



Metrics Monitoring Tool Technical Documentation

Layout, Project Data, Charts and Weekly Report Editor, Redmine Data Reading

07.02.2015

Table of Contents

1. Introduction	4
2. Main Layout	4
2.1 Project List (project_list.php)	5
2.2 Project Details (project_details.php, subpage of Project List)	6
2.3 Compare Metrics (project_comparison.php)	6
2.4 Weekly Report (readweekly.php)	7
2.5 Redmine (redmine.php)	7
2.6 Facebook (/facebook_forum/initialization.php)	8
3. Retrieving project data	9
3.1 Database queries (database_out.php)	9
4. Charts	11
4.1 Making data for the charts (metrics-makedata-1.0.0.js.)	12
5. Weekly report editor	13
5.1 Parser	13
5.2 Adding and hiding fields	14
5.3 Sending the data	14
6. Redmine Interaction	15
6.1 Introduction	15
6.2 Documentation about Redmine API	15
6.2.1 Redmine API	15
6.2.2 A simple Object Oriented PHP Redmine API client	15
6.2.3 Creating Client	15
6.2.4 Getting array of data	15
6.3 Example about getting this data	16
6.4 File: getnumbers.php	17
6.5 File: redmine_working_hours.php	17
6.6 File: todatabase.php	17
6.7 Redmine Links	17
7. Facebook Interaction	18
7.1 Introduction	18
7.2 File listing	18
7.2.1 login.php	18
7.2.2 initialization.php	18

7.2.3 results.php	19
7.2.4 addgroup.php	19
7.2.5 refresh.php	20
7.2.6 update.php	20
7.3 Global issues	21
7.3.1 Group permissions.....	21
7.3.2 Error handling.....	21
7.3.3 Input validation.....	21
7.4 Miscellaneous.....	22
7.5 Further development suggestions.....	22

1. Introduction

This documentation describes the metrics monitoring tool's layout, project data retrieval, weekly report editor and chart functions. All of the PHP files are located in the project's */main/* folder unless stated otherwise. All of the script files are located in */main/scripts/*.

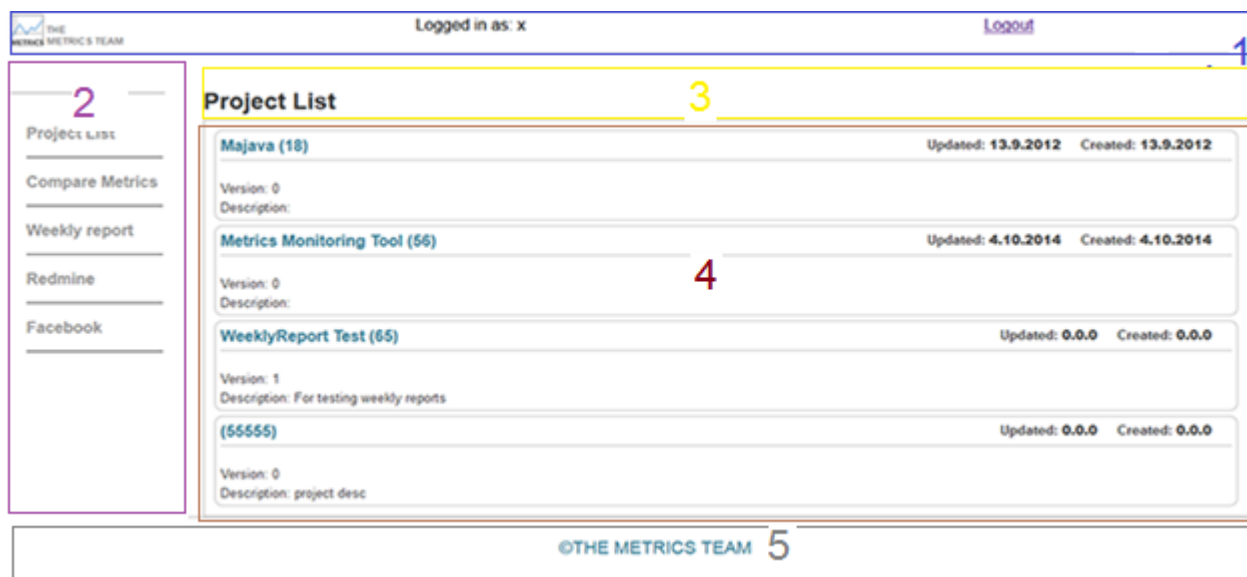
External scripts used in the project:

<http://jquery.com/>

<http://www.highcharts.com/>

2. Main Layout

The layout is designed to be light and easy to use. Basic elements of the website are placed in logical positions for optimal user experience. The layout uses a CSS style file (*main/css/style.css*). Reoccurring elements of the page are created using JavaScript functions to assure easy updates to the page layout.



The layout of the website consists of these main elements:

1. Top
2. Navigation
3. Header
4. Content area
5. Footer

These five main elements are present on every subpage, and are created by using JavaScript functions. Every page uses the script file **metrics-elements-1.0.0.js**, which holds all the functions necessary for creating the elements.

Let's take a closer look at these functions:

```
createTop("<?php Print($user_check); ?>");
```

The **createTop** function creates the top panel of the website. The function takes in one value and uses it as username. Username is defined in the PHP session when logging in. The **createTop** function is also responsible for creating all the other elements in the top panel, such as the logo image and the "sign out" link.

```
createNavig();
```

The **createNavig** function is responsible for creating the website navigation. The creation of navigation links is fully dynamic and adding or deleting links is easy.

```
createHeader("Project List", 1);
```

createHeader creates the header. It takes in two values. The first value is the header text itself and the second value is used for determining if the header has some unique elements. These unique elements can be sorting options or like the fields seen in weekly report page's header.

```
createFooter();
```

Creates the footer section of the website.

Each page has a unique content area, therefore there is no single function populating it. Most of the functions relating the content area are located in **metrics-makedata-1.0.0.js**.

2.1 Project List (project_list.php)

Acts as the main page of the website after the user logs in. This page displays a list of projects and some useful information related to them. The list of projects comes from the metrics database, which is queried to load only the necessary information. Ajax is used for communicating with the php.

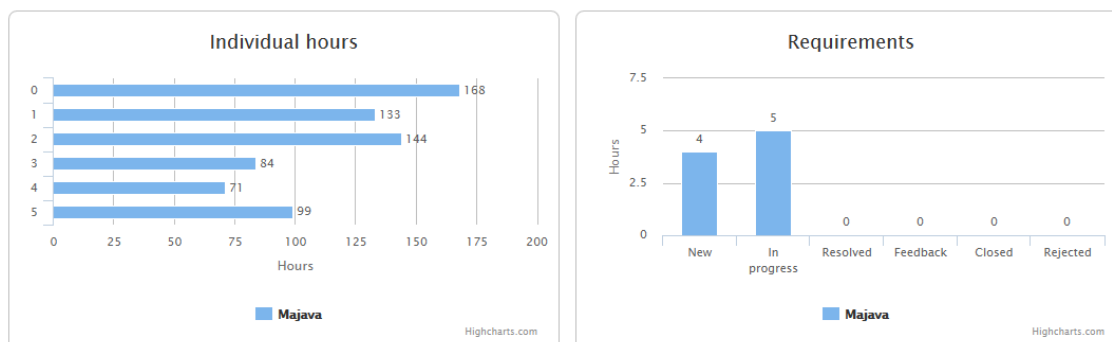
Initiated using this function located in **metrics-makedata-1.0.0.js**

```
getProjectList(0);
```

By clicking a project's name, the user can see information related to that specific project. Project details are shown on the project details page.

2.2 Project Details (project_details.php, subpage of Project List)

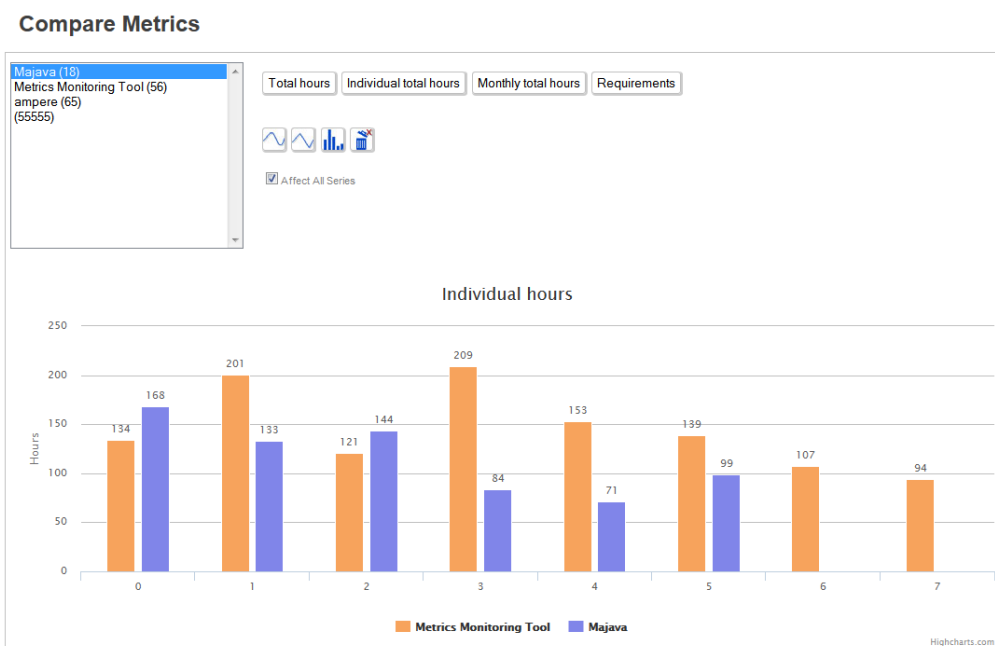
Gets project id from the clicked project and queries the metrics database for project data. Some project details are displayed as charts. The charts are first created as blank dummy charts, and populated after getting actual data from the database.



This page also shows information about project test cases, requirements, code revisions and commits to version control.

2.3 Compare Metrics (project_comparison.php)

The comparison page allows the user to compare different project metrics. Just like with the project list, a list get populated with basic project information. The user can then choose a project from the list and use the given buttons to show wanted metrics. Chart type can be changed between column, line and spline. Chart deletion is also made possible and by unticking the “Affect All Series” the user can affect only the latest series, or a series that has been selected by clicking.



The functions behind all of the buttons are located in **metrics-makedata-1.0.0.js**. When a chart data type button is pressed the code checks for a selected project and fetches the data from the database.

The function takes in an integer that represents the different nature of the button's intended purpose.

`GetSelectedOptions(3)`

The chart type buttons use a function that updates the current chart with the new selected type.

2.4 Weekly Report (readweekly.php)


The weekly report editor is built to parse information from a text file and show it in a more comprehensive and easily editable format. Parsing from a text file is optional, since creating a completely new weekly report is also possible.

Creating a new weekly report is fairly simple, the user can add fields and textboxes that fit their predefined roles. The functions used on this site are located in **read_weeklyreport.js**

After finishing the weekly report, the user can upload it to the database.

2.5 Redmine (redmine.php)

This page is used for manually updating the metrics database with data from redmine. A progress bar is displayed in the top right corner of the site to indicate the process.

Processing individual working hours...

 494 row(s) processed.

Project List

Compare Metrics

Weekly report

Redmine

Facebook

Redmine

Individual_work :

already imported in local database : 934
 numbers of data in redmine database : 933

Requirements :

already imported in local database : 34
 numbers of data in redmine database : 30

Projects :

already imported in local database : 4
 numbers of data in redmine database : 2

The redmine page also shows the current amount of data in both redmine and metrics database.

2.6 Facebook (/facebook_forum/initialization.php)

The Facebook page is used for getting information from Facebook groups. The user can limit fetching dates and choose a certain person in a group. Facebook group id is needed for fetching the data.

Facebook Query

Date from:

2014-01-01

Date until:

2015-02-04

Group ID:

652060828245934

Everyone ▼

Update member list in DB

Add group to list

Recent posts:

5



Lähetä kysely

3. Retrieving project data

All interaction with the database is handled with JQuery's Ajax functions. There are two Ajax functions: the other one is for getting a list of projects and the other one is for more customisable queries. Both of the functions use Post method for passing the variables to **database_out.php**. These two Ajax functions are located in **metrics-makedata-1.0.0.js** and they are called:

getProjectList(options)

and

getProjectData(id,operation,querytype,value,containername)

getProjectList takes in an integer variable that defines what to do with the information after it is successfully retrieved. If the variable is set to 1, it means that the data is for the comparison page and if the variable is 0, the data is for the main page's project list.

getProjectData takes in project id, operation to be executed in php, query type for handling database query, value; and container name for chart creation after successful retrieval of data.

3.1 Database queries (database_out.php)

The database_out.php file is used for fetching data from the metrics database. The php file includes functions for selecting from *weekly_report*, *weekly_report_requirement*, *project*, *participation*, *member*, *requirement* and *individual_work*. It gets project_id, querytype and operation variables by Post method. After checking the database connection and deeming it successful, the code checks the wanted query type.

echoResults(\$operation, \$project_id, \$con, \$scope, \$where, \$equal)

If the variable querytype is 0, the code runs a function called **echoResults**, which is used for querying only single tables. The **echoResults** function takes in an integer variable "operation", that defines which table will get echoed. The other values are: project id, database connection, scope; where and equal (used for select query).

getDataForCharts(\$project_id, \$con)

Else if the querytype is 1, **getDataForCharts** function is called. the **getDataForCharts** function is used for getting information from multiple tables and packing them all into an object.

```

getWeeklyReports($project_id, $con, $scope, $where, $equal)
getWeeklyRequirements($project_id, $con, $scope, $where, $equal)
getProjects($project_id, $con, $scope, $where, $equal)
getParticipation($project_id, $con, $scope, $where, $equal)
getRequirements($project_id, $con, $scope, $where, $equal)
getIndividual($project_id, $con, $scope, $where, $equal)

```

The code consists of functions that all have their own respective table to query. The functions used for table queries all take in the same values. All of the functions use normal MySQL selecting queries, and all the variables they take in are related to controlling those queries. All the information the code gets from a table is put into an array. The arrays containing the data are then encoded into a Json object and sent back to the Ajax function.

```

Sort by key
+ weekly_report

- project
  - 0
    project_id
    project_name
    created_on
    updated_on
    status
    version
    description
  participation
+ individual

+ requirement

weekly_requirement

```

The project objects are constructed as shown above. Each table gets its own key and sub keys for all information in that table regarding selected project id.

makeLineData(value, containername)

After successfully getting all project data, **makeLineData** function is called. It takes in two values. First value is an integer and it determines what kind of chart needs to be drawn and the second value is the name of the div container housing the chart.

4. Charts

The charts are made possible by an external JavaScript library called **HighCharts**.

The HighCharts library has many built-in functions for manipulating the charts. These built-in functions include updating the chart's data and type without completely re-drawing the whole chart. However, custom functions for controlling the built-in functions had to be created for easier presentation and modification of the charts. All of the custom code regarding the charts can be found in **metrics-makechart-1.0.0.js**.

CreateChart(type, x_label, y_label, container_id, chart_title)

The **CreateChart** function is used for creating a chart. It takes in five values. First value dictates the type of the chart. Type is given as string ("bar", "column", "line", "spline", ...). 2nd and 3rd values are used for labelling the x and y axis. container_id is the string id of the html div container to house the chart object. chart_title is the string title of chart. The chart created with this function is merely a dummy chart waiting for real data and values.

addLine(DataArray, xCategories, containername, chart_title, types, seriesname, chartno)

The **addLine** function is used for adding a new series to a chart. The function takes in DataArray, which contains the data that will be shown on the chart. xCategories is an array used for naming the x axis categories. containername is the string name of the container where the chart is. chart_title string defines the title of the chart. types is an integer that is used for detecting if the chart needs to be redrawn. seriesname string defines individual series names. chartno integer is used for identifying the right chart if there are multiple charts on one page.

removeSeries()

removeSeries function is used for removing all series from a chart. Optionally, if "affect all" is not checked, only the selected series will be deleted. By default the latest series is always selected.

change(newstyle)

change function is used for changing the chart's visual style. It takes in a string value, which is directly used as the new type. The new style can be any of the styles described in highcharts documentation. The styles used in this project are bar, column, line and spline.

There is also a function for creating a test chart with random data. This function is called:

ChartWithRandomData(type)

It takes in a type string, which dictates the type of the chart just like in the CreateChart function. Data is randomized using a function called:

RandomizeData()

4.1 Making data for the charts (metrics-makedata-1.0.0.js.)

Data from the database is not initially ready to be displayed by the charts. The HighCharts charts can read objects, but the objects must have predefined key names in order for them to work properly. The data also needs to be calculated and right x-categories need to be chosen. All this is made in **metrics-makedata-1.0.0.js**.

There are 8 main functions that all have their own role in making the data into the right, displayable format. These functions are as follows:

```
getMonthlyHours()
getIndiHours()
getAllHours()
getRequirements()

projectRequirements()
projectTestcases()
projectCoderevisions()
commitsToVersionCtrl()
```

They all have their own role and calculations, but they all use the same project object that has been retrieved from the database. These functions are called in the **makeLineData(value, containername)** function, in the makeLineData function there is first a check to see what kind of data needs to be drawn. For example if there is a chart for monthly hours, then only the **getMonthlyHours()** function will be run. The object format for all the chart data is: {y: int, name: string}. The y key is the value of the line point and the name is the name of that point.

Other functions in the **metrics-makedata-1.0.0.js** file are:

```
parseDate(date)
parseDateTime(scope, datetime)
```

These functions are used for parsing and making sense of the date values in the database. The date values in the database are in a rather unusable format. **parseDate(date)** takes in the raw date format and uses the **parseDateTime** function to make it into dd.mm.yyyy format.

```
clearArrays()
```

Clears arrays.

```
createOptions(name,id)
getSelectedOptions(value)
```

createOptions(name,id) creates the list of projects in the comparison page. Name and id come from the queried project list object.

getSelectedOptions(value) checks which project in the list is selected and calls the **getData** function, which houses the Ajax function used to retrieve project information.

5. Weekly report editor

The weekly report editor uses `readweekly.php` file and its functions are located in `read_weeklyreport.js`. One of the main functionalities of this editor is the ability to read and parse text files. After parsing a text file the page shows all the parsed information in a more readable and editable form. The editor also allows creation of new text fields, which means that parsing information from a text file is optional.

5.1 Parser

readSingleFile(evt)

Is the parser function. The parser searches the text file for keywords indicated with #. When a keyword is found, all data after it are considered nodes belonging to that keyword. Many of the keyword categories have different kinds of functions related to them, like for example, some require textboxes and some need two text fields instead of one.

getData(i, regex, idprefix, arrayOfLines, detectedkey)

getData is called every time the parser hits a keyword. It is responsible for looping through sub lines of the detected key word. It takes in 5 values:

`i` (int) nth round of the parser loop.

`regex` (string) determines which regular expression to use. mainly uses email regex.

`idprefix` (string) prefix for unique id.

`arrayOfLines` (array) all the text file lines.

`detectedkey` (string) detected keyword.

It then creates the needed fields into their own respective container divs using the **CreateInput** function.

CreateInput(text,idprefix,order,counter,detectedkey,classname)

CreateInput function is responsible for creating input fields on to the page. It takes in 6 values:

`text` (string) used as input value attribute.

`idprefix` (string) used as a prefix for unique element ids .

`order` (int) used to detect if a line break is needed.

`detectedkey` (string) detected key word.

`classname` (string) used for giving non-unique class names to fields.

document.createElement("input");

document.createElement("textarea");

Fields are then created using JavaScript's own **createElement** function.

5.2 Adding and hiding fields

addToSection(detectedkey)

This function is used when adding new input fields into the report. It takes in the detected key which is used to identify what kind of field the user wants to add. The **addToSection** function then calls for **CreateInput** with the right values.

hideSection(targetid, headerid, btnid)

Hides the wanted section of weekly report. Does not really serve any purpose other than visual. Takes in values required for identifying the right target div, header and button.

5.3 Sending the data

clicked()

When the user clicks the submit button **clicked()** is called. This function reads all the text fields and constructs an object that will be sent to the server using Ajax.

JSON

```
+ client
+ completed_tasks
+ managers
- milestone
  + 0
  - 1
    name
+ other_test
+ otherinfo
+ problems
+ requirements
+ revisions
+ tasks_next
+ unit_test
+ workinghours
```

Above is an example object created from the data in the weekly report editor.

6. Redmine Interaction

6.1 Introduction

These files are used to get the necessary information or data from UTA Redmine website [1] under a defined account (currently reporter account) and save it to Metrics database. Location of these files is in Metrics-Project/reading from Redmine and all of these files use a **simple Object Oriented PHP Redmine API client** [2] so the files in library and vendor folders are basically this compiled API by composer.

6.2 Documentation about Redmine API

Following is the description about setting up Redmine API.

6.2.1 Redmine API

Redmine exposes some of its data through a REST API. This API provides access and basic CRUD operations (create, update, delete) for the resources defined in Redmine API website [3] under API description.

6.2.2 A simple Object Oriented PHP Redmine API client

A simple Object Oriented wrapper for Redmine API, written with PHP5 uses Redmine API. To install it composer [4] program is required for compiling it after that use the following code in PHP in order to include the API.

```
require_once 'vendor/autoload.php';
```

6.2.3 Creating Client

To create a client use the following code:

```
$client = new Redmine\Client('https://redmine.sis.uta.fi',  
'0561bd402d6d292e14179b0131d8f0f59070e0a4');
```

First parameter is the website address and second one is API accesses code which is possible to get from Redmine website (currently uses a reporter account to get information base on that).

6.2.4 Getting array of data

The following code is used to get an array of data from Redmine

```
$issue = $client->api('issue')->all([  
    'limit' => $count_issue  
]);
```

This array is getting 'issue' resource from Redmine API and to see full list of resources please check Redmine Rest API website [3] under API description and on each resources there is another link which can give you information about what kind of data you can get from this resources.

With all we can get all data and 'limit' is necessary for setting it to maximum amount because automatically this API has a limit on how many data it can get (probably it will only get 20-30 last data) with \$count_workh I set this limit to maximum data that is available in the Redmine db.

6.3 Example about getting this data

As you can see the following xml file is written in [5] as part of Issues can be get from Redmine API,

```
<issues type="array" count="1640">
  <issue>
    <id>4326</id>
    <project name="Redmine" id="1"/>
    <tracker name="Feature" id="2"/>
    <status name="New" id="1"/>
    <priority name="Normal" id="4"/>
    <author name="John Smith" id="10106"/>
    <category name="Email notifications" id="9"/>
    <subject>
      Aggregate Multiple Issue Changes for Email Notifications
    </subject>
    <description>
      This is not to be confused with another useful proposed feature that
      would do digest emails for notifications.
    </description>
    <start_date>2009-12-03</start_date>
    <due_date></due_date>
    <done_ratio>0</done_ratio>
    <estimated_hours></estimated_hours>
    <custom_fields>
      <custom_field name="Resolution" id="2">Duplicate</custom_field>
      <custom_field name="Texte" id="5">Test</custom_field>
      <custom_field name="Boolean" id="6">1</custom_field>
      <custom_field name="Date" id="7">2010-01-12</custom_field>
    </custom_fields>
    <created_on>Thu Dec 03 15:02:12 +0100 2009</created_on>
    <updated_on>Sun Jan 03 12:08:41 +0100 2010</updated_on>
  </issue>
  <issue>
    <id>4325</id>
    ...
  </issue>
</issues>
```


The array is defined as it has been said in section 2.3 now to use this array (which took a lot of my time to understand it because of lack of documentation in Redmine Rest API website) use the following code to get 'id' section and 'name' and \$i is the array index.

```
$project_id = $project['projects'][$i]['id'];
```

```
$project['projects'][$i]['name'];
```

6.4 File: getnumbers.php

Summary: Get and count all the needed data from Redmine using Redmine API

Description: This file start with includes for getting a connection to server and also for API. **\$client** is creating a new client connection for the server which gets the address and API accesses code then **\$con** connect to local database **mysqli_connect_errno(\$con)** checking for the possible errors, after that the codes SELECT working hours, issues, project of local database and save it to variables (**\$count_workh_localdb**, **\$count_issue_localdb**, **\$count_project_localdb** respectively) so I can show on the website, then I count the client (redmine) to get the total count from redmine database (variables : **\$count_workh**, **\$count_issue**, **\$count_project**).

6.5 File: redmine_working_hours.php

Summary: Read data from Redmine website

Description: Reading all (needed) the information from Redmine and saving inside of variables (namely: **\$working_hours**, **\$issue**, **\$project**) with the limits set to maximum numbers in the redmine database which is getting from previous file (getnumbers.php), there are some error checking part which needs to be better and currently its commented out, there is an issue with this code I searched and I could found any solution for it, the problem is when someone deleted any data from redmine database it can't be deleted from the local database (the possible solution for that can be redmine database send this codes notification when a data removed).

6.6 File: todatabase.php

Summary: This file is used to send data to local database and also includes codes for progress bar

Description: Firstly, use basic mysqli connection (with setting the maximum PHP execution time to 5mins) then, each time gets an element for progress bar calculation codes put into variables (for complete list look at the codes) and then insert to database without any error checking for duplicate data (simply pass duplicate data and getting this information from Project / Issue / Time_entry from redmine API) then close the connection.

6.7 Redmine Links

1. <http://redmine.sis.uta.fi/>
2. http://www.redmine.org/projects/redmine/wiki/Rest_api_with_php

3. http://www.redmine.org/projects/redmine/wiki/Rest_api
4. <https://getcomposer.org/download/>
5. http://www.redmine.org/projects/redmine/wiki/Rest_Issues

7. Facebook Interaction

7.1 Introduction

This document intends to provide a general overview of the Facebook-related part of the metrics monitoring tool. In particular, it lists the files involved in this aspect of the product, their functions, dependencies and known issues. An enumeration of the more generic problems with the application is followed by a brief discussion of future development opportunities and directions.

7.2 File listing

All of the files mentioned below are located in the `main/facebook_forum` directory. A more brief representation of the code flow is given after the list. Files are listed in the order they would normally be accessed by the application.

7.2.1 login.php

Inputs: alternatively, a Facebook access code or none

Outputs: saves a Facebook access token as a session variable

Possible callers: any project page

Calls: `initialization.php`, Facebook login service

Known issues: the request for the code-token exchange is made over plain HTTP, without SSL

`login.php` is the first page normally called by the application. (The expected “entry point” can be found in `main/scripts/metrics-elements-1.0.0.js`.) If it fails to detect a Facebook access token as a session variable, it directs the user to the login page. The login process is such that the user returns to `login.php` with an access code attached to the redirect link. An HTTP request then exchanges this code for the token proper and saves it. The page ultimately directs the user to `initialization.php`; a user already logged onto Facebook elsewhere would not be offered a login dialog at all. Other Facebook-related pages also direct the user here if there is no token in their session.

Importantly, `login.php` is the only file in this segment of the application that must include an absolute URL on two occasions – Facebook is not able to return the user to the right place given just a relative URL. All other files use exclusively relative URLs for redirection, and therefore should operate normally as long as the general project structure is preserved. It is only this `login.php` file where the links must be set properly in any case.

7.2.2 initialization.php

Inputs: Facebook access token as a session variable

Outputs: none, user interaction via a Web form

Possible callers: `login.php`

Calls: `login.php` (for unauthorized users), `results.php` (on form submission), `addgroup.php`, `refresh.php`, `update.php` (via AJAX)

Known issues: HTML incompatibilities likely resulting in a browser-specific display

This page integrates a Web form into the frames, the header and the footer maintained across the rest of the application's pages. The form features several fields for selecting the desired date range, number of posts to view and the ID of the necessary Facebook group, with some JavaScript-based input validation methods. Buttons are also available for adding new groups to the database and updating their member lists. These requests, as well as automatic updating of the name filter based on the group ID, are implemented via AJAX. To avoid redundancy, the same function dispatches the request, while the data sources and the target scripts can be specified as its arguments. Submitting the form displays the query results in another page, which, however, attempts to follow the same rendering style. See section 2.3 for details on that page.

7.2.3 results.php

Inputs: Facebook access token as a session variable, query parameters as POST variables

Outputs: query-specific listings and statistics

Possible callers: `initialization.php`

Calls: `login.php` (for unauthorized users)

Known issues: broken logic for post creation/update time

`results.php` serves to display the outcome of a query submitted via the Web form. It mentions the creation time and author of the most recent post, the total number of posts created within the specified date range, then lists every included post together with its comment and like counts. Comments are also provided in their entirety. Posts not containing any text are omitted by Facebook when returning the response. The actual logic behind the current combination of Facebook request parameters and server-side filtering is such that the output contains all the posts created in the given date range, except for those updated past that range: these do not make it into any subsequent figures derived.

7.2.4 addgroup.php

Inputs: Facebook group ID as a GET variable, access token as a session variable

Outputs: none

Possible callers: `initialization.php` (via AJAX)

Calls: `login.php` (for unauthorized users)

Known issues: lacks proper reporting whether the operation succeeded

This script must be called whenever a completely new Facebook group is to be added to the application and the database. It records the new group in `facebook_group`, requests the member list from Facebook and adds any newly encountered names to `facebook_member` (`INSERT` statements are sent separately, so that no single duplicate could abort the whole process). Note that this does not update the group-member relationships in `link_table`. While this file is used by pressing the "Add group to list" button, the update in `link_table` must be triggered afterwards by pressing the other button. See section 2.6.

7.2.5 refresh.php

Inputs: Facebook group ID as a GET variable

Outputs: all group members wrapped in HTML options

Possible callers: `initialization.php` (via AJAX)

Calls: nothing

Known issues: none

Requested via AJAX, this script fires once when `initialization.php` is loaded and whenever the group ID field changes in any way. It tries to look up the corresponding group's member list in the database and returns that list as the "inside" of an HTML `select` element. Per default behaviour, the "Everyone" option is necessarily included at the start of the list, so that it is returned even if the ID is not recognized.

7.2.6 update.php

Inputs: Facebook group ID as a GET variable, access token as a session variable

Outputs: newly-fetched group memberships in `link_table`

Possible callers: `initialization.php` (via AJAX)

Calls: `login.php` (for unauthorized users)

Known issues: member IDs are drawn via an SQL query with a potentially long `IN (...)` clause

This file is called by using the form button labeled "Update member list in DB". When the form contains a valid Facebook group identifier, the script proceeds to fetch the group's database ID from `facebook_group` and also requests the group's current member list from Facebook. It then matches member names to their database IDs in `facebook_member`, deletes all the corresponding group-member pairs from `link_table` and writes them over (thus reflecting any new group commitments made by Facebook users since the last such update operation). This was seen as simpler to implement than trying to update only those users who have left or joined a particular group.

Fig. 1 below demonstrates the relations between the scripts listed.

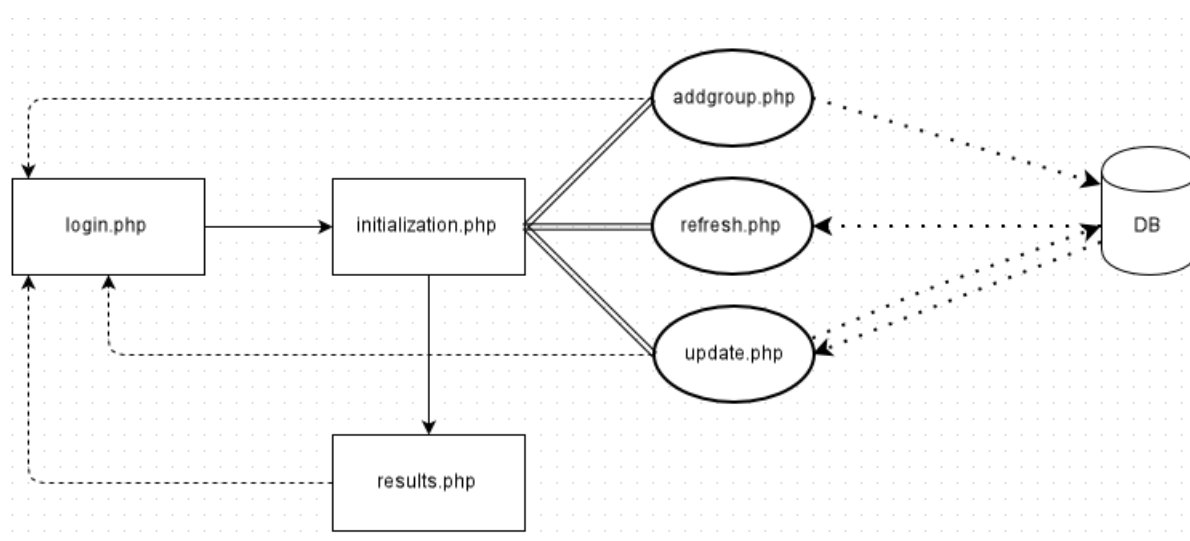


Fig. 1 – State transfers within the application.

In the figure, solid single lines indicate expected control flow, solid double lines denote AJAX connections, and dashed lines indicate abnormal flow and dotted lines stand for database connections.

7.3 Global issues

In addition to the script-specific issues listed in section 2, we also mention a number of more general problems relevant to the Facebook aspect of the tool on the whole. Some of these merely require additional development effort, while some are bound by external circumstances and may call for a reconsideration of the project requirements.

7.3.1 Group permissions

An application must be aware of the user's belonging to a group for a call to the group's feed (and thus all comment and post data) to succeed. Not knowing that will fail the call even if the user does belong to the group. There is (probably) no way to notify the application of this other than by explicitly displaying permission dialog and asking the user for this information. However, the group list is an extended permission that cannot even be asked for by default; instead, Facebook must review the application before that is possible. Even so, it is clearly stated that new applications on typical platforms, like Web or Android apps, will not be granted this permission anyway.

The only way to work around this known so far is to add at least the maintainers of the product as the application's administrators. Administrators can give their own applications all kinds of permissions, including extended ones. The process will work as long as the administrators belong to every group they are going to monitor. For the development process, it may be appropriate to include also the whole project team as administrators, if only temporarily. This action must be extended to every new member by the original administrator (or possibly any other one), and each new member must first register as a Facebook developer to be able to accept it.

7.3.2 Error handling

In the current code, exceptions thrown by Facebook requests may at least have the message printed out, but otherwise the errors are displayed as-is. There is still room for testing pages in all sorts of conditions until no PHP errors can be seen; the exact style of the replacement message is, of course, customizable. The following test cases may be of interest:

- refusing to login when asked to;
- providing mistaken or downright missing form data;
- selecting a group ID without belonging to that group;
- using the application without being on its developer list;
- failing database queries.

7.3.3 Input validation

While the code performs some error-checking on most pages, input validation in general is not completely covered. It should be conducted primarily on the server side and secondarily on the client side. This applies to every input field where arbitrary input is possible. Special care must be given to controls interacting with the database, preventing all kinds of SQL injections. One way to aid that is to rewrite the database connections using PDO and prepared statements instead of MySQLi.

7.4 Miscellaneous

The code makes a number of questionable assumptions about the IDs used by Facebook. In particular, the group ID is not necessarily a 15-digit number. It is also worth checking whether a non-numeric ID is acceptable and whether it always corresponds to an obtainable numeric value.

Occasionally, Facebook fails to return all the group members in a request. It can be checked that the request itself misses a couple of names, even when run within Facebook's own API sandbox, but the member listing on the group page shows them all. Exactly what is special about the missing entities is unclear.

The application does not track the user logging out of Facebook, and therefore does not handle the matters of their access token being invalid.

7.5 Further development suggestions

Apart from resolving the issues mentioned above, there are also some feature-oriented proposals which may hint at future directions in the development process for the current product:

- Statistics derived from Facebook could be summarized and displayed in various graphical representations, such as the charts already utilized in another part of the tool.
- The interface could be reorganized to include more functions for interacting with the database and updating it as necessary.
- As long as the user can derive and enter their metrics for evaluating the project quality, it could be rated based on the actual project statistics. (For example, one could declare "list all projects with fewer than 5 weekly comments by at least 3 unique users as potentially problematic".)