Metropolia University of Applied Sciences

Information Technology

Program documentation

Project **“Web Interface for ABB Ventilation Controller”**

in **Internet of Things**

Students:

Philip Mertz
(philime@metropolia.fi)

Philipp Schuler
(philisc@metropolia.fi)

Raphael Müller
(raphaelm@metropolia.fi)

Daniel Richter
(daniri@metropolia.fi)

1 Contents

2 Introduction

This is an application to control a fan system through a web frontend. In this frontend the user can view the current status of the fan, change the fan settings and view an aggregate of the fan data over a selectable time period. Most functions of the webapp are secured behind a user login. A logged in user can also view logs of user activity on the website.

This application uses node.js for the backend and the Express framework to handle network requests. The communication with the fan is done using the mqtt-protocol. The fan-data, user login credentials as well as user statistics are stored in a Mongo database. The communication between the backend and the frontend is mainly done through websockets.

3 Frontend

3.1 Communication with the Backend

The client side is developed by using HTML, CSS and JavaScript. The connection to the backend happens by using the WebSocket Object. The websocket will listen to four different types of messages:

1. ‘initial-data’: When the client initially connects to the server, the fifteen most recent datapoints of the fan-device, stored in the database, will be sent to the client. This data will be processed by the frontend and the current pressure, fan-speed, setpoint and mode of the fan will be displayed to the user. In addition to that, the pressure and fan-speed will be displayed in a graph by using the chart.js library (see below for more explanation).
2. ‘continuous-data’: The backend will send the fan-data that is received from the fan-device to the client over the websocket. Each time the client receives new data under the ‘continuous-data’ message, it will update the fan stats on the website, add the most recent fan data point to the graph and delete the oldest datapoint from the graph. As long as the user is viewing the live status of the fan, there will always be 15 datapoints displayed in the fan.
3. ‘aggregate-data’: If the user requests to view the fan-data over a certain time period, the backend will answer with that data and send it over the ‘aggregate-data’ message. This data will override the current live data in the chart and display the aggregate data instead. The client will still receive updates on the current pressure, fan-speed, setpoint and mode of the fan, but not see the live data graphically. In order to stop the graph, that is displaying aggregate data, from updating with the most recent data, the variable GRAPH_IS_IN_CONTINOUS_MODE will be set to false, if the user requests to view aggregate data. As long as this variable is false, the chart will not update with recent data.
4. ‘most-recent-data’: If the user requests to see the current status of the graph again after having viewed aggregate data, the client will send a request to the server. The backend will send the data with the ‘most-recent-data’ message. This response includes the fifteen most recent data points and displays them in the chart again. The GRAPH_IS_IN_CONTINOUS_MODE variable will also be

set back to true, in order to let the chart be updated every time new data is received from the fan-device.

3.2 Graphical display of the fan activity

In order to display the fan-data graphically, the chart.js library is used. Both pressure and fan-speed values are displayed in the same chart to see the correlation of both values over time. In addition to the pressure and fan-speed values, the chart is also supported with time values on the x-axis. While the chart is displaying continuous data, the x-axis will display hour, minute and second values. If the chart is displaying aggregate-data, the x-axis will display day and month as well as the hour of the datapoint.

The aggregation of the fan-data is done in one-hour intervals. The user can select a time frame by selecting a start day, month and hour as well as an end day, month and hour. This time-data will be sent to the backend, where the corresponding data will be extracted from the database and sent back to the client with the 'aggregate-data' message. In aggregate mode the pressure and fan-speed data will always be rounded to one decimal place. If there is no data available for the selected time-period, the user will be notified through an error message on the page.

3.3 Interaction with the Fan-Device

In order to interact with the fan-device, a mode and setpoint has to be selected. The user can select between auto and manual mode through two buttons. If the auto mode is selected, the user will have only one slider to adjust the pressure value. If the manual mode is selected, the user will only see a slider to adjust the fan-speed value. After selecting a setpoint value with the slider, the selection can be confirmed with the 'Set Pressure' or 'Set Fan-Speed' button, depending on which mode is selected. This will send the selected data to the server through the websocket, where it will be sent to the device to change the fan-settings.

3.4 Pressure Error Handling

If the pressure of the fan-device does not settle on a selected pressure point after some time, the device will send back an error and the user will be notified on the page. If the user enters a different pressure-value and the device starts to adjust to that new selected pressure, the device will keep returning an error until the new pressure setpoint is reached. In order to not keep displaying the error message to the user, a 60 second timer is implemented to give the device enough time to reach the selected pressure. If the device is also unable to settle on the newly selected pressure, the error will be displayed again.

3.5 Style

The styling of the frontend is achieved by standard CSS and the Tailwind library.

4 Backend

4.1 Communication with the fan-system

To communicate with the fan, the node-module ‘mqtt’ is used. The communication between the backend and the fan is handled through the ‘mosquitto’ mqtt broker.

The application backend is subscribed to the ‘controller/status’ topic of the fan-device. When a message from the fan is received, the application adds the current date and time to the fan data, stores it in the database and sends the data to the frontend through a websocket.

When a user changes the fan settings in the frontend, the application sends the data for the requested change back to the fan-device by publishing to the ‘controller/settings’ topic.

4.2 Communication with the client

The communication to the client is handled through websockets by using the ‘ws’ node-module.

If a page is loaded, the client will connect to the websocket. If the client is on a page that requires fan-data, the backend will send the fifteen most recent datapoints of the fan immediately and keeps sending subsequent data points that are being sent by the fan-device.

If the user requests to see aggregate data of the fan-device, the time and date information, that the user has entered will be sent back to the server and the corresponding data will be extracted from the database and sent back to the client.

If the user requests to see the current data again, after viewing aggregated data, a new request is sent to the server. The server will answer with the fifteen most recent datapoints.

The websocket will close if the user leaves the page.

4.3 Authentication

The controls for the fan, aggregate data as well as the user statistics are secured behind a login.

This is achieved through an authentication middleware. A user will be prompted to enter a username and password. The login data will be validated in the backend. The username and password of all registered users is stored in the database. The password is encrypted by using the ‘crypto’ module and the pbkdf2 function. The password that the user entered will be encrypted and compared to the encrypted password in the database. If the password and username match, the user will be directed to the ‘fan-control’ ´page. If the credentials are incorrect the user will be prompted to enter his data again.

On the ‘fan-control’ page, the user can choose to logout which will notify the user of the logout by sending him to a logout page.

4.4 Database

The application uses a mongo-database.

The database has three collections:

1. Fan-Data:

   Stores every datapoint of the fan-device that has ever been sent to the backend. Each entry stores an individual ID, the speed and pressure of the fan, the current mode of the fan, the error value that the device returns, the current time as an ISO string as well as the day, the hour and minute in order to ease aggregation operations.

2. User-Data:

   Stores data for user activity. Every time a user is logging into the website, the username and login time gets stored.

3. User-Login-Data

   Stores the login information of each user, by saving the username and the encrypted password.

4.5 User Activity Logging

The application logs the activity of every user that is logged in to the website. The username, activity and time of the activity is saved to the database. This is achieved by extracting the username from the http header when a http request is sent from the client. In addition to that, a timestamp is created, and the activity is set depending on the URL that was requested. For example, when a user navigates to the ‘/user-stats’ page, a GET request is sent to the ‘/user-stats’ address of the backend, triggering the corresponding function. In addition to handling the request, the function will also log the username, current time and the activity ‘looking at user statistics’ to the database.

Activity is logged when a user visits the ‘/fan-control’ and ‘/user-stats’ page, but also when a user changes pressure or fan-speed and when a user is requesting aggregate data. Whenever a user clicks the ‘Set Pressure’, ‘Set Fan-Speed’ or ‘Show data for Time Period’ button, a GET request will be sent to the backend, depending on which button has been clicked. For example, when the ‘Set Pressure’ button has been clicked, a request to ‘/pressure’ is sent. The function that is handling the ‘/pressure’ request has only the purpose of logging the user activity to the database.

4.6 Routings

This application communicates with the client over the http-protocol by using the Express framework.

Some routings are accessible without a login, and some require authentication.

4.6.1 Routings without authentication:

GET '/' sends the index.html file which displays the current fan data and shows a chart of current fan activity.

GET '/logout' renders the 'logout' ejs-file, which informs the user of the logout and offers a button back to the home page.

4.6.2 Routings with authentication:

GET '/fan-control' sends the 'fan-control' file, which is the main page of the application. It is displayed after the user has successfully logged into the website and gives full control of the fan by letting the user switch between auto and manual mode and set the pressure or fan-speed accordingly. On this page, the user is also able to view an aggregate of the fan data in a chart, by selecting a date and time frame in which the data should be displayed. Furthermore, on this page, the user can request to see logs of the user activity, which will send the user to the /user-stats page.

GET '/user-stats' sends the 'user-stats' ejs-file, which lists all the activity of every user with a timestamp.

GET '/pressure' logs the username and the user activity of 'adjusting pressure' to the database.

GET '/fan-speed' logs the username and the user activity of 'adjusting fan-speed' to the database.

GET '/aggregate-data' logs the username and the user activity of 'looking at aggregate data' to the database.

4.7 Error Handling

When a non-existing URL is navigated, the application will display a 404 Error page by returning an error ejs-file if possible, and a simple "ERROR 404" text if not.