Bachelor's thesis

Information Technology

2018

Dmitri Kalinin

# DATABASE WEB APPLICATION

BACHELOR’S THESIS | ABSTRACT

TURKU UNIVERSITY OF APPLIED SCIENCES

Information Technology

2018 | 30

Dmitri Kalinin

# DATABASE WEB APPLICATION

The thesis is based on the development need for a database for the Research Water Engineering group. The research group expertise lies in water quality monitoring and has monitoring stations which collect the data.

The purpose of this thesis was to create the research group’s own database platform for the work with data and visualization of specified data into the charts on the web pages of the research group.

The carried-out project included research of the methods for achieving the necessary goals and implementation of the found solutions. Free license IT tools such as MySQL database, Laravel framework, Dygraph and DataTables JavaScript libraries were used for the project. This thesis documents the development process, analyses the provided choices and discusses the resolutions strengths and weaknesses of the selected solutions.

The result was successfully achieved. The database offers many features for convenient work with the data. Some of its functionality includes searching, filtering, sorting, marking, editing and other operations with the data. The result was a functional web database application which will be used for data handling and visualization in the charts for the purposes of the research group. Database security and performance were carefully considered and achieved.

In conclusion, the implementation of the database web application demonstrated the aspects of development process and objectives realization. Finally, the results are used for the purposes of the research group.

KEYWORDS:

Database, Web Application, Data, SQL, Charts, Laravel, PHP, Security

# CONTENTS

# FIGURES

# LIST OF ABBREVIATIONS (OR) SYMBOLS

| | |
|---|---|
| AJAX | Asynchronous JavaScript and XML |
| CMS | Content Management System |
| CSRF | Cross-Site Request Forgery |
| CSS | Cascading Style Sheets |
| CSV | Comma-Separated Values |
| FTP | File Transfer Protocol |
| HTML | Hypertext Markup Language |
| HTTP | Hypertext Transfer Protocol |
| JS | JavaScript |
| JSON | JavaScript Object Notation |
| PDO | PHP Data Objects |
| PHP | Hypertext Preprocessor |
| SQL | Structured Query Language |
| SSL | Secure Socket Layer |
| XML | eXtensible Markup Language |
| XSS | Cross-Site Scripting |
| Abbreviation | Explanation of abbreviation (Source) |

# 1 INTRODUCTION

Databases are a significant part of a modern rapidly developing digital world. Even though it is not essentially obvious at first sight once examples are brought to the point, clarity immediately appears. Starting from widescale banks to the low distributed enterprises databases are used in every industry and have massive impact on the century of information technologies. Correspondingly it is hard to imagine a modern world without databases. Thus, the databases together with online services are at extremely high relevance now.

This thesis represents the idea and realization of a functional database web application for real working life tasks. This thesis project was commissioned by Research Water Engineering group. It comprises the need of database application to provide multiple features required for comfortable usage and stable work for the users. Previously, the group had used external services for data processing and analysis but decided to develop and hold its own separate databases. The implementation of an own platform for a specific project is a great advantage when looking at future convenience and costs.

The author’s objectives in this thesis work were to research the web application and find appropriate methods and solutions for its implementation within a specified time-frame. Therefore, thesis describes the concrete challenges which were faced during the development of the database and their resolution. Data results analysis was not a part of the thesis but application tools choice was analyzed and presented.

# 2 RESEARCH

## 2.1 Background

“The Water Engineering research group at TUAS has firm expertise on water protection and monitoring issues, the marine environment, and wastewater treatment. The research group has long experience in R&D projects related to the protection and monitoring of aquatic environment and organisms, the restoration of water bodies and the water quality monitoring (WQM)” (TUAS 2018).

The group has special proficiency in the use of continuous WQM devices in natural water bodies. For the thesis project, the data collected by continuous cyanobacteria monitoring stations were used as a test data. Monitoring stations have been used by TUAS Water Engineering research group since 2006 (TUAS 2018). Cyanobacteria (also known as blue-green algae) may form harmful algal blooms which can be harmful both for human health and water resources management. Continuous monitoring stations with real-time data collection and transmission can be used as a warning system in the areas with high recreational importance. Monitoring is implemented as a service for public or private stakeholders, such as cities or tourism businesses. In 2017 three stations in total were deployed, one on lake Littoistenjärvi (city of Kaarina) and two stations on public swimming beaches (City of Parainen). Monitoring stations were equipped with a phycocyanin (PC) fluorescence sensor (TriOS microFlu-blue, trios.de) and a separate temperature sensor (Fig. 1).

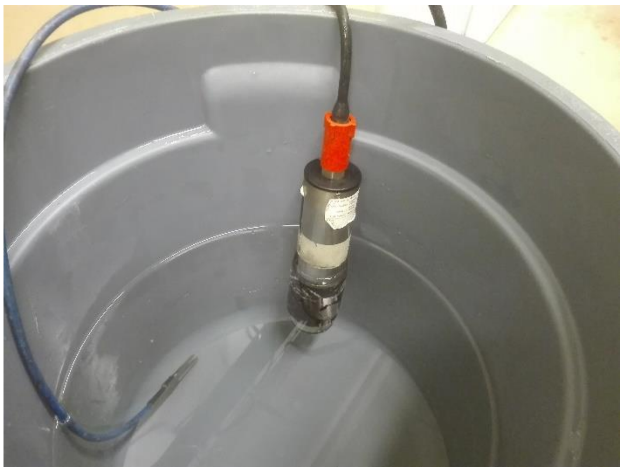


Figure 1. The measurement sensor used at measurement sites.

Measured PC fluorescence results are converted to cyanobacteria concentration by data logger (Fig. 2). To achieve this a calibration formula obtained from water sample analysis is used. In addition to actual measurement results, the datalogger sends the station battery voltage to decrease the need of maintenance visits.

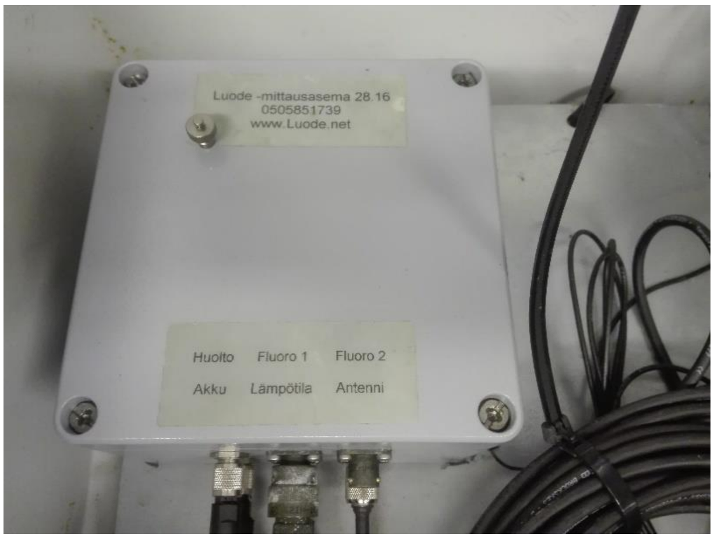
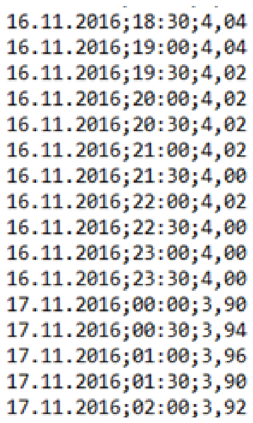


Figure 2. Data logger.

The collected data is sent to a server at TUAS where it is stored in CSV files. Each parameter is stored in separate CSV file (Fig. 3).



```
16.11.2016;18:30;4,04
16.11.2016;19:00;4,04
16.11.2016;19:30;4,02
16.11.2016;20:00;4,02
16.11.2016;20:30;4,02
16.11.2016;21:00;4,02
16.11.2016;21:30;4,00
16.11.2016;22:00;4,02
16.11.2016;22:30;4,00
16.11.2016;23:00;4,00
16.11.2016;23:30;4,00
17.11.2016;00:00;3,90
17.11.2016;00:30;3,94
17.11.2016;01:00;3,96
17.11.2016;01:30;3,90
17.11.2016;02:00;3,92
```

Figure 3. Example of unformatted raw data in CSV file. Semicolon is used as delimiter.

The data includes a date-time stamp and value of measured parameter. However, when all the raw data comes to one server, it needs to be properly handled before it is presented to the public. To achieve that, data is transferred to the web application into a script where formatting and filtering occurs. The next phase is the visualization of processed data to end users. It happens through web application via charts. Data charts are hosted at the server and links to them are provided to the customers so they can share charts on their pages. Coupled with the chart nearby located simple “traffic light” system is presented and indicates the level of cyanobacteria concentration. High concentrations may cause risk for human health. Therefore, if the value becomes too high a color changes as visual warning to prevent people from going into the water.

## 2.2 Purpose of database realization

The development of a database comes with needs of data processing and speeding improvement, maintenance, simple interaction and quick access. Charts without a database can be only generated straight from a CSV file. However, sometimes devices can crash or a sensor may require calibration. In those cases, some data errors may show up in the chart so editing CSV files with a huge quantity of records is going to be very labor-intensive and time-consuming. Besides that, it is not the best idea to modify original CSV files because this not appropriate way of fixing a problem and it can cause extra bugs. The creation of database application allows solving such issues by implementing flexible interaction with data and control over it. That allows wrong information to be rectified in a proper manner. Therefore, having a database is great advantage which simplifies the whole process of presentation and storing data in correct formats. Moreover, the options such as connection between data sets and timing, track of editing activity, administration panel and prerequisites for further database development with possible stations expansion become available. In addition, security markedly plays a positive role.

This thesis project had the following list of requirements:

- Research and design of database web application.
- Data presentation via charts.
- Modifying data features.
- Marking data errors.
- Filtering and search options.
- Tracking of station’s location changes.
- Automatic daily data updates.
- Adding new stations feature (database expansion).

- Documentation for the client, i.e. the research group.

During the development stage multiple obstacles were revealed. Here is the list of main challenges related to chart or database development:

- Research of solutions and methods for development.
- Converting data in appropriate format.
- Combining values according correct date and time
- SQL data handling and performance.
- Administration tools and security.
- Testing and transfer to the hosting.

Figure 4 and 5 shows the same chart with and without error reduction respectively.

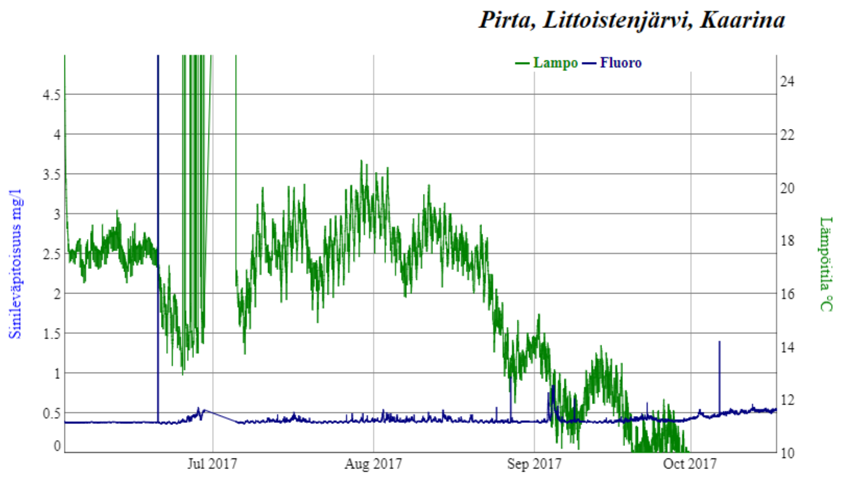


Figure 4. Chart’s outlook with some erroneous values.

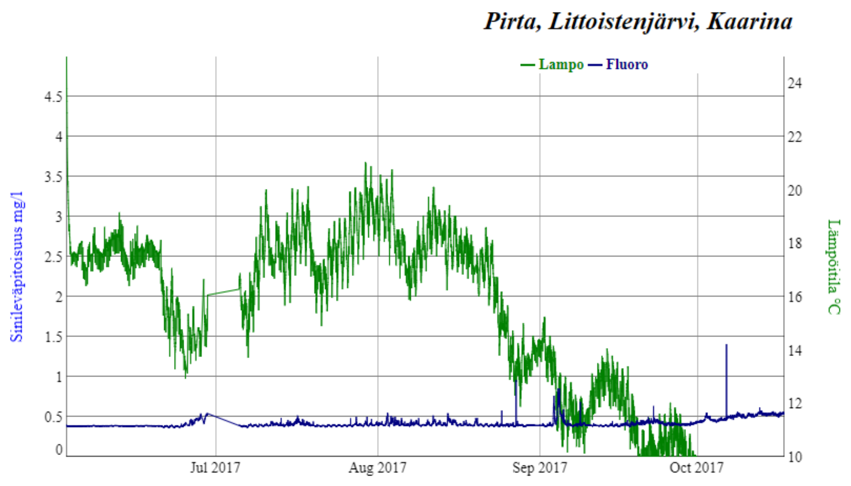


Figure 5. Chart’s outlook without erroneous values.

## 2.3 Implementation

All collected data from the stations are transferred to a server located in the TUAS Campus. The server receives raw files in CSV format and sends them to the web hosting. Laravel PHP framework is used to build this application in combination with the MySQL database management system on the hosting server. The PHP language does all the main tasks and is responsible for operations with the data on the server-side. On the other hand, the JavaScript language is responsible for the client-side part together with HTML and CSS. In addition, Dygraph and DataTables open source JS libraries are used in the project. They are significantly useful for data presentation view and other features. This mentioned model of implementation plan is visualized in Figure 6.

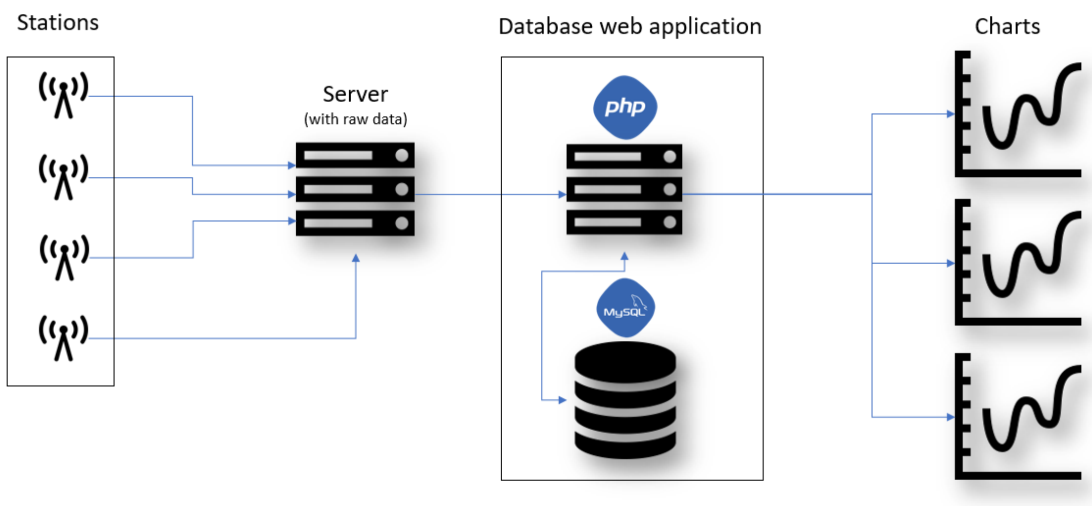


Figure 6. Visual plan of database web application.

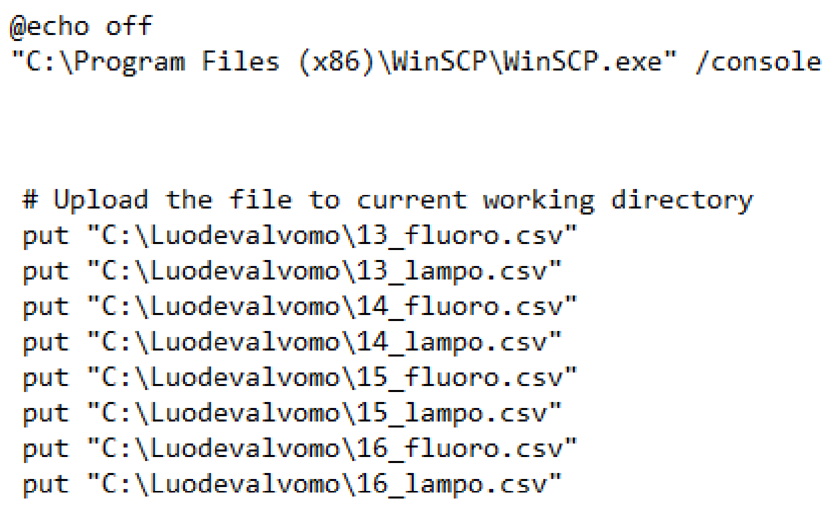
### 2.3.1 Data dispatching

The server receives data from stations into a determined folder and the process takes approximately from 5 to 10 minutes. Nevertheless, every station transfers the data at different times four times a day.

Here is example of stations data dispatching times:

- 00:20, 06:15, 12:15, 18:15 - Kaarina (station 13)
- 23:45, 05:45, 11:45, 17:45 - Nauvo: (station 14)
- 00:05, 06:00, 12:00, 18:00 - Parainen (station 15)
- 01:15, 06:45, 12:30, 17:30 - Nauvo (station 16)

Since difference in time is not too great, the update of all charts is implemented at the same time. Thus, transferring of all data to the web application starts right after the server receives data from the last station. A simple script (Fig. 7) executes the transfer operation via FTP of the WinSCP client using hosting credentials. A batch file calls the script when new data arrives into the folder.

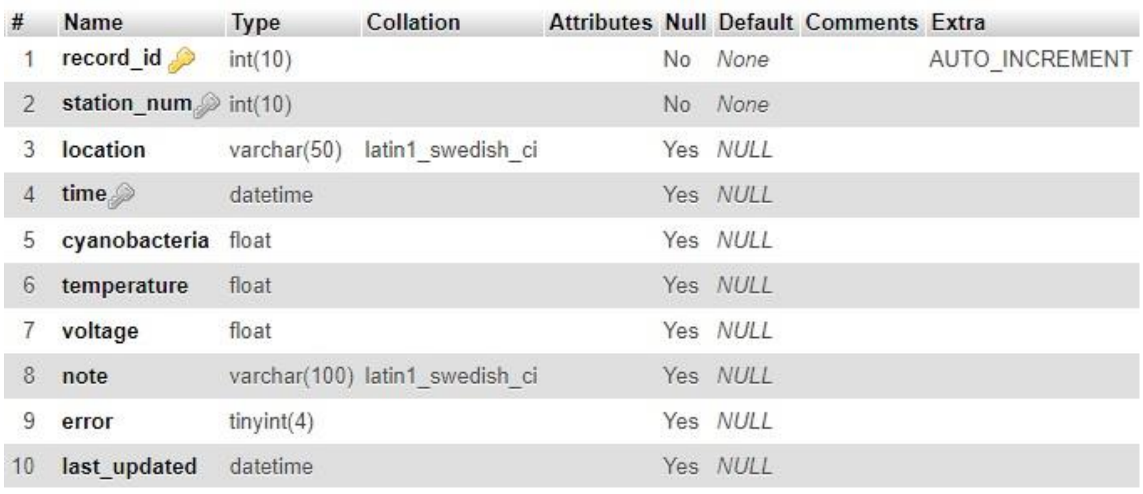


```
@echo off
"C:\Program Files (x86)\WinSCP\WinSCP.exe" /console


# Upload the file to current working directory
put "C:\Luodevalvomo\13_fluoro.csv"
put "C:\Luodevalvomo\13_lampo.csv"
put "C:\Luodevalvomo\14_fluoro.csv"
put "C:\Luodevalvomo\14_lampo.csv"
put "C:\Luodevalvomo\15_fluoro.csv"
put "C:\Luodevalvomo\15_lampo.csv"
put "C:\Luodevalvomo\16_fluoro.csv"
put "C:\Luodevalvomo\16_lampo.csv"
```

Figure 7. Snippet of codes from the script programmed to send CSV files to the hosting.

## 2.3.2 Database design

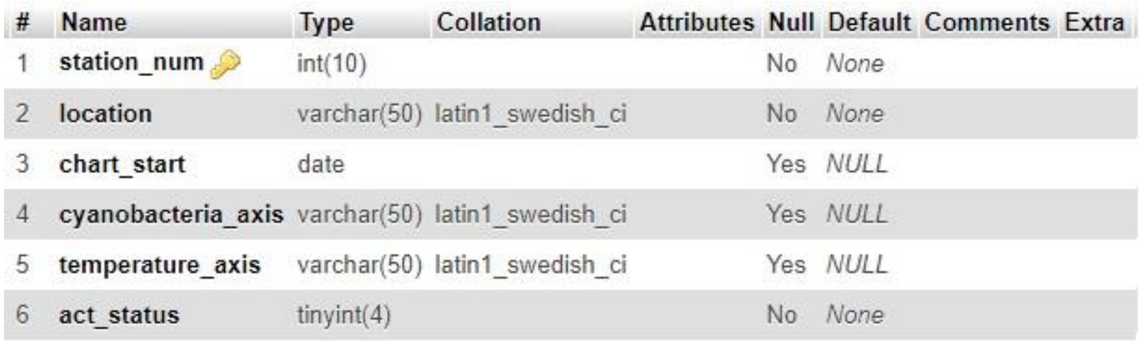
The database is designed in such a way that it has a main table template where each station's data is stored (Fig. 8). The table consists of columns such as record ID, station number, time and other data parameters. The primary key is set for "`record_id`" field and represents the unique ID of the record. It also has the auto increment feature which stands for automatic increase of ID by one with every new insert statement. Another column, "`time`", is bounded with the "`station_num`" column as a composite key which similarly can only be a unique value same as the ID despite that it is not a primary key. "`latin1_swedish_ci`" is set by default and supports possible Finnish language special characters.



| # | Name | Type | Collation | Attributes | Null | Default | Comments | Extra |
|---|---|---|---|---|---|---|---|---|
| 1 | record_id | int(10) | | | No | None | | AUTO_INCREMENT |
| 2 | station_num | int(10) | | | No | None | | |
| 3 | location | varchar(50) | latin1_swedish_ci | | Yes | NULL | | |
| 4 | time | datetime | | | Yes | NULL | | |
| 5 | cyanobacteria | float | | | Yes | NULL | | |
| 6 | temperature | float | | | Yes | NULL | | |
| 7 | voltage | float | | | Yes | NULL | | |
| 8 | note | varchar(100) | latin1_swedish_ci | | Yes | NULL | | |
| 9 | error | tinyint(4) | | | Yes | NULL | | |
| 10 | last_updated | datetime | | | Yes | NULL | | |

Figure 8. Design of the "`station's records`" table.

Another table (Fig. 9) is used to store details about each station such as location, activity status, beginning of presenting data date and axes titles. Since the deployment location of monitoring stations may change, it is important to have the ability to change locations because users need to clearly see when a specific station's data has different locations. Changing the locations in the same table would cause overwriting of all old locations. For this reason, the database is designed to fetch information about the location from a separate table during the insertion of new records.

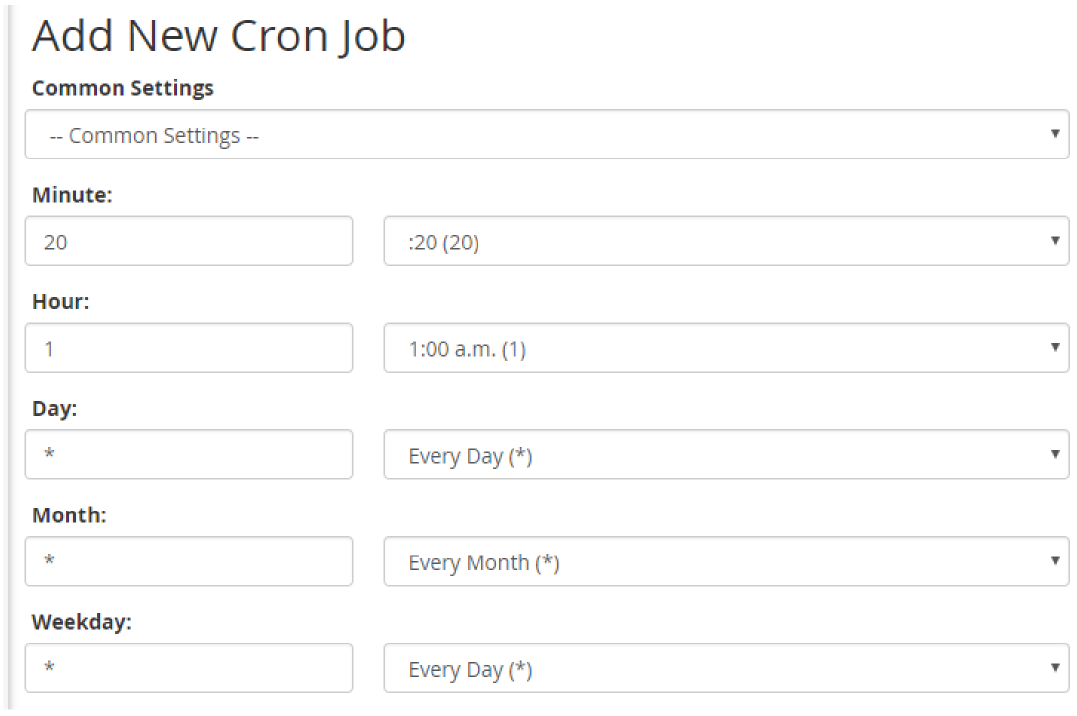


| # | Name | Type | Collation | Attributes | Null | Default | Comments | Extra |
|---|---|---|---|---|---|---|---|---|
| 1 | station_num | int(10) | | | No | None | | |
| 2 | location | varchar(50) | latin1_swedish_ci | | No | None | | |
| 3 | chart_start | date | | | Yes | NULL | | |
| 4 | cyanobacteria_axis | varchar(50) | latin1_swedish_ci | | Yes | NULL | | |
| 5 | temperature_axis | varchar(50) | latin1_swedish_ci | | Yes | NULL | | |
| 6 | act_status | tinyint(4) | | | No | None | | |

Figure 9. Design of "`all stations`" table.

### 2.3.3 Insertion and automatic updates to database

Hosting has a feature called Cron Job (Fig. 10) which executes a determined script at a certain time preset by the user. The required settings are the script's execution time and its path. Multiple Cron Jobs can be used at once with flexible control according to the project needs. In the given project, the script is executed on a 4-times a day frequency.

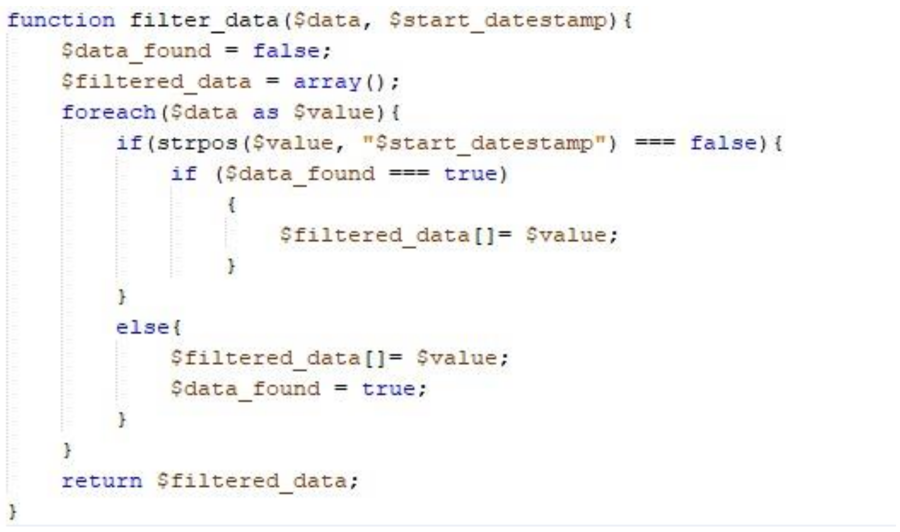


Add New Cron Job

| | |
|---|---|
| **Common Settings** | -- Common Settings -- |
| **Minute:** 20 | :20 (20) |
| **Hour:** 1 | 1:00 a.m. (1) |
| **Day:** * | Every Day (*) |
| **Month:** * | Every Month (*) |
| **Weekday:** * | Every Day (*) |

Figure 10. Cron Job example.

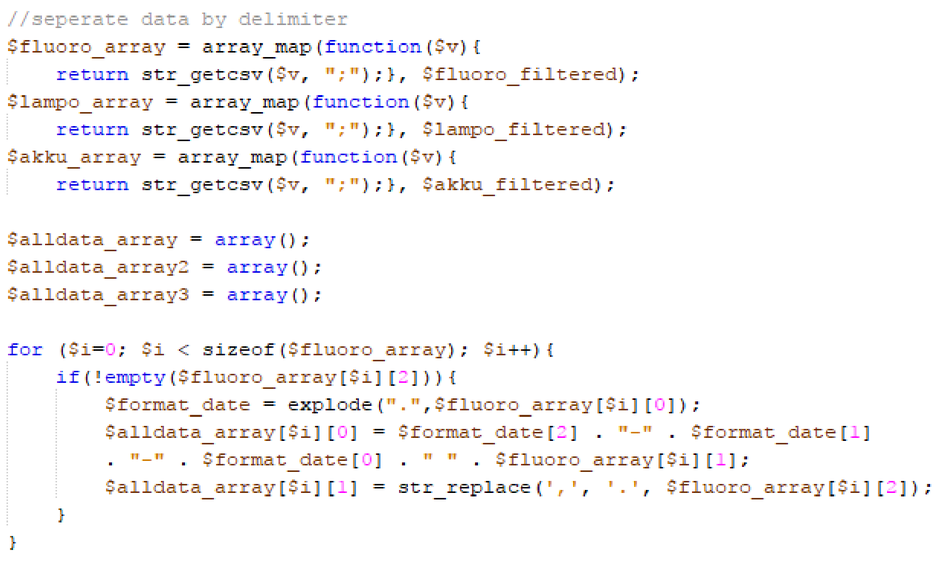
The PHP script is written to add new data from three CSV files per station (13_fluoro.csv, 13_lampo.csv, 13_akku.csv). "`fluoro`" stands for cyanobacteria concentration, "`lampo`" for temperature and "`akku`" for voltage of the battery. To do so standard "`file ()`" function is being used. Later script uses the "`filter_data ()`" function (Fig. 11) to filter data which is needed for current datetime database insertion. This function receives two parameters data itself and current datetime, selects all new data and assigns results to its new filtered arrays.



```
function filter_data($data, $start_datestamp){
    $data_found = false;
    $filtered_data = array();
    foreach($data as $value){
        if(strpos($value, "$start_datestamp") === false){
            if ($data_found === true)
                {
                    $filtered_data[]= $value;
                }
        }
        else{
            $filtered_data[]= $value;
            $data_found = true;
        }
    }
    return $filtered_data;
}
```

Figure 11. Function “filter_data ()” programmed filtering data corresponding to dates.

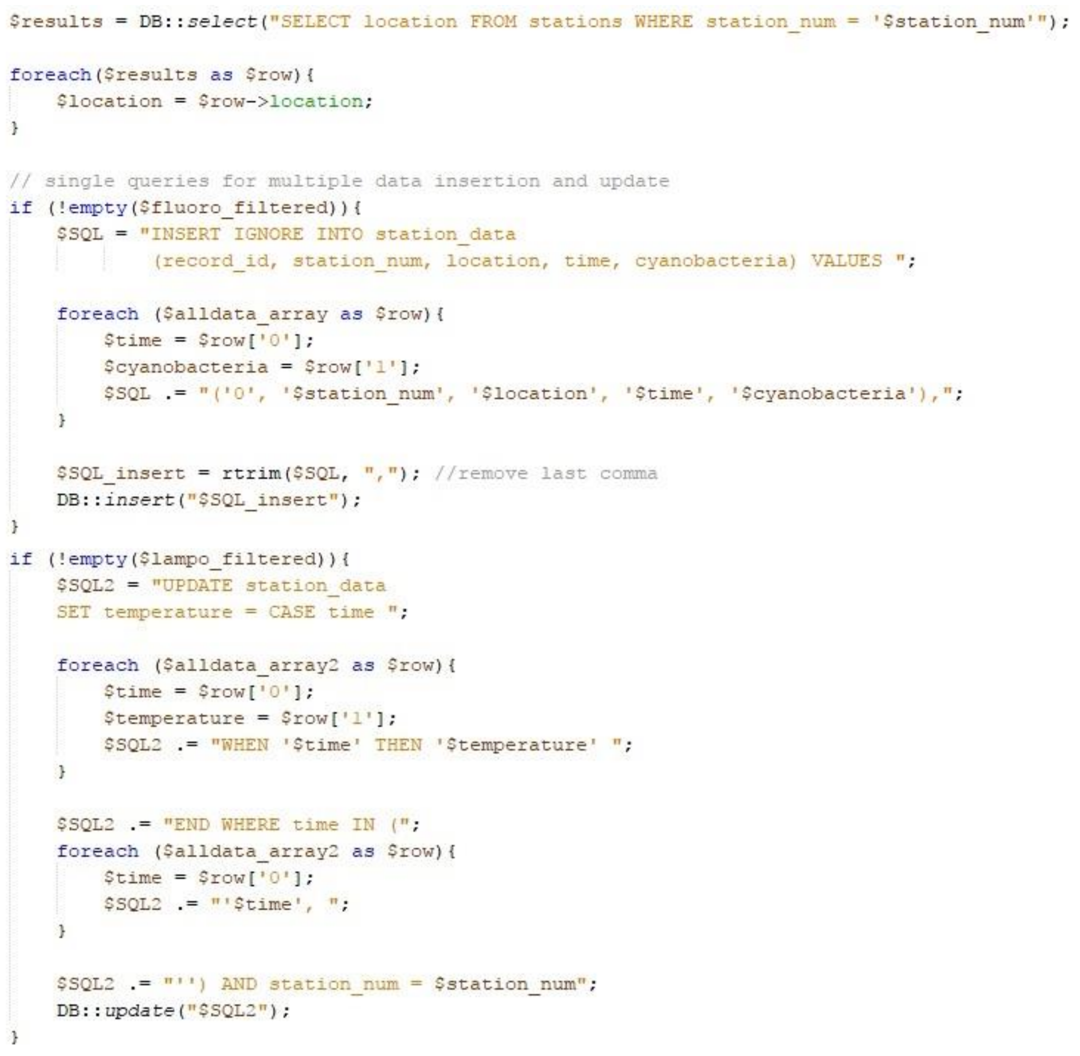
Next step is formatting the data to the correct format (Fig. 12).



```
//seperate data by delimiter
$fluoro_array = array_map(function($v){
    return str_getcsv($v, ";");}, $fluoro_filtered);
$lampo_array = array_map(function($v){
    return str_getcsv($v, ";");}, $lampo_filtered);
$akku_array = array_map(function($v){
    return str_getcsv($v, ";");}, $akku_filtered);

$alldata_array = array();
$alldata_array2 = array();
$alldata_array3 = array();

for ($i=0; $i < sizeof($fluoro_array); $i++){
    if(!empty($fluoro_array[$i][2])){
        $format_date = explode(".",$fluoro_array[$i][0]);
        $alldata_array[$i][0] = $format_date[2] . "-" . $format_date[1]
        . "-" . $format_date[0] . " " . $fluoro_array[$i][1];
        $alldata_array[$i][1] = str_replace(',', '.', $fluoro_array[$i][2]);
    }
}
```

Figure 12. Formatting data by separating delimiters and switching positions of dates, hours, etc.

Voltage values have measurements only one time per hour whenever the other two have measurements values every half an hour. Additionally, there happened to appear some device’s errors which resulted in the shift of timestamps between cyanobacteria and temperature data. This is the reason why script inserts one data first and the further after comparing timestamps, it updates the record with other data if the time is matched. SQL queries executes insert and update statements (Fig. 13). Only single SQL queries are used to increase performance.



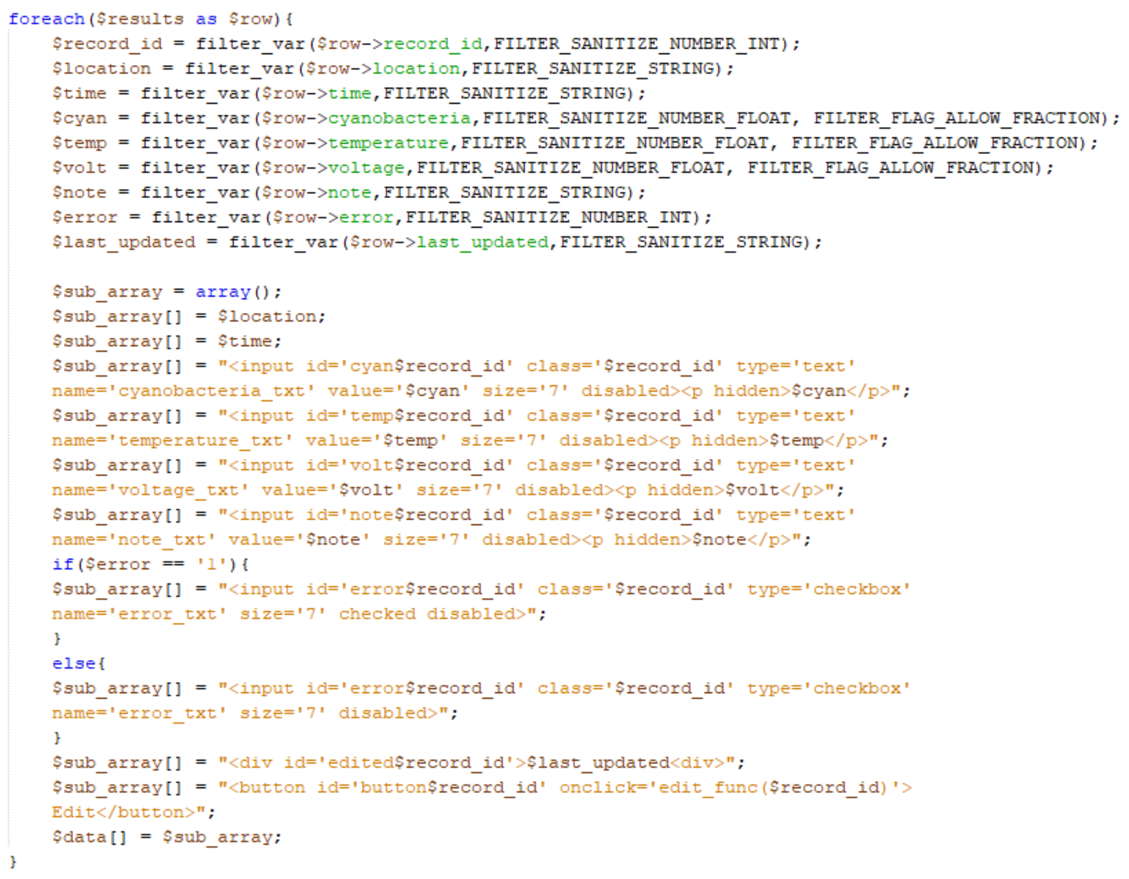
```
$results = DB::select("SELECT location FROM stations WHERE station_num = '$station_num'");

foreach($results as $row){
    $location = $row->location;
}

// single queries for multiple data insertion and update
if (!empty($fluoro_filtered)){
    $SQL = "INSERT IGNORE INTO station_data
            (record_id, station_num, location, time, cyanobacteria) VALUES ";

    foreach ($alldata_array as $row){
        $time = $row['0'];
        $cyanobacteria = $row['1'];
        $SQL .= "('0', '$station_num', '$location', '$time', '$cyanobacteria'),";
    }

    $SQL_insert = rtrim($SQL, ","); //remove last comma
    DB::insert("$SQL_insert");
}
if (!empty($lampo_filtered)){
    $SQL2 = "UPDATE station_data
    SET temperature = CASE time ";

    foreach ($alldata_array2 as $row){
        $time = $row['0'];
        $temperature = $row['1'];
        $SQL2 .= "WHEN '$time' THEN '$temperature' ";
    }

    $SQL2 .= "END WHERE time IN (";
    foreach ($alldata_array2 as $row){
        $time = $row['0'];
        $SQL2 .= "'$time', ";
    }

    $SQL2 .= "'') AND station_num = $station_num";
    DB::update("$SQL2");
}
```

Figure 13. Insert and update statements to combine data.

For the automatic updates, the script fetches the requested list of all active stations so the same process is repeated for each station by changing data input sources.

## 2.3.4 Data handling with DataTables.js

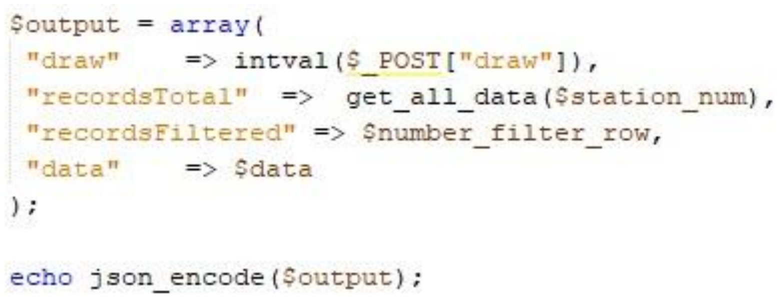
The JS jQuery library helps on the client-side and by being used together with AJAX technologies, it makes possible data handling on the fly without reloading the page (Cimo 2017, 14). In the web application, the data presented in tables. To accomplish higher quality results DataTable.js was utilized for the project. DataTables.js is an extremely useful plugin for jQuery which already has many built-in tools, such as search and sorting. Each station has more than 20 000 records, so to initialize all the data would take too much time and unreasonable load. Therefore, the server-side processing is implemented in this project. When the page is accessed DataTables.js receives information about the total quantity of records from the chosen station and the last 10 entries are loaded from the database. However, it is still possible to choose from 10 to 1000 rows to be presented in the table at once. The server-side script places data into PHP array as part of HTML code (Fig. 14) to structure the final table layout.



```php
foreach($results as $row){
    $record_id = filter_var($row->record_id,FILTER_SANITIZE_NUMBER_INT);
    $location = filter_var($row->location,FILTER_SANITIZE_STRING);
    $time = filter_var($row->time,FILTER_SANITIZE_STRING);
    $cyan = filter_var($row->cyanobacteria,FILTER_SANITIZE_NUMBER_FLOAT, FILTER_FLAG_ALLOW_FRACTION);
    $temp = filter_var($row->temperature,FILTER_SANITIZE_NUMBER_FLOAT, FILTER_FLAG_ALLOW_FRACTION);
    $volt = filter_var($row->voltage,FILTER_SANITIZE_NUMBER_FLOAT, FILTER_FLAG_ALLOW_FRACTION);
    $note = filter_var($row->note,FILTER_SANITIZE_STRING);
    $error = filter_var($row->error,FILTER_SANITIZE_NUMBER_INT);
    $last_updated = filter_var($row->last_updated,FILTER_SANITIZE_STRING);

    $sub_array = array();
    $sub_array[] = $location;
    $sub_array[] = $time;
    $sub_array[] = "<input id='cyan$record_id' class='$record_id' type='text'
    name='cyanobacteria_txt' value='$cyan' size='7' disabled><p hidden>$cyan</p>";
    $sub_array[] = "<input id='temp$record_id' class='$record_id' type='text'
    name='temperature_txt' value='$temp' size='7' disabled><p hidden>$temp</p>";
    $sub_array[] = "<input id='volt$record_id' class='$record_id' type='text'
    name='voltage_txt' value='$volt' size='7' disabled><p hidden>$volt</p>";
    $sub_array[] = "<input id='note$record_id' class='$record_id' type='text'
    name='note_txt' value='$note' size='7' disabled><p hidden>$note</p>";
    if($error == '1'){
    $sub_array[] = "<input id='error$record_id' class='$record_id' type='checkbox'
    name='error_txt' size='7' checked disabled>";
    }
    else{
    $sub_array[] = "<input id='error$record_id' class='$record_id' type='checkbox'
    name='error_txt' size='7' disabled>";
    }
    $sub_array[] = "<div id='edited$record_id'>$last_updated<div>";
    $sub_array[] = "<button id='button$record_id' onclick='edit_func($record_id)'>
    Edit</button>";
    $data[] = $sub_array;
}
```

Figure 14. Example of the code where elements have their own ID’s, which allows jQuery and DataTables.js have flexible control over it.

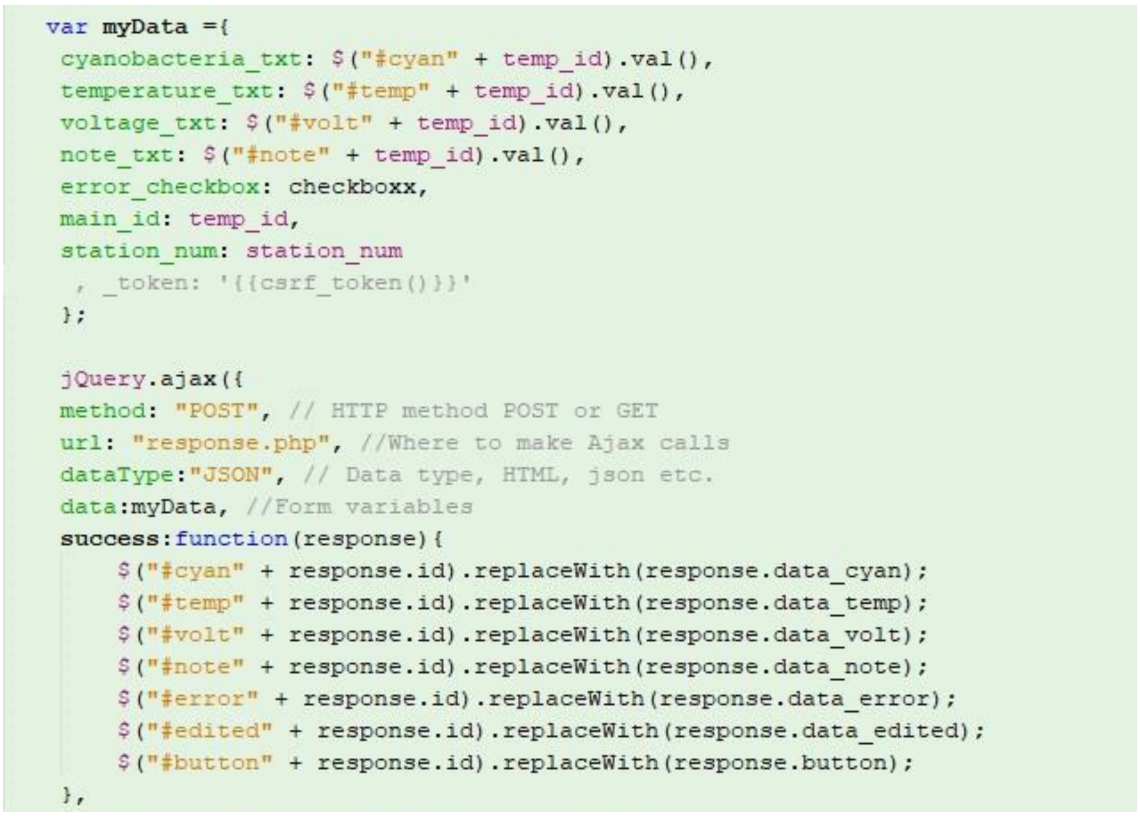
Afterwards, the data is converted to JSON (Fig. 15) format and is passed to the DataTable.js to be presented on the web page.



```
$output = array(
 "draw"      => intval($_POST["draw"]),
 "recordsTotal"   =>  get_all_data($station_num),
 "recordsFiltered" => $number_filter_row,
 "data"      => $data
);

echo json_encode($output);
```

Figure 15. JSON data conversion.

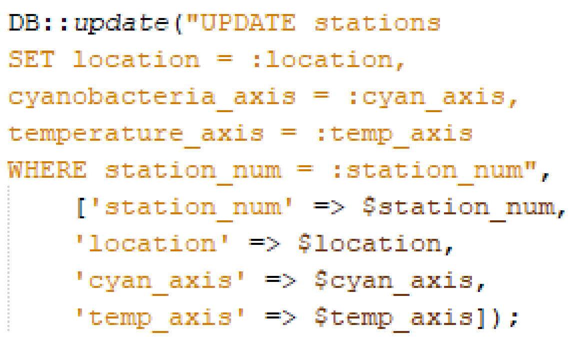
After the data has been loaded, the user is able to choose how to sort or filter table's data, and select what rows to edit. All those actions will initiate Ajax (Fig. 16). POST requests.



```
var myData ={
 cyanobacteria_txt: $("#cyan" + temp_id).val(),
 temperature_txt: $("#temp" + temp_id).val(),
 voltage_txt: $("#volt" + temp_id).val(),
 note_txt: $("#note" + temp_id).val(),
 error_checkbox: checkboxx,
 main_id: temp_id,
 station_num: station_num
  , _token: '{{csrf_token()}}'
 };

 jQuery.ajax({
 method: "POST", // HTTP method POST or GET
 url: "response.php", //Where to make Ajax calls
 dataType:"JSON", // Data type, HTML, json etc.
 data:myData, //Form variables
 success:function(response){
     $("#cyan" + response.id).replaceWith(response.data_cyan);
     $("#temp" + response.id).replaceWith(response.data_temp);
     $("#volt" + response.id).replaceWith(response.data_volt);
     $("#note" + response.id).replaceWith(response.data_note);
     $("#error" + response.id).replaceWith(response.data_error);
     $("#edited" + response.id).replaceWith(response.data_edited);
     $("#button" + response.id).replaceWith(response.button);
 },
```

Figure 16. Ajax POST request.

With the Ajax request, ID's of the selected rows are passed to the SQL update statement (Fig. 17) and result is immediately returned as JSON response without page reloading.

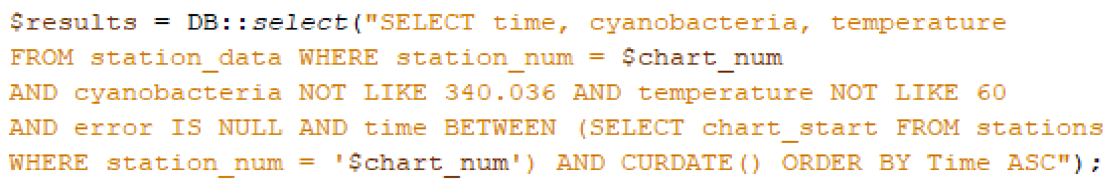


```
DB::update("UPDATE stations
SET location = :location,
cyanobacteria_axis = :cyan_axis,
temperature_axis = :temp_axis
WHERE station_num = :station_num",
    ['station_num' => $station_num,
    'location' => $location,
    'cyan_axis' => $cyan_axis,
    'temp_axis' => $temp_axis]);
```

Figure 17. Update SQL statement for modifying rows.

### 2.3.5 Data presentation with Dygraph.js

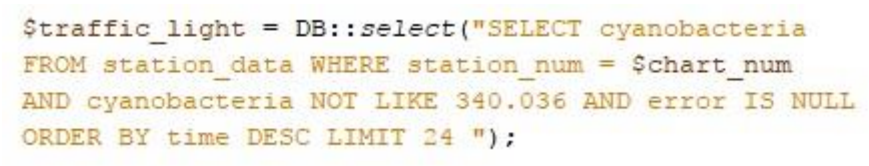
For presenting of data charts, the interactive Dygraph.js JavaScript library is used. This library specializes on charts and graphs. It requests the data through the SQL select statement (Fig. 18) and processes only time, cyanobacteria concentration, and temperature values.



```
$results = DB::select("SELECT time, cyanobacteria, temperature
FROM station_data WHERE station_num = $chart_num
AND cyanobacteria NOT LIKE 340.036 AND temperature NOT LIKE 60
AND error IS NULL AND time BETWEEN (SELECT chart_start FROM stations
WHERE station_num = '$chart_num') AND CURDATE() ORDER BY Time ASC");
```

Figure 18. Select statement for the Dygraph.js.

In effect, the "`where`" clause specifies which erroneous values should be eliminated from the chart. By default, sorting is set for the "`time`" column in ascending order. Once Dygraph.js receives the selected data, it parses information and draws the chart in the specified <div> tag. Axes are programmed to be auto scalable according to maximum and minimum values. It is important to have correct data formats for correct chart library work and that is also why data has been formatted into correct format before it is inserted in the database. Likewise, there is the SQL select statement (Fig. 19) which finds the highest cyanobacteria concentration value from the current day. This is needed for the traffic light system to be able set the appropriate color in accordance with the level of danger.



```
$traffic_light = DB::select("SELECT cyanobacteria
FROM station_data WHERE station_num = $chart_num
AND cyanobacteria NOT LIKE 340.036 AND error IS NULL
ORDER BY time DESC LIMIT 24 ");
```

Figure 19. Select statement to set correct traffic light color.

Only last 24 inserted values are taken to be recent enough. Thus, the values are checked to make sure that in last 12 hours cyanobacteria level concentration was low, and if not then warning will appear as a red signal.

The clients use charts on their pages by embedding them through an HTML5 <iframe> tag. However, the charts themselves are situated on the same with the database hosting.

### 2.3.6 Administration panel

By 2018 Laravel is most popular PHP framework (11 Best PHP Frameworks for Modern Web Developers in 2018, 2018). Laravel makes applications more secure, simplifies the work and has some great built-in features. For instance, login and registration in this project are configured by the framework as well as the main CSS layouts of the application administration panel (Fig. 20).

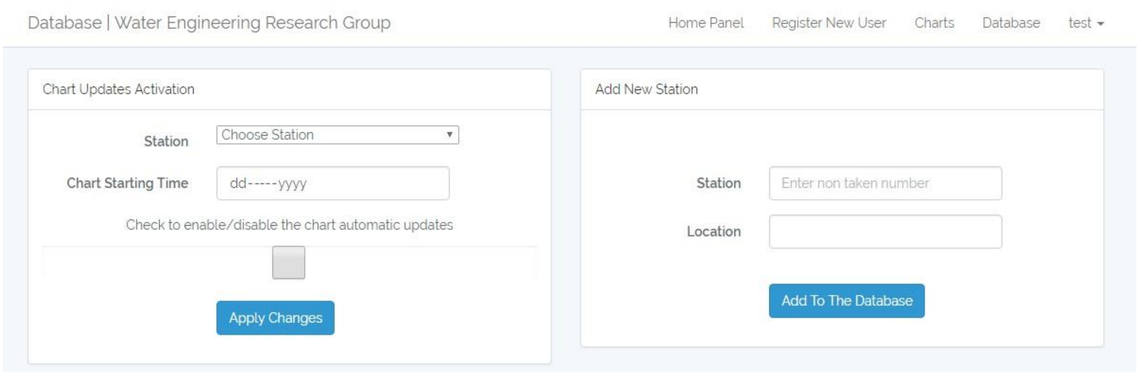
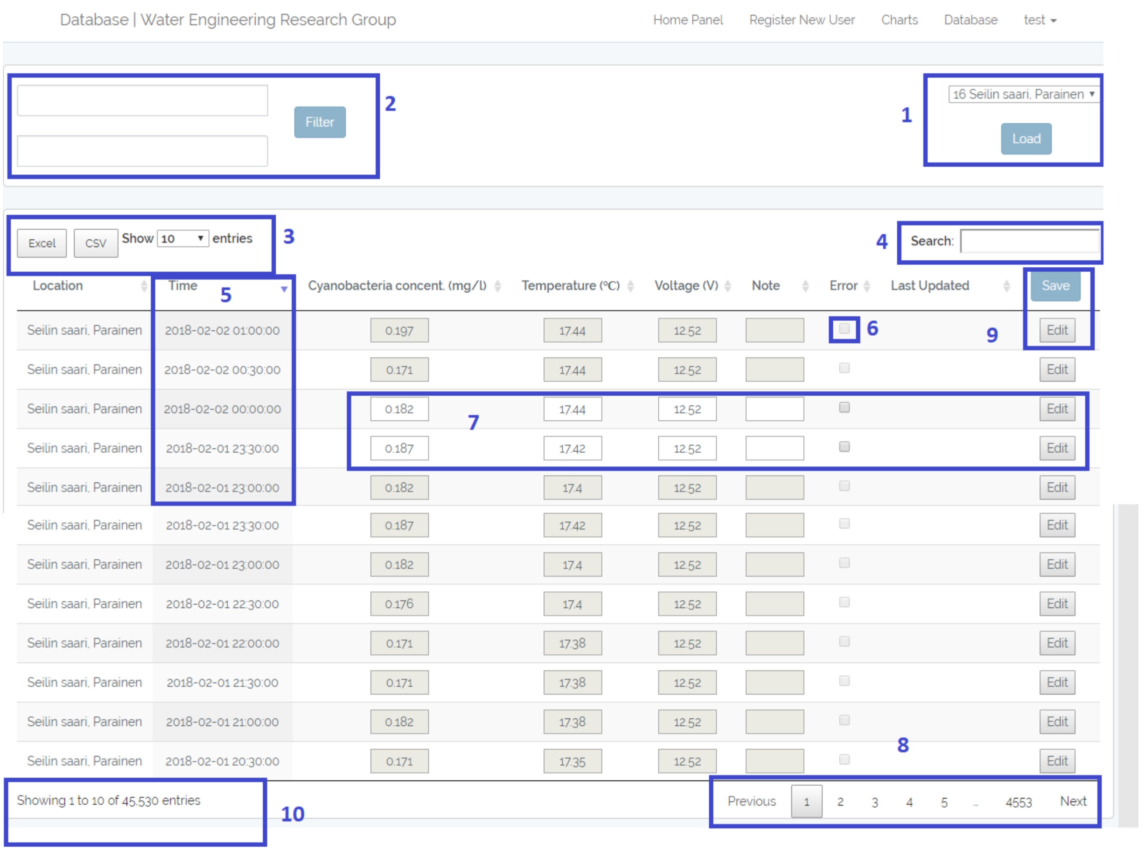


Figure 20. Example of navigation menu with some admin sections.

## 2.4 Results

A functional web application is the result of all mentioned aspects of work which includes scripts for handling the data, database tables and charts for visual view of the data. To give an illustration of result database (Fig. 21) and chart (Fig. 22) overviews are provided below.

### 2.4.1 Database overview



| Location | Time | Cyanobacteria concent. (mg/l) | Temperature (°C) | Voltage (V) | Note | Error | Last Updated | Save |
|---|---|---|---|---|---|---|---|---|
| Seilin saari, Parainen | 2018-02-02 01:00:00 | 0.197 | 17.44 | 12.52 | | ☐ | | Edit |
| Seilin saari, Parainen | 2018-02-02 00:30:00 | 0.171 | 17.44 | 12.52 | | ☐ | | Edit |
| Seilin saari, Parainen | 2018-02-02 00:00:00 | 0.182 | 17.44 | 12.52 | | ☐ | | Edit |
| Seilin saari, Parainen | 2018-02-01 23:30:00 | 0.187 | 17.42 | 12.52 | | ☐ | | Edit |
| Seilin saari, Parainen | 2018-02-01 23:00:00 | 0.182 | 17.4 | 12.52 | | ☐ | | Edit |
| Seilin saari, Parainen | 2018-02-01 23:30:00 | 0.187 | 17.42 | 12.52 | | ☐ | | Edit |
| Seilin saari, Parainen | 2018-02-01 23:00:00 | 0.182 | 17.4 | 12.52 | | ☐ | | Edit |
| Seilin saari, Parainen | 2018-02-01 22:30:00 | 0.176 | 17.4 | 12.52 | | ☐ | | Edit |
| Seilin saari, Parainen | 2018-02-01 22:00:00 | 0.171 | 17.38 | 12.52 | | ☐ | | Edit |
| Seilin saari, Parainen | 2018-02-01 21:30:00 | 0.171 | 17.38 | 12.52 | | ☐ | | Edit |
| Seilin saari, Parainen | 2018-02-01 21:00:00 | 0.182 | 17.38 | 12.52 | | ☐ | | Edit |
| Seilin saari, Parainen | 2018-02-01 20:30:00 | 0.171 | 17.35 | 12.52 | | ☐ | | Edit |

Showing 1 to 10 of 45.530 entries

Previous 1 2 3 4 5 … 4553 Next

Figure 21. Station’s data generated from database using DataTables.js. Numbers from the figure are described in the following paragraph.

The database table has option of choosing station to load from the all stations available list (1). It is necessary to choose the starting and ending dates to filter date (2) ranges. If another station is loaded, the date filter should be applied again. By default, ten records are presented in the table but it is possible to choose the quantity of rows needed to be shown on the page. All visible rows on the page can be saved in Excel or CSV formats (3). The search feature (4) works for all rows of the chosen station, even for not visible data rows. Sorting (5) works in the same way for all columns and rows of the chosen station. Check box (6) marks the data row which needs to be hidden from the chart. By default, all rows are disabled for editing. Rows (7) must be chosen by clicking the edit buttons to become editable and highlighted. Switching (8) between pages with data work through the server-side as well. The save and edit buttons are in the same column (9). All changes are carried out only by the save button confirmation. The left bottom corner (10) shows the total amount of records in the chosen station.

### 2.4.2 Chart overview

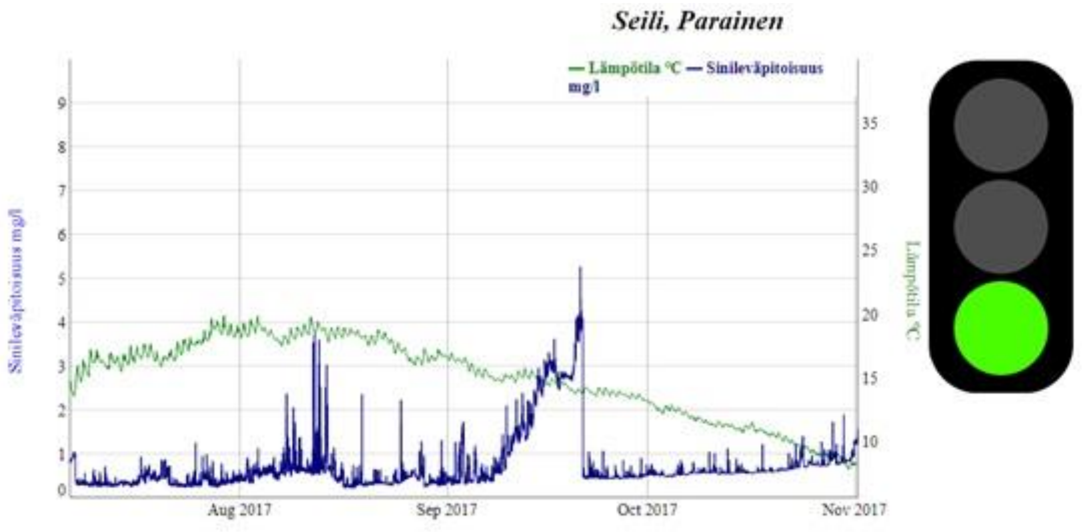


Figure 22. Chart with traffic lights system generated using Dygraph.js.

Chart fetches data straight from the Database. The blue color indicates cyanobacteria concentration and the green color indicates temperature. It is possible to modify the location title situated on the top of the chart and Y-axes titles in administration panel. Chart shows data starting from the station’s chart starting time which is set in the administration panel till current day. Erroneous and checked checkboxes values are hidden from the chart. Minimum and maximum temperature limit values and maximum cyanobacteria limit value are taken from the database and calculated from the period of chart starting time till the current day. The minimum cyanobacteria limit value is set to 0. Charts are auto scalable according to minimum and maximum parameters. It is possible to scale them manually as well. Traffic light color value fetches from the database the last 24 records of station related to the chart. It changes color if any of the last 24 records exceeds the limit. Cyanobacteria level color values are set between 0 and 2 for green, between 2 and 10 for yellow, and 10 or greater for red.

# 3 ANALYSIS

## 3.1 Analysis of the tools choice

The MySQL open source database has been used for 22 years. “MySQL is the most used database with PHP and both small and large companies such as Facebook, Twitter, and Wikipedia use it” (Kromann 2016, 333). So, it was chosen on the grounds of its verified status and decent advantages list. These include speed, scalability and security. Naturally, in some aspects MySQL might lose to other options since it has disadvantages, such as transactions reliability or some functional limitations. Nevertheless, for the web application purpose it suits perfectly with its simplicity of use and features choice. (Tezer 2014.)

In the beginning it was planned to use pure PHP but later after the analysis Laravel became the notably useful development tool for the app. Although on the web are plenty available options for example content management systems like WordPress, which is much easier for non-experienced users to manage, but it is certainly not better than using frameworks.

Firstly, it is much easier for other developers to become familiar with the framework projects unlike CMS systems projects. “Frameworks provide conventions that reduce the amount of code a developer new to the project has to understand - if you understand how routing works in one Laravel project, for example, you understand how it works in all Laravel projects” (Stauffer 2017, 24).

Secondly, frameworks are much better in performance and more flexible for developers. As example when Laravel fully supports PHP 7, WordPress might have missed its compatibility in some themes or plugins (Kazankov 2017).

Thirdly, some CMS plugins are made by mediocre developers and programmed in non-proficient manner or even includes security holes. That is how a website has more risks to be hacked. Laravel (Fig. 23) on another hand, has many security solutions available and its versatile secure authentication logic feature is another confirmation of that.

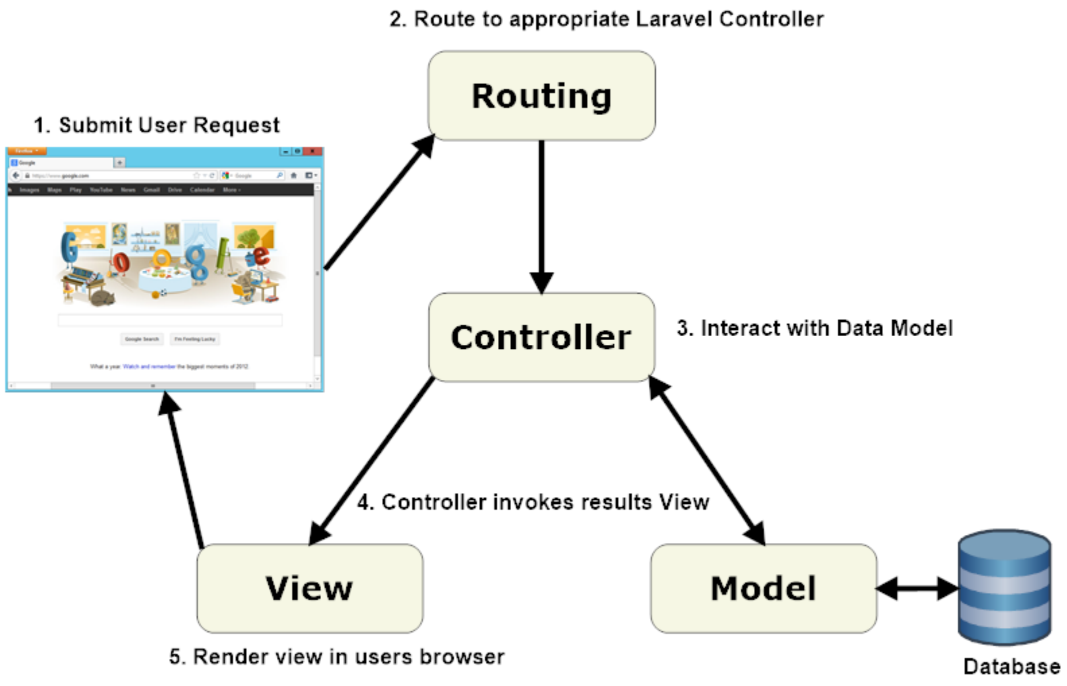

2. Route to appropriate Laravel Controller
Routing
1. Submit User Request
Controller
3. Interact with Data Model
4. Controller invokes results View
View
Model
5. Render view in users browser
Database

Figure 23. Laravel model view controller based architecture. (Malhotra 2018.)

## 3.2 Security

It is always important to keep an application secure and use standard protections against possible attacks such as SQL injections, XSS and CSRF hacks. For instance, attacks can be achieved by inserting undesirable or even harmful code either by GET or POST requests. Even though all application's inputs are allowed only for registered users and official workers of the research group, it is still a better policy to filter every unit of data coming from the users. Different security layers are applied in this application which ensure protection from the mentioned threats. (Rathore 2016.)

Most importantly in Laravel protective techniques are used to avoid undesirable issues. For example, CSRF attacks are prevented by using hidden _token input element. The entered data are handled by sanitizing the user inputs. Some fields are validated in relation to required formats via the "`FILTER_VALIDATE_FLOAT PHP`" filter. Others are sanitized by the PHP function "`htmlspecialchars`" which converts special characters into safe format. Occasionally inputs must not be converted at all provided that this need might happen if data must include some rare special characters or tags. In the given scenario, the only needed special characters are Finnish language letters which are not being converted. Escaping values via "`FILTER_SANITIZE_STRING`" are performed before delivering data to the client-side to make sure that users are protected against XSS.

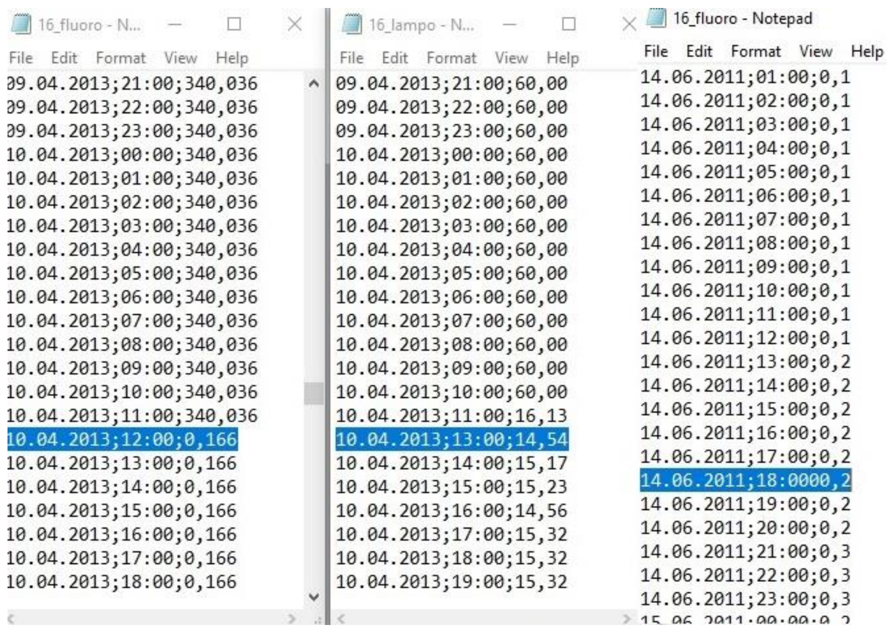
Furthermore, prepared statements and bind parameters are extremely good techniques against SQL injections. Unfortunately, they were not used for the insertion script. It was decided to leave them out because data is not entered via inputs in the CSV insertion script and due to performance profit. However, it is important that in all possible web application's input fields PDO's bind parameters are used during the query to prevent SQL injections. (PHP Documentation 2018.)

Moreover, the hosting site has the SSL certificate which protects the application from potential hackers trying to retrieve user information via session variables, coming as a plain text. So, for instance, attackers on the same network can read those non-protected data but SSL certificates prevent that by encrypting variables so information such as passwords becomes unavailable to malefactors. Those websites have URLs starting with HTTPS, where last letter stands for "secure".

A possible drawback is presence of the database on the same server because it is also a good practice to have it on separate one. On the other hand, this is not crucial thing and makes the process simpler. Another arguable drawback is the database table design. For convenience reasons, database tables could be designed in such way that each station has its own table. This web application is oriented towards quite a small database on a global scale usage in general. Correspondingly, the current database design corresponds to the needs of a small database.

## 3.3 Testing

Testing the application was a large part of the developing process. To create the stable application, it is important to detect the application's errors in advance. Some errors already existed, for example in the raw data (Fig. 24) sources themselves and this caused inconvenience.


16_fluoro - N...
File Edit Format View Help
16_lampo - N...
File Edit Format View Help
16_fluoro - Notepad
File Edit Format View Help

```
09.04.2013;21:00;340,036
09.04.2013;22:00;340,036
09.04.2013;23:00;340,036
10.04.2013;00:00;340,036
10.04.2013;01:00;340,036
10.04.2013;02:00;340,036
10.04.2013;03:00;340,036
10.04.2013;04:00;340,036
10.04.2013;05:00;340,036
10.04.2013;06:00;340,036
10.04.2013;07:00;340,036
10.04.2013;08:00;340,036
10.04.2013;09:00;340,036
10.04.2013;10:00;340,036
10.04.2013;11:00;340,036
10.04.2013;12:00;0,166
10.04.2013;13:00;0,166
10.04.2013;14:00;0,166
10.04.2013;15:00;0,166
10.04.2013;16:00;0,166
10.04.2013;17:00;0,166
10.04.2013;18:00;0,166
```

```
09.04.2013;21:00;60,00
09.04.2013;22:00;60,00
09.04.2013;23:00;60,00
10.04.2013;00:00;60,00
10.04.2013;01:00;60,00
10.04.2013;02:00;60,00
10.04.2013;03:00;60,00
10.04.2013;04:00;60,00
10.04.2013;05:00;60,00
10.04.2013;06:00;60,00
10.04.2013;07:00;60,00
10.04.2013;08:00;60,00
10.04.2013;09:00;60,00
10.04.2013;10:00;60,00
10.04.2013;11:00;16,13
10.04.2013;13:00;14,54
10.04.2013;14:00;15,17
10.04.2013;15:00;15,23
10.04.2013;16:00;14,56
10.04.2013;17:00;15,32
10.04.2013;18:00;15,32
10.04.2013;19:00;15,32
```

```
14.06.2011;01:00;0,1
14.06.2011;02:00;0,1
14.06.2011;03:00;0,1
14.06.2011;04:00;0,1
14.06.2011;05:00;0,1
14.06.2011;06:00;0,1
14.06.2011;07:00;0,1
14.06.2011;08:00;0,1
14.06.2011;09:00;0,1
14.06.2011;10:00;0,1
14.06.2011;11:00;0,1
14.06.2011;12:00;0,1
14.06.2011;13:00;0,2
14.06.2011;14:00;0,2
14.06.2011;15:00;0,2
14.06.2011;16:00;0,2
14.06.2011;17:00;0,2
14.06.2011;18:0000,2
14.06.2011;19:00;0,2
14.06.2011;20:00;0,2
14.06.2011;21:00;0,3
14.06.2011;22:00;0,3
14.06.2011;23:00;0,3
15.06.2011;00:00;0,3
```

Figure 24. Examples of errors which should be detected for correct work.

In this solved case different timestamps could cause shifting and not correct insertion to the database. Accordingly, it is important to have both parameters matching the same timestamps to be shown also in the chart. If there was difference in timestamp or one of the parameters had no values at some timestamp, then the second parameter would be hidden on the chart as well. In addition, time-consuming tests for application performance and security were handled and analyzed during the project which helped to find sensible solutions.

Many modern technologies were involved in the project. For instance, lightweight JSON which “is a human readable method of storing arrays and objects with values as strings. It is used primarily for data transfer and is a lot less verbose than some of the other options such as XML. Commonly, it is used when the front-end part of your application requires some data from the back-end without a page reload. This is commonly achieved using JavaScript with an AJAX” (Rees 2016, 22).

## 3.4 Final analysis

The application was created in the PHP language with MySQL relational database on the versatile Laravel framework. The final choice included solutions, such as powerful Dygraph.js and DataTables.js libraries. During the development time, research faced many web development topics and forced to find appropriate solutions for them. Emphasis was placed on performance and security to avoid vulnerabilities.

All things considered, the results of accurate completed work match planned objectives of the project. All the main requirements have been implemented through information research and goals analysis. Similarly, this work was very challenging since equivalent scale work experience had not been done previously and its achievement relied on one person.

Overall, the web application fulfilled all the requirements and achieved its goals. During the development all kind of knowledge from university studies to internships and work placements contributed support in resolving project issues.

# 4 CONCLUSION

In conclusion, this thesis demonstrated the aspects of development process and goals realization. The main objective was the development of web database application with the functionality of visualization the data in the charts. Implementation was achieved through research, testing and analysis of the solutions. Results reflect the functional web database application which will be used for the research group purposes. Database security and performance were carefully considered. While at the same time, the limitations of the chosen tools were insignificant. Many challenges such as selection of better methods and solving pop-up issues were faced during the development. Proper results finding relied on previous knowledge from the studies. For instance, skills acquired from university courses, such as Databases and Web Development contributed to the principal work. In addition, experience from internships played a crucial role because they provided unique experience not only about the IT field but also about team work as well as working independently.

Above all the most important experience was achieved during the thesis work. Finally, the results are used for research group purposes. This work could be further developed in the direction of database expansion with the introduction of new database stations or data types.

# LIST OF REFERENCES

Cimo, F. 2015. jQuery Programming Cookbook Hot Recipes for jQuery Development.

Kazankov, V. 2017. WordPress Vs Laravel. Consulted 11.2017. https://belitsoft.com/laravel-development-services/cms-or-framework-wordpress-or-laravel

Kromann, F. 2016. PHP and MySQL Recipes. A Problem-Solution Approach Second Edition.

Malhotra, M. 2018. Why Laravel Is the Best PHP Framework in 2018. Consulted 01.2018. https://www.valuecoders.com/blog/technology-and-apps/laravel-best-php-framework-2017

PHP Documentation. Consulted 2018. http://php.net/docs.php

Rathore, D. Consulted 11.2017. https://www.dunebook.com/5-best-security-tips-for-a-laravel-application/

Rees, D. 2016. Laravel: Code Smart. The Laravel Framework Version 5 For Beginners.

Stauffer, M. 2017. Laravel Up & Running. A Framework for Building Modern PHP Apps.

Tezer, O. 2014. A Comparison of Relational Database Management Systems. Consulted 11.2017. https://www.digitalocean.com/community/tutorials/sqlite-vs-mysql-vs-postgresql-a-comparison-of-relational-database-management-systems#mysql

Turku University of Applied Sciences (TUAS) Water Engineering. Consulted 04.2018. https://www.tuas.fi/en/research-and-development/research-groups/water-engineering/

Turku University of Applied Sciences (TUAS) Water Engineering. Consulted 03.2018. https://www.tuas.fi/media-en/filer_public/2016/02/24/water_engineering.pdf

11 Best PHP Frameworks for Modern Web Developers in 2018. Consulted 01.2018. https://coderseye.com/best-php-frameworks-for-web-developers/