MED CUT
Management- and Utility tool for Medical coding

Implemented at the Hopital cantonal de Fribourg

Bachelor's Thesis

Fabian Falamischia

February 2024

Thesis supervisors:

Prof. Dr. Jacques Pasquier-Rocha
Software Engineering Group

UNI FR
UNIVERSITÉ DE FRIBOURG
UNIVERSITÄT FREIBURG

Software Engineering Group
Department of Informatics
University of Fribourg (Switzerland)

Software Engineering Group

Acknowledgements

I want to thank the Hôpital Cantonal de Fribourg for allowing me the opportunity to do a practical implementation of my thesis. The medical coding team has been a huge help in testing and implementing the MED-CUT application. I'd also like to extend my gratitude towards prof. Pasquier for letting me realise such an open ended, practical project.

Abstract

A specialised Oracle Apex web application was developed and implemented at the Hôpital cantonal de Fribourg. The application "MED-CUT" implemented several features for facilitating the work of the medical coding team. Its primary function allows users to manage and distribute cases for medical coding. Secondary functions, such as accessing coding lists, displaying the state of medical documentation and a dialysis calculation page were added during development.

Keywords: Oracle Apex, PL/SQL

Table of Contents

List of Figures

List of Tables

Listings

1 Introduction

1.1. Hôpital cantonal de fribourg

The Hôpital cantonal de fribourg (HFR) has admitted 19'501 acute hospitalized patients, 1'679 rehabilitation cases and 532'787 ambulant treatments in 2022 across its 4 locations [8] [7]. In addition to the main Hospital in Fribourg, the HFR is also treating patients in its auxiliary location Meyriez-Murten, Riaz and Tafers. All year round patients are admitted through each of its urgent care centers or through appointments. From a quick 20 minutes consultation with the doctor, to a prolonged stay in one of the specialized hospital beds, each case is medically documented and managed with the help of specialized administrative software. The implementation of administrative software in hospitals is essential for organizational effectiveness and ultimately for enhancing the quality of patient care. It allows for doctors and healthcare staff to focus more on patient care and less on manual administrative tasks. For general hospitals like the HFR, which treat all types of ailings, the administrative work behind the medical treatments can be especially difficult. In addition, the bilingual nature of the HFR and the multiple locations make managing cases even more complex. Hospitals like the HFR have a constant need for more and better software to facilitate administrative tasks and allow for an efficient and correct workflow.

This project targets a specific section of the hospital administrative work. The section of medical coding, which is at a cross section of patient administrative work and medical documentation, is in need of a specialised software solution for their work. They have a particularly complex task of working through all hospitalised patient dossiers and coding them for the purpose of correct invoicing for the insurances. That same work is also submitted to the Bundesamt für Statistik (BFS) and used in Swiss-wide medical statistics to release public articles in their health section [3].

1.2. The System SwissDRG

Since 2012, hospitals in Switzerland are required to record hospitalised cases and to invoice them according to the standard of the system Swiss- Diagnosis Related Groups (DRG) [5]. The DRG system attempts to classify cases by their medical similarities and to make the cost and treatment remunerations for cases within the same group uniform. A medical dossier is coded into one of around 1000 DRG codes ranging from A01A (organ transplant) to Z86B (general minor factors which impact health). Each code consists of a letter which classify the case to a corresponding medical category and a specification. The DRG is calculated by taking International Catalogue of Diseases (ICD) and Schweizerische Operationsklassifikation (CHOP) codes into consideration. ICD are a collection of codes corresponding to medical diagnoses. All known medical conditions and ailing have an ICD code. The CHOP codes on the other hand are representing all the medical treatments and operations. Listing all the relevant ICD and CHOP codes for a case and applying the rules from the medical coding manual [4] will categorise a case into the correct DRG.

The practice of reading medical reports and translate them into the codes given by the DRG system is called medical coding. Medical coding is typically done with the help of coding software, which have all the rules for DRG categorisation stored. For general hospitals like the HFR, which have a wide range of DRGs, good knowledge of all the DRG systems and the medicine behind them is required to perform medical coding. Since SwissDRG dictates exactly how medical cases are to be invoiced, it is in the hospital's best interest to have a high quality standard in medical coding. Mistakes, such as accidentally omitting a treatment code, can quickly drop the invoiced cost of a case by several thousand Swiss franks.

1.3. Problem statement

The administrative challenge posed by having to code each case is very large and most hospitals in Switzerland struggle to solve it without the help of external enterprises. The situation in francophone Switzerland is especially problematic, as there is also a shortage of bilingual coders. The complex nature of medical coding and the multitude of different case types makes it very difficult to distribute the work in an efficient manner. Until now all the medical dossiers at the HFR had to be distributed manually and sent with several Excel sheets to the corresponding coders for treatment. There are 6 intern coders and two external enterprises which handle the medical coding for the HFR currently and each one has their preference of case types. For example, most internal coders do not code in German, so medical cases from Meyriez or Tafers cannot to be sent to them. Clearly there is a need for a software solution which can facilitate this task.

1.4. Feature expansion

This project originally set out to only solve the administrative work of managing dossiers for medical coding. During development and once the Apex environment became more familiar, ideas for additional features surfaced. Additional features which have been successfully implemented into the application include the functionalities provided by the Statistics and Coding list pages, which further help medical coders to focus on their work. Furthermore, a Dialyse page has been added to help with medical documentation of dialysis treatments. The case managers of the intensive care unit in Fribourg have expressed interest in the application during development and have volunteered to test a dialysis calculation tool in the MED-CUT application.

2
Oracle APEX

2.1. Oracle APEX

The HFR's administrative and medical data are stored in Oracle data bases [11]. Oracle offers the Oracle APEX system for developing and running APEX web applications to all their licensed clients [1]. For the purposes of developing an application, which has to closely work with its data base, APEX seemed the ideal choice. Oracle APEX offers a large set of tools for constructing applications. Employing a 3-tier architecture, Oracle APEX channels requests from the browser through a web server to the database. All processing and data manipulation occur within the database, ensuring zero-latency data access. As a result APEX applications offer exceptional performance even for large data bases. APEX accesses data stored in tables and Procedural Language / Structured Query Language (PL/SQL) code. In APEX Uniform Resource Locator (URL) requests from your browser are translated into the appropriate APEX PL/SQL call, with the database executing the PL/SQL and relaying the results back to your browser in Hypertext Markup Language (HTML) format. The APEX engine renders assembled page attributes into one viewable HTML page. These page attributes have many inbuilt functionalities, which facilitate the development process. Once the APEX application is running, the web browser initiates a web request directed to Oracle Representational State Transfer (REST) Data Services (ORDS), which subsequently forwards the request to Oracle Database for execution. Within the database, Oracle APEX processes the request. After completing the processing, the outcome is conveyed back to the browser through ORDS. The full process flow can be seen in figure 2.1.

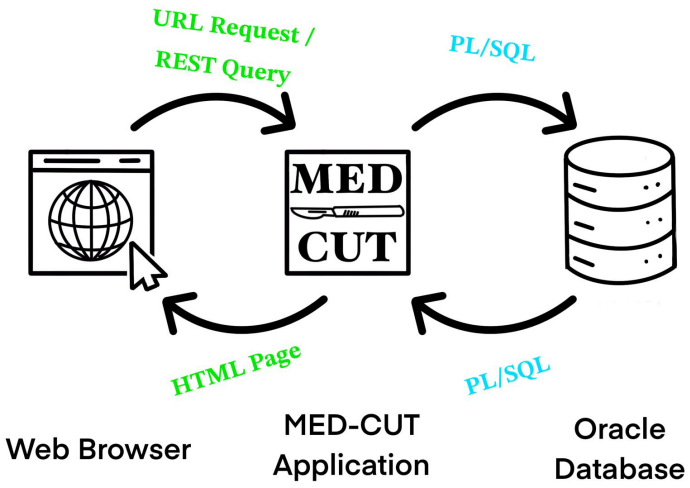

Fig. 2.1.: Process flow of the application

2.2. Working wiht the HFR's database

2.2.1. Data

The first challenge in working with large databases lies in making the data organized and accessible. The HFR stores all of its administrative data on patients across hundreds of Oracle tables. These administrative tables are used on productive servers with the application Opale from Bluepearl [10] for everyday use in storing and editing patient data. Administrative staff, nursing personnel and doctors work with this application throughout the day. The medical data is stored in a separate data base and used with a different application. Medfolio from NEXUS/KIS [9] is the information system used at the HFR for all medical reports. Fortunately the data base architects responsible for the Medfolio system provided me the data for the medical reports for use in the MED-CUT application.

The safest and easiest to maintain solution is to create a shared Data Base Link (DB link) to the productive database backup server. This DB link contains the Structured Query Language (SQL) schema needed for the application MED-CUT. For the purposes of the application, a real-time connection to the productive data is not required. Statistics, attribution of dossiers, coding lists and dialyse calculations can be managed with the backup data, which is updated each night. As we will see in the specific application pages an overnight updating dataset can even be used as a means of day to day control. A shared DB link has furthermore the advantage of providing all users of the app access to the remote database (the productive server) over the same connection, which speeds up processes significantly. For readability only the data tables and views which are pulled from the productive server schema will be listed in their entirety in the following section and the Code Listings in the Appendix.

It should be noted that this data is still only a very small fraction of all the administrative data available from the productive server.

2.2.2. Application Tables and Views

In order to have the application work with the data from the DB link in an efficient way and to provide its own data storage, it needs to have the data stored in SQL views and tables. By convention the naming of the tables start with a T_, which are used for storing and modifying application data. Views, starting with a V _, which are queries pulled from DB link views, application views or tables are generally used to display the data on application pages. APEX provides extensive documentation and helpful guidelines for building SQL databases for their application [2].

Application Tables

T_HELPER

The T_HELPER table is the primary data storage table for most application functionalities. With the same primary key (DOSSIER) as all the views created from the HFR data base it can easily be joined into larger views.

Column Name	Data Type	Detail
DOSSIER	NUMBER	**Primary key**, Unique ID for a case
ASSIGNED	VARCHAR2(10 CHAR)	Visa of in-app assigning
GROUPEMENT	VARCHAR2(10 CHAR)	Visa of previous case if groupement eligible
LS_READY	CHAR(1 BYTE)	State indicator of medical report
POS_READY	CHAR(1 BYTE)	State indicator of operation protocols
DONE	CHAR(1 BYTE)	State indicator for the coding list
REMARQUE	VARCHAR2(200 CHAR)	Commentary field for storing a commentary
REMARQUE_PLUS	VARCHAR2(200 CHAR)	Secondary commentary field
GROUP_VALID	CHAR(1 BYTE)	Groupement validation indicator

Tab. 2.1.: The T_HELPER table (storing app data for all cases)

T_ASSIGN

The T_ASSIGN table is used for temporary case assigning in the shopping cart of the attrib page. It shares most of its data structure with the T_HELPER table as it is needed to copy entries into and from that table. The T_ASSIGN Table can be seen in the Appendix under table B.1.

T_LIST_ASSIGNVISA and T_USERS_VISA

The T_USERS_VISA Table B.2 and T_LIST_ASSIGNVISA Table B.3 hold user information which are necessary for the application to know which user has which Visa (in other HFR programs), which category of case preferences they belong to and what rights they have within the app.

CAS and SERVICE

The T_CAS_TYPES Table 2.2 stores information about the different types of cases. A case can be of many different types, which in the database is referred to as "CAS". This indicates the main category of a patients stay at the hospital. As a general rule this has to be where the patient has received the major treatment. The "SERVICE" refers to the section of the hospital the patient was last at prior to his release. For example if a patient enters the hospital in Fribourg via urgent care (SERVICE: URG-FRI) and is then transferred into the Orthopedic service (SERVICE: ORTHO-FRI) to receive treatment for several days. When he is then transferred to the inner medicine (SERVICE: MEDE-FRI) for a quick exam and then released, the case (CAS) categorisation will be ORTHO and the service (SERVICE) will be MEDE-FRI. For the distribution of cases the "CAS" and "SERVICE" category are usually sufficient as they provide information about the majority of the documentation and the location of a patient.

Column Name	Data Type	Detail
CAS	VARCHAR2(10 CHAR)	**Primary key** Case type e.g. CHIR, PALA etc.
TYPE	VARCHAR2(10 CHAR)	App identifier for queries e.g. XAIGU, XW etc.
DETAIL	VARCHAR2(50 CHAR)	Text description of the case type e.g. Chirurgie, Residence Palliative etc.

Tab. 2.2.: The T_CAS_TYPES table (all types of cases)

A very important functionality for most pages in the application lies in distinguishing the group to which cases belong to. The distinguished groups are acute cases (XAIGU), rehabilitation cases (XREA), waiting bed patients (XW) and nursing patients (XNURS). Each group has many different "CAS" types. By adding a table field of "TYPE" to each case category, we can pull queries with those group identifiers. For example a query where "TYPE" is LIKE _REA we can have a list of all cases belonging to the rehabilitation patients group. The X prefix for all the categories was also purposefully chosen, so that a query with LIKE X% can return all cases regardless of category.

The preferences of each Visa are stored in two tables. The T_EXCLU_CAS Table B.4 and T_EXCLU_SERVICE Table B.5 contain the exclusion cases for the "CAS" and "SERVICE" category respectively.

Dialyse App

The T_DIALYSE_APP table B.6 stores all the necessary information to provide the Dialyse page with its functionalities. Up to 7 dialysis treatments per patient can be stored. In our experience that should be more then enough.

Views

The views for the application, where patient data is concerned, are pulled from the following 4 DB link. Across them they have over 300 columns so they will not be listed here. A short description for each one and the complete queries which draw from them will instead be provided.

t_opale_pop@common

The pop patients table contains basic patient information such as name, date of birth etc. It is also the only table which does not have the Dossier number as primary key.

t_opale_adm@common

The adm patients table contains general administrative data of patients. E.g. hospital case, transfers, entry and exitdate etc.

t_opale_adm_fac@common

The adm_fac patients table contains more specialised information pertaining to the administrative state of the invoice dossier. E.g. validation visa, billing state etc.

t_opale_ofs@common

The ofs patients table contains all information concerning the medical coding of a case. This includes the entire coding list for each patient.

MEDFOLIO DB link

The DB link HFR_PATIENTS_CODING_LIST@RHFMFPROD_MF_CUSTOM .HOPFR.NET.FR.CH provides information on the state of medical reports and operation protocols.

Constructing V_COMPLETE

When constructing large combined views it is generally a good practice to not join them all in one enormous query. The V_ADM_FAC view B.1 combines the t_opale_adm@common and the t_opale_adm_fac@common from the DB link into one view containing all relevant cases for our application. By applying several WHERE clauses we have a view containing only hospitalised cases. The V_POP view B.2 contains all the personnel patient information, which we want combine into our complete view. Finally the V_OFS view B.3 has the entire medical coding listed per patient.

The V_COMPLETE view combines the V_ADM_FAC, V_POP and T_HELPER table to be used in most queries for the application.

```
1
2 CREATE OR REPLACE FORCE EDITIONABLE VIEW "V_COMPLETE" ("DOSSIER", "PATIENT", "FID", "
    NOM", "PRENOM", "DATNAIS", "DATDEC", "CAS", "SERVICE", "APDRG_VISA", "
    STATUTCODAGE", "GESTIONNAIRECODAGE", "TYPADM", "GENRE", "DATENT", "DATSOR", "
    APDRG", "GROUPEMENT", "LS_READY", "POS_READY", "DONE", "REMARQUE", "REMARQUE_PLUS
    ", "ASSIGNED", "GROUP_VALID", "ETATCODAGE") AS
3  SELECT adm.DOSSIER, adm.PATIENT, adm.FID, pop.NOM, pop.PRENOM, pop.DATNAIS, pop.
     DATDEC, adm.CAS, adm.SERVICE, adm.APDRG_VISA, adm.STATUTCODAGE, adm.
     GESTIONNAIRECODAGE, adm.TYPADM, adm.GENRE, adm.DATENT, adm.DATSOR, adm.APDRG,
     helper.GROUPEMENT, helper.LS_READY, helper.POS_READY, helper.DONE, helper.
     REMARQUE, helper.REMARQUE_PLUS, helper.ASSIGNED, GROUP_VALID, adm.ETATCODAGE
4        FROM V_ADM_FAC adm
5             LEFT JOIN V_POP pop ON pop.PATIENT = adm.PATIENT
6             LEFT JOIN T_HELPER helper ON adm.dossier = helper.DOSSIER
7    ORDER BY adm.DATSOR DESC;
```

List. 2.1: V_COMPLETE view

The V_COMPLETE_OFS view B.4 is an additional view, which has the same structure as the V_COMPLETE table but has an additional join with the V_OFS view. This ways we have access to a view containing the entire medical coding for each case. By having the V_COMPLETE and V_COMPLETE_OFS separate we can use a more lightweight and faster view to use with most application pages. The heavyweight V_COMPLETE_OFS is used for the more heavy duty statistical pages.

Groupements

A very important mechanism, which will be referred to from here on out as groupement, is the grouping rule imposed by SwissDRG. After a patient is released from a hospital stay and has to be admitted again within 18 days for a treatment within the same DRG, a case grouping has to be done. Effectively the two separate stays will be combined into one case with one total DRG. This rule imposes additional administrative challenges as the groupement should ideally be done by the same person that has coded the individual cases. As such it is important to keep track of who has coded which case and take potential groupements into consideration when assigning further cases.

The V_GROUPEMENTS view B.5 pulls a list of potential groupement cases. By having nested queries, which group patient cases by patient and order them by exit date, it compares current entry date with past exit dates and can detect potential groupements. The LAG function is particularly helpful in achieving this. Ultimately this query produces a list of all cases where a previous case of the same patient took place within 18 days. The "PREV_GESTION" column is particularly useful as it displays the validation Visa of the previous case. If we have a potential groupement but the previous case has not yet been validated the column will be set to NULL.

Groupement validation

The primary way of identifying a validated dossier is by means of the APDR_VISA field, which should be NULL if non-validated and containing a visa if validated. A realization that followed a few weeks into test deployment of the application was that several dossier appeared to be not validated despite them being clearly marked in the

productive database. These exceptions are caused by dossier groupement. When two ore more dossiers are grouped, a main dossier will eventually contain the entire medical coding for all other cases involved. During this process several coding files are created and can be seen in the _opale_ofs@common tables under different sequences. The main sequence file (sequence 0) contains all the right information, except the validation visa on specifically grouped dossier become NULL again.

To solve this issues an additional View containing only these grouped validated dossiers (sequence 99) was created. It was then used to signal in the T_HELPER table which of these dossiers can be considered as validated. The V_VALIDATED_GROUPEMENTS view B.6 thus gives us list of these validated cases.

MEDFOLIO views

Pulling the necessary information from the Medfolio DB link the V_MEDFOLIO_CODING_LIST view B.8 displays the state of medical reports and operation protocols per case. Since this view only displays "yes" or "no" value rows per medical report or operation protocol, a little additional logic and ordering is required to make a per case view. The two views V_MEDFOLIO_DOSSIER_LS view B.9 and V_MEDFOLIO_DOSSIER_PO view B.10 pull the necessary information by using MAX and MIN functions. The functions work on string values and have the benefit of pulling per case non-ready report lines. For the operation protocols, the query pulls the MIN value for protocol rows, meaning if a "no" is present it will be prioritised over "yes". Since a case is only ready if all rows contain a "yes" it can be used as an indicator for the readiness of a case. The medical report query uses a MAX function instead because there should be only one "definitive" exit report per case. It happens rather often that medical secretaries enter "provisional" reports into the system and leave them as they are.

V_YEAR

The V_YEAR view B.7 extracts the years of current cases in the application. This view is used to limit selection menus, which allow for a date or year to be picked.

3
MED-CUT

3.1. Overview

The APEX development environment provides a page where the user can access and work on created apps. It also provides a SQL workshop page where the user can edit the SQL databases which the apps work with. For the purpose of this report, only the most important objects and features will be explained in detail. An extensive list of all the functionalities can be found in the APEX app manual [1].

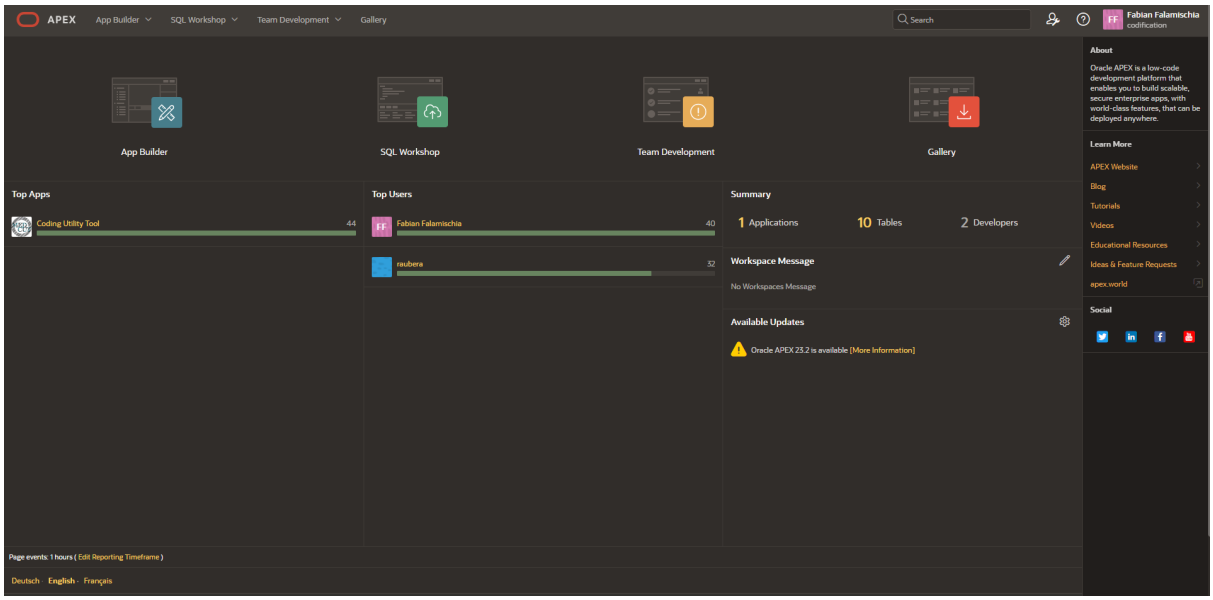

Fig. 3.1.: Development Page of the Application

Once an existing application is selected or a new one created a listing of all application pages is shown as in Figure 3.2.

Fig. 3.2.: Application Pages

In Oracle APEX each web page is constructed in the page development environment. In this following section a development page is shown and explained with its layout and functionalities. For the individual page sections, the development pages will not be displayed as they have nested and difficult to show components. Instead we will take a closer look at an example now, which should showcase the concepts used for all following pages.

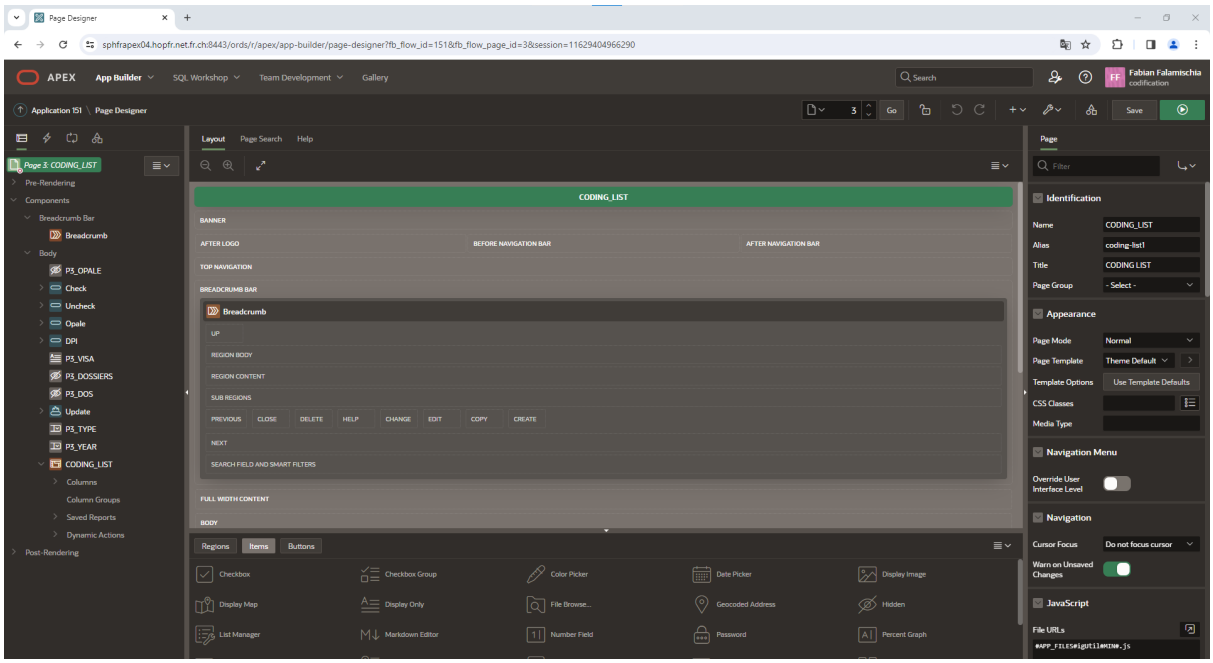

Fig. 3.3.: MED-CUT Development Page

A development page as shown in figure 3.3 lists all components on the left, which can also be oriented in the middle screen section to anchor them to specific places when they are rendered in a viewable HTML. All APEX components and items can be referenced in PL/SQL or JavaScripts.

Interactive Grid

An interactive grid is used on all pages where data needs to be displayed and interacted with. The interactive grid item is an APEX object which can display dynamic data from a table, view or PL/SQL query. Its main advantages, which are used throughout the application, lies in its provided functionalities for the user. The interactive grid has Actions and Column Heading menus which let the user apply their own filter and column layouts. These layout can be stored in public and private reports. Interactive grids can also be exported into Excel sheets, CSV files or PDF and downloaded through the browser.

Authorisation

APEX supports authorisation schemas, which are used to identify and authorise user access to the application, application pages and even individual page items. With the example PL/SQL script in listing B.11 and user groups, which are pulled from the HFR tables of collaborators we can ensure access only for authorised users. Since the APEX application has then also access to those credentials, users can directly login to the application without the need of a password.

Translation

The MED-CUT application was initially programmed and set in the English language. Due to the bilingual nature and naming conventions of the various data at the HFR,

many items and text objects have either German or French components. In order to make the language used in the application more uniforms, translation files were used. APEX supports the implementation of XML Localization Interchange File Format (XLIFF) files. APEX can generate these files for an entire application. They can be edited to have all their target components in the corresponding language. An exemple of a XLIFF file can be seen in listing B.24. Two such files, one for German and one for French, are used by MED-CUT. They allow the application to run in either language and have all translatable components render in their respective language.

3.1.1. Shared components

APEX supports application wide files and functionalities. In the Shared components section of the development environment we can create files which can be used on several pages. In the following section we will take a look at some of the commonly used scripts and components for the MEDCUT application.

Navigation Bar

The navigation bar is a table stored in the shared components which, if enabled, is rendered on each page to allow for navigation across pages which have a navigation entry. For many types of pages, like modal pages, form pages or pop ups there are purposefully no navigation entries, so they can not be navigated to and from arbitrarily. Navigation entries for this application include all the listed pages in the MED-CUT chapter.

Checkbox Script

For a very useful functionality with the interactive grid in APEX we would like to be able to check off one or several lines and then do something with them. The checkbox item can easily be implemented into interactive grids, but in order for them to then process the corresponding data a little additional work is required. The file responsible for making this work is the javascript #APP_FILES#igUtil#MIN#.js static file and can be imported into pages which need it. The static file defines the function igiUtil which can then be used on pages with the correct parameters to store a colon delimited list of all checked values in an APEX item. This will prove very useful for some pages which require the user to select lines in a grid which have then be processed further with a script. The entire #APP_FILES#igUtil#MIN#.js static file and an example of a function call on an application page can be seen in the code listing under listing B.22 for the static file and listing B.23 for the function call. On an application page the checkbox script can then be used as seen in listing 3.1.

```
if (!igUtil.selectedPKs("ATTRIB", "P12_DOSSIERS", "Please select at least one Entity!"
    ))
  {
    // The function will return FALSE if the user does not select at least one entity.
         apex.da.cancel() will stop the subsequent steps in the Dynamic Action from
        running.
    apex.da.cancel();
  }

igUtil.selectedPKs("ATTRIB", "P12_DOSSIERS", "Please select at least one dossier!");
```

List. 3.1: Script on the Attrib page for Checkbox selection

The function needs an APEX static region, namely the interactive grid, an APEX object into which the values need to be stored and an error message. From there on the APEX item can be submitted for further actions, like running a PL/SQL script.

Row coloring

For several pages we want to have conditional rendering of rows in Interactive grids based on a criteria. Enabling this script allows us to highlight rows in an interactive grid which has non empty cells in the status_column. By slightly modifying the script in listing 3.2 conditional row coloring can be achieved. This is used on several application pages.

```
function highlight_ig_cells() {
    // for each cell in marked column
    $('.highlight td.status_column').each(function() {
    // get cell text
    cellData = $(this).text();

    // rules for coloring
    if (cellData !== ""){
        // full row coloring
        $(this).parent().children().css('background-color','#bf80ff');
    }
    })
};
```

List. 3.2: Coloring script of the Attrib page

There are several more scripts which are used across pages but they are much more simplistic and are not listed here. Scripts which are used to assign and retrieve values from APEX items are used on every page.

3.1.2. Automation

The automation module is a very useful feature in the APEX environment. It allows for scheduling a variety of scripts to be run at fixed times/intervals on tables or the entire application.

In MED-CUT there are several such scripts, which have to be run daily to ensure that the application data and functionality are up to date.

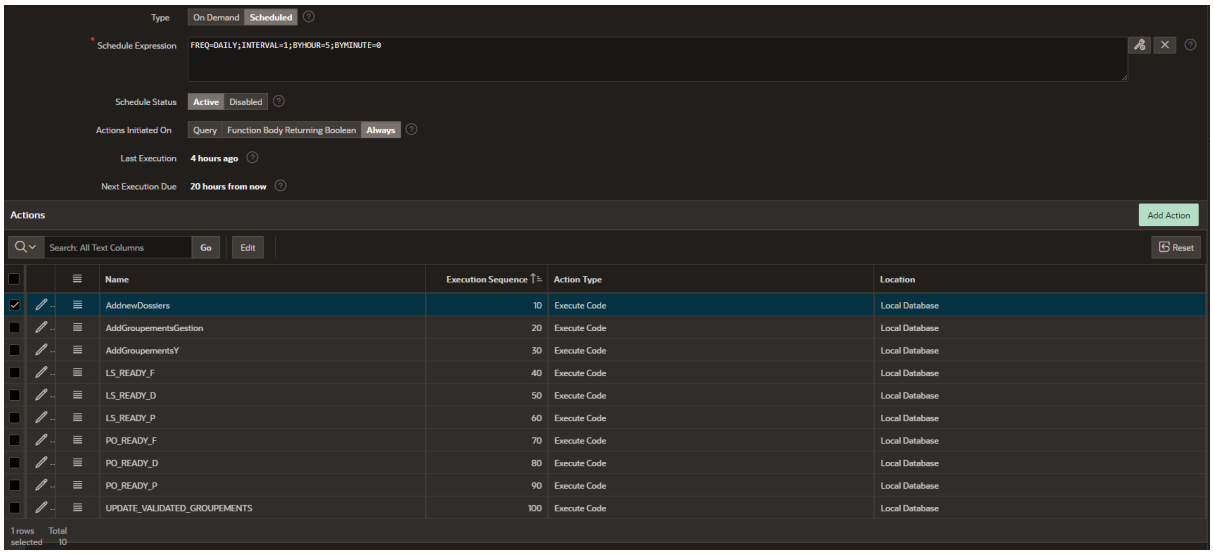

Fig. 3.4.: Automations of the MED-CUT app

As seen in the Data section, the T_HELPER table contains all the necessary information for functionalities in the application. Joining the views, which pull the necessary information onto this table, would slow down queries involving this table significantly. Especially since the views like the V_GROUPEMENTS in listing B.5 are very heavy in computation. Thus a better solution is to update the table each day with the automation module. This keeps the constantly used T_HELPER table fast and lightweight for users and the automation module can perform the heavy queries once per day. These automations are set to run each day at 5:00 am.

With the "AddnewDossier" PL/SQL script in listing 3.3 new rows are inserted into the T_HELPER table so the application can work with them.

```
1 BEGIN
2 INSERT INTO T_HELPER (DOSSIER)
3 SELECT DOSSIER FROM V_ADM_FAC WHERE DOSSIER NOT IN (SELECT DOSSIER FROM T_HELPER);
4 END;
```

List. 3.3: Updating the T_HELPER TABLE

In the T_HELPER table all the additional information columns need to be updated. The "AddGroupementsGestion" and "AddGroupementsY" scripts in listing B.12 and listing B.13 fill the "GROUPEMENT" column with the Visa of the potential groupement case, or a "y" if the case was not already validated.

The medical report information from the medfolio section is also merged into the T_HELPER table. By running the script in listing B.14, which contains the combined scripts for inserting all the different report states, is the information then accessible in the T_HELPER table. The three status indicators "F", "D" and "P" show the readiness of the medical documentation per case. "F" stems from the term "figé" in the system and is the indicator that the documentation is ready for coding. "D" is only a "definitive" report which has not been signed off by a doctor. These reports can sometimes be used for coding as they are rarely missing important information. "P" stands for provisionally and is generally insufficient to code a dossier with.

Finally, the "UPDATE_VALIDATED_GROUPEMENTS" script adds an indicator to all cases which are a validated groupement. From the Groupement validation section the script in listing B.15 inserts an identifier into all validated groupemnt cases in the T_HELPER table.

3.2. Home page

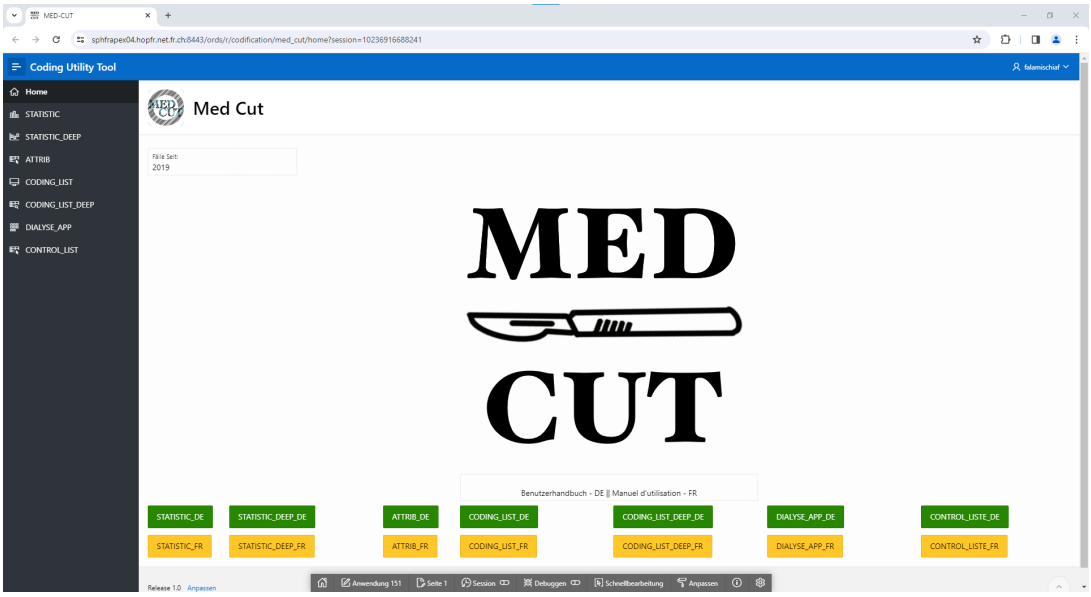

Fig. 3.5.: MED-CUT Home page

3.2.1. Home page functionality

The MED-CUT homepage serves, as homepages do, as the first stop in the application. As the authentication schema bypasses the login screen entirely, users who access the MED-CUT application link will be directed to the home page. It displays the MED-CUT logo and the time frame of all cases within the app. At the bottom users can open user manuals for all pages in both French and German.

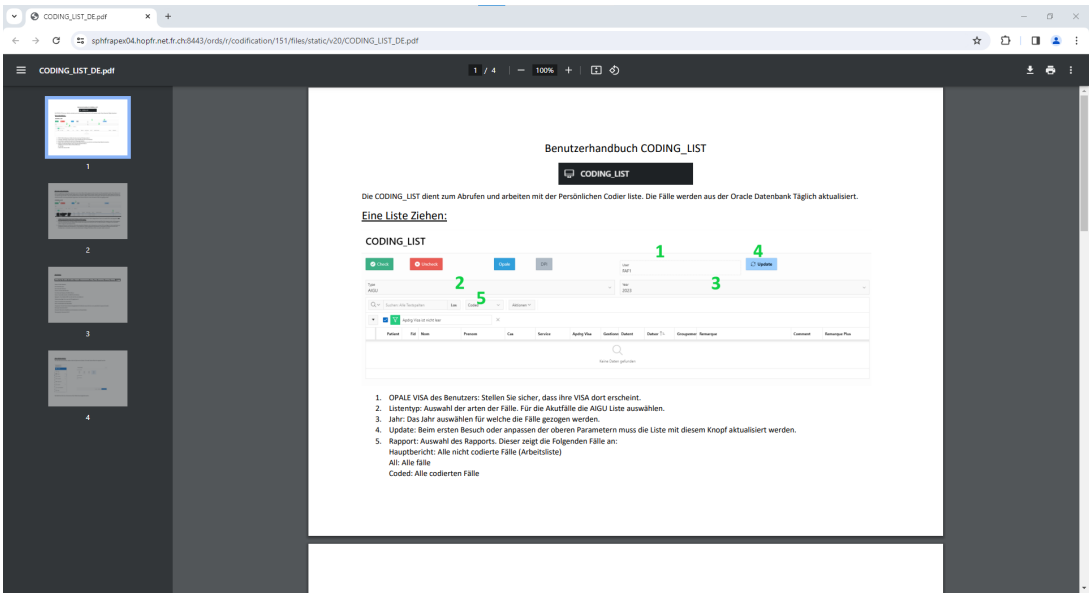

Fig. 3.6.: MED-CUT User Manuals

As seen in figure 3.6 the manuals open a pdf file in a new tab in the user's browser. These manuals contain a short explanation with illustrations on how to use each page for users.

3.2.2. Home page detail

The MED_CUT logo is a png file which is stored in the application shared components and is rendered together with the other page components.

The display which shows the time frame of cases consists of a display only APEX object which pulls from a SQL query. The query selects the MIN value of CODEYEAR to see the oldest year cases can be from.

The manual buttons are links, which point to a shared component. With the JavaScript in listing 3.4 as a button function it causes it to open the resource it points to in a new tab on the users browser.

```
1 javascript:window.open("#APP_FILES#CODING_LIST_DE.pdf","_new");
```

List. 3.4: Javascript to open a pdf resource in a new tab

3.3. Statistics page

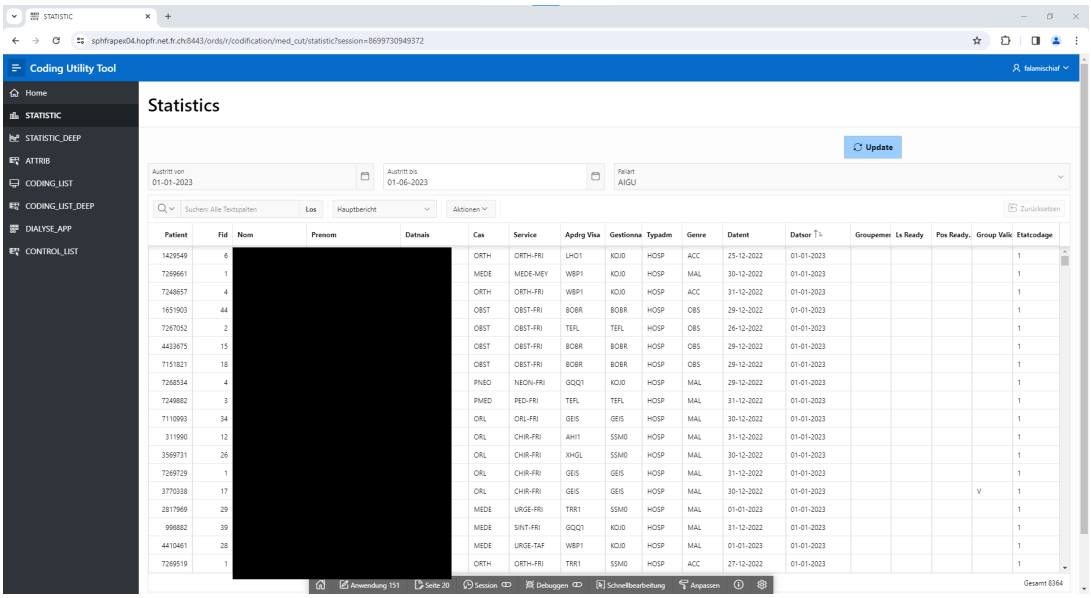

Fig. 3.7.: MED-CUT Statistics page

3.3.1. Statistics page functionality

The STATISTICS page allows for users to pull lists of dossiers in a very fast and intuitive way. The topmost selection fields allow to pick the patient exit date from and to a chosen date. The Patient field allows to chose which type of cases are pulled from the database. With the refresh button the corresponding list will be pulled. Core functionalities of the interactive grid object provided by APEX allow for hiding/showing of individual columns, filtering, and ordering of the data directly in the browser. Pre-built reports are available to all users. The main purpose of this page is for administrative workers to have an easy way to have an overview over all cases. Additionally, with the comprehensive filter and sorting functionalities users can easily pull statistically relevant list from the application and export them into Excel sheets.

3.3.2. Statistics page detail

The statistics page consists of an interactive grid which displays the query of listing 3.5. After constructing the V_COMPLETE view in the Data section not much more SQL is needed here. The :P20_DATE_END date picker and :P20_TYPE selection list APEX objects are used to have a dynamic listing of cases within a time frame and a group of cases. Users set those items and click on the refresh button, which submits those items and runs the query. This concept of having user pick options from select lists and date picker objects in order to have a dynamic SQL query displayed in their interactive grid is reused in most pages.

```
1 SELECT
2 DOSSIER,PATIENT,FID,NOM, PRENOM,DATNAIS,CAS,SERVICE,APDRG_VISA,STATUTCODAGE,
    GESTIONNAIRECODAGE,TYPADM,GENRE,DATENT,DATSOR,GROUPEMENT,LS_READY,POS_READY,
    GROUP_VALID,ETATCODAGE
```

```
3 FROM V_COMPLETE
4 WHERE DATSOR <= :P20_DATE_END AND DATSOR >= :P20_DATE_START AND
5 CAS IN (SELECT CAS FROM T_CAS_TYPES WHERE TYPE LIKE :P20_TYPE);
```

List. 3.5: Statistics page SQL query

It is important in this section to point out the potential for security risks. SQL injections can be a potential problem with dynamic SQL queries. SQL injections are usually possible when an application exposes an input line of SQL code to the user. Experienced users could at that point insert malicious/not intended SQL commands into the dynamic query. A safety mechanism, which is used throughout the application is the use of APEX items which only allow for their return values to be submitted to the SQL query.

3.4. Statistics deep page

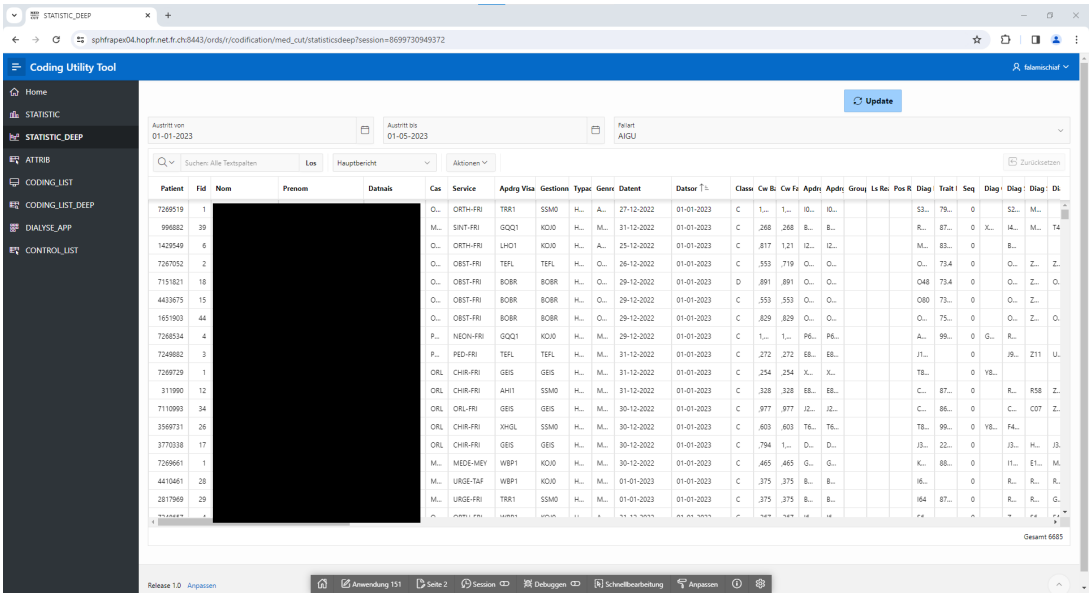

Fig. 3.8.: MED-CUT Statistics page

The Statistics deep page provides all the functionalities of the statistics page. The only difference lies in the accessed SQL query. Instead of drawing from the V_COMPLETE list it draws from the V_COMPLETE_OFS list, which contains the entire medical coding for each case. The separation of these two lists aims at providing a better performing access for the more often used pages. The statistics deep page pulls data significantly slower due to the additional 150 fields in the query. Its primary purpose is to allow for users to create lists based on criteria found in the medical coding. The feature to search for cases containing specific codes was highly requested by the medical coding team.

3.5. ATTRIB page

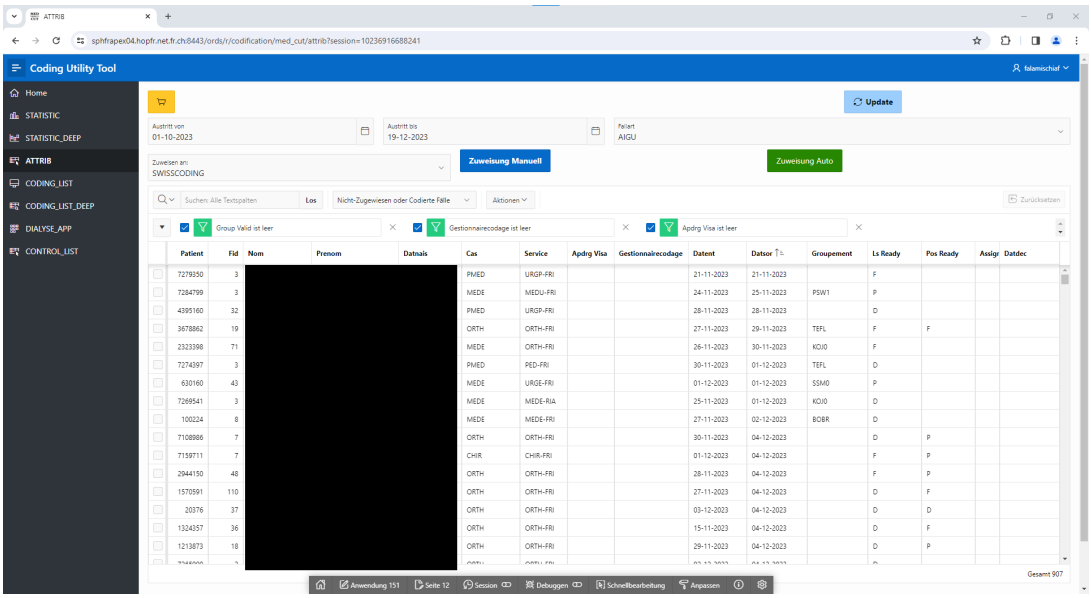

Fig. 3.9.: MED-CUT ATTRIB Page

3.5.1. ATTRIB page functionality

The core functionality of the ATTRIB page is to assign cases to different coders in a very easy and intuitive way. In addition it allows users to have a quick overview of all the not-coded and non-atributed cases. With the selection boxes, similar to the statistics page, can users select the date and type of cases pulled from the database. By default the primary report displays all cases which need to be assigned. The mechanism behind the assigning of cases was inspired by how shopping websites handle the purchase of articles. A shopping cart temporarily contains all cases which are to be assigned. This allows users to see in real time which cases they are about assign to whom. It also allows for easy on the go reassigning in either the shopping cart directly or in the pulled list.

There are two ways to assign cases from this list: By selecting cases individually with the checkbox, cases can then be assigned with the manual assign to the person selected by the selection box. This is intended as a quick and easy way to assign up to about 20 cases. An assigning of more cases then that will quickly become tedious. This assigning process has no imposed restrictions, meaning any type of case can be attributed to anyone. This is intentional, as this is the desired way to attribute cases which were not attributed with the automatic attribution feature. When pressing the auto attrib button a dialog page will open with the following options seen in figure 3.10.

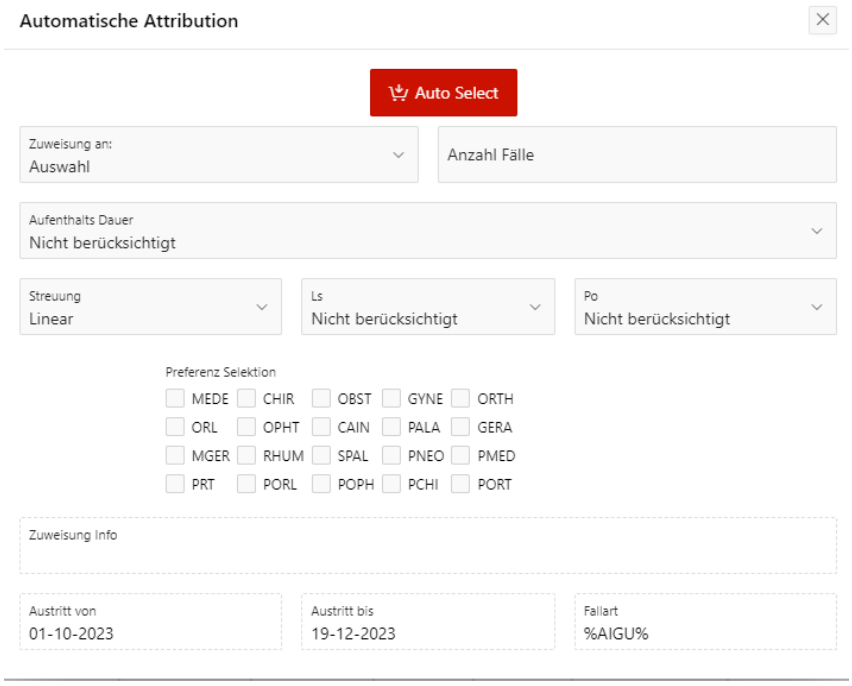

Fig. 3.10.: MED-CUT AUtomatic Attribution

From this window, users can select many different criteria, which then automatically assigns cases fulfilling those criteria to the selected Visa. From top left to bottom right the options are as follows; **Atribution Visa** specifies whom the cases should be assigned to. This also applies exclusion criteria given by the selected Visa. **Number of cases** specifies the desired number of cases, which should be assigned. **Limitation of Stay duration** is an exclusion criteria. When selected, no cases which have a stay of 50 or more days are automatically assigned. **Scattering** applies a slight scattering of cases in the selected range. **LS required** only assigns cases which have a ready medical report. **PO required** only assigns cases which have ready operation protocols. **Case selection** allows user to specify individual case types they want to assign. For a mixed list all case types should be checked. **Assigning Info** displays the exclusion rules of the selected Visa. **Exit date from** and **Exit date to** take the exit date range from the initial query on the ATTRIB page. **Case Type** siplays the selected case group from the ATTRIB page.

Once cases are assigned through either the manual or automatic assigning they are temporarily stored in a "shopping cart"

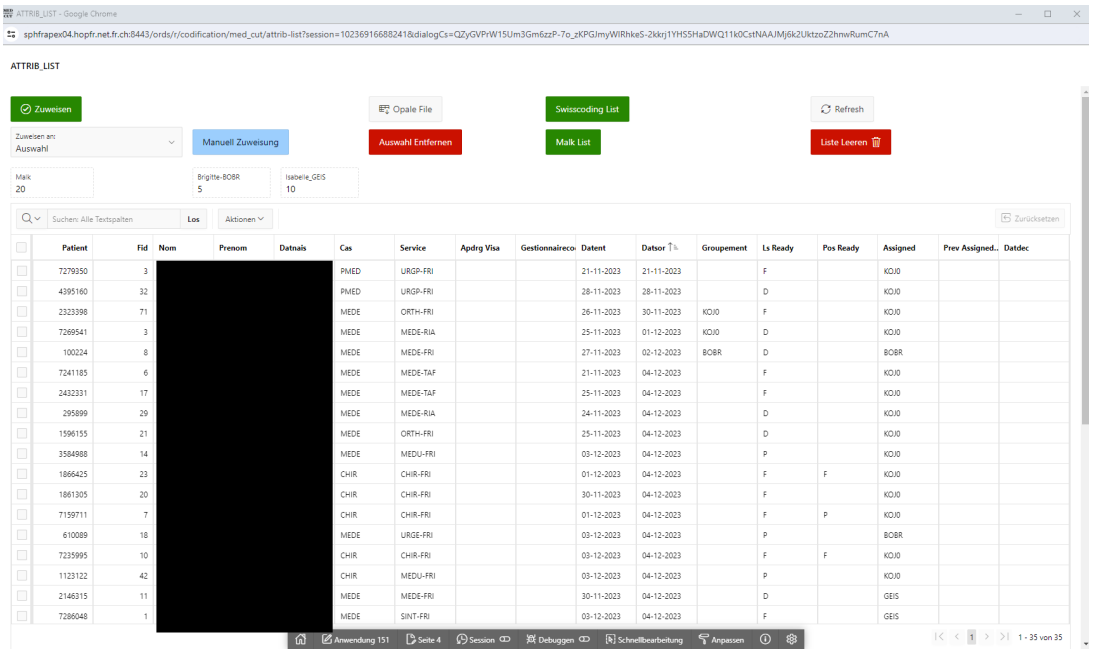

Fig. 3.11.: MED-CUT ATTRIB Shopping Cart

This is where the current list of assigned cases is displayed. The current number of assigned cases to a Visa can be seen at the top. By selecting cases with the checkbox and then pressing the assign button, they can be reassigned to the currently selected person in the selection box. The delete selection button will instead remove the cases from the shopping list and makes them available again for automatic assigning. This way, easy on the fly changes can be made directly in this list, without the need to search the cases on the ATTRIB page.

3.5.2. ATTRIB page detail

ATTRIB page

The displayed interactive grid on the ATTRIB page works similarly to the statistics page. The manual selection process works with a series of dynamic actions. The checkbox script stores the currently selected cases in a hidden APEX item, which will then be used to run a series of PL/SQL scripts. The selected rows have to be inserted into the "shopping cart", which consists of the T_ASSIGN table. Noteworthy in this script are the different steps required if the cases are already in the list. In that case they have to be reassigned to the newly selected Visa. Otherwise they are inserted into the table. The entire script can be seen in listing B.16. The automatic assigning process required a much larger PL/SQL script. In the listing 3.6 the attribution script with no scattering can be seen. It includes all the different option from the figure 3.10 automatic attribution selections. The sequential AND clauses make it so only cases with the correct criteria are selected. In the full script, which can be seen in listing B.17, a CASE statement is required to separate a query with and without scattering. The scattering query has an additional nested sub query. It scrambles the list of the selected cases so that they are randomly distributed.

```
1   --Without scattering
2   INSERT INTO T_ASSIGN (DOSSIER, PATIENT, FID, NOM, PRENOM, DATNAIS, CAS, SERVICE,
        APDRG_VISA, GESTIONNAIRECODAGE,DATENT, DATSOR, GROUPEMENT,LS_READY,POS_READY,
        ASSIGNED, PREV_ASSIGNED, DATDEC)
3   (SELECT COM.DOSSIER,COM.PATIENT,COM.FID,COM.NOM,COM.PRENOM,COM.DATNAIS,COM.CAS,COM
        .SERVICE,COM.APDRG_VISA,COM.GESTIONNAIRECODAGE,COM.DATENT,COM.DATSOR,COM.
        GROUPEMENT,COM.LS_READY,COM.POS_READY,:P5_ATTRIB,COM.ASSIGNED,COM.DATDEC
4   from V_COMPLETE COM
5   WHERE DATSOR <= :P5_DATE_END AND DATSOR >= :P5_DATE_START AND
6   CAS IN (SELECT CAS FROM T_CAS_TYPES WHERE TYPE LIKE :P5_TYPE) AND
7   CAS NOT IN (SELECT CAS FROM T_EXCLU_CAS WHERE EXCLU LIKE :P5_EXCLU) AND
8   SERVICE NOT IN (SELECT SERVICE FROM T_EXCLU_SERVICE WHERE EXCLU LIKE :P5_EXCLU)
        AND
9   ((:P5_LS = 'y' AND LS_READY = 'F') OR (:P5_LS = 'n')) AND
10  ((:P5_PO = 'y' AND (POS_READY = 'F' OR POS_READY IS NULL)) OR (:P5_PO = 'n')) AND
11  ((:P5_DAUER = 'y' AND (DATSOR - DATENT) <= '50') OR (:P5_DAUER = 'n')) AND
12  APDRG_VISA IS NULL AND
13  GESTIONNAIRECODAGE IS NULL AND
14  ASSIGNED IS NULL AND
15  GROUP_VALID IS NULL AND
16  (GROUPEMENT = :P5_ATTRIB OR GROUPEMENT IS NULL) AND
17  CAS MEMBER OF (SELECT apex_string.split (p_str => :P5_SELECT, p_sep => ':') FROM
        dual)
18  ORDER BY DATSOR ASC
19  FETCH FIRST :P5_NUMBER ROWS ONLY);
```

List. 3.6: Excerpt of the automatic attribution script

After an automatic selection the selected cases have to be merged into the T_HELPER table from the T_ASSIGN table in the same way as with the manual attribution from listing B.16. The page runs the highlight_ig_cells() command on refresh, which allows for rows with an ASSIGNED Visa to be clearly visible.

Shopping cart

The Shopping cart displays the T_ASSIGN Table in an interactive grid to show which cases are currently being assigned. The page renders display only objects, which show how many cases are assigned to which Visa. These objects only appear dynamically when their value is larger then 0. Functionalities to re-assign or delete cases from this list work similarly to the ATTRIB page. An important difference lies in the PREV_ASSIGNED column, which is used to store the previously assigned Visa. This is to prevent cases from changing their assigned Visa if they are deleted from the T_ASSIGN table after reassigning Visas. When deleting from the T_ASSIGN table they are thus restored to their PREV_ASSIGN Visa. The full scripts can be seen in listing B.18. Furthermore with the buttons "Malk" and "Swisscoding", the application navigates to a modal page which displays the lists in correct format for printing. This is useful for creating formatted lists, which can then be sent to the external coding enterprises that work for the HFR.

3.6. Coding list

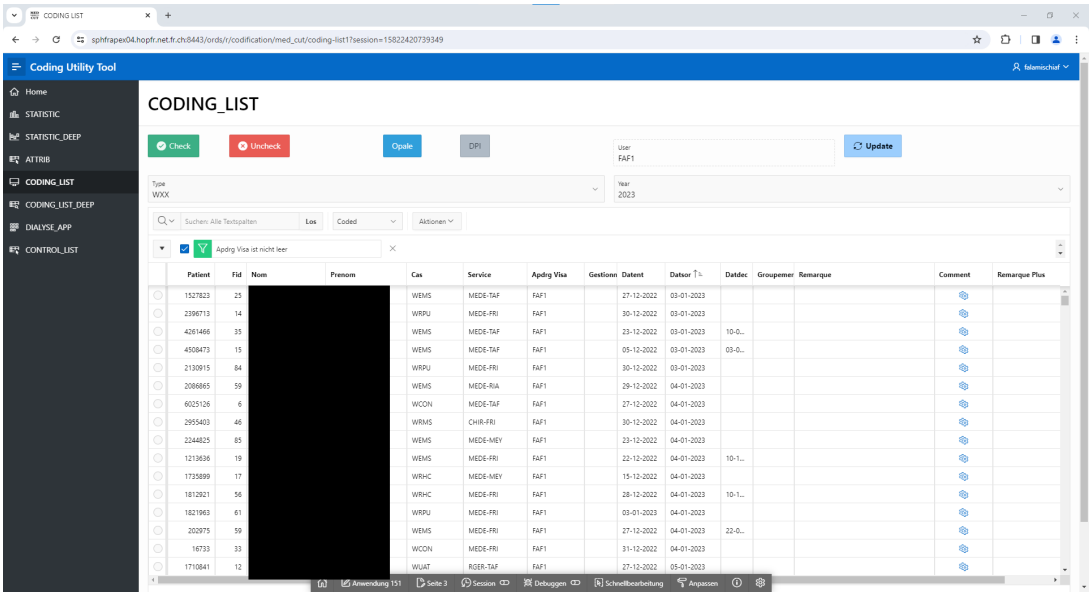

Fig. 3.12.: MED-CUT Coding List Page

3.6.1. Coding list functionality

The coding list page is intended to be used by medical coders to have a personal list of their cases. By providing useful administrative functionalities and a customizable list this should facilitate the medical coding workflow. A user will have a list of all cases, which are assigned to his/her own Visa on this page. The standard report filters only display non-validated cases. However, should a coder desire to search in already validated cases they can easily change the filters themselves. Feedback from a test user has confirmed that the functionality of seeing the state of medical documentation is especially useful. It saves a lot of time for medical coders who no longer need to open individual cases and check for the documentation themselves. The buttons "Check" and "Uncheck" are used to mark cases as completed for the day. If a case is validated in the administrative software Opale, they will disappear from their default coding list the day after. This is due to the overnight updating of the data. This can be used as good validation check, as the next day there should be no more green cases in their list. Additionally, each case has two commentary fields, which coders can fill with their own remarks. In figure 3.13 the functionalities can be seen in action.

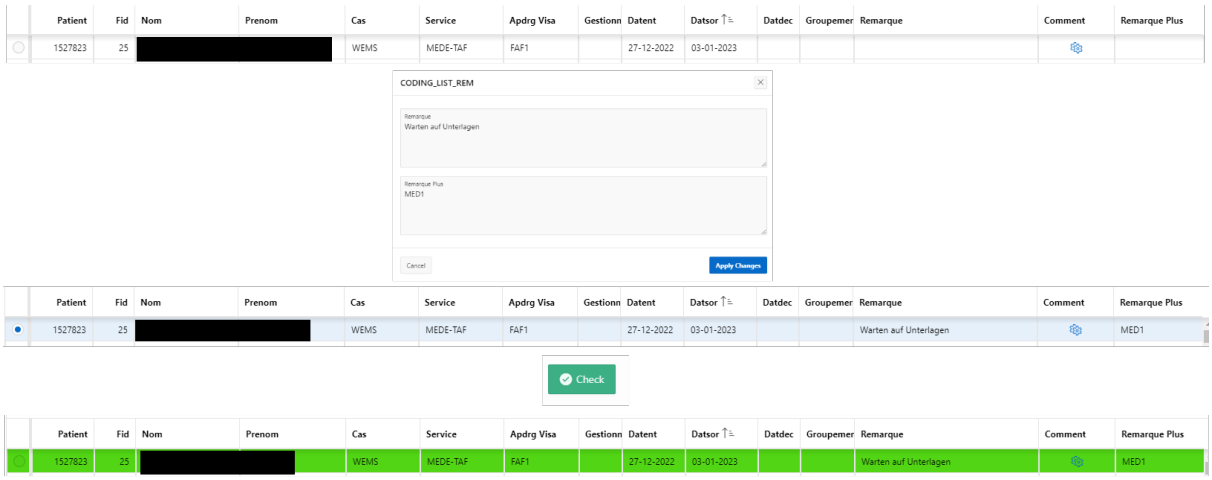

Fig. 3.13.: Additional functionality of the Coding List

Lastly, the buttons "Opale" and "DPI" copy the selected case's patient identification in a specific format to the clipboard. As medical coders need to open patient dossiers in different applications, this makes it so they can press the corresponding button and paste the patient identifications.

3.6.2. Coding list detail

The coding list query does have a few particularities. As seen in listing 3.7, the displayed cases need to have the :P3_VISA APEX object in their respective assigned Visas (GESTIONNAIRECODAGE). This object is set through a SQL query which sets it to the Visa corresponding to the :APP_USER). This allows for users, who have a Visa, to see their respective list. If a user has no Visa in the T_LIST_ASSIGNVISA table then they won't have any entries in their list.

```
1 SELECT
2 DOSSIER,PATIENT,FID,NOM,PRENOM,DATNAISDATDEC,CAS,SERVICE,APDRG_VISA,GESTIONNAIRECODAGE
      ,DATENT,DATSOR,GROUPEMENT,LS_READY,POS_READY,
3 '<span class="t-Icon fa fa-gears" aria-hidden="true"></span>' LINKCOLUMN,
4 DONE,REMARQUE,REMARQUE_PLUS,GROUP_VALID,ETATCODAGE
5 from V_COMPLETE
6 WHERE CAS IN (SELECT CAS FROM T_CAS_TYPES WHERE TYPE LIKE :P3_TYPE) AND
7 EXTRACT(YEAR FROM DATSOR) = :P3_YEAR AND
8 (GESTIONNAIRECODAGE = :P3_VISA OR APDRG_VISA = :P3_VISA);
```

List. 3.7: SQL Qery for the CODING_LIST page

The script responsible for copying the patient identification number of cases in specific formats can be seen in listing B.25.

3.7. Coding list deep

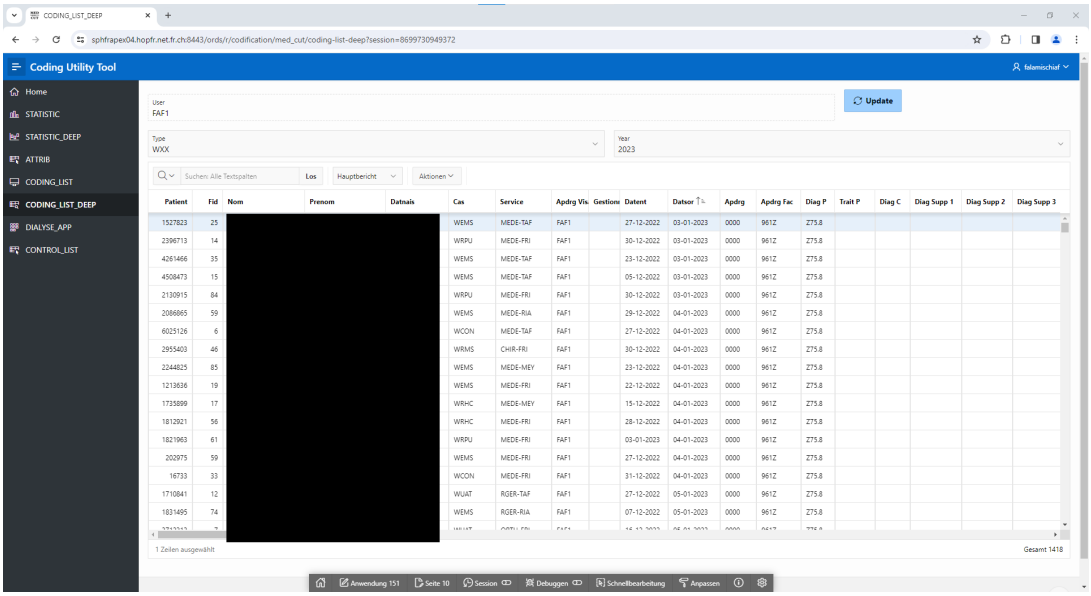

Fig. 3.14.: MED-CUT Coding List Deep

As with the Statistic and Statistic Deep pages the main difference lies in the accessed query. Coding List Deep pulls its cases for its interactive grid from the V_COMPLETE_OFS view and thus contains the entire medical coding. This page is intended as a tool for coders to search through their validated cases by medial codes. Since this page is not intended to be used as the working list of coders it does not have all the functionalities of the regular Coding list. Checking and commenting cases can only be done on the Coding list page.

3.8. Dialyse page

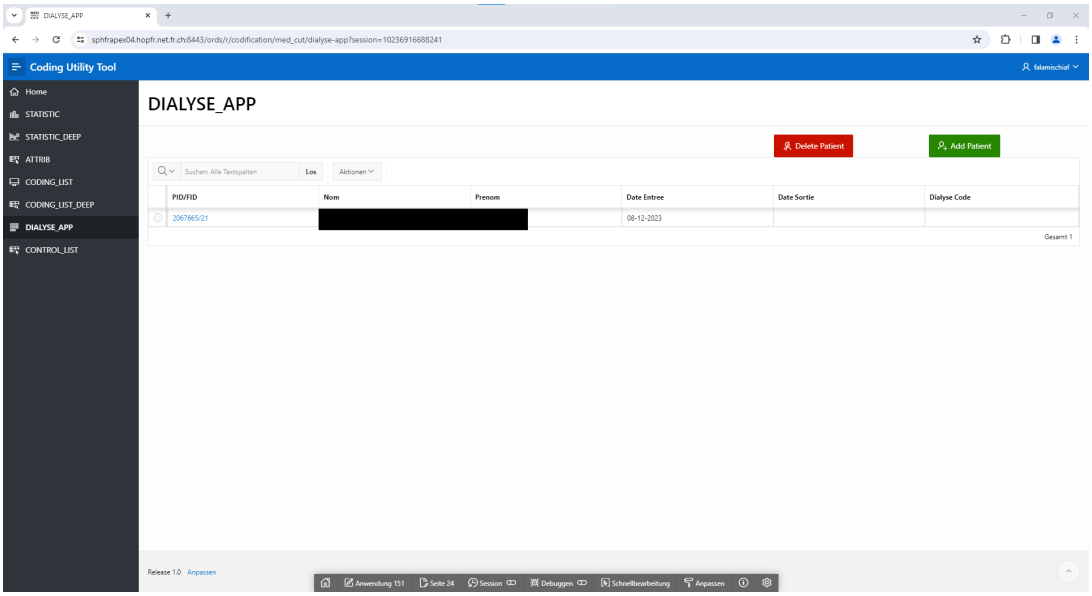

Fig. 3.15.: MED-CUT Dialyse page

3.8.1. Dialyse page functionality

The Dialyse page is intended for use for the intensive care case manager and aims to facilitate the calculation of dialysis times and codes. The Dialyse page consists of a Interactive grid drawing from the T_DIALYSE_APP table. Its intended use is to store all patients whose dialyses time need to be calculated and stored. Instead of having the user enter the patient data themselves, they can search the patient list via the Add Patient button. The Button Navigates the user to the DIALYSE_ADD modal page seen in figure 3.16.

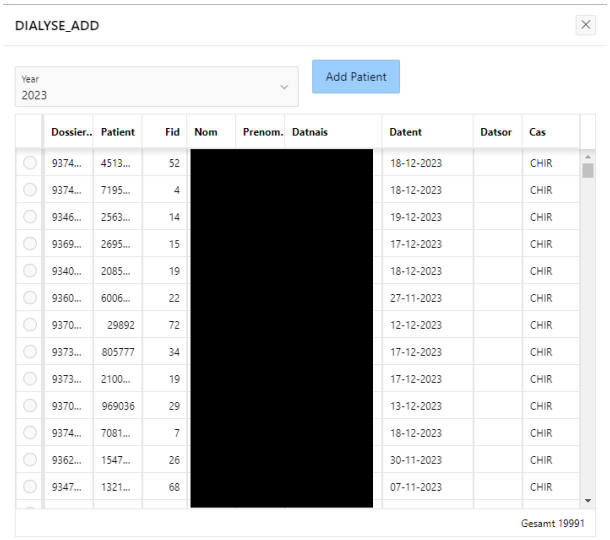

Fig. 3.16.: MED-CUT Add Dialyse Patient Modal Page

Once a patient is selected with the checkbox the Add Patient button will create the corresponding entry of that patient in the Dialyse_APP table. From there each patient can be opened by clicking on their line to open their dailysis form page seen in figure 3.17.

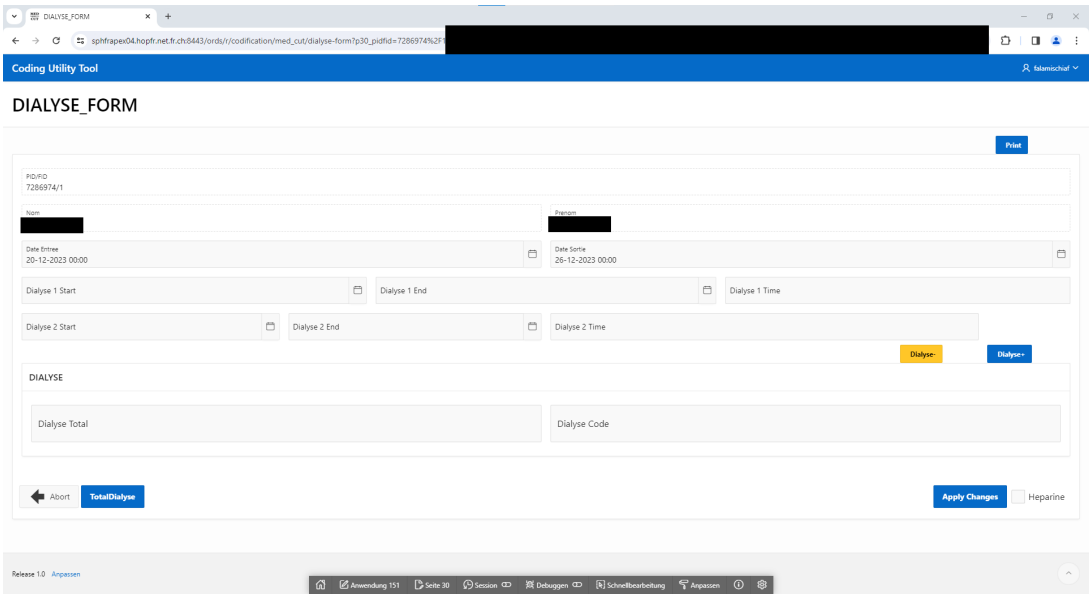

Fig. 3.17.: MED-CUT Dialyse Form Page

Into this form, users can enter the dialysis times for that patient. By default only one line of dialysis treatment is displayed. With the "Dialyse+" button an additional row of treatment can be displayed. In the same manner "Dialayse"- will hide the bottom most dialysis row. After the user has entered all dialysis times, they can press the button "TotalDialyse" to have the application calculate the total dialysis time and code.

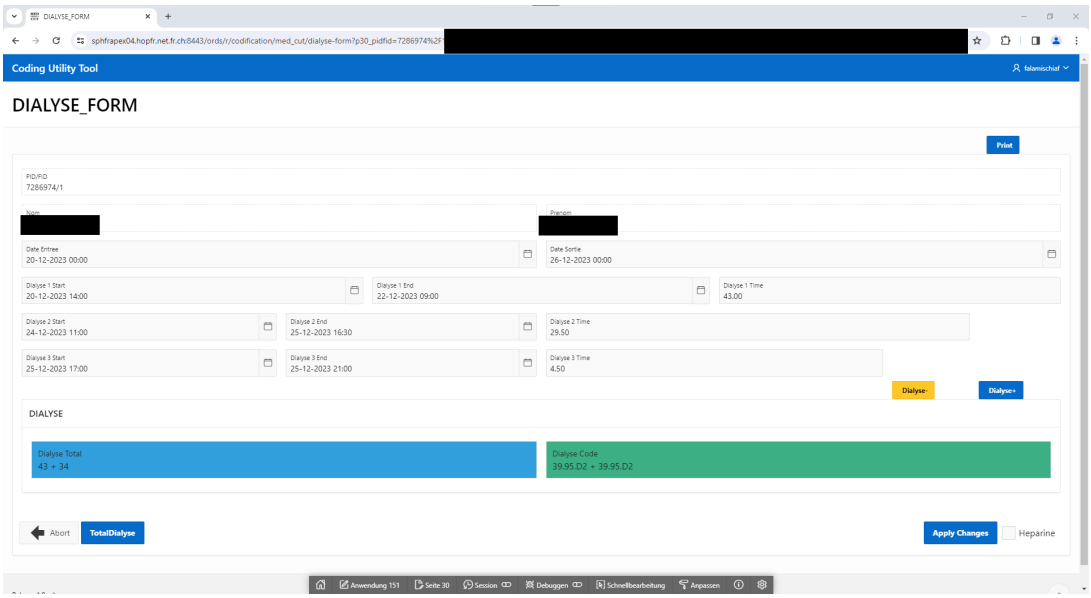

Fig. 3.18.: MED-CUT Dialyse Form Calculation Page

The rules for correct calculation of dialysis time are not trivial. There are specific rules as to when to count pauses between treatments. Pauses in between treatments have to be ignored if the pause is less then 4 hours. If the pause lasts between 4 and 24 hours the pause has to be subtracted. Pauses longer then 24 hours result in a new dialysis treatment code. Having an application which calculates these times has proven to be very useful. After calculating the total dialysis of a case the "Print" button opens the browser print dialogue to print the form page in a pleasant to read pdf.

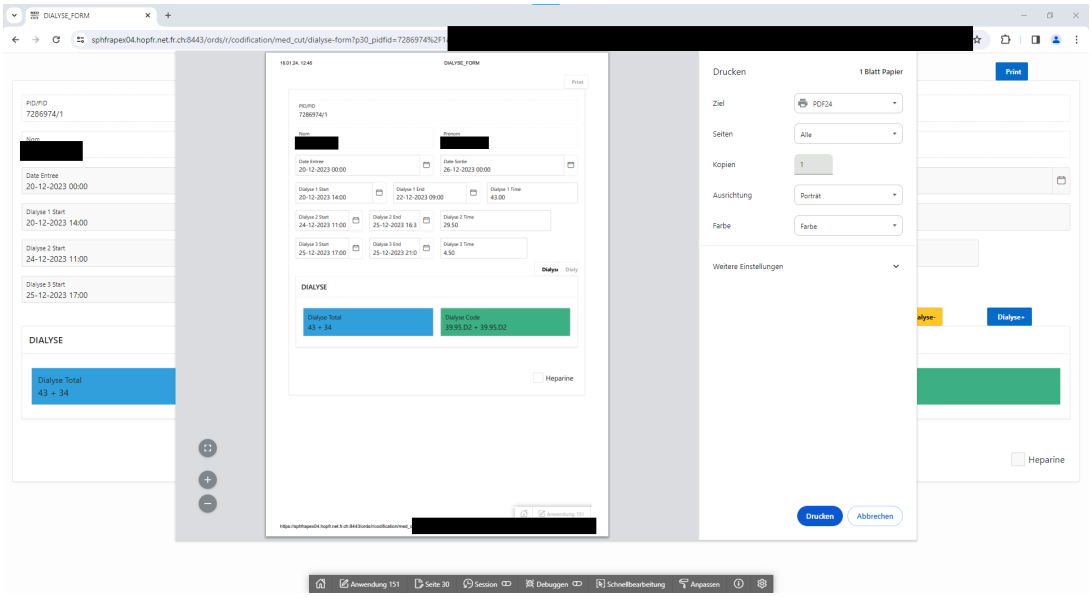

Fig. 3.19.: MED-CUT Dialyse Print Page

3.8.2. Dialyse page detail

In order to have a list of patients, which include still stationed ones, a different query then with the other pages is needed. As seen in listing B.19 the page pulls not from the V_COMPLETE view but rather from the V_DIALYSE_DOSSIERS view. This view is not separately listed as it is very similar to the V_ADM_FAC view, with the only difference being that not-exited patients are included as well. With the help of the filter function users can easily search for a specific patient. After selecting a patient the Checkbox selection script and the query in listing B.20 add the patient to the list. Deleting a patient works similarly with a delete PL/SQL script in listing B.21.

A dynamic action which triggers upon closing the dialog page refreshes the Dialyse_APP table so that after adding a Patient the table will immediately display the addition. The patient column &PIDFID is linked to a form page where the dialysis times can be submitted. On the form page each dialysis line past the first has a start time, end time, pause type and total time fields. A JavaScript which calculates the respective dialysis times runs once the user presses the "TotalDialyse". The entire script for calculating the correct times is rather lengthy as it needed to include functions for the detection of pause types and all dialysis codes. The entire script is separated into 3 files. The listings B.26 provides the functions to calculate time differences between DATE formats and the pause types. The script from listing B.27 runs iteratively through all dialyses lines and calculates the total dialyses time in respect to their pause type. And finally the script in

listing B.28 sets the correct codes for the corresponding times. Once the correct times have been calculated the entire form can be printed with the script in listing B.29 to have a nice dialysis report.

4
Conclusion

4.1. Review

Regardless of how well the application seemed to work in simulated tests, it ultimately needed to be ready for everyday use by an entire team. As such the application underwent a two month testing phase, in which an administrative worker, intensive care case manager and medical coder used the app extensively. Many bugs could be fixed and several additional proposed features implemented as a result of the testing phase. Feedback for the application in its current state has been very positive. The administrative capabilities of managing and overseeing the distribution of medical dossiers have greatly sped up those processes. Feedback from the medical coder and case manager have also confirmed the value the application can bring to their respective tasks.

4.2. Final statements and outlook

Timing wise, the deployment of the application was not very ideal, as the workload for everybody in the team of medical coding is at its highest in the months November through January. These months are filled with the hectic work of finishing up the previous year and are as such not well suited for making large changes in the work methods. The team will thus start working entirely with the application in the middle of February. Hopefully, MED-CUT will be up for to the challenge and help the team to start their medical coding of the new year with enhanced efficiency.

A
Common Acronyms

HFR	Hôpital cantonal de fribourg
BFS	Bundesamt für Statistik
DRG	Diagnosis Related Groups
ICD	International Catalogue of Diseases
CHOP	Schweizerische Operationsklassifikation
URL	Uniform Resource Locator
DB link	Data Base Link
PL/SQL	Procedural Language / Structured Query Language
HTML	Hypertext Markup Language
XLIFF	XML Localization Interchange File Format
HTTP	Hypertext Transfer Protocol
REST	Representational State Transfer
SQL	Structured Query Language
URL	Uniform Resource Locator
XML	eXtensible Markup Language

B
Code Listings

SQL Tables

T_ASSIGN

Column Name	Data Type	Detail
DOSSIER	NUMBER	**Primary key**, Unique ID for a case
PATIENT	NUMBER	Unique ID for a patient
FID	NUMBER	File number for a patient
NOM	VARCHAR2(50 CHAR)	Name of a patient
PRENOM	VARCHAR2(50 CHAR)	First name of a patient
DATNAIS	DATE	Date of birth of a patient
CAS	VARCHAR2(10 CHAR)	Type of a case (ORTHO, CHIR etc.)
SERVICE	VARCHAR2(10 CHAR)	Service of a case (MEDEFRI etc.)
APDRG_VISA	VARCHAR2(10 CHAR)	Validation Visa of a case
GESTIONNAIRECODAGE	VARCHAR2(10 CHAR)	Assigned Visa of a case
DATENT	DATE	Begin of Hospitalisation Date
DATSOR	DATE	End of Hospitalisation Date
GROUPEMENT	VARCHAR2(10 CHAR)	Visa of previous case if groupement eligible
LS_READY	CHAR(1 BYTE)	State indicator of medical report
POS_READY	CHAR(1 BYTE)	State indicator of operation protocols
ASSIGNED	VARCHAR2(10 CHAR)	Visa of in-app assigning
PREV_ASSIGNED	VARCHAR2(10 CHAR)	Storing of previously assigned Visa
DATDEC	DATE	Date of patient death

Tab. B.1.: The T_ASSIGN table (shopping basket)

T_USERS_VISA

Column Name	Data Type	Detail
NAME	VARCHAR2(30 CHAR)	**Primary key** User login name (used by the apex application)
VISA	VARCHAR2(10 CHAR)	Visa of the corresponding user
RIGHTS	VARCHAR2(10 CHAR)	Per user rights

Tab. B.2.: The T_USERS_VISA table (storing of user Visa and apex login names for authorisation)

T_LIST_ASSIGNVISA

Column Name	Data Type	Detail
NAME	VARCHAR2(30 CHAR)	**Primary key** Full user name name
VISA	VARCHAR2(10 CHAR)	Visa of the corresponding user
ASSIGN	VARCHAR2(200 BYTE)	Text for displaying the exclusion cases

Tab. B.3.: The T_LIST_ASSIGNVISA table (storing of user Visa and login names)

T_EXCLU_CAS

Column Name	Data Type	Detail
CAS	VARCHAR2(10 CHAR)	**Primary key** Case type e.g. CHIR, PALA etc.
EXCLU	VARCHAR2(20 CHAR)	Identifiers for whom the cases are excluded e.g. -MALK-SWISS-

Tab. B.4.: The T_EXCLU_CAS table (exclusion case types)

T_EXCLU_SERVICE

Column Name	Data Type	Detail
SERVICE	VARCHAR2(10 CHAR)	**Primary key** SERVICE type e.g. CHIR-FRI, MEDE-TAF etc.
EXCLU	VARCHAR2(20 CHAR)	Identifiers for whom the cases are excluded e.g. -MALK-SWISS-

Tab. B.5.: The T_EXCLU_SERVICE table (exclusion service types)

T_DIALYSE_APP

Column Name	Data Type	Detail
PIDFID	VARCHAR2(20 BYTE)	**Primary key** PID and FID fusion
NOM	VARCHAR2(50 CHAR)	Name of a patient
PRENOM	VARCHAR2(50 CHAR)	First name of a patient
DATE_ENTREE	DATE	Begin of Hospitalisation Date
DATE_SORTIE	DATE	End of Hospitalisation Date
DIALYSE_1_S	DATE	Begin of first Dialysis
DIALYSE_1_E	DATE	End of first Dialysis
DIALYSE_1_T	NUMBER	Time of first Dialysis
DIALYSE_2_S	DATE	Begin of second Dialysis
DIALYSE_2_E	DATE	End of second Dialysis
DIALYSE_2_A	CHAR(1 BYTE)	Category of second dialysis (for calculating the pause)
DIALYSE_2_T	NUMBER	Time of second Dialysis
...	...	...
DIALYSE_7_T	NUMBER	Time of seventh Dialysis
DIALYSE_CODE	VARCHAR2(20 BYTE)	Dialysis CHOP-code
DIALYSE_TOT	NUMBER	Dialysis total time

Tab. B.6.: The T_DIALYSE_APP table (for storing the dialysis app cases)

SQL Views

The V_ADM_FAC view

```
1
2 CREATE OR REPLACE FORCE EDITIONABLE VIEW "V_ADM_FAC" ("DOSSIER", "PATIENT", "FID", "
    CAS", "SERVICE", "APDRG_VISA", "STATUTCODAGE", "GESTIONNAIRECODAGE", "TYPADM", "
    GENRE", "DATENT", "DATSOR", "CLASSE", "CW_BASE", "CW_FACT", "APDRG", "APDRG_FAC",
    "ETATCODAGE") AS
3  SELECT adm.DOSSIER, adm.PATIENT, adm.FID, adm.CAS, adm.SERVICE, fac.APDRG_VISA, fac.
     STATUTCODAGE, fac.GESTIONNAIRECODAGE, adm.TYPADM, adm.GENRE, adm.DATENT, adm.
     DATSOR, adm.CLASSE, fac.CW_BASE, fac.CW_FACT, fac.APDRG, fac.APDRG_FAC, fac.
     ETATCODAGE
4       FROM t_opale_adm@common adm, t_opale_adm_fac@common fac
5      WHERE adm.dossier = fac.DOSSIER
6            AND adm.TYPADM = 'HOSP'
7            AND --exclude Test Patients
8                (fac.PARTICULARITE != 'PTEST' OR fac.PARTICULARITE IS NULL)
9                --only include real exits (STATUT 1 are reservations)
10           AND adm.STATUT = '3'
11           AND --limit by DATSOR, only query all cases within the last 4 years, and
                 only patients with an Exitdate
12               EXTRACT (YEAR FROM DATSOR) >=
13               EXTRACT (YEAR FROM CURRENT_DATE) - 4
14   ORDER BY adm.DATSOR DESC;
```

List. B.1: V_ADM_FAC view

The V_POP view

```
1
```

```
2 CREATE OR REPLACE FORCE EDITIONABLE VIEW "V_POP" ("PATIENT", "NOM", "PRENOM", "
    DATNAIS", "DATDEC") AS
3  SELECT pop.PATIENT, pop.NOM, pop.PRENOM, pop.DATNAIS, pop.DATDEC
4     FROM t_opale_pop@common pop;
```

List. B.2: V_POP view

The V_OFS view

```
1
2 CREATE OR REPLACE FORCE EDITIONABLE VIEW "V_OFS" ("DOSSIER", "SEQ", "FLCODAGE", "
    DIAG_P", "DIAG_C", "DIAG_SUPP_1", ... "DIAG_SUPP_50", "TRAIT_P", "TRAIT_SUPP_1",
    ... "TRAIT_SUPP_100") AS
3  SELECT ofs.DOSSIER, ofs.SEQ, ofs.FLCODAGE, ofs.DIAG_P, ofs.DIAG_C, ofs.DIAG_SUPP_1,
4    ......................
5          ofs.DIAG_SUPP_50,
6          ofs.TRAIT_P, ofs.TRAIT_SUPP_1,
7    ......................
8          ofs.TRAIT_SUPP_100
9     FROM t_opale_ofs@common ofs
10    WHERE ofs.SEQ = 0;
```

List. B.3: V_OFS view (simplified)

The V_COMPLETE_OFS view

```
1
2  CREATE OR REPLACE FORCE EDITIONABLE VIEW "V_COMPLETE_OFS" ("DOSSIER", "PATIENT", "
    FID", "NOM", "PRENOM", "DATNAIS", "CAS", "SERVICE", "APDRG_VISA", "STATUTCODAGE"
    , "GESTIONNAIRECODAGE", "TYPADM", "GENRE", "DATENT", "DATSOR", "CLASSE", "
    CW_BASE", "CW_FACT", "APDRG", "APDRG_FAC", "GROUPEMENT", "LS_READY", "POS_READY"
    , "DONE", "REMARQUE", "REMARQUE_PLUS", "ASSIGNED", "GROUP_VALID", "SEQ", "DIAG_P
    ", "DIAG_C", "DIAG_SUPP_1", ... "DIAG_SUPP_50", "TRAIT_P", "TRAIT_SUPP_1", ... "
    TRAIT_SUPP_100") AS
3  SELECT adm.DOSSIER, adm.PATIENT, adm.FID, pop.NOM, pop.PRENOM, pop.DATNAIS, adm.CAS,
     adm.SERVICE, adm.APDRG_VISA, adm.STATUTCODAGE, adm.GESTIONNAIRECODAGE, adm.
    TYPADM, adm.GENRE, adm.DATENT, adm.DATSOR, adm.CLASSE, adm.CW_BASE, adm.CW_FACT,
     adm.APDRG, adm.APDRG_FAC, helper.GROUPEMENT, helper.LS_READY, helper.POS_READY,
     helper.DONE, helper.REMARQUE, helper.REMARQUE_PLUS, helper.ASSIGNED, helper.
    GROUP_VALID, ofs.SEQ, ofs.DIAG_P, ofs.DIAG_C,
4          ofs.DIAG_SUPP_1,
5    ......................
6          ofs.DIAG_SUPP_50,
7          ofs.TRAIT_P, ofs.TRAIT_SUPP_1,
8    ......................
9          ofs.TRAIT_SUPP_100
10      FROM V_ADM_FAC adm
11         LEFT JOIN V_POP pop ON pop.PATIENT = adm.PATIENT
12         LEFT JOIN V_OFS ofs ON adm.dossier = ofs.DOSSIER
13         LEFT JOIN T_HELPER helper ON adm.dossier = helper.DOSSIER
14   ORDER BY adm.DATSOR DESC;
```

List. B.4: V_COMPLETE_OFS view (simplified)

V_GROUPEMENTS

```
1
2  CREATE OR REPLACE FORCE EDITIONABLE VIEW "V_GROUPEMENTS" ("DOSSIER", "PREV_GESTION")
     AS
```

```
SELECT DOSSIER, PREV_GESTION
    FROM T_HELPER
         INNER JOIN
         (SELECT DOSSIER AS GROUPDOSSIER, PREV_GESTION
            FROM ( SELECT DOSSIER, PATIENT, FID, DATENT, DATSOR,
                          LAG (DATSOR)
                              OVER (PARTITION BY PATIENT ORDER BY DATENT)
                              AS PREV_SORT,
                          LAG (GESTIONNAIRECODAGE)
                              OVER (PARTITION BY PATIENT ORDER BY DATENT)
                              AS PREV_GESTION
                     FROM V_COMPLETE
                    WHERE CAS IN (SELECT CAS
                                    FROM T_CAS_TYPES
                                   WHERE TYPE LIKE '_AIGU')
                          AND PATIENT IN
                                 ( SELECT PATIENT
                                     FROM V_COMPLETE
                                    WHERE CAS IN (SELECT CAS
                                                    FROM T_CAS_TYPES
                                                   WHERE TYPE LIKE '_AIGU')
                                  GROUP BY PATIENT
                                    HAVING COUNT (PATIENT) > 1)
                 ORDER BY PATIENT)
           WHERE EXTRACT (YEAR FROM DATENT) = EXTRACT (YEAR FROM CURRENT_DATE)
                 AND EXTRACT (YEAR FROM PREV_SORT) = EXTRACT (YEAR FROM CURRENT_DATE)
                 AND DATENT - PREV_SORT < 18)
             ON T_HELPER.DOSSIER = GROUPDOSSIER;
```

List. B.5: V_GROUPEMENTS view

V_VALIDATED_GROUPEMENTS

```

CREATE OR REPLACE FORCE EDITIONABLE VIEW "V_VALIDATED_GROUPEMENTS" ("DOSSIER") AS
SELECT DOSSIER FROM t_opale_ofs@common WHERE SEQ = '99' AND FLCODAGE = '1';
```

List. B.6: V_VALIDATED_GROUPEMENTS view

V_YEAR

```

CREATE OR REPLACE FORCE EDITIONABLE VIEW "V_YEAR" ("CODEYEAR") AS
SELECT DISTINCT EXTRACT (YEAR FROM DATSOR) AS CODEYEAR
     FROM V_COMPLETE
  ORDER BY CODEYEAR DESC;
```

List. B.7: V_YEAR view

V_MEDFOLIO_CODING_LIST

```

CREATE OR REPLACE FORCE EDITIONABLE VIEW "V_MEDFOLIO_CODING_LIST" ("PID_FID", "
    LETTER", "LETTER_FREEZED", "PROTOCOLE", "PROTOCOLE_FREEZED") AS
SELECT PID_FID,
     LETTER,
     LETTER_FREEZED,
     PROTOCOLE,
     PROTOCOLE_FREEZED
```

```
8 FROM V_HFR_PATIENTS_CODING_LIST@RHFMFPROD_MF_CUSTOM.HOPFR.NET.FR.CH;
```

List. B.8: V_MEDFOLIO_CODING_LIST view

V_MEDFOLIO_DOSSIER_LS

```
1
2  CREATE OR REPLACE FORCE EDITIONABLE VIEW "V_MEDFOLIO_DOSSIER_LS" ("DOSSIER", "
       PID_FID", "LETTER", "LETTER_FREEZED") AS
3  SELECT V_COMPLETE.DOSSIER, PID_FID, LETTER, LETTER_FREEZED FROM
4  (SELECT PID_FID, MAX(LETTER) AS LETTER, MAX(LETTER_FREEZED) AS LETTER_FREEZED FROM
       V_MEDFOLIO_CODING_LIST GROUP BY PID_FID)
5  LEFT JOIN V_COMPLETE ON (V_COMPLETE.PATIENT || '-' || LPAD(V_COMPLETE.FID, 3, 0)) =
       PID_FID;
```

List. B.9: V_MEDFOLIO_DOSSIER_LS view

V_MEDFOLIO_DOSSIER_PO

```
1
2  CREATE OR REPLACE FORCE EDITIONABLE VIEW "V_MEDFOLIO_DOSSIER_PO" ("DOSSIER", "
       PID_FID", "PROTOCOLE", "PROTOCOLE_FREEZED") AS
3  SELECT V_COMPLETE.DOSSIER, PID_FID, PROTOCOLE, PROTOCOLE_FREEZED FROM
4  (SELECT PID_FID, MIN(PROTOCOLE) AS PROTOCOLE, MIN(PROTOCOLE_FREEZED) AS
       PROTOCOLE_FREEZED FROM V_MEDFOLIO_CODING_LIST GROUP BY PID_FID)
5  LEFT JOIN V_COMPLETE ON (V_COMPLETE.PATIENT || '-' || LPAD(V_COMPLETE.FID, 3, 0)) =
       PID_FID;
```

List. B.10: V_MEDFOLIO_DOSSIER_PO view

PL/SQL Scripts

Authorisation

```
1
2 BEGIN
3   IF PKG_COMMON_UTILS.LDAP_AUTHORIZATION@COMMON(v('APP_USER'), '
        ACT_APEX_CODIFICATION_ADMIN') THEN
4      RETURN TRUE;
5   ELSE
6      RETURN FALSE;
7   END IF;
8 END;
```

List. B.11: Authorisation PL/SQL

```
1 BEGIN
2 MERGE INTO T_HELPER USING V_GROUPEMENTS
3 ON (T_HELPER.DOSSIER = V_GROUPEMENTS.DOSSIER)
4 WHEN MATCHED
5    THEN
6    UPDATE
7    SET T_HELPER.GROUPEMENT = V_GROUPEMENTS.PREV_GESTION;
8 END;
```

List. B.12: Adding Groupements to the T_HELPER TABLE

```
BEGIN
MERGE INTO T_HELPER USING V_GROUPEMENTS
ON (T_HELPER.DOSSIER = V_GROUPEMENTS.DOSSIER)
WHEN MATCHED
    THEN
    UPDATE
    SET T_HELPER.GROUPEMENT = 'y' WHERE V_GROUPEMENTS.PREV_GESTION IS NULL;
END;
```

List. B.13: Adding Y-Groupements to the T_HELPER TABLE

```
LS_READY_F

BEGIN
MERGE INTO T_HELPER USING V_MEDFOLIO_DOSSIER_LS
ON (T_HELPER.DOSSIER = V_MEDFOLIO_DOSSIER_LS.DOSSIER)
WHEN MATCHED
    THEN
    UPDATE
    SET T_HELPER.LS_READY = 'F' WHERE V_MEDFOLIO_DOSSIER_LS.LETTER = 'yes' AND
        V_MEDFOLIO_DOSSIER_LS.LETTER_FREEZED = 'yes';
END;

LS_READY_D

SET T_HELPER.LS_READY = 'D' WHERE V_MEDFOLIO_DOSSIER_LS.LETTER = 'yes' AND
    V_MEDFOLIO_DOSSIER_LS.LETTER_FREEZED = 'no';

LS_READY_P

SET T_HELPER.LS_READY = 'P' WHERE V_MEDFOLIO_DOSSIER_LS.LETTER = 'no' AND
    V_MEDFOLIO_DOSSIER_LS.LETTER_FREEZED = 'no';

PO_RADY_F

BEGIN
MERGE INTO T_HELPER USING V_MEDFOLIO_DOSSIER_PO
ON (T_HELPER.DOSSIER = V_MEDFOLIO_DOSSIER_PO.DOSSIER)
WHEN MATCHED
    THEN
    UPDATE
    SET T_HELPER.POS_READY = 'F' WHERE V_MEDFOLIO_DOSSIER_PO.PROTOCOLE = 'yes' AND
        V_MEDFOLIO_DOSSIER_PO.PROTOCOLE_FREEZED = 'yes';
END;

PO_RADY_D

SET T_HELPER.POS_READY = 'D' WHERE V_MEDFOLIO_DOSSIER_PO.PROTOCOLE = 'yes' AND
    V_MEDFOLIO_DOSSIER_PO.PROTOCOLE_FREEZED = 'no';

PO_RADY_P

SET T_HELPER.POS_READY = 'P' WHERE V_MEDFOLIO_DOSSIER_PO.PROTOCOLE = 'no';
```

List. B.14: Adding Medical Report status to the T_HELPER TABLE

```
BEGIN
```

```
MERGE INTO T_HELPER USING V_VALIDATED_GROUPEMENTS
ON (T_HELPER.DOSSIER = V_VALIDATED_GROUPEMENTS.DOSSIER)
WHEN MATCHED
    THEN
    UPDATE
    SET T_HELPER.GROUP_VALID = 'V';
END;
```

List. B.15: Updating the T_HELPER table with validated groupements

```
--Set the newly assigned cases that are already in the
table

UPDATE T_ASSIGN
SET ASSIGNED = :P12_MANUAL
WHERE dossier
MEMBER OF (SELECT apex_string.split_numbers (p_str => :P12_DOSSIERS, p_sep => ':')
FROM dual);

--Insert newly assigned cases that are not already in the table

INSERT INTO T_ASSIGN (DOSSIER, PATIENT, FID, NOM, PRENOM, DATNAIS, CAS, SERVICE,
    APDRG_VISA, GESTIONNAIRECODAGE,DATENT, DATSOR, GROUPEMENT,LS_READY,POS_READY,
    ASSIGNED, PREV_ASSIGNED, DATDEC)
SELECT DOSSIER, PATIENT, FID, NOM, PRENOM, DATNAIS, CAS, SERVICE, APDRG_VISA,
    GESTIONNAIRECODAGE, DATENT, DATSOR, GROUPEMENT, LS_READY, POS_READY, :P12_MANUAL,
    ASSIGNED, DATDEC
FROM V_COMPLETE WHERE DATSOR >= :P12_DATE_START AND DATSOR <= :P12_DATE_END AND
DOSSIER NOT IN (SELECT DOSSIER FROM T_ASSIGN) AND
DOSSIER MEMBER OF (SELECT apex_string.split_numbers (p_str => :P12_DOSSIERS, p_sep =>
    ':') FROM dual);

--merge into T_HELPER

MERGE INTO T_HELPER USING T_ASSIGN
ON (T_HELPER.DOSSIER = T_ASSIGN.DOSSIER)
WHEN MATCHED
THEN UPDATE SET T_HELPER.ASSIGNED = T_ASSIGN.ASSIGNED;
```

List. B.16: Manual Attibution script

```
BEGIN

CASE
    --With scattering
    WHEN :P5_RANDOM = 'G' THEN

    INSERT INTO T_ASSIGN (DOSSIER, PATIENT, FID, NOM, PRENOM, DATNAIS, CAS, SERVICE,
        APDRG_VISA, GESTIONNAIRECODAGE,DATENT, DATSOR, GROUPEMENT,LS_READY,POS_READY,
        ASSIGNED, PREV_ASSIGNED, DATDEC)
    SELECT * FROM
    (SELECT * FROM
    (SELECT COM.DOSSIER,COM.PATIENT,COM.FID,COM.NOM,COM.PRENOM,COM.DATNAIS,COM.CAS,COM
        .SERVICE,COM.APDRG_VISA,COM.GESTIONNAIRECODAGE,COM.DATENT,COM.DATSOR,COM.
        GROUPEMENT,COM.LS_READY,COM.POS_READY,:P5_ATTRIB,COM.ASSIGNED,COM.DATDEC
    from V_COMPLETE COM
    WHERE DATSOR <= :P5_DATE_END AND DATSOR >= :P5_DATE_START AND
```

```
13     CAS IN (SELECT CAS FROM T_CAS_TYPES WHERE TYPE LIKE :P5_TYPE) AND
14     CAS NOT IN (SELECT CAS FROM T_EXCLU_CAS WHERE EXCLU LIKE :P5_EXCLU) AND
15     SERVICE NOT IN (SELECT SERVICE FROM T_EXCLU_SERVICE WHERE EXCLU LIKE :P5_EXCLU)
           AND
16     ((:P5_LS = 'y' AND LS_READY = 'F') OR (:P5_LS = 'n')) AND
17     ((:P5_PO = 'y' AND (POS_READY = 'F' OR POS_READY IS NULL)) OR (:P5_PO = 'n')) AND
18     ((:P5_DAUER = 'y' AND (DATSOR - DATENT) <= '50') OR (:P5_DAUER = 'n')) AND
19     APDRG_VISA IS NULL AND
20     GESTIONNAIRECODAGE IS NULL AND
21     ASSIGNED IS NULL AND
22     GROUP_VALID IS NULL AND
23     (GROUPEMENT = :P5_ATTRIB OR GROUPEMENT IS NULL) AND
24     CAS MEMBER OF (SELECT apex_string.split (p_str => :P5_SELECT, p_sep => ':') FROM
           dual)
25     ORDER BY DATSOR ASC
26     FETCH FIRST :P5_RANDOM_RANGE ROWS ONLY)
27     ORDER BY dbms_random.value
28     FETCH FIRST :P5_NUMBER ROWS ONLY)
29     ORDER BY DATSOR ASC;
30
31
32     ELSE
33
34     --Without scattering
35     INSERT INTO T_ASSIGN (DOSSIER, PATIENT, FID, NOM, PRENOM, DATNAIS, CAS, SERVICE,
           APDRG_VISA, GESTIONNAIRECODAGE,DATENT, DATSOR, GROUPEMENT,LS_READY,POS_READY,
           ASSIGNED, PREV_ASSIGNED, DATDEC)
36     (SELECT COM.DOSSIER,COM.PATIENT,COM.FID,COM.NOM,COM.PRENOM,COM.DATNAIS,COM.CAS,COM
           .SERVICE,COM.APDRG_VISA,COM.GESTIONNAIRECODAGE,COM.DATENT,COM.DATSOR,COM.
           GROUPEMENT,COM.LS_READY,COM.POS_READY,:P5_ATTRIB,COM.ASSIGNED,COM.DATDEC
37     from V_COMPLETE COM
38     WHERE DATSOR <= :P5_DATE_END AND DATSOR >= :P5_DATE_START AND
39     CAS IN (SELECT CAS FROM T_CAS_TYPES WHERE TYPE LIKE :P5_TYPE) AND
40     CAS NOT IN (SELECT CAS FROM T_EXCLU_CAS WHERE EXCLU LIKE :P5_EXCLU) AND
41     SERVICE NOT IN (SELECT SERVICE FROM T_EXCLU_SERVICE WHERE EXCLU LIKE :P5_EXCLU)
           AND
42     ((:P5_LS = 'y' AND LS_READY = 'F') OR (:P5_LS = 'n')) AND
43     ((:P5_PO = 'y' AND (POS_READY = 'F' OR POS_READY IS NULL)) OR (:P5_PO = 'n')) AND
44     ((:P5_DAUER = 'y' AND (DATSOR - DATENT) <= '50') OR (:P5_DAUER = 'n')) AND
45     APDRG_VISA IS NULL AND
46     GESTIONNAIRECODAGE IS NULL AND
47     ASSIGNED IS NULL AND
48     GROUP_VALID IS NULL AND
49     (GROUPEMENT = :P5_ATTRIB OR GROUPEMENT IS NULL) AND
50     CAS MEMBER OF (SELECT apex_string.split (p_str => :P5_SELECT, p_sep => ':') FROM
           dual)
51     ORDER BY DATSOR ASC
52     FETCH FIRST :P5_NUMBER ROWS ONLY);
53
54 END CASE;
55
56 END;
```

List. B.17: Automatic Attribution script

```
1
2 UPDATE T_ASSIGN
3 SET ASSIGNED = PREV_ASSIGNED
```

```
WHERE dossier
MEMBER OF (SELECT apex_string.split_numbers (p_str => :P4_DOSSIERS, p_sep => ':')
FROM dual);

MERGE INTO T_HELPER USING T_ASSIGN
ON (T_HELPER.DOSSIER = T_ASSIGN.DOSSIER)
WHEN MATCHED
THEN UPDATE SET T_HELPER.ASSIGNED = T_ASSIGN.ASSIGNED;

DELETE FROM T_ASSIGN
WHERE dossier
MEMBER OF (SELECT apex_string.split_numbers (p_str => :P4_DOSSIERS, p_sep => ':')
FROM dual);
```

List. B.18: Updating and deleting in the T_ASSIGN table (shopping basket)

```
SELECT
DOSSIER, PATIENT, FID, NOM, PRENOM, DATNAIS, CAS, DATENT, DATSOR
FROM V_DIALYSE_DOSSIERS
WHERE (EXTRACT(YEAR FROM DATSOR) = :P25_YEAR OR (EXTRACT(YEAR FROM DATENT) = :P25_YEAR
     AND DATSOR IS NULL)) AND
CAS IN (SELECT CAS FROM T_CAS_TYPES WHERE TYPE LIKE '%AIGU%')
```

List. B.19: Query for adding a Patient to the Dialyse Page

```
BEGIN
INSERT INTO T_DIALYSE_APP (PIDFID, NOM, PRENOM, DATE_ENTREE, DATE_SORTIE)
SELECT PATIENT || '/' || FID, NOM, PRENOM, DATENT, DATSOR
FROM V_DIALYSE_DOSSIERS WHERE PATIENT || '/' || FID NOT IN (SELECT PIDFID FROM
    T_DIALYSE_APP) AND
DOSSIER MEMBER OF (SELECT apex_string.split_numbers (p_str => :P25_DOSSIER, p_sep => '
    :') FROM dual);
END;
```

List. B.20: Adding a Patient to the Dialyse Page

```
BEGIN
DELETE FROM T_DIALYSE_APP WHERE PIDFID MEMBER OF (SELECT apex_string.split (p_str => :
    P24_PIDFID, p_sep => ':') FROM dual);
END;
```

List. B.21: Deleting a Patient from the Dialyse Page

Java Scripts and XML

#APP_FILES#igUtil#MIN#.js Checkbox script static file

```
/**
* @namespace var igUtil = {};
**/
var igUtil = {};

/**
* @function selectedPKs
```

```
* @example igUtil.selectedPKs("customers", "P10_CUSTOMER_IDS", "Please select at least
    one customer");
**/
igUtil.selectedPKs = function (IGStaticId, returnPageItem, minSelectionMsg) {
  // Get the Interactive Grid View
  var gridView = apex.region(IGStaticId).widget().interactiveGrid("getViews").grid;
  // Get the currently seledcted/checked records from the IG view
  var records = gridView.getSelectedRecords();
  // Create Array of Primary Key Values (getRecordId) from the selected records
  var ids = records.map(function(r) { return gridView.model.getRecordId(r); } );
  // Populate APEX Page Item with the selected IDs delimited with a ':'
  apex.item(returnPageItem).setValue( ids.join(":") );

  // If minSelectionMsg is populated then user must select at least one item.
  if (ids.length === 0 && minSelectionMsg) {
    // User did not select at least 1 record, so show the error message and return
       false.
    apex.message.clearErrors();
    apex.message.showErrors([
     {type: "error",
      location: "page",
      message: minSelectionMsg,
      unsafe: false}]);
    return false;
  } else {
    // All good.
    return true;
  }
}
```

List. B.22: Static file javascript for the Checkbox selection

Checkbox script example function call

```

var igUtil={selectedPKs:function(e,t,r){var a=apex.region(e).widget().interactiveGrid(
    "getViews").grid,i=a.getSelectedRecords().map((function(e){return a.model.
    getRecordId(e)}));return apex.item(t).setValue(i.join(":")),apex.debug.info("IG
    Region Static ID: "+e),apex.debug.info("Return Page Item: "+t),apex.debug.info("
    Count Selected IDs: "+i.length),apex.debug.info("Selected IDs: "+i.join(":")),0!==
    i.length||!r||(apex.message.clearErrors(),apex.message.showErrors([{type:"error",
    location:"page",message:r,unsafe:!1}]),!1)}};
```

List. B.23: Call of the function for a page

Translation File example

```

<?xml version="1.0" encoding="UTF-8"?>
<!--
  ******************
  ** Source : 144
  ** Source Lang: en-ch
  ** Target : 1
  ** Target Lang: de-ch
  ** Filename: f144_1_en-ch_de-ch.xlf
  ** Generated By: FALAMISCHIAF
  ** Date: 23-NOV-2023 10:14:31
  ******************
```

```
 -->
<xliff version="1.0">
<file original="f144_1_de-ch_de-ch.xlf" source-language="en-ch" target-language="de-ch
    " datatype="html">
<header></header>
<body>
<trans-unit id="S-5-0-144">
<source>Global Page</source>
<target>Global Page</target>
...
```

List. B.24: Excerpt of translation file german

```
//Function to copy Opale number to clipboard
function copy_Opale(){
    cellData = $(this).text();
    cellData.select();
    cellData.setSelectionRange(0,999);
    navigator.clipboard.writeText(cellData.value);
    };

--PL/SQL script to copy Opale number to clipboard
DECLARE
opale_ID varchar(20);

BEGIN
SELECT (PATIENT || '/' || FID) INTO opale_ID
FROM V_COMPLETE WHERE DOSSIER = :P3_DOSSIERS;

 apex_util.set_session_state('P3_OPALE', opale_ID);
    exception
    when others then
      apex_util.set_session_state('P3_OPALE', null);
END;


//On page a dynamic action then executes

navigator.clipboard.writeText(apex.item("P3_OPALE").getValue());
apex.message.showPageSuccess("Opale PID/FID");

//This copies the PID/FID in the correct format to clipboard

//FOr the DPI Number the same thing is done but for the different 000 format the
    following query is used instead

SELECT (PATIENT || '-' || LPAD(FID, 3, 0)) INTO opale_ID
FROM V_COMPLETE WHERE DOSSIER = :P3_DOSSIERS;
```

List. B.25: Scripts to copy Opale/DPI number to clipboard

```
//function used to calculate the time difference
function Datediff(d1, d2){
    if (d1 == "" || d2 == ""){return ""}

    parts1p1 = d1.split("-");
    parts1p2 = parts1p1[2].split(" ");
```

```
7     parts1p3 = parts1p2[1].split(":");
8
9     tdate1 = new Date(parts1p2[0],parts1p1[1]-1,parts1p1[0],parts1p3[0],parts1p3[1]);
10
11    parts2p1 = d2.split("-");
12    parts2p2 = parts2p1[2].split(" ");
13    parts2p3 = parts2p2[1].split(":");
14
15    tdate2 = new Date(parts2p2[0],parts2p1[1]-1,parts2p1[0],parts2p3[0],parts2p3[1]);
16
17    return ((tdate2 - tdate1)/(1000*60*60)).toFixed(2);
18
19 }
20
21 //d1 previous dialyse end, d2 current dialyse start
22 //this function is used to determine the type of pause
23 //less then 4 hours - S
24 //more then 4 hours but less then 24 - l
25 //more then 24 hours - t
26 function pausetype(d1, d2){
27    if (d1 == "" || d2 == ""){return ""}
28
29    datediff = Datediff(d1,d2);
30    if (datediff<4){
31        return "s";
32    } else if (datediff>4 && datediff<24){
33        return "l";
34    } else {
35        return "t";
36    }
37 }
38
39 //d1 date 1 of current Dialyse, d2 date 2 of current Dialyse, d3 date of previous
      Dialyse, t type of current Dialyse
40 //this function returns the total dialse time depending on the pausetype
41 function Datedifftype(d1, d2, d3){
42    t = pausetype(d3,d1);
43    if (d1 == "" || d2 == "" || d3 == ""|| t == ""){return ""}
44
45    //short pause -- ignore pause, time is from previous til end of current
46     if(t == "s"){
47        return Datediff(d3, d2);
48    }
49    //long pause, time is only current dialyse
50    //or if pause is > 24 then new code
51    if(t == "l"|| t == "t"){
52        return Datediff(d1, d2);
53    }
54
55 return "";
56
57 }
58 //basic setter for apex items
59 function setitem(item, value){
60    apex.item(item).setValue(value);
61    return;
62
```

```
}
```

List. B.26: Javascipt functions for calculating the dialyse time

```

//Dialyse 1
//This sets the first Dialyse line total;
setitem("P30_DIALYSE_1_T",Datediff(apex.item("P30_DIALYSE_1_S").getValue(),apex.item("
    P30_DIALYSE_1_E").getValue()));

let apexitem = "P30_DIALYSE_";

//set all P30_DIALYSE_X_A like:
//This is an iterative loop which sets all following Dialyse lines pausetypes
for (i = 0; i<6; i++){
    setitem(apexitem.concat(i+2,"_A"), pausetype(apex.item(apexitem.concat(i+1,"_E")).
        getValue(),apex.item(apexitem.concat(i+2,"_S")).getValue()));

}

//set all P30_DIALYSE_X_T like:
//This is an iterative loop which sets all following Dialyse lines total Dialyse times
for (i = 0; i<6; i++){
    setitem(apexitem.concat(i+2,"_T"), Datedifftype(apex.item(apexitem.concat(i+2,"_S"
        )).getValue(), apex.item(apexitem.concat(i+2,"_E")).getValue(), apex.item(
        apexitem.concat(i+1,"_E")).getValue()));


}
```

List. B.27: Javascript to calculate the partial dialyse times

```
let apexitem = "P30_DIALYSE_";
//placeholders for up to 3 dialyse codes are initialised here
tot_dial1 = Number(apex.item("P30_DIALYSE_1_T").getValue());
tot_dial2 = 0;
tot_dial3 = 0;
tot_dialswitch = 1;

//set all dialyse times and sum them
for (i = 0; i<7; i++){
    if(apex.item(apexitem.concat(i+2,"_A")).getValue() == "t"){
        tot_dialswitch++;

    }


    switch(tot_dialswitch){
        case 1:
        tot_dial1 = tot_dial1 + Number(apex.item(apexitem.concat(i+2,"_T")).getValue())
            ;
        break;

        case 2:
        tot_dial2 = tot_dial2 + Number(apex.item(apexitem.concat(i+2,"_T")).getValue())
            ;
        break;
```

```
24
25          case 3:
26          tot_dial3 = tot_dial3 + Number(apex.item(apexitem.concat(i+2,"_T")).getValue())
                ;
27          break;
28      }
29  }
30
31    switch(tot_dialswitch){
32          case 1:
33          apex.item("P30_DIALYSE_TOT").setValue(tot_dial1);
34          break;
35
36          case 2:
37          apex.item("P30_DIALYSE_TOT").setValue(String(tot_dial1).concat(" + ", String(
                tot_dial2)));
38          break;
39
40          case 3:
41          apex.item("P30_DIALYSE_TOT").setValue(String(tot_dial1).concat(" + ", String(
                tot_dial2), " + ", String(tot_dial3)));
42          break;
43
44
45      }
46  //this function assigns the corresponding code to the dilyse time
47  function assignCode(time, heparine){
48
49      switch (true){
50
51          case (time == 0 && heparine == "Y"):
52          return "39.95.C0";
53
54          case (time <= 24 && heparine == "Y"):
55          return "39.95.C1";
56
57          case (time <= 72 && heparine == "Y"):
58          return "39.95.C2";
59
60          case (time <= 144 && heparine == "Y"):
61          return "39.95.C3";
62
63          case (time <= 264 && heparine == "Y"):
64          return "39.95.C4";
65
66          case (time <= 432 && heparine == "Y"):
67          return "39.95.C5";
68
69          case (time > 432 && heparine == "Y"):
70          return "39.95.C6";
71
72
73          case (time == 0 && heparine == "N"):
74          return "39.95.D0";
75
76          case (time <= 24 && heparine == "N"):
77          return "39.95.D1";
78
```

```
79          case (time <= 72 && heparine == "N"):
80          return "39.95.D2";
81
82          case (time <= 144 && heparine == "N"):
83          return "39.95.D3";
84
85          case (time <= 264 && heparine == "N"):
86          return "39.95.D4";
87
88          case (time <= 432 && heparine == "N"):
89          return "39.95.D5";
90
91          case (time > 432 && heparine == "N"):
92          return "39.95.D9";
93
94          default:
95          return "39.95.C";
96
97
98      }
99
100
101
102
103 }
104
105
106 tot_code = assignCode(tot_dial1, apex.item("P30_HEPARINE").getValue());
107
108 if(tot_dial2 > 0){
109     tot_code = tot_code.concat(" + ", assignCode(tot_dial2, apex.item("P30_HEPARINE").
            getValue()));
110
111 }
112
113 if(tot_dial3 > 0){
114     tot_code = tot_code.concat(" + ", assignCode(tot_dial3, apex.item("P30_HEPARINE").
            getValue()));
115
116 }
117
118
119 apex.item("P30_DIALYSE_CODE").setValue(tot_code);
```

List. B.28: Javascript to calculate the total dialyse

```
1  //////// Hide ////////
2  //Hide Navigation Bar List
3  $("#t_Header").hide();
4  //Hide Navigation Menu
5  $("#t_Body_nav").hide();
6  //Hide Breadcrumb
7  $("#t_Body_title").hide();
8  //Hide Content Offset
9  $("#t_Body_content_offset").hide();
10 //Hide Report Column Edit Link
11 $(".apex-edit-page").hide();
12 //Hide Report Download Links
```

```
13 $(".t-Report-links").hide();
14 //Hide Buttons
15 //$(".t-Button").hide();
16 $x_Hide("cancel");
17 $x_Hide("calcD");
18 $x_Hide("totalD");
19 $x_Hide("save");
20 //Hide Footer
21 $(".t-Footer").hide();
22
23 //////// Browser Print ////////
24 window.print();
25
26 //////// Show ////////
27 //Show Navigation Bar List
28 $("#t_Header").show();
29 //Show Navigation Menu
30 $("#t_Body_nav").show();
31 //Show Breadcrumb
32 $("#t_Body_title").show();
33 //Show Content Offset
34 $("#t_Body_content_offset").show();
35 //Show Report Column Edit Link
36 $(".apex-edit-page").show();
37 //Show Report Download Links
38 $(".t-Report-links").show();
39 //Show Buttons
40 //$(".t-Button").show();
41 $x_Show("cancel");
42 $x_Show("calcD");
43 $x_Show("totalD");
44 $x_Show("save");
45 //Show Footer
46 $(".t-Footer").show();
```

List. B.29: Print script

C

License of the Documentation

Referenced Web Resources

[1] Oracle apex documentation, app builde. https://docs.oracle.com/en/database/oracle/apex/23.2/htmdb/index.html (accessed November 05, 2023).

[2] Oracle apex documentation, sql workshop. https://docs.oracle.com/en/database/oracle/apex/23.2/aeutl/index.html (accessed November 05, 2023).

[3] Bundesamt für Statistik Gesundheits Sektor. https://www.bfs.admin.ch/bfs/de/home/statistiken/gesundheit.html (accessed November 05, 2023).

[4] Bundesamt für statistik, Medizinischen Codierhandbuch. https://www.bfs.admin.ch/bfs/de/home/statistiken/gesundheit/nomenklaturen/medkk/instrumente-medizinische-kodierung.assetdetail.23446572.html (accessed November 05, 2023).

[5] SWISSDRG Akutsomatik. https://www.swissdrg.org/de/akutsomatik/swissdrg (accessed November 05, 2023).

[6] Free Documentation Licence (GNU FDL). http://www.gnu.org/licenses/fdl.txt (accessed July 30, 2005).

[7] HFR official annual report 2022. https://www.h-fr.ch/sites/default/files/2023-04/rapport-annuel-2022_20230425_de.pdf (accessed November 05, 2023).

[8] HFR hospital activity 2022, 2021. https://www.h-fr.ch/de/jahresbericht/2022/unsere-spitalaktivitaet (accessed November 05, 2023).

[9] Nexus/kis — clinic information systems. https://www.nexus-ag.de/klinik/klinikinformationssystem (accessed November 05, 2023).

[10] Opale bluepearl — opale solutions sa, hospital administrative tool. opale-solutions.ch (accessed November 05, 2023).

[11] Oracle data bases. https://www.oracle.com/database/ (accessed November 05, 2023).