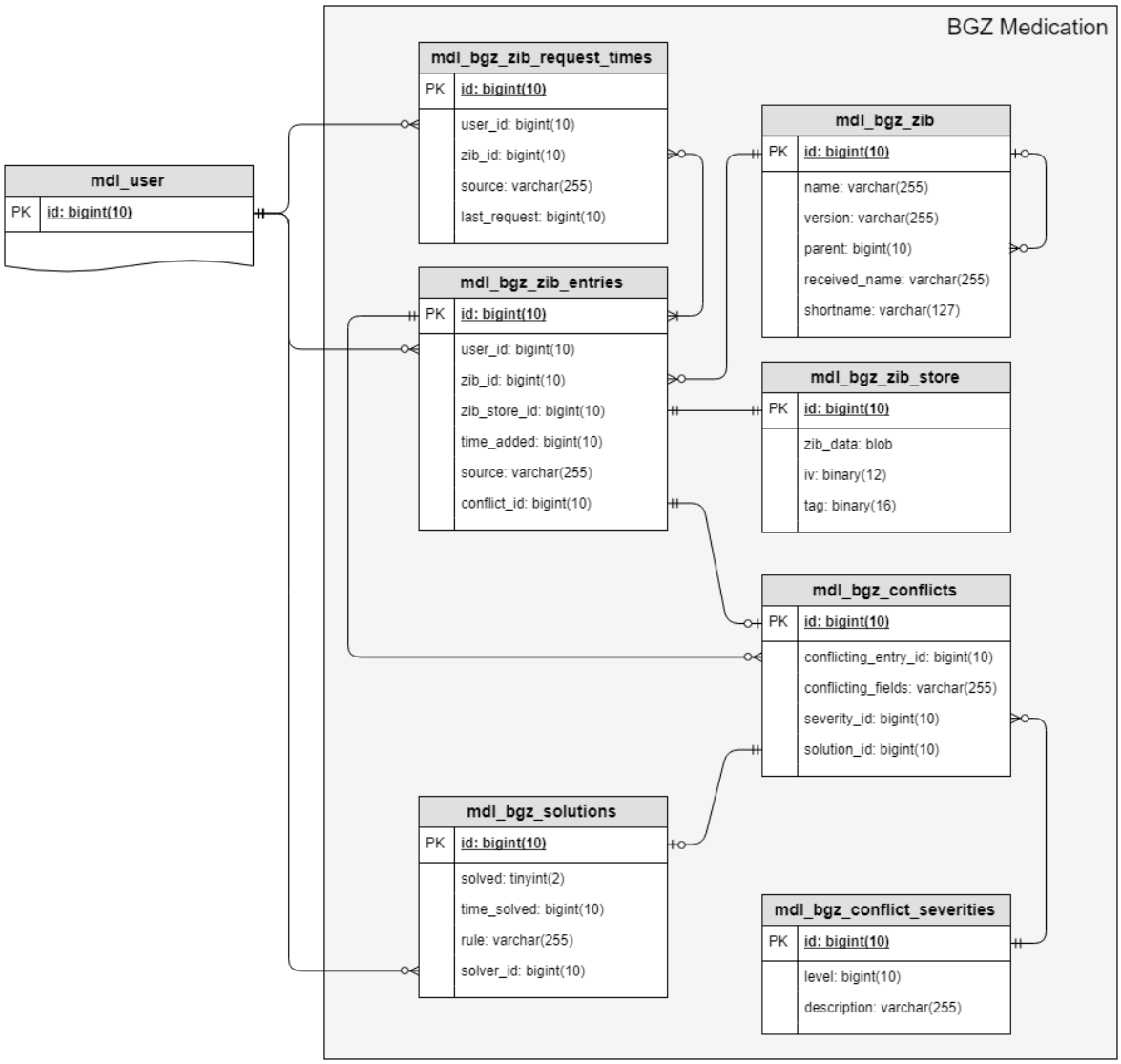


Figure 5.1: Database ER diagram, the prefix 'mdl_' is used for all Moodle related database elements, and the prefix 'mdl_bgz_' is used for all BGZ Medication related database elements.

Database normalization

Database normalization is a technique of organizing the data in the database. By normalizing the database issues like data redundancy, inconsistency with insertions, deletions and updates are avoided. For this reason we decided to normalize our database as much as possible. Below is explained step by step which normal form we ended up with and how we achieved it [3].

First Normal Form
The first normal form (1NF) only allows attribute values that are single atomic values, meaning that the fields in the database can not be split up into smaller pieces (e.g. name into first name and last name). Within our database design the only possible fields that contain more than an id or a word are fields that contain a description or are the encrypted fields. These fields can not be split into smaller pieces. For that reason our database is at least in 1NF.
Second Normal Form
The second normal form (2NF) states that every non-prime attribute of the relation is fully functionally dependent on the primary key in the relation. In other words: there should be no partial dependencies. Within the database design of the HCIM there are 4 tables that got more keys that just the primary key. In these 4 cases there are unique keys. It is made sure that these unique keys do not determine parts of their representing database table. The other tables that only have a primary key are automatic in 2NF.
Third Normal Form
The third normal form (3NF) states that there should be no transitive dependencies. In other words, a non-prime attribute should not depend on other non-prime attributes. We've made sure that all attributes of an entry are depending on a prime attribute of that entry. If a transitive dependency did occur during the design process, a new table was created that replaced the transitive dependency in the original table. The original table then had a reference to the new table.
Boyce and Codd Normal Form
The Boyce and Codd normal form is an extension of 3NF and states that for a dependency from A to B, A should be a super key. The dependencies in our database are all to other table rows, which means that A is always the super key. Therefore our database is in BCNF.
Fourth Normal Form
The fourth normal form (4NF) states that tables should not have any multi-valued dependencies. This means that there should be no relations between A en B where B can have multiple different values for the same A. Within our database design we managed to work around such cases by putting everything that possibly had multiple values for A for into an array. This array then is encoded to JSON and saved in the database. This way it is ensured that the database is in 4NF.

5.2. User interface

The Ivido application uses a block-like interface structure, where distinct parts of the platform are treated like isolated plug-ins. This design choice is made by our client to enable different health care institutions to provide a specific set of functionalities of Ivido to their clients. We used this design principle of Ivido to create the plug-in for collecting and managing data from medication HCIM: the BGZ Medication plug-in.

5.2.1. Environments and activities

The plug-in consists of two environments: the medication block (Figure 9.1) and the medication page (Figure 5.5). The primary activity of the medication block is checking which medication the patient has to take. Within the medication block it is also possible to get more information about the medication and to access a visual medication leaflet called the Kijksluiter (Figure 5.3). Before the medication data can be displayed, the patient has to give Ivido consent to access his or her data. When this hasn't happened yet, the entire medication block is dedicated to notifying the patient to give that consent (Figure 5.4). After the patient has granted Ivido access to all medical data required for the plug-in to function, the data will be shown in the medication block.

The medication block provides access to the medication page. The primary activity of the medication page is viewing the entire medication process and assisting in solving possible conflicts concerning medication data. Solving conflicts is done by displaying the existing HCIM on the left and the conflicting incoming HCIM on the right (Figure 5.6). The data sources are displayed on top to indicate trustworthiness. After the correct HCIM is selected,

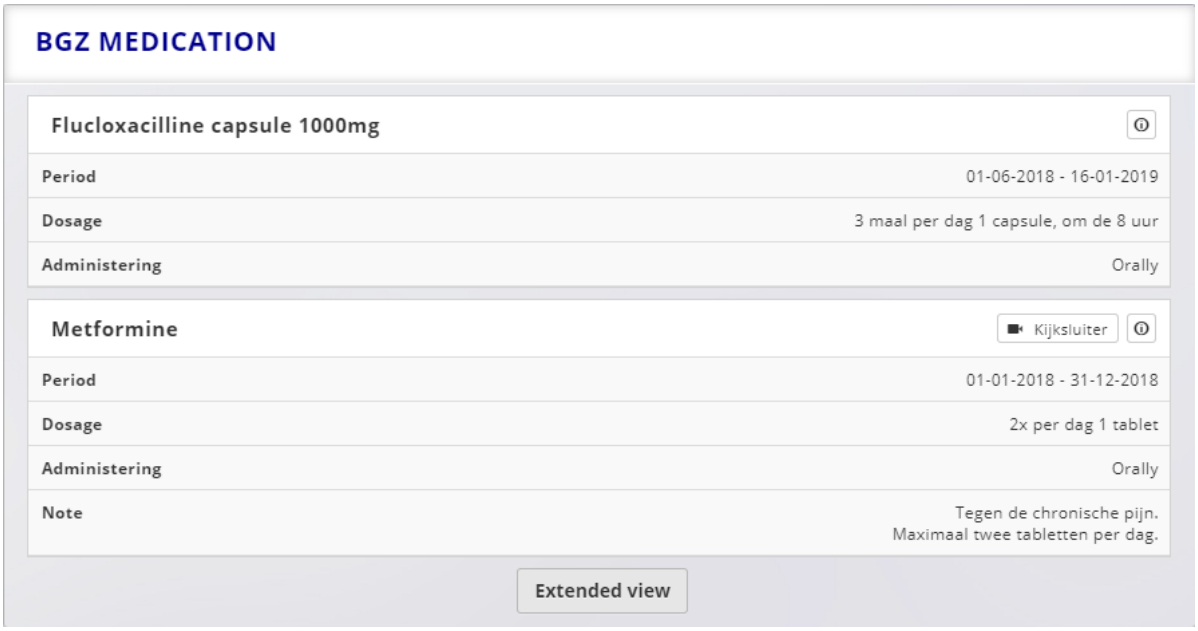


Figure 5.2: BGZ medication block, as found on the personal health page.

the conflict can be solved by clicking the bottom right button.

5.2.2. Design guidelines

For people used reading from left to right, leftmost text is interpreted as being more important as rightmost text This has been stated in, among others, the work of Eviatar [4]. As this application will be used only by left-to-right readers, visual elements will be placed in the following order from important to less important: from left to right and from top to bottom, where left-right is the more dominant direction. Elements concerning time will be treated in the same way, but from old to new. This raises the following issue: in most cases, older data will not be the more relevant data. Therefore objects concerning relevance and time should never be placed within the same element, unless one property can be ordered from left to right and a second from top to bottom. Relevance of objects within an element will always be treated over chronological order of the objects unless the subject of the element is time itself, e.g. the case of the element being a time line.

Examples of this ordering in order of priority are found in the medication block: clicking the details and Kijksluiter buttons is not part of the primary activity, these buttons are therefore placed on the right. In case of it being unclear how the medication has come about after interacting with the medication sub blocks, the patient could press the 'Extended view' button at the bottom of the page, leading him to the BGZ medication page (Figure 5.5). On the conflict page, the conflict solution submit button is found at the least important place: the bottom right. This stimulates the user to first check the solution before marking it as solved.

5.2.3. Usage of colour

The application of Ivido itself has blue background colouring, therefore we will design our elements in white and light grey tones to improve readability of the plug-in. Colours will be used to highlight elements which should be recognised throughout the plug-in such as whether the displayed medication element originates from a user, pharmacist or hospital. Recognising elements throughout the plug-in is aided by the use of icons. Icons indicating the same subject should therefore have the same colour and shape throughout the entire plug-in. Colour will also be used to acquire the user's attention, such as marking conflicting fields when solving conflicts.

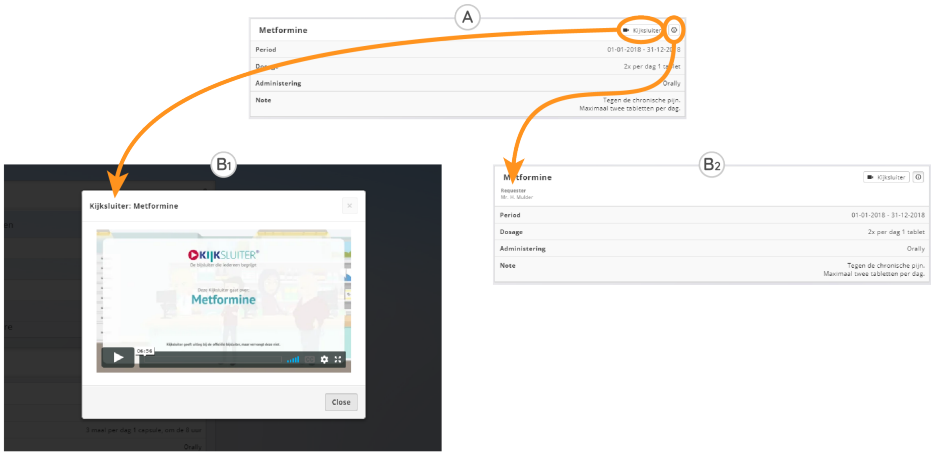


Figure 5.3: Ways to get more information about a medication (A) A medication sub block as displayed in the BGZ medication block (B_1) Showing the Kijksluiter pop-up (B_2) Showing extra information regarding this medication, should this be available.

5.2.4. Future improvements

At the moment, there are objects of two different subjects displayed in the same dimension (left to right) within a single element: the tab panel. These subjects are the chronological order and correctness of the data. A solution to this is to implement a single tab for correct HCIMs and a single tab for conflicts. A user should be able to filter out past, present, future medications tab pane for correct HCIMs.

When the plug-in is used as intended, the patient won't be checking out the medication page daily. The patient needs to be drawn to the medication page when there is a conflict, therefore there could be a notification badge displaying the number of conflicts (if any) on the 'Extended view' button to draw the user to the medication page.

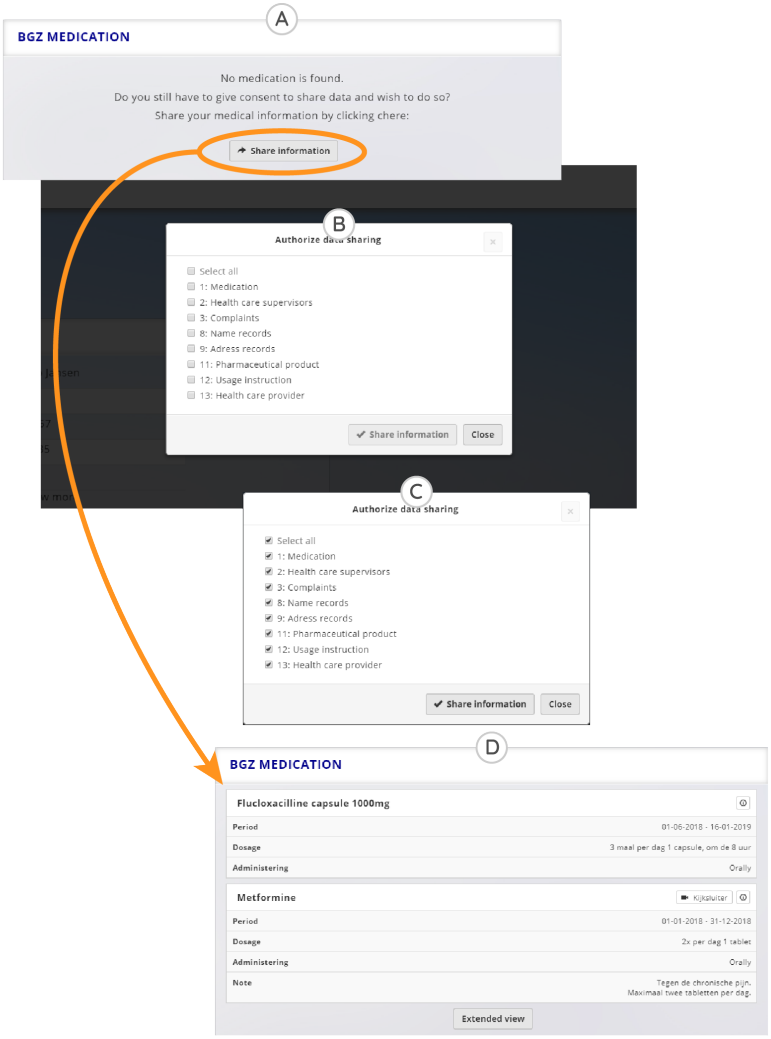


Figure 5.4: Data sharing, where the patient gives consent to Ivido to access to his or her data. (A) The initial BGZ medication block when the patient hasn't yet shared his or her data. (B) (C) The share pop-up, the patient has to grant Ivido access to all of these HCIMs before the medication information can be shown to the patient (D) The BGZ medication block showing the medication of the patient.

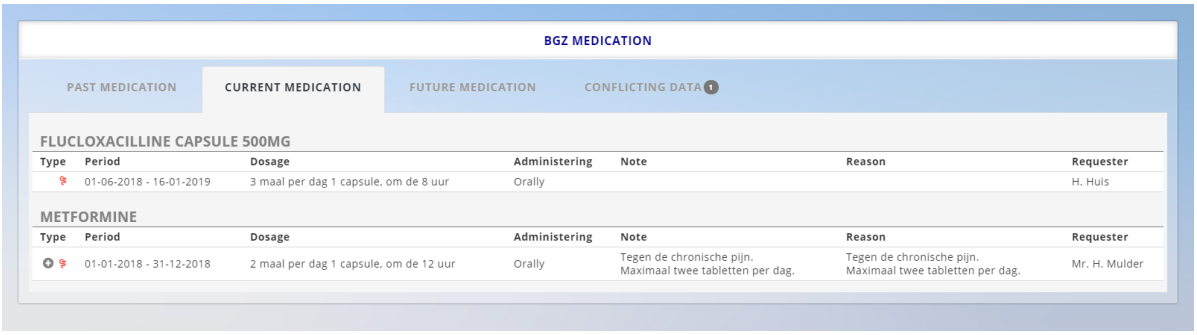


Figure 5.5: BGZ medication page, as found when pressing 'Extended view' at the bottom of the BGZ medication block. Note the unsolved conflicts bubble next to the conflicts tab to draw the user's attention to unsolved conflicts.

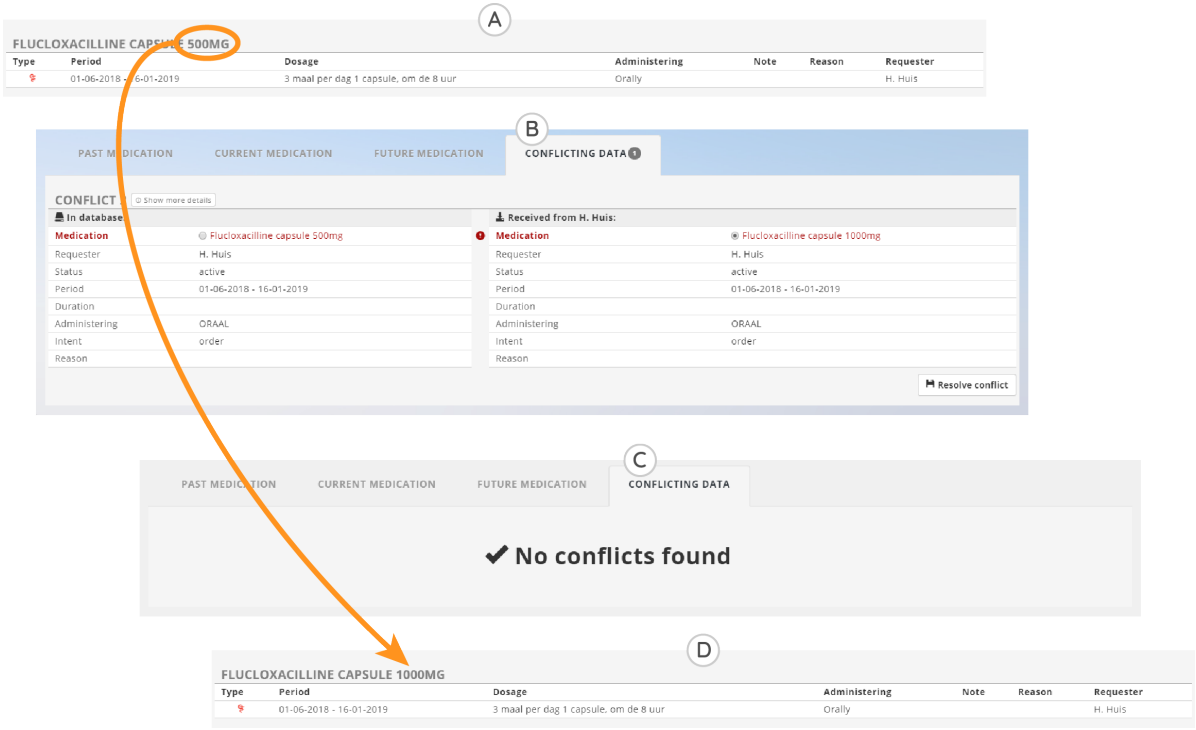


Figure 5.6: Process of conflict solving: (A) Existing medication data. (B) Conflict tab showing conflicting data, on this tab page the correct HCIM can be selected. (C) No conflicts remain after the conflict solution has been submitted. (D) Current medication showing updated medication data.

6

Data collection

This chapter covers all information about how we implemented communication with other sources and store the data and storing the data. First, a flow of how data is retrieved is given. After that, the requirements for implementing communication will be given. Finally, the implementation will be discussed.

6.1. Communication

Data has to be retrieved from several different sources. All data collection goes through the Dutch National Switch Point (LSP). Before a PGO can use this, MedMij requirements have to be followed. In figure 6.1 it can be seen how communication works. First, a patient gives consent to Ivido to retrieve data from a certain source. Let this be a hospital, a caregiver or any health care institution. The request is send to the LSP. The LSP then retrieves the data from the source and creates a FHIR structure of the data if needed. This data is send back to Ivido. Ivido then does a conflict and duplicate check. Finally, it makes the data human readable, so that the client can view all his medication data.

To verify the patient requesting data DigID is used. This is a Dutch identity management platform used by the government. This step should be added as soon as the user gives consent to Ivido to retrieve data. This can be done by a login request to the LSP. After that, logging in and verifying the patient is done by DigID. This structure is still in development and thus cannot be added to our product. The product can be extended when the DigID technology becomes available.

6.2. MedMij requirements

For sending request to the LSP, MedMij has created a few requirements. These requirements guarantee security for communication. These requirements are strict and no design choices could be made for communication with the LSP. The requirements are as follows:

1. Communication must be done via a TLS connection.
2. The REST API is used, as the FHIR standard prescribes.
3. The source and the LSP must be white listed at the requester. Sources not white listed should not be communicated with.
4. Both the requester of data and the source must be white listed in the LSP.

The LSP is not running yet, but should become active in the near future. However, a similar connection can be made for testing purposes with Nictiz, a Dutch company specialized in e-health. This source is being referred to as the Nictiz endpoint. The Nictiz endpoint is not as complex as the LSP, but to make our solution usable in the near future, our implementation will use the Nictiz endpoint as it would use the LSP.

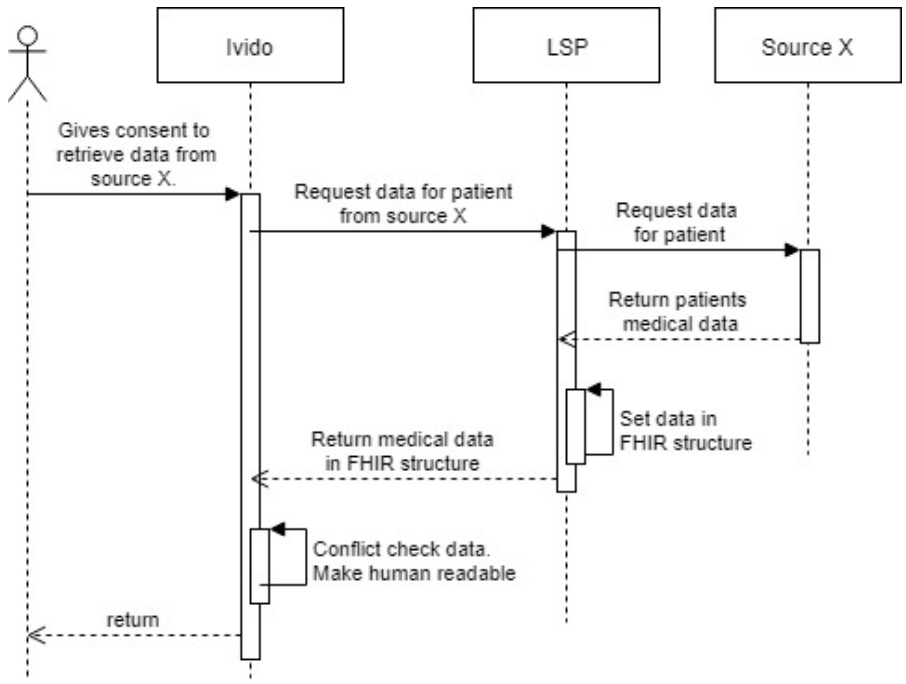


Figure 6.1: Flow of the data retrieval.

6.3. Implementation

For realizing a TLS connection, it was decided to use the free library OpenSSL. This library not only supports TLS 1.3, but also contains a general purpose encryption library. OpenSSL was chosen for two reasons. First of all, it is used in the Ivido platform for several applications. This keeps our product consistent with existing code and easier to understand or refactor for Ivido developers if needed. Secondly, OpenSSL supports encryption needed discussed in chapter 7.1.

A class with static functions is created, which makes calls to the Nictiz endpoint using a TLS connection. This class automatically constructs an correct request and communicates with an URL selected from a file. This file contains all white listed sources and endpoints. With this simple structure, the first three requirements are checked; communication is done on a TLS connection while using REST API, REST is used and only sources and endpoints are selected from a white listed file. The final requirement, white listing Ivido, cannot be implemented by us and must be done by the LSP. However, with this application all requirements are, met making it possible for Ivido to be white listed.

6.4. Remarks

Currently it is unknown if Ivido will have consent to continuously retrieve data when consent has been given by a patient to retrieve data. This is due to the fact that patient has to verify he has given consent with DigID, which cannot be done automatically. It can be possible that a user must log in ever so often to retrieve new medical data. Our implementation requests data a single time, but can be extended easily. More on this can be found in chapter 12.

Finally, for each data retrieval at a different source, a new call has to be made. This is due to the fact that a patient must give explicit consent to Ivido to retrieve data from a source. Mass retrieval from multiple sources would not be allowed, but if it ever will be it can easily be implemented. It can be done by looping over the white list of sources where also the LSP endpoint is found. Then, for each source data can be retrieved without breaking regulations by communicating with unverified sources.

7

Data management

This chapter handles all functionality that happens in the back-end of our product. This includes the encryption of stored data, followed by the conflict finding algorithm. Finally, the conflict solver shall be discussed.

7.1. Data encryption

One of the requirements for MedMij, is that all stored data is encrypted on disk- or database level with an algorithm approved by the National Cyber Security Centre (NCSC) [6]. The NCSC has approved 4 different encryption and hashing algorithms. During the research phase the different options were researched. The four algorithms that could be used are:

- RSA
- AES GCM
- SHA
- ECDH(E)

For these algorithms only some version are allowed to use. For example, RSA must have a key length of at least 3072 bits or AES should have a block length of either 128 or 256.

7.1.1. Choosing encryption

First of al, we had to consider the goal of the encryption. A patient in the Ivido platform should be able to retrieve his own data. Afterwards, the patient can choose to share this data with other patients or professionals. The fact that data can be viewed again by patient and professional on the Ivido platform, rules out the usage of SHA. Hashing data would prevent the viewing of the original data.

Secondly, the choice between asymmetric or symmetric encryption had to be made. The data should be decrypted within a second when someone want to see it. Speed is here of high importance, since patients and professionals do not want to wait several seconds to see their data. Therefore, the choice was made to go with AES encryption, since with this algorithm decryption can be done more quickly than with other algorithms.

Third, the choice had to be made between AES-128-GCM and AES-256-GCM. A script was created to test the speed difference of the two algorithms. AES-128 was expected to be about 40% faster due to the way the algorithm works. It makes 40% less passes on the data during encryption compared with his 256-GCM counterpart. However, this was not noticeable in the test script results. On average, both encryption methods had the about the same runtime. We expect that the larger blocks size AES-256-GCM uses to encrypt their data is the cause for the little runtime difference. AES-256-GCM should have 50% less blocks and with encrypted

data being several KB's large, this difference will mean several hundred blocks less have to be encrypted and decrypted. The speed testing script also showed that decryption of 10000 records took about 0.02 seconds. The client does not expect to have that many records to be decrypted for a single patient. If in the future patients will have more records, loading times of the data would increase linear. In cases of the loading time getting to slow, new solutions have to be thought of. More on this in chapter 12.

Finally, the security difference of AES-128-GCM and AES-256-GCM is little. Both are expected to be extremely safe, when used correctly. AES-256-GCM has one advantage in the security for quantum computing. AES-256-GCM is encrypted with twice as large blocks. Decryption by brute force would take longer for AES-256-GCM, thus this variant is safer against quantum computing. This is however, not a save solution when quantum computing with a large amount of qubits is realized.

To conclude, no speed advantage is found in any of the AES-GCM algorithms with our usage. AES-256-GCM should be more secure than its 128 counterpart, specifically when looking to the future and quantum computers. While AES-128-GCM is a good and secure choice, it offers no advantage over AES-256-GCM in our application. With AES-256-GCM being more secure and more future proof, it is the better option of the two.

7.1.2. Implementation and recommendations

To implement the encryption in our back-end we used the library OpenSSL. This library is also used for setting up a TLS connection, as shown in chapter 6. PHP 7+ is needed for AES-GCM encryption. Ivido meets these requirements. New encrypt functions have been developed, where classes can be send to the encrypt function. The decrypt function handles decryption of a complete database record, so no data manipulation has to be done to decrypt data.

AES-GCM uses an initialization vector and authorization tag to decrypt the data. These are constructed and returned in the encryption function. These can be stored in the database without restraining the security. One thing to mention is that using the same initialization vector for encrypting two different records can hurt the security. Some files might be retrieved partially when initialization vectors are reused. However, this is not expected to happen in the near future. Our recommendation is to modify the key with patient specific details. This creates a patient unique key. Due to the fact that encryption happens on application level, this can be realised with ease. For example: the existing key can be concatenated with some characters of the patients BSN, since this patient information will never change. This way, all patients enrolled in Ivido will have a different password to encrypt their data. With more passwords it is less likely that the AES-256-GCM algorithm will create duplicate initialization vectors on the same key and security is maintained.

Finally, it should be noticed that all data is encrypted with a single key. It is of high importance that this key should be stored safely. This means, it should not appear in anywhere in the public html folder of the platform. Therefore, it was decided to add the key, a pseudo random generated character string, to the config.php file. This should result in the option to use different passwords on the different, available environments (local, test and live).

7.2. Conflict finding

This section is about what conflicts exists and can solve these conflicts. The algorithm behind the conflicts will be briefly explained as well.

7.2.1. Conflict use cases

At the start of the project, all possible and realistic conflicts that could occur in data were created. It is unknown if these conflicts will happen when going live, since no real data is available yet. However, these conflicts could harm patient safety when happening. There-

fore, these conflicts should be spotted. All conflict cases are validated by our client. During the coding phase of the project, these conflict cases were used to create the conflict finding algorithm. All conflict cases are based on the idea that the two records being compared both have the same parent. This means, that both records are bound to the same Medication Treatment or Medical statements are part of the same Medication dispense. Medication dispenses should be part of the same Medication request.

It is also possible for a record to be conflicting within itself. This happens when a certain information field is missing or incorrect within a retrieved Health and Care Information Module (HCIM).

All conflict cases are in the following form: First an explanation of what conflict could arise, then who should be responsible for solving a conflict, which indicates the severity of the conflict. Solving the conflict can be done by reviewing it in the conflict tab, as seen in chapter 5.2. When a professional must solve the conflict, only professionals with access to the patients data should be able to do so.

Conflict case 1:
Conflict: Two records are compared and contain different periods of use or usage duration. None of these records have an indication of which is more recently updated.
Solved by: A professional, since wrong values can harm patient safety. When one value is more accurate (e.g. it contains a date and time, while the other record only contains the date), it can be solved automatically.

Conflict case 2:
Conflict: A single record is missing a status, which indicates if it is active, cancelled, intended ect.
Solved by: A professional, without a status it is impossible to know if medication should be used or if it is cancelled.

Conflict case 3:
Conflict: Two records have different dosages or medication, yet no indication about which records is more recently updated.
Solved by: A professional, this is a huge mistake that could seriously harm the patient.

Conflict case 4:
Conflict: One record contains supporting information, such as patient height or weight, or additional notations.
Solved by: The system. This shows only additional information and wrongly adding this data could not harm the patient.

Conflict case 5:
Conflict: Two medication request records have different dates on which the data was authored to be used.
Solved by: A professional. False data cannot harm the patient but only a professional knows when exactly a request was authored.

Conflict case 6:
Conflict: A records contains information about an issue that has been detected.
Solved by: This is unsolvable, but must be highlighted to the patient.

Conflict case 7:
Conflict: On two medication dispenses, different receivers of medication are found. No indication of which record is newer is present in both records.
Solved by: This cannot be solved, but must be highlighted to notify the patient.

7.2.2. Conflict finding algorithm

The conflicting finding algorithm has a couple of steps. It compares new records with existing data. Any non duplicate data will be stored, due to the fact that the conflict should be stored for logging purposes. The algorithm compares all new records to all existing records of the same type. Since in a single request to the LSP we only retrieve data from a single source, we assume the source will not have conflicts in their own data. However, if in the future it would become clear that conflicting data at a single source is a possibility, the function running the algorithm can easily be extended to compare the new records with themselves.

First, the algorithm confirms that two HCIMs compared are about the same subject. If they do not have the same parent, there is no reason these HCIMs would conflict. They just contain different data. If the records are about different subjects, the comparison is skipped.

Next, we know that two records are being compared are about the same subject. Now there are three different options. The first one is that data is duplicate, the second one is that it contains conflicts and finally data can be new data, which overwrites the old data.

If the records contains information about which is newer, we can assume the newest data overwrites old data. However when this is not the case (time construction indicators are not required data in the FHIR structure), the real conflict checking can begin.

All information fields in the HCIMs are compared by calling all the getters in the HCIM class. All getters return a standardized value when no information is set, so no errors can occur during comparison of data. If the information in a field is in the form of an array or an object, all fields in the structure are compared to determine differences. When the getter returns different values for two existing HCIMs, the getter is added to an array. This array will be returned at the end of calculating differences.

When no differences are found, it can safely be said that the data is duplicate. The new data is exactly equal to data already stored in the database, so this record does not have to be added. Duplicate data does not have to be logged or saved, so the comparison to all other HCIMs in the database is interrupted. The next newly retrieved HCIM can be compared now.

When differences are found, there is no way of knowing which data is correct, since time indications are missing in this step. The getter methods that return different data for two HCIMs, are stored in a class. This class determines who should solve the conflict by calling private functions in the class equal to the names of the getter. For example, in conflict case 3, the method get_dosage_() returned different values. It is determined that a professional must solve the conflict. If any conflicting information field found before should be solved by the user or the system, this gets overwritten and all conflicts should be solved by the professional.

Finally, when the solver has been determined and it has been determined which information fields conflict between two HCIMs, the data is stored. The new record is added, for logging purposes, even when it contains conflicts. The conflicting fields are stored in the database, so that when the data is viewed, it can be retrieved without making calculations.

7.2.3. Runtime

The runtime of the algorithm is heavily dependent on how data will be retrieved. As explained in chapter 6, data can be either retrieved continuously or a patient must give consent each time data is retrieved. For now, all data will be retrieved a single time. Both options can be implemented in our functionality but will influence the runtime of the algorithm.

The runtime is determined as follows. A new record is compared to each record in the database of a similar type. Since there are two options of retrieving data, a continuous or a sporadic retrieval, the amount of retrievals should be added to this calculation. With the amount of data retrievals, the average of new data should be used. In the worst case, the

new record has to be compared with all data in the database by comparing all get functions.

$$O(retrievals * data_{avg-new} * data_{database} * functions) \rightarrow$$

$$O(r * d_{avg-new} * d_{database} * f)$$

Since the amount of functions f compared is a constant based on the type of HCIM retrieved:

$$O(r * d_{avg-new} * d_{database} * f) = O(r * d_{avg-new} * d_{database})$$

We know that there cannot be more data stored in the database for a patient than there exists unique data for that patient on all sources combined. This is due to the fact that our function removes duplicate data. So, $d_{database} \leq d_{all}$. This results in the following runtime:

$$O(r * d_{avg-new} * d_{database}) \leq$$

$$O(r * d_{avg-new} * n) \text{ where } n = d_{all}$$

The equation contains $r * d_{new}$, which multiplied give the total amount of new records. This covers both data retrieval options, continuous and sporadic. Continuous will have a higher amount of request l. Sporadic retrieval will have a higher average retrieved data m. Since it is not possible to retrieve more data than there exists, $r * d_{avg-new} \leq n = d_{all}$. Finally we can thus conclude that a worst case runtime would be:

$$O(r * d_{avg-new} * d_{database}) \leq O((r * d_{avg-new}) * n) \leq O(n * n) \leq O(n^2) \text{ where } n = d_{all}$$

The total time spend retrieving a certain amount of records and finding conflicts will be the same for both options. But in the case of continuously retrieving data, the conflict finder will spread the runtime over multiple periods, whenever new data is retrieved. When getting data is done once every time the patient gives consent the amount of data that has to be compared is larger and will take more time in a single period.

As a final remark on the runtime, for the application it would be preferable to retrieve data continuously. With sporadic retrieval, data will only be retrieved when the patient gives consent. The runtime will start the moment the patient gives consent. The runtime can take quite long due to a quadratic runtime and the patient must wait a longer time before he can view his data. With continuous retrieval, the runtime is spread over multiple smaller periods. When a patient views his data, it will already be loaded and no extra waiting time is needed.

7.2.4. Extending the algorithm

As a final remark, the conflict checker can easily be extended. For example, a check can be implemented to see if medication from a newly retrieved HCIM has negative reactions in combination with another medication that is currently used by a patient. Within the conflict finder this is easily implemented and might prevent medication errors. More on this can be found in chapter 12

8

Ethics

Medication data is one of the most secured types of data. It should be, as leaked medication data could be used to discriminate people based on their health. As the jurisdiction for handling of personal data is quite strict, it is a requirement that data is stored in a safe and anonymous way. All regulation regarding personal data aside, there are some ethical issues regarding the handling of medical data when showing data to a patient and automatically solving conflicting medical data from different sources. These ethical issues are discussed within this chapter.

8.1. Responsibilities

In order to minimize the risk of errors in the code leading to security breaches, the classes responsible for handling the personal data should be sufficiently tested. Our responsibility as programmers is that the line coverage should be at least 90% or higher on the code that deals with the medical data. In the next chapter (chapter 9.2) we will elaborate our practices regarding testing.

Furthermore, it is our responsibility that the automatically solved conflicts have a negligible risk regarding the patients health. Such instances are adding additional information or adding a note to a list of notes. This kind of information won't affect the medication description and therefore has no negative impact on the patient. For any other conflicts that might occur it is not our responsibility to solve the conflicts, since patient safety is at risk and therefore should a professional of a patient solve the conflict.

8.2. Accountability of mistakes

One important aspect to note about our product is that it does not produce conflicts, it is created to find conflicts in data. Errors made by administrations or health care professionals or irregularities between different information sources are found by our algorithm and the algorithm does not create conflicting data itself. Our product might miss human created mistakes, therefore the issue of who would be responsible for errors still remains. In the Netherlands, a health care professional or health care institution can be held accountable for mistakes made in the medication process [1]. An automated conflict finder that does not find conflicts, might be held responsible for making mistakes, just like a health care professional. In our opinion, the health care professional is still responsible for making mistakes, not the tool which he uses (our product). There is, however, no literature that supports this claim, nor is there literature that states the other way. This is probably due e-health and PGOs being fairly new and little research has been done on conflicts that could arise.

For conflict solving, other rules will apply. Some conflicts should be solved automatically, while others are solved by health care professionals. Mistakes can still be made when solving the conflict. Due to some conflicts being solved automatically, showing a patient the wrong

medication overview can be easily blamed on the people who built the application. Therefore, only risk free problems are solved automatically. This brings no accountability to Ivido or its developers.

In the case of an expert solving these conflicts, it will become quite easy to trace back whether a health problem of a patient was due to incorrect solving of the problem. This will make the health care professional accountable, but this might hinder the process of creating a sound medication overview for the patient. Specialists might hesitate to resolve conflicts when they are to blame for mistakes. On the other hand, cases of clear negligence of a medical specialist towards a patient can be proved more easily. In these cases, the medical specialist should be held responsible for his actions.

In the future, it might be preferable for our client to solve more than risk-free conflicts automatically. In this case, accountability should be shared with Ivido and the creator of the conflicting data. Bad implementations for conflict solving can result in risks for the patient, yet due to the wrong data created the patient was already in increased risk. When more automatic solving is implemented, there should be a clear set of rules in place which we use to solve conflicts in the data. This set of rules should not be verified by the client, but by health care specialists and scientific proven rules. This reduces the chance of automatically solving conflicts in an incorrect way.

8.3. Data storage

As required by MedMij, but also to protect patient privacy and medical data, all fields containing information about the patient are encrypted with AES-256-GCM. All implemented databases are anonymous and all personal medical data has been encrypted. If by any chance the data would be able to be decrypted by an outside source, the data becomes identifiable. Sadly, there is no other way to store the data and therefore security must be treated as highly important. More information about how security was implemented in our product can be found in chapter 7.1.

9

Deployment and Testing

As discussed in the chapter 8.1, we work with one of the most sensitive types of personal data. This requires us to minimize the risk of coding errors in elements of the application handling this data. In this chapter we will discuss how we will make guarantees about the quality by discussing its deployment and testing of our product.

9.1. Deployment

The product is to be used in both the testing and live environments of the Ivido platform. Our solution can be seamlessly integrated into the testing environment because of the modular structure of the platform. Here non-functional tests can be performed to confirm the usability and stability of the product. Before our product can be deployed on the live server of Ivido, the National Switch Point (LSP: Landelijk Schakel Punt) has to be able to process our requests and be able to return health care information using the FHIR protocol. The ability to receive and process that data is already implemented on our side.

Possible issues our product, especially the conflict finder, could run into while running on the live environment can occur when actual patient data is retrieved from the LSP. As our product is currently based on theoretical data errors, it is currently unknown whether and how often conflicts will occur. These testing conflicts might differ from real situations with medical data from patients Ivido has enrolled on their platform. Testing the usability of the conflict solver can be performed using real patient data, as our product is created in such a way that only duplicate data will not be stored. Thus, when new conflicts would occur, our product would still store the new information and personal data will not be discarded. In case of the uncovering of more conflict cases that should be dealt with automatically, we implemented the conflict solver in such a way that new conflict types are easily appended to the existing set of conflict types by merely adding a new function to an array of conflict solving functions.

During the coding the functionality of PHP 7 was used to its potential. Most functions have return types and use type declaration. This is used to make sure that when something goes wrong, a function will still return something useful for the function that called it, in order to prevent data loss.

When the product will be deployed in a live environment, we are sure no data will be discarded unless the algorithm is 100% sure it is duplicate data. The only thing that could happen is that new conflicts, conflicts that are not illustrated in chapter 7.2, will not be marked as conflicts. This will not result in an increased risk for a patient, due to the conflict not being created by us. Our solution can only increase patient safety, by finding conflicting and contradicting data. All conflicts and contradictions have been made accidentally by health professionals, which without our product would not have been spotted at all.

9.2. Testing

Due to handling medical data, our product needs to be tested thoroughly. This section provides information about how our product was tested.

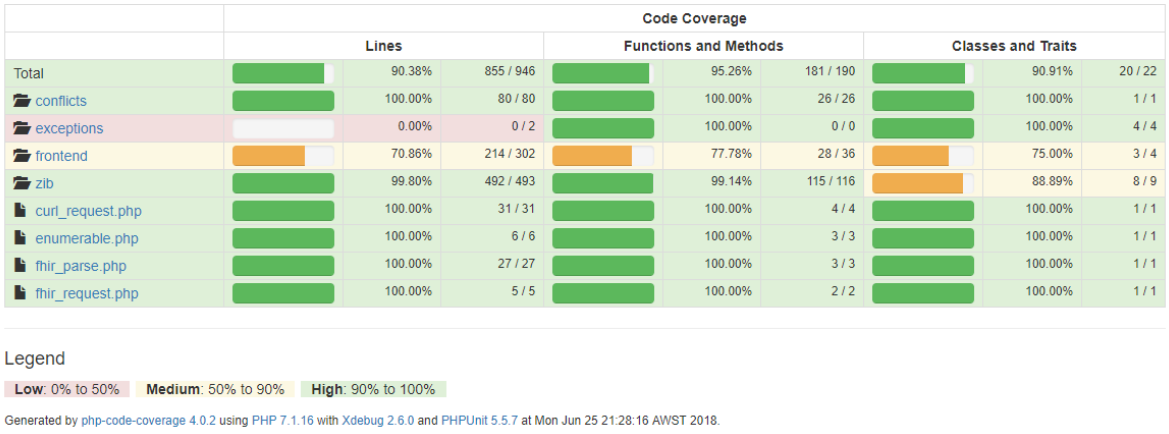
9.2.1. Environment

As our testing environment, we have made use of the build-in testing environment of Ivido: PHPUnit. Testing shall only be done on a local environment and testing files will not be published in the public html folder of the platform, as they support the arbitrary creating, requesting and deleting of data.

9.2.2. Unit tests

Since the back-end of the application largely deals with medical data, it is of utmost importance that these functions are flawless. Therefore, PHPUnit testing was used to test functionality. Testing was done on all functions and classes that do not rely on database reads. Testing the database required to install a local testing database each time adjustments were made within the test database files, which was time consuming due to the large database Ivido has. Therefore SQL queries were tested manually with injecting test data in the database and retrieving it with the query.

All functions that did not make database reads or writes, were tested. For functions that use database records, the records were constructed as how they would be read from the database. Due to handling sensitive data, the team aimed for a high code coverage. The overall code coverage is almost 70%. All classes except for classes that produce HTML and other front-end code were tested with a 100% coverage. Since the determination of conflicts and its solving is largely done in classes, these functionalities are also 100% covered by tests.



	Code Coverage								
	Lines			Functions and Methods			Classes and Traits		
Total		90.38%	855 / 946		95.26%	181 / 190		90.91%	20 / 22
conflicts		100.00%	80 / 80		100.00%	26 / 26		100.00%	1 / 1
exceptions		0.00%	0 / 2		100.00%	0 / 0		100.00%	4 / 4
frontend		70.86%	214 / 302		77.78%	28 / 36		75.00%	3 / 4
zib		99.80%	492 / 493		99.14%	115 / 116		88.89%	8 / 9
curl_request.php		100.00%	31 / 31		100.00%	4 / 4		100.00%	1 / 1
enumerable.php		100.00%	6 / 6		100.00%	3 / 3		100.00%	1 / 1
fhir_parse.php		100.00%	27 / 27		100.00%	3 / 3		100.00%	1 / 1
fhir_request.php		100.00%	5 / 5		100.00%	2 / 2		100.00%	1 / 1

Legend

Low: 0% to 50% Medium: 50% to 90% High: 90% to 100%

Generated by php-code-coverage 4.0.2 using PHP 7.1.16 with Xdebug 2.6.0 and PHPUnit 5.5.7 at Mon Jun 25 21:28:16 AWST 2018.

Figure 9.1: Line coverage of classes at almost 100%. Lines not covered in exceptions are file include statements, not testable as the exception class itself is empty. The few lines missing in the zib class are caused by a function that is only enabled when the application is not running on a local environment.

As a remark, it must be said that the 100% code coverage on classes might be a bit higher than the actual value. This is due to the fact that in the null coalescing operator was used a lot. The null coalescing operator (??) is an operator which replaces the statement of 5 lines:

```
if (isset($x)) {
  return $x;
} else {
  return /* some default value */;
}
```

by a single line of code:

```
return $x ?? /* some default value */;
```

While the result is exactly the same, PHPUnit will mark the single line notation as tested as soon as one of the two options is fulfilled. This means that when a line is marked as covered by tests, only one branch might actually be covered by testing. With the first statement, both the if and the else branches have to be covered to get a complete code coverage with testing. Our classes contain a lot of these statements, coding them in a single line has a positive effect on the readability, which we prioritize above compromising the meaning of a high degree of code coverage. We have tried covered both cases of this one line implementation, but we might have overlooked some of them.

9.2.3. User test

As described in chapter 5.2, a user interface was created where a patient has an overview of his or her medical data. A user test has been performed in order to confirm that the user interface of our product is allowing the user to perform the required activities. However, this user test is deducted at the end of the project by our client. Ivido has a vast base of voluntary test users, which means that it might take a while before such a user test is executed. Therefore, the user test results will arrive after finishing the report.

A prediction of which results the user test will yield, is that the medication page will be more popular under health professionals who are more familiar with the entire medication process of the patient. For the patient on the other hand, it is expected that the medication block will be more popular as this block displays which medication has to be taken when and how often. Next to confirming that the required tasks can be performed by the users, the goal of the user test is to find more usages of the product. For example, whether patients expect more information to be shown on the medication block and page. Results of the user tests might be implemented in the final week of coding, but will most likely have to be implemented by Ivido developers after this project has been finished.

10

Process

Within this chapter the process of the project is discussed. This covers all resources and working methods that were used. First SCRUM will be discussed, followed by all development resources. Finally a reflection of the project is given with at the end a personal reflection of each project member.

10.1. SCRUM

During the entire project SCRUM has been used to maintain an agile work flow and have a clear view on the status of the project. The team chose a cycle length of one week, with a sprint meeting at the first day of the week.

10.2. Development resources

Multiple different resources were used during the project that helped with the development. This section will go into more detail about each one of them.

10.2.1. GitLab

GitLab was used as the version control tool during this project. It is the tool the company already used, so it was easy for the team to embed the project within the companies GitLab repository. A bep_master branch was made from the original master branch to separate our project. From this new temporary master branch a bep_develop branch was made. From this branch all other branches were made for each individual assignment. Before a branch was merged into our bep_develop at least one team member had to review and approve the merge request. More complex merge requests had to be reviewed and approved by all team members.

10.2.2. Asana

The tool Instagant, which is an extension of Asana, was used to keep track of the process during the project along Asana. Instagant creates a Gantt graph of your planning created at Asana. At the beginning of the project a major outline was created as an estimation what was done in which week. Furthermore, each week during the SCRUM meeting on the first day of the week more specific tasks were added for that week together with a estimated duration and the personal responsible for the task.

10.2.3. Google Drive

Google Drive was used to store all related files. All notes of the meetings, design ideas and UML diagrams can be found within the drive.

10.3. Reflection

10.3.1. Project reflection

The SCRUM process ended up having a good effect on the team. Every week it was clear what each member had to do the upcoming days and it was easy to see if the project development was on schedule or not. The Instagant tool was used the most, since it gave a good visualization of all deadlines, relations between tasks, and which tasks were finished and/or overdue. The major outline on the beginning of the project was not very specific, but the main outline was pretty accurate. The only thing that changed up was that the user face implementation was done earlier in the project and the conflict algorithm was moved up a week or 2 due to meetings that gave more insight to some decision rules.

10.3.2. Personal reflections

Chris Berg

In the beginning of the project, I did mostly back-end work and took care laying the foundation of for example the HCIM class structure. But after the first few weeks, my primary activity shifted to front-end work. Front-end development and caring about user experience is something I am experienced in and prefer doing. Collaboration with Paul and Emiel during the project went great. Together with Paul I had a few long and heated discussions, where the bottom line mostly was that we agreed on the subject but just were talking over each other's heads. Later on in the project these discussions became scarcer, as we understood each other better. A great property of the group was that we did not hesitate to address each others mistakes. The ability to all out criticise the work of peers without this leading to tensions between group members is in my opinion the indication of a healthy group, which is able to produce a stable final product.

I learned the most during conversations with people from outside our project. Because, as an IT student, we very much live in a bubble of IT specialists. Talking to people from outside our industry about project-related issues was therefore quite a learning experience. In the beginning of the project, communication during meetings with our supervisor from the TU Delft, Alessandro Bozzon, was not very well prepared and therefore not very efficient. We could have get more value out of those first meetings should we have taken a more leading role in the conversations, taking responsibility as owners of the project. We realised this midway through the project and spend more time preparing meetings with our supervisor, which improved a lot in quality in return.

Emiel Rietdijk

At the start of the project I knew the two other team members, but have collaborated with Paul a lot more than with Chris. This gave some minor issues at the beginning of the project regarding communication, but within two weeks these issues were gone. Also, we all told each other what our skills were regarding the project. This ended up having a very positive effect on the work flow in the upcoming weeks, since we knew each other capabilities and limitations.

I started working on the database design together with some back-end development, since these were the fields I was the most experienced with. Later on I switched to front-end development, which was one of my weaker fields and brought up more difficulties than the back-end development. However, with the help of Chris I eventually learned a lot about the front-end development. Finally I was working on a mixture of both front and back-end.

During the project I learned a lot more about front end development and enjoyed attending the meetings with the different companies that our project was collaborating with. It gave a way better perspective of how the final product should eventually become and how it should be used. In any future projects I would prefer to have the same work sort of structure and roles, including attending meetings.

Paul Verkooijen

At the start of the project, I was used to work projects with Emiel. I had no prior experience with working with Chris. This made the teamwork in the beginning a bit difficult, due to both trying to be the leading force of the project. After the first few days, it became clear we both had good communication skills, but we just had to take a bit more time to listen to what the other had to say. From the second week, it became a lot more clear how our skills were divided, who liked to do what and communication went almost flawless in my opinion. i feel like we all learned from each other. There was a really good vibe and work environment.

I primarily worked in the back-end of our product, since I enjoy that the most. During the beginning of the project I've made a basis for the front-end, but I do not enjoy doing it as much. Chris and Emiel took over the front-end development, so that everyone worked in both the back-end and the front-end. I have the feeling everyone did about an equal amount of work and everyone has taken part of the conflict finding algorithm.

The thing I learned most about are client and coach meetings. I have had client meeting before and know decently how to communicate with a client with less technical knowledge. However, at the start we were a bit sloppy with our coach meetings. By spending so much time researching our object we just assumed Mr. Bozzon understood what we were talking about. From the first meeting on, our communication with Bozzon has improved greatly and I would like to thank Mr. Bozzon once more for his patience and good coaching.

11

Conclusions

Over an period of 10 weeks a solution was found for the problem of how medical data can be collected whilst finding and handling errors in data. This has to comply with MedMij regulations. Our final product will be used to answer the sub-questions created in chapter 4.3 and it will be compared with our MoSCoW model given in chapter 4.4.

First of all, the problem had to be solved while all regulations and technologies surrounding it were still in a prove of concept. While this has made some problems a bit more difficult, like determining realistic conflicts that could occur in data, our product has been made in such a way that it follows all regulations as they are stated at moment of publishing this paper.

A project boundary was set so that only the medication Health and Care Information Module (HCIM) had to be implemented. This HCIM is expected to be the most difficult HCIM to implement, yet this was still realizable in the time span of the project. The coding was done without any prior knowledge on the FHIR structure, MedMij, or conflict finding algorithms. Some of HCIMs contain data that is unlikely to conflict, such as insurance details or patient details. Also, our algorithm should be usable for the other HCIMs. We expect that all other HCIMs can be implemented within the time span of half a year, but can be faster with enough prior knowledge about FHIR and MedMij regulations.

MedMij has made regulations about storing and sharing data. Ivido now complies with these regulations and these are usable for other HCIMs. For both data collection and data storage, the library OpenSSL is used. For collecting data, OpenSSL is used to set up a TLS connection with an endpoint. For data storage, OpenSSL is used for it's AES-256-GCM encryption. Functions developed for encryption and collecting can be used for all existing HCIMs and is thus scalable.

Conflicts are found by comparing the new retrieved the data with data stored in the database. When it is found that they are about the same subject and medical treatment, they will be compared for conflicts. After the comparison data will be marked as new data, conflicting data or duplicate data. Both new and conflicting data will be stored. Duplicate data will not be stored for trivial reasons. Conflicts will be marked and a severity will be calculated to determine how and by who the conflict can be solved. A low severity means it can be solved automatically and a high severity means a health professional must solve the conflict.

The constructed user interface makes it possible to solve conflicts and view data retrieved from multiple sources. It is possible to show only key information about active data, but also to get an extended view of past, current and upcoming medication agreements.

In table 11.1 is an overview of what prior determined tasks we managed to complete dur-

ing the project. As seen in the table all must haves have been completed, along with some should haves.

To conclude, it is possible to create a data collecting algorithm for medical data while finding conflicts. OpenSSL can help with encryption and communications, while the conflict checker should compare records as a whole. There are other ways, but since this is medical data a lot of regulations have to be followed and freedom of design choices is limited. Finally, it should not occur that data is falsely discarded, so any solution should thoroughly be tested.

Table 11.1: Table containing the finished MoSCoW goals

ID	Description
M.1	Implement the HCIM medication
M.2	Comply with MedMij: encryption and communication
M.3	Retrieve data through RESTful API and TLS
M.4	Duplicate data check
M.5	Conflicting data check
M.6	Block view on my health page
M.7	Medication page view with medication filtered on past, current and future
M.8	Conflict view with highlighted conflicting fields
M.9	Solve the conflicts
S.1	Additional information should be automatically solved
S.2	Data can be reconstructed to the FHIR structure
S.3	Patient can choose which data to share with company

12

Recommendations

To conclude this report, a few recommendations are made about our work. The product is finished and can be deployed, but improvements can be made. Here are a few recommendations we see as great additions for the next couple of years.

12.1. Design

The design of the user interface of the extended page (fig. 5.5) focusses on medication agreements. This contains information about medication dispenses, which in turn contain information about medication statements. User tests have been done, but the results will arrive after finishing this report. We expect patients would like to see an extended overview which focusses on what times or which days medication should be used. Our recommendation is to add role based interfaces. Simply check if the user accessing the extended view is the patient himself or a health professional whom the data has been shared with. Then show the medication data in a way most preferable for the user role.

12.2. Database

The database table 'bgz_conflicts' could be improved to be more accessible. Now, a new Health and Care Information Module (HCIM) record points to the conflict table which has a pointer to the existing, conflicting data. This is primarily done since this makes the automatic front-end construction for both the overview and conflict solver easy. However, if at any time an interface will be developed where only conflicts are shown and no other data, the table in its current form is not optimal. It is probable that health professionals will only review conflicts and ignore the rest of a patients medication view. Our recommendation is to change it to be more optimal for retrieving only conflicts. The database is easily changed by adding a pointer to both conflicting HCIMs. The back-end needs more changes, in all functions regarding storing and retrieving conflicting data.

12.3. Add drug interactions

As stated before in chapter 7.2.4, more conflict checks can be added to spot negative drug interactions. Ivido has access to more medical data, such as patient gender, age, allergies, alcohol consumption and other health care related information. All this data can be added to improve the conflict checker. Some medication cannot be used with certain allergies, alcohol consumption or other medications. Ivido has access to this data, and with consent of the patient it should use it. Then, the algorithm can find negative drug interactions for a patients before the medication would be used and increase patient safety. Our recommendation is to find an API which contains all negative drug interactions and add a check for it during the conflict finding algorithm.

12.4. Data retrieval

As mentioned in chapter 6, it is unknown if Ivido is allowed to continuously retrieve medical data about a patient once consent has been given. The two options are that a patient must either give consent once and new data can be retrieved continuously or to give it multiple times and each time new data can be retrieved. For both cases, we have recommendations which are easily implemented in our solution.

In the case that a patient has to give consent once, Ivido must store information about when and for which medical data the patient has given consent to retrieve. This should be done for both logging purposes, as well as only retrieving data Ivido has gotten access to. Using cron, a time-based job scheduler already used in the Ivido platform, a data retrieval script can be ran at certain times. Another option would be to retrieve data on user login. We discourage retrieval of data when viewing the medication page. The conflict checker can run for quite some time and a patient wants to view their data as fast as possible.

When a patient must give consent each time data is retrieved, Ivido should remind a patient to update their medication retrieval. Our application stores information about the last time data was retrieved, so the platform should send messages to the patient in the form of: *"It has been X days since you last retrieved your medical data. To keep your overview up to date, please give us consent to retrieve your new data"*. Exact styling and design must be determined with new user tests.

12.5. Revert conflict

It might happen that a conflict is wrongly solved by a wrong selection by the professional or patient. Even though there is an pop-up message that asks the solver if all correct fields are selected of a conflict, there is still a possibility the solver want to revert his conflict solution.

Currently it is not possible to revert a solution, but the database is designed it such a way that it is possible to do so. In the back-end, however, is more work needed, since this a whole new project outside the scope of the conflict checking algorithm.

Bibliography

[1] Artsenfederatie KNMG (Royal Dutch Medical Association). Medische aansprakelijkheid (medical accountability). URL https://www.knmg.nl/advies-richtlijnen/dossiers/medische-aansprakelijkheid.htm.

[2] Consumentenbond (Dutch Consumers Association). Medische fouten (medical errors), 2018. URL https://www.consumentenbond.nl/je-rechten-als-patient/medische-fouten.

[3] Ramez Elmasri and Sham Navathe. *Fundamentals of database systems*. Pearson, 6 edition, 2014.

[4] Z. Eviatar. Reading direction and attention - effects on lateralized ignoring. *Brain and cognition*, 29(2):137–150, 1995. Cited By :43.

[5] MedMij. Pgo, 2018. URL https://www.medmij.nl/pgo/.

[6] Stichting MedMij, 2018. URL https://afsprakenstelsel.medmij.nl/display/PUBLIC/Normenkader+informatiebeveiliging.

[7] Fast Healthcare Interoperability Resources, 2018. URL https://www.hl7.org/fhir/.

[8] P.M.L.A. van den Bemt and T.C.G. et al. Egberts. Hospital admissions related to medication (harm). Technical report, Division of Pharmacoepidemiology & Pharmacotherapy, Utrecht Institute for Pharmaceutical Sciences, 2006.

A

Research paper

A.1. Introduction

Important abbreviations

This paper will contain a lot of medical abbreviations, we will explain the ones that often appear:

1. **PGO** Persoonlijke gezondheidsomgeving - personal health environment. An online platform for looking up and controlling access to personal health data.

2. **LSP** Landelijk Schakelpunt - National datalink for the Netherlands. Link for exchanging personal health care information data between different health care organisations and PGOs.

3. **BgZ** Basisgegevensset Zorg - The basic set of health care info. All patient information relevant to all providers of health care. This information is shared with different organisations through the LSP.

4. **Zib** Zorginformatiebouwsteen - Data blocks for health care information. The BgZ is divided into smaller subsets of information, all with a theme; e.g. patient details or allergies. A single zib contains all information about a single theme. A patient can have multiple zibs of the same type; e.g. a patient can have multiple allergies, for each allergie a single zib exists.

5. **FHIR** Fast Healthcare Interoperability Resources, pronounced as “fire”. A standard for sharing health information. This standard contains a standardized data structure and sharing of data with RESTful API. FHIR STU3 is used for sharing zibs in the Netherlands.

Client

Ivido B.V., referred as ‘Ivido’ in the following chapters, is a PGO which aims to improve communication and data sharing in the Dutch health care system. It has created a platform where patients, professionals and organizations (e.g. hospitals or health insurance companies) come together. A course can be created for a patient, which will be used during the entire duration of the illness. In this course the patient will be guided and updated by health care professionals participating in that course. When we refer to user(s) of Ivido we refer to both patients and professionals.

Problem

In the Netherlands online health care has seen a growth. In the past 6 years, the use of online health care has increased from an average of 9% to 24% (CBS, 2018), resulting in more online medical data. Often this medical data from patients is stored on different servers with different kind of information, since a hospital has other data about a patient than a pharmacy

and they both safe their data in their own databases. Yet for health professionals helping patients, all the medical data is important. Worldwide a new standard has been developed, HL7 FHIR, and has been stated to change the domain of clinical IT significantly (Andersen et al., 2018). FHIR is currently being implemented in Dutch online health care on several fronts. To retrieve data from a source, FHIR is used to communicate with the LSP. In the Netherlands, this medical data is structured in different zibs.

Ivido has given us the task to develop a system in which Ivido will able to join this sharing of data. If a user gives permission, Ivido can gain the rights to locally store all medical data from different sources. This data should be checked such that duplicate data from different sources is removed. Since we will lay a basis for implementing code, we should focus on a single zib. Our developments should be able to be extended, such that other zibs can be implemented.

The single zib we shall implement is the medication zib (Appendix A). This consists of three subzibs containing information about medicational agreements, administering agreements and medication usage. This zib is of great importance and regulations for this specific zib has been created. The most recent, medication process v9 (Registratie aan de bron, 2018), is the one we should implement.

Ivido has legal responsibilities to which our project has to comply. These rules are stated in the MedMij regulations (MedMij, 2018). This is a Dutch agreement system created for sharing medical data.

With data being stored on multiple servers and needing to be shared with Ivido, the question arises how these datastreams can be combined. All data processing should be completely automated, due to the great amount of data some patients might have. Therefore, this report will aim to answer the following question:

How can Dutch medical data, structured as stated by FHIR standard, from multiple sources be combined in a single database?

To answer this question, we separate the problem in three subproblems. First, a new database has to be designed. Since multiple medicational zibs can exist for a single patient, lots of medication records can be created. Reading speed of medication data has a high priority, since users want to access the data quickly.

Secondly there is the problem of the merging of medication data. Data can come from different sources and some sources might overrule existing data. When data is merged and written to the database speed is not an important factor.

Finally, the last problem has to do with using the data. When a user wants to see his data, it should be viewable from within the Ivido platform. An user interface must be created which shows all necessary information.

A.2. Problem analysis

With the given project several problems arise. In this chapter all encountered problems will be explained. We will analyze possibilities, but no conclusions will be made. Any conclusions and design choices will be explained in section Solution.

First of all, a flow chart was created to show the main outline of the problem. It shows what happens when a patient requests to share his medicational data with Ivido. A request is send to the LSP which then redirects the request to the institution the patient has chosen. The medicational data is send from the institution back to Ivido through the LSP. The data is then checked whether it is valid, duplicate or conflicting. Finally it is stored in the database

in the appropriate way.

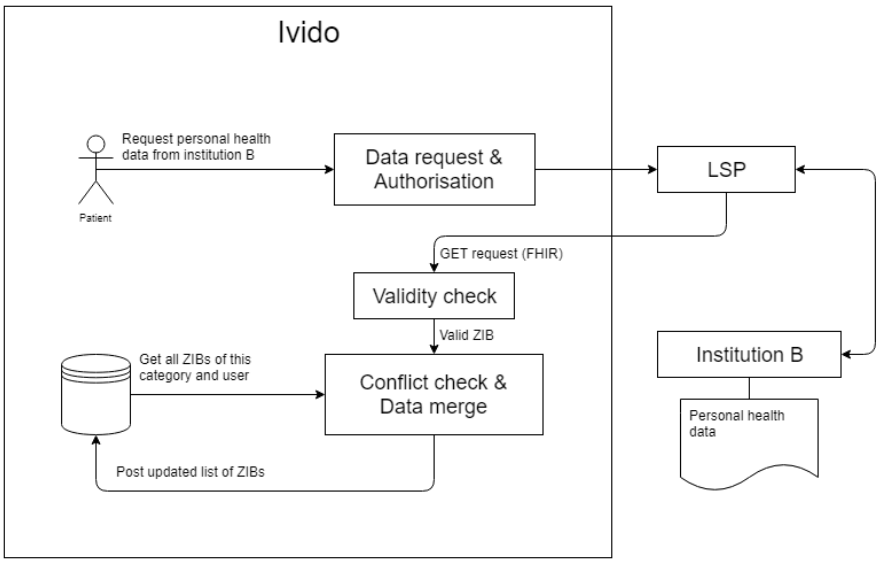


Figure A.1: Flowchart of an user giving consent to Ivido to store data locally.

MedMij regulations

The regulatory compliance should be based on the MedMij regulationsIn the following parts of this chapter these regulations will be taken into account and explained where needed.

Database analysis

For the construction of the database, a two factors have to be kept in mind. The first one is the encryption of data, due to the MedMij rules. Second, data has to be stored in a way that it supports its functionality.

Data encryption

For the encryption of data in the database MedMij has stated that saved data has to be encrypted on either disk-level or database-level with a function reviewed by the National Cyber Security Centrum (NCSC) as good (MedMij, 2018). This left three different encryption options: RSA with a private key length at least 3072 bits, AES-256-GCM or AES-128-GCM, or ECDH.

Data usage

When given permission, Ivido can store a patient's medication data locally. Any user with permission, should be able to read the data at any time. As writing to the database is expected to occur less frequent than reading, we favor read speeds over write speeds. Furthermore, new data has to be compared to existing data, so that false data or duplicate data can be removed. This process will take longer compared to only reading data, since this results in first reading existing data, then comparing the new record to the existing data, and finally writing the resulting data back to the database. In case of this process taking more than a few seconds, the patient can be notified with a message.

Data

All retrieved medication zibs should be stored in the database. Beforehand, it should be checked if data is duplicate or conflicting with existing data. According to the Ivido supervisor, no data exists which is completely false. This means, that we do not need to verify if data given is actually correct. We can assume that data received is correct. We do need to filter duplicates or falsely structured zibs. What to do with conflicting data is unknown at the moment of publishing (04/05/2018), and our supervisor is looking into it.

The data comparison to remove duplicate data and find conflicts, should compare entire zibs. Comparison shall be done by comparing all information. Similar information or conflicting information should be recognized on key information in a medication zib. For medication, this shall often be the medicine and dose. Deciding what information can be compared to decide similarity, shall be done by the Ivido supervisor.

Existing zibs in the database will never be updated, as stated by MedMij. Whenever a zib of the same category and user comes into the database the old zib will become outdated and a new zib will be created. This way we ensure that no old data is removed that might contain important information for in the future.

Interface

A patient must be able to view his own data. This should be visible in the Ivido platform. To comply with Ivido styling, the user interface should exist of a so called 'block', with recent information. This block should show the most important information about medication, which are decided by the Ivido supervisor. The option to view more details should be available. This would redirect the user to a new page, showing all information known in a medication zib.

Resources

Throughout the project several frameworks and other resources will be used. Below are all resources mentioned which we will encounter.

Moodle

Ivido has created an online PGO based on, amongst others, the Moodle framework. Moodle is an open-source online learning environment, consisting mostly of PHP and SQL. It is commonly used for course management. All coding that happens should be done to comply with Moodle. This means primarily coding in PHP and SQL, although front-end development such as HTML, CSS and Javascript is possible.

Moodle is created with the idea that there exists a core and user created plugins. The core is created, updated and published by Moodle developers. User plugins can be created freely. When coding, the Moodle core should not be modified. This is due to the fact that with updates the core files will be overwritten and remove any modifications made

FHIR

Fast Healthcare Interoperability Resources (FHIR) is a standard that has been created to share medical data online. The medical data is structured in resources, each containing information.

Resource sharing through FHIR is done with the RESTful API. Therefore, we use HTTP requests to reader write data. Such a request will be send to an 'endpoint', a web address. On a request, json containing data in the FHIR structure will be returned.

Endpoints

Nictiz is the leading expert in e-health in the Netherlands. They compose standards, advice and work with MedMij. For testing purposes during the project, Nictiz has provided an FHIR endpoint with test data.

PHP cURL

When creating a connection with an endpoint to exchange data, MedMij has stated that a connection using TLS is a must. To obtain such a connection the PHP/CURL library will be used. CURL can be used to automatically obtain such an connection.

A.3. Solution

After the problem analysis, solutions were created. First, the database design will be shown and explained. Second, the user interface that shall be created will be shown. Third, the different styles of testing the implementation will be given. Finally a MoSCoW method will illustrate more closely what will be absolutely necessary to finish the project.

Database design

Considering the problem as described in the problem statement, the database should be able to meet requirements in the areas of scaling and encrypting.

Scaling

For the database of Ivido to be scalable for the future, a document store would be a great solution to store all zibs. This is due to the fact that zibs are changing every few months, since the MedMij regulations are still developing. Also, not all fields that consists in a zib are required to be filled in, making storage of the information inconsistent. Also with more users in the future, both the owner of the data and people the medical data has been shared with will access the data on a regular basis. The database load can increase immensely with users and stored data. Distributing the document database would reduce this load.

However, this solution is not relevant for this stage of the development of Ivido. Therefore, the zibs will be stored in the relational database. The part of the zib containing information will be stored in json in a single field. This way data is easily usable. At the same time, upgrading to a document store becomes a possibility; only a single table has to be changed, with a single field per zib.

Encryption

In section Problem analysis, several data encryption algorithms have been presented. RSA, AES_128_GCM, AES_256_GCM and ECDH are options noted as good by NCSC. We have chosen for a application-level encryption with AES_128_GCM that only encrypts the medical data from zibs. This is done for two main reasons. First of all, we chose for AES encryption due to this being symmetric encryption. This ensures encryption and especially decryption time is faster than asymmetric algorithms. Decryption is of higher importance, since we expect data will be read more often than written. Secondly, we chose for an application-level encryption due to the fact that it is most efficient for us to encrypt a single column with medical data.This keeps the existing relational database intact and easy to read. With the fact that encryption is done in the application layer, it can be shown to MedMij what algorithms for encryption are used and that is complies with their rules. Finally, we chose AES_128_GCM over AES_256_GCM since AES_128_GCM is about 40% faser. This speedup comes from less cycles of repetition of data with smaller keys(Schneier et al., 2000).

Database structure

Considering these three classes of requirements, we created the following ER diagram (figure A.2). There are four entities. A user, which is an existing database in the Ivido platform. A zib, which contains zib names, the version and an unique ID. Both the user and a zib have a combined zib_entry. For every unique user and zib combination, only a single zib_entree can exists. A zib_entry has zib_pointers, which refer to the zib_store. The zib_store contains all zib data for a user.

With these relations, we created an UML (fig A.3) to store all medication data. Each table will be explained briefly, followed by an explanation about field choices.

user: an existing table in the platform and contains basic user information and an user identifier. This identifier is used for a lot of purposes in the Ivido platform. It is expected that in the tens to hundreds of thousands users will enroll in the platform.

zib: will contain basic information about zibs such as the name and the version. Version

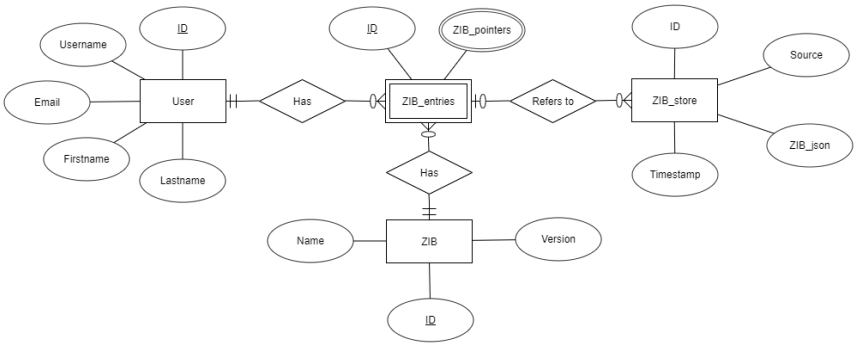


Figure A.2: Entity-relationship diagram.

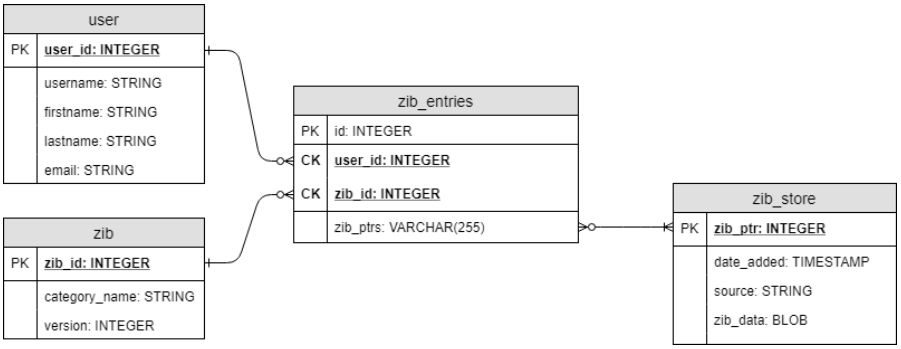


Figure A.3: UML of the database regarding storing of zibs.

is stored due to BgZ changing from time to time. Expected size is small, at the moment of publishing only 18 zibs of the current version are used, yet more revised versions exist.

zib_entries: will combine an user ID and a zib ID as a composite key. It contains pointers which point to the relevant zib_store records. The maximum number of records in this table will be at most (|user| * |zib|), because every user is able to share all existing zibs with Ivido. as.

zib_store: will store the actual data from a zib. A single record will store all data of a single zib from a user. The source from the data and date_added are stored here as well. Source is important for a user viewing his medicinal information to see where it came from. Date_added is implemented to help when comparing duplicate items. Multiple zibs of the same type can exist for a user (e.g. someone can use multiple medicines). A single zib_entries record will point to at least one zib_store record, so the expected size is larger than zib_entries, but approximations are difficult to make. This is due to BgZ being a new development and it being unknown how many zibs a patient will have on average.

For the design, the choice to split zib_store: zib_data from zib_entries has been made. This way an upgrade to a document database is done easily. This would be beneficial, as explained in section Problem analysis. Not only this, but retrieving a single record from zib_store without a zib/user combination does not store useful information, since it cannot be connected to a user or a zib. With zib_store not storing user or zib IDs, a single zib_store record is useless information. Zib_entries: zib_ptrs will contain multiple pointers to zib_store. These pointers will be stored as a view. With a view, a compound key on user_id and zib_id is possible, improving read times. Finally, in the zib_store data will be stored in BLOB format in the column zib_data. The medication data received following FHIR standards will be in json format. This entire json will be stored, retaining the structure defined by FHIR.. If the data

would be stored without structure, comparing and reading data would become more difficult.

User interface

The user interface is an important factor during the project. Patients should be able to see the data they have retrieved from other sources. To solve this, using the design principles of the Ivido platform, a block shall be created where information about current medication usage is shown. The design is derived from a plugin which already exists in Ivido, which is used to show a patient some specific medical details. Our project is to implement medication details, which should be designed in a similar way as the existing plugin. An example of how this plugin block should look like is shown in Figure 4 and Appendix B.

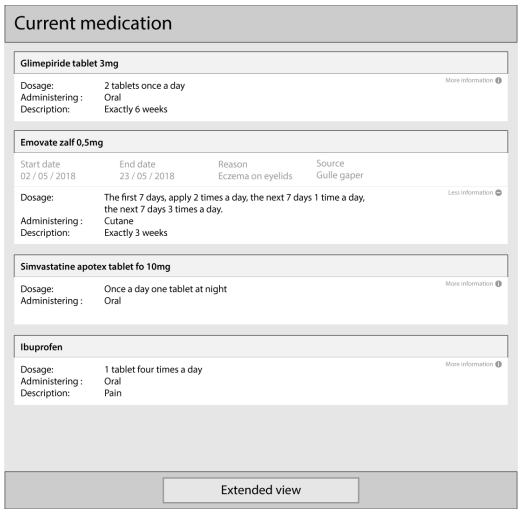


Figure A.4: Medication plugin block within the patient's health page, showing all medication which should currently be used by the patient. The extended view is shown in Appendix B.

According to the design principles of Ivido, a button can be found at the bottom of the block. This button can be used to open an extended view in a new page. This extended view contains all medication zib records for the patient, examples of this extended view can be found in Appendix B. These zib records are separated by recently ended (Appendix B1), current (Appendix B2) and upcoming medication, referring to whether the medication zibis no longer active, active or active in the future. The subzibs medicational agreements, administering agreements and medication usage are also visible in this extended view, giving the patient insight into the procedure leading to the overview of medicine to be used in the medication plugin block as shown in Figure 4.

Testing

Testing is an important aspect of software engineering. The testing of our product will be done on two different levels. Firstly we will implement unit tests where possible. We will use PHPUnit (Bergmann, 2018) for this matter, which is integrated in Moodle. Secondly, we will make use of different testing environments provided by Ivido. There is a local and a live environment. Developments have to be made on the local environment. This environment will also be used by the programmer himself to test the functionality. New developments are stored on a branch on Gitlab. When the code checks out to be sufficient, the branch will be merged with the master branch on Gitlab. Before it is merged at least one other programmer has to check the written code in order to maintain quality and stability. Once the new code is merged with the master the code is uploaded to the live testing server of Ivido. This server is a running server with testing data, to simulate the real server, where employees tests the new features on performance and usability. All data residing on the testing server is dummy data, and not actual health care data from patients.

MoSCoW

In order to prioritise tasks and properly define boundaries for our project, we use the MoSCoW method as formulated by H. van Vliet (2008). Using this method, we split tasks in four different categories: in the Must category tasks are found we must do or else we would consider our project a failure. In the Should category tasks are found that are highly desirable to be implemented in the project, but are not required to make this project a success. In the Could category gimmicks are found that we will do when there is time to spare at the end of the project. In the Won't category are tasks that benefit the project but won't be implemented by us, this category defines the boundary of our project.

Must haves

- Medicine zib database integration
- Send request to LSP
- Retrieve answer from LSP
- Store received data
- Encrypt local saved data
- Create user interface
- Solve duplicate data issue
- Solve conflicting data issue

Should haves

- Selection of certain data to request
- More zibs integrated closely related to medication

Could haves

- Implement all zibs

Won't haves

- Store zibs in document store
- Send data to other institution on receiving request

A.4. Organisation

This section will give insights into how the group will cooperate during the project.

Code sharing

Code sharing shall be done with GitLab. This is company procedure and shall therefore not be changed.

Schedule

We will use a scrum based approach (Schwaber & Sutherland, 2018) for our project to keep track of the process and stay on track. We will have sprints of one week, which will be discussed at the beginning of every week. The scrum master for each week will be rotating between the three team members.

Every monday a small meeting with the client will take place to inform about progress and exchange new information from the client's side. Once a week a meeting with the Delft University of Technology coach will take place. More meetings can be arranged if needed in both cases.

Quality
To maintain quality of the project, group members will check delivered work from other group members. This shall be done for all delivered work, such as uml's, coding through merge requests in GitLab, and report writing.

Communication
All members are expected to work on the office each day. Communication should therefore not be a problem. Communication not discussed at the office shall be done through the communication app Slack. All planning and task management will be done with the online tool Asana. These applications are are already procedure of Ivido and are therefore part of our project as well.

Roadmap
Below is a complete overview of what we expect to do and be finished in which week:

- **Week 1:**
  Medicine zib database integration
- **Week 2:**
  Send request to LSP
  Retrieve answer from LSP
- **Week 3:**
  Store received data
  Encrypt saved data
- **Week 4:**
  Create user interface
- **Week 5:**
  Solve duplicate data issue
- **Week 6:**
  Solve conflicting data issue
- **Week 7:**
  Bug fixes and quality check

References
Andersen, B., Kasparick, M., Ulrich, H., Franke, S., Schlamelcher, J., Rockstroh, M., & Ingenerf, J. (2018). Connecting the clinical IT infrastructure to a service-oriented architecture of medical devices. Biomedical Engineering/Biomedizinische Technik, 63(1), 57-68.

Bergmann, S. (2018). PHPUnit. Retrieved April 26, 2018, from https://phpunit.de/index.html

CBS. (2018, May 2). Internet; toegang, gebruik en faciliteiten. Retrieved May 2, 2018, from http://statline.cbs.nl/Statweb/publication/?DM=SLNL&PA=83429ned&D1=35%2c39%2c47-48%2c61&D2=0%2c3-6&D3=0&D4=a&VW=T

Codd, E. F. (1972). Further Normalization of the Data Base Relational Model, Data Base Systems, Courant Computer Science Symposia Series 6, R. Rustin.

MedMij. (2018). MedMij Afsprakenstelsel.
Retrieved May 3, 2018, from https://afsprakenstelsel.medmij.nl/

Nationaal Cyber Security Centrum. (2014). ICT-beveiligingsrichtlijnen voor Transport Layer Security. Retrieved from https://www.ncsc.nl/actueel/whitepapers/ict-beveiligingsrichtlijnen-voor-transport-layer-security-tls.html

Registratie aan de bron. (2018). BGZ, Specificatie gebaseerd op zibs release 2017. Retrieved from https://www.registratieaandebron.nl/pdf/BgZ_specificatie_obv_zibs_2017_v1.1.pdf

Savage, B. (2009, November 20). Designing Databases: Picking The Right Data Types. Retrieved April 25, 2018, from https://www.brandonsavage.net/designing-databases-picking-the-right-data-types/

Schneier, B., Kelsey, J., Whiting, D., Wagner, D., Hall, C., Ferguson, N., . . . Stay, M. (2000). The Twofish Team's Final Comments on AES Selection.
Retrieved from https://www.schneier.com/academic/paperfiles/paper-twofish-final.pdf

Schwaber, K., & Sunderland, J. (2018). Scrum Guide.
Retrieved from https://www.scrumguides.org/scrum-guide.html

Thales eSecurity. (n.d.). Selecting the Right Encryption Approach. Retrieved April 26, 2018, from https://www.thalesesecurity.com/products/data-encryption/selecting-the-right-encryption-approach

Van Vliet, H. (2008). Software Engineering: Principles in Practice (3rd ed.). Hoboken, United States of America: John Wiley & Sons.

Appendix A

10	MEDICATIE				
10.1	**Medicatieafspraak**	**Bekende medicatieafspraken**			
	Zib Medicatieafspraak	Zib	0..*		nl.zorg.Medicatieafspraak-v1.0.1
	Voorschrijver::Zorgverlener		0..1	context, reference	nl.zorg.Zorgverlener-v3.2
	RedenVanVoorschrijven::Probleem		0..1	context, reference	nl.zorg.Probleem-v4.1
	::Lichaamsgewicht		0..1	context, reference	nl.zorg.Lichaamsgewicht-v3.1
	::Lichaamslengte		0..1	context, reference	nl.zorg.Lichaamslengte-v3.1
	::Gebruiksinstructie		0..1	data, reference	nl.zorg.part.Gebruiksinstructie-v1.1
	Afgesprokengeneesmiddel::Farm.Prod.		1	data, reference	nl.zorg.part.FarmaceutischProduct-v2.0
	MedicatieafspraakdatumTijd	TS	1	data	
	GeannuleerdIndicator	BL	0..1	data	
	Gebruiksperiode::TijdsInterval		0..1	data, reference	nl.zorg.part.TijdsInterval-v1.0
	MedicatieafspraakAanvullendeInformatie	CD	0..*	data	
	RedenMedicatieafspraak	CD	0..1	data	
	MedicatieafspraakStopType	CD	0..1	data	
	Toelichting	ST	0..1	data	
10.2	**Toedieningsafspraak**	**Bekende toedieningsafspraken**			
	Zib ToedieningsAfspraak	Zib	0..*		nl.zorg.Toedieningsafspraak-v1.0.1
	Verstrekker::Zorgaanbieder		0..1	context, reference	nl.zorg.Zorgaanbieder-v3.1.1
	::Gebruiksinstructie		0..1	data, reference	nl.zorg.part.Gebruiksinstructie-v1.1
	::Medicatieafspraak		0..1	data, reference	nl.zorg.Medicatieafspraak-v1.0.1
	GeneesmiddelBijToedieningsAfspraak::Farm.Prod.		1	data, reference	nl.zorg.part.FarmaceutischProduct-v2.0
	ToedieningsafspraakDatumTijd	TS	0..1	data	
	RedenAfspraak	ST	0..1	data	
	ToedieningsafspraakAanvullendeInformatie	CD	0..*	data	
	ToedieningsafspraakStopType	CD	0..1	data	
	Gebruiksperiode::TijdsInterval		1	data, reference	nl.zorg.part.TijdsInterval-v1.0
	GeannuleerdIndicator	BL	0..1	data	
	Toelichting	ST	0..1	data	
10.3	**MedicatieGebruik**	**Bekend medicatiegebruik**			
	Zib MedicatieGebruik2	Zib	0..*		nl.zorg.MedicatieGebruik2-v1.0.1
	Voorschrijver::Zorgverlener		0..1	context,reference	nl.zorg.Zorgverlener-v3.2
	GebruiksProduct::FarmaceutischProduct		1	data,reference	nl.zorg.part.FarmaceutischProduct-v2.0
	::Gebruiksinstructie		0..1	data,reference	nl.zorg.part.Gebruiksinstructie-v1.1
	RedenGebruik	ST	0..1	data	
	GebruikIndicator	BL	1	data	
	Toelichting	ST	0..1	data	
	MedicatieGebruikStopType	CD	0..1	data	
	RedenWijzigenOfStoppenGebruik	CD	0..*	data	
	Gebruiksperiode::Tijdsinterval		0..1	data	nl.zorg.part.TijdsInterval-v1.0
	MedicatieGebruikDatumTijd	TS	1	data	
	VolgensAfspraakIndicator	BL	0..1	data	

Figure A.5: 10.1 shows medicational agreements. Who gave permission to what medicine, for what reason, for how long ect.
10.2 shows administering agreements. How much can you get from the pharmacy, for how long, ect.
10.3 shows medicine usage. How much has a patient actually used, when has he used it, ect.

Appendix B

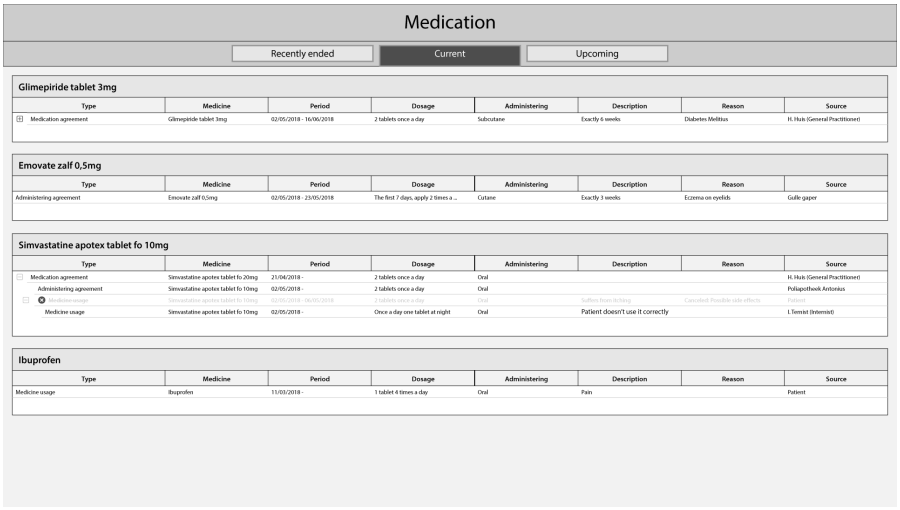


Figure A.6: Extended view, showing all Medication zibs which are currently active in a tree table. The subzibs medicational agreements, administering agreements and medication usage are shown under ‘Type’.

Figure A.7: Extended view, showing all Medication zibs which are no longer active in a tree table.

B

SIG Feedback

De code van het systeem scoort 3.3 sterren op ons onderhoudbaarheidsmodel, wat betekent dat de code marktgemiddeld onderhoudbaar is. We zien Unit Size en Unit Complexity vanwege de lagere deelscores als mogelijke verbeterpunten.

Voor Unit Size wordt er gekeken naar het percentage code dat bovengemiddeld lang is. Het opsplitsen van dit soort methodes in kleinere stukken zorgt ervoor dat elk onderdeel makkelijker te begrijpen, te testen en daardoor eenvoudiger te onderhouden wordt. Binnen de langere methodes in dit systeem, zoals bijvoorbeeld, zijn aparte stukken functionaliteit te vinden welke ge-refactored kunnen worden naar aparte methodes.

Bij jullie project is magic() in locallib.php een goed voorbeeld. Het hele algoritme is nu als één grote methode uitgeschreven. Dat maakt het op termijn moeilijker om te bepalen wat er in welke situatie moet gebeuren, en maakt het ook moeilijker om goede unit tests te schrijven. Je kunt de stappen van het algoritme, die nu met commentaar worden aangegeven, beter uitsplitsen.

Voor Unit Complexity wordt er gekeken naar het percentage code dat bovengemiddeld complex is. Het opsplitsen van dit soort methodes in kleinere stukken zorgt ervoor dat elk onderdeel makkelijker te begrijpen, makkelijker te testen is en daardoor eenvoudiger te onderhouden wordt. Door elk van de functionaliteiten onder te brengen in een aparte methode met een descriptieve naam kan elk van de onderdelen apart getest worden en wordt de overall flow van de methode makkelijker te begrijpen.

Hier geldt eigenlijk hetzelfde als bij Unit Size: methodes bevatten te grote stukken functionaliteit en te weinig abstractie, waardoor ze al snel veel complexiteit bevatten. Ook hier is het dus beter om een wat kleinere scope te kiezen om je code zo beter onderhoudbaar te houden.

De aanwezigheid van testcode is in ieder geval veelbelovend. De hoeveelheid tests blijft nog wel wat achter bij de hoeveelheid productiecode, hopelijk lukt het nog om dat tijdens het vervolg van het project te laten stijgen.

C

Info sheet

General information

Title of the project: Managing medication data collection from multiple sources.
Name of the client organisation: Ivido
Date of the final presentation: July 2, 2018

Description

Nowadays, health care institutions do not only store health care information generated at their own facility, but also information retrieved from other institutions. This distribution of data over different institutions raises problems of duplicate and conflicting data. Recently a new standard for exchanging medical data has been developed: FHIR (pronounce: ”fire”). In the Netherlands, this FHIR structure is being implemented to model the different aspects of the Dutch health care system, in the form of ”Health and Care Information Module” or HCIM.

It is our tasks to implement this new FHIR standard in the Personal Health Environment from Ivido, an online platform where patients can manage their own medical data. It is our responsibility to implement one of the more important FHIR structures, the medication data. Our job involved creating functionality to communicate with other health institutes, correctly handle FHIR data and create a user interface to show users their data. Finally, the designed algorithms should be able to recognize duplicates and conflicts in retrieved data.

The main challenge within this problem is to determine the decision rules whether medicinal data is in conflict with other data. Medication data is personal information and should be treated very carefully. Making assumptions or altering data can result in high risk situations for patients.

At the start of the project a MoSCoW model was made to determine what should be implemented or not. Based on this model the team made a plan for the upcoming 7 weeks. At the begin of every week a SCRUM based approach was used to determine all tasks and to ensure previous tasks were finished on time. After a few weeks of work, it became clear that there was very little test data available, which made the code difficult to test. The team decided that at the end of the project, the test code coverage should be at a high level of at least 80%.

As a final product an overview of the medication data was created with a small view showing only current medication and a big view showing the past, current, future and conflicting medication. Furthermore can the conflicts be solved by selecting the right medication option.

Our product will be used as foundation for further implementations of the other HCIMs and it will be actually used by the company.

Members of the project team

C.C. Berg
Interests: User experience, Database and Algorithm design principles.
Contributions: Front-end design and development. Conflict finding.

E.A. Rietdijk
Interests: back-end development, algorithm design, usability
Contributions: Front-end and back-end development. Conflict solving.

P.J.M. Verkooijen
Interests: Data science, AI, back-end development and algorithm design.
Contributions: Back-end development and testing. Conflict finding and testing.

All team members contributed to preparing the report and the final project presentation.

Client & Coach

Tristan Garssen - Ivido B.V.
Alessandro Bozzon - Web Information Systems, Delft University of Technology.

Contact information

C.C. Berg - chris.berg@live.nl
E.A. Rietdijk - emielrietdijk@gmail.com
P.J.M. Verkooijen - paulverkooijen93@gmail.com

The final report for this project can be found at: http://repository.tudelft.nl