University of Zurich[UZH]

BACHELOR THESIS – Communication Systems Group, Prof. Dr. Burkhard Stiller

Security, Privacy, and Transparency Improvements in CoMaDa

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

Supervisor: Corinna Schmitt, Bruno Rodrigues
Date of Submission: February 1, 2018

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

CSG
Department of Informatics
Communication Systems Group

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

Abstract

The work of this Bachelor thesis includes the development for improving security, privacy and transparency functionalities in CoMaDa (Configuration, Management, and Data handling) which is part of the SecureWSN project at the University of Zurich. These tasks are merely separated in two work processes. The first one addresses the security functionality of CoMaDa. Therefore a two-factor user authentication is implemented to ensure only WSN owners can enter the functionality of CoMaDa. The other aspect concerns SQL-query injection hence a check is implemented that a WSN only contains of letters and numbers. The second task engages the improvement of privacy and transparency. This builds on my assignment and the therefore implemented filtering option. Hence the filtering option is extended by adding more fine grained filter options explicitly the selection of a certain sensor, the selection of not just the date but also time range. Furthermore to reach 100 percent transparency allowing the filtering of push messages performed by either the data owner or an authorized user and an additional option for the owner to inspect the log of the filtering, meaning each executed filtering is logged by time, requested filter options and if an output file was created or not. Exactly those output files are now additionally secured by encryption. As a result of the performed tasks the existing database solution had to be extended by tables for logging. In addition the changes in CoMaDa which affect WebMaDa are performed as well.

Zusammenfassung

Die Bachelorarbeit beinhaltet die Entwicklung von Verbesserungen im Bereich der Sicherheit, dem Datenschutz und der Nachvollziehbarkeit der Funktionen von CoMaDa (Configuration, Management, and Data handling), welches teil des SecureWSN Projekts an der Universität Zürich ist. Die Arbeit ist dadurch mehrheitlich in zwei Teile gegliedert. Der erste Teil behandelt die Funktionen von CoMaDa bezüglich der Sichherheit. Deshalb wird beim Start von CoMaDa eine 2-Faktor Authentifizierung implementiert, welche sicherstellt, dass nur der Besitzer des WSNs Zugriff auf die Funktionalitäten darin hat. So wird ebenfalls eine Prüfung des Benutzer vor Eintritt in die Filtering Option hinzugefügt, da darin sensible Daten punktgenau ausgelesen werden können. The andere Aspekt betrifft die Eindämmung der Möglichkeit einer SQL-Query injection dafür wird eine Überprüfung des WSN-Namens vorangeschickt, ob dieser nur aus Buchstaben und Nummern besteht. Der zweite Aufteilungsschritt befasst sich mit der Verbesserung des Datenschutzes und der Nachvollziehbarkeit. Dies baut auf meiner Vertiefungsarbeit auf und der dabei hinzugefügten Filteroption. Infolgedessen ist diese Option mit weiteren Möglichkeiten zum Filtern ergänzt worden. Darunter gehört das Auswählen bestimmter Sensoren und die Auswahl der exakten Uhrzeit und nicht nur des Datums. Zusätzlich um die angestrebte 100% Nachvollziehbarkeit zu erreichen, ist es möglich nach Pushnachrichten vom Dateninhaber als auch von berechtigten Nutzern zu filtern ausserdem wird eine weitere Möglichkeit hinzugefügt um das Logging des Filtering einzusehen. Das bedeutet, jedes durchgeführte Filtering wird mit Zeitstempel, angewählten Filteroptionen und Überprüfung ob ein Outputfile erstellt wurde gespeichert. Um den Datenschtutz zu erhöhen, ist eben das oben genannte Outputfile zusätzlich noch mit einer Verschlüsselung versehen. Als Result dieser Ausführungen musste die existierende Datenbanklösung erweitert werden um die Log-Daten zu speichern. Ausserdem sind Änderungen in CoMaDa, welche WebMaDa betreffen auch ergänzt worden.

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 all phases of the Bachelor Thesis. I would also like to thank Prof. Dr. Burkhard Stiller for his support and the possibility to complete this Bachelor Thesis 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 Sensor 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. Exactly such kind of data can be sensitive and it is important to be accessed only by authorized personal. In this case especially data owners have concerns on security and privacy. Therefore only data owners should be able to control the data flow and have the possibility to follow the data access and enter the filtering options for special data extraction. [6]

This bachelor thesis builds on 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 the Configuration, Management, and Data handling framework (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]. In the current setup in CoMaDa 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 my last assignment was to allow the data owner to access a user-friendly view with filtering options to display who, when and what data was collected for all his users. With this solution a user is able to filter current and old data to gain access control and partial transparency. This thesis now aims to gain further security and transparency as well as solve privacy concerns for data owners in CoMaDa.

1.2 Description of Work

The task of this Bachelor thesis is to improve the security and privacy functionalities of SecureWSNs CoMaDa part. To reach this goal the following work is two-folded and needs to be done: The first part of the task focuses on security improvement and the second part on privacy improvement together with transparency. For improving security a two-factor-user-authentication on CoMaDa needs to be realized. This is necessary, because when CoMaDa is started the credentials are stored on the hosting machine and are not further checked when the data is analysed or accessed (e.g., data filtering for transparency and privacy purpose in case of data access control). Second, for securing the total system against SQL-query injection for setting up a new WSN, a special security check need to be integrated in the total system checking the naming of new WSNs only includes letters and numbers. For improving privacy together with transparency of the system this Bachelor thesis builds on the performed assignment of M. Balmer [11] by extending the already developed filtering options to gain 100% transparency and integrating security feature for the filtering results. Therefore, the existing filtering solution needs to become more fine-grained, in order to allow the data owner to filter the data for selected sensors and data within data/time range as well as including filtering of push messages distinguished between pushes performed by the data owner and granted to authorized users. For both the existing data base solutions needs to be extended in its existing structure (e.g., adding new tables with corresponding information). Further, the performed filtering request needs to be logged in the system including among other information about performing time, requested filtering, and if the result was printed in a PDF or not. Currently a printing of the filtering result is possible but insecure. This needs to be solved by securing the printed PDF using encryption. Finally, to optimize the use-friendly idea of CoMaDa a description of filtering options need to be included and in case CoMaDa updates will effect WebMaDas configuration this must be addressed, too. The result of this Bachelor will be a user-friendly GUI extension for CoMaDa addressing existing security and privacy issues together with addressing the 100% transparency request of data owners offering different filtering options allowing specific information requests.

Through the motivation and description of work the following research questions omitted.

- How can the security and privacy risks in CoMaDa be reduced?
- What extensions for the filtering option addressing transparency need to be implemented?
- How can we gain further transparency and traceability?

1.3 Thesis Outline

The rest of the thesis report is structured as follows. Chapter 2 includes related work to give a brief overview of the works done in the assignment and the works related to this thesis. Chapter 3 discusses the design decisions to reach consensus on security, privacy and

transparency questions. Chapter 4 show the process of the implementation. In Chapter 5 an evaluation concerning workflow, filtering and performance will be done by a proof of operability. In the final Chapter 6 the conclusion is drawn.

Chapter 2

Related Work

This chapter goes over the relevant topics for making the work possible. Therefore a closer look is taken at the SecureWSN framework with its two parts CoMaDa and WebMaDa as this is the workplace for this thesis. For this reason we will show the communication between CoMaDa and WebMaDa for a better idea about their relationship and the authentication process. A more detailed part explains the functionality of CoMaDa with a special section about the existing filtering option.

2.1 SecureWSN

In the beginning of project SecureWSN [2] the goal was to develop an efficient data transmission protocol. Messages within a WSN have in common that they include meta information and measure values at the same time which are send automatically in a predefined interval. [2] As the project evolved the security request by the users was taken on. Therefore three different security solutions which keep the device resources in mind have been implemented. To take care of the user friendliness, a graphical user interface (GUI) was designed in SecureWSN. The framework to Configure, Manage and handle Data (CoMaDa) fulfils exactly this purpose. To address the mobility requirement of the user, the Web-based Mobile Access and Data Handling Framework (WebMaDa) was designed. [2] All those components together present the current SecureWSN framework. For a better understanding all the relevant components are shown in Figure 2.1 in a simplified way. It shows CoMaDa which is used only by a specific WSN owner and is the framework where a WSN is deployed. The Web-based part in the SecureWSN, WebMaDa is available for Guests as well as WSN owners and is connected to CoMaDa via the Communication System Groups (CSG) Infrastructure, which contains of the MySQL Database for WebMaDa and CoMaDa as well as the Webserver for making the connection possible.

Another overview on the SecureWSNs components and especially the communication paths is shown in Figure 2.2. It can again be seen, that CoMaDa is responsible for the deployment of a WSN. For the data transmission between the two components an Upload and Pull interface was implemented by Claudio Anliker [10].

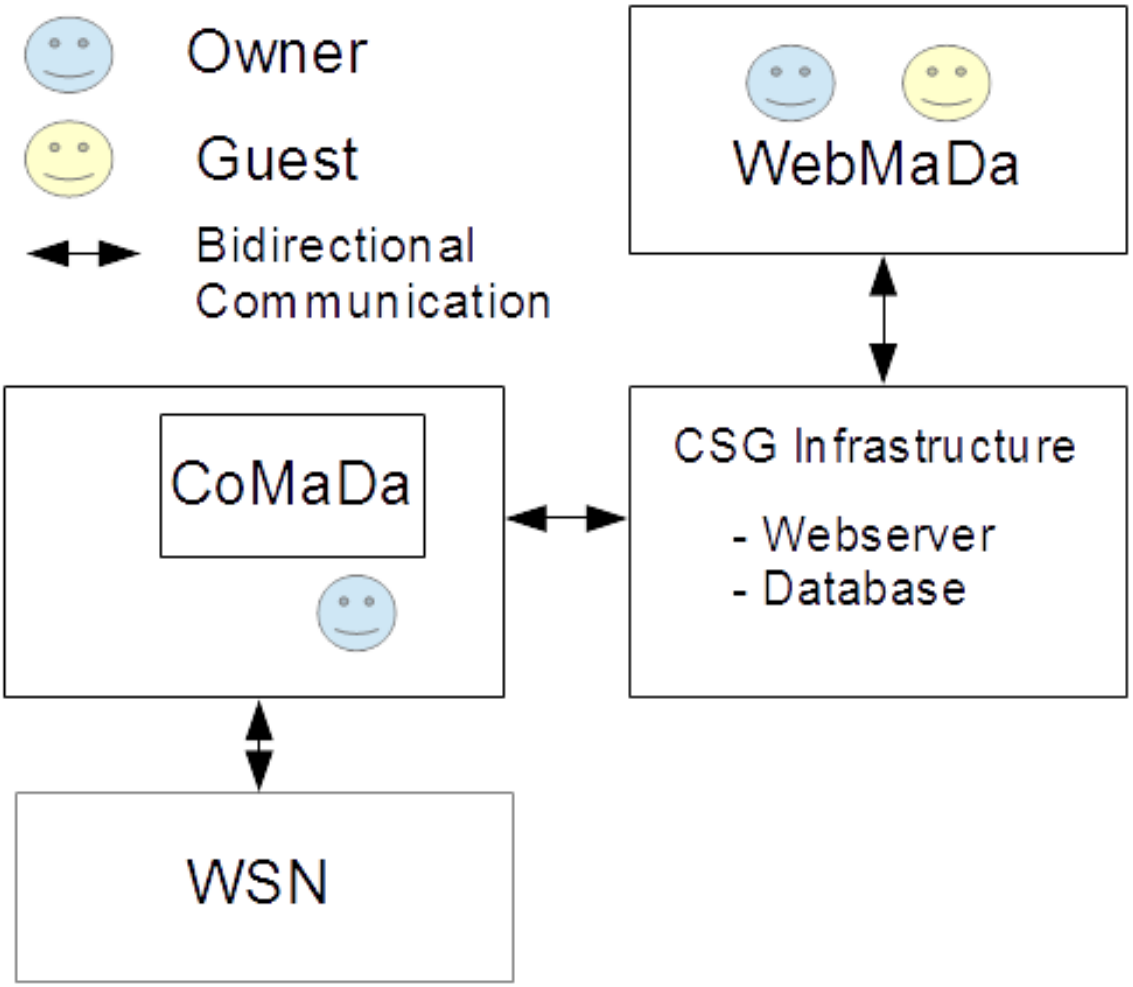


Figure 2.1: Components of SecureWSN

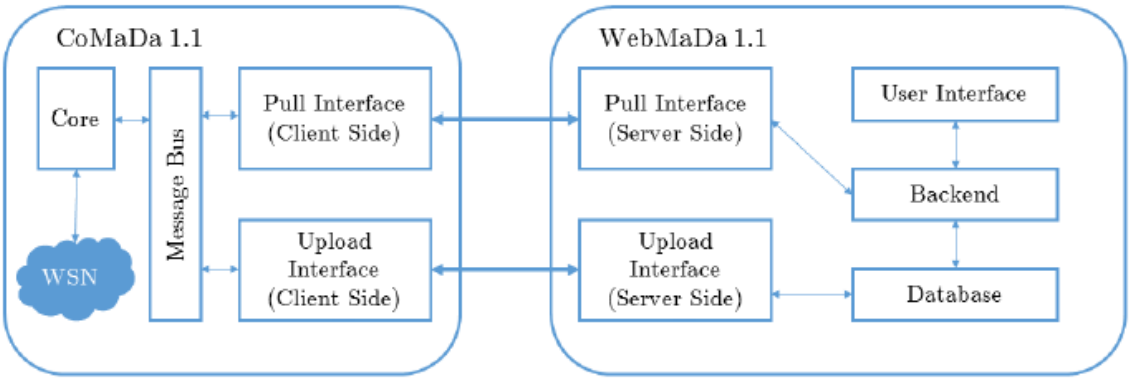


Figure 2.2: Overview of Upload/Pull interface [10]

2.2 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. Therefore a local webserver is started to display this GUI. 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 backend 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 were important for the assignment [11]. As 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 and Javascript-Files. Those communicate through a WebServer integrated in the backend with the rest of the system. [11]

2.2.1 Filtering Option CoMaDa

In the assignment of Neva Silvestri [12] and Michael Balmer [11] a user-friendly extension for WebMaDa and CoMaDa was introduced. On a new filtering page a WSN owner is able to filter sensor data for users, date and push/pull option. The owner of the WSN then sees the selected sensor data. Additionally the owner can download the data as a PDF file. For this bachelor thesis exactly this filtering function on the CoMaDa side is a main subject.

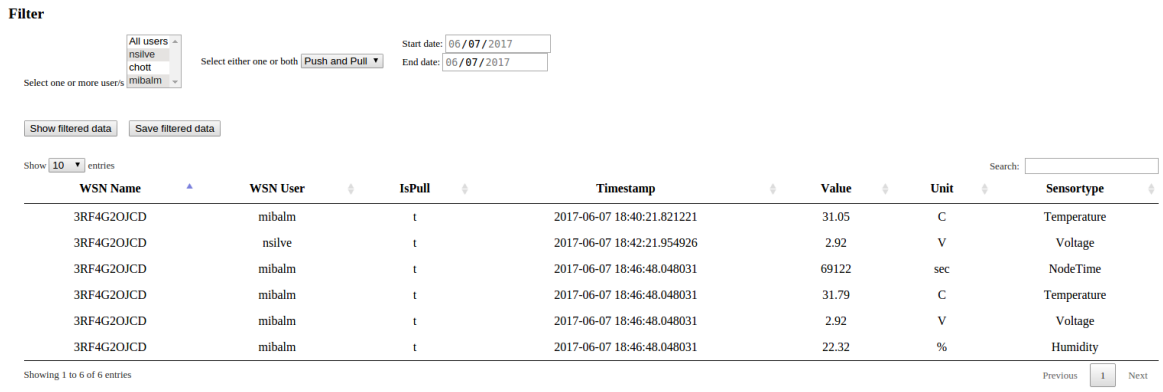


Figure 2.3: Filter option in CoMaDa [11]

In Figure 2.3 the filtering option is shown in CoMaDa. It allows a WSN owner to search a certain set of data from the WSN. The users who operated on the WSN can be selected. The type of request namely Push or Pull or Both and a Start and End date. By pressing the ”Show filtered data” button the result is then displayed in a datatable together with the exact timestamp the selected fiter options and the sensor data. The ”Save filtered data” button creates a PDF of the current datatable and saves it in the download folder.

2.3 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. [13]

2.4 Findings

The biggest part of this work lies within the CoMaDa framework with the extension of the filtering option which uses the back- and frontend. For this part the assignment "CoMaDa Extension Addressing Transparency Request for Data Owners" of Michael Balmer [11] is the reference model as is the assignment of Neva Silvestri [12] for WebMaDa. The upload and pull interface also gains on importance as it is used to write information to the WebMaDa database. This knowledge can be acquired from the work of Claudio Anliker [10] who implemented the newer versions 1.1 for CoMaDa and WebMaDa.

Chapter 3

Design Decisions

In this chapter a conclusion for the design choices will be made. Therefore we refer to the research questions in the introduction:

- How can the security and privacy risks in CoMaDa be reduced?
- What extensions for the filtering option addressing transparency need to be implemented?
- How can we gain further transparency and traceability?

and figure 2.1 of the components of SecureWSN. Putting those two togther gives us a combination as shown in figure 3.1

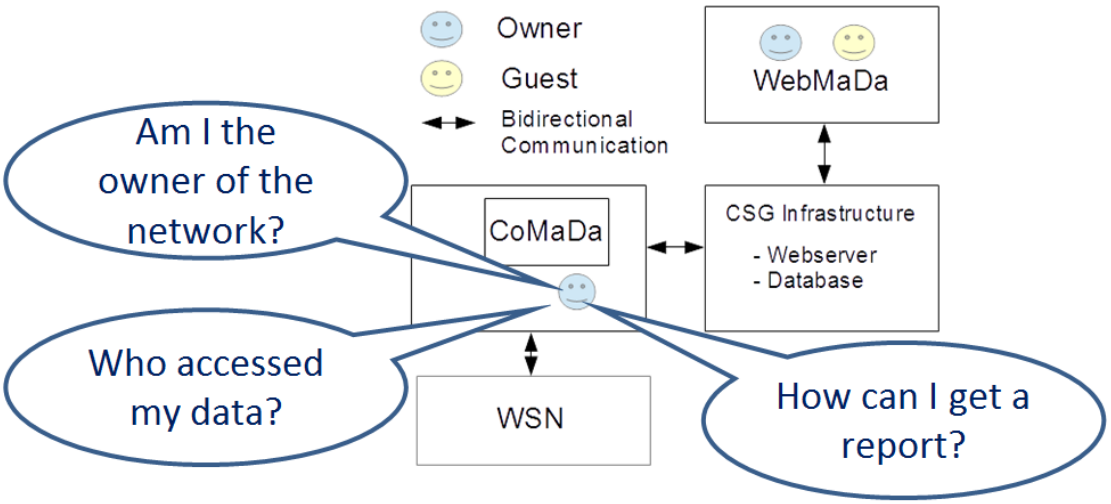


Figure 3.1: Research questions in SecureWSN

Therefore these three questions are processed in different chapters to make the design decisions.

- Am I the owener of the network?
- Who accessed my data?
- How can I get a report?

The first question refers to the security and privacy concerns in CoMaDa and is discussed in Chapter 3.1 Security and Privacy risks in CoMaDa. The second question takes up the further transparency and traceability gain and is examined in Chapter 3.2 Traceability. The final question refers to the filtering options and its transparency request and is reviewed in Chapter 3.3 Transparency.

3.1 Security and Privacy risks in CoMaDa

The current 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 has access to. But usually the reality looks different. As the sensor data on a WSN can also be sensitive it is even more important to take security seriously. The current implementation looks as follows: A user in WebMaDa can request a WSN ID from an admin to configure the config file in CoMaDa and gain access to all features in CoMaDa eg. deploy a WSN, filter data and extract it. For this reason the first assumption that only the WSN owner has access to CoMaDa is not secure enough as no further security check is performed. Taking everything into consideration it can never be guaranteed that only a WSN owner has access to CoMaDa as not only technical but especially organisational measures matters. Thus for this work only technical measures can be realized. A two-way authentication to check the credentials would increase the security. Therefore a new login screen before the filtering page must be implemented. A mockup for such a login screen is shown in figure 3.2.

Another security issue appears as soon as sensitive data can get extracted from CoMaDa. This feature was implemented together with the filtering option on the assignment of Michael Balmer [11]. The PDF file is stored in the download folder and can be opened by any person on the workstation. This offends the security and privacy measure of an authentication. Therefore the output files need to be encrypted.

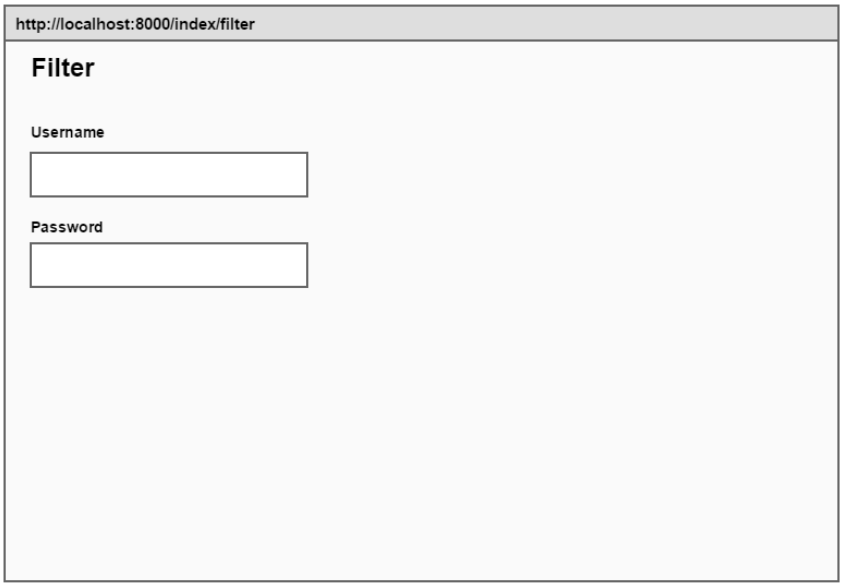


Figure 3.2: Mockup for the login screen

3.2 Traceability

In this chapter we evaluate the question ”Who accessed my data?” in depth. For a data owner it is essential to have the possibility to track the activity in CoMaDa. This can be for statistical evaluations or up to clearing cases of fraud. In order to make this possible certain movements in CoMaDa need to be logged. Hence interesting aspects are security and privacy relevant activities. Addressing these security and privacy relevant activities the database needs to be revisited. In Figure 3.1 an overview of the current database tables is shown. The left side of the overview shows all tables which are created for every WSN. They are mainly used for sensor relevant data and the tables exist in the WebMaDa as well as in the CoMaDa database. The right side shows the tables which are created just once. This tables are only available on the MySql database in WebMaDa. The main purpose for the tables is the administration for User and their WSNs in WebMaDa.

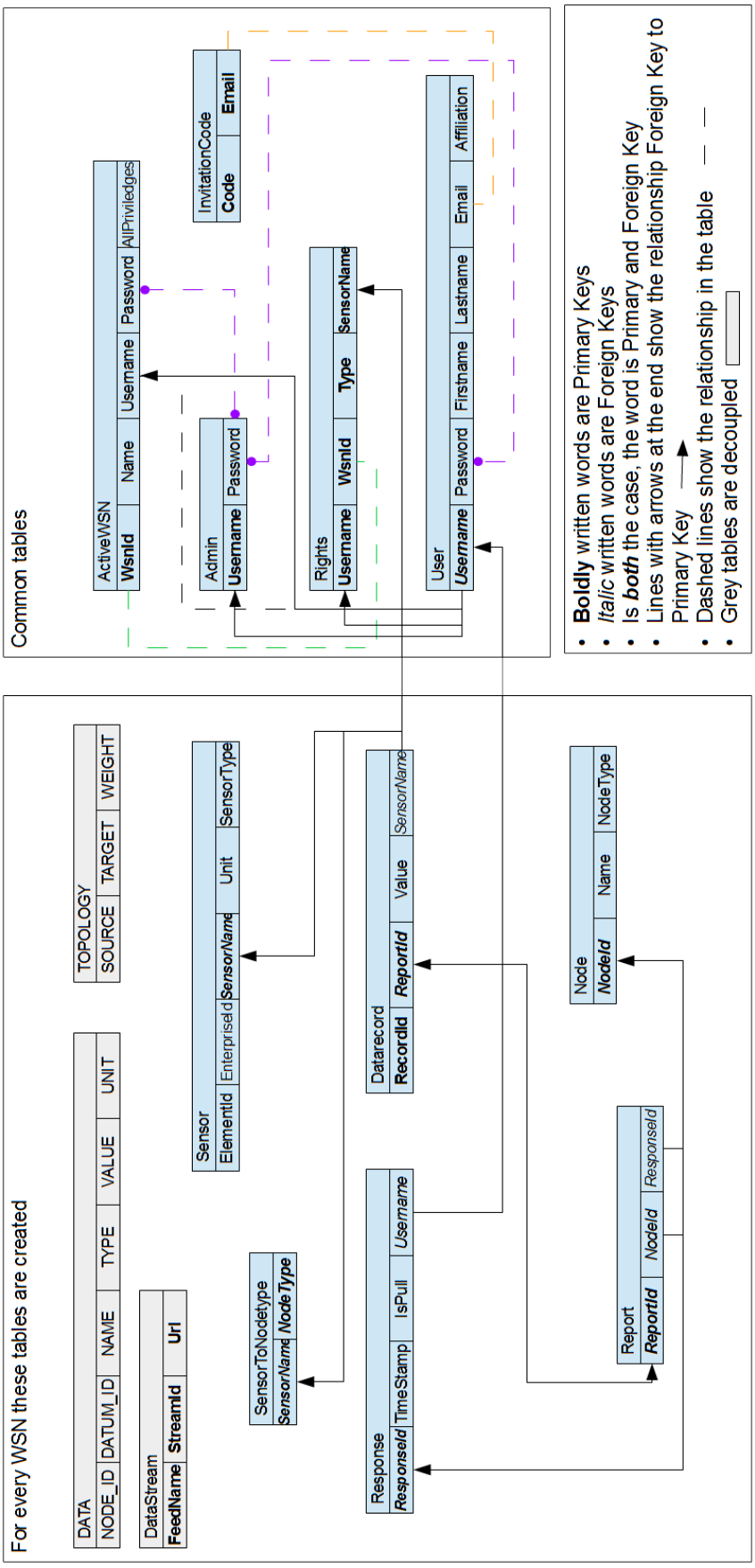


Figure 3.3: Overview database tables [12]

One part for gaining more privacy and transparency is by logging the use of the filtering option. This assures traceability for a WSN owner in case of forbidden usage. In order to achieve this, the selected filtering options need to be stored in a table. Figure 3.4 shows such a table. The Filter_Log table consists of seven columns. The WsnId is the first column with datatype character varying to recognize which WSN is currently active on CoMaDa. The Owner is the second column with datatype character varying which represents the WSN owner. The Timestamp shows the timestamp on the moment the query for the filtering was made. The last four columns contain the information on the filtering. Column Users with datatype character varying consists of all selected Users in the filtering. PP_Type contains the option Push/Pull or both. SDate and EndDate include the selected Start and End Date with datatype timestamp.

Filtering_Log						
WsnId	Owner	Timestamp	Users	PP_Type	SDate	EndDate

Figure 3.4: Table Filtering_ Log

An additional way for more privacy and transparency that involves database adjustment would be a tracking for the login into the filtering option. This table could look like in Figure 3.5. This table consists of five columns. To track the login the most important columns are the WsnId for the active WSN, the Username for the user who gained access or was denied as well as the Timestamp when the query was sent. The last two columns contain additional information on the login. The Role includes information whether the username entered is the WSN owner or a guest. The Access Type shows whether the login was successful or got rejected.

Login Track				
WsnId	Username	Timestamp	Role	Access Type

Figure 3.5: Table Login Track

3.3 Transparency

In the assignment the filtering option was implemented with multiple filtering options to gain transparency for the data owner. Therefore these options include WSN Users according to the activated WSN, a start date and an end date and finally the option to chose between Push and Pull. Concerning the extension of the filtering page a closer look is taken on three different parts. The first section will take care of the 100% transparency request. In a second step a more fine grained filtering option is discussed. The last point talks about the possibilities for the output of the filtered data.

3.3.1 Transparency inside the filtering option

The transparency question was already addressed in the work of my assignment. As a result a transparency of 75% could be acquired in other words the filtering option allows the owner to filter the currently selected WSN in CoMaDa for users ,date and push/pull. The restriction lies within the pushed data, which only shows the owners data who pushes. Therefore to gain full transparency in the filtering, exactly those pushes from users need to be included in the data. This can only be achieved by checking the users within a WSN about their rights. This table is only available in WebMaDa. Therefore the mentioned pull interface in the related works needs to be extended to obtain the necessary data.

3.3.2 Filtering extension

As in figure 3.6 can be witnessed, a WSN owner can select the WSN users, the date and the push/pull option.

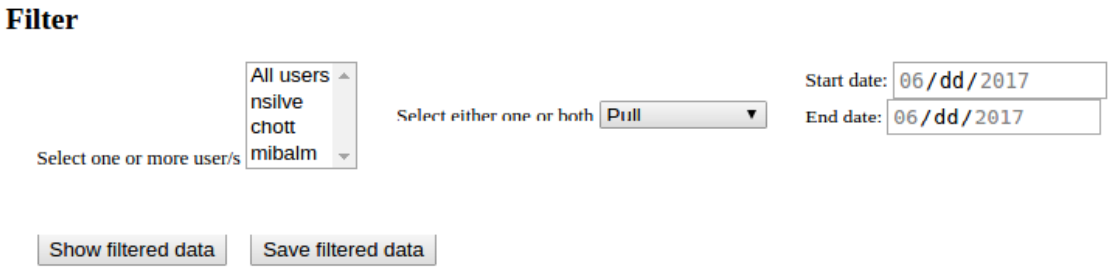
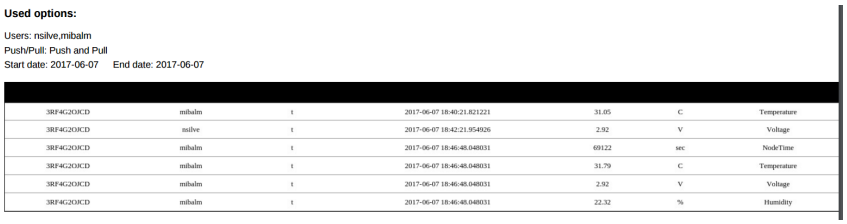


Figure 3.6: Filter possibilities

The filtering extension addresses the problem that a WSN owner is not able to limit the filter options enough especially only with date. As a whole day of data can be a large number of sensor data. Therefore the filtering option will be extended to date and time, which allows the WSN owner to select not only the date but also hours and minutes.

3.3.3 Output options

The existing implementation allows a WSN owner to download a PDF-File of the currently selected filtered data. The file is then saved in the download folder under the name of the WSN with the timestamp. This can be seen in figure 3.7.



Used options:

Users: nsilve,mibalm
Push/Pull: Push and Pull
Start date: 2017-06-07 End date: 2017-06-07

3RF4G2OJCD	mibalm	t	2017-06-07 18:40:21.821221	31.05	C	Temperature
3RF4G2OJCD	nsilve	t	2017-06-07 18:42:21.954926	2.92	V	Voltage
3RF4G2OJCD	mibalm	t	2017-06-07 18:46:48.048031	69122	sec	NodeTime
3RF4G2OJCD	mibalm	t	2017-06-07 18:46:48.048031	31.79	C	Temperature
3RF4G2OJCD	mibalm	t	2017-06-07 18:46:48.048031	2.92	V	Voltage
3RF4G2OJCD	mibalm	t	2017-06-07 18:46:48.048031	22.32	%	Humidity

Figure 3.7: Example of PDF file

For the downloaded data the same security requirements have to be met. As only WSN owner should be able to read the data the security issue can be resolved by encrypting the data and providing it with a password. As the password should make sure only the owner of the WSN can open the file, it makes sense to use the owners password stored in the database.

Chapter 4

Implementation

This chapter documents the implementation of the previous discussed design decisions. It is structured in the following way. The first section shows the database adjustments to increase the traceability. The next section shows the added features in the filtering option namely, more accurate filter options, the log of the selected filtering and the encryption of the output file. The last chapter shows in detail the implementation of the login process.

4.1 Database Adjustments

The tables in the design decisions were implemented as described. Figure 4.1 and Figure 4.2 show these tables in PostGresql respectively. The columns selected proofed to be accurate for their purpose. The detailed table insertion will be discussed in the previous chapters. Therefore the _filter_log table is picked up again in the subsection Filtering_Log in the chapter Filter extensions whereas table _logintrack is picked up again in the login process within chapter Login.

Exactly these tables have also been created in the WebMaDa database as all the features shall be carried over to WebMaDa in a later phase.

```
-- Table: public._filter_log

-- DROP TABLE public._filter_log;

CREATE TABLE public._filter_log
(
  wsnowner character varying(50) NOT NULL,
  tstamp_filter timestamp without time zone NOT NULL DEFAULT now(),
  fuser character varying(50) NOT NULL,
  pptype character varying(50) NOT NULL,
  sdate timestamp without time zone NOT NULL,
  edate timestamp without time zone NOT NULL,
  wsnid character varying(20)
)
WITH (
  OIDS=FALSE
);
ALTER TABLE public._filter_log
  OWNER TO wsnadmin;
```

Figure 4.1: PostGres filter log Table

```
-- Table: public._logintrack

-- DROP TABLE public._logintrack;

CREATE TABLE public._logintrack
(
  wsnid character varying(20) NOT NULL,
  username character varying(50) NOT NULL,
  "timestamp" timestamp without time zone NOT NULL DEFAULT now(),
  role character varying(50),
  access_type character varying(50)
)
WITH (
  OIDS=FALSE
);
ALTER TABLE public._logintrack
  OWNER TO wsnadmin;
```

Figure 4.2: PostGres logintrack Table

4.2 Filter extensions

This chapter will go over the added features in the Filtering page in a process like manner. Therefore the first section presents the more accurate version from Start date and End date together with Start time and End time. The second section goes over the implementation of the filter log and the final section shortly presents the password protected output file.

4.2.1 Date and Time

The date and time was a request as filtering just for the date was too inaccurate. As an active WSN pushes data around every five seconds by default. A search for data in a time range of one day is not meaningful enough. Figure 4.3 shows such a filtered datatable. It can be witnessed that the search for only one person and at one day gives 1872 entries.

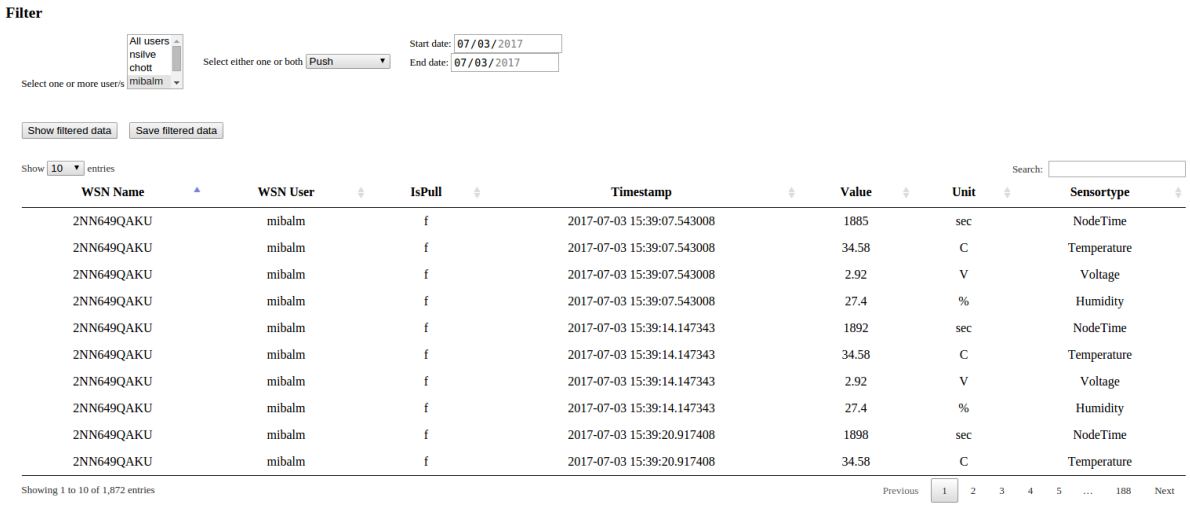


Figure 4.3: Filtered data with date

The date picker has automatically the date preselected the first time data was received in the Start date as well as the last time data was received in the End date. This predefined data comes from the function ”ArrayList<String> getDate()” in the DBAccessPostgresql.java file. Exactly this function is illustrated in Figure 4.5. The updated function has new SQL Queries for not just the date but a new time format as recognizable in the SQL statement on line 364 and 365. It selects the maximum timestamp and minimum timestamp in a ”YYYY-MM-DD HH24:MI” format. Those timestamps are then added to the result arraylist on line 370 and 373. The result of this function is sent to an ajax function in the WSNHTTPIndexController.java file. This serves as communication point between the frontend and the backend of CoMaDa. From there on the data is forwarded to the http.get request in the Filterwidget.js file displayed in Figure 4.5.

The updates in this function concern the new date format forwarded. Therefore in line 56-59 and line 61-64 the format is split into two segments the date and the time. Those values are added to the new corresponding datetime element throughout the block on line

```
353     @Override
354     public ArrayList<String> getDate() {
355         Connection connection = connect(host, user, password);
356         PreparedStatement stmnt = null;
357         PreparedStatement stmnt2 = null;
358         ArrayList<String> result = new ArrayList<>();
359         if (connection != null){
360             try{
361
362                 // get max/min date and max/min time on this date
363
364                 stmnt = connection.prepareStatement("SELECT to_char(MAX(timestamp), 'YYYY-MM-DD HH24:MI') FROM _response");
365                 stmnt2 = connection.prepareStatement("SELECT to_char(MIN(timestamp), 'YYYY-MM-DD HH24:MI') FROM _response");
366
367                 ResultSet un = stmnt.executeQuery();
368                 ResultSet un2 = stmnt2.executeQuery();
369                 while(un.next()){
370                     result.add(un.getString("to_char"));
371                 }
372                 while(un2.next()){
373                     result.add(un2.getString("to_char"));
374                 }
375
376             }catch (SQLException e) {
377                 System.out.println("Statement creation Failed!");
378                 e.printStackTrace();
379             }finally {
380                 try {
381                     if (stmnt != null){
382                         stmnt.close();
383                     }
384                     connection.close();
385                 }catch (SQLException e) {
386                     System.out.println("Couldn't close connection!");
387                     e.printStackTrace();
388                 }
389             }
390             System.out.println(result);
391         } return result;
392     }
```

Figure 4.4: getdate in DBAccessPostgresql.java

71 until line 85. The result of this function is represented in Figure 4.6 which shows the Start date/time and the End date/time.

```
52          $http.get('/index/getdate').then(function(data) {
53
54              var dates = data.data;
55
56              var max = dates[0];
57              var maxstamp = max.split(" ");
58              var maxdate = maxstamp[0];
59              var maxtime = maxstamp[1];
60
61              var min = dates[1];
62              var minstamp = min.split(" ");
63              var mindate = minstamp[0];
64              var mintime = minstamp[1];
65
66
67              /* alert("date: "+ maxdate + " time: " + maxtime); */
68
69              /* add values to html object date/time   */
70
71              document.getElementById("startDate").min = mindate;
72              document.getElementById("startDate").max = maxdate;
73              document.getElementById("startDate").value = mindate;
74
75              document.getElementById("endDate").min = mindate;
76              document.getElementById("endDate").max = maxdate;
77              document.getElementById("endDate").value = maxdate;
78
79              document.getElementById("startTime").min = mintime;
80              document.getElementById("startTime").max = maxtime;
81              document.getElementById("startTime").value = mintime;
82
83              document.getElementById("endTime").min = mintime;
84              document.getElementById("endTime").max = maxtime;
85              document.getElementById("endTime").value = maxtime;
86          });
```

Figure 4.5: getdate in Filterwidget.js

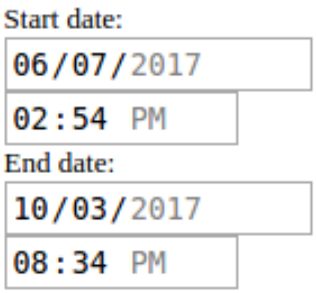


Figure 4.6: Start date/time and End date/time

The result of the new date/time in the Filtering option is illustrated in Figure 4.7. The time range of the Start date and End date only adds up to one minute which results in a datatable set of only 32 entries and a more fine grained possibility to search for certain data.

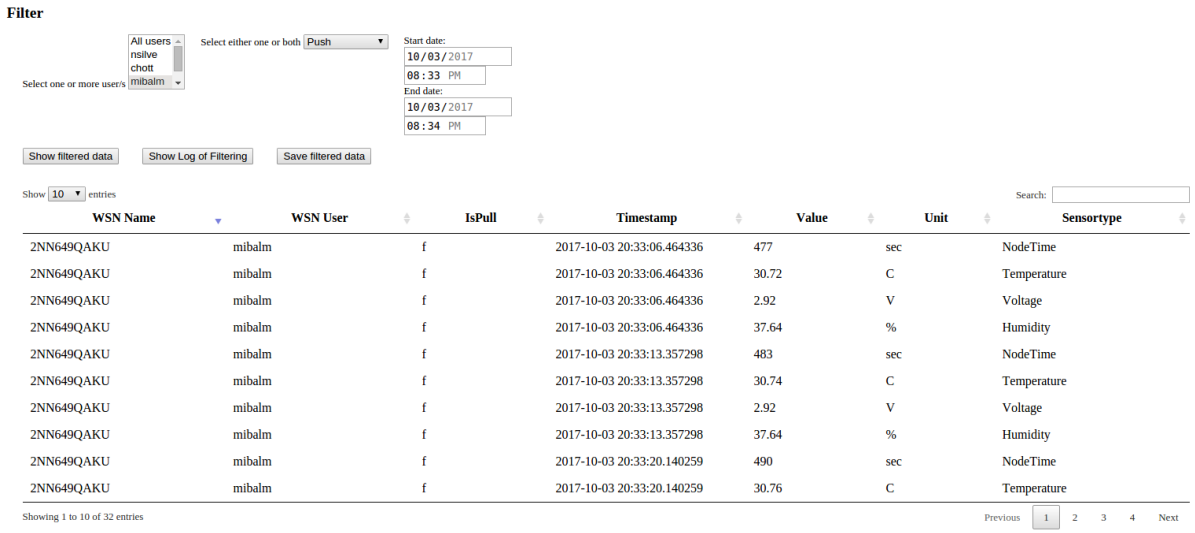


Figure 4.7: Date/time example

4.2.2 Show filtered data

The change from date to date/time changed the request on the current database. Therefore an adjustment has been made in displaying the filtered data. As the filter log table uses the same data from the selected filter options this task is integrated in the implementation on how to display filtered data. The workflow is identical to the one described in section Date and Time. First a request with help of an ajax call is done towards the WSNHTTPIndexController.java file where the request is forwarded to DBAccessPostgresql.java file for the actual data manipulation and database access. In Figure 4.8 the snippet of the additionally added code is shown. For the query on the filtered data it was possible to directly us the statements from the passed arguments, the only simple change in the query from datatype date to timestamp, which is not illustrated as it is trivial. On the other hand it shown how to add selected filter options to the filter_log table. In a first step the data to be inserted had to be prepared. Therefore from line 529 up to line 538 the current timestamp is created as well the arguments Start date and End date are parsed into a timestamp to match the datatype in the filtered_log table. The part from line 543 up to line 555 and from line 558 up to line 567 respectively shows the query for the insertion of the filtered options. The arguments listed are put into the insert function wsn_add_filter_log. Exactly this stored procedure is illustrated in Figure 4.9. For every column a value is assigned to insert the specific data. With this additional code snippet on pressing the ”Show filtered data” button the selected filter options are also successfully logged.

```
522                 //Startdate and Enddate for Query
523                 String sd = statements.get(2);
524                 String ed = statements.get(3);
525                 stmnt.setString(1, sd);
526                 stmnt.setString(2, ed);
527
528                 //timestamp
529                 Date date = new Date();
530                 long time = date.getTime();
531                 Timestamp current = new Timestamp(time);
532
533                 //Change sd / ed to Timestamp
534                String sdate = sd + ":00";
535                String edate = ed + ":00";
536                System.out.println(sdate);
537                java.sql.Timestamp sqlsdate = java.sql.Timestamp.valueOf(sdate);
538                java.sql.Timestamp sqledate = java.sql.Timestamp.valueOf(edate);
539
540                 // Add Filter data to Filter_Log table
541                 PreparedStatement stmnt_filter_log = null;
542
543                 if (users.contains("All users") == false) {
544                     for (int i = 0 ; i < numuser; i++) {
545                         String stmnt_user = splitArray[i];
546                         stmnt_filter_log = connection.prepareStatement("SELECT wsn_add_filter_log(?,?,?,?,?,?,?);");
547                         stmnt_filter_log.setString(1,owner);
548                         stmnt_filter_log.setTimestamp(2, current);
549                         stmnt_filter_log.setString(3, stmnt_user);
550                         stmnt_filter_log.setString(4, pp);
551                         stmnt_filter_log.setTimestamp(5, sqlsdate);
552                         stmnt_filter_log.setTimestamp(6, sqledate);
553                         stmnt_filter_log.setString(7, wsnid);
554
555                         stmnt_filter_log.executeQuery();
556                     }
557                 } else {
558                     String all = "All users";
559                     stmnt_filter_log = connection.prepareStatement("SELECT wsn_add_filter_log(?,?,?,?,?,?,?);");
560                     stmnt_filter_log.setString(1,owner);
561                     stmnt_filter_log.setTimestamp(2, current);
562                     stmnt_filter_log.setString(3, all);
563                     stmnt_filter_log.setString(4, pp);
564                     stmnt_filter_log.setTimestamp(5, sqlsdate);
565                     stmnt_filter_log.setTimestamp(6, sqledate);
566                     stmnt_filter_log.setString(7, wsnid);
567                     stmnt_filter_log.executeQuery();
568                 }
569
```

Figure 4.8: getfilteredData in DBPostGresql.java

```
-- Function: public.wsn_add_filter_log(character varying, timestamp without time zone, character varying, character varying, timestamp without time zone, timestamp without time zo

-- DROP FUNCTION public.wsn_add_filter_log(character varying, timestamp without time zone, character varying, character varying, timestamp without time zone, timestamp without tim

CREATE OR REPLACE FUNCTION public.wsn_add_filter_log(
    p_owner character varying,
    p_timestamp timestamp without time zone,
    p_all character varying,
    p_pp character varying,
    p_sd timestamp without time zone,
    p_ed timestamp without time zone,
    p_wsnid character varying)
  RETURNS void AS
$BODY$BEGIN
        INSERT INTO _filter_log(wsnowner, tstamp_filter, fuser, pptype, sdate, edate, wsnid)  VALUES (p_owner, p_timestamp, p_all, p_pp, p_sd, p_ed, p_wsnid);

END$BODY$
  LANGUAGE plpgsql VOLATILE
  COST 100;
ALTER FUNCTION public.wsn_add_filter_log(character varying, timestamp without time zone, character varying, character varying, timestamp without time zone, timestamp without time
  OWNER TO postgres;
```

Figure 4.9: stored procedure wsn_add_filter_log

4.2.3 Show the filter log

This subsection explains the implementation of the feature to display the content of the table filter_log. The start is the retrieval of the data from the table. This is done in the getFilterLog function in the DBAccessPostgresql.java file as shown in Figure 4.10. The data is retrieved with the query on line 437. The data is then processed and every element is added to the Arraylist ”res”. This Arraylist is then sent to the ajax module filter_log in the WSNHTTPIndexController.java file shown in Figure 4.11. This ajax function forwards the data to the filterWidget.js file in the front end. The part responsible within filterWidget.js is illustrated in Figure 4.12. The actual initialization of the above described workflow can be found on line 91 with the http.get request after the ”Show Log of Filtering” button was pressed. The returning data is then added row by row on line 100 and pushed into a datatable for each column on line 106 to 112. This datable is then displayed on the filtering page. An example of such a datatable is shown in Figure 4.13.

```
426     @Override
427     public ArrayList<Map<String, String>> getFilterLog() {
428             Connection connection = connect(host, user, password);
429             PreparedStatement stmnt = null;
430
431             ArrayList<Map<String,String>> res = new ArrayList<>();
432             if (connection != null){
433                 try{
434
435
436                     // Get Filter data from the Filter_Log table
437                     stmnt = connection.prepareStatement("SELECT * FROM _filter_log");
438
439                     ResultSet rs = stmnt.executeQuery();
440                     ResultSetMetaData rsmt = rs.getMetaData();
441                     int columnCount = rsmt.getColumnCount();
442                     while(rs.next()){
443                         Map<String,String> resultMap = new HashMap<>();
444
445                         for(int i = 1 ; i <= columnCount; i++){
446                           resultMap.put(rsmt.getColumnName(i).toLowerCase(),rs.getString(i));
447
448                         }
449                         res.add(resultMap);
450                     }
451                 }catch (SQLException e) {
452                     System.out.println("Statement creation Failed!");
453                     e.printStackTrace();
454                 }finally {
455                     try {
456                         if (stmnt != null){
457                             stmnt.close();
458                         }
459                         connection.close();
460                     }catch (SQLException e) {
461                         System.out.println("Couldn't close connection!");
462                         e.printStackTrace();
463                     }
464                 }
465             }
466             return res;
467         }
```

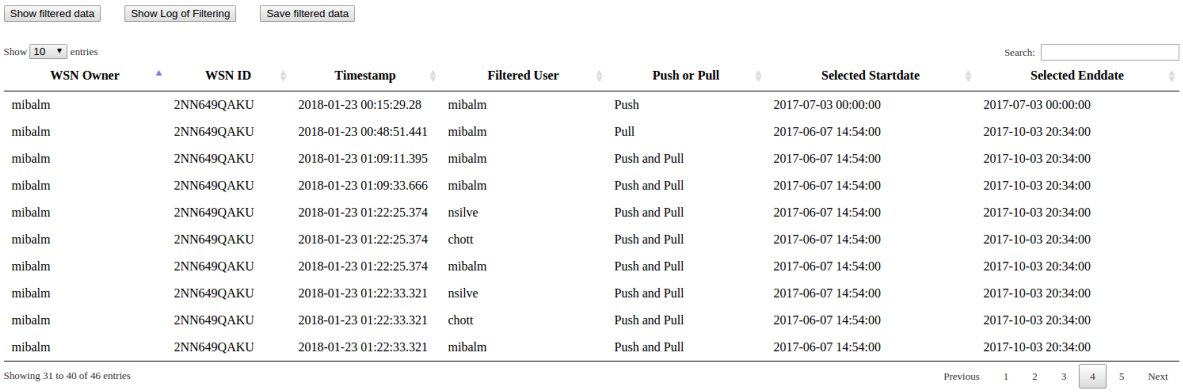
Figure 4.10: getfilterlog in DBPostGresql.java

```
/**
 * ajax update action, returns the filter_log data
 * @param request
 * @param response
 *
 * @author Michael Balmer
 */

public void getfilterlogAction(HTTPRequest request, HTTPResponse response) {
    Map<String,Object> jsonResult = new HashMap<String,Object>();

    response.body = JSONValue.toJSONString(jsonResult).getBytes();

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

    jsonResult.put("data", dbAccess.getFilterLog());
    response.body = JSONValue.toJSONString(jsonResult).getBytes();
}
```

Figure 4.11: get filterlog ajax call

```
88
89          document.getElementById('showLogBtn').addEventListener('click', function() {
90
91               $http.get('/index/getFilterLog').then(function(data) {
92          if (data.data.data != 0) {
93              document.getElementById('filteredData').style.display = "none";
94              $('#filteredData').parents('div.dataTables_wrapper').first().hide();
95              $('#filterLogData').parents('div.dataTables_wrapper').first().show();
96              document.getElementById('noData').style.visibility = "hidden";
97              document.getElementById('filterLogData').style.display = "";
98              $("#filterLogData tr:gt(0)").remove();
99              var outter = [];
100                         for (var key in data.data.data) {
101                                     var entry = data.data.data[key];
102                                     var keys = Object.keys(entry);
103                                     var values = data.data.data[key];
104
105                                     var inner = [];
106                                     inner.push(data.data.data[key].wsnowner);
107                                     inner.push(data.data.data[key].wsnid);
108                                     inner.push(data.data.data[key].tstamp_filter);
109                                     inner.push(data.data.data[key].fuser);
110                                     inner.push(data.data.data[key].pptype);
111                                     inner.push(data.data.data[key].sdate);
112                                     inner.push(data.data.data[key].edate);
113                                     outter.push(inner);
114
115                         }
116         var table = $('#filterLogData').DataTable();
117         table.clear().rows.add(outter).draw();
118
119
120         }else{
121             document.getElementById('filterLogData').style.display = "none";
122             document.getElementById('filteredData').style.display = "none";
123             document.getElementById('noData').style.visibility = "visible";
124         }
```

Figure 4.12: Show filter_log javascript



Show filtered data | Show Log of Filtering | Save filtered data

Show 10 entries — Search:

WSN Owner	WSN ID	Timestamp	Filtered User	Push or Pull	Selected Startdate	Selected Enddate
mibalm	2NN649QAKU	2018-01-23 00:15:29.28	mibalm	Push	2017-07-03 00:00:00	2017-07-03 00:00:00
mibalm	2NN649QAKU	2018-01-23 00:48:51.441	mibalm	Pull	2017-06-07 14:54:00	2017-10-03 20:34:00
mibalm	2NN649QAKU	2018-01-23 01:09:11.395	mibalm	Push and Pull	2017-06-07 14:54:00	2017-10-03 20:34:00
mibalm	2NN649QAKU	2018-01-23 01:09:33.666	mibalm	Push and Pull	2017-06-07 14:54:00	2017-10-03 20:34:00
mibalm	2NN649QAKU	2018-01-23 01:22:25.374	nsilve	Push and Pull	2017-06-07 14:54:00	2017-10-03 20:34:00
mibalm	2NN649QAKU	2018-01-23 01:22:25.374	chott	Push and Pull	2017-06-07 14:54:00	2017-10-03 20:34:00
mibalm	2NN649QAKU	2018-01-23 01:22:25.374	mibalm	Push and Pull	2017-06-07 14:54:00	2017-10-03 20:34:00
mibalm	2NN649QAKU	2018-01-23 01:22:33.321	nsilve	Push and Pull	2017-06-07 14:54:00	2017-10-03 20:34:00
mibalm	2NN649QAKU	2018-01-23 01:22:33.321	chott	Push and Pull	2017-06-07 14:54:00	2017-10-03 20:34:00
mibalm	2NN649QAKU	2018-01-23 01:22:33.321	mibalm	Push and Pull	2017-06-07 14:54:00	2017-10-03 20:34:00

Showing 31 to 40 of 46 entries — Previous 1 2 3 4 5 Next

Figure 4.13: Datatable log of filtering

4.3 Login

The new Login functionality is the main security feature added. In this section the files involved are revisited and the implementation is explained. For a better understanding this section is divided in two parts. The first one is the Client side which shows the usage of the upload interface on CoMaDa. The second part shows the data processing on the Server side in this context the WebMaDa part of the upload interface.

4.3.1 Client side

In this subsection the implementation of the CoMaDa part to upload the login credentials is described. The entry point of the workflow is the Login screen shown in Figure 4.14.

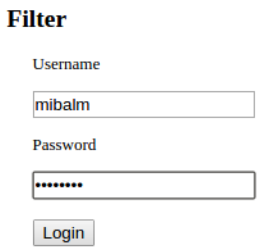


Figure 4.14: Login Screen

The code of the Login screen is displayed in Figure 4.15. The code snippet shows the inbuilt javascript part of the filter.html. On line 20 and 21 we read the input of the document elements username and password. From line 24 to 31 we make a simple ajax call to the getloginlog handler in WSNHTTPIndexController.java. The code after the ajax call will be discussed in a later stage.

The Ajax call handler is illustrated in Figure 4.16. On line 567 and 568 the given arguments are read and converted to type String and forwarded as arguments of the function LoginLog on line 571. Function LoginLog also resides in the DBAccessPostgresql.java file.

The function is shown in Figure 4.17. This is where the function is called to upload the credentials to WebMaDa for the login check. To make this work the upload interface from Claudio Anliker [10] had to be extended. An upload can be triggered with a call to a BasicUploaderStarter.java method. This happens on the line 307.

As mentioned to make this work the upload interface had to be extended. The first file is the LoginUploadFacade.java its a simple method to set the username and password within it. The idea behind the file is to store only the important data for the upload if more data would be sent than necessary.

The next file shown in Figure 4.19 is a specified variant of a DefaultUploader It is needed to manipulate the UploadMessageType and the received arguments.

```
14      <script type="text/javascript">
15
16
17          $(document).ready(function(){
18              $('#loginButton').click(function(){
19              <!--location.href = "/index/widgets/filter/filterWidget.html" -->
20              var username = document.getElementById("username").value;
21              var password = document.getElementById("password").value;
22
23
24              $.ajax({
25                  "url": "/index/getloginlog",
26                  "data": {username, password},
27                  "dataType": "text",
28                  success: function(data){
29                  },
30                  error: function(){},
31              });
32
33
34
35              var login = data;
36
37
38
39                  if (login == true ) {
40
41                      var x = document.getElementById("widgetFilter");
42                          if (x.style.display === "none") {
43                              x.style.display = "block";
44                          } else {
45                              x.style.display = "none";
46                          }
47                      var y = document.getElementById("login");
48                          if (y.style.display === "none") {
49                              y.style.display = "block";
50                          } else {
51                              y.style.display = "none";
52                          }
53                  }else{
54
55                      alert("Username or password wrong!");
56                  }
57
58              });
59          });
```

Figure 4.15: filter.html

In Figure 4.20 the function uploadLogin on line 103 in the BasicUploadStarter.java file is added. This handler is used to instantiate the upload of the login data.

This functionality is needed to upload the credentials. Therefore in Figure 4.17 on line 296 a LoginUploadFacade is instantiated at first. From line 297 up to line 301 is implemented

```
555     /**
556      * ajax update action, check login data and log it.
557      * @param request
558      * @param response
559      *
560      * @author Michael Balmer
561      */
562
563     public void getloginlogAction(HTTPRequest request, HTTPResponse response) {
564         boolean jsonResult;
565
566
567         String username = request.arguments.get("username").toString();
568         String password = request.arguments.get("password").toString();
569
570         IDBAccessLayer dbAccess = new DBAccessPostgresql(this.getServerModule().app().getProperties());
571         jsonResult = dbAccess.LoginLog(username, password);
572         response.body = JSONValue.toJSONString(jsonResult).getBytes();
573     }
```

Figure 4.16: Ajax call loginlog

for an additional authentication to the upload interface. In line 305 the dataUploader is instantiated as a BasicUploadStarter with the secure connection parameters. Finally on line 307 the login credentials are sent to the server side of the upload interface.

```
289    @Override
290    public boolean LoginLog(String username, String pw) {
291        Connection connection = connect(host, user, password);
292        PreparedStatement stmnt = null;
293        boolean res = true;
294        String suc;
295
296        LoginUploadFacade login_cred = new LoginUploadFacade(username, pw);
297        cosm = new CosmAPI("tumi8", "c8R0rBphAZ2b31khePtdK0DSOrqSAKxZTW5sMkZKZGhqMD0g");
298        IUploaderFactory factory = new BasicUploaderFactory(appPro, cosm);
299        IHttpConnector httpConn = new BasicAuthenticator(new HttpConnector(appPro),
300                new UploadCredentials(appPro.getProperty("upload.password"),
301                        appPro.getProperty("wsn.id")));
302        //UploadLogin login_data = new UploadLogin(username, pw);
303        //IHttpConnector httpConn = new BasicLogin(new HttpConnector(appPro), login_data);
304
305        dataUploader = new BasicUploadStarter(factory, httpConn);
306
307        dataUploader.uploadLogin(login_cred);
308
309        if(response == "S_AUTH_SUCCESS" ){
310            res = true;
311            suc = "sucess";
312
313        }else{
314
315            res = false;
316            suc = "denied";
317        }
318
319
320        if (connection != null){
321            try{
322                Date date = new Date();
323                long time = date.getTime();
324                Timestamp current = new Timestamp(time);
325
326                stmnt = connection.prepareStatement("SELECT wsn_add_logintrack(?,?,?,?,?);");
327                stmnt.setString(1, wsnid);
328                stmnt.setString(2, username);
329                stmnt.setTimestamp(3, current);
330                stmnt.setString(4, "");
331                stmnt.setString(5, suc);
332
333                stmnt.executeQuery();
```

Figure 4.17: Loginlog in PostGres.java

```java
package de.tum.in.net.WSNDataFramework.Modules.Web.Upload.Facade;


/**
 * Facade-like helper class to encapsulate Login data for the handling
 * at the upload interface.
 *
 * @author Michael
 *
 */
public class LoginUploadFacade {

    private String username;
    private String password;


    public LoginUploadFacade(String username, String password){
        this.password = password;
        this.username = username;
    }

    protected String getPassword() {
        return password;
    }

    protected String getUsername() {
        return username;
    }
}

```

Figure 4.18: LoginUploadFacade

```java
package de.tum.in.net.WSNDataFramework.Modules.Web.Upload.Factory;

import de.tum.in.net.WSNDataFramework.Modules.Web.Upload.UploadMessage;


/**
 * Uploader to handle login uploads.
 *
 * @author Michael
 *
 */
public class LoginUploader extends DefaultUploader{

    @Override
    public UploadMessage formMessage(Object input) {
        if(input instanceof LoginUploadFacade){
            return new UploadMessage(UploadMessageType.C_LOGIN, input);
        }
        System.out.println("no login data");
        return null;
    }
}
```

Figure 4.19: LoginUploader in Factory

```java
100     /**
101      * This handler initiates the upload of the login data.
102      */
103     public void uploadLogin(LoginUploadFacade content) {
104         UploadMessage toUpload = LoginUploader.formMessage(content);
105         LoginUploader.upload(toUpload, httpConn);
106     }
107 }
```

Figure 4.20: Function uploadLogin in BasicUploadStarter

4.3.2 Server side

When the client side sends a message. The incoming message is handled by the index.php file in the upload folder. In Figure 4.21 the index.php is presented. In the beginning of the file the multiple different handlers for the messagetypes are instantiated. The triage for the different handlers happens on line 50 and 51, where the messagetype is checked and forwarded to the correct Handler file. In case of messagetype ”C_LOGIN”, the LoginHandler.php is started.

```
18 $loginhandler = new LoginHandler;
19 $handlers['C_LOGIN'] = $loginhandler;
20 $handlers['C_DATA'] = $datahandler;
21 $handlers['C_NODE'] = $nodehandler;
22 $handlers['C_SENSOR'] = $sensorhandler;
23 $handlers['C_STREAM'] = new StreamHandler;
24 $handlers['C_RAW_PACKAGE'] = new RawPackageHandler;
25
26 $uploadPdo = null;
27 if(!$auth->ensurePost()){
28     return;
29 }
30
31 if(!$auth->handle($_POST['message'], $uploadPdo))
32 {
33     return;
34 }
35
36 if($uploadPdo === null){
37     error_log('upload pdo is null');
38     return;
39 }
40
41
42 $message = json_decode($_POST['message'], true);
43 if(!isset($message['content']) || !isset($message['messageType'])){
44     sendResponse('S_ERROR', 'Decoding JSON string failed.');
45     return;
46 }
47
48 $type = $message['messageType'];
49
50 if(isset($handlers[$type])){
51     $handlers[$type]->handle($message, $uploadPdo);
52 }
53
54 function sendResponse($messageType, $content)
55 {
56     $message = array(
57         'messageType' => $messageType,
58         'content' => $content,
59     );
60     $response = json_encode($message);
61     echo $response;
62 }
```

Figure 4.21: Upload index.php

The LoginHandler.php implementation can be seen in Figure 4.22 the incoming message is first checked on several parameters to make sure, the correct messagetype has been provided as well as the content is complete. Afterwards the passwordhash of the provided user is written into a variable on Line 38. On line 40 starts the credential check on which s outcome a S_AUTH_ERROR or a S_AUTH_SUCCESS is returned.

```php
class LoginHandler extends AbstractHandler
{

    public function handle($message, &$uploadPdo)
    {
        if($message['messageType'] !== C_LOGIN){
            $this->sendResponse(S_AUTH_MISS, 'Expect a C_LOGIN message.');
            return false;
        } else {
            $content = $message['content'];
            if(!(isset($content['password']))
                || !(isset($content['username']))){
                $this->sendResponse(S_AUTH_MISS, 'Incomplete C_LOGIN provided.');
                return false;
            }

            $pass = $content['password'];
            $username = $content['username'];

            /* Get a user's data from database to compare the provided
            * password with the hash stored.
            */
            function getUserFromDatabase($username){
                $getUserHandle = Config::$loginPdo->prepare('CALL GetPasswordHash(?);'
                $getUserHandle->bindValue(1, $username);
                $getUserHandle->execute();
                $user = $getUserHandle->fetch(PDO::FETCH_ASSOC);
                $getUserHandle->closeCursor();
                return $user;
            }

            //check credentials
            try{
                $pdo = $this->createTemporaryPdo();
                $userFromDb = getUserFromDatabase($username);

                if (password_verify($pass, $userFromDb['password'])) {

                } else {
                    $this->sendResponse(S_AUTH_ERROR, 'Wrong password: ' . $pass);
                    return;
                }
            } catch (PDOException $e) {
                $this->sendResponse(S_AUTH_ERROR, $e->getMessage());
                return false;
            }

            $uploadPdo = $pdo;

            $this->sendResponse(S_AUTH_SUCCESS, 'Credential check successful');
            return true;
        }
```

Figure 4.22: LoginHandler in WebMaDa

4.3.3 Logintrack

The Logintrack implementation is done in the LoginLog function which we earlier visited. Depending on the outcome of the logincheck the variable on line 331 is either way success or denied. The query is handled by a stored procedure named wsn_add_logintrack. In Figure 4.23 exactly this procedure is presented. For each variable exists an argument for the INSERT command.

```
320         if (connection != null){
321             try{
322                 Date date = new Date();
323                 long time = date.getTime();
324                 Timestamp current = new Timestamp(time);
325 
326                 stmnt = connection.prepareStatement("SELECT wsn_add_logintrack(?,?,?,?,?);");
327                 stmnt.setString(1, wsnid);
328                 stmnt.setString(2, username);
329                 stmnt.setTimestamp(3, current);
330                 stmnt.setString(4, "");
331                 stmnt.setString(5, suc);
332 
333                 stmnt.executeQuery();
334             }catch (SQLException e) {
335                 System.out.println("Statement creation Failed!");
336                 e.printStackTrace();
337             }finally {
338                 try {
339                     if (stmnt != null){
340                         stmnt.close();
341                     }
342                     connection.close();
343                 }catch (SQLException e) {
344                     System.out.println("Couldn't close connection!");
345                     e.printStackTrace();
346                 }
347             }
348         }
349         return res;
350     }
```

Figure 4.23: Logintrack in LoginLog function in DBPostgre.java

```
-- Function: public.wsn_add_logintrack(character varying, character varying, timestamp without time zone, character varying, character varying)

-- DROP FUNCTION public.wsn_add_logintrack(character varying, character varying, timestamp without time zone, character varying, character varying);

CREATE OR REPLACE FUNCTION public.wsn_add_logintrack(
    p_wsnid character varying,
    p_username character varying,
    p_timestamp timestamp without time zone,
    p_role character varying,
    p_acc_type character varying)
  RETURNS void AS
$BODY$BEGIN
        INSERT INTO _logintrack(wsnid, username, timestamp, role, access_type)  VALUES (p_wsnid, p_username, p_timestamp, p_role, p_acc_type);

END$BODY$
  LANGUAGE plpgsql VOLATILE
  COST 100;
ALTER FUNCTION public.wsn_add_logintrack(character varying, character varying, timestamp without time zone, character varying, character varying)
  OWNER TO postgres;
```

Figure 4.24: Stored procedure wsn_add_logintrack

Chapter 5

Evaluation

5.1 Proof of Operability

The evaluation is done by a Proof of Operability as it is best fit to test the running system. The testing includes the Login, the time/date filtering option, the show log of filtering functionality and the logintrack in the background.

Testsetting: username: mibalm pw: korrekt
Expected: Console S_AUTH_SUCCESS
Result: Correct

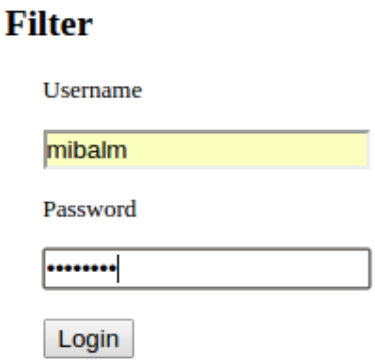


Figure 5.1: Login Test 1

```
Authenticate at upload interface...
Response: {"messageType":"S_AUTH_SUCCESS","content":"Authentication successful"}
Authentication at upload interface was successful!
Response: {"messageType":"S_AUTH_SUCCESS","content":"Credential check successful"}
```

Figure 5.2: Console output Test 1

Testsetting: username: mibalm pw: falsch
Expected: Console S_AUTH_ERROR
Result: Correct

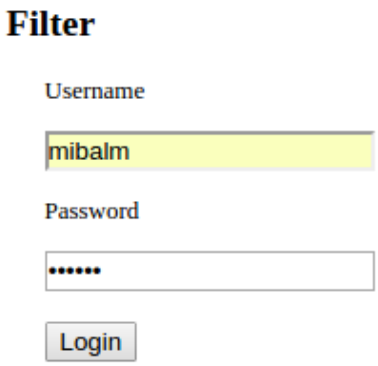


Figure 5.3: Login Test 1

```
Authentication at upload interface was successful!
Response: {"messageType":"S_AUTH_ERROR","content":"Wrong password: falsch"}
```

Figure 5.4: console output Test 2

Testsetting: Start Date: 10/03/2017 End Date: 10/03/2017
Start Time: 08:33 PM End Time: 08:34 PM
Exected: Correct data

Result : Correct

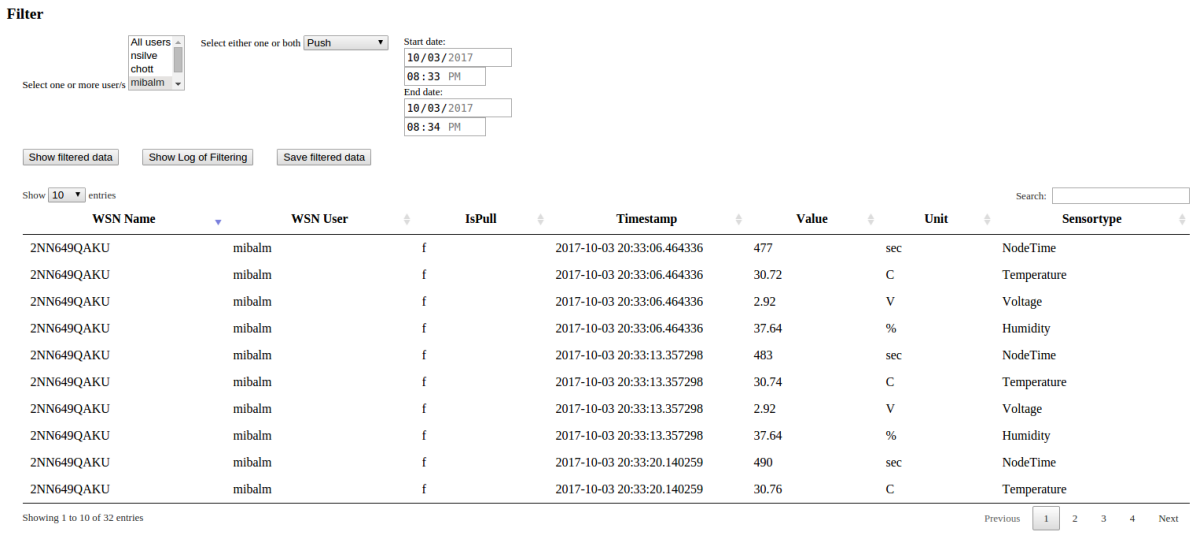


Figure 5.5: Date/Time Test

Testsetting:
Expected: Display Log of Filtering and show previous time/date filtering
Result: Correct

Show 10 entries Search:

WSN Owner	WSN ID	Timestamp	Filtered User	Push or Pull	Selected Startdate	Selected Enddate
mibalm	2NN649QAKU	2018-01-23 01:27:45.818	All users	Push and Pull	2017-06-07 14:54:00	2017-10-03 20:34:00
mibalm	2NN649QAKU	2018-01-23 01:28:02.88	mibalm	Push	2017-06-07 14:54:00	2017-10-03 20:34:00
mibalm	2NN649QAKU	2018-01-23 01:29:46.036	mibalm	Push	2017-06-07 14:54:00	2017-10-03 20:34:00
mibalm	2NN649QAKU	2018-01-23 01:31:00.89	All users	Pull	2017-06-07 14:54:00	2017-10-03 20:34:00
mibalm	2NN649QAKU	2018-01-23 01:31:19.254	mibalm	Push	2017-06-07 14:54:00	2017-10-03 20:34:00
mibalm	2NN649QAKU	2018-02-01 04:54:11.551	mibalm	Push	2017-10-03 20:33:00	2017-10-03 20:34:00
mibalm	2NN649QAKU	2018-02-01 20:55:52.894	mibalm	Push	2017-06-07 14:54:00	2017-06-07 14:56:00
mibalm	2NN649QAKU	2018-02-01 20:56:03.733	All users	Push and Pull	2017-06-07 14:54:00	2017-06-07 14:56:00

Showing 41 to 48 of 48 entries Previous 1 2 3 4 5 Next

Figure 5.6: filterelog Test

Chapter 6

Summary and Conclusions

The goal of this thesis to improve the security, privacy and transparency functionalities of CoMaDa are met. Additional filter extensions allow the WSN owner to search more accurate eg. the search for date and time has been added which allows the WSN to search for data accessed within one minute this clearly leads to an increase in transparency. The Logging functionality for the filtering options as well as the logintrack allows the WSN owner to track the movement on the CoMaDa side leading to more privacy and transparency. The concern about security issues could be reduced with the new login screen which implements an extend version of the upload interface to transmit the credentials to the WebMaDa server side and perfom an login check. This functionality allows only the WSN owner to get access to the more sensitive filtering option and the log options.

The work on this thesis showed me that CoMaDa could be made more secure and the threat for miss use could be reduced for sure. The work with the Upload/Pull interface was characterized by many difficulties. As it included many modules and dependencies which first had to be spotted. In the end it can be said that the extension of the Upload/Pull interface could bring even more transparency to CoMaDa it self. Therefore an extension for gaining data from the WebMaDa database would be an increase in any aspect of transparency and privacy this work treated.

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 January. 29, 2018.

[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, Zürich, 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 January. 15, 2018.

[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, Zürich, 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, Zürich, Switzerland, January 2017.

[10] C. Anliker, *Secure Pull Request Development for TinyIPFIX in Wireless Sensor Networks*; Master's thesis, Department of Informatics, University of Zurich, Zürich,Switzerland, Nov 2015.

[11] M. Balmer, *CoMaDa Extension Addressing Transparency Request for Data Owners*; Assignment, University of Zurich, Communication Systems Group, Department of Informatics, Zürich, Switzerland, July 2017.

[12] N. Silvestri, *WebMaDa Extension Addressing Transparency Request for Data Owners*; Assignment, University of Zurich, Communication Systems Group, Department of Informatics, Zürich, Switzerland, July 2017.

[13] WebMaDa, URL: `https://webmada.csg.uzh.ch/`, last visited January. 29, 2018.

[14] W3SCHOOLs, URL: `https://www.w3schools.com/jquery/jquery_intro.asp`, last visited January 15 , 2018.

[15] datatables, URL: `https://datatables.net/`, last visited January. 15, 2018.

[16] oracle, URL: `http://docs.oracle.com/javase/tutorial/jdbc/basics/prepared.html`, last visited January. 15, 2018.

[17] mySql, URL: `https://dev.mysql.com/doc/`, last visited November. 21, 2017.

[18] postGreSql, URL: `https://www.postgresql.org/docs/9.1/static/index.html`, last visited November. 30, 2017.

List of Figures