Specifications

A Data Harmonization Portal

Athens University of Economics and Bussiness
Charoula Velissarakou
11/4/2017

1. Scope

1.1 Overview

The Data Harmonization Portal is a tool designed for laboratory use, so that its registered users will be able to manage medical variables in a way that reliability is guaranteed.

This document will cover the registration and log in process and the detailed ways that users follow to upload their data, edit it, and download it. It will also describe the second purpose of the tool, the data harmonization, which will be achieved with the help of all the crowd sourcing functionalities that the tool will use to combine all these data that come from all the different registered users.

1.2 Objectives

The objectives of that web platform application are as following:

- To support user login and registration to application
- To effectively upload meta-data and download it.
- To share meta-data with other members of the community.
- To easily manage and combine data from heterogeneous sources into an integrated information product.
- To enhance the quality of the data and reduce errors.
- To eliminate redundancies and duplications of data.
- To easily transform data into a structure that can be used from other tools.

1.3 Installation Guide

The following steps should be followed for the tool to be tested:

a) Download and install XAMPP.
b) In the XAMPP control panel click 'Start' under the 'Actions' for the Apache and mySQL module.
c) Move the tool's files in the htdocs folder, that exists in the xampp folder.

d) Open your web browser and type in: http://localhost/phpmyadmin .

e) Load the 'dh_users' database that its schema will be described later.

f) Now type in: http://localhost and the DHPortal tool is ready to be tested.

2. Database schema

The 'dh_users' database contains information about every user's account for example their personal data, such as e-mail or password, and also friendship requests from a user to another. The schema is shown below:

The 'rights' column of 'users' table is an integer that identifies whether a user is an admin or not. Thus, the 'rights' can be either 0 or 1 and can be further used during the log in session which is done with the help of the 'initialization_f.js' and the 'users_handler.php' files.

3. Flowchart

3.1 Log in Flowchart

When the user opens the Data Harmonization Portal for the first time, the log in Page will be shown.

The users should be registered so as to use the tool and their data should exist in the "users" table of the "dh_users" database which is created for that purpose.

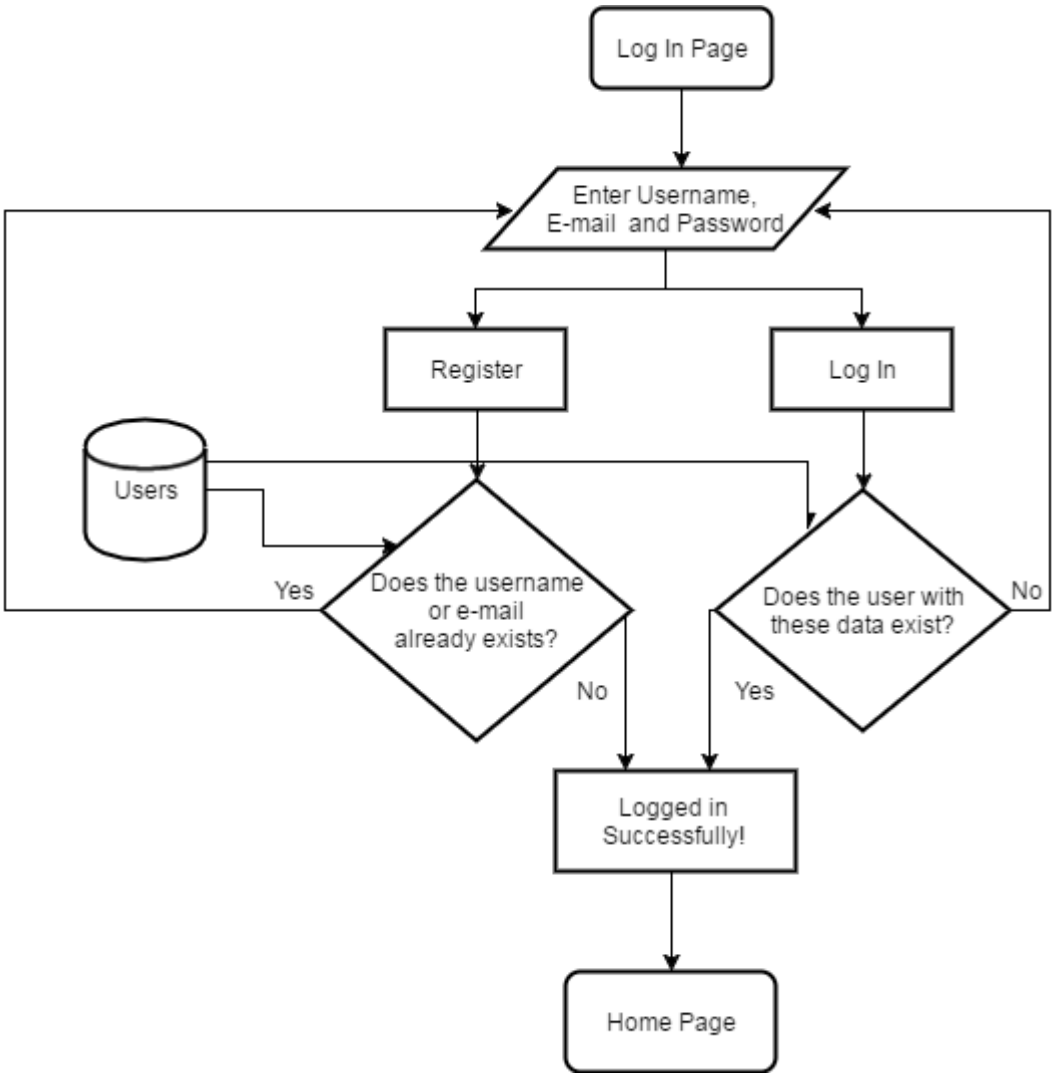
After typing a username, e-mail and a password there will be two use cases.

- Register to Application:
  If the imported username, mail and password do not exist in the database, the user will be able to log in automatically with these data, once they are stored into the database. This will be done, when the user presses the "Register" button. In the tool's folder that exists in htdocs, a new folder will

be created named by the user's username. A Public and a Private folder will be created inside it too, in which users will later save their data.

- Log In to Application:
  If the imported e-mail and password exist in the database the user will be logged in, otherwise they will have to correct the username, e-mail or the password again.

  Note: It should be mentioned that whether or not the user has rights as an admin or as a simple user are not checked during the registration or the log in session which are initialized in the index.php.
  The Log in flowchart is the following:



3.2 Home Page Flowchart

When the user logs in, the Home Page will be shown. Then the user can manage their data as they wish. The home.php uses the most of the tool's files, such as:

- **main.css**: the main stylesheet of the tool
- **style.css:** the stylesheet of the visualized circle packing
- **friend_f.js:** contains functions related to friend requests, friend local files etc.
- **variable_f.js:** contains functions related to the variable table (saveV,deleteV,editV, fillformV,findVar,select_tab etc)
- **group_f.js:** contains functions related to the circle packing and the creation of the JSON file (saveG,deleteG,editG, fillformG,draw,zoomTo,search, show details, add_option, delte_option etc)
- **menu.js:** contains functions related to the navigation menu events.

On the left, there is a navigation menu that provides the following functionalities:

1. **Upload meta-data:**
   The user will have to choose a file, that has a .json extension once this button is clicked. This JSON file contains all the data of grouped medical variables, that are universal and commonly accepted by the community that uses this tool. A file sample should be like this:

```
[
 {
  "code": "Group 1",
  "label": "Group 1",
  "parent": "None",
  "description": "Group 1",
  "children": [
   {
    "code": "Group 3",
    "label": "Group 3",
    "parent": "Group 1",
    "description": "Group 3",
    "children": [
    ]
   }
  ]
 },
 {
  "code": "Group 2",
  "label": "Group 2",
  "parent": "None",
  "description": "Group 2",
  "children": [
   {
    "code": "var 6",
    "label": "var 6",
    "type": "Binominal",
    "group": "Group 2",
    "description": "var 6",
    "methodology": "var 6"
   }
  ]
 }
]
```


Each group object has a unique code, a label, a parent, a description and an array with its children. The array can include variables and also other groups with their own attributes. In this way, a hierarchy is formed.
Note: The hierarchy and the JSON objects are created in *group_f.js* and *variable_f.js* files. The latter includes the saveV() function which is mainly responsible of how the variable object is formed and the former has the saveG() one among many useful functions, which creates the group object.

Every time a group or a variable is created, edited or deleted the search() function which is called in all the related functions of the *group_f.js* and the *variable_f.js* will search in the localStorage.JSONobj - which is the dynamically created JSON of the user- and return the node that has to be edited.

With the use of the D3 javascript library, which is used in the draw() function in *group_f.js* file, this hierarchy is plotted on the first panel on the left so that the user can see this file's visualized result.

2. **Download meta-data:**
When the user is finished, they can locally download the new meta-data.

3. **Upload Variables:**
The user will have to choose a file, that has a .csv extension once this button is clicked. A valid CSV file contains all the data of ungrouped medical variables. Each variable has a unique code, a label, type, description and methodology separated by commas (,).
A valid CSV could look like that:

```
Code,Label,Type,Description,Methodology
var A,var 20,Real,var 20,var 20
var V,var 19,Polynominal,var asas,var 19
var R,var 18,Polynominal,var 18,var 18
var J,var 17,Real,var 17,var 17
```

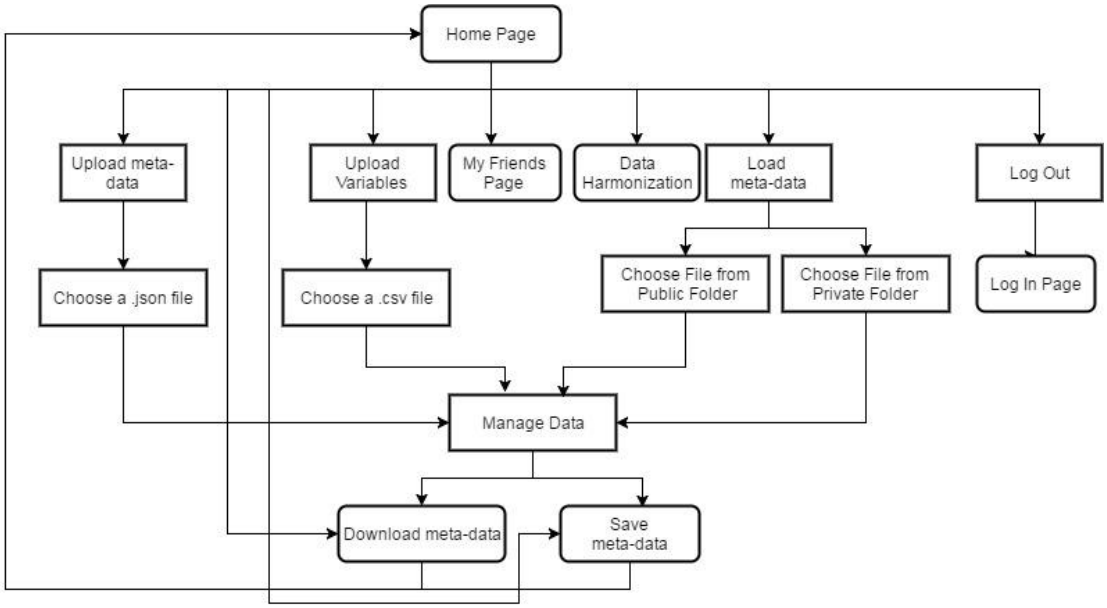
When the file is uploaded the user can manage these variables which will be shown at the variable table on the first panel on the right, and add them in a group that he/she will create or upload.

Note: If the csv file is invalid and contains more columns, an alert box will inform the user to upload another csv file.

4. **Load meta-data:**
The user will have to choose a file to load from either the public or the private folder that exist in server. if it is a json file, the hierarchy that has been created so far will be replaced with the new one and the new variables will be appended to the old ones. Then the user can choose which variables they want to keep or delete or even edit.

5. **Save meta-data:**
The user can save their data to the server. If they choose the Public folder, their friends will be able to see and use their data for later use. If they choose the Private Folder the files will be accessible only by them. The data will be saved in a json format.

The Home Page flowchart is the one below.



The "Manage Data" action, "My Friends" tab and "Data Harmonization" tab will be analyzed by a flowchart too.

3.3 Manage Data Flowchart

There are four ways to manage data.

- Add a new Group:
  Using the "Selected Group" form, the user will be able to fill the data of the new group, he/she needs to add. Code and Label are required fields. If the code is already taken by another group that is saved to the meta-data (JSON), an alert message will appear and the user will have to change it so that it can be added to the pack. Then the new group will be saved to the meta-data (The search() function in the *group.js* will return the node in which the new group should be added) and will be plotted as a circle on the first panel on the left using the draw() function. Then, a click event will be added to that circle and whenever a user clicks that group, its details will be showed up at the "Selected Group" form.

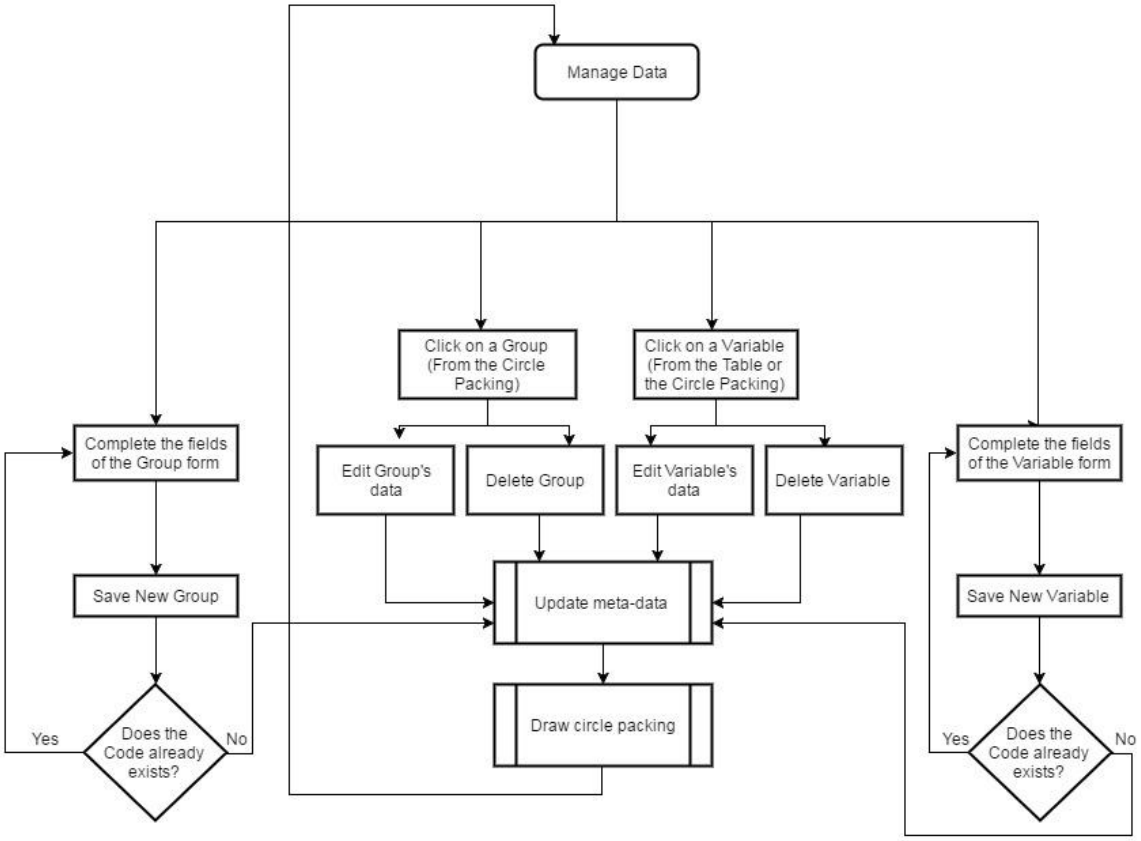
- Click on a group circle:
  When the appropriate form will be filled using the fillformG(), the user can edit or delete the selected group. If that happens, the JSON meta-data will be

updated properly and the circle packing will be drawn again.

- Add a new Variable:
  Using the "Selected Variable" form, the user will be able to fill the data of the new variable, he/she needs to add. Code and Label are required fields. If the code is already taken by another variable that exist at the table, an alert message will appear and the user will have to change it so that it can be added to the table. Then the new variable will be saved to the meta-data, its details will be added to the variable table and it will be plotted as a circle on the first panel on the left. Then, a click event will be added to that circle and whenever a user clicks that variable, its details will be showed up at the "Selected Variable" form and the row that has its details, will get highlighted.

  The new variable will be new and grouped, thus it can also be found in the 'Grouped' Tab.
- Click on a variable circle:
  When the "Selected Variable" form will be filled, the appropriate row will be highlighted and the user will be able to edit or delete the selected variable. If that happens, the JSON meta-data will be updated properly and the circle packing will be drawn again because the hierarchy might change (If the parent is different after editing or if the variable is deleted from a group). If the user choose to edit the variable, the highlighted row will be deleted from the table and a new one with the new data will be added. If the user choose to delete the row then it will be permanently deleted from the table.

The "Manage Data" flowchart is the following:



The D3 Library

D3.js is a JavaScript library for manipulating documents based on data using SVG, CSS and HTML5 standards. It produces interactive data visualizations in web browsers binding arbitrary data to DOM. For instance, it can easily create a simple HTML table or an SVG circle packing which smoothly zoom in and out once these circles are clicked.
D3 provides numerous methods for mutating nodes: setting attributes or styles, registering event listeners, adding, removing or sorting nodes and changing HTML or text content.
Nodes can easily be manipulated by using selectors which are defined by the W3C Selectors API.
For example, the code to remove all the circles from an svg element is the following:

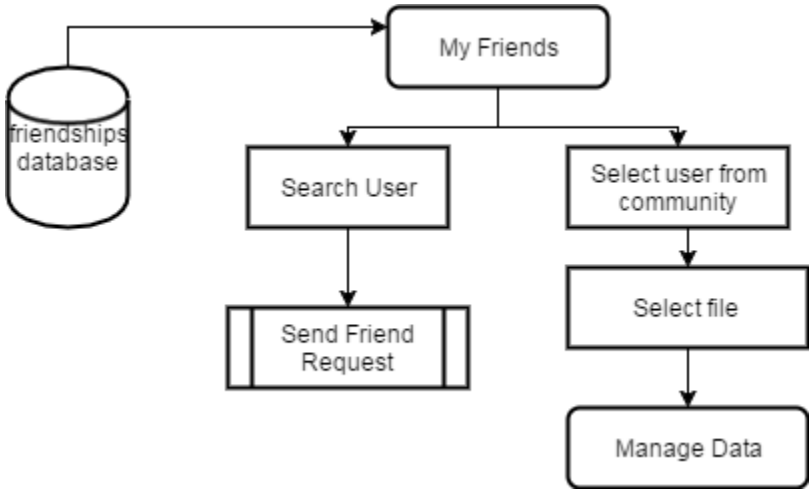
```
d3.select("svg").selectAll("circle").remove();
```

which selects all the circle elements in the first svg element of the DOM and then removes them.

3.4 My Friends Tab Flowchart

"My friends" Tab, contains the users that one wants to share their data with. Once this page will be loaded, a user will be able to add new friends or load files from one of their friend's public folder. Users can see admins' folders without the need to send them a friend request. That means, that whenever a user registers he or she will be able to see admin's files from the very start in the Community.

Note: Friendships between users are saved in "friendships" table from the "dh_users" database.



When a user wants to add another member, they will need to enter their username or email at the search box. If the user exists, a friend request will be sent and a friendship pair from this user to the other one will be saved at the friendships table in the database. If the other user has a friendship pair with the logged in one, then that member will be immediately added in the "My Community" Field.

Then the user can load files from their friend's public folder to their home page or the harmonization page. Any changes that will be done to these data, will not change the content of their friend's file because this content is just copied to the localStorage variable in the browser.

When the user selects the file then he or she will be asked to chose whether these data are to be uploaded on the Home Page or the Harmonization one. If it is a JSON it will immediately be uploaded at the Home Page. If it is a csv then if the file is to be uploaded at the Home Page those variables will be shown at the variable table with a different color for a more efficient use. If the csv is a harmonized one, an alert error message will be shown.
If the csv is to be uploaded in the harmonization Page then depending on the csv type (if it a harmonized one or not) the variable data will be loaded accordingly.

If the user wants to remove a friend from their community, they can press the 'Unfriend' button which will delete the pair: from_user, to_user and the other way around from the friendships table in dh_users database.

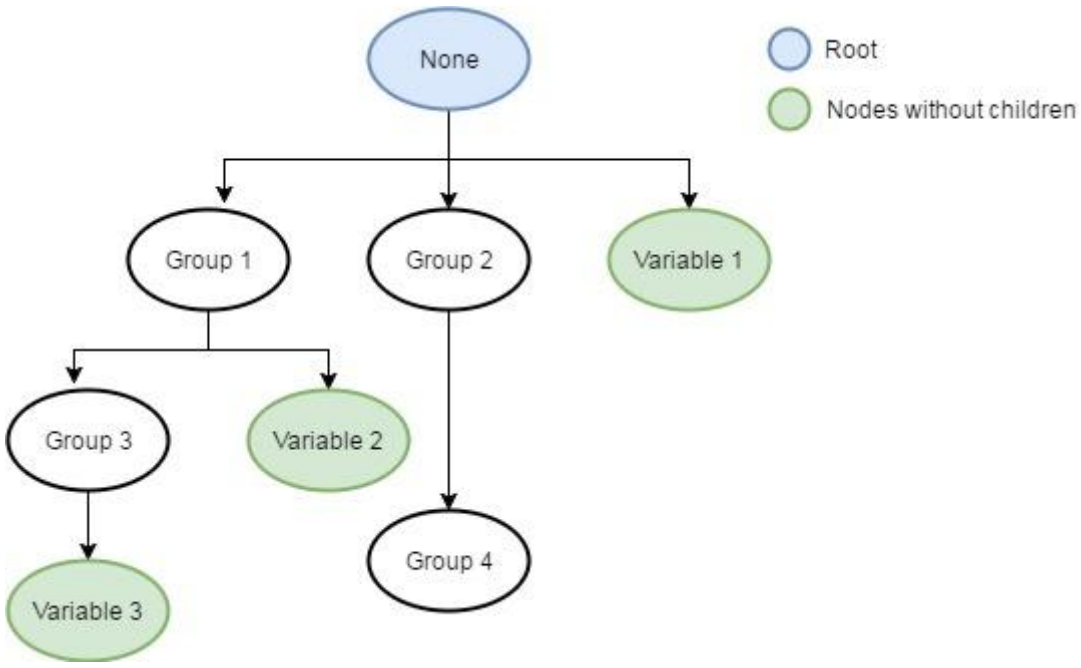
4. Update functionalities

The Data Harmonization Portal is basically based on the meta-data creation that happens during a user's session. These meta-data need to get updated after every change made on them. Consequently, the hierarchy is constantly changing, so the plotting should also change after the meta-data update.
Based on the "Manage Data" flowchart the functionalities that happen each time the data need to update are:

- Update meta-data
- Draw Circle Packing

4.1 Update meta-data

The data are saved in JSON format in the form of a hierarchy as shown before. A visualized hierarchy example is:



"None" is the root of the hierarchy, it is the supergroup that contains every node that is created. Group nodes can have children, unlike variable nodes.

Whenever an update (edit, delete or add) takes place, there will be a need to find the parent of the specified node and then the child that the action will be done to. As far as the edit action is concerned, the delete action will take place first and then the add action should follow. This happens because the edited node might have a different parent, so it will have to be deleted from the old one first, so that it can be added to the new one later.

The updated nested JSON object will be used for the circle packing update.

Note: So as to never lose the hierarchy or the variables that have been created so far while browsing through the tabs of the tool, two variables are saved to the localStorage of the browser. One variable is used for storing nested JSON object and the other one for storing the variables. Whenever a user logs out those variables will be cleared.

4.2 Update circle packing

This is done with the help of the D3 javascript library, using the JSON hierarchy to pack the groups together. Zoom events on circle click are initialized and the labels are added in the middle of each circle.

5. Combining Data - Data Harmonization

With the use of the Data Harmonization tab the user will be able to match new variables that are saved in a .csv format with the old ones that are already uploaded. Firstly, the user should upload their meta-data from friends, locally, or from the server. Once that is done, the localStorage will be initialized and the variables will be used to form the plot on the left side-where the user can also see the info of each element of the graph- and the table on the right side, once the user uploads a csv file with the variables that he/she needs to match. He can either upload a harmonized csv so that all these data will be initialized at the table.

If the csv contains Methodology Cell, it means that a variable csv is loaded. In the other case it will be a harmonized CSV so it will be split on the comma delimeter (if a comma is not found between “” then the delimeter is not right) and that will create the harmonization table with all the information that this csv contains. The function

that make it possible is the readHCSV() at the harm_f.js file.

Note: The functions that make this matching possible exist in the functions.js. These functions that are used to transform one variable to another are initialized once the new csv is uploaded, using the function array that exists in functions.js in js folder.

The csv can be saved at the server in the form:
old_variable, new_variable format, missing timestamp, subject ref., pseudonymization, type, unit, range, function, transformation

where transformation is the altered selected function or functions, in a form that can be used from another tool.

The columns can be easily edited from these files:

- harmonization.php: edit the <th> tags of the table.
- harm_f.js : edit column's elements,edit attributes etc. in initVar()
- harm_f.js : edit the new columns to the csv and the table data in it. CSVtobeTransformed()

The supported functions are the following:

abs(x)	abs(cast (x as float))
aggregation('function','attribute params')	aggregation('function','attribute params')
append(str1, str2)	str1 || str2
acos(x)	acos (cast (x as float))
asin(x)	asin (cast (x as float))
atan(x)	atan (cast (x as float))
atan2(y,x)	atan2(cast (y as float),cast (x as float))
ceil(x)	ceil(cast (x as float))
contains(str, subStr)	str like '% str_replace("\'","",subStr)%'
containCount(str, subStr)	(length(str)- length(regexp_replace(str,subStr),'','g'))/ length(subStr)
cos(x)	cos(cast (x as float))
cosh(x)	(exp(cast(x as float))+exp(-cast(x as float)))/2
currentYear()	extract(year from current_date)
date()	current_date
datetime()	date_trunc('second',localtimestamp)
exp(x)	exp(cast(x as float))
floor(x)	floor(cast(x as float))
if(cond, trueval, falseval)	case when cond then trueval else falseval end
indexof(str, subStr)	position(subStr in str)

isNotNull(arg)	arg is not null
isNull(arg)	arg is null
isNumeric(str)	str ~ \'^[-]?[0-9]*\\.?[0-9]+([eE][-+]?[0-9]+)?$\'
len(str)	length(str)
log(x)	log(cast(x as float))
ln(x)	ln(cast(x as float))
mod(x,y)	mod(round(cast(x as numeric)),round(cast(y as numeric)))
null()	null
pow(x,y)	power(cast(x as float), cast(y as float))
replace(str, text1, text2)	replace(str,text1,text2)
round(x,[y])	round(cast(x as numeric),cast(cast(y as numeric) as integer)) ή round(cast(x as float))
sin(x)	sin(cast(x as float))
sinh(x)	(exp(cast(x as float))-exp(-cast(x as float)))/2
sqrt(x)	sqrt (cast(x as float))
substring(str, start, [end])	substring(str from start+1 for end-start) ή substring(str from start+1)
tan(x)	tan(cast(x as float))
tanh(x)	(exp(cast(x as float))-exp(-cast(x as float)))/ (exp(cast(x as float))+exp(-cast(x as float)))
todate(str, pattern)	to_date(str,pattern)
todouble(x)	cast(x as float)
toint(x)	round(cast(x as numeric))
tolower(str)	lower(str)
tostring(data)	cast(data as text)
toupper(str)	upper(str)

These functions, once the user downloads the CSV, will be transformed with the help of *func_transform.php* and will be ready for another tool to use.

The splitExpression makes it possible to recursively create the transformed function that the user inserted. This is done by first using the split_it function which splits the parent function the commas and saves its arguments into a parameter array. If the function contains other functions then the split_it will be called recursively by the splitExpression for each one of the nested functions.

The replaceFunctionText which is called in the splitExpression, takes as input the functionName and an array with the parameters of the according function each time, starting the transformation from the inside. For example for the string

$function = “if(var1>10, null(), if(var2<2,'33','100’))”

we will have two kinds of data: (1) $f = ‘if’ that is the function name and an array with its parameters $parameters={‘var1>10’,’null()’,’if(var2<2,'33','100’)’}.

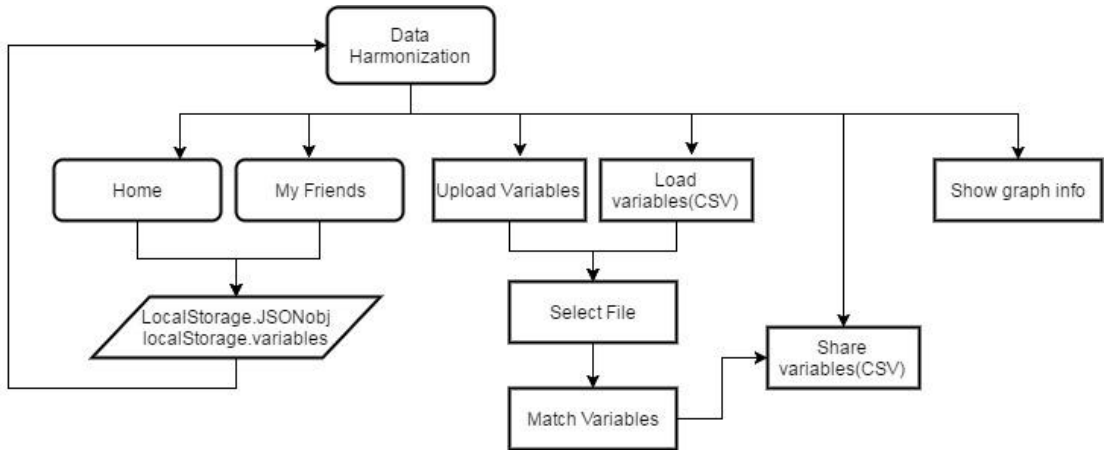
If one of those parameters contain a function then this function will be split and transformed as well. So for the ’if(var2<2,'33','100’) ’ we will have a new (2) $f=’if’ and the array will be the following: $parameters={‘var2<2’,’33’,’100’}.

The replaceFunctionText will transform the (2) above (replacing variables with the placeholder-with the help of replaceVar function) and it will return it to the (1) $parameters. So the new array of the first if will become:

```
$parameters = {
      ‘var1>10’,
      ’null’, /*The null() has been transformed to ‘null’*/
     ’case when _mipmap_function_<2 then ‘33’ else ‘100’ end’
     /*Where the var2 has been replaced with the placeholder, because it is a
       variable. */
}
```

In the end the first if will be transformed accordingly too.

The Data Harmonization flowchart is show below.



6. Glossary

Meta-data	Data that provides information about other data.
JSON	**J**ava**S**cript **O**bject **N**otation JSON is a syntax for storing and exchanging data.
CSV	**C**omma **S**eparated **V**alues CSV stores data in a table structure format.
LocalStorage	With local storage, web applications can store data locally within the user's browser
DOM	**D**ocument **O**bject **M**odel