



METROPOLIA UNIVERSITY OF APPLIED SCIENCES

INTERNET OF THINGS

GROUP PROJECT

**Web Interface for ABB Ventilation Controller
Technical Documentation**

Authors:

Janine PASCHEK

Maya HORNSCHUH

Theresa BRANKL

Simon SCHÄDLER

Submitted on: September 23, 2021

Contents

1	Introduction	3
2	Interface pages	4
2.1	Landing page	4
2.2	Control panel	4
2.3	Settings	4
2.3.1	Login history	4
2.3.2	Change password	5
2.3.3	Add user	5
2.4	Logout	5
3	Authentication and Authorization	6
3.1	Front end authentication	6
3.2	User database	6
3.3	Routing	6
3.4	User Authentication	7
3.5	User authorization	7

1 Introduction

Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet. Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet.

2 Interface pages

2.1 Landing page

2.2 Control panel

2.3 Settings

The settings page is available for all logged-in users through the `/settings` route. The settings page displays the same elements for each user, but depending on the users' permission, the content and allowed actions might differ. The settings page displays the login history depending on the user as well as two forms for changing the password and adding a new user to the system.

2.3.1 Login history

The client will automatically fetch a list of all login events the server has logged to the database. If the user has admin privileges, the server will return all login events, otherwise just those of the current user. The data gets returned as a JSON array. After receiving the data, the client checks the number of received events and divides it into a dynamic amount of pages with eight events each. This is done for user experience purposes only. The server then generates a flexbox for each of the eight events and adds it to the login history. It also adds two buttons to switch between the pages of data. If the user clicks one of the navigation buttons, the client will reset the displayed list and add the next pages' data by accessing the received data at another index. The buttons are only active if there are more pages available to load. If a page contains less than eight events, the client will add space holder items instead to keep the design consistent.

When the server receives a request for the login history it first checks the permission of the user. If the user has admin privileges, the server will select all data from the `log_users` table of the main database and return it to the user. The `log_users` table is part of the main database and logs all new logins in the following format:

```
CREATE TABLE log_users(timestamp INT, user TEXT);
```

If the user does not have admin privileges, the server will select all entries, where the username equals the user that send the request and return those to the client.

2.3.2 Change password

The client displays a text input and a button to enable the user to set a new password. The placeholder property will ask the user to enter a password. Both, the input and the button are embedded into a form that has an event handler for "submit" events. Therefore, the user can set a new password by pressing the "OK" button or the "Enter" key. The input field has the required parameter set to prevent the user from sending a post request with empty parameters.

When the server receives the request to change the password from an authenticated user, it does not need to check permissions. All users are authorized to change their passwords. The server just generates a hash based on the username and the new password and updates the users' entry in the database. The server then will return a status message with a status code.

When the client receives the result from the server, it will clear the input and display an alert to inform the user whether the password was successfully changed or not. If the password was changed, the user needs to log in with the new credentials. For user experience purposes, the client will automatically log out the user and redirect to the logout page.

2.3.3 Add user

Adding a new user to the system works similarly to changing the password. The client displays two input boxes and asks the user to enter a username and a password. Both inputs are required and the password input is of the type "password". That will hide the entered password and just display dots instead. After the user submitted the input, the client will send a post request to the server to add the new user to the system.

After receiving the request, the server needs to check the users' permission before adding a new user to the system. Only admin accounts are authorized to add new users. If the user does not have the required permission, the server will return a 403 'Forbidden' error to the client. Otherwise, the server will first check if the requested user already exists in the database. If the user already exists, a 409 'Conflict' error will be returned. If the client requested to add a user that does not already have an entry in the database, the server will first generate a hash based on username and password and then insert a new user to the database. The role of all users added to the system using the web interface is 'default'. The timestamp is initially set to 0 so the users' first login will automatically be handled as a new login (chapter 3.4). After successfully adding a new user, the server will return a 200 'OK' status to the client.

When the client receives the result from the server, it will clear the input fields and display an alert depending on the received status code.

2.4 Logout

3 Authentication and Authorization

Authentication and authorization are both very important for this interface. Only logged-in users are allowed to open the page. Also, not every logged-in user has permission to use all available features. Therefore, on each request a client sends, the server will try to authenticate the user before providing any information. In this project, basic HTTP authentication is used to log in/log out users.

3.1 Front end authentication

When a user initially connects to the web interface, the server will return a 401 error code and ask for authentication. The browser will automatically show a form and ask the user to enter a username and a password. If the entered credentials are correct, the server will redirect to the landing page.

Every page of the interface provides a log-out button, which enables the user to manually log out. If a user logs out, the client will first send a basic get request to log out from the server. Then, the client sends an invalid authentication request and redirects to the logout page. Sending a separate logout request before the invalid authentication enables the server to tell apart the logout request from any other invalid authentication like logging in with incorrect credentials. This is important to keep track of the users that are currently logged in and can be used to log all login activities.

3.2 User database

The server uses one central SQLite database (*data/data.db*) to store all kinds of data. The *users* table contains information about each user in the system in the following format:

```
CREATE TABLE users(username TEXT, hash TEXT, timestamp INT, role TEXT);
```

Next to the username and the hashed password, the table contains a timestamp of the users' last login (in milliseconds since 01.01.1970 00:00:00 UTC) as well as a role. The timestamp can be used to distinguish new logins and navigation between different subpages, but also for logging all login activities on the server (chapter 3.4). The role indicates whether the user has admin privileges or not.

3.3 Routing

All routes that the server handles require the client to authenticate before serving any page or data. The only exception to this is the */logout* route, that returns the *views/logout.ejs* file to the client

without authentication. All other routes call the function `auth_user(req, res, next, redirect)` (chapter 3.4) and pass a string of the requested redirect as a parameter. The function will check the authentication parameters provided by the client. If the provided information is valid, the function will call the requested function and return information to the client.

3.4 User Authentication

If a route is called by a client, it will call the `auth_user(req, res, next, redirect)` function with information about the requested service. The function reads authorization parameters from the request body and checks if valid data was received. If so, the function generates a hash based on the provided username and password and compares it with the hashes stored in the database. If the username and password match the information in the database, the user is authenticated successfully. If a client was successfully authenticated, the function checks if the request was a new login or just an authenticated request for a page or service. To do so, the current time is being compared to the users' timestamp in the database. Three cases can be detected that way:

- If the timestamp equals zero, it either is still zero from its first initialization or has been reset during a logout procedure. Therefore, the client is performing a new login. Then, the login will get logged and the timestamp in the users' database entry will get updated.
- If the difference between the current time and the timestamp of the users' last login is greater than 30 minutes, the request will be interpreted as a new login. Then, the login request will get logged and the timestamp in the users' database entry will get updated.
- If the difference between the current time and the timestamp of the users' last login is less than 30 minutes, the request will be interpreted as a request for changing the page or fetching data. The timestamp will not get updated and the request will not get logged in the database.

After detecting a new login by checking the conditions explained above, the server will add a new row to the `log_users` table (chapter 2.3.1) with a current timestamp and the current username.

After deciding, if the database needs to get updated because of a new login, the function switches depending on the passed `redirect` parameter. If a page is requested, the parameter will directly include the filename of the page that should be rendered. Otherwise, the corresponding function will get called to perform actions and return the requested data to the client.

3.5 User authorization

Users are allowed to use most of the features provided by the web interface. Still, some actions can get performed by authorized users only. For example, only admins are allowed to add a new user to the system or to see all users' login activity. As described in chapter 3.2, the `users` database has an attribute that tells the role of each user. There are two roles, the `default` and the `admin` role. If the client requests a service that is available to admins only, or that returns different results depending on the users' role, the `auth_user()` function will pass the current users' role as an argument to the function, that performs the requested actions. Then, the server decides if the user is authorized or

not. In general, the server returns one of the following status codes on requests that require specific privileges:

200	'OK'	The requested action was successfully executed
403	'Forbidden'	The user has no permission to execute the requested action
409	'Conflict'	The action could not be executed because of conflicting arguments

The server sends those status codes alongside the resulting data or message. Then, the client deals with received data, takes the user to a different page, or displays an alert depending on the result.