University of Zurich^UZH

ASSIGNMENT – Communication Systems Group, Prof. Dr. Burkhard Stiller

CoMaDa Extension Addressing Transparency Request for Data Owners

Michael Balmer
Suhr, Switzerland
Student ID: 12-923-363

Supervisor: Corinna Schmitt, Sina Rafati
Date of Submission: July 4, 2017

University of Zurich
Department of Informatics (IFI)
Binzmühlestrasse 14, CH-8050 Zürich, Switzerland

Assignment
Communication Systems Group (CSG)
Department of Informatics (IFI)
University of Zurich
Binzmühlestrasse 14, CH-8050 Zürich, Switzerland
URL: http://www.csg.uzh.ch/

Abstract

This work handles the development of a user-friendly user interface the CoMaDa (Configuration, Management, and Data handling) framework used for wireless sensor network (WSN) administration addressing the transparancy request of the collected data within a sensornetwork, without using databaseaccess as before. The development takes place in the Java framework CoMaDa, which contains a graphical user interface for networkmanagement, configuration and visualization of data in a sensor network. The new request interface is implemented fitting into the existing user interface. The new graphical user interface (GUI) includes multiple filtering options like WSN User, return option(pull / push) and date. The request is returned in a table with the set filter options and the sensor data. Sensor data includes the sensor name as well as the values. To ensure the data representation mostly HTML and Javascript is used. After the request the user has the possibility to print the request directly or save it as a PDF-File. The evaluation of the implementation is done as a proof of operability to proof the requested filtering functionality.

Zusammenfassung

Der Inhalt der Arbeit umfasst die Entwicklung einer benutzerfreundlichen Oberfläche auf CoMaDa (Configuration, Management, and Data handling) zur transparenten Abfrage von gesammelten Daten in einem Sensornetzwerk. Dies soll einem WSN owner ohne wie anhin nur mit direktem Datenbankzugriff möglich sein. Die neue Abfrageoberfläche wurde passend in die bestehende Benutzeroberfläche in das Java framework eingebettet. Die neue Oberfläche beinhaltet mehrere Filteroptionen, wie WSN Name, Benutzer, Rueckgabeoptionen(Push/Pull) und das Datum. Die Rückgabe erfolgt in einer Tabelle mit den gewählten Filteroptionen und den Sensordaten. Zu den Sensordaten gehören der Sensorname sowie deren Werte. Um diese Datenrepräsentation zu gewaehrleisten wird hauptsächlich HTML und Javascript verwendet. Nach der Abfrage besteht die Möglichkeit, diese Daten im PDF-Format speichern zu lassen. Die Auswertung der Implementation wurde anhand eines Proof of Operability auf Funktionalität getestet.

Acknowledgments

First, I would like to sincerily thank my supervisor Dr. Corinna Schmitt for her support, time and endurance and her valuable and motivating inputs and comments during the assignment. I would also like to thank Prof. Dr. Burkhard Stiller for his support and the possibility to complete this assignment at the Communications Systems Group at the Department of Informatics of the University of Zurich.

Contents

Chapter 1

Introduction

1.1 Motivation

Due to the growth of the Internet and the device diversity together with their communication capability the Internet of Things (IoT) is a hot topic. The IoT is not limited to Peer-to-Peer (P2P) networks and devices like server, computers, and routers any more. It also includes wireless sensor devices connectd in a Wireless Sensr Network (WSN). [1]

The application range goes from intelligent homes, logistic, health care to environmental monitoring. All applications have in common a huge amount of collected sensor data (e.g., temperature, brightness, humidity) under different operating systems. As the collected data can also be sensitive or might only be accessed by authorized persons. It is important for a data owner to specifically know the data flow, especially who accessed which data and at what time. [6]

The goal of this assignment is to develop an extention feature for the existing framework SecureWSN, which was developed to manage and configure WSNs using different security algorithms for communication, data processing and accessing possibilities, and visualisation solutions. Inside the SecureWSN framework exists CoMaDa which represents the server side of the network. It shows the data flow within the interface and allows hardware configuration, management of network components, data storage as well as the visualization of the data [2]. The drawback in the current setup in CoMaDa is that the data owner can only see his collected data and granted rights to authorized users and only the database administrator can access the database where all the collected data is stored. Therefore, the contribution of this assignemnt is to allow the data owner access to a user-friendly view with filtering options to display who, when and what data was collected for all his users.

1.2 Description of Work

The work for this assignemnt is to develop a user-friendly view for CoMaDa with multiple filtering options allowing the data owner to keep track of his data, especially who accessed the data and when. Therefore, in a first step, the existing database using PostGreSQL [9] needs to be analysed on how the data is stored and is linked together for each network. In a second phase, the question how the database can be accessed in a secure manner is worked on. This includes how the data can be extracted and ported to the view in CoMaDa to display the information to the data owner as well as the handling of a secure access to the view only for the data owner. In a next step the implementation takes place. This includes the integration of the access and data transfer solution as well as the implementation of a user-friendly GUI with filtering options like date and user. In a final step, the implementation is evaluated concerning the performance as well as based on a proof of operability.

1.3 Thesis Outline

The rest of the assignment report is structured as follows. Chapter 2 includes related work which correlates with this assignment and gives a brief overview of the works. Chapter 3 contains the design decisions concerning access security and the user-friendly GUI. In chapter 4 the process of the implementation is shown. Chapter 5 presents the evaluation of the implementation. It includes a short evaluation on performance and a more thorough proof of operability. The conclusion is drawn in the final Chapter 6.

Chapter 2

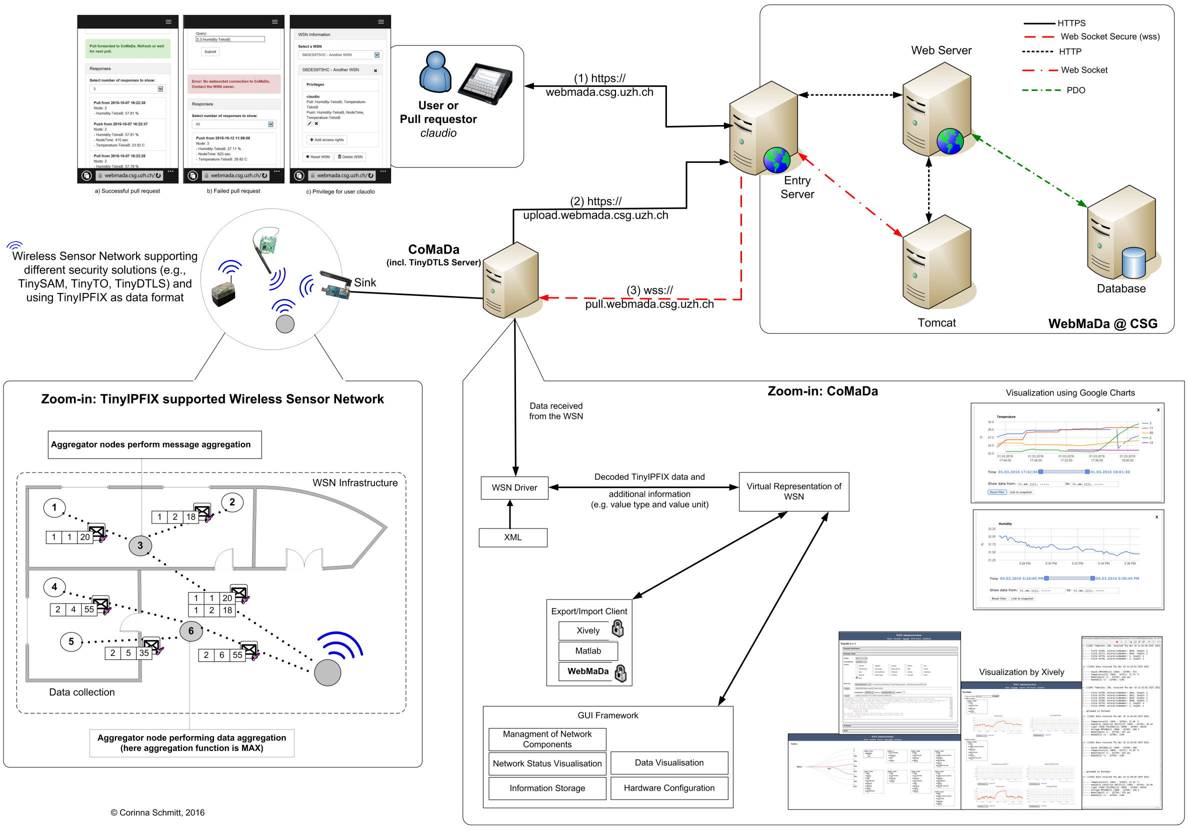
Related Work

This chapter goes deeper into the relevant topics for this work. Therefore, a closer look is taken at the framework SecureWSN on which this work builds on. The focus lies on CoMaDa for the backend integration including the database communication and on the otherhand the front end integration for displaying data in the GUI. A quick look is taken at the communication between CoMaDa and WebMaDa mainly for authentication reasons and their relationship.

2.1 SecureWSN - framework

The goal in the beginning of the project SecureWSN [2] was to develop different solutions for secure and efficient data transmission in wireless sensor networks. In a later stage a GUI was developed, which would allow the user to configure the WSN in a handsome manner. Therefore, the Configuration, Management and Data Handling Framework (CoMaDa) was designed which is one of the main components. The other main component is the Web-based Mobile Access and Data Handling Framework (WebMaDa) allowing users to deploy their WSN data online [10].

The figure above illustrates the cooperation between all components in the established SecureWSN (status 2015)[2]



© Corinna Schmitt, 2016

Figure 2.1: Cooperation between all components in the SecureWSN [2]

2.1.1 CoMaDa

The framework for Configuration, Management and Data Handling (CoMaDa) was built with a user-friendly GUI to simplify the operations with wireless sensor networks. The WSNDataFramework [2] is written in Java and allows the configuration and communication between nodes in a WSN. This is the main task of the back end of CoMaDa. In a newer version there is also a PostGreSQL-Database [9] integrated in CoMaDa to display the graphical visualization of sensor data from Tim Strasser [8] offline. Both works are important for this assignment. The new database deployment on CoMaDa allows a simplified access to the data on the database as the interface is also implemented for a graphical visualization. It leads to the other part of CoMaDa the frontend and its GUI. CoMaDa is built in Modules so is the frontend, which are mainly HTML, Javascript-Files. Those communicate through a WebServer integrated in the backend with the rest of the system.

2.1.2 WebMaDa

Web-based Mobile Access and Data Handling Framework WebMaDa is an extension for CoMaDa to bring mobility support to the CoMaDa framework. It consists of the following four components: An Online Database written in MySQL which contains tables including access rights, active WSNs, and corresponding data including datastream, topology, and raw data. With the management tool in WebMaDa a user is able to set the access right of different users according the needs. Therefore, three tables in the database manage these rules. This module requires successful authentication of the account and then pushes all monitored data to the online database storage that is linked to the web site. In the backend of the system tables are created that log the deployed WSNs and display them to the user on the website. According to the access rights, a user must first authenticate himself on the website before he can visualize data similarly to CoMaDa because depending on the access rights the user is able to see currently active WSNs or not. [10]

2.2 Research question

Developing a user-friendly extension for CoMaDa with different filtering options to receive data about the usage of his own WSN. It is important to make sure the connection to the database is secure as well as to ensure that only a rightful owner of a WSN can request certain data. To make sure the user can use the requested data, a printing or saving option is a must as well as ordering the data as wished.

2.3 Findings

The most essential part for this work lies within the CoMaDa framework. The implementation takes place in its back- and frontend. Especially the work of Christian Ott [9] for an more simple accesss to the database which is a great part of this assignment to ensure the database requests for the filtering options. The link to WebMaDa can be found within the managment tool and its data within the database. Therefore with an addtional credential check the rightful owner of mulitiple WSNs can be determined and the correct data can be displayed. The printing/ saving option can be compared the export/import option within CoMaDa.

Chapter 3

Design Decision

In this chapter the design choices for this assignment will be clarified. This includes three primary steps. First the secure access to the filtering view and the database connection will be discussed. The second step handles the architectural design decision for the new GUI view. In a last part multiple options for saving the filtered data will be discussed.

3.1 Accesss on view 'access control'

The current existing solution of CoMaDa assumes that only the WSN owner has access to it. This setting is similar to a locked room where only authorized personal have access to. But usually the reality looks different. Thus, an additional authorization check must be integrated. The answer to this question lies within the filtering option within the new view. A WSN-owner has to identify himself to gain access to all sensor networks he owns which is not recognisable just with the access control into CoMaDa.

3.2 Filtering

During the development of the structural design of the new view. Multiple filtering options emerged. Starting with the WSN-Name, a WSN-Owner can choose between one or multiple WSNs he owns to filter data. The second option includes the WSN User which one of the main goal of this assignment to reach transparency for the data owner. This amount of WSN users is variable according to the selected WSN-Names. Therefore no user can be selected who is not part of a sensor network, like in the previous option one or multiple WSN-Users can be selected. The third option includes the action in a sensor network. Thus a data owner can choose between Push and Pull. The decision for the final filter option concluded in the Date option as it is highly required to make a reasonable request. Its importance is underlined by all other visualizations as a timestamp or date filter option like in "charts" cannot miss. Therefore a start date earlier than the enddate, which can be chosen as the actual clock time has to be set. After setting up the filter

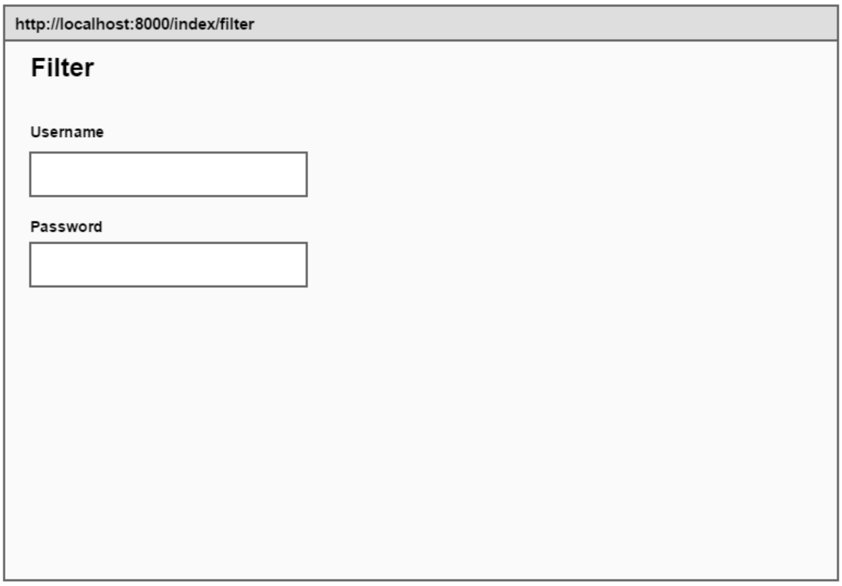


Figure 3.1: Mockup for the login screen

options, the requested data is shown in a table, where it is possible to order the data according to any filter option given.

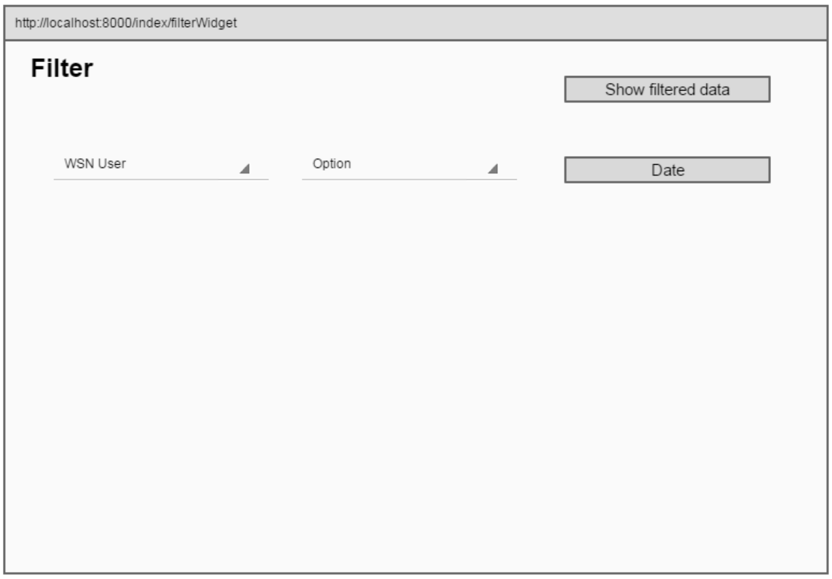
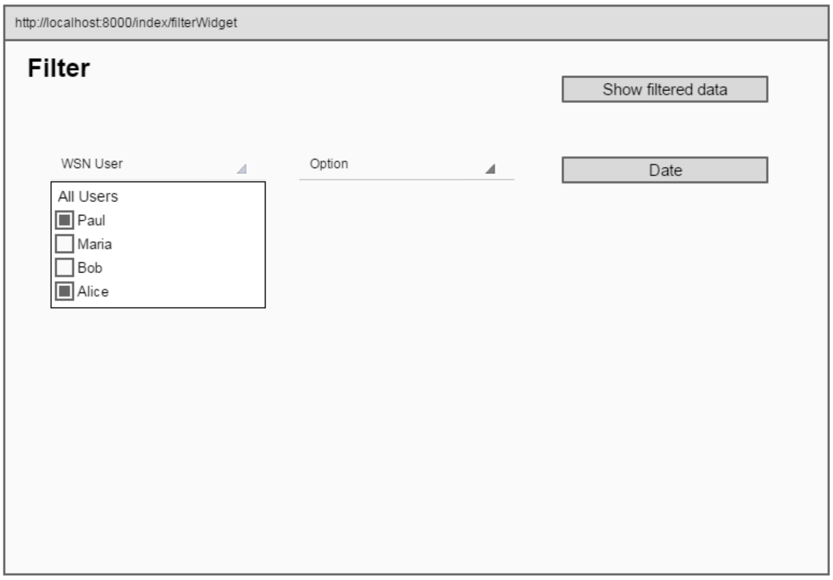


Figure 3.2: Mockup with available filter options



All Users

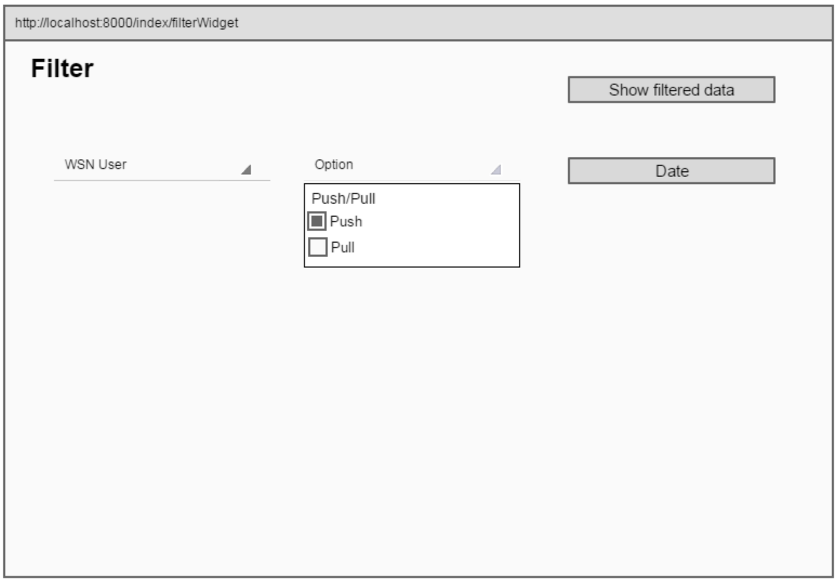
☒ Paul

□ Maria

□ Bob

☒ Alice

Figure 3.3: Mockup for WSN-User dropdown menu

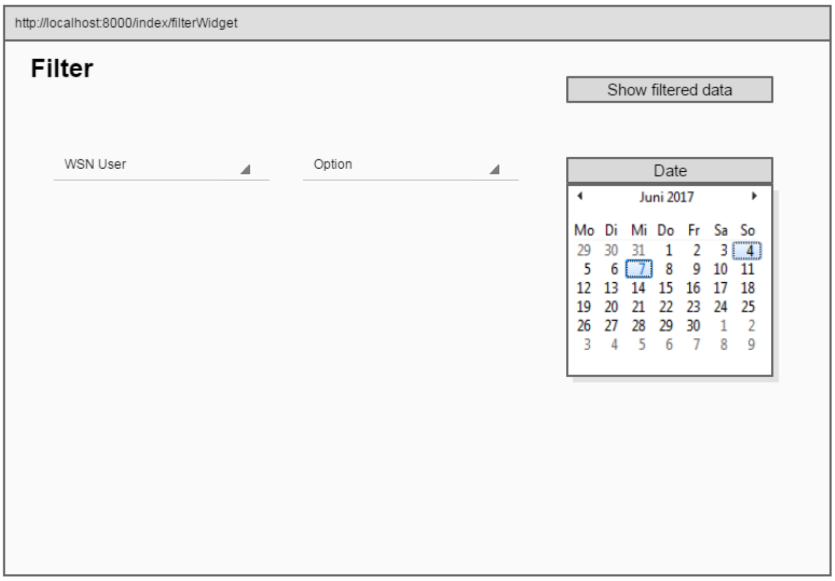


Push/Pull

☒ Push

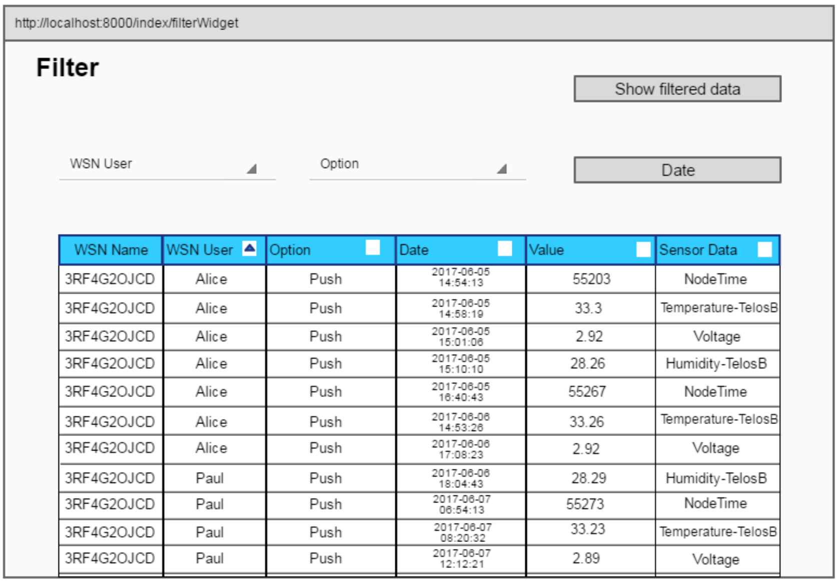
□ Pull

Figure 3.4: Mockup for the request options



Mo	Di	Mi	Do	Fr	Sa	So
29	30	31	1	2	3	4
5	6	7	8	9	10	11
12	13	14	15	16	17	18
19	20	21	22	23	24	25
26	27	28	29	30	1	2
3	4	5	6	7	8	9

Figure 3.5: Mockup for the date filter



WSN Name	WSN User	Option	Date	Value	Sensor Data
3RF4G2OJCD	Alice	Push	2017-06-05 14:54:13	55203	NodeTime
3RF4G2OJCD	Alice	Push	2017-06-05 14:58:19	33.3	Temperature-TelosB
3RF4G2OJCD	Alice	Push	2017-06-05 15:01:06	2.92	Voltage
3RF4G2OJCD	Alice	Push	2017-06-05 15:10:10	28.26	Humidity-TelosB
3RF4G2OJCD	Alice	Push	2017-06-05 16:40:43	55267	NodeTime
3RF4G2OJCD	Alice	Push	2017-06-06 14:53:26	33.26	Temperature-TelosB
3RF4G2OJCD	Alice	Push	2017-06-06 17:08:23	2.92	Voltage
3RF4G2OJCD	Paul	Push	2017-06-06 18:04:43	28.29	Humidity-TelosB
3RF4G2OJCD	Paul	Push	2017-06-07 06:54:13	55273	NodeTime
3RF4G2OJCD	Paul	Push	2017-06-07 08:20:32	33.23	Temperature-TelosB
3RF4G2OJCD	Paul	Push	2017-06-07 12:12:21	2.89	Voltage

Figure 3.6: Mockup for the requested table

3.3 Saving filtered Data for printing

After successfully displaying the selected data a data owner should be able to save his request. Therefore, two options are at hand. First an immediate print out and secondly the creation of a PDF file. The question has been raised how name the files as well as tag them inside the file. Not many options came into conclusion. Therefore, the decision fell for WSN Name_Date_time because it can be easily sorted within a directory. It is therefore possible to chronologically order the files by date_time. Inside the file the header also includes the WSN_Name, the selected WSN users and date_time. This allows for an easy overview of the selected filter options inside the file.

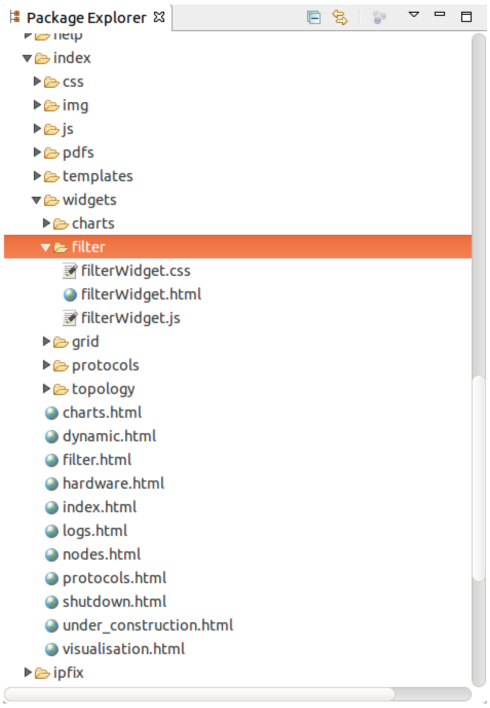
Chapter 4

Implementation

This chapter gives a deeper insight into the implementation. For this implementation a specific WSN was chosen to implement the filter options as the information about active WSNs lies in the WebMaDa database from which no data could be retrieved yet.

4.1 Front end prototype

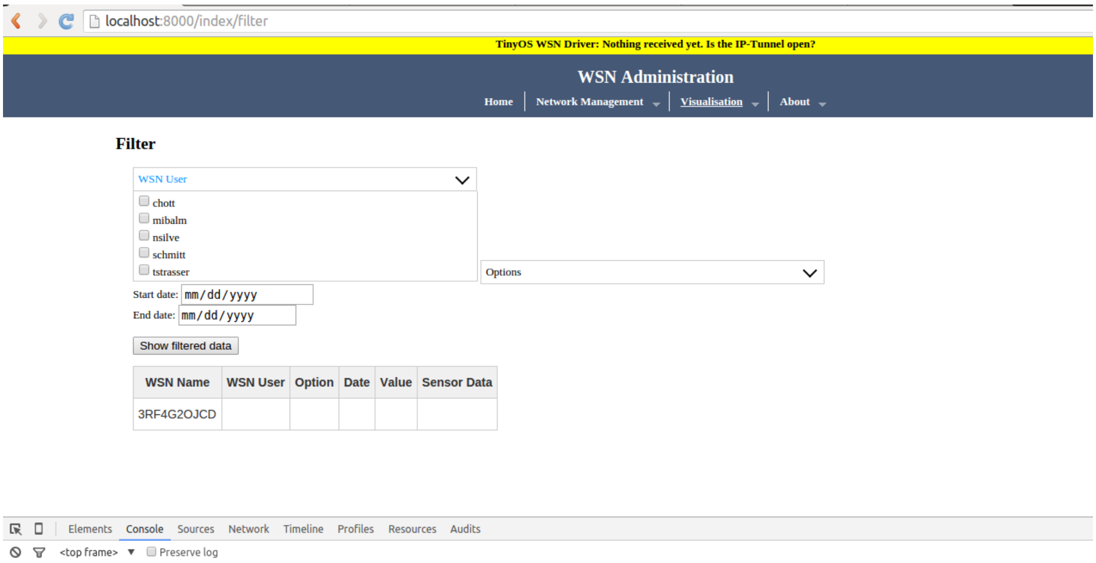
For the front end a prototype of the filter view is implemented. As shown in the figure 4.1 a new widget folder with a CSS, HTML, and Javascript file has been created.



```
▸ help
▾ index
  ▸ css
  ▸ img
  ▸ js
  ▸ pdfs
  ▸ templates
  ▾ widgets
    ▸ charts
    ▾ filter
        filterWidget.css
        filterWidget.html
        filterWidget.js
    ▸ grid
    ▸ protocols
    ▸ topology
    charts.html
    dynamic.html
    filter.html
    hardware.html
    index.html
    logs.html
    nodes.html
    protocols.html
    shutdown.html
    under_construction.html
    visualisation.html
▸ ipfix
```

Figure 4.1: Filter widget folder

In a first step the HTML file has been modified to yield all filter options in a basic layout with dummy data to get an overview on the containers to use. In figure 4.2 the first prototype of the GUI can be seen. Therefore, containers for selecting the WSN users, the pull and push option as well as a calender for the start- and enddate.



TinyOS WSN Driver: Nothing received yet. Is the IP-Tunnel open?

WSN Administration

Filter

WSN User

□ chott
□ mibalm
□ nsilve
□ schmitt
□ tstrasser

Options

Start date: mm/dd/yyyy
End date: mm/dd/yyyy

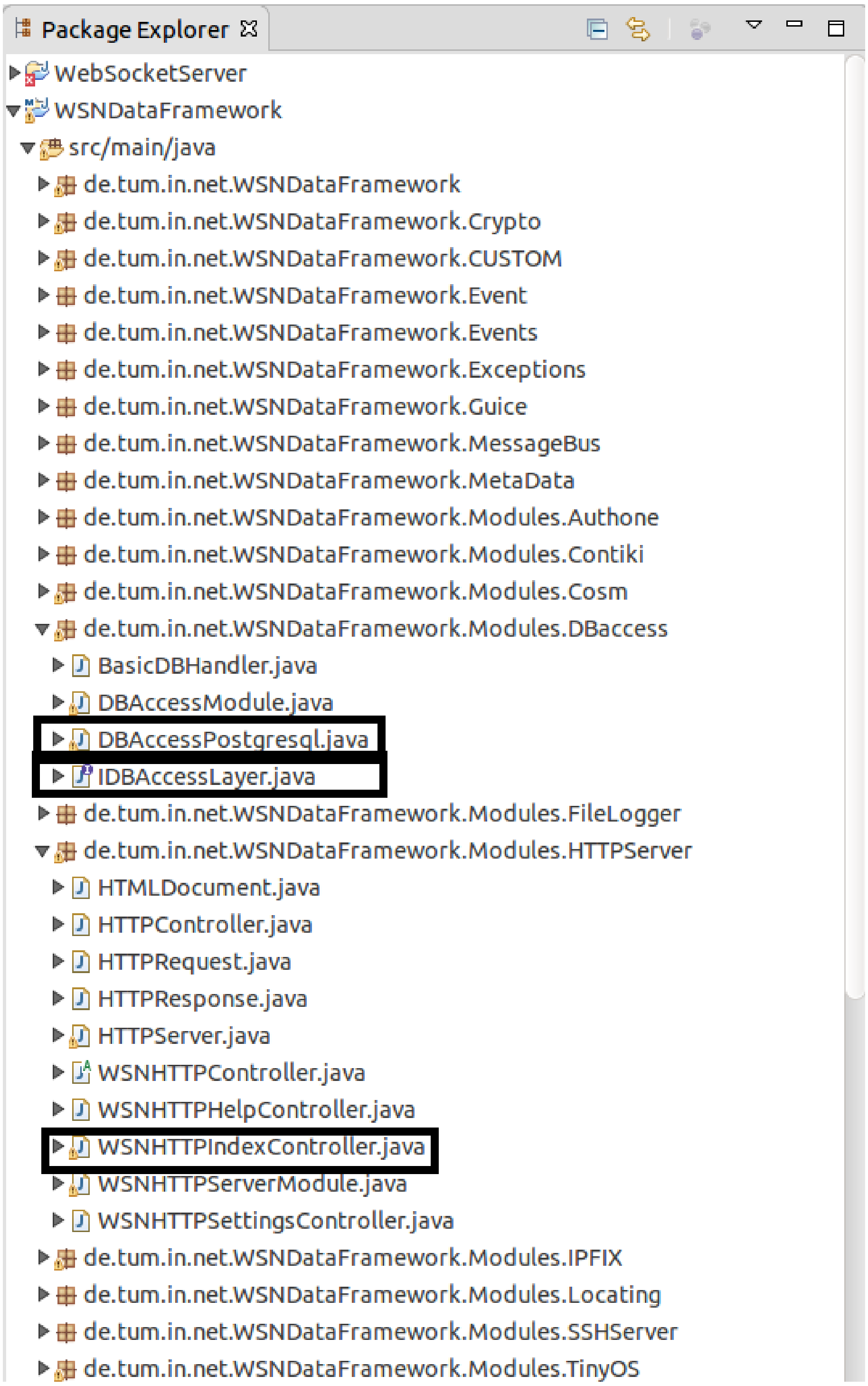
WSN Name	WSN User	Option	Date	Value	Sensor Data
3RF4G2OJCD					

Figure 4.2: Overview filter page

After the implementation of the GUI Elements, it was necessary to fill up the WSN user container as well as the date container with data from the database to ensure only available options can be selected.

4.2 Representing first data in front end

To access the database and forward the data to the FilterWidget javascript methods in multiple existing modules had to be integrated. Figure 4.3 highlights the three modules in the package explorer.

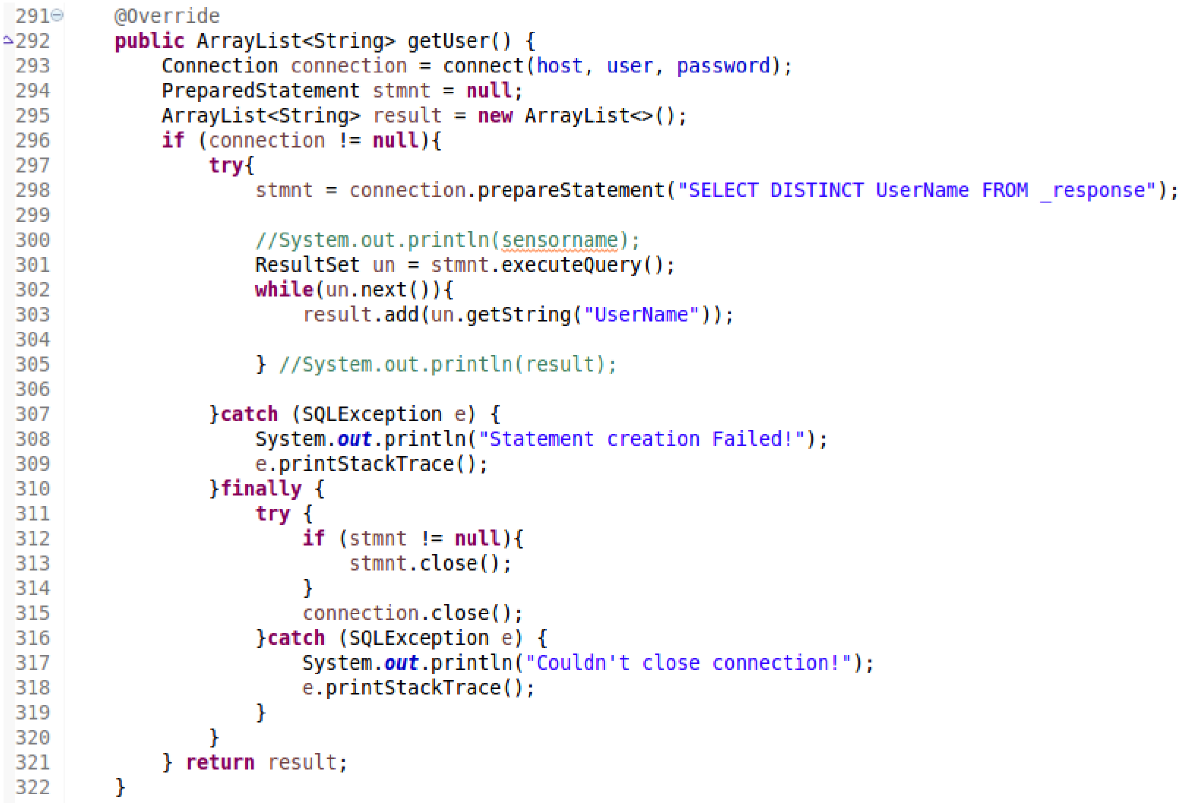


Package Explorer

```
▶ WebSocketServer
▼ WSNDataFramework
  ▼ src/main/java
    ▶ de.tum.in.net.WSNDataFramework
    ▶ de.tum.in.net.WSNDataFramework.Crypto
    ▶ de.tum.in.net.WSNDataFramework.CUSTOM
    ▶ de.tum.in.net.WSNDataFramework.Event
    ▶ de.tum.in.net.WSNDataFramework.Events
    ▶ de.tum.in.net.WSNDataFramework.Exceptions
    ▶ de.tum.in.net.WSNDataFramework.Guice
    ▶ de.tum.in.net.WSNDataFramework.MessageBus
    ▶ de.tum.in.net.WSNDataFramework.MetaData
    ▶ de.tum.in.net.WSNDataFramework.Modules.Authone
    ▶ de.tum.in.net.WSNDataFramework.Modules.Contiki
    ▶ de.tum.in.net.WSNDataFramework.Modules.Cosm
    ▼ de.tum.in.net.WSNDataFramework.Modules.DBaccess
      ▶ BasicDBHandler.java
      ▶ DBAccessModule.java
      ▶ DBAccessPostgresql.java
      ▶ IDBAccessLayer.java
    ▶ de.tum.in.net.WSNDataFramework.Modules.FileLogger
    ▼ de.tum.in.net.WSNDataFramework.Modules.HTTPServer
      ▶ HTMLDocument.java
      ▶ HTTPController.java
      ▶ HTTPRequest.java
      ▶ HTTPResponse.java
      ▶ HTTPServer.java
      ▶ WSNHTTPController.java
      ▶ WSNHTTPHelpController.java
      ▶ WSNHTTPIndexController.java
      ▶ WSNHTTPServerModule.java
      ▶ WSNHTTPSettingsController.java
    ▶ de.tum.in.net.WSNDataFramework.Modules.IPFIX
    ▶ de.tum.in.net.WSNDataFramework.Modules.Locating
    ▶ de.tum.in.net.WSNDataFramework.Modules.SSHServer
    ▶ de.tum.in.net.WSNDataFramework.Modules.TinyOS
```

Figure 4.3: Package explorer with back end files

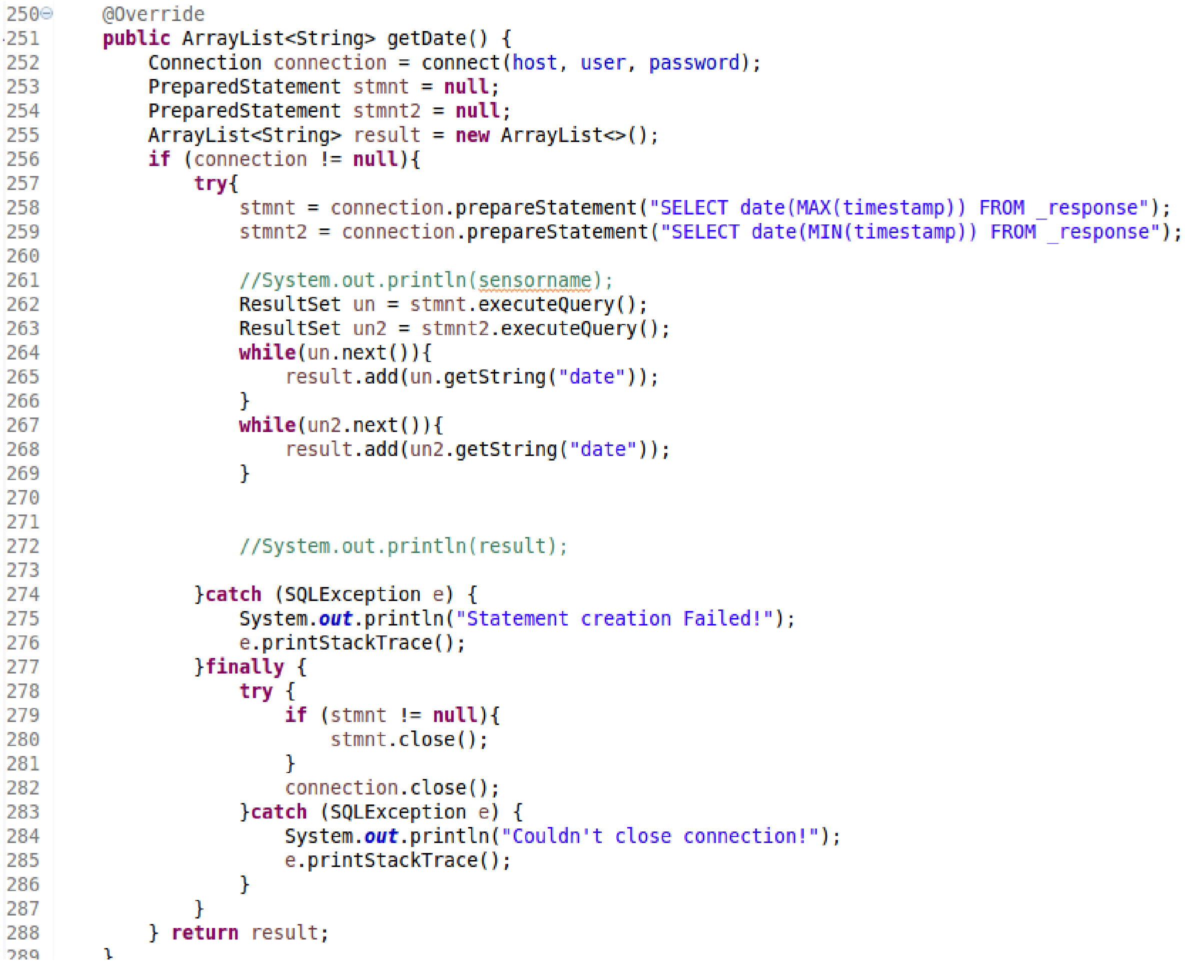
The highest level to manipulate the data is inside the DBAccessPostgresql.java file. Therefore, the two methods in figure 4.4 and 4.5 have been created.



```
291     @Override
292     public ArrayList<String> getUser() {
293         Connection connection = connect(host, user, password);
294         PreparedStatement stmnt = null;
295         ArrayList<String> result = new ArrayList<>();
296         if (connection != null){
297             try{
298                 stmnt = connection.prepareStatement("SELECT DISTINCT UserName FROM _response");
299 
300                 //System.out.println(sensorname);
301                 ResultSet un = stmnt.executeQuery();
302                 while(un.next()){
303                     result.add(un.getString("UserName"));
304 
305                 } //System.out.println(result);
306 
307             }catch (SQLException e) {
308                 System.out.println("Statement creation Failed!");
309                 e.printStackTrace();
310             }finally {
311                 try {
312                     if (stmnt != null){
313                         stmnt.close();
314                     }
315                     connection.close();
316                 }catch (SQLException e) {
317                     System.out.println("Couldn't close connection!");
318                     e.printStackTrace();
319                 }
320             }
321         } return result;
322     }
```

Figure 4.4: Code for getUser DBAccess

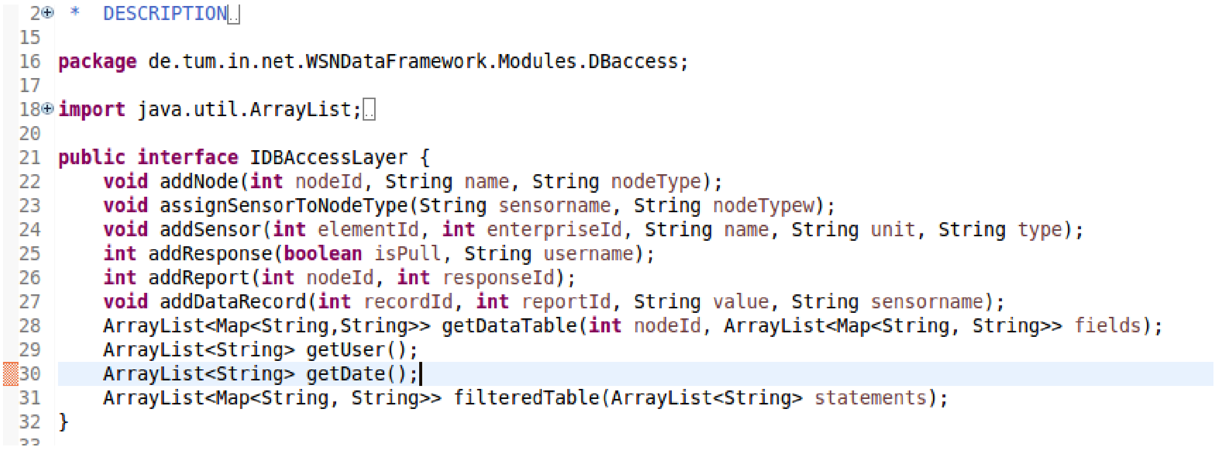
To enable the database access, a connection object has to be created. Together with a prepareStatement as it enables the writing of a query with JDBC. In figure 4.4 on line 298 together the query is formulated ad on line 301 it gets executed. The returning result is given back in a ResultSet, which is then processed row by row and saved in the return ArrayList<String> result. The result ArrayList<String> now yields all Wsn users who are stored in the database meaning who actively participated in using the WSN.



```java
    @Override
    public ArrayList<String> getDate() {
        Connection connection = connect(host, user, password);
        PreparedStatement stmnt = null;
        PreparedStatement stmnt2 = null;
        ArrayList<String> result = new ArrayList<>();
        if (connection != null){
            try{
                stmnt = connection.prepareStatement("SELECT date(MAX(timestamp)) FROM _response");
                stmnt2 = connection.prepareStatement("SELECT date(MIN(timestamp)) FROM _response");

                //System.out.println(sensorname);
                ResultSet un = stmnt.executeQuery();
                ResultSet un2 = stmnt2.executeQuery();
                while(un.next()){
                    result.add(un.getString("date"));
                }
                while(un2.next()){
                    result.add(un2.getString("date"));
                }


                //System.out.println(result);

            }catch (SQLException e) {
                System.out.println("Statement creation Failed!");
                e.printStackTrace();
            }finally {
                try {
                    if (stmnt != null){
                        stmnt.close();
                    }
                    connection.close();
                }catch (SQLException e) {
                    System.out.println("Couldn't close connection!");
                    e.printStackTrace();
                }
            }
        } return result;
    }
```

Figure 4.5: Code for getDate DBAccess

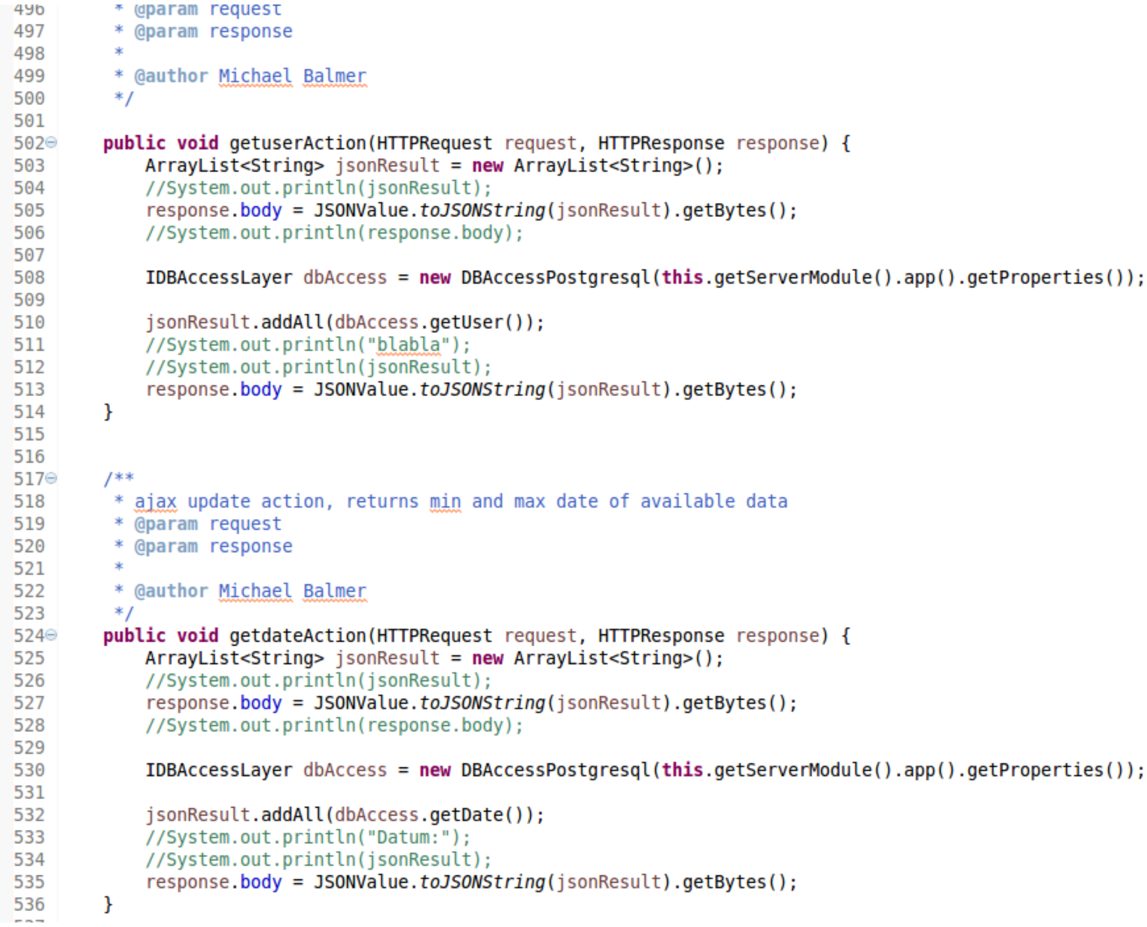
Similarly to the getUser() function the getDate() function has been implemented. The return value in this function is again a ArrayList<String> containing the max and min timestamp in the database. These values can be used as Start and Enddate in the date container on the filter page. The steps can be veryfied in figure 4.5. Finally to make these functions available for use in other modules. They need to be added to the IDBAccesss-Layer as shown in figure 4.6.



```
 *  DESCRIPTION

package de.tum.in.net.WSNDataFramework.Modules.DBaccess;

import java.util.ArrayList;

public interface IDBAccessLayer {
    void addNode(int nodeId, String name, String nodeType);
    void assignSensorToNodeType(String sensorname, String nodeTypew);
    void addSensor(int elementId, int enterpriseId, String name, String unit, String type);
    int addResponse(boolean isPull, String username);
    int addReport(int nodeId, int responseId);
    void addDataRecord(int recordId, int reportId, String value, String sensorname);
    ArrayList<Map<String,String>> getDataTable(int nodeId, ArrayList<Map<String, String>> fields);
    ArrayList<String> getUser();
    ArrayList<String> getDate();
    ArrayList<Map<String, String>> filteredTable(ArrayList<String> statements);
}
```

Figure 4.6: Code for IDBAccessLayer

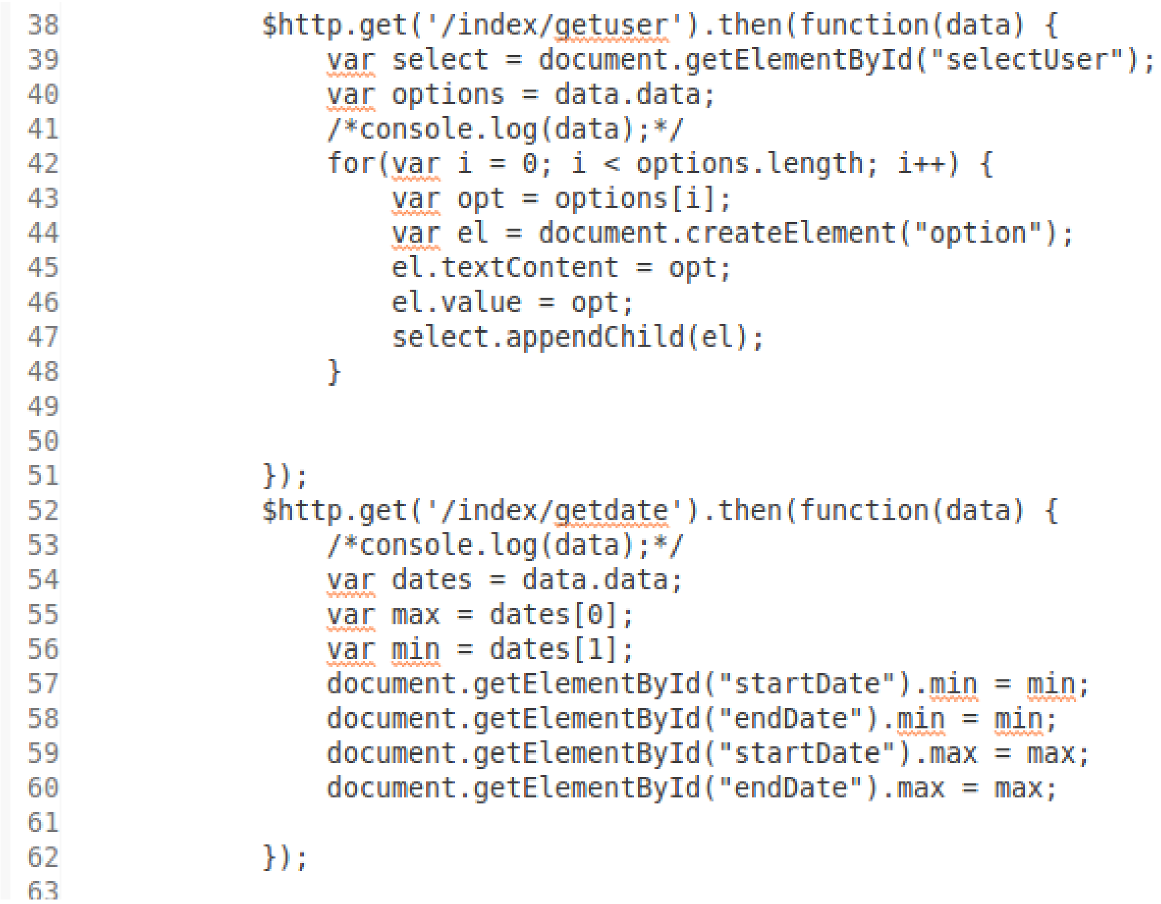
Now that the function is made available through the IDBAccessLayer. The implementation of the method in the WSNHTTPIndexController coud be done. The methods implemented can be seen in figure 4.7. These are mainly asynchronious http requests and responses, which are used to enable the communication to the front end specifically the filterWidget.js file. For example in the getuserAction method a dbAccess object is created containing our getUser() method. The return value of our getUser() method is then added to a ArrayList<String> called jsonResult, which is converted into a JSONString and added to the http response body.



```
     * @param request
     * @param response
     *
     * @author Michael Balmer
     */

    public void getuserAction(HTTPRequest request, HTTPResponse response) {
        ArrayList<String> jsonResult = new ArrayList<String>();
        //System.out.println(jsonResult);
        response.body = JSONValue.toJSONString(jsonResult).getBytes();
        //System.out.println(response.body);

        IDBAccessLayer dbAccess = new DBAccessPostgresql(this.getServerModule().app().getProperties());

        jsonResult.addAll(dbAccess.getUser());
        //System.out.println("blabla");
        //System.out.println(jsonResult);
        response.body = JSONValue.toJSONString(jsonResult).getBytes();
    }


    /**
     * ajax update action, returns min and max date of available data
     * @param request
     * @param response
     *
     * @author Michael Balmer
     */
    public void getdateAction(HTTPRequest request, HTTPResponse response) {
        ArrayList<String> jsonResult = new ArrayList<String>();
        //System.out.println(jsonResult);
        response.body = JSONValue.toJSONString(jsonResult).getBytes();
        //System.out.println(response.body);

        IDBAccessLayer dbAccess = new DBAccessPostgresql(this.getServerModule().app().getProperties());

        jsonResult.addAll(dbAccess.getDate());
        //System.out.println("Datum:");
        //System.out.println(jsonResult);
        response.body = JSONValue.toJSONString(jsonResult).getBytes();
    }
```

Figure 4.7: Code for AJAX calls

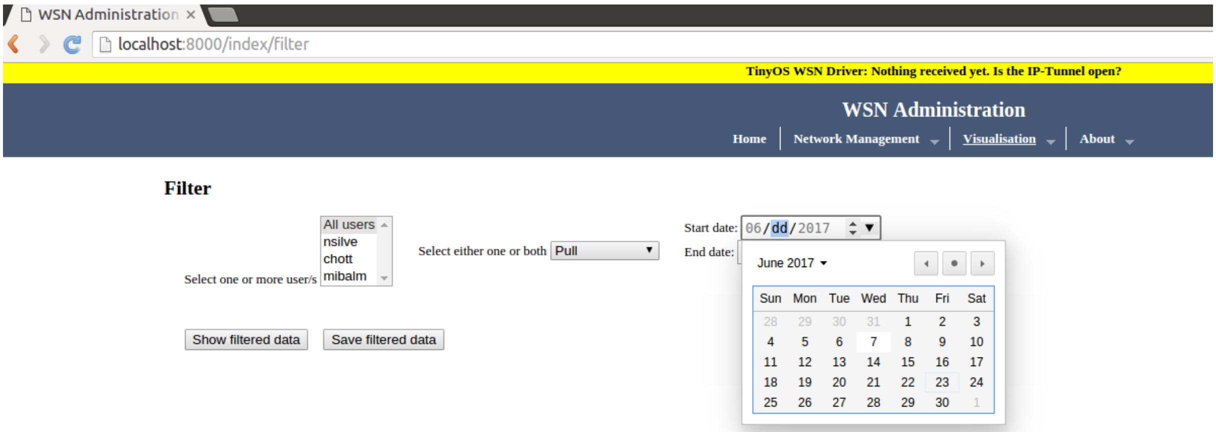
The final step for the representation of our values on the filterWidget.html page takes place in the filterWidget.js file.



```
38          $http.get('/index/getuser').then(function(data) {
39              var select = document.getElementById("selectUser");
40              var options = data.data;
41              /*console.log(data);*/
42              for(var i = 0; i < options.length; i++) {
43                  var opt = options[i];
44                  var el = document.createElement("option");
45                  el.textContent = opt;
46                  el.value = opt;
47                  select.appendChild(el);
48              }
49
50
51          });
52          $http.get('/index/getdate').then(function(data) {
53              /*console.log(data);*/
54              var dates = data.data;
55              var max = dates[0];
56              var min = dates[1];
57              document.getElementById("startDate").min = min;
58              document.getElementById("endDate").min = min;
59              document.getElementById("startDate").max = max;
60              document.getElementById("endDate").max = max;
61
62          });
63
```

Figure 4.8: Code in filterWidget javascript file

Figure 4.8 shows the implementation of the http.get() which is a call to the methods just recently added in the WSNHTTPIndexController. Namely the getuserAction on line 38 and the getdateAction method on line 52. Afterwards the received data is processed like in the getuser part, where the wsn users are iterated over and dynamically added to a multiselect option container in the html file.



TinyOS WSN Driver: Nothing received yet. Is the IP-Tunnel open?

Filter

Select one or more user/s: All users, nsilve, chott, mibalm

Select either one or both: Pull

Start date: 06/dd/2017

End date:

June 2017

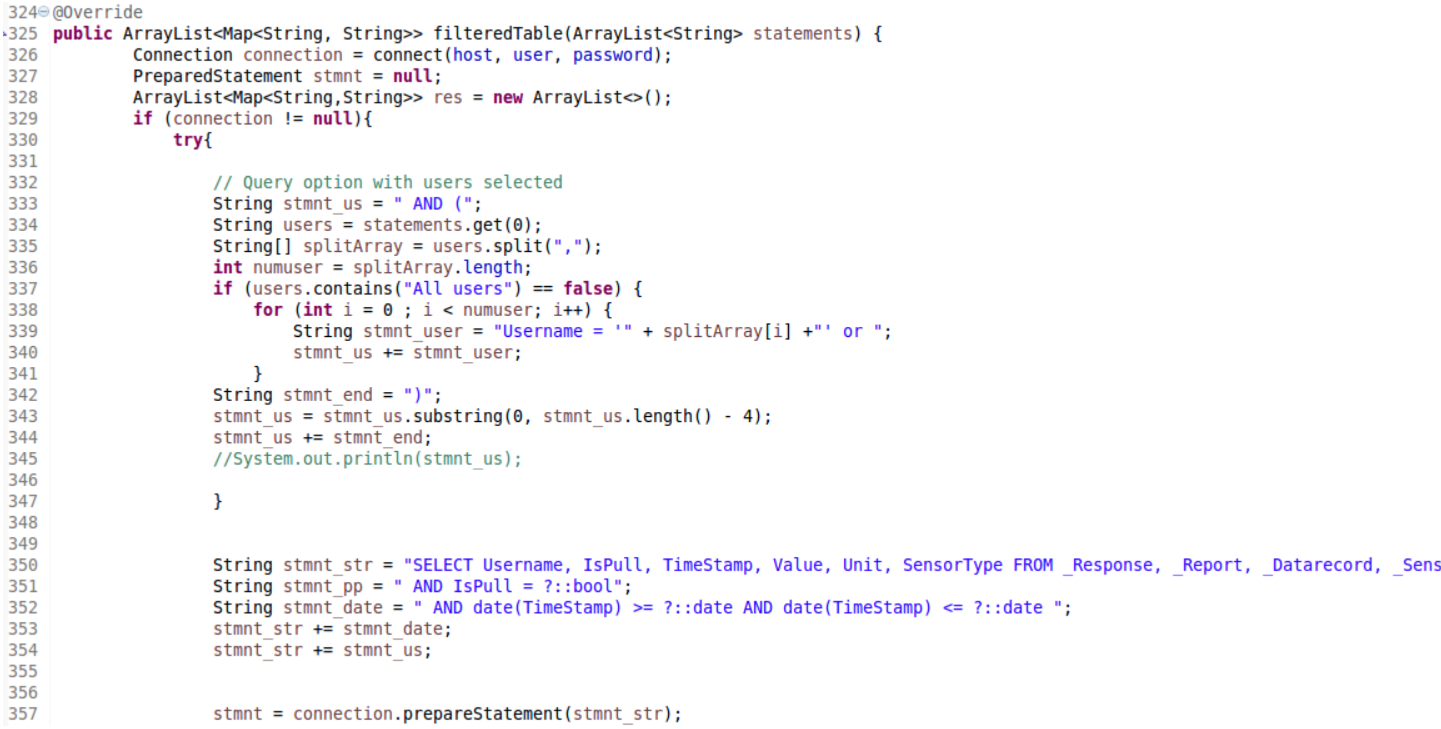
Sun	Mon	Tue	Wed	Thu	Fri	Sat
28	29	30	31	1	2	3
4	5	6	7	8	9	10
11	12	13	14	15	16	17
18	19	20	21	22	23	24
25	26	27	28	29	30	1

Figure 4.9: Preset of filter options

With this final step added we can see the result of our chain of actions in Figure 4.9. The user is able to select multiple WSN users which have been active in the WSN and within the date window, we can see that just one date is white which indicates the date with data activity.

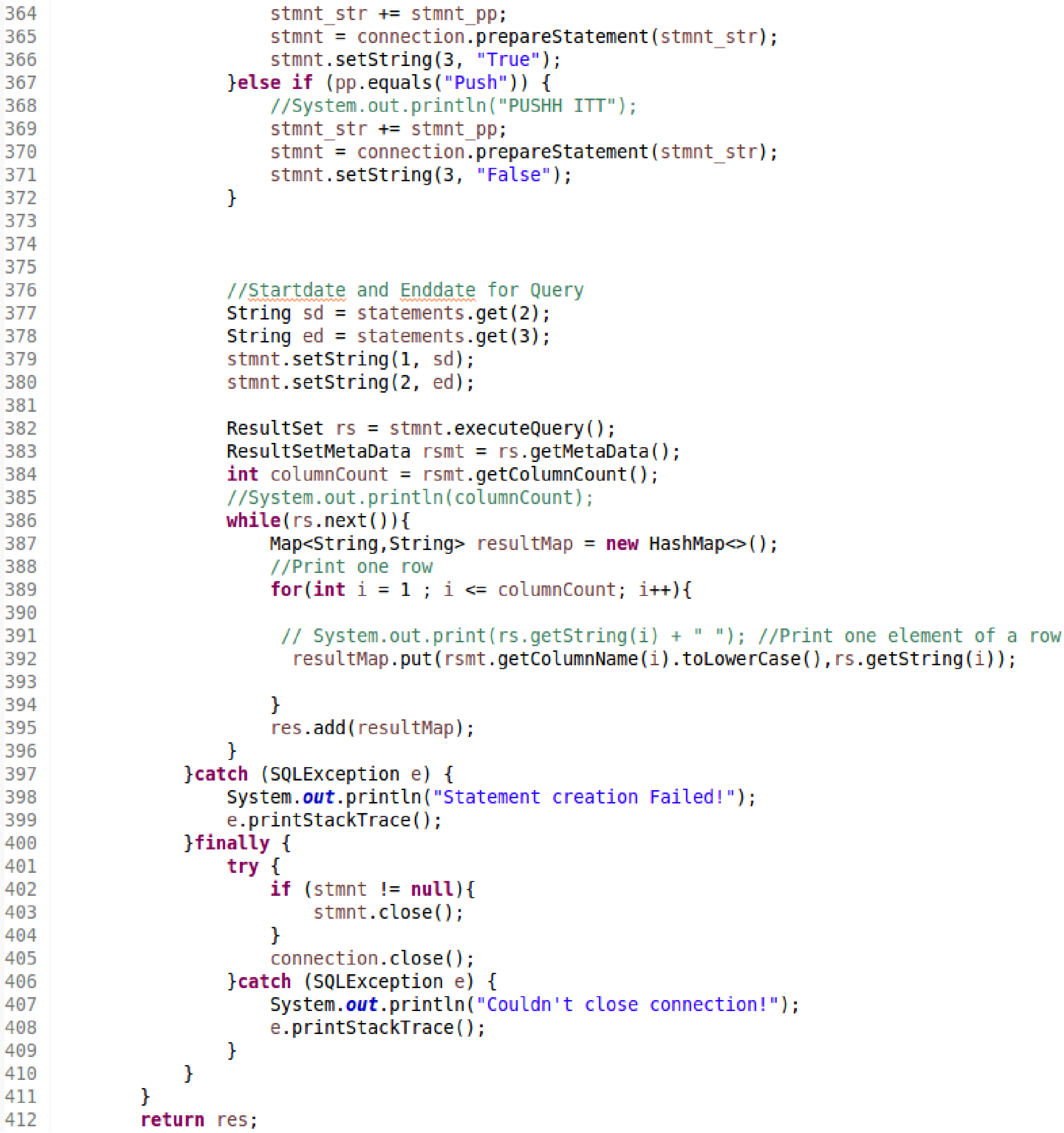
4.3 Integration of data table with filter options

In this section, the final implementation of the filtered data is described. This is done from backend to frontend like in the previous section.



```
@Override
public ArrayList<Map<String, String>> filteredTable(ArrayList<String> statements) {
        Connection connection = connect(host, user, password);
        PreparedStatement stmnt = null;
        ArrayList<Map<String,String>> res = new ArrayList<>();
        if (connection != null){
            try{

                // Query option with users selected
                String stmnt_us = " AND (";
                String users = statements.get(0);
                String[] splitArray = users.split(",");
                int numuser = splitArray.length;
                if (users.contains("All users") == false) {
                    for (int i = 0 ; i < numuser; i++) {
                        String stmnt_user = "Username = '" + splitArray[i] +"' or ";
                        stmnt_us += stmnt_user;
                    }
                String stmnt_end = ")";
                stmnt_us = stmnt_us.substring(0, stmnt_us.length() - 4);
                stmnt_us += stmnt_end;
                //System.out.println(stmnt_us);

                }


                String stmnt_str = "SELECT Username, IsPull, TimeStamp, Value, Unit, SensorType FROM _Response, _Report, _Datarecord, _Sens
                String stmnt_pp = " AND IsPull = ?::bool";
                String stmnt_date = " AND date(TimeStamp) >= ?::date AND date(TimeStamp) <= ?::date ";
                stmnt_str += stmnt_date;
                stmnt_str += stmnt_us;


                stmnt = connection.prepareStatement(stmnt_str);
```

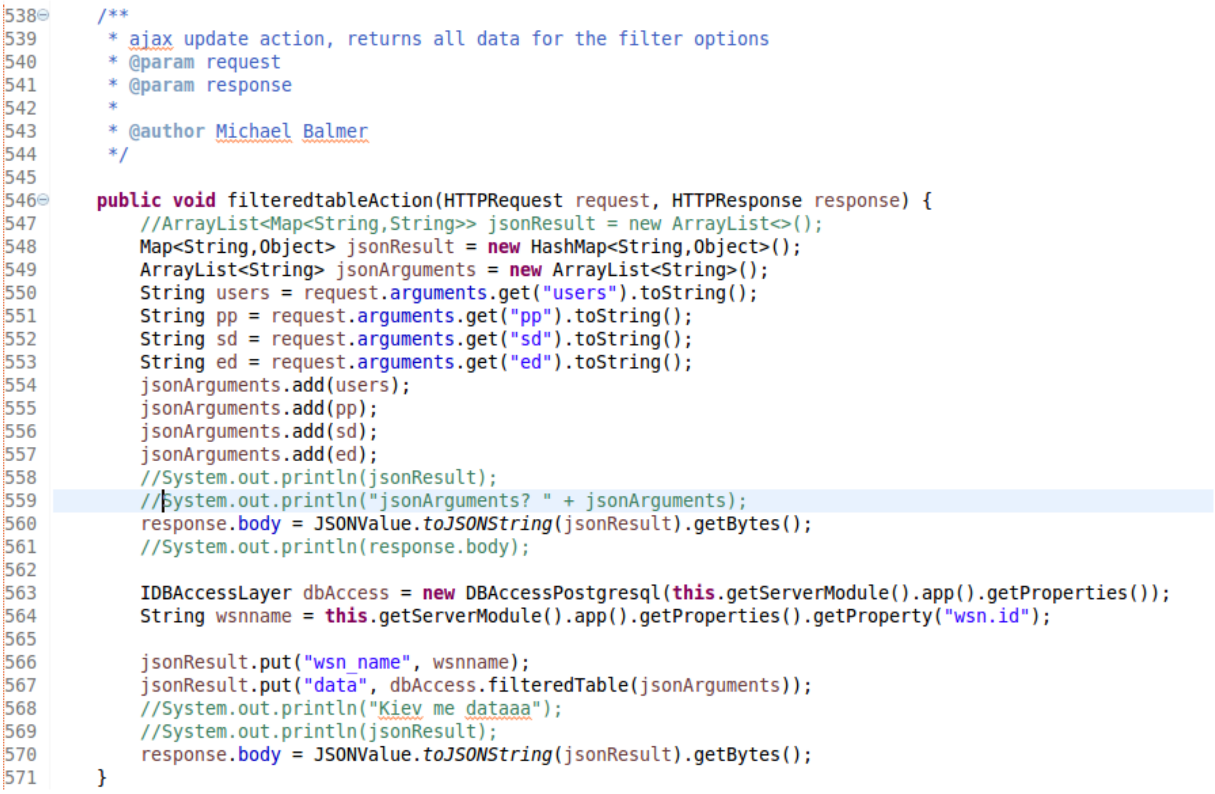
Figure 4.10: Code for filteredData DB Access 1



```java
            stmnt_str += stmnt_pp;
            stmnt = connection.prepareStatement(stmnt_str);
            stmnt.setString(3, "True");
        }else if (pp.equals("Push")) {
            //System.out.println("PUSHH ITT");
            stmnt_str += stmnt_pp;
            stmnt = connection.prepareStatement(stmnt_str);
            stmnt.setString(3, "False");
        }



        //Startdate and Enddate for Query
        String sd = statements.get(2);
        String ed = statements.get(3);
        stmnt.setString(1, sd);
        stmnt.setString(2, ed);

        ResultSet rs = stmnt.executeQuery();
        ResultSetMetaData rsmt = rs.getMetaData();
        int columnCount = rsmt.getColumnCount();
        //System.out.println(columnCount);
        while(rs.next()){
            Map<String,String> resultMap = new HashMap<>();
            //Print one row
            for(int i = 1 ; i <= columnCount; i++){

             // System.out.print(rs.getString(i) + " "); //Print one element of a row
              resultMap.put(rsmt.getColumnName(i).toLowerCase(),rs.getString(i));

            }
            res.add(resultMap);
        }
    }catch (SQLException e) {
        System.out.println("Statement creation Failed!");
        e.printStackTrace();
    }finally {
        try {
            if (stmnt != null){
                stmnt.close();
            }
            connection.close();
        }catch (SQLException e) {
            System.out.println("Couldn't close connection!");
            e.printStackTrace();
        }
    }
}
return res;
```

Figure 4.11: Code for filteredData DB Access 2

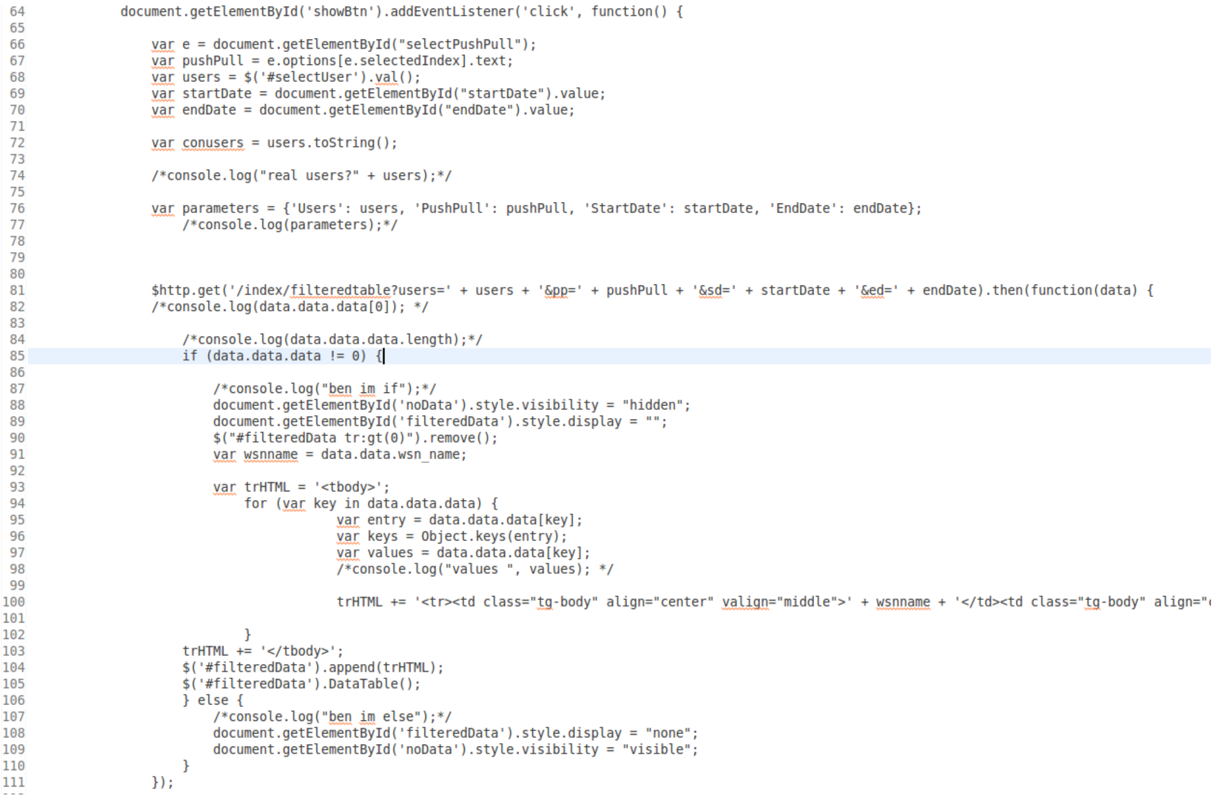
In figure 4.10 and 4.11 the method for the queries to the database is shown. As variables we receive an ArrayList of Strings containing the selected options for filtering. According to these statements we concatenate the query statement. Like in between line 333 and 344 whether the filter is set to All Users, multiple WSN users or even just a single user. The method then returns a ArrayList of a Map<String, String> which is filled starting from line 386 where each row is processed and stored together with its column name.



```
    /**
     * ajax update action, returns all data for the filter options
     * @param request
     * @param response
     *
     * @author Michael Balmer
     */

    public void filteredtableAction(HTTPRequest request, HTTPResponse response) {
        //ArrayList<Map<String,String>> jsonResult = new ArrayList<>();
        Map<String,Object> jsonResult = new HashMap<String,Object>();
        ArrayList<String> jsonArguments = new ArrayList<String>();
        String users = request.arguments.get("users").toString();
        String pp = request.arguments.get("pp").toString();
        String sd = request.arguments.get("sd").toString();
        String ed = request.arguments.get("ed").toString();
        jsonArguments.add(users);
        jsonArguments.add(pp);
        jsonArguments.add(sd);
        jsonArguments.add(ed);
        //System.out.println(jsonResult);
        //System.out.println("jsonArguments? " + jsonArguments);
        response.body = JSONValue.toJSONString(jsonResult).getBytes();
        //System.out.println(response.body);

        IDBAccessLayer dbAccess = new DBAccessPostgresql(this.getServerModule().app().getProperties());
        String wsnname = this.getServerModule().app().getProperties().getProperty("wsn.id");

        jsonResult.put("wsn_name", wsnname);
        jsonResult.put("data", dbAccess.filteredTable(jsonArguments));
        //System.out.println("Kiev me dataaa");
        //System.out.println(jsonResult);
        response.body = JSONValue.toJSONString(jsonResult).getBytes();
    }
```

Figure 4.12: Code for filtereddataAction

Figure 4.12 is an important point of communication between the Javascript file and the PostGreSqlAccess file. In the first part we receive the arguemnts from the selected filter options and add those to an ArrayList of Strings called jsonArguments. Like in the previous two Action methods a dbAccess object is generated on which we call our filteredTable method together with the selected options. The result from the query is then stored and assigned to the response body.



```
document.getElementById('showBtn').addEventListener('click', function() {

    var e = document.getElementById("selectPushPull");
    var pushPull = e.options[e.selectedIndex].text;
    var users = $('#selectUser').val();
    var startDate = document.getElementById("startDate").value;
    var endDate = document.getElementById("endDate").value;

    var conusers = users.toString();

    /*console.log("real users?" + users);*/

    var parameters = {'Users': users, 'PushPull': pushPull, 'StartDate': startDate, 'EndDate': endDate};
        /*console.log(parameters);*/



    $http.get('/index/filteredtable?users=' + users + '&pp=' + pushPull + '&sd=' + startDate + '&ed=' + endDate).then(function(data) {
    /*console.log(data.data.data[0]); */

        /*console.log(data.data.data.length);*/
        if (data.data.data != 0) {

            /*console.log("ben im if");*/
            document.getElementById('noData').style.visibility = "hidden";
            document.getElementById('filteredData').style.display = "";
            $("#filteredData tr:gt(0)").remove();
            var wsnname = data.data.wsn_name;

            var trHTML = '<tbody>';
                for (var key in data.data.data) {
                            var entry = data.data.data[key];
                            var keys = Object.keys(entry);
                            var values = data.data.data[key];
                            /*console.log("values ", values); */

                            trHTML += '<tr><td class="tg-body" align="center" valign="middle">' + wsnname + '</td><td class="tg-body" align="

                }
        trHTML += '</tbody>';
        $('#filteredData').append(trHTML);
        $('#filteredData').DataTable();
        } else {
            /*console.log("ben im else");*/
            document.getElementById('filteredData').style.display = "none";
            document.getElementById('noData').style.visibility = "visible";
        }
    });
```

Figure 4.13: Code for displaying data in javascript

On hitting the show filtered data button this whole cascade of events is triggered. As can be seen in figure 4.13. On line 66 the selected filter options are read and sent with the http.get() on line 81. When the data returns we iterate through each row on line 94 and display in the table on the html file (line 100).

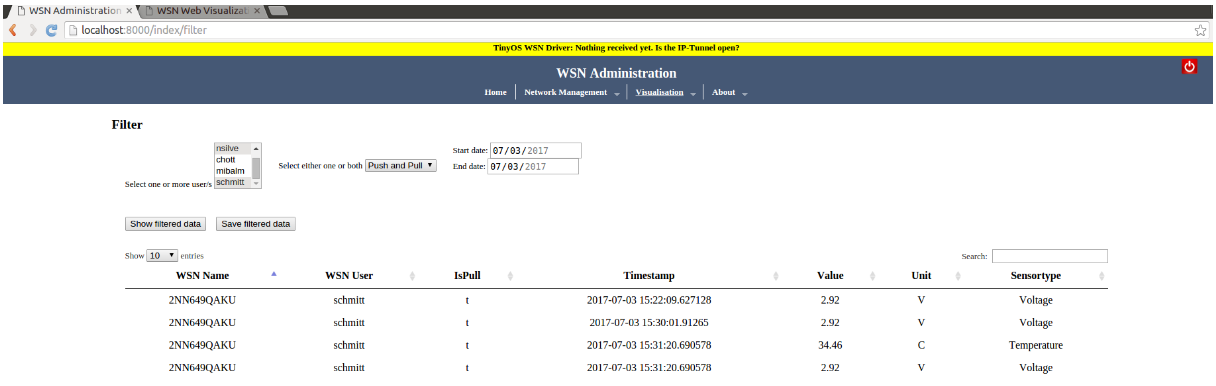
Chapter 5

Evaluation

As Evaluation a proof of operability is done as the functionality of this assignment is best tested and shown at work. The expected results are taken from queries on the database.

5.1 Proof of operability

Filtersettings: nsilve,schmitt / PushPull / 7/3/2017 Expected: 4 pulls of schmitt Results: correct



TinyOS WSN Driver: Nothing received yet. Is the IP-Tunnel open?

Select one or more user/s: nsilve, schmitt

Select either one or both: Push and Pull

Start date: 07/03/2017

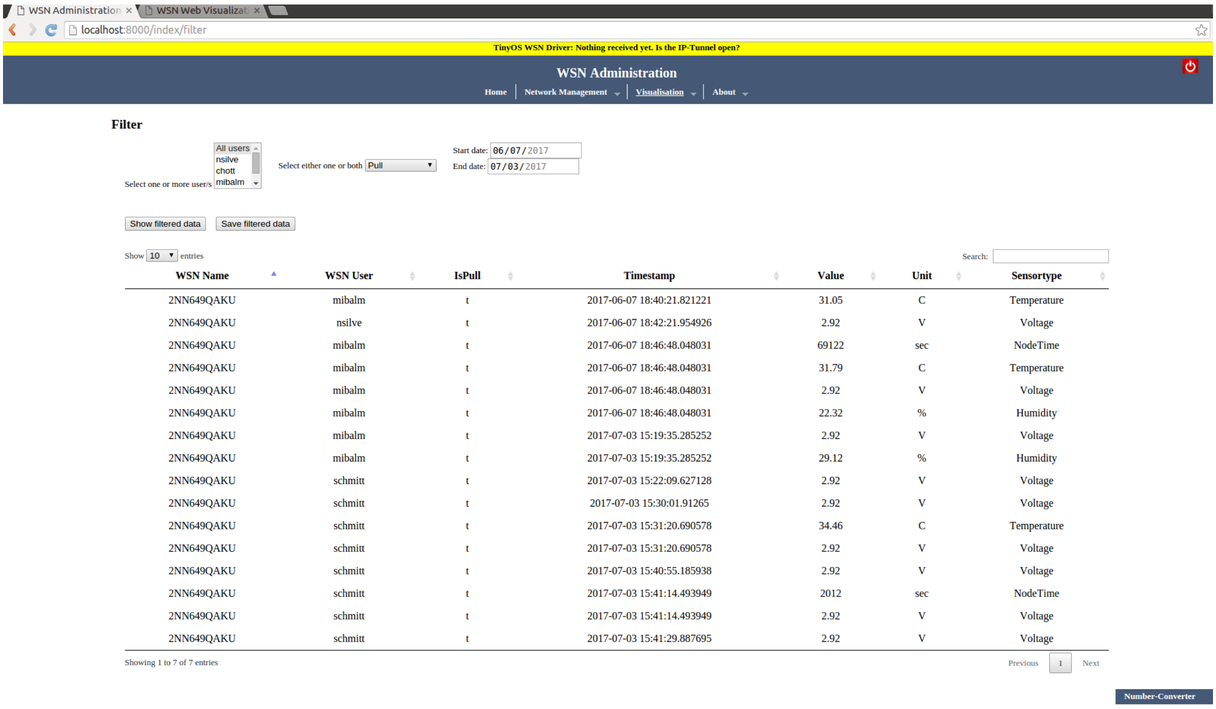
End date: 07/03/2017

Show 10 entries

WSN Name	WSN User	IsPull	Timestamp	Value	Unit	Sensortype
2NN649QAKU	schmitt	t	2017-07-03 15:22:09.627128	2.92	V	Voltage
2NN649QAKU	schmitt	t	2017-07-03 15:30:01.91265	2.92	V	Voltage
2NN649QAKU	schmitt	t	2017-07-03 15:31:20.690578	34.46	C	Temperature
2NN649QAKU	schmitt	t	2017-07-03 15:31:20.690578	2.92	V	Voltage

Figure 5.1: Evaluationsatz 1

Filtersettings: All users / Pull / 6/7/2017 - 7/3/2017 Expected: 16 rows Results: correct



TinyOS WSN Driver: Nothing received yet. Is the IP-Tunnel open?

WSN Administration

Filter

Select one or more user/s: All users, nsilve, chott, mibalm

Select either one or both: Pull

Start date: 06/07/2017

End date: 07/03/2017

Show 10 entries

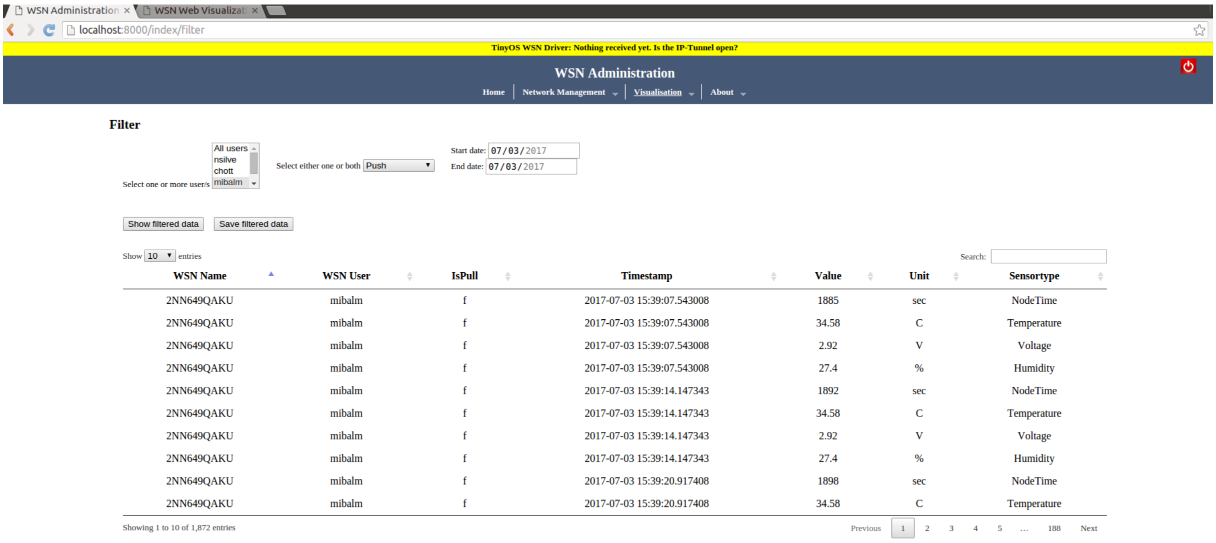
Search:

WSN Name	WSN User	IsPull	Timestamp	Value	Unit	Sensortype
2NN649QAKU	mibalm	t	2017-06-07 18:40:21.821221	31.05	C	Temperature
2NN649QAKU	nsilve	t	2017-06-07 18:42:21.954926	2.92	V	Voltage
2NN649QAKU	mibalm	t	2017-06-07 18:46:48.048031	69122	sec	NodeTime
2NN649QAKU	mibalm	t	2017-06-07 18:46:48.048031	31.79	C	Temperature
2NN649QAKU	mibalm	t	2017-06-07 18:46:48.048031	2.92	V	Voltage
2NN649QAKU	mibalm	t	2017-06-07 18:46:48.048031	22.32	%	Humidity
2NN649QAKU	mibalm	t	2017-07-03 15:19:35.285252	2.92	V	Voltage
2NN649QAKU	mibalm	t	2017-07-03 15:19:35.285252	29.12	%	Humidity
2NN649QAKU	schmitt	t	2017-07-03 15:22:09.627128	2.92	V	Voltage
2NN649QAKU	schmitt	t	2017-07-03 15:30:01.91265	2.92	V	Voltage
2NN649QAKU	schmitt	t	2017-07-03 15:31:20.690578	34.46	C	Temperature
2NN649QAKU	schmitt	t	2017-07-03 15:31:20.690578	2.92	V	Voltage
2NN649QAKU	schmitt	t	2017-07-03 15:40:55.185938	2.92	V	Voltage
2NN649QAKU	schmitt	t	2017-07-03 15:41:14.493949	2012	sec	NodeTime
2NN649QAKU	schmitt	t	2017-07-03 15:41:14.493949	2.92	V	Voltage
2NN649QAKU	schmitt	t	2017-07-03 15:41:29.887695	2.92	V	Voltage

Showing 1 to 7 of 7 entries

Figure 5.2: Evaluationsatz 2

Filtersettings: mibalm / Push / 6/7/2017 - 7/3/2017 Expected: 1872 Results: correct



TinyOS WSN Driver: Nothing received yet. Is the IP-Tunnel open?

WSN Administration

Filter

Select one or more user/s: All users, nsilve, chott, mibalm

Select either one or both: Push

Start date: 07/03/2017

End date: 07/03/2017

Show 10 entries

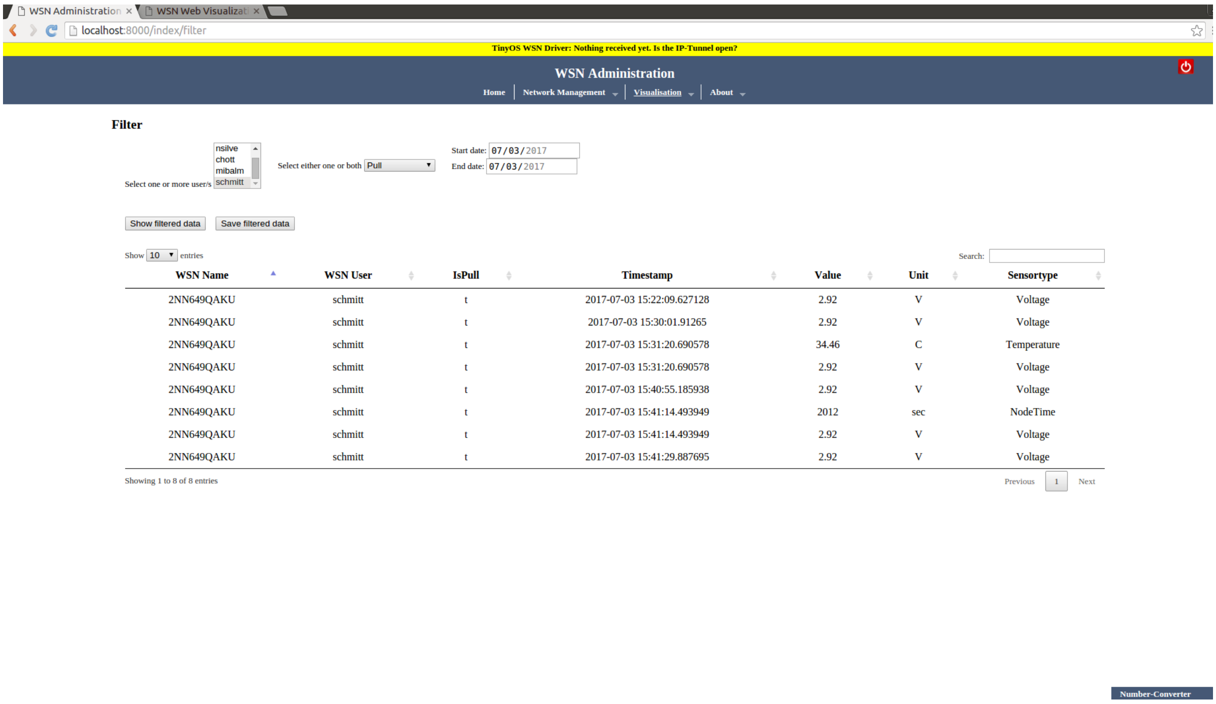
Search:

WSN Name	WSN User	IsPull	Timestamp	Value	Unit	Sensortype
2NN649QAKU	mibalm	f	2017-07-03 15:39:07.543008	1885	sec	NodeTime
2NN649QAKU	mibalm	f	2017-07-03 15:39:07.543008	34.58	C	Temperature
2NN649QAKU	mibalm	f	2017-07-03 15:39:07.543008	2.92	V	Voltage
2NN649QAKU	mibalm	f	2017-07-03 15:39:07.543008	27.4	%	Humidity
2NN649QAKU	mibalm	f	2017-07-03 15:39:14.147343	1892	sec	NodeTime
2NN649QAKU	mibalm	f	2017-07-03 15:39:14.147343	34.58	C	Temperature
2NN649QAKU	mibalm	f	2017-07-03 15:39:14.147343	2.92	V	Voltage
2NN649QAKU	mibalm	f	2017-07-03 15:39:14.147343	27.4	%	Humidity
2NN649QAKU	mibalm	f	2017-07-03 15:39:20.917408	1898	sec	NodeTime
2NN649QAKU	mibalm	f	2017-07-03 15:39:20.917408	34.58	C	Temperature

Showing 1 to 10 of 1,872 entries

Figure 5.3: Evaluationsatz 3

Filtersettings: schmitt / Pull / 7/3/2017 Expected: 8 pulls of schmitt Results: correct



TinyOS WSN Driver: Nothing received yet. Is the IP-Tunnel open?

Filter

Select one or more user/s: nsilve, chott, mibalm, schmitt

Select either one or both: Pull

Start date: 07/03/2017

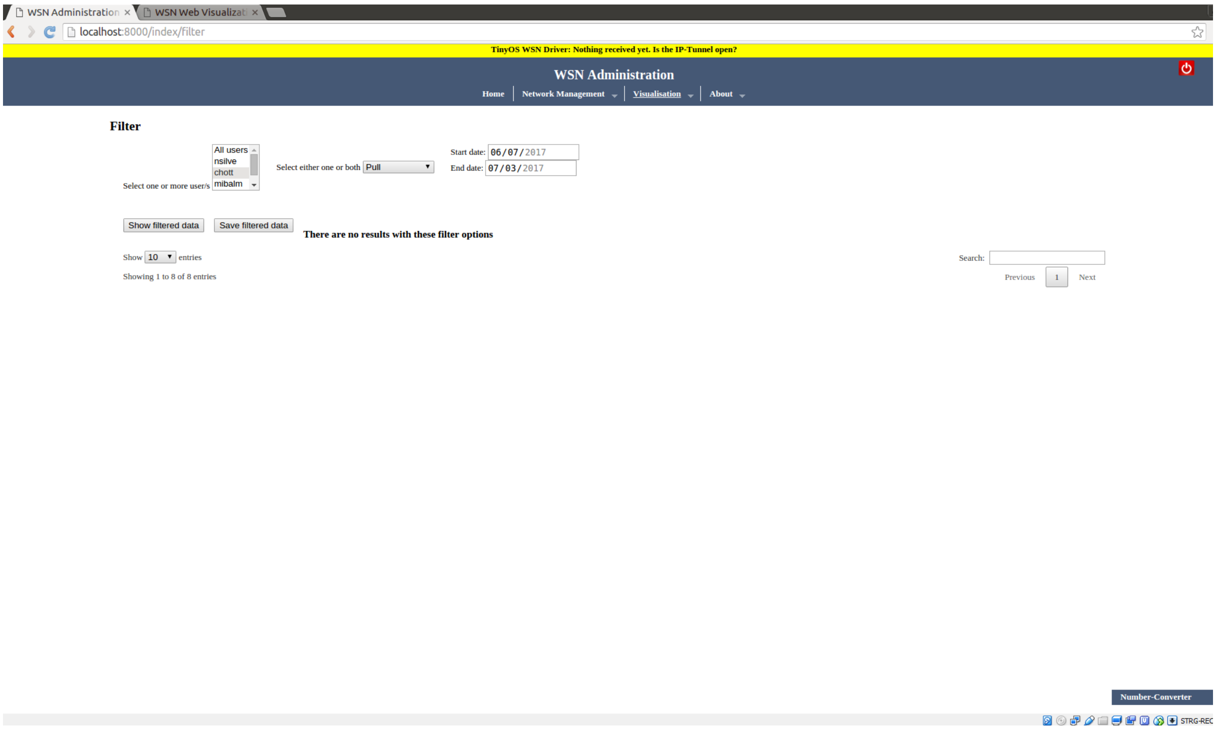
End date: 07/03/2017

WSN Name	WSN User	IsPull	Timestamp	Value	Unit	Sensortype
2NN649QAKU	schmitt	t	2017-07-03 15:22:09.627128	2.92	V	Voltage
2NN649QAKU	schmitt	t	2017-07-03 15:30:01.91265	2.92	V	Voltage
2NN649QAKU	schmitt	t	2017-07-03 15:31:20.690578	34.46	C	Temperature
2NN649QAKU	schmitt	t	2017-07-03 15:31:20.690578	2.92	V	Voltage
2NN649QAKU	schmitt	t	2017-07-03 15:40:55.185938	2.92	V	Voltage
2NN649QAKU	schmitt	t	2017-07-03 15:41:14.493949	2012	sec	NodeTime
2NN649QAKU	schmitt	t	2017-07-03 15:41:14.493949	2.92	V	Voltage
2NN649QAKU	schmitt	t	2017-07-03 15:41:29.887695	2.92	V	Voltage

Showing 1 to 8 of 8 entries

Figure 5.4: Evaluationsatz 4

Filtersettings: chott / Pull / 6/7/2017 - 7/3/2017 Expected: none Results: correct



TinyOS WSN Driver: Nothing received yet. Is the IP-Tunnel open?

Filter

Select one or more user/s: All users, nsilve, chott, mibalm

Select either one or both: Pull

Start date: 06/07/2017

End date: 07/03/2017

There are no results with these filter options

Show 10 entries

Search:

Showing 1 to 8 of 8 entries

Figure 5.5: Evaluationsatz 5

Chapter 6

Summary and Conclusions

A user-friendly GUI has been added to the existing WSN framework on CoMaDa. Therefore, a view exists addressing the active transparency request of the collected data within a sensor network without database access as before. For the new view all requested filtering options are implemented including the WSN user , option (pull / push) and date. The filtered data is displayed in a table with all requested columns additionally the table owns a sorting functionality. To keep a certain filtered table a save button is integrated, which allows the user to save the current table as a PDF.

As a first conclusion it can be said, that given the goals of adding a user-friendly GUI addressing the transparency request of the collected data within a sensor network without database access could be implemented. Although it was challenging to work into CoMaDa and understanding the modular structure and the relationship between them.

Bibliography

[1] H.Karl and A. Willig, *Protocols and Architectures for Wireless Sensor Networks*, John Wiley and Sons, Vol 1, ISBN: 0470519231, GB, 2007.

[2] SecureWSN, URL: `http://www.csg.uzh.ch/research/SecureWSN`, last visited June. 6, 2017.

[3] C. Schmitt, T. Strasser, B. Stiller, *Third-party-independent Data Visualization of Sensor Data in CoMaDa*; 12th IEEE International Conference on Wireless and Mobile Computing, Networking and Communications, New York, NY, U.S.A., Oct. 2016, pp 1-8.

[4] C. Schmitt, T. Strasser, B. Stiller, *Efficient and Secure Pull Requests for Emergency Cases Using a Mobile Access Framework*; in: M. Sheng, Y. Qin, L. Yao, B. Benatallah (Edt.),*WoT-book-Managing the Web of Things: Linking the Real World to the Web*, Elsevier, New York, NY, U.S.A., Feb. 2016, pp 1-19.

[5] Andr"e Freitag, Corinna Schmitt, Georg Carle, *CoMaDa: An Adaptive Framework with Graphical Support for Configuration*; 9th International Conference on Network and Service Management, Z"urich, Switzerland, October 2013, ISBN 978-3-901882-53-1, pp 211-218.

[6] Communication Systems Group (CSG), URL: `http://www.csg.uzh.ch/`, last visited June. 6, 2017.

[7] C. Schmitt, M. Keller, and B. Stiller, *WebMaDa: Web-based Mobile Access and Data Handling Framework for Wireless Sensor Networks (Demo Paper)*; In International Conference on Networked Systems (NetSys), Cottbus, Germany, March 2015.

[8] T. Strasser, *Method for Graphical Visualization of Sensor Data*; Assignment, University of Zurich, Communication Systems Group, Department of Informatics, Z"urich, Swizerland, March 2016.

[9] C. Ott, *Database Solution for Offline Graphical Visualization of Sensor Data*; Assignment, University of Zurich, Communication Systems Group, Department of Informatics, Z"urich, Switzerland, January 2017.

[10] WebMaDa, URL: `https://webmada.csg.uzh.ch/`, last visited June. 6, 2017.

[11] W3SCHOOLs, URL: `https://www.w3schools.com/tags/att_select_multiple.asp`, last visited June. 29 , 2017.

[12] tablegen, URL: http://www.tablesgenerator.com/html_tables#, last visited June. 29, 2017.

[13] oracle, URL: http://docs.oracle.com/javase/tutorial/jdbc/basics/prepared.html, last visited June. 29, 2017.

List of Figures

List of Tables

Appendix A

Contents of the CD

The attached CD contains the following files and directories:

- `Thesis VA Michael Balmer.pdf`: PDF of the submission
- `code`: Directory containing the CoMaDa source code
- `tex`: Directory containing the latex sources of this report
- `presentation`: Directory containing the final presentation