Metropolia University of Applied Sciences

Internet of Things (IoT) Project

TX00CI65-3007

IntelliDawn – Documentation

Authors:

Roger Castellvi Rubinat
Laurin Rossmeier
Cisse T'Syen
Simon Schädler

Submitted on: December 17, 2021

Contents

1 Introduction

The project idea is a lamp which aims to wake you up comfortably by slowly increasing its luminosity at a programmed time, apart from the customizable sound alarm. As initial ideas, the lamp should also be able to connect to your sleep tracking device (e.g. smart watch) to monitor your sleeping schedule and to set up itself at the perfect time at the right sleep cycle. It would also provide sleeping recommendations which will be mentioned later among other features.

In order to see the possible usability that would be given to the project we created a Google Form with different questions. Since it's release the 28th of October, the form has received up to 95 answers. With these, a small study can be made, focusing on customer preferences and possible additional features proposed via feedback. The results are as mentioned in the following page.

Figure 1.1: Lamp at 15% brightness.

Figure 1.2: Different camera exposure.

2 Google Forms Summary

Nowadays mobile phones come with many features and one of them is programming an alarm to wake you up whenever you want. So it doesn't come as a surprise that out of the 95 answers received, 99% agree to be using an alarm on a daily basis. But what if we used light instead of an abrupt sound to wake up? Thanks to the results of the form, we can affirm that the production of our project is not aimed for a small specific group from the public but for the wider part of it. This is a big reason to support the idea, as it's main usability would still be functioning as an alarm (although it would not use sound but a slowly brightening light). Furthermore, 83% of the voters would be willing to buy the product should it exist in the real market, which strengthens the idea to continue with this idea for the project.

The other features, though as not as important as the main one, should be still present since only 58% of the answers to the form agree to have a healthy sleeping schedule. This is where this such features can come in hand, for example, by providing various sleeping recommendations to increase that percentage. Some default recommendations had already been planned and introduced to the form as an example. The most useful (according to most of the votes) is displaying the best time to go to sleep, followed by food and drink recommendations, ideal light exposure and ideal room temperature.

Feedback has taken an important role in the Google Form, as some extra recommendations to be displayed and additional features have been proposed by the voters. A very common proposal repeated by many voters is the possibility to add any song/music you like as a wake up sound in addition to the default slow brightening light. It is true that all of this is subjective when it comes to usability, as everyone wants what they think is useful to them, but some answers such as this could make a great addition should the project be taken seriously.

- Click here to go to the Git Hub's repository
- Click here to go to the Google Form's results
- Click here to go to the LaTeX source file

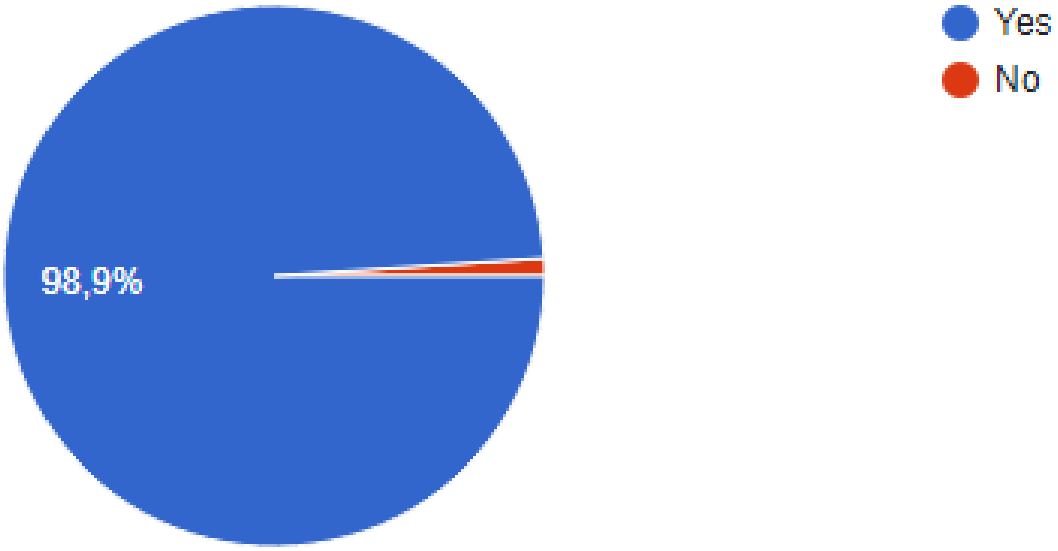


Figure 2.1: Alarm usage percentage

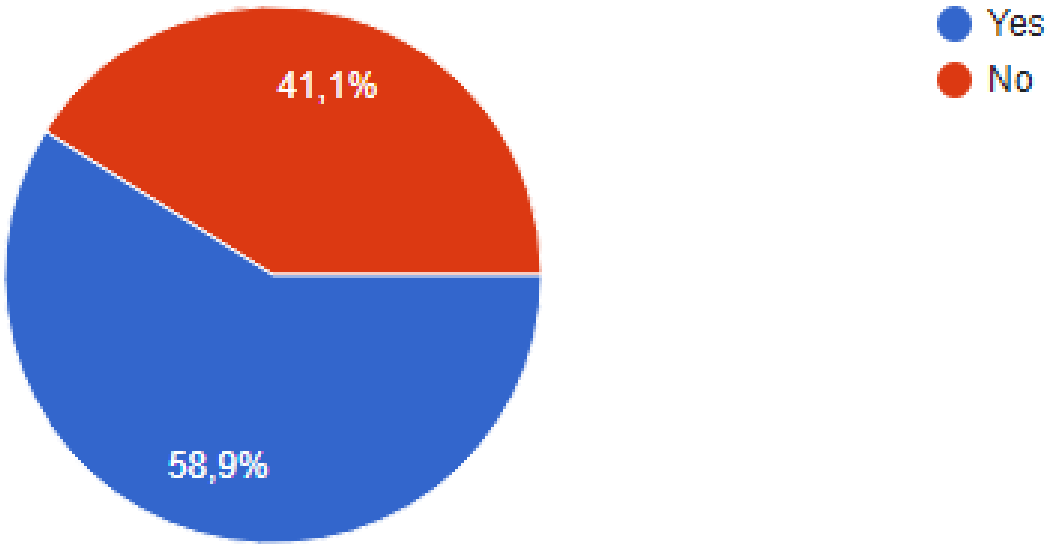


Figure 2.2: Sleeping schedule results

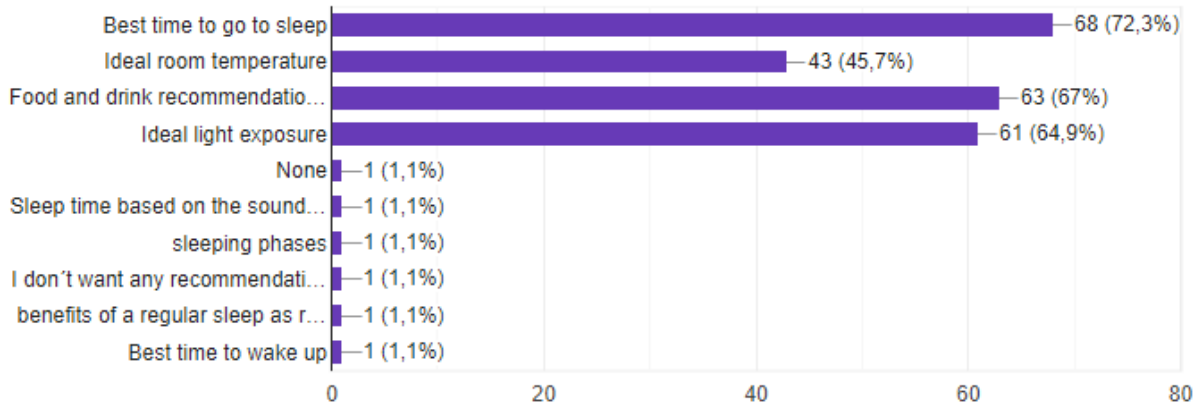


Figure 2.3: Recommendations selected

Soft and progressive sound. Same with the light, not too bright. Something like a sunrise

slowly increasing the light intensity. if it rapidly turns on i feel blind

Wake up call mode choice, maybe even an oldschool Radio which fucks me up every morning

Colours, relaxing music

It can wake up also with the music and not only with the lights!

The best time for you to go sleep, good food and also the analysis of your sleeping

Choose the sounds of the alarme or the intensity and color of the lamp

Unable to postpone it from the bed, being forced to stand up to switch it off

Idk if that's the idea, but slowly increasing the light intensity to wake the person up instead of using sound

Figure 2.4: Some of the features proposed via feedback

3 PCB

A custom PCB was designed through EasyEDA for this project, which is a simple online pcb design tool. This PCB was made by JLCPCB, a manufacturer which is a partner of EasyEDA. Firstly, a drawing of the wiring diagram is done, from which a PCB design can be later generated.

3.1 Wire diagram

Following this explanation is a wiring diagram of the lamp. In the middle the esp32 can be seen with all the connections to the components. There is 2 different voltages in the whole circuit: 12V and 5V. The 12V is necessary for the LED strip and the piezzo buzzer, which is controlled with a mosfet (13N10L). The 12V can be converted to 5V through a linear voltage regulator (LM317) to power the rest of the components. At the top right it can be seen the connection pins for the temperature and humidity sensor (DHT22). Finally, there is also a number of pin headers for the touch pins and additional pins for possible expansion

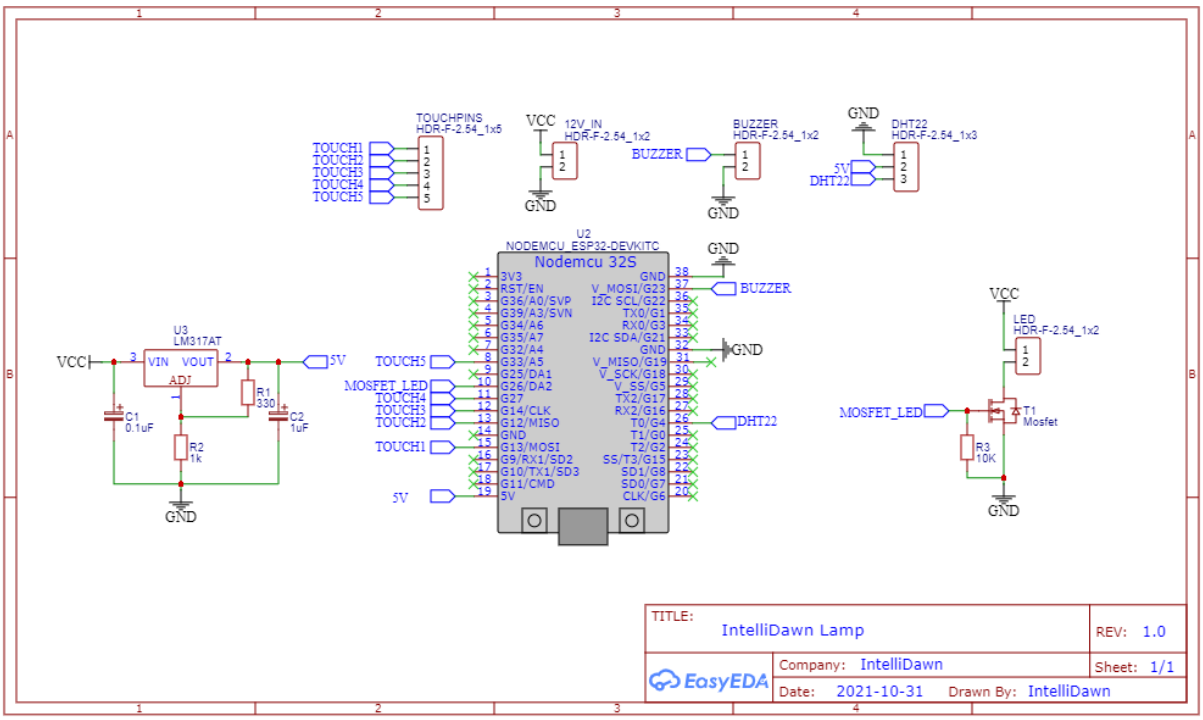


Figure 3.1: PCB schematic lamp

3.2 PCB diagram

Below this explanation is the PCB schematic of the lamp. It was made as compact as possible so that it clicks onto the esp32 and fits snugly into the lamp. Then, a GROUND plane was put all over the PCB, which ensures that there is less interference that can affect the ESP32's signals.

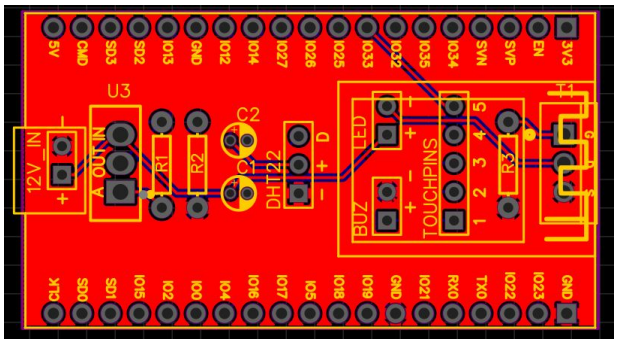


Figure 3.2: PCB diagram

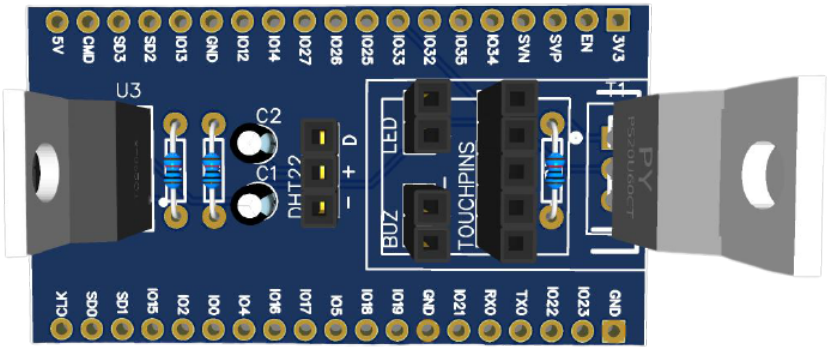


Figure 3.3: PCB 3D design front

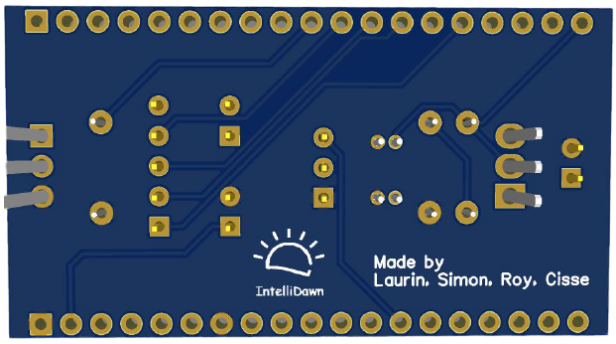


Figure 3.4: PCB 3D design back

4 Server and user interface

Most of the functionality of the system as well as the user interface is being provided by a Node.js server. The server triggers alarms on the device, stores information about users and each user's alarms in a database and allows the user to do all kind of settings through a web interface. This chapter includes detailed information about the system design, the implementation, and the user interface design.

4.1 Server installation

This project is based on Node.js which needs to be installed to run the server. Please visit nodejs.org and follow the instructions to install node (LTS or latest version). The following node packages are required to run the server. The packages can be installed by running *"npm install <package>"*.

- body-parser
- cookie-parser
- crypto
- ejs
- express
- express-session
- mqtt
- multer
- sqlite3
- ws

The server also requires a running MQTT broker. To connect the server to a running broker, please open the */src/server/server.js* file and check the MQTT configuration section. Please enter the correct IP, port, and topics by adjusting the following parameters. The configuration section also allows you to change further values in case a different port or database should be used:

```
// SERVER
const SERVER_PORT = 30001;
const HTTPS\_OPTIONS = {
   key: fs.readFileSync("/etc/letsencrypt/live/domain.com/privkey.pem"),
   cert: fs.readFileSync("/etc/letsencrypt/live/domain.com/fullchain.pem")
}

// MQTT
const MQTT_IP = 'mqtt://212.83.56.219';
const MQTT_PORT = 1883;

const MQTT_USER = 'IntelliDawn';
const MQTT_PASSWORD = 'vNuEAzy3dqCTCacWcBOpnsiSjkshzaO9ykgdy9tF';

const MQTT_TOPIC_STATUS = 'IntelliDawn/status';
```

```
const MQTT_TOPIC_BRIGHTNESS = 'IntelliDawn/brightness';
const MQTT_TOPIC_SOUND = 'IntelliDawn/sound';

// DATABASE
const PATH_DB = __dirname + '/data/data.db';
const DB_ALARMS = 'alarms';
const DB_USERS = 'users';

// AUTHENTICATION
const MAX_SESSION_AGE = 1000 * 60 * 60 * 1;
```

After installing the required packages and setting up the MQTT parameters, the server can be started by running *"node src/server.js"*. The server displays status information in the terminal if everything started up correctly.

Please note that this server is optimized to run on a Ubuntu 20.04. web server requiring a valid SSL/TLS certificate. Running the server without a valid certificate or in a different environment may cause errors.

4.2 SQLite database

The server uses an SQLite database which is set up as a single global database to store all kinds of data. The database (*/src/server/data/data.db*) contains the following tables to store data:

```
CREATE TABLE alarms(id TEXT, user TEXT, days_selected INT, icon INT, time_alarm
    INT, time_light INT, smooth_shape TEXT, sound INT, snooze INT, device TEXT);
CREATE TABLE devices(id TEXT, user TEXT, name TEXT, status INT);
CREATE TABLE log_temp(device_id TEXT, temperature INT, timestamp INT);
CREATE TABLE users(email TEXT, hash TEXT, name TEXT, last_name TEXT, sign_up
   INT, next_alarm INT);
```

The *alarms* table contains a list of all alarms which are currently set. This data set is being used to trigger alarms on the device but also to display a list of currently set alarms to the users allowing them to add, edit, or delete alarms at any time through the web interface. The *users* table contains information about all users that have signed up to the system. This data set is being used to sign users in and out as well as for some further user experience oriented features.

4.3 Session based authentication

Authentication is very important to keep the systems integrity. Unauthenticated users must not be able to set, edit, or delete any alarms. Also, no personal information of signed up users may be accessible to others. Therefore, on each request a client sends, the server will try to authenticate the user before providing any information or performing any action. In this project, session based

authentication is used to authenticate users as well as to log them in and out. To protect HTTP requests from being read or manipulated by third parties, the server encrypts all the communication using the HTTPS protocol.

4.3.1 Routing requests

The server uses Express to manage the routing. To prevent unauthenticated users from accessing data, every route refers to the *auth_user(req, res, next, redirect, arg_dyn = ' ')* function to authenticate users. The following example shows the routing for */get_alarm*:

```
app.get('/get_alarm', async (req, res, next) => {
   auth_user(req, res, next, 'get_alarm', req.query.id);
});
```

In this example, the server will pass all necessary parameters for authenticating the user and returning a response, the requested alarm ID and information about which route route was called. If the user is logged in, the server will call a function to read and return information about the requested alarm. If the user is not logged in, the *auth_user()* function will deny this request (chapter 4.3.5).

4.3.2 Front end authentication

When a user initially connects to the web interface, the server will automatically redirect to the login page. There, the user can enter an e-mail and a password which will then be sent to the server. The login page also allows new users to sign up to create a new account.

Every page of the interface provides a sign-out button, which enables the user to manually log out. If a user wants to log out, the client will send a basic get request to log out from the server. The server will log the user out and redirect to the login page.

4.3.3 User database

The *users* table of the central database is used to store the login information of all users (chapter 4.2). Next to the e-mail and the hashed password, the table also contains information about the users name, next alarm and sign-up date. This information is only being used to enhance the user experience on the web interface.

New entries to user database can be added by signing up with a new e-mail. Aside from manually adding or editing an entry by directly connecting to the database and adding data with SQL commands, the web interface also provides multiple settings options to edit or even delete the users' account (chapter 4.7.1).

4.3.4 Middleware

All routes the server handles require the client to authenticate before serving any page or data. The only exception to this is the */logout* route, which deletes the current session and returns the *views/ login.ejs* file to the client without authentication. All other routes call the function *auth_user(req, res, next, redirect)* (chapter 4.3.5) 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.

4.3.5 Back end user authentication

Session-based authentication uses session cookies to authenticate users. To log in, the client sends an e-mail and a password to the server using a POST request. Then, the *login_user* method generates a hash based on the provided e-mail and password and compares it with the hashes stored in the database. If the e-mail and password match the information in the database, the user is authenticated successfully. The server will then create a new session with a unique session ID. This ID will be returned to the client as a session cookie and will be appended to all following requests to the server.

Every time another request is received from any client, the server needs to check if the client is authenticated before executing any method or returning any data. Therefore, all routes call the method *auth_user(req, res, next, redirect, arg_dyn = '')* and pass all necessary connection parameters as well as information about the requested redirect. This function checks, if the user is authenticated and logged in by comparing the passed session ID with the currently active sessions. If no session ID or an invalid one has been received, the server will redirect to the login page and ask the user to sign in or to create a new account. After authenticating the user, the function returns the requested page or calls a method to perform specific actions depending on the *redirect* parameter. This parameter can either contain a direct path of a file that should be rendered and returned, but can also indicate a number of functions that need to be called.

If the user wants to sign out, the client sends a get request to the */req_logout* route. After receiving this request, the server will delete the current session and redirect the user to the login page. Deleting the session will automatically invalidate all incoming requests with the former session ID. The user is then logged out.

4.4 MQTT communication

The server and the lamp communicate with each other using MQTT. Messages are being sent on three topics over a online MQTT broker. To prevent third party access to the broker, it requires all clients to authenticate before a connection is being established. Then, the MQTT connection is responsible for exchanging all required data between the server and the device.

Topic	Data[0]	Data[1]	Data[2]	Data[3]	Data[4]
IntelliDawn/brightness	***device_id [TEXT]***	***brightness [INT]***			
values	XXXX-XXXX	0-255			
IntelliDawn/sound	***device_id [TEXT]***	***sound [INT]***			
values	XXXX-XXXX	0/1/2/3/4			
meaning		(OFF/Beep/Tetris/ Arkanoid/Mario) sync/status)			
IntelliDawn/status	***device_id [TEXT]***	***status [INT]***	***event [INT]***	***value [INT]***	***type [INT]***
values	XXXX-XXXX	0/1/2/3/4/5/6	0/1/2/3/4/5/6	0-255	0/1
meaning		(ok/error/warning)	(brightness/sound/ temp/snooze/stop/ sync/status)		(req/resp)

Figure 4.1: Message specification for the three MQTT topics

Example: To set a brightness on a device, the server sends a message over the *IntelliDawn/brightness* topic addressing a specific device and providing information about the requested brightness. After receiving a command, the device will respond by sending a status OK on the *IntelliDawn/status* topic.

While the brightness and sound topic are used in one direction only, both the server and client are sending messages on the status topic. Apart from acknowledging received messages, it is also being used for fetching a device status, synchronizing, temperature logging, and sending snooze and stop inputs to the server.

4.5 Setting and running alarms

Users can set multiple alarms using the "Alarms" page of the web interface. The client will fetch a list of all alarms that are currently and display them in a list of clickable cards. Each cards displays an icon, the time of the alarm, and on which days the alarm is active. The user can click on any card to edit or delete the alarm.

When setting or editing an alarm, the user can do multiple settings before activating an alarm which will be described in the following sections. After setting an alarm, all settings will be stored in the database. When a user wants to edit an alarm, the client will fetch all the required information from the database to display the current settings correctly.

4.5.1 Setting a repetition of the alarm

By default, an alarm will run only at the time set by the user. But an alarm can also be set up to repeat on selected days of the week. To do so, the user just needs to select all the days by clicking on the buttons displaying the week days. Selected days will be displayed in color, others in white. We used bit masking to keep track of the selected days for storing them to the database and visualizing selected days on the interface. The following table shows how an eight bit integer can be used to

store all possible combinations of selected days. The example represents the case that the user has set up an alarm to run on weekends.

	Bit 0	Bit 1	Bit 2	Bit 3	Bit 4	Bit 5	Bit 6	Bit 7
Meaning	Run once	Mo	Tu	We	Th	Fr	Sa	Su
Example	0	0	0	0	0	0	1	1

Each button has an id equivalent to the bit it's related to. Clicking a button will trigger an event listener which updates the value indicating the currently selected days. Biwise operators allow us to update the current state without comparing each buttons state and running threw a lot of if statements.

```
if(0x1 & day) {
    days_selected = 0x1;
} else {
    days_selected = days_selected | 0x1;
    days_selected = days_selected ^ (day | 0x1);
    if(!days_selected) {
        days_selected = 0x1;
    }
}
```

The *day* parameter indicates which button has been pressed (0x1 shifted depending on the selected day). If the "1x" button has been pressed, the selected days will be reset. Otherwise, the first bit is being set to make sure to disable the "1x" button after applying the changes. The changes will be applied using a XOR operator on the days_selected and the *day* parameter which will automatically set the first bit to zero. However, if pressing the button would deselect all days, the *days_selected* will be reset to 0x1.

After updating the *days_selected* parameter, the client will update the class of all buttons to display the selected days correctly. This parameter will later be used to update the next alarm time when running an alarm (chapter 4.5.4).

4.5.2 Setting up the light to increase brightness

The actual alarm will run at the time set by the user. To make waking up more comfortable, the system allows the lamp to increase its brightness before the actual alarm. The user can set up how long before the actual alarm the light should turn on. The systems also provides three different modes. In "smooth" mode, the light will smoothly increase over the selected period of time. In the "steps" mode, the brightness will increase in five steps and in "rough" mode, the light will immediately turn on at maximum brightness.

The open source graphing library Plotly is being used to visualize the increasing brightness over the selected period of time. When changing the mode or the selected time interval, the plot will be updated in real time.

4.5.3 Other settings and saving the alarm

Next to the described settings, the user can also select a specific time when the alarm should turn on, a sound pattern, and a snooze interval. Users can also select one of the devices connected to their account (chapter 4.7.2). This is important because alarms are being sent to only one device.

After confirming changes or adding a new alarm, the client will send all information to the server as a JSON object using a POST request. The server will then update the database entries according to the changes made.

4.5.4 Running an alarm

The *alarms* table of the database contains two timestamps telling the server when to turn on the light an the sound (chapter 4.2). Every twenty seconds, the server checks if there are any entries with a timestamp in the past which indicates that an alarm should be running. For each alarm found, the server will update the database entry and start sending MQTT messages to the associated device.

First, the server will update the timestamps in the database by calculating the next time of the alarm depending of the selected repetition. If an alarm should run just once, the alarm will be deleted from the database. Then, the identifies how often the brightness needs to be set depending on the selected mode (chapter 4.5.2) and time interval to calculate the time between the commands. The server will then send MQTT messages to set the brightness on the device and after the calculated period of time, a command for the sound will be sent as well.

To ensure that these commands are actually being received by the device, the server keeps sending the message up to ten times until the device returns a message on the *IntelliDawn/status* topic to acknowledge the command. As multiple alarms can be running at the same time for different devices, semaphores have been implemented to wait for the acknowledgement. Before a command for the first time, the semaphore is being locked. The server then sends the command and tries to acquire the semaphore for the current alarm. If the semaphore can not be acquired within 500 milliseconds, it will time out and the message will be sent again. As soon as an acknowledgement is being received by the MQTT subscriber, it will release the semaphore for the associated device. This will stop the server from sending the same command again. This method is being used for sending brightness and sound commands only, because those are the only critical messages.

After the command for the sound has been sent and acknowledged, the semaphore and alarm are not being deleted yet, because the user might snooze the alarm which requires the server to know which snooze time has been set. After receiving a snooze request, the server will set up a simple timeout and repeat to send the sound command afterward. As soon as a stop request has been received, the server will delete the associated semaphore and all information loaded about this alarm. Both the snooze and stop request are being acknowledged by the server to tell the device to stop sending further messages.

4.6 Information about the users sleep

The "My sleep" page of the web interface provides helpful information about how a perfect sleep can be achieved as well as a visualization of the temperatures measured by the users devices.

Clicking on "Perfect sleep" will load a sub-page displaying a list of tips on how to improve the users sleeping quality. Based on studies made by multiple official institutions, we are able to derive some tips and tricks which are being displayed in this section of the interface.

The "Temperatures while sleeping" sub-page on the other hand allows the user to select a specific period of time to display a plot of the temperatures measured for each connected device. If no devices are connected or one of the devices did not measure any temperatures in the selected period of time, the client will tell so.

In this project, the open source library Chart.js is being used to visualize the temperature data in multiple plots. Each temperature sample the server receives from any device will be stored to the *log_temp* table of the database (chapter 4.2). After the user has selected a period of time, the client will fetch an array of all database entries where the timestamp is in between the selected borders. To reduce traffic and to prevent performance issues on the interface, the server will minimize the amount of data to a maximum of 2.000 samples per device.

After receiving the data, the client will assign it to each device. The timestamps are being formatted to display hours and minutes before being displayed on the x-axis of a plot. As temperatures are not changing very frequently at normal circumstances, the plots are not being updated in real time.

4.7 Account and device settings

The settings page of the web interface is allows the user to change all settings related to the account but also to connect new devices or to check the status of those that already have been connected the account.

4.7.1 Account settings

In the top section of account settings sub-page, the client displays general information about the account like the current profile picture, the users name and email, as well as when the account has been created. This information is being fetched from the server an will be updated immediately after changes have been applied. Furthermore, the account settings allow the user to change the display name and the account password, or to delete the account and all associated data.

Changing the profile picture

Multer is a Node.js middleware for handling *multipart/form-data*, which in this case is being used for uploading files. Clicking on the profile picture on the settings page will load a HTML file selector allowing the user to select a single file. The allowed file formats are *.png*, *.jpg* and *.jpeg*. To upload the file, the client sends a POST request to the server with the file added to the messages' body. The server handles the request using multer and updates the users' profile picture in the */src/server/public/img/profile_pictures* folder. The picture will be named based on the users' e-mail: "*example_user@example_mail.com.png*". Even if multiple file formats are allowed to be uploaded, the server converts all images to *.png* files to simplify access to the profile pictures without storing multiple pictures per user.

If the picture has been uploaded and stored successfully, the server responds with a status 200 (OK). Even if the client allows image files only, the server also checks if the uploaded file format is correct. If not, the server will respond with a status 415 ("Only .jpg, .jpeg and .png files are allowed!").

After receiving a status 200 (OK), the client will refresh the *src* parameter of the profile pictures to display the new profile picture immediately without reloading the page.

Changing the display name

The account settings also provide a form to update the users' first and last name. These names only affect the appearance of the web interface and have no impact whatsoever. Therefore, after entering a new name, no further confirmation is required. After submitting the form, the client will send a POST request to the server. If the user is authenticated, the server will update the first and last name in the database. After receiving a status 200 (OK), the client updates the displayed names on the dashboard and the account settings page.

Changing the password

Additionally, the user can change the account password by entering a new password into the form. As the password is essential to authenticate users, the user has to confirm the new password by entering the same password again before submitting the form.

Each users' password is saved as a hashed value in the database. If a user sends a request to change the password, the server first needs to generate a new hash using a PBKDF2 function. The server uses the accounts e-mail as the salt parameter and generates a 64-bit hash by iterating 10.000 times:

```
crypto.pbkdf2(password, username, 100000, 64, 'sha512', (err, key) => {
    if (err) throw err;

    let hash = key.toString('hex');
    change_password(req, res, next, username, hash);
});
```

After a new hash has been generated, the server connects to the database and updates the current users' entry in the *users* table by replacing the old hash with the new one. It is important, to catch errors while generating the hash and updating the database. The user might be unable to log back into the interface if the system suggests that a new password has been set successfully but has not. To prevent this, the server will respond with an internal error code to the client if any error occurs. This will trigger an alert telling the user that something went wrong. If the password was changed successfully, the server responds with a status 200 (OK). The client will change the appearance of the text inputs and button to indicate that the requested action has been performed successfully.

Deleting the account

The account settings don't just provide forms to update the account settings but also allow the user to delete the account with all of its associated data. Clicking on the "Delete account" button will display an alert asking for confirmation. The user can either cancel without any effect, or delete the account.

If a user wants to delete an account, no user-related data must persist. This means that the users' entry from the *users* table, all of the users' alarms in the *alarms* table, and all connected devices in the *devices* table of the database, as well as the users' profile picture, need to be deleted. After receiving an authenticated DELETE request on the */delete_account* route, the server will perform all these actions and return a status 200 (OK) if the account has been successfully deleted.

After the account has been deleted, the client will tell so to the user by displaying an alert. After closing such an alert, the user will automatically get redirected to the login page.

4.7.2 Device settings

When entering the device settings, the client will fetch a list of all devices connected to the account to display those devices in a list of clickable cards. Each card displays an icon, the device name, and the current connection status of the device. After fetching the devices, the client requests the device status of each device loaded. The server will send status requests via MQTT and return the connection status to the client depending on if a response has been received in time or not. Depending on this response, the client will set the displayed text to "connected" or "not connected". If no device has been added yet, the server will display a card asking the user to add a new device.

Adding a new device

When the user wants to add a new device to the account, a new sub-page will be rendered to guide the user through the setup progress. The client will tell the user how to set up a new connection to a device in three steps. After the user has plugged in the power connection of the device and connected to the access point, the device asks the user to enter a device ID. To ensure, that each device has

a unique ID, the client fetches an ID from the server before displaying it on the web interface. The server generates a nine-digit ID based on a current timestamp and ensures, that the generated ID does not already exist before returning it to the client. The user can simply copy the provided ID to the clipboard by clicking on it. This simplifies the setup progress if the user is connected to the web interface and the access point with the same device. As soon as the client displays the ID received by the server, it frequently fetches information about this device from the server by sending GET requests. As long as the device has not been connected to the server, the server will return an error 404.

When the server receives a sync message from a device, it validates the request and adds a new entry to the *devices* table of the database (chapter 4.2). As soon as an entry with a matching device ID has been added to the database, the server will return this entry on the clients' requests. After receiving a valid response indicating that the device has successfully been connected, the client automatically moves on to the next and last step of the setup progress. In the last step, the user can assign a custom name to the device which makes it easy to tell multiple devices apart. After finishing the setup progress, the client will navigate back to the main device settings page and add the new device to the list of all connected devices.

Changing device settings

If a user clicks on one of the connected devices in the list, the client will load a new sub-page to allow the user to do some settings. The edit page displays the name and ID of the device as well as the connection status. The user can change the display name of each device at any time. Furthermore, the user can delete a connected device from the account. If a device has been removed, the whole setup progress has to be repeated before it can be used again. Therefore, the client requires the user to confirm deleting the device using an alert.

If the server receives a valid DELETE request for a device, the associated entry will be removed from the *devices* table of the database. However, alarms that have been set on the device will remain. This allows the user to assign a new device to those alarms without setting a new alarm entirely.

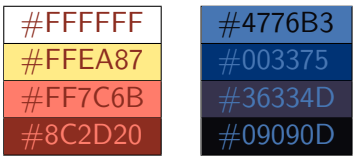
4.8 Web interface design

The web page is the only interface between the user and the IntelliDawn system. Therefore, the interface needs to allow the user to use all functionalities provided in an intuitive way. Most of the functionality is implemented in the back end, so this will not be explained any further. But next to the provided functionality, there is an additional main requirement for the web interface: The interface needs to look nice and modern, use a design that matches the system, and display all content in a consistent and intuitive way.

4.8.1 Color schemes

One of the most important factors for designing a web page consistently is to use a fixed color scheme on all pages. That means that a set of matching colors needs to be chosen and then those colors are being used for every colored element used on the web page.

The IntelliDawn lamps are made to wake up users in the morning by turning on the light. This change from the dark to the light, day to night inspired the following design: The web interface uses two different color schemes, one for imitating the night, one for imitating the day. The login/logout page of the interface should use the night design, but when logged in, the interface should use a warm, enlightening design. To do so, the following two custom color schemes were created by trying and comparing different color combinations on the web page and are now being used on every interface page:



The chosen colors not only match the style and idea of the project in general but also work very well in combination with the general design of the web interface. To improve the user experience when logging in or signing up, not only the color scheme is being changed but it is being changed in a smooth animation as well. More information about animations can be found in chapter 4.8.3.

4.8.2 Glass design of pages and sub-pages

To match the very modern approach of an alarm and a light, the web interface also needs to have a nice and modern appearance. Therefore, we designed a glass-like page style. The background of the web page should be a smooth transition between two colors depending on the current color scheme. Above that, a smaller frame is being placed looking like a piece of glass. This frame contains all elements of the web page. The sites' content is being placed in an additional layer above to be able to move the content in and out later. Then, the last layer is the navigation menu which is being placed above the content to allow content slides to slide in and out between the glass frame of the navigation menu and the main glass frame in the back.

If a user wants to open a sub-page, for example when adding an alarm, a new content box is being generated with all necessary elements. The box's dimensions match the original content box, but the position is set to absolute to be able to move it independently (chapter 4.8.3). The new content box shares all design settings with the main content box and is being displayed instead when requested.

4.8.3 Animations

When navigating between different pages and sub-pages of the web interface, adding animations improves the user experience a lot. There is a simple way of animating these transitions between different pages. First, the content box will slide out to the left disappearing when leaving the main frame. This always happens before redirecting to the new page. When loading the new page, the content will be replaced and moved to the right to slide back in. The navigation menu also indicates the current page by applying a filter to the current pages text. This is also being animated smoothly to match the smooth animation of the content.

When loading a sub-page like adding a new alarm, the new content box is being rendered outside the main frame. As soon as the new content is rendered, the new content box is moved to the left to slide in and overlap the main content box. While the new content slides in, the opacity of the main content is being reduced to 0 smoothly. The main content is not being cleared, which allows the sub-page to add or manipulate elements of the main content box while being hidden. When going back to the parent page, the sub-page is being moved back to the right disappearing when leaving the main frame. After sliding out, the new content box is being cleared and hidden to prevent it from showing up when navigating to a different interface page. These operations are being used for all transitions between main pages and sub-pages. The only difference is the animations when logging in or out, and when signing up.

When a user signs in, the content of the navigation menu and the content box need to be updated. Also, the color scheme needs to change and the moon which is being displayed in the background needs to disappear while the sun-like shape in the bottom left corner should slide in. To move the sun and moon, the client just changes the position of both objects in a smooth transition to slide in or out. To replace the moon with a brighter round shape, this new shape is being hidden by default and just increases its opacity while the moon is disappearing. To change the color scheme, the background color is being changed smoothly. However, the background is not a static color but a transition between two colors. As transitions are not supported for linear-gradients, the client changes the colors multiple times to simulate a smooth transition. Then, to change the content, the content box with all elements for the sign up slide out to the left underneath the navigation menu. Then, the navigation menu slides out as well as its content needs to be replaced with the users profile picture, information about the next alarm, and the real navigation menu. After disappearing, the content of the navigation menu box is being updated and it slides back in afterward. After completing these animations, the client will redirect to the landing page (alarms page) using the default animations. When signing out, the same animations are being used to go back to the night-light appearance but running in the opposite direction.

Additionally, a lot of small animations are being used for all kinds of purposes, but these are not being explained any further. To find out how additional animations work or to get more detailed information about the animations described before, please check the *.js* files triggering the animations and the *style.css* file which includes all style settings, formatting, and animations.

5 Google Sleep API

Google Sleep API is an attempt to implement a sleeping API to the project in order to provide more functionalities. It is a simple mobile app that consists of a button which when pressed, requests the user permission to subscribe to the sleeping data API. As long as the user is subscribed, the app will receive "sleep events" and store them in a database.
It is based on the Android Sleep API Codelab and uses different components which are key to its functioning.
The app works fine, as sleeping events are received and printed successfully on the user interface but it was not possible to access the Room database in which they are stored and retrieve them to be sent to the server. However, we considered it was necessary to mention the attempt, as if achieved, it would have been a great addition to the project.

5.1 App permissions

AndroidManifest.xml is a file which describes the essential information about the app, such as permissions required, the Android build tools, the Android operating system and more. Therefore, as it is required to subscribe to the Sleep API, a dependency must be declared in the manifest. It is also needed to add the runtime permission for API version 29.
Once declared, the main activity will check the activity recognition permissions. If not granted, a snackbar will pop up in order to explain the needs for permission and allow users to go to system settings in order to approve it.
If the Activity Recognition permission is approved, and if the user subscribed to sleep data, the app subscribes to sleep updates. Otherwise, it unsubscribes.

5.2 Processing the sleep events

Whenever a sleep event occurs, the app will receive an *Intent* callback. From such, a list of SleepSegmentEvents (which represent the result of segmenting sleep after the user is awake) or a list of SleepClassifyEvents (which represent a sleep classification event including the classification timestamp, the sleep confidence, device motion and ambient light level) can be extracted. The latter are reported at regular intervals, such as every 10 minutes.
Once received, they will be classified and saved in a Room database. This is later accessed through a ViewModel using LiveData.

6 Conclusion

Once finished, the project has lead to different conclusions. The first of all is that should the topic for the project not be something one enjoys or finds interesting, it can get tiring and difficult, as not the same amount of interest will be put on it. Since the project was discussed in the beginning and all members agreed on the same idea, it has been less tiring, as all members were excited with every small achievement.
In the initial phase of the project, many and many ideas for it were discussed and all of them fitted perfectly to the project. However, it was necessary to reduce the plans to small functionalities and take it slow. Therefore the second conclusion is that the best way to develop the project was to segment its functionalities and go one by one. This way, advancing to the next step was a safe process as all functions that were in the project had been tested and functioned with no errors, establishing a safe base for every new addition.
In relation to the previous conclusion, the project did not consist of one specific field of studies but of many. Every addition to the project has allowed the members to compenetrate and distribute the work according to each one's knowledge.

So all in all, the group has managed to 3D print a model which, when joined with a series of LDS, a custom PCB (and all the required extra components) and connected to a working server (with a customized user interface that allows the user to edit many settings of the lamp, besides other user settings), perfectly functions as a lamp/alarm. The final result is a project which can be easily done at home provided with access to a 3D printer, the project's documentation and the GitHub containing all base code required for it to work. It is also cheap and efficient, resulting in a great addition to anyone's bedroom and sleep life, providing a smooth experience to wake up comfortably and non abruptely.