



DOKUMENTATION ZUR ABSCHLUSSARBEIT

Gesamtprojekt

IPSUM IQ

Entwicklung eines smarten Lichtschalters

Abschlussarbeit – Nummer
HTL-2023/06

Projektanten

Jonas Feyersinger	4YFIT-1
Bernhard Rieder	4YFIT-11
Goran Kosic	4YFIT-7
Kilian Grassauer	4YFIT-4

Betreuer:

Dietmar Duft
Herbert Kapeller
Bernhard Schwaiger

Eidesstattliche Erklärung

Ich erkläre an Eides statt, dass ich die vorliegende Abschlussarbeit selbstständig und ohne fremde Hilfe verfasst, andere als die angegebenen Quellen und Hilfsmittel nicht benutzt und die den benutzten Quellen wörtlich und inhaltlich entnommenen Stellen als solche erkenntlich gemacht habe.

Projektmitarbeiter:

Goran Kosić

Jonas Feyersinger

Bernhard Rieder

Kilian Grassauer

Saalfelden, am

Abschlussarbeit

4YFIT - Abschlussprüfung 2022/23

Thema:

Entwicklung eines smarten Lichtschalters, welcher über eine Webanwendung gesteuert werden kann.

Aufgabenstellung:

Ein herkömmlicher Lichtschalter sollte durch einen smarten ersetzt werden. Dieser sollte mittels Weboberfläche bedienbar sein und sich nach einer einstellbaren Zeit wieder ausschalten. Die Oberfläche sollte in responsivem Design ausgeführt sein. Ungebetener Zugriff wird mittels Authentifizierung verhindert. Jede Änderung des Schaltzustandes wird automatisch dokumentiert.

Kandidaten/Kandidatinnen		Betreuer
Jonas Feyersinger	4YFIT-1	Dietmar Duft
Bernhard Rieder	4YFIT-11	Herbert Kapeller
Goran Koscic	4YFIT-7	Bernhard Schwaiger
Kilian Grassauer	4YFIT-4	

Danksagung:

Einen besonderen Dank wollen wir an Herrn Andreas Rainer für die Fertigung des Holzgestells des Versuchsaufbaus und die Bereitstellung vieler Bauteile aussprechen. Einen weiteren Dank wollen wir an Herrn BEd Herbert Kapeller aussprechen für die Bereitstellung einiger Bauteile und die Hilfe beim Entwurf der Schaltung.

Kurzfassung (Goran K.)

Beim Projekt „IPSUM IQ – Entwicklung eines smarten Lichtschalters“ handelt es sich um ein Smart Home System. Ein Licht wird über eine Webseite angesteuert und jedes An- und Ausschalten mit der Webseite wird durch Datenbankeinträge, welche auf der Webseite ersichtlich sind, dokumentiert.

Abstract (Goran K.)

The project „IPSUM IQ – Development of a smart light-switch“ is a smart home system. The light can be toggled via a website and every light toggle done with the website is documented with database entries, which can be looked up on the website.

Inhaltsverzeichnis

1	Einführung (Goran K.)	10
1.1	Projektfindung	10
1.2	Ausgangslage	10
1.3	Projektziele	10
1.3.1	Muss-Kriterien	10
1.3.2	Soll-Kriterien	10
2	Projektorganisation	11
2.1	Projektrollen	11
2.2	Projektzeitplan.....	11
2.3	Zeitaufstellung Projektteam	12
3	Allgemeines	14
3.1	Github / Git (Bernhard R).....	14
4	Besorgung & Beschreibung der Hardware (Jonas F.):.....	15
4.1	Hardwareliste:	15
4.2	Beschreibung der wichtigsten Hardwarekomponenten	16
4.2.1	ESP8266-01S:	16
4.2.2	Raspberry Pi 4 2GB:	16
4.2.3	LM1117 3.3V Stepdown Board:.....	16
4.2.4	IGBT IRLZ44N Feldeffekttransistor:	16
4.2.5	Relais:.....	17
5	Entwicklung der Schaltung / Zeichnen des Schaltplans (Jonas F.):.....	18
5.1	Zeichnen des Schaltplans:	18
5.2	Erklärung der Schaltung:.....	19
5.2.1	Spannungsversorgung:	19
5.2.2	Hauptfunktion der Schaltung:	19
5.2.3	Zusatzfunktionen der Schaltung:	19
6	Aufbau Schaltung auf Steckboard (Jonas F.):	20
7	Aufbau Schaltung auf Lochrasterplatine (Jonas F.)	22
8	Versuchsaufbau (Jonas F.):	25
8.1	Versuchshardware:	25
8.2	Netzwerkumgebung:	27
	Vorbereitung Ipsum IQ Server(Jonas F.):	28
8.3	Installation Raspberry Pi OS	28

8.4	Steuerung des Raspberry PI's.....	30
8.4.1	Herausfinden der IP-Adresse:	30
8.4.2	Steuerung via SSH:.....	30
8.4.3	VNC-Service aktivieren:	31
8.5	Installation Ressourcen Ipsum Server:	32
8.5.1	Update des Betriebssystems:.....	32
8.5.2	Installation Npm	32
8.5.3	Installation der IPSUM IQ Webapp.....	33
8.6	Installation des Datenbankservers MariaDB:	34
8.6.1	Zugriff auf Datenbank ermöglichen (Remote Client Access):.....	36
9	Hardwareprogrammierung mit Arduino IDE (Bernhard R)	37
1.1	Programmierung	37
9.1	Debugging.....	37
9.2	Konfiguration des ESPs	38
9.2.1	WiFiManager (tzapu, 2023).....	38
9.2.1.1	Installation des WifiManagers.....	38
9.2.1.2	Funktionsweise	38
9.2.1.3	Zusätzliche Parameter	39
9.2.1.4	Das Endergebnis des WifiManagers:	40
9.2.2	Speichern der Daten	40
9.2.2.1	EEPROM	40
9.3	Websocket	42
9.4	Interaktion mit der Hardware	44
9.4.1	GPIO-Pins	45
9.4.2	Zeitmessung	45
9.4.2.1	Reset Timer / Knopf	45
9.4.3	Erkennung steigender Flanken.....	46
9.4.4	Ein und Ausschaltung der Lampe.....	47
10	Frontend (Kilian Grassauer).....	48
10.1	Präsentationsebene	48
10.1.1	Corporate Design	48
10.1.1.1	Corporate Design bei Ipsum IQ	48
10.1.1.1.1	Logo	48
10.1.1.1.2	Farben.....	48
10.1.1.1.3	Schriftarten.....	49
10.1.1.1.4	Grafiken	49

10.1.1.1.4.1	Favicon	49
10.1.1.1.4.2	SVG	49
10.1.1.1.4.3	Sonstige Grafiken	49
10.1.1.1.4.4	Glühbirne	50
10.2	Frontend-Development	50
10.2.1	Tech Stack	50
10.2.1.1	Tech Stack bei Ipsum IQ	50
10.2.2	HTML	50
10.2.3	CSS	50
10.2.4	CSS-Frameworks	50
10.2.5	Tailwind	51
10.2.5.1	Tailwind-Installation	51
10.2.6	React.js	52
10.2.6.1	React Hooks	52
10.2.6.1.1	useState	52
10.2.6.1.2	useEffect	52
10.2.6.2	Installation von React und Erstellung eines Projektes	52
10.2.7	Login	53
10.2.8	Mainpage Komponente	55
10.2.9	ESP Komponente	55
10.2.9.1	Glühbirnen-Button	56
10.2.9.2	Inputfelder	56
10.2.10	Timer Komponenten	58
10.2.10.1	Timer.tsx	58
10.2.10.2	Clock.tsx	59
10.2.11	About Komponente	61
10.2.12	Log Komponenten	62
10.2.12.1	Querylist.tsx	62
10.2.12.2	Entries.tsx	63
10.2.13	Usershow	63
10.2.14	React Router	63
10.2.15	Navbar und Mobile-Menu	64
10.2.16	Stylen in verschiedenen Browsern	65
11	Webanwendung: Backend (Goran K.)	66
11.1	Backend	66
11.2	JavaScript	66

11.3	Node.js	66
11.4	Node Package Manager (NPM)	66
11.5	Ein NPM-Projekt erstellen	66
11.6	Package.json	66
11.7	Express.js	67
11.7.1	Express Routing	67
11.8	Datenbanken	68
11.9	Projektdatenbank	68
11.10	Authentifizierung	69
11.10.1	Allgemein	69
11.10.2	Erstellung der Benutzer und Passwörter	69
11.10.3	Login	70
11.10.4	Login in React	71
11.10.5	Session Cookies	73
11.10.6	Prüfen, ob der User eingeloggt ist	74
11.10.7	Logout Button	75
11.11	WebSocket Server	76
11.12	Gruppierung aller Clients in einen Array	77
11.12.1	SocketIO Server	78
11.13	Steuerung des Lichtes	79
11.14	Automatische Dokumentation	80
11.14.1	Anzeigen der Einträge	81
11.15	Automatisches Ausschalten / Berechnung des Timers	82
11.15.1	Abbrechen des Timers	83
12	Literaturverzeichnis	85
13	Abbildungsverzeichnis	88

1 Einführung (Goran K.)

Im folgenden Abschnitt wird das Abschlussprojekt genauer definiert.

1.1 Projektfindung

Das Projektteam hat nach einem Weg gesucht, Strom einzusparen. Nach etwas Überlegung kam das Team zum Entschluss, etwas in Richtung Smart Home entwickeln zu wollen. Dadurch ist das Team auf die Idee IPSUM IQ gekommen.

1.2 Ausgangslage

In Zeiten von hohen Energiekosten sollte Licht nur bei Bedarf eingeschaltet sein. Häufig aber passiert es, dass man vergisst das Licht wieder auszuschalten und dadurch Energie verschwendet wird. Mit diesem Projekt wird eine Lösung für dieses Problem angeboten.

1.3 Projektziele

Ein herkömmlicher Lichtschalter sollte durch einen smarten ersetzt werden. Dieser sollte mittels Weboberfläche bedienbar sein und sich nach einer einstellbaren Zeit wieder ausschalten. Die Oberfläche sollte in responsivem Design ausgeführt sein. Ungebetener Zugriff wird mittels Authentifizierung verhindert. Jede Änderung des Schaltzustandes wird automatisch dokumentiert.

1.3.1 Muss-Kriterien

- Steuerung des Lichtes über die Webseite
- Login-System
- Responsive Webseite
- Automatische Dokumentation
- Licht schaltet sich nach einstellbarer Zeit selbst aus

1.3.2 Soll-Kriterien

- Eine „About“ Unterseite
- Ein Knopf zum Löschen der Datenbankeinträge
- Ein Logout Button

2 Projektorganisation

Die Projektorganisation des Abschlussprojekts gliedert sich in folgende Bereiche.

2.1 Projektrollen

Auftraggeber: HTL Saalfelden

Hauptbetreuer: Dietmar Duft

Nebenbetreuer: Herbert Kapeller

Nebenbetreuer: Bernhard Schwaiger

Projektleiter: Bernhard Rieder

Projektteam:

Jonas Feyersinger

Goran Kosić

Kilian Grassauer

2.2 Projektzeitplan

Im Zuge der Projektanalyse ergaben sich für die Realisierung des Projekts folgende

Zeiterfordernisse:

Tätigkeit	Zeitaufwand (h)	Datum Beginn:	Datum Ende:			
Pflichtenheft						
Präsentieren						
Beschaffung der Hardware						
Einrichten der Entwicklungsumgebung						
Verdrahtungsplan/Schaltplan erstellen						
Installation Betriebssystem RPI						
Verkabelung Hardware auf Steckboard						
Installation/Betrieb benötigter Systemsoftware						
Fertigstellung eigener Softwareanwendung						
Testumgebung, Simulation						

Goran Kosić		Bernhard Rieder		Kilian Grassauer		Jonas Feyersinger	
-------------	--	-----------------	--	------------------	--	-------------------	--

2.3 Zeitaufstellung Projektteam

Die Projektmitarbeiter schätzen ihren Zeitaufwand zur Realisierung der einzelnen Aufgaben folgendermaßen ein:

Tätigkeiten Jonas Feyersinger	Stunden
Erstellung des Pflichtenhefts	5h
Beschaffung der Hardware	5h
Verdrahtungsplan / Schaltplan	10h
Verkabelung Hardware auf Steckboard	10h
Aufbau der Testumgebung	30h
Erstellung der Dokumentation	30h
Gesamtstunden	90 h

Tätigkeiten Bernhard Rieder	Stunden
Erstellung des Pflichtenhefts	5h
Einrichtung der Entwicklungsumgebung	10h
Konfiguration des ESP8266	15h
Fertigstellung der Softwareanwendung	30h
Dokumentation	30h
Gesamtstunden	90 h

Tätigkeiten Goran Koscic	Stunden
Erstellung des Pflichtenhefts	2h
Einrichtung der Entwicklungsumgebung	3h
Modellierung der Datenbank	2h
Verbindung Webserver - Datenbank	3h
Erstellung Login-System	10h
Automatische Einträge in DB	20h
Fertigstellung der Softwareanwendung	20h

Dokumentation	30h
Gesamtstunden	90h
Tätigkeiten Kilian Grassauer	
Erstellung des Pflichtenhefts	5h
Einrichtung der Entwicklungsumgebung	5h
Dokumentation	30h
Logo Erstellung	5h
Erstellung/Design Hauptseite	20h
Erstellung/Design Login-Seite	15h
Erstellung/Design Unterseiten	10h
Gesamtstunden	90h

3 Allgemeines

Es wurde entschieden, für das Hosten der Webanwendung einen zentralen Server zu benutzen. Für jede Lampe und jeden Knopf gibt es einen Mikrocontroller, welcher sich mit dem Server über WLAN verbindet:

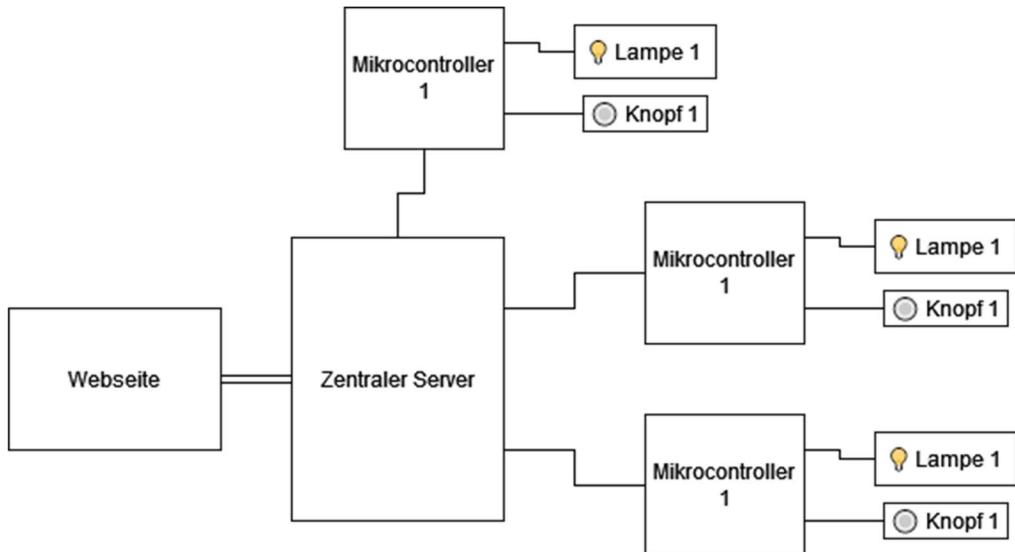


Abbildung 1: Organigramm der Hardware

Als zentralen Server wird ein Raspberry Pi 4 benutzt und als Micro Controller wurde der ESP8266-01S gewählt.

3.1 Github / Git (Bernhard R)

Git ist ein Quellcodemanagementsystem, welches mehreren Programmierern erleichtert an demselben Projekt, dem so genannten Repository, gleichzeitig zu arbeiten. Git wurde 2005 von Linus Torvalds erstellt. Der Git-Client kann mit einer Vielzahl von Servern verwendet werden, unter anderen GitLab, BitBucket, Gitea und GitBucket. Für dieses Projekt wurde der von Microsoft betriebene Git-Server GitHub verwendet. Es wurde entschieden, das Repository während der Entwicklungsarbeiten privat zu halten und sobald die Entwicklungen abgeschlossen waren, zu veröffentlichen. Das Repo befindet sich unter dem Link: <https://github.com/Moosbee/ipsum-iq>.

4 Besorgung & Beschreibung der Hardware (Jonas F.):

4.1 Hardwareliste:

<u>Name</u>	<u>Herkunft des Bauteils</u>	<u>Nutzen im Projekt</u>
Raspberry Pi 4 2GB RAM	Besitz Jonas Feyersinger	Dient als Server, sprich auf ihm laufen die Datenbank und die Webapplikation.
Netzteil Raspberry Pi 4	Besitz Jonas Feyersinger	Spannungsversorgung des Raspberry Pi's
Verbatim 16 GB Micro SD Card	Besitz Jonas Feyersinger	Dient als Speichermedium des Raspberry Pi's
Epoxid Lochrasterplatine	Besitz Jonas Feyersinger	Hauptplatine des Mikrocontrollers
Steckboard / Breadboard	Besitz Jonas Feyersinger	Test der Schaltung / 2te Schaltung im Versuchsaufbau
2x ESP8266-01S Mikrocontroller	Besitz Bernhard Rieder	Microcontroller, die mit dem Server verbunden die Lampen ansteuern
2x LM1117 3.3V Stepdown Board	Von Schule zur Verfügung gestellt	Spannungsversorgung des Mikrocontrollers / Bauteil der Schaltung
2x Metallschichtwiderstand 10kΩ	Besitz Jonas Feyersinger	Bauteil der Schaltung
2x Metallschichtwiderstand 100kΩ	Besitz Jonas Feyersinger	Bauteil der Schaltung
4x Metallschichtwiderstand 1kΩ	Besitz Jonas Feyersinger	Bauteil der Schaltung
SIL-Sockel weiblich 2x4 Pins	Von Schule zur Verfügung gestellt	Bauteil der Schaltung
2x SIL-Sockel männlich 2 Pins	Besitz Jonas Feyersinger	Bauteil der Schaltung
2x IOR IRLZ44N FET Transistor	Von Schule zur Verfügung gestellt	Bauteil der Schaltung
OEG OJ-SS-112DM 12V Relais	Von Schule zur Verfügung gestellt	Bauteil der Schaltung
Fujitsu F1CL012R 12V Relais	Von Schule zur Verfügung gestellt	Bauteil der Schaltung
2x 6x6x4,3 4 Pin-Mikrotaster	Besitz Jonas Feyersinger	Reset Button / Bauteil der Schaltung
2x Busch Jaeger 230V Taster	Besitz Jonas Feyersinger	Manueller Lichtschalter / Bauteil der Schaltung

4.2 Beschreibung der wichtigsten Hardwarekomponenten

4.2.1 ESP8266-01S:

Der ESP8266-01S ist ein Mikrocontroller, der sich mit dem Server verbindet und sich über die Webapp steuern lässt. Dieser öffnet und schließt das Relais und nimmt die manuelle Eingabe über den Taster entgegen.

Der ESP8266 ist ein System auf einem Chip (SoC), das von der chinesischen Firma Espressif hergestellt wird. Es gibt viele verschiedene Module, wie der NodeMCU oder der WeMos D1. Die Entscheidung fiel auf das ESP8266 ESP-01 Modul -oder kurz ESP-01, da es ein kleines und kostengünstiges Modul darstellt und sehr stromsparend ist. Da das Modul keinen internen Spannungsregler besitzt und nur 6 GPIO-Pins hat ist das Programmieren des Moduls nur über einen externen USB Programmer möglich. (ESP8266 Wikipedia, 2023)



Abbildung 2 ESP8266-01S / eigene Aufnahme

4.2.2 Raspberry Pi 4 2GB:

Der Raspberry Pi 4 ist ein kostengünstiger und verhältnismäßig leistungsstarker ARM Einplatinencomputer, auf dem Linux installiert ist. Hergestellt und verkauft wird er von der Raspberry Pi Foundation. Er wird als Server eingesetzt, sprich auf dem Pi wird die Webapp gehostet. Dieser nimmt Befehle von der Benutzeroberfläche entgegen und schickt diese über das Netzwerk an den ESP8266-01S weiter. (Raspberry Pi 4 Model B kaufen, 2023)



Abbildung 3 Raspberry PI 4 2Gb / eigene Aufnahme

4.2.3 LM1117 3.3V Stepdown Board:

Das LM1117 3.3V Stepdown Board wandelt Spannungen von 6 – 12V DC in 3.3V DC um. Es versorgt die Schaltung mit Spannung. Das Herzstück des Boards ist der Voltage Regulator Chip LM1117, der von Texas Instruments entwickelt wurde. (LM1117 800-mA, Low-Dropout Linear Regulator datasheet (Rev. Q), 2023)

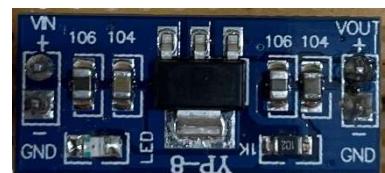


Abbildung 4 3.3V Stepdown Board / eigene Aufnahme

4.2.4 IOR IRLZ44N Feldeffekttransistor:

Der IOR IRLZ44N ist ein Feldeffekttransistor, der von der Firma Infineon entwickelt wurde. Die Entscheidung fiel auf diesen Feldeffekttransistor, da er nur am Beginn des Schaltprozesses einen Stromstoß benötigt. Dies bedeutet weniger Last am ESP8266, was wiederum bedeutet, dass die Schaltung länger ohne einen möglichen Defekt arbeiten kann. (IRLZ44N - Infineon Technologies, 2023)



Abbildung 5 IOR IRLZ44N FET/ eigene Aufnahme

4.2.5 Relais:

Als Relais wird ein 12Volt Relais benutzt, dass bis zu 250 Volt bei 8 Ampere schalten kann. Das Relais schaltet den Endverbraucher aus oder ein. Bei den beiden Schaltungen wurden verschiedene Relais verwendet. Auf der Lochrasterplatine wurde ein OEG OJ-SS-112DM (ENG_DS_OJ_OJE_series_relay_data_sheet_E_1120.pdf, 2023) verwendet und auf dem Steckboard ein Fujitsu F1CL012R (Datasheet: FTR-F1R relay - ftr_f1r-944176.pdf, 2023). Die beiden Relais sind fast identisch nur das Fujitsu hat 4 Pins mehr, die in unserer Schaltung jedoch nicht benutzt werden.



Abbildung 6 FUJITSU F1CL012R 12V Relais /
eigene Aufnahme

5 Entwicklung der Schaltung / Zeichnen des Schaltplans (Jonas F.):

5.1 Zeichnen des Schaltplans:

Der Schaltplan wurde mithilfe der Open Source Software Fritzing entwickelt (Learning Electronics with Fritzing, 2023). Die Handhabung der Software ist einfach und war schnell erlernt:

Bauteile können in diesem Feld ausgewählt werden und mittels Drag & Drop ins Zeichenfeld gezogen werden:



Abbildung 7 Fritzing Bauteile Menü / eigene Aufnahme

Anschließend werden sie durch Anklicken und Ziehen miteinander verbunden wie z.B hier in dieser Beispielschaltung:

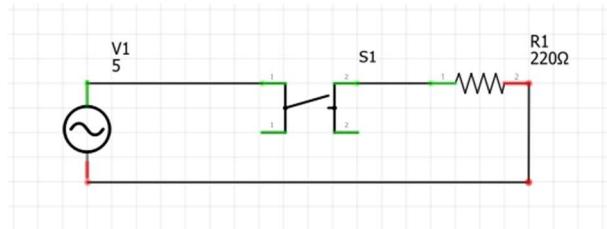
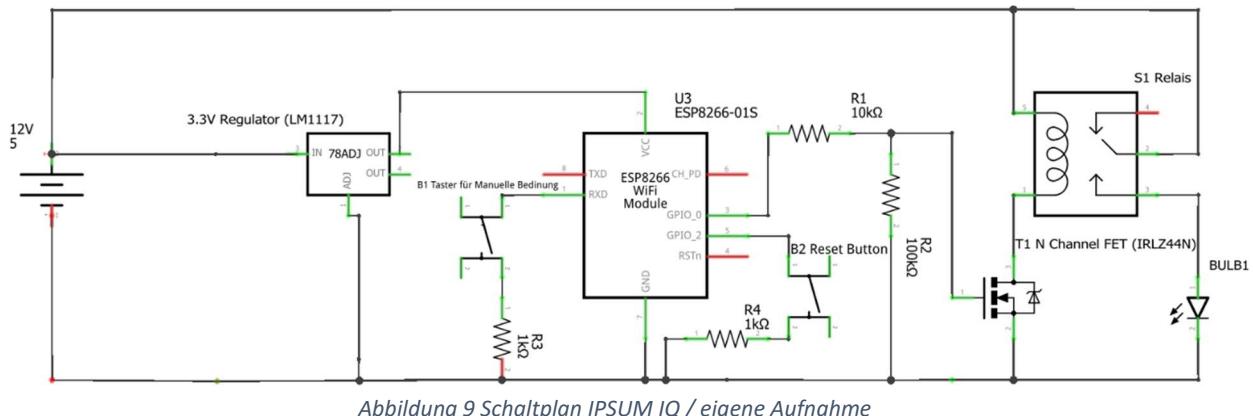


Abbildung 8 Fritzing Beispielschaltplan / eigene Aufnahme

Mit diesem einfachen Prinzip wurde der Schaltplan gezeichnet:



5.2 Erklärung der Schaltung:

5.2.1 Spannungsversorgung:

Als Spannungsversorgung dient ein 1.5A Schaltnetzteil. Dieses versorgt zum einen die LED-Spots, aber auch die Schaltung, welcher noch zusätzlich ein 3.3V Spannungsregler (LM1117) vorgeschalten ist.

5.2.2 Hauptfunktion der Schaltung:

Der Verbraucher sollte über die Webapplikation, aber auch direkt an der Schaltung gesteuert werden können. Das wird über ein Relais realisiert, welches eine Betriebsspannung von 12V benötigt wobei der ESP aber mit 3.3V arbeitet. Die Lösung ist der IOR IRLZ44N Feldeffekttransistor. Er wird vom ESP über den GPIO 0 Pin leitend oder sperrend geschalten. Der Transistor wiederum schaltet auf der Masseseite das Relais ein oder aus. Somit kann über einen Umweg das Relais mit dem ESP angesteuert werden.

5.2.3 Zusatzfunktionen der Schaltung:

Um die lokale Ansteuerung zu ermöglichen, wird ein Schalter benötigt. Dieser schließt die Verbindung zwischen den Pin RX und dem Minuspol. Er wird mit der SIL-Steckverbindung mit der Platine verbunden und ist somit genauso wie der ESP tauschbar. Außerdem besitzt die Schaltung eine RESET Funktion. Falls man den ESP mit einem neuen Netzwerk verbinden will, kann man ihn zurücksetzen und den Konfigurationsprozess erneut starten.

6 Aufbau Schaltung auf Steckboard (Jonas F.):

Um die Funktion der Schaltung zu überprüfen, wurde sie zuerst auf dem Steckboard aufgebaut. Dieser Versuchsaufbau wurde gleich zur Ansteuerung eines weiteren Verbrauchers belassen. Um Bauteile auf dem Steckboard miteinander zu verbinden, wurde ein Stück Cat 7 Verlegekabel benutzt, da es fast perfekt in die Löcher des Breadboards passt. Der ESP8266-01S wurde mittels Jumper Kabel mit dem Board verbunden. Hier zu sehen:

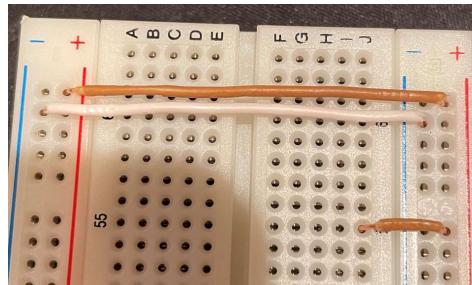


Abbildung 10 Steckboard Beispiel / eigene Aufnahme

Nachdem alles mit dem Draht verbunden war, wurden die Bauteile in das Steckboard gesteckt:

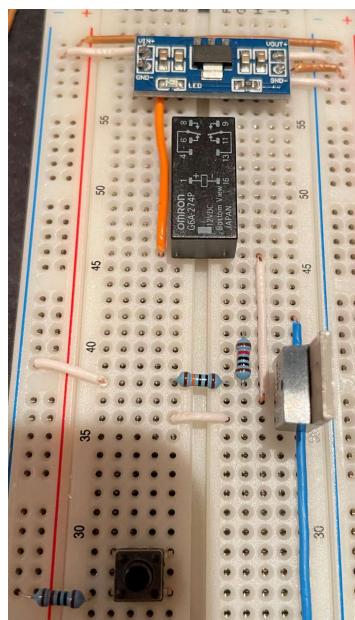


Abbildung 11 Steckboard IPSUM IQ halbfertig / eigene Aufnahme

Die folgende Abbildung zeigt den fertigen Aufbau:

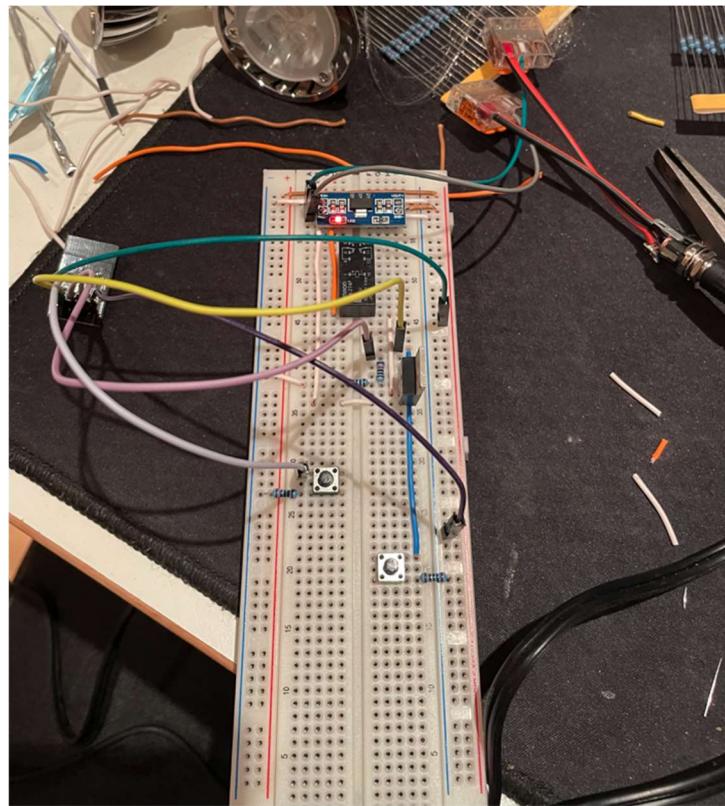


Abbildung 12 Steckboard IPSUM IQ fertig / eigene Aufnahme

Die Schaltung wurde mittels WAGO-Klemmen mit dem Buchsen Stecker verbunden. Die Schaltung funktionierte und es wurde mit dem Bau des Lochraster Prototypen begonnen.

7 Aufbau Schaltung auf Lochrasterplatine (Jonas F.)

Nachdem sicher war, dass die Schaltung funktioniert, wurde der Lochraster Prototyp hergestellt. Benutzt wurden eine Lötstation, die auf 300° Celsius eingestellt war, Lötzinn mit Flux und Lötfett. Fädeldraht wurde verwendet, um die Leiterbahnen zu legen.

Hier sieht man die Unterseite der Platine. Der SIL-Sockel des ESP8266-01S wird durch das Step Down Modul bereits mit Spannung versorgt. Auf die zu verlötende Stelle wurde Lötfett aufgetragen, Fädeldraht verlegt und anschließend verlötet.

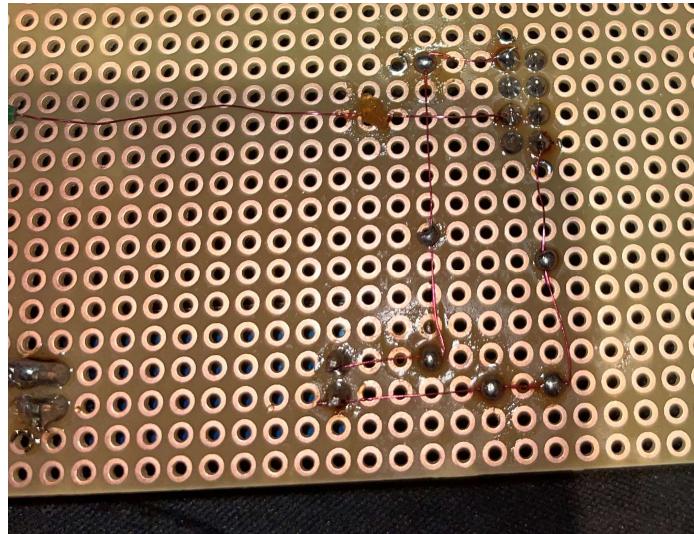


Abbildung 13 Beispiel Fädeldraht / eigene Aufnahme

Hier die Oberseite der Platine. Zu sehen ist oben der 3.3V Wandler in Blau, mittig dem 10k Widerstand zwischen GPIO0 und Gate des Transistors und rechts unten der weibliche SIL-Sockel für den ESP:

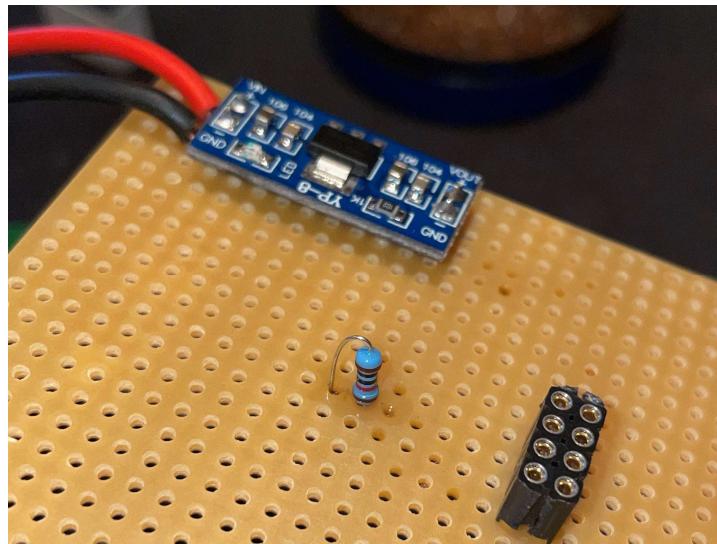


Abbildung 14 Lochraster Bild 1 / eigene Aufnahme

Der Transistor, das Relais und die Widerstände sind hier zu sehen. Die Leitung vom GPIO 0 Pin zum Transistor wurde über die Oberseite gelegt:

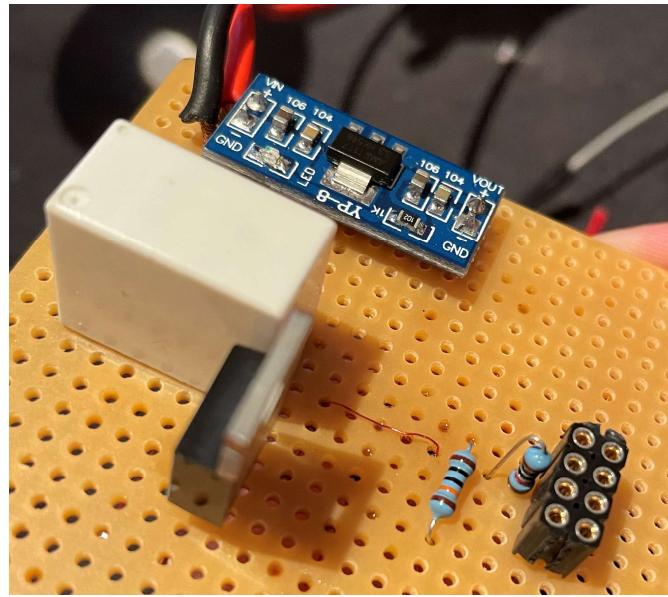


Abbildung 15 Lochraster Bild 2 / eigene Aufnahme

Als nächstes wurde der Reset Button verbaut. Er wird benötigt, um den ESP neu konfigurieren zu können. Die Leitung wurde an der Oberseite verlegt. Im unteren Bereich der Taster mit dem zugehörigen 1k Widerstand und dem Fädeldraht:

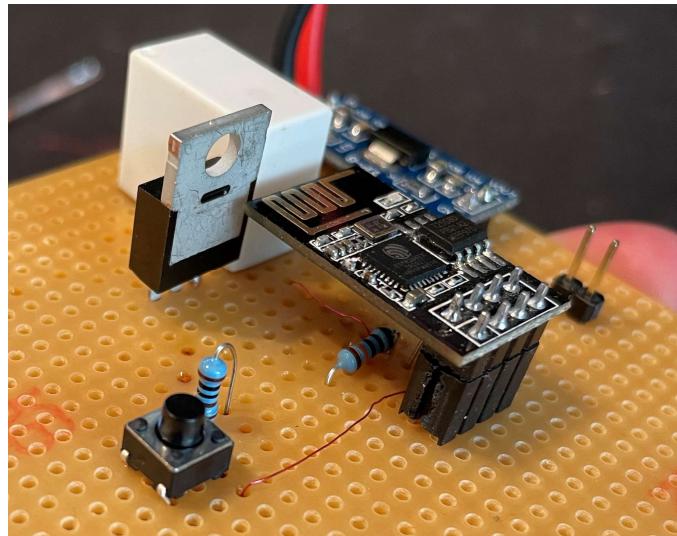


Abbildung 16 Lochraster Bild 3 / eigene Aufnahme

Da der Taster zur manuellen Bedienung nicht auf der Platine liegen soll wurde ein männlicher SIL-Sockel auf der Platine aufgelötet. Er ermöglicht den Anschluss des Tasters mittels Kabel und dessen externe Positionierung.

Hier neben dem 3.3V Wandler ist der SIL-Sockel und der zugehörige 1k Widerstand zu sehen.

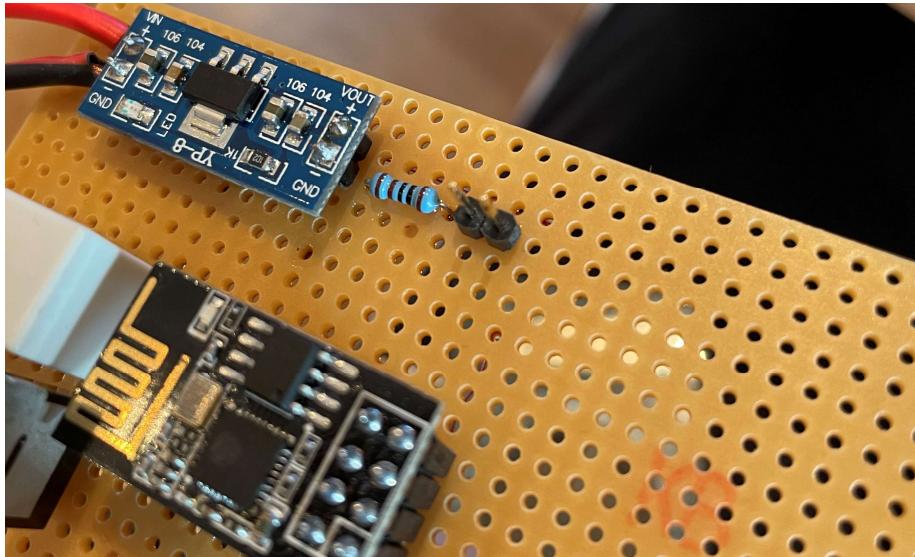


Abbildung 17 Lochraster Bild 4 / eigene Aufnahme

Hier noch Aufnahmen der fertigen Schaltung:

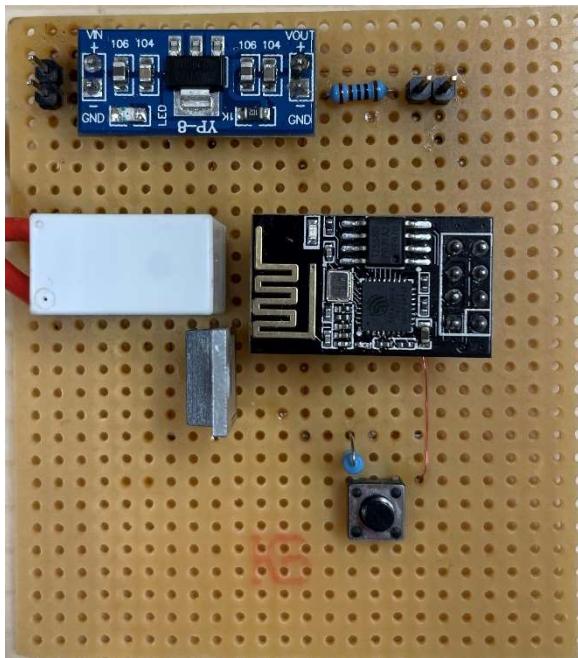


Abbildung 19 Lochrasterplatine IPSUM IQ Vorderseite / eigene Aufnahme

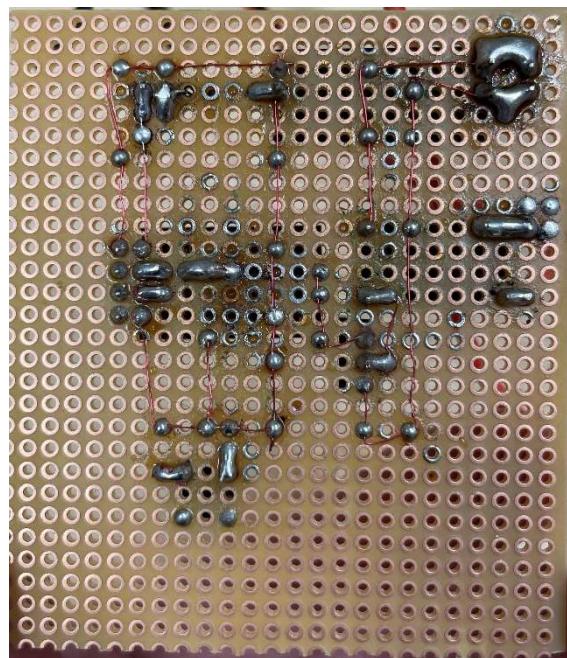


Abbildung 18 Lochrasterplatine IPSUM IQ Rückseite / eigene Aufnahme

8 Versuchsaufbau (Jonas F.):

8.1 Versuchshardware:

Zur besseren Präsentation des Projekts wurde ein Holzgestell vorbereitet:



Abbildung 20 IPSUM IQ Versuchsaufbau leer / eigene Aufnahme

Um die Schaltungen mit Spannung zu versorgen und um die Spots zu verbinden wurden WAGO-Klemmen benutzt. Die 12V LED-Spots sitzen frontal zwischen Logo und Taster im Holzgestell. Die Taster wurden mit Unterputz Dosen im Gestell befestigt.

Hier die fertige Frontansicht:



Abbildung 21 IPSUM IQ Versuchsaufbau Vorderseite / eigene Aufnahme

Die Rückansicht:

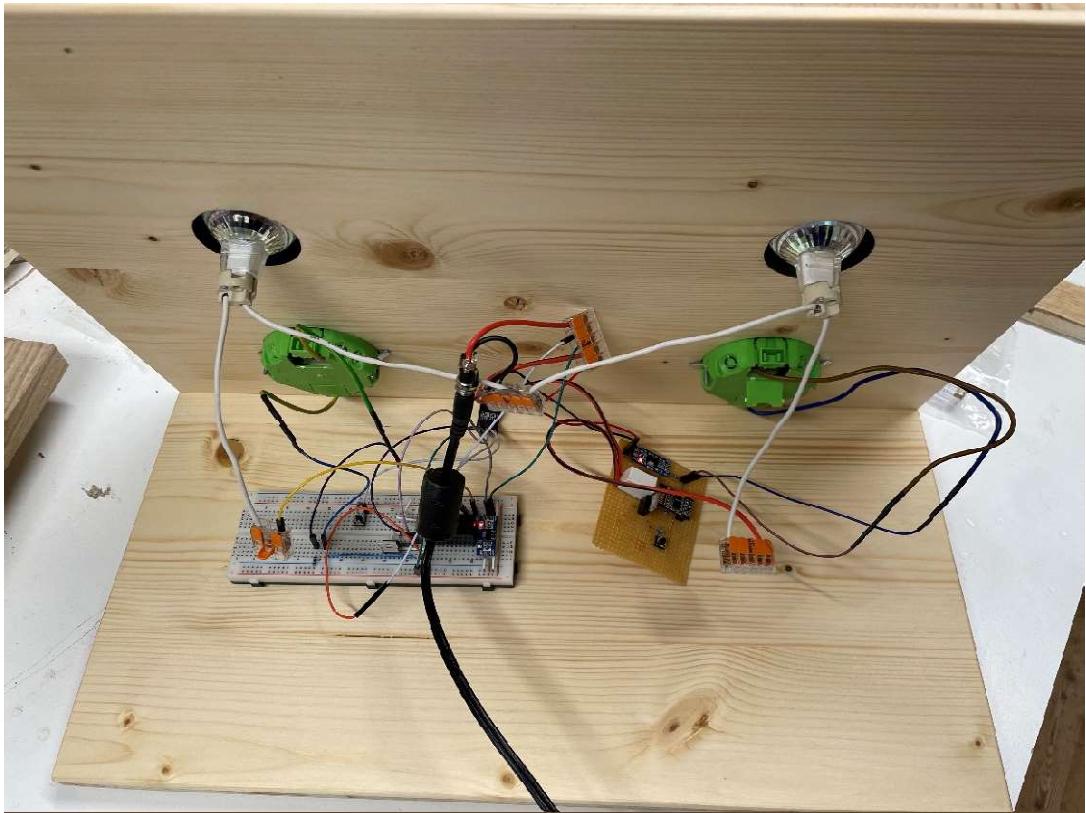


Abbildung 22 IPSUM IQ Versuchsaufbau Rückseite / eigene Aufnahme

Beide Schaltungen und Spots beziehen Spannung vom selben Netzteil. Alle Kabel, die mit den WAGO-Klemmen in Berührung kommen, wurden mit Aderendhülsen bearbeitet. Um die 230V Busch-Jaeger Taster am Steckboard und an der Lochrasterplatine anzuschließen wurden Adapter von 1.5mmØ starren Kabel zu flexiblen Jumper Kabeln gefertigt:

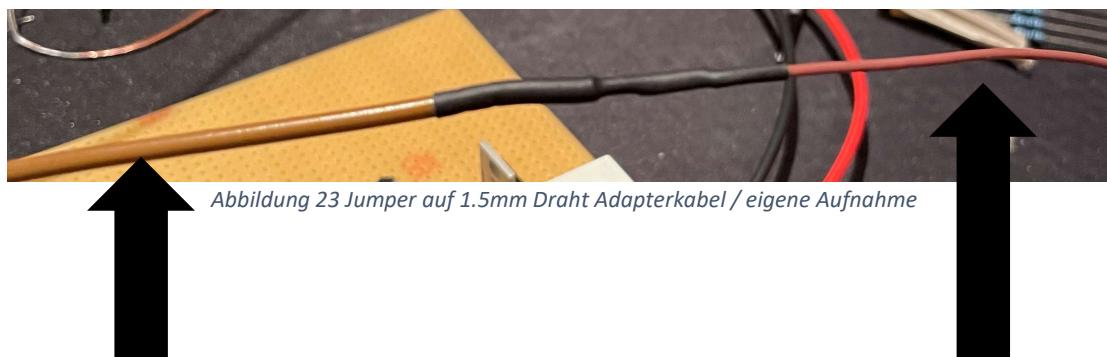


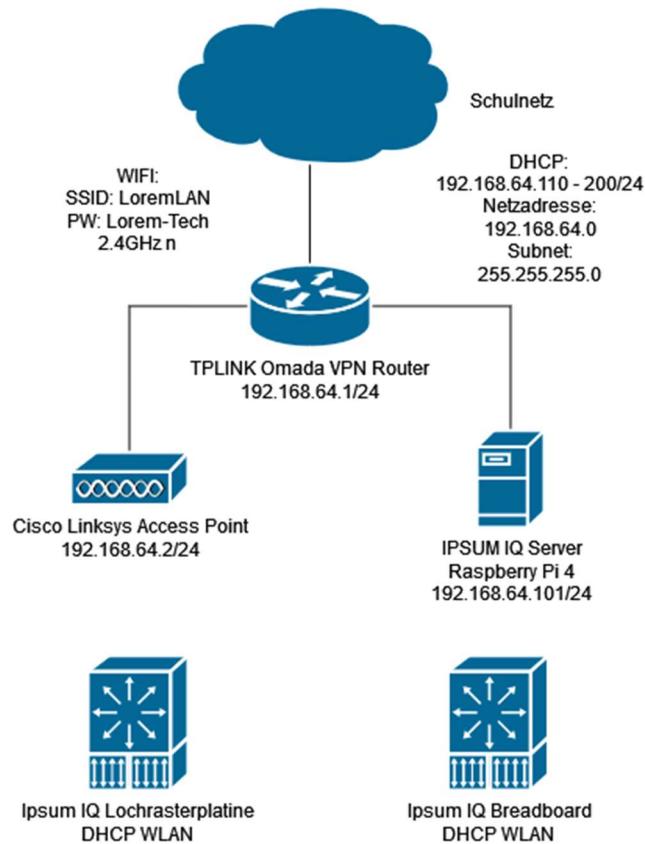
Abbildung 23 Jumper auf 1.5mm Draht Adapterkabel / eigene Aufnahme

1.5mm Ø starres Kabel
Taster Seite

Jumper Kabel
Seite der Schaltung

8.2 Netzwerkumgebung:

Die Kommunikation der einzelnen Komponenten wird über eine Netzwerkverbindung hergestellt. Im Folgenden ist der Aufbau dieses Netzwerkes beschrieben:



Der Access Point sendet das LoremLAN WLAN aus mit dem sich die ESP's verbinden. Der DHCP-Server und die Netzwerkverbindung wird vom TPLINK Omada VPN Router bereitgestellt.

Vorbereitung Ipsum IQ Server(Jonas F.):

Der Server wird benötigt, um die Webapplikation zu hosten und die Befehle (Ein, Aus, Timer, ...) an die Schaltungen weiterzugeben. Dafür wird ein Raspberry Pi 4 2gb verwendet, da dieser stromsparend, günstig in der Anschaffung und platzsparend ist.

8.3 Installation Raspberry Pi OS

Der Raspberry Pi besitzt keinen internen Speicher, deshalb bootet man hier von einer Micro SD Karte. Das Betriebssystem wird mit dem Raspberry Pi Imager auf die SD-Karte geflasht: Dies ist das Hauptfenster des Tools. Hier wählt man aus, welches Betriebssystem auf welche SD-Karte geflasht wird. In unserem Fall wurde Raspberry Pi OS 32 Bit gewählt.



Abbildung 24 Raspberry Pi Imager Hauptseite / eigene Aufnahme

Der PI soll nicht via Monitor und Tastatur gesteuert werden, sondern Headless laufen. Dies bedeutet, dass der Pi rein über das Netzwerk ohne jegliche Peripheriegeräte gesteuert wird. Dies gelingt, indem SSH (Secure Shell) aktiviert wird und dem Pi ein Benutzername und ein Passwort zugewiesen wird.

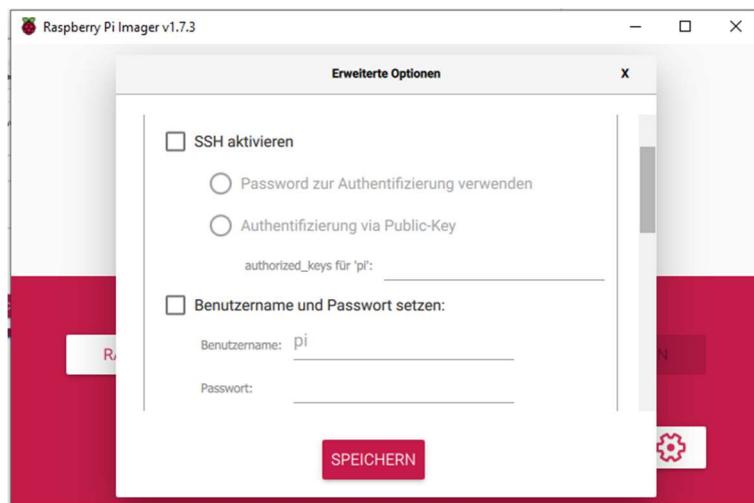


Abbildung 25 Raspberry Pi Imager Erweiterte Optionen / eigene Aufnahme

Nachdem eine SD-Karte gewählt wurde (in unserem Fall Generic Storage Device) kann der Flash-Vorgang gestartet werden.



Abbildung 26 Raspberry Pi Imager Schreiben / eigene Aufnahme

Ist der Schreibvorgang abgeschlossen, wird die SD-Karte in den Raspberry Pi gesteckt.

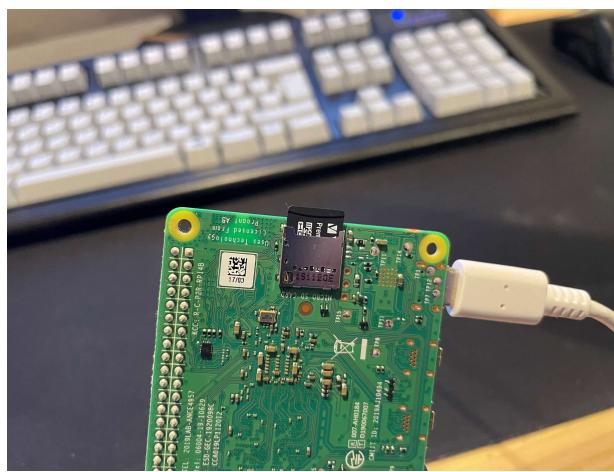


Abbildung 27 IPSUM IQ Server Rückseite / eigene Aufnahme

Damit der Pi Headless betrieben werden kann, benötigt er eine stabile Netzwerkverbindung und muss im gleichen Netzwerk sein wie der PC, mit dem man ihn steuern will.



Abbildung 28 IPSUM IQ Server Vorderansicht / eigene Aufnahme

8.4 Steuerung des Raspberry PI's

8.4.1 Herausfinden der IP-Adresse:

Um mittels SSH auf den Server zuzugreifen, wird zuerst die IP-Adresse benötigt. Diese kann mittels Netzwerkscan mit dem Tool Colasoft MAC Scanner herausgefunden werden. Den Raspberry Pi erkennt man an dem Hersteller (aus Datenschutzgründen wurden die restlichen Adressen und die MAC-Adresse zensiert):

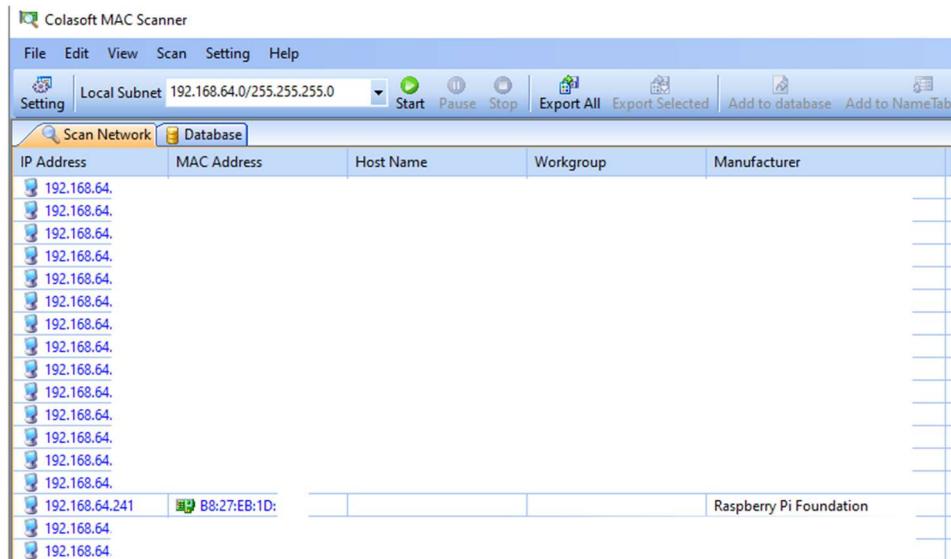


Abbildung 29 Colasoft MAC Scanner / eigene Aufnahme

8.4.2 Steuerung via SSH:

Mittels ssh-Befehl kann im Windows-Terminal oder der Power-Shell eine SSH-Verbindung zum Raspberry hergestellt werden:

ssh pi@192.168.64.241.

Bei der Passwortabfrage wird das zuvor festgelegte Kennwort eingegeben. Jetzt kann der Pi über das Netzwerk gesteuert werden.

The screenshot shows a Windows Terminal window titled "pi@raspberrypi: ~". It is a PowerShell session. The output shows the following text:
Windows PowerShell
Copyright (C) Microsoft Corporation. Alle Rechte vorbehalten.
Lernen Sie das neue plattformübergreifende PowerShell kennen – <https://aka.ms/pscore6>
PS C:\Users\Jonas> ssh pi@192.168.64.241
pi@192.168.64.241's password:
Linux raspberrypi 5.15.76-v7+ #1597 SMP Fri Nov 4 12:13:17 GMT 2022 armv7l

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Mon Jan 16 11:54:54 2023 from 192.168.64.68

Wi-Fi is currently blocked by rfkill.
Use raspi-config to set the country before use.

pi@raspberrypi:~ \$ |

Abbildung 30 Windows Terminal SSH / eigene Aufnahme

8.4.3 VNC-Service aktivieren:

Als erstes muss das Tool `raspi-config` geöffnet werden:

```
sudo raspi-config
```

Diese Oberfläche wird mittels Pfeiltasten und Enter gesteuert. Dort kann unter Interface Options > VNC > Yes der VNC-Server aktiviert werden. Dieser ermöglicht es den Raspberry Pi über die GUI zu steuern:

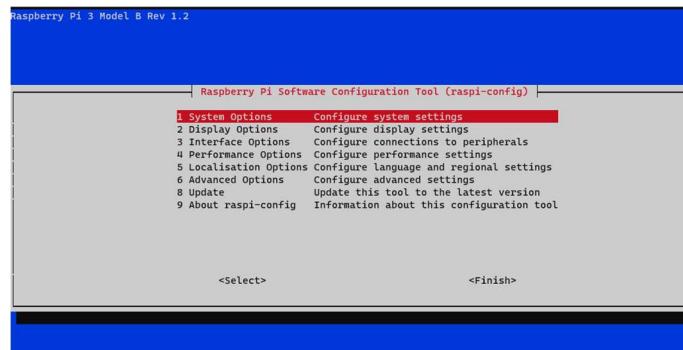


Abbildung 31 raspi-config / eigene Aufnahme

Nun wird der VNC-Viewer installiert. Auf dem Pi wird zugegriffen indem in der Leiste oben die zuvor herausgefundene IP-Adresse eingegeben wird. Bestätigt wird mit Enter. Bei der Passwortabfrage ist das gleiche Passwort einzugeben wie zuvor bei der SSH-Verbindung:

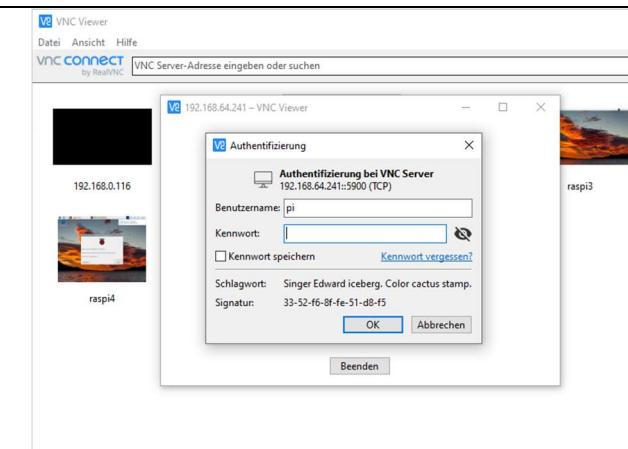


Abbildung 32 VNC Viewer Verbinden / eigene Aufnahme

:

Wenn die Verbindung gelingt wird man vom Desktop der Linux Umgebung begrüßt:

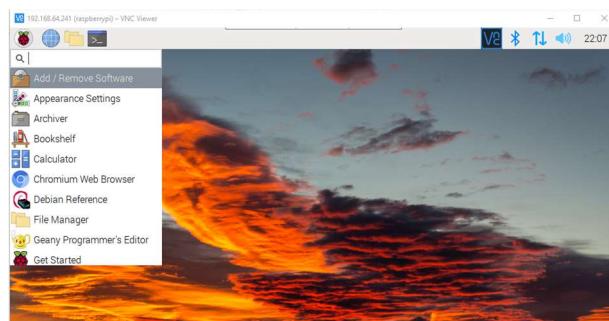


Abbildung 33 Raspberry Pi Desktop / eigene Aufnahme

8.5 Installation Ressourcen Ipsum Server:

Hier eine kleine Liste der Dienste, die auf dem Raspberry Pi laufen:

SSH (Secure Shell)	Zugriff auf Pi über Netzwerk (Text Mode)
VNC (Virtual Network Computing)	Zugriff auf Pi über Netzwerk (Grafischer Modus)
MariaDB	Datenbankserver für die Webapp IPSUM IQ
Frontend	Grafische Oberfläche der Website
Server / Backend	Logik hinter der Webapp

8.5.1 Update des Betriebssystems:

Damit die Installation der für den Server benötigten Programme, Ressourcen und Dienste reibungslos verläuft wird als erstes das gesamte OS aktualisiert. Ein Terminalfenster kann geöffnet werden mit der Tastenkombination:

Strg+alt+t

Der Befehl zum Updaten lautet `sudo apt update && sudo apt upgrade`
bestätigt wird zweimal mit Y für Yes (kann je nach Internetverbindung und Geschwindigkeit der SD-Karte einige Zeit in Anspruch nehmen):



```
pi@raspberrypi: ~
File Edit Tabs Help
pi@raspberrypi: $ sudo apt update && sudo apt upgrade
Hit:1 http://archive.raspberrypi.org/debian bullseye InRelease
Hit:2 http://raspbian.raspberrypi.org/raspbian bullseye InRelease
Hit:3 http://archive.raspbian.org/raspbian stretch InRelease
Reading package lists... 24%
```

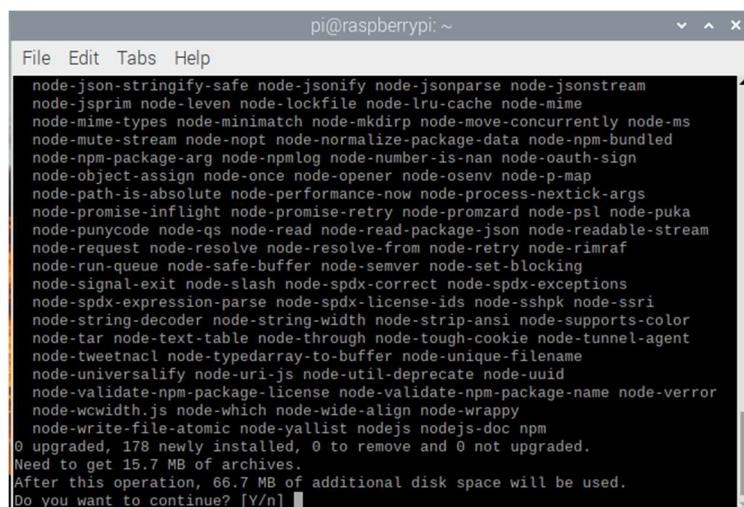
Abbildung 34 Terminal apt-get update / eigene Aufnahme

8.5.2 Installation Npm

Da alle benötigten Ressourcen für die Webapp über npm installiert werden können wird das Tool nun installiert. Der Befehl dazu lautet

sudo apt install npm

bestätigt wird mit Y und der Enter-taste:



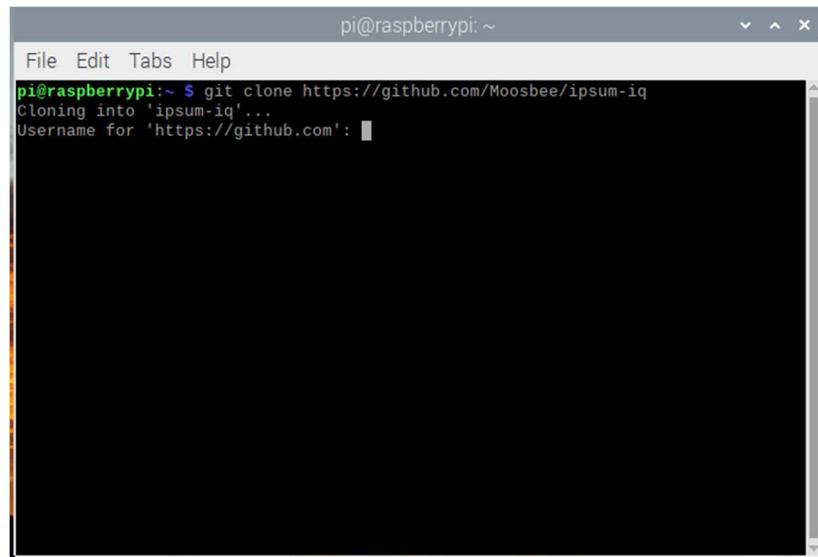
```
pi@raspberrypi: ~
File Edit Tabs Help
node-json-stringify-safe node-jsonify node-jsonparse node-jsonstream
node-jsprom node-leven node-lockfile node-lru-cache node-mime
node-mime-types node-minimatch node-mkdirp node-move-concurrently node-ms
node-mute-stream node-nopt node-normalize-package-data node-npm-bundled
node-npm-package-arg node-npmlog node-number-is-nan node-oauth-sign
node-object-assign node-once node-opener node-osenv node-p-map
node-path-is-absolute node-performance-now node-process-nextick-args
node-promise-inflight node-promise-retry node-promzard node-psl node-puka
node-punycode node-qs node-read node-read-package-json node-readable-stream
node-request node-resolve node-resolve-from node-retry node-rimraf
node-run-queue node-safe-buffer node-semver node-set-blocking
node-signal-exit node-slash node-spdx-correct node-spdx-exceptions
node-spdx-expression-parse node-spdx-license-ids node-sshpk node-ssri
node-string-decoder node-string-width node-strip-ansi node-supports-color
node-tar node-text-table node-through node-tough-cookie node-tunnel-agent
node-tweetnacl node-typedarray-to-buffer node-unique-filename
node-universalify node-uri-js node-util-deprecate node-uuid
node-validate-npm-package-license node-validate-npm-package-name node-verror
node-wcwidth.js node-which node-wide-align node-wrapify
node-write-file-atomic node-yallist nodejs-nodejs-doc npm
0 upgraded, 178 newly installed, 0 to remove and 0 not upgraded.
Need to get 15.7 MB of archives.
After this operation, 66.7 MB of additional disk space will be used.
Do you want to continue? [Y/n] 
```

Abbildung 35 apt-get install npm / eigene Aufnahme

8.5.3 Installation der IPSUM IQ Webapp

Der Source Code der Webapp liegt auf der Plattform GitHub. Um die App Auf den Server ausführen zu können, wird sie zuerst heruntergeladen:

```
git clone https://github.com/Moosbee/ipsum-iq
```



A screenshot of a terminal window titled "pi@raspberrypi: ~". The window shows the command "git clone https://github.com/Moosbee/ipsum-iq" being entered. The output of the command "Cloning into 'ipsum-iq'..." is visible, along with a prompt for a GitHub username.

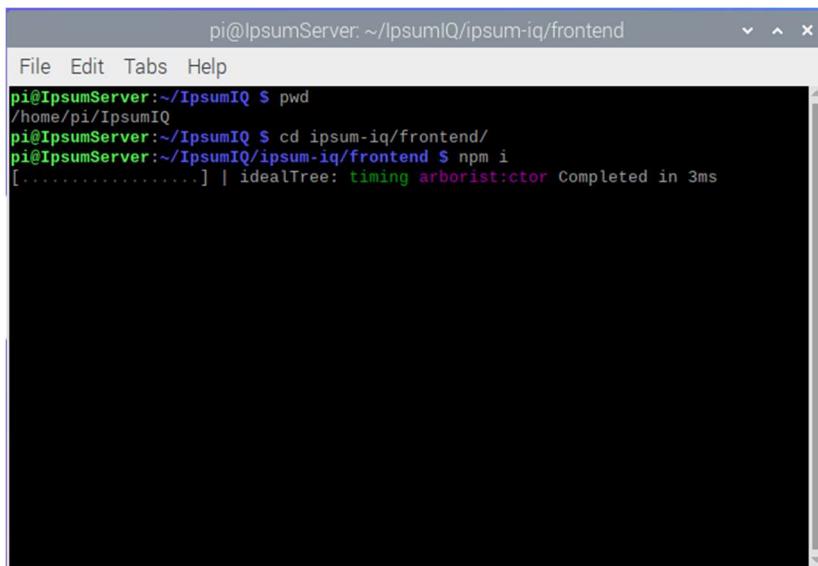
Abbildung 36 git clone / eigene Aufnahme

Nun wird im Verzeichnis, in dem der Befehl ausgeführt wurde ein Ordner erstellt mit den Inhalten des Repositorys. Wo dies ist, kann mit dem Befehl `pwd` herausgefunden werden. In dieses Verzeichnis wird nun gewechselt:

```
cd ipsum-iq
```

Jetzt können die Abhängigkeiten installiert werden:

```
npm i
```



A screenshot of a terminal window titled "pi@IpsumServer: ~/IpsumIQ/ipsum-iq/frontend". The window shows the command "npm i" being entered. The output includes the command itself and a message indicating that the idealTree timing arborist:ctor completed in 3ms.

Abbildung 37 npm i / eigene Aufnahme

8.6 Installation des Datenbankservers MariaDB:

Die Datenbank MariaDB wird mit NPM nicht mitinstalliert. Deshalb muss dieser manuell installiert werden. Das gelingt mit:

```
sudo apt install mariadb-server
```

Wieder mit Yes bestätigen. Hier bereits installiert:

```
pi@IpsumServer:~/IpsumIQ/ipsum-iq/frontend $ sudo apt install mariadb-server
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
mariadb-server is already the newest version (1:10.5.15-0+deb11u1).
The following package was automatically installed and is no longer required:
  libfuse2
Use 'sudo apt autoremove' to remove it.
0 upgraded, 0 newly installed, 0 to remove and 0 not upgraded.
pi@IpsumServer:~/IpsumIQ/ipsum-iq/frontend $
```

Abbildung 38 apt-get install mariadb-server / eigene Aufnahme

Nun kann die Datenbank importiert werden. Dazu wird die mariadb Shell gestartet:

```
sudo mysql -u root:
```

```
pi@IpsumServer:~/IpsumIQ/ipsum-iq/frontend $ sudo mysql -u root
Welcome to the MariaDB monitor.  Commands end with ; or \g.
Your MariaDB connection id is 39
Server version: 10.5.15-MariaDB-0+deb11u1 Raspbian 11

Copyright (c) 2000, 2018, Oracle, MariaDB Corporation Ab and others.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

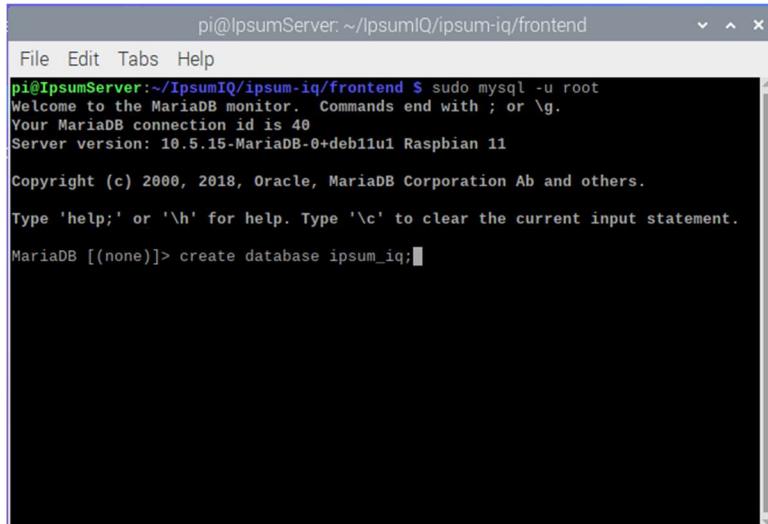
MariaDB [(none)]>
```

Abbildung 39 MySQL Shell / eigene Aufnahme

In der MySQL Shell kann via SQL mit dem Datenbankserver kommuniziert werden. Um nun die Datenbank zu importieren, wird zuerst eine neue Datenbank erstellt und diese dann mit der .sql Datei gefüllt. Dies gelingt mit

CREATE DATABASE ipsum_iq;

Die Datenbank muss ipsum_iq heißen, da sonst der Server nicht auf sie zugreifen kann.



```
pi@IpsumServer:~/IpsumIQ/ipsum-iq/frontend $ sudo mysql -u root
Welcome to the MariaDB monitor.  Commands end with ; or \g.
Your MariaDB connection id is 40
Server version: 10.5.15-MariaDB-0+deb11u1 Raspbian 11

Copyright (c) 2000, 2018, Oracle, MariaDB Corporation Ab and others.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

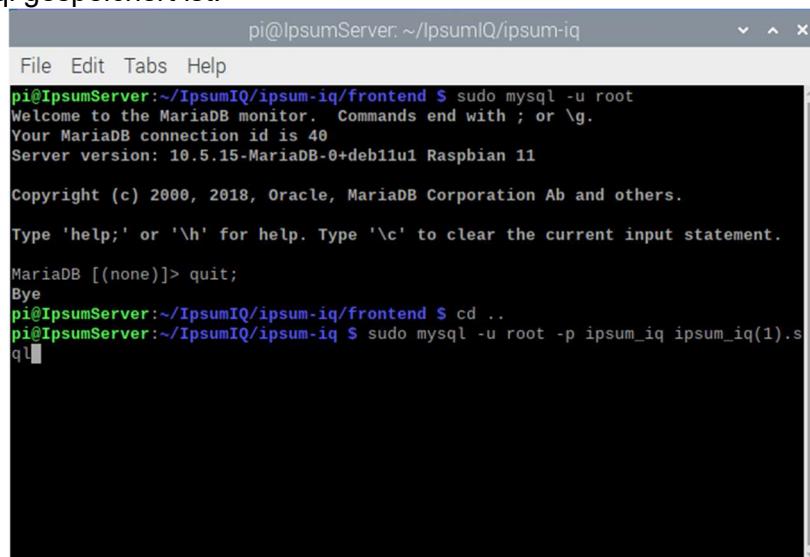
MariaDB [(none)]> create database ipsum_iq;
```

Abbildung 40 create database / eigene Aufnahme

quit; schließt die Shell. Die Datenbank wird mit diesem Befehl importiert:

sudo mysql -u root -p ipsum_iq ipsum_iq(1).sql

Dieser Befehl muss in dem Verzeichnis ausgeführt werden, wo auch die Datei ipsum_iq(1).sql gespeichert ist.



```
pi@IpsumServer:~/IpsumIQ/ipsum-iq/frontend $ sudo mysql -u root
Welcome to the MariaDB monitor.  Commands end with ; or \g.
Your MariaDB connection id is 40
Server version: 10.5.15-MariaDB-0+deb11u1 Raspbian 11

Copyright (c) 2000, 2018, Oracle, MariaDB Corporation Ab and others.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

MariaDB [(none)]> quit;
Bye
pi@IpsumServer:~/IpsumIQ/ipsum-iq/frontend $ cd ..
pi@IpsumServer:~/IpsumIQ/ipsum-iq $ sudo mysql -u root -p ipsum_iq ipsum_iq(1).sql
```

Abbildung 41 Importieren der SQL Datei / eigene Aufnahme

8.6.1 Zugriff auf Datenbank ermöglichen (Remote Client Access):

In MariaDB kann aus Sicherheitsgründen mit dem User Root nur über einen Local Socket auf die Datenbank zugegriffen werden. Da die Webanwendung aber mit dem TCP/IP Protokoll auf die Datenbank zugreift, müssen wir dies zuerst konfigurieren. Kurzgesagt: Es wird ein neuer User erstellt und die Zugriffsberechtigungen für MariaDB erteilt. (Configuring MariaDB for Remote Client Access - MariaDB Knowledge Base, 2023):

Dazu muss die mysql Shell geöffnet werden:

```
mysql -u root -p
```

Danach wird der neue User, der user heißen soll erstellt:

```
CREATE USER user;
```

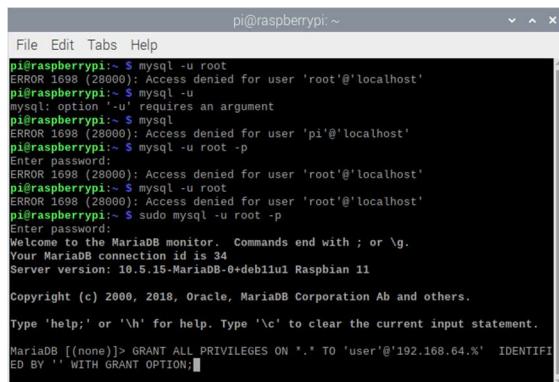
Jetzt werden die nötigen Berechtigungen erteilt:

```
GRANT ALL PRIVILEGES ON *.* TO 'user'@'192.168.64.%' IDENTIFIED BY '' WITH  
GRANT OPTION;
```

Dieser Befehl gibt dem Benutzer user Administratorrechte, die er auch benötigt, da die Webapp sonst nichts in der Datenbank verändern kann.

Anschließend müssen die Änderungen übernommen werden:

```
FLUSH PRIVILEGES;
```



```
pi@raspberrypi:~ $ mysql -u root  
ERROR 1698 (28000): Access denied for user 'root'@'localhost'  
pi@raspberrypi:~ $ mysql -u  
mysql: option '--u' requires an argument  
pi@raspberrypi:~ $ mysql  
ERROR 1698 (28000): Access denied for user 'pi'@'localhost'  
pi@raspberrypi:~ $ mysql -u root  
ERROR 1698 (28000): Access denied for user 'root'@'localhost'  
pi@raspberrypi:~ $ sudo mysql -u root -p  
Enter password:  
ERROR 1698 (28000): Access denied for user 'root'@'localhost'  
pi@raspberrypi:~ $ mysql -u root  
ERROR 1698 (28000): Access denied for user 'root'@'localhost'  
pi@raspberrypi:~ $ sudo mysql -u root -p  
Enter password:  
Welcome to the MariaDB monitor. Commands end with ; or \g.  
Your MariaDB connection id is 34  
Server version: 10.5.15-MariaDB-0+deb11u1 Raspbian 11  
  
Copyright (c) 2000, 2018, Oracle, MariaDB Corporation Ab and others.  
Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.  
  
MariaDB [(none)]> GRANT ALL PRIVILEGES ON *.* TO 'user'@'192.168.64.%' IDENTIFI  
ED BY '' WITH GRANT OPTION;
```

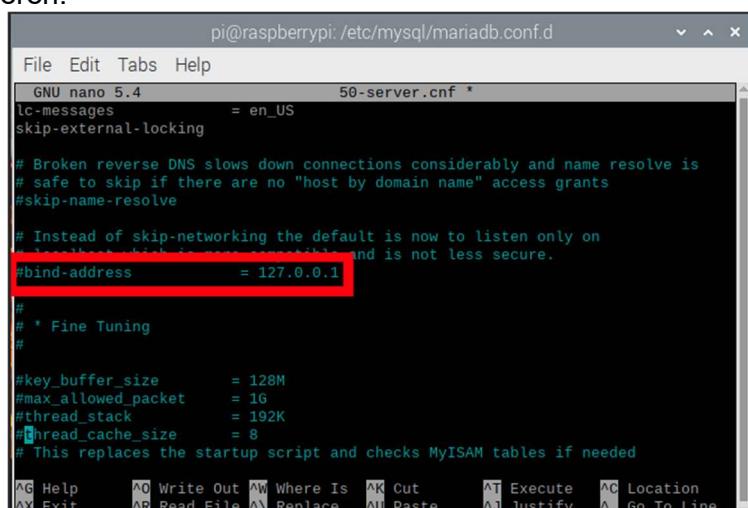
Abbildung 42 MySQL Berechtigungen erteilen / eigene Aufnahme

Jetzt muss nur noch die conf Datei verändert werden, damit der Zugriff über die Webapp ermöglicht wird.

Die Datei 50-server.cnf mit dem Nano-Editor öffnen:

```
sudo nano /etc/mysql/mariadb.conf.d/50-server.cnf
```

Dort nach der Zeile wo bind-address steht suchen und diese mit dem Raute Symbol (#) auskommentieren:



```
pi@raspberrypi:/etc/mysql/mariadb.conf.d  
File Edit Tabs Help  
GNU nano 5.4 50-server.cnf *  
lc-messages = en_US  
skip-external-locking  
  
# Broken reverse DNS slows down connections considerably and name resolve is  
# safe to skip if there are no "host by domain name" access grants  
#skip-name-resolve  
  
# Instead of skip-networking the default is now to listen only on  
# localhost which is safe and is not less secure.  
#bind-address = 127.0.0.1  
  
# * Fine Tuning  
  
#key_buffer_size = 128M  
#max_allowed_packet = 1G  
#thread_stack = 192K  
#thread_cache_size = 8  
# This replaces the startup script and checks MyISAM tables if needed  
  
^G Help ^O Write Out ^W Where Is ^K Cut ^T Execute ^C Location  
^X Exit ^R Read File ^L Replace ^U Paste ^J Justify ^_ Go To Line
```

Abbildung 43 bind-address / eigene Aufnahme

9 Hardwareprogrammierung mit Arduino IDE (Bernhard R)

1.1 Programmierung

Wie in der Beschreibung der Hardware schon erwähnt, ist der ESP8266-01S etwas aufwendiger zu programmieren. Es muss einerseits eine zusätzlichen Boardverwalter-URL(https://arduino.esp8266.com/stable/package_esp8266com_index.json) hinzugefügt werden und andererseits muss ein USB-Schnittstelle verwendet werden, um den ESP mittels der Arduino IDE zu programmieren.

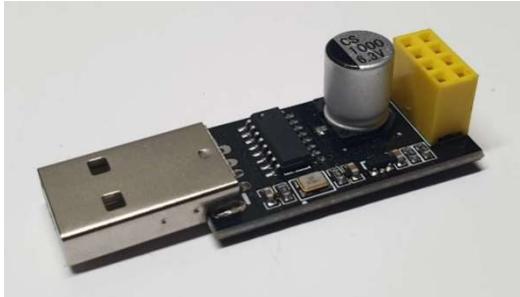


Abbildung 44 EPS-01S USB Programmierer; Selbstaufnahme

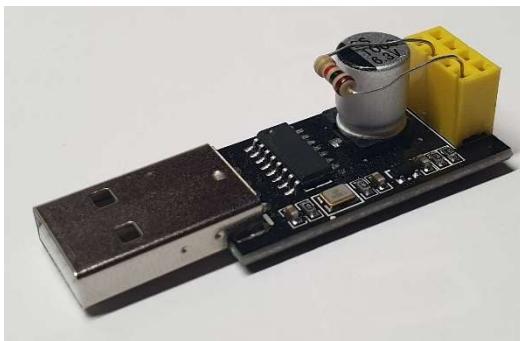


Abbildung 45 EPS-01S USB Programmer mit Widerstand zwischen GPIO0 und GND; Selbstaufnahme

Um den ESP8266-01S in den Programmiermodus zu setzen, muss GPIO0 mit GND verbunden werden. Danach kann der ESP mit der Arduino IDE problemlos programmiert werden.

9.1 Debugging

Die Arduino-IDE verfügt über mehrere Möglichkeiten, um den Code zu Debuggen. In diesem Projekt wird der C++ Präprozessor eingesetzt:

```
#define DEBUG 1
// debug/debugln for debugging
#if DEBUG == 1
#define debug(x) Serial.print(x)
#define debugln(x) Serial.println(x)
#else
#define debug(x)
#define debugln(x)
#endif
```

Abbildung 46 C++ Debugging Makro; Quelle: (Bacon, 2023)

Der C++ Präprozessor ist ein Teil des C++ Compilers, welcher vor den eigentlichen Compiler ausgeführt wird (Preprocessor - cppreference.com, 2023).

Im Debug-Code werden zwei Präprozessor Makros definiert, „debug“ und „debugln“. Diese werden über ein weiteres Makro „DEBUG“ gesteuert. Wenn „DEBUG“ „1“ ist, werden

„debug“ und „debugln“ mit „Serial.print“ und „Serial.println“ ersetzt und leiten den Input zu einer seriellen Schnittstelle weiter.

Ist der Wert von „DEBUG“ nicht „1“, so werden „debug“ und „debugln“ als leere Makros definiert, was bedeutet, dass sie nichts tun und keine Ausgabe generieren, wenn sie aufgerufen werden.

Dies ermöglicht das schnelle Aktivieren und Deaktivieren von Debug-Ausgaben, indem das Makro „DEBUG“ bearbeitet wird, ohne dass große Modifikationen am Quellcode gemacht werden. Aus Gründen der Leserlichkeit werden „debug“-Statements aus den in dieser Dokumentation enthaltenen Programmschnitzel entfernt und die Programmschnitzel werden auch ein wenig abgeändert (Bacon, 2023).

9.2 Konfiguration des ESPs

Die Verwendung von mehreren ESPs, ohne jeden ESP mit individueller Firmware zu programmieren, erfordert die Möglichkeit die W-LAN-Zugangsdaten (Server-IP + Port und der Name des ESPs) beim ersten Start manuell eingegeben zu können.

9.2.1 WiFiManager (tzapu, 2023)

Um dies zu ermöglichen, wird ein WiFiManager benutzt (tzapu, 2023). Diese ist eine Bibliothek für den ESP8266, ESP32, ESP32S2 und ESP32C3, welche es ermöglicht, die Daten über ein Web-Portal einzugeben.

9.2.1.1 Installation des WifiManagers

Die Installation des WifiManager ist ein mehrstufiger Prozess, welcher mit dem Auffinden des GitHub Repository beginnt. Der Link ist <https://github.com/tzapu/WiFiManager>.

Als nächsten Schritt muss der Arduino Sketch Ordner aufgesucht werden. Dieser ist standardmäßig in dem Dokumente-Ordner der Person. In diesem befindet sich der „libraries“-Ordner. In diesem Ordner kann jetzt mittels „git clone“ das Repository heruntergeladen werden.

```
\Documents\Arduino\libraries>git clone https://github.com/tzapu/WiFiManager.git
```

Abbildung 47 git clone befehl | Selbstaufnahme

9.2.1.2 Funktionsweise

```
#include <WiFiManager.h>
//https://github.com/tzapu/WiFiManager WiFi Configuration Magic
WiFiManager wifiManager;
wifiManager.setConfigPortalTimeout(180);
// Connect to Wi-Fi
wifiManager.autoConnect("IPSUM|IQ", "IQ-ESP8233");
```

Abbildung 48 WiFiManager minimal code; Selbstaufnahme

Nachdem die Bibliothek inkludiert wurde, haben wir Zugriff auf die WiFiManager Class. Mit dieser kann in Zeile 3 der WiFiManager initialisiert und mit „`wifiManager.autoConnect("IPSUM|IQ", "IQ-ESP8233");`“ der WiFiManager gestartet werden. Zu Beginn versucht der ESP sich mit bereits gespeicherten W-Lans zu verbinden. Sobald eine Verbindung aufgebaut wurde, wird der Code ausgeführt. Im Falle, dass es nicht funktioniert, wechselt der ESP in den Accesspoint-Modus. Die SSID und das Passwort für den Access Point werden in der „autoConnect“ Funktion übergeben. Nachdem man sich mit dem Access Point verbunden hat, kann auf die Konfigurationsseite zugegriffen und ein neues W-Lan eingegeben werden. Eingegebene W-Lan SSID und Passwort werden vom WiFiManager automatisch gespeichert.

9.2.1.3 Zusätzliche Parameter

Der WiFiManager bietet die Möglichkeit, weitere Eingabefelder auf der Konfigurationsseite anzuzeigen. Jedoch müssen zusätzliche Werte manuell gespeichert werden. Zusätzlich zu den W-Lan-Zugangsdaten muss die Server IP, der Server Port und der „Name“ des ESPs angegeben werden. Die Variablennamen sind „websocket_server“, „websocket_port“ und „websocket_path“. Alle drei Variablen sind globale Zeichenketten mit vordefinierter Größe. Diese brauchen mehr Arbeitsspeicher als dynamische Zeichenketten, erfordern jedoch wenig manuelles Memory-Management und das Speichern dieser erfordert nur sehr simplen Code. Des Weiteren erfordert der WiFiManager die Größe der Zeichenkette während des Erstellens der Weboberfläche.

```
WiFiManager wifiManager;

char websocket_server[40] = "";
char websocket_port[6] = "";
char websocket_path[40] = "/";

bool shouldSaveConfig = false;

void saveConfigCallback() {
    shouldSaveConfig = true;
}

void setup() {
    loadConfigFromEEPROM();

    wifiManager.setConfigPortalTimeout(180);

    // id/name, placeholder/prompt, default, length
    WiFiManagerParameter custom_websocket_server("server",
        "websocket server", websocket_server, 40);
    WiFiManagerParameter custom_websocket_port("port",
        "websocket port", websocket_port, 6);
    WiFiManagerParameter custom_websocket_path("path",
        "Name of the ESP(has to start with '/')", websocket_path, 40);

    wifiManager.addParameter(&custom_websocket_server);
    wifiManager.addParameter(&custom_websocket_port);
    wifiManager.addParameter(&custom_websocket_path);

    wifiManager.setSaveConfigCallback(saveConfigCallback);

    // Connect to Wi-Fi
    wifiManager.autoConnect("IPSUM|IQ", "IQ-ESP8233");

    if (shouldSaveConfig) {
        strcpy(websocket_server, custom_websocket_server.getValue());
        strcpy(websocket_port, custom_websocket_port.getValue());
        strcpy(websocket_path, custom_websocket_path.getValue());
        saveConfigToEEPROM();
    }
}
```

Abbildung 49 WiFiManager Gesamtcode; Selbstaufnahme

9.2.1.4 Das Endergebnis des WiFiManagers:

WiFiManager

IPSUMIQ

Configure WiFi

Info

Exit

Update

**Not connected to Premium
AP not found**

bauern hotspot

htlwlan

SSID

htlwlan

Password

Show Password

websocket server

192.168.64.101

websocket port

8080

Name of the ESP(has to start with '/')

/ESP1

Save

Refresh

**Not connected to Premium
AP not found**

Abbildung 51 Hauptmenü der WiFiManager Konfigurationsseite; Selbstaufnahme

Abbildung 50 WiFiManagers Konfigurationsseite; Selbstaufnahme

9.2.2 Speichern der Daten

Es gibt mehrere Möglichkeiten, um Daten auf dem ESP so zu speichern, dass sie nach einem Neustart des ESPs noch vorhanden sind. Die einfachste Möglichkeit ist, die „[EEPROM.h](#)“ Standardbibliothek zu verwenden.

9.2.2.1 EEPROM

Ein EEPROM (“electrically erasable programmable read-only memory”), auch E²PROM genannt, ist ein nicht-flüchtiger Speicher, welcher, im Gegensatz zu EPROM, elektronisch beschrieben und gelöscht werden kann. Der EEPROM hat eine begrenzte Anzahl von Schreibvorgängen, normalerweise zwischen 10.000 und einer Million und eine begrenzte Zeit der Datenerhaltung von ungefähr 10 Jahre. Viele Arduino Board benutzen sie als Speicher, der vom Code veränderbar ist. Zugriff auf den EEPROM bietet die „[EEPROM.h](#)“ Standardbibliothek. Damit kann der EEPROM Byte für Byte mittels der „`EEPROM.write(index, value)`“ Funktion geändert und der „`EEPROM.read(index)`“ Funktion gelesen werden.

Der ESP8266-01S hat keinen EEPROM, sondern einen FLASH. Dieser ist im Vergleich zum EEPROM schneller, aber auch teurer.

Mit der „[EEPROM.h](#)“ kann ein EEPROM auf dem FLASH emuliert werden, die Daten werden aber erst nach dem „`EEPROM.end()`“ in den FLASH geschrieben.

```

char websocket_server[40] = "";
char websocket_port[6] = "";
char websocket_path[40] = "/";

void loadConfigFromEEPROM() {
    EEPROM.begin(128);
    for (int i = 0; i < 40; i++) {
        byte readByte = EEPROM.read(i + offset);
        websocket_server[i] = readByte;
    }
    for (int i = 0; i < 6; i++) {
        byte readByte = EEPROM.read(i + 50 + offset);
        websocket_port[i] = readByte;
    }
    for (int i = 0; i < 40; i++) {
        byte readByte = EEPROM.read(i + 60 + offset);
        websocket_path[i] = readByte;
    }
    EEPROM.end();
}

void saveConfigToEEPROM() {
    EEPROM.begin(128);
    for (int i = 0; i < 40; i++) {
        byte writeByte = websocket_server[i];
        EEPROM.write(i + offset, writeByte);
    }
    for (int i = 0; i < 6; i++) {
        byte writeByte = websocket_port[i];
        EEPROM.write(i + 50 + offset, writeByte);
    }
    for (int i = 0; i < 40; i++) {
        byte writeByte = websocket_path[i];
        EEPROM.write(i + 60 + offset, writeByte);
    }
    EEPROM.end();
}

```

Abbildung 52 EEPROM Code; Selbstaufnahme

Bevor Daten geschrieben werden, muss der EEPROM mit 128 Bytes initialisiert werden. Da es eine bekannte Anzahl von Variablen gibt und deren Größe im Vorhinein festgelegt ist, können simple For-Loops eingesetzt werden. Die For-Loops gehen Byte für Byte über die Zeichenketten und schreiben diese jeweils ein Byte pro Zyklus in den (virtuellen) EEPROM. Als EEPROM-Index wird der Index in der Zeichenkette plus einen frei wählbaren Offset plus der Längen der zuvorkommenden Zeichenketten verwendet. Zum Schluss wird der EEPROM „beendet“ und die Daten werden auf den FLASH geschrieben.

9.3 Websocket

Um die Möglichkeit zu haben, vom Server den ESP jederzeit zu aktivieren und zu deaktivieren und von ESP den Server jederzeit den derzeitigen Status mitzuteilen, ist eine bidirektionale Verbindung nötig. Websocket wurde verwendet, da dieses Protokoll am ESP ohne Zusatz-Bibliotheken unterstützt wird. Am Server muss trotzdem eine weitere JavaScript Bibliothek installiert werden.

Der Server erwartet einen JSON-String, welcher die Variable „status“ enthält, diese Variable ist eine „1“, wenn die LED ein ist und „0“, wenn sie aus ist. Die Befehle für den ESP sind „on“, um die LED einzuschalten, „off“, um ihn auszuschalten und „toggle“, um den ESP einzuschalten, wenn er aus ist und auszuschalten, wenn er ein ist.

Die Verbindung wird mittels der aus dem EEPROM geladenen Server IP, Port und Pfad erstellt. Um dem Server mitteilen zu können, welchen Namen der ESP haben soll, wird der Pfad verwendet. Zu beachten ist, dass der Port, den die „`webSocket.begin`“ erwartet, ein Integer ist, der aus dem EEPROM gelesene Port ist jedoch eine Zeichenkette, um dieses Problem zu beheben, muss der EEPROM Port mittels der „`atoi(number)`“ Funktion in eine Zahl umgewandelt werden. (<https://cplusplus.com/reference/cstdlib/atoi/>, 2023)

Sobald der Websocket eine Veränderung (Event) feststellt, wird die „`webSocketEvent`“ Funktion ausgeführt, Veränderungen sind:

- Verbindung aufgebaut
- Verbindung getrennt
- Text Nachricht empfangen
- Binäre Nachricht empfangen
- Ping erhalten
- Gesendeten Ping wiedererhalten
- Ein Fehler ist aufgetreten

Ist das Event nicht eine Text Nachricht, so wird diese ignoriert. Bei Textnachrichten wird die Funktion „`handleWebSocketMessage(payload, length)`“ ausgeführt. In dieser Funktion wird überprüft, ob der „payload“ also die Nachricht „on“, „off“ oder „toggle“ enthält und ändert den Status der LED dementsprechend.

```

#include <WebSocketsClient.h>

char websocket_server[40] = "";
char websocket_port[6] = "";
char websocket_path[40] = "/";

WebSocketsClient webSocket;

void handleWebSocketMessage(uint8_t* data, size_t len) {
    data[len] = 0;
    if (strcmp((char*)data, "toggle") == 0) {
        setLED(!ledState);
    }
    else if (strcmp((char*)data, "on") == 0) {
        setLED(true);
    }
    else if (strcmp((char*)data, "off") == 0) {
        setLED(false);
    }
}
void webSocketEvent(WStype_t type, uint8_t* payload, size_t length) {
    switch (type) {
        case WStype_DISCONNECTED:
            break;
        case WStype_CONNECTED:
            break;
        case WStype_TEXT:
            handleWebSocketMessage(payload, length);
            // send message to server
            break;
        case WStype_BIN:
            break;
        case WStype_PING:
            // pong will be send automatically
            break;
        case WStype_PONG:
            // answer to a ping we send
            break;
    }
}
void setup() {
    loadConfigFromEEPROM();
    // server address, port and URL
    webSocket.begin(websocket_server, atoi(websocket_port), websocket_path);
    // event handler
    webSocket.onEvent(webSocketEvent);
    // try ever 5000 again if connection has failed
    webSocket.setReconnectInterval(5000);
}
void loop() {
    webSocket.loop();
}

```

Abbildung 53 Websocket Code; Selbstaufnahme

9.4 Interaktion mit der Hardware

```
bool ledState = false;
const int ledPin = 0;

bool btnState = false;
const int btnPin = 3;

const int resetPin = 2;
unsigned long zeitResetPin = 0;

unsigned long zeit2 = 0;
char jsonResp[] = "{!status!:?}";

WebSocketsClient webSocket;

void setLED(bool newState) {
    ledState = newState;
    jsonResp[10] = 48 + ledState;
    digitalWrite(ledPin, ledState);
    webSocket.sendTXT(jsonResp);
}

void setup(){
    pinMode(btnPin, INPUT_PULLUP);
    pinMode(resetPin, INPUT_PULLUP);
    pinMode(ledPin, OUTPUT);

    jsonResp[1] = 34;
    jsonResp[8] = 34;
    jsonResp[10] = 48 + ledState;
}

void loop() {
    unsigned long zeit1 = millis();
    if (zeit2 <= zeit1) {
        zeit2 = zeit1 + 5000;
        webSocket.sendTXT(jsonResp);
    }
    if (digitalRead(resetPin)) {
        zeitResetPin = zeit1;
    }
    if ((zeit1 - 5000) > zeitResetPin) {
        resetESP();
    }
    if (btnState && btnState != digitalRead(btnPin)) {
        // code here gets executed on raising edge of btn
        setLED(!ledState);
    }
    btnState = digitalRead(btnPin);
    delay(1);
}
```

Abbildung 54 Hardware-Interaktionscode; Selbstaufnahme

9.4.1 GPIO-Pins

Drei GPIO-Pins sind für den ESP nötig: „ledPin“, „btnPin“ und „resetPin“. „ledPin“ ist der Pin, mit welchen die Lampe geschaltet wird, dieser Pin muss als OUTPUT definiert werden. „btnPin“ wird verwendet, um einen Knopf an den ESP anzuschließen, er wird als „INPUT_PULLUP“ definiert. Das definiert den Pin als Input und verbindet diesen über einen Widerstand zu Vcc, das erleichtert die Verkabelung der Knöpfe um ein Vielfaches, im Gegenzug muss im Code daran gedacht werden, dass, wenn der Pin „true“ ausgibt, der Knopf **nicht** gedrückt ist. Zu guter Letzt gibt es noch den „resetPin“, welcher verwendet wird, um den ESP neu zu konfigurieren.

```
const int ledPin = 0;
const int resetPin = 2;
const int btnPin = 3;

void setup(){
    pinMode(btnPin, INPUT_PULLUP);
    pinMode(resetPin, INPUT_PULLUP);
    pinMode(ledPin, OUTPUT);
}
```

Abbildung 55 Pindefinition; Selbstaufnahme

9.4.2 Zeitmessung

Wenn die „delay()“ Funktion zu oft verwendet wird, kann dies zum Ignorieren der Tasterbetätigung führen. Daher wird die Funktion „millis()“ verwendet, diese Funktion gibt die Anzahl der Millisekunden seit dem Starten des ESPs aus, diese wird nach knapp 50 Tagen wieder auf 0 zurückgesetzt werden. Nun kann ein Zeitgeber erstellt werden. Dieser Timer wird verwendet, um den Server alle fünf Sekunden seinen Status mitzuteilen, unabhängig davon, ob er sich verändert hat.

```
unsigned long zeit2 = 0;

void loop() {
    unsigned long zeit1 = millis();
    if (zeit2 <= zeit1) {
        zeit2 = zeit1 + 5000;
        webSocket.sendTXT(jsonResp);
    }
}
```

Abbildung 56 Timer-Code; Selbstaufnahme

9.4.2.1 Reset Timer / Knopf

Während der „resetPin“ nicht gedrückt wird, wird der Variable „zeitResetPin“ immer die derzeitige Zeit zugewiesen. Sobald der Reset-Knopf gedrückt wird, wird „zeitResetPin“ nicht mehr auf die aktuelle Zeit gesetzt und fällt zurück. Wenn die Differenz mehr als 5000 Millisekunden ist, dies kann nur passieren, wenn der Knopf 5 Sekunden lang durchgehend gedrückt worden ist, werden die W-Lan-Zugangsdaten gelöscht und der ESP neu gestartet.

Da kein W-Lan-Netz gespeichert worden ist, startet der WifiManager danach die Konfigurationsseite.

```
const int resetPin = 2;
unsigned long zeit2 = 0;
unsigned long zeitResetPin = 0;

void loop() {
    unsigned long zeit1 = millis();
    if (digitalRead(resetPin)) {
        zeitResetPin = zeit1;
    }
    if ((zeit1 - 5000) > zeitResetPin) {
        resetESP();
    }
}
```

Abbildung 57 Reset Timer; Selbstaufnahme

9.4.3 Erkennung steigender Flanken

Sobald der eigentliche Schalter am ESP gedrückt worden ist, sollte die Lampe ihren Zustand ändern. Dies kann mit einen „rising-edge-detector“ erreicht werden.

```
if (btnState && btnState != digitalRead(btnPin)) {
    // code here gets executed on raising edge of btn
    setLED(!ledState);
}
btnState = digitalRead(btnPin);
delay(10);
```

Abbildung 58 rising edge detector; Selbstaufnahme

„btnState“ ist der Status des Knopfes vom vorherigen Durchgang des Programmes. Sobald der „alte“ „btnState“ nicht mehr mit dem „neuen“ „btnState“ übereinstimmt, also der Knopf von „LOW“ auf „HIGH“ geht oder von „HIGH“ auf „LOW“ geht und der „alte“ „btnState“ „HIGH“ war, „btnState“ ist das entgegengesetzte von dem eigentlichen Status des Knopfes, ändert die Lampe ihren Status.

9.4.4 Ein und Ausschaltung der Lampe

Um den Status der Lampe von überall zu setzen, wird die Funktion „setLED(newState);“ erstellt:

```
bool ledState = false;
char jsonResp[] = "{!status!:?}";

void setLED(bool newState) {
    ledState = newState;
    jsonResp[10] = 48 + ledState;
    digitalWrite(ledPin, ledState);
    webSocket.sendTXT(jsonResp);
}

void setup() {
    jsonResp[1] = 34;
    jsonResp[8] = 34;
    jsonResp[10] = 48 + ledState;
}
```

Abbildung 59 setLed Funktion; Selbstaufnahme

Zwei Hauptvariablen sind dafür notwendig. „ledState“, sie speichert den Status der Lampe und „jsonResp“, das ist die Zeichenkette, die zum Server gesendet wird. Um den Status in die „jsonResp“ einzufügen, wird das 11 Zeichen direkt angesprochen. Um eine „0“ oder eine „1“ einzufügen, muss der zugehörige ASCII-Code benutzt werden. „0“ hat einen ASCII-Code von 48 und „1“ hat einen ASCII-Code von 49. „48 + ledState“ erreicht genau das.

Danach wird der neue „ledState“ in den Output gesetzt und „jsonResp“ wird mittels „webSocket.sendTXT“ an den Server gesendet.

Es ist nennenswert, dass JSON nur „“ akzeptiert und C++ Zeichenkettendefinitionen nur mit „“ erlaubt. Um diese Limitationen zu umgehen, kann einerseits mit „\“ die für JSON nötigen Anführungszeichen umgangen werden oder sie können im Nachhinein mit dem ASCII-Code, der 34 ist, eingefügt werden.

10 Frontend (Kilian Grassauer)

Das Frontend ist die Präsentationsebene, das ist der Teil einer Applikation, den der Benutzer sehen kann, zum Beispiel in Form einer grafischen Benutzeroberfläche (graphical user interface, kurz GUI). Das Frontend setzt sich aus zwei Teilen zusammen Web-Design und Frontend Development. Das Web-Design beschäftigt sich mit dem Erstellen und Bearbeiten von Grafiken für Webseiten. Frontend Development ist die Umsetzung der Gestaltungsebene der Website, dafür wird meist HTML, CSS und JavaScript verwendet. Das Frontend bietet den Benutzer eine Schnittstelle zum backend und verbessert User-Experience. (Was ist Frontend — was Backend?, 2023)

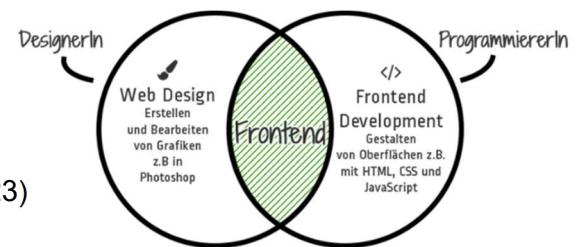


Abbildung 60 Grafik Frontend (Was ist Frontend — was Backend?, 2023)

10.1 Präsentationsebene

10.1.1 Corporate Design

Das Corporate Design beschäftigt sich mit dem Erscheinungsbild des Unternehmens, nach außen und innen. Die wichtigsten Bestandteile sind Aspekte wie Logos, Firmenfarben, Schriftarten, Geschäftsdruckkarten, Werbemittel, Online-Präsenzen, Point of Sales Gestaltung und Berufskleidung. Dies sind sogenannte Touch Points (Berührungs punkt). (Corporate Design – Wie gut präsentiert sich Ihr Unternehmen?, 2023)



Abbildung 61 Grafik Corporate Design (Corporate Design – Definition, Beispiele und Tipps, 2023)

10.1.1.1 Corporate Design bei Ipsum IQ

10.1.1.1.1 Logo

Bei dem Logo handelt es sich um einen einfachen Firmenschriftzug (Wortmarke), der für einen besseren Kontrast auf allen Hintergründen in den Farben Schwarz oder Weiß gehalten wurde. Das Logo in der Farbe Schwarz ist das primäre Logo und kann auf der Website ebenso wie auf dem Gerät selbst gefunden werden. Die Farbe Schwarz steht für Eleganz Seriosität sowie Glaubwürdigkeit und wurde aus diesen Gründen gewählt. Das weiße Logo ist nur auf den Raspberry Pi Server zu finden. Als Schriftart wurde „Frutiger Bold“ verwendet, da sie über Charakteristiken wie Funktionalität, Lebendigkeit und Präzision verfügt. Zudem ist Frutiger eine gut Lesbare Schrift mit klaren Linien und eignet sich deshalb für alle Medien. Das Logo wurde mithilfe von Adobe Photoshop erstellt und hat einen sogenannten Schlagschatten, der dem Text noch mehr Tiefe verleiht. (Bedeutung der Farben ["Farbsymbolik und Farben in der Psychologie"], 2023) (Farblehre - Bedeutung der Farbe Schwarz | Polster Fischer, 2023) (Schriftportrait Frutiger - Frutiger kaufen, 2023)



Abbildung 62 Logos - Ipsum | IQ

10.1.1.1.2 Farben

Als Primärfarben wurde ein bläuliches Lila (#9333EA) und ein helles Blau (#3B82F6) verwendet. Blau wird oft mit Technik, Innovation und Zuverlässigkeit in Verbindung gebracht, während Lila für Luxus und Reichtum steht. Die Sekundärfarben auf der Website sind Schwarz und Weiß, diese Farben bilden zusammen ein schlichtes und elegantes Design. (Bedeutung der Farben ["Farbsymbolik und Farben in der Psychologie"], 2023)



Abbildung 63 Farben - Ipsum / IQ

10.1.1.1.3 Schriftarten

Ipsum IQ verfügt nur beim Firmenschriftzug über eine spezifische Typografie, um es Benutzern zu ermöglichen die Schriftarten auf der Website via den Browser zu ändern

10.1.1.1.4 Grafiken

10.1.1.1.4.1 Favicon

Das Favicon ist ein kleines 16 x 16 oder 32 x 32 Pixel großes Symbol, das der Wiedererkennbarkeit einer Website dient. Meist handelt es sich um eine vereinfachte Version des Firmenschriftzuges oder einer Bildmarke, welche neben dem Seitentitel im Browser angezeigt werden. Das Favicon bietet Benutzern, die mehrere Tabs gleichzeitig nutzen oder Tabs anheften eine wichtige Orientierungshilfe. (Favicon - Was ist das - Seobility Wiki, 2023)

10.1.1.1.4.1.1 Ipsum IQ Favicon

Das Favicon basiert auf dem Ipsum IQ Logo, wobei das Wort Ipsum und die Trennlinie entfernt wurden. Zudem wurde die Text Farbe auf Weiß geändert und der Hintergrund Schwarz eingefärbt. Anschließend wurde das Bild in eine .ico Datei umgewandelt. Zur Bearbeitung wurde wieder Adobe Photoshop verwendet.



Abbildung 64 Favicon - Ipsum / IQ

10.1.1.1.4.2 SVG

Scalable Vector Graphics (SVG) ist ein Web-freundliches Vektorformat, dabei handelt es sich nicht um pixelbasierte Rastergrafiken wie bei PNG und JPEG, sondern um mathematisch definierte Bilder, die sich aus Linien und Punkten in einem Raster zusammensetzen. Aus diesem Grund können Vektorgrafiken ohne Qualitätsverlust skaliert werden, deswegen eignen sie sich ideal komplexe Grafiken und Logos.

Außerdem sind SVGs in XML-CODE geschrieben, was es ermöglicht die Grafiken mit HTML und CSS zu manipulieren und es erlaubt Suchmaschinen wie Google Schlüsselwörter aus den SVGs auszulesen. (SVG-Dateien | Adobe, 2023)

10.1.1.1.4.3 Sonstige Grafiken

Als Icons, die zur Veranschaulichung von Buttons dienen, wurden die von Tailwind bereit gestellten SVGs auf der Website [Heroicons](#) verwendet.



Abbildung 65 Icons
(Heroicons, 2022)

10.1.1.1.4.4 Glühbirne

Die Bilder für Ipsum IQ wurden mittels Adobe Illustrator als Vektorgrafiken bearbeitet und erstellt, danach wurden diese im SVG Format exportiert.



Abbildung 66
Glühbirne aus



Abbildung 67
Glühbirne an

10.2 Frontend-Development

10.2.1 Tech Stack

Eine Auflistung aller Technologiedienste, die zur Erstellung und Ausführung einer Anwendung benötigt werden, wie zum Beispiel Programmiersprachen, Bibliotheken und Frameworks, wird als Tech Stack oder auch Technologie-Stapel bezeichnet. (Tech Stack | Überblick der Top-Technologien | 2022, 2023)

10.2.1.1 Tech Stack bei Ipsum IQ

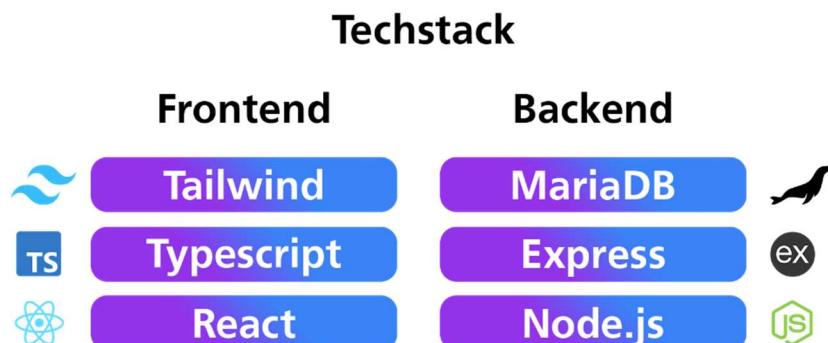


Abbildung 68 Techstack - Ipsum | IQ

10.2.2 HTML

Hypertext Markup Language oder kurz HTML ist eine einheitliche textbasierte Auszeichnungssprache. HTML besteht aus Auszeichnungs- und inhaltsleeren Elementen, diese Elemente werden auch Tags genannt. HTML ermöglicht Webbrowsern wie Safari, Chrome und Firefox das grafische Darstellen und Verknüpfen von Webseiten. Zurzeit werden die Versionen HTML5 und XHTML am häufigsten verwendet. (Html einfach und verständlich erklärt - SEO-Küche, 2023)

10.2.3 CSS

Cascading Style Sheets (CSS) gehört zusammen HTML zu den Kernsprachen des Internets, dabei handelt es sich um eine Gestaltungs- und Formatierungssprache, die in der Lage ist, das Aussehen von HTML-Inhalten zu bestimmen. Es können zum Beispiel Farben, Höhen, Breiten und Abstände von verschiedenen Inhalten manipuliert werden (CSS - Ausführliche Erklärung aus dem Hosting-Lexikon, 2023)

10.2.4 CSS-Frameworks

CSS-Frameworks ermöglichen eine effizientere und schnellere Entwicklung von Webprojekten, da sie bereits fertige Stylesheets zur Verfügung stellen. Einige Frameworks enthalten integrierte Komponenten-Klassen wie Slider, Cards und Navigationskomponenten. Zudem können CSS-Frameworks in zwei Klassen unterteilt werden, component- und utility-based. Komponentenbasierte Frameworks wie Bootstrap legen ihren Fokus auf bereits vorgefertigte Elemente, während utility-based Frameworks wie Tailwind sich auf Flexibilität fokussiert. (CSS Frameworks - mfg, 2023)

10.2.5 Tailwind

Tailwind ist ein Utility-First-CSS-Framework, das Benutzern Utility Klassen zu Verfügung stellt, mit denen schnell und einfach einzigartige Designs erstellt werden können. Das Framework stellt unter anderem Grundlagen wie Farben, Positionierungen und Ränder bereit. Komponenten lassen sich inline stylen, weshalb es nicht nötig ist eine separate CSS-Datei zu erstellen. Tailwind wurde aufgrund seiner Beliebtheit, guten Dokumentation und zeitsparenden sowie hilfreichen Eigenschaften ausgewählt



Abbildung 69 Tailwind Logo (Tailwind CSS, 2022)

10.2.5.1 Tailwind-Installation

Zuerst muss Tailwind via npm installieren werden, danach muss die Tailwind.config.js erstellt werden.

A screenshot of a dark-themed terminal window. The title bar says "Terminal". In the main area, there are two command-line entries: "npm install -D tailwindcss" and "npx tailwindcss init".

Abbildung 70 Befehl zur Installation von Tailwind (Installation - Tailwind CSS, 2023)

Füge die Pfade der Templates in die Tailwind.config.js Datei ein.

A screenshot of a code editor with a dark theme. The file is named "tailwind.config.js". The code contains a single export object with properties for "content", "theme", and "plugins".

```
tailwind.config.js
/** @type {import('tailwindcss').Config} */
module.exports = {
  content: ["./src/**/*.{html,js}"],
  theme: {
    extend: {},
  },
  plugins: [],
}
```

Abbildung 71 Tailwind.config.js (Installation - Tailwind CSS, 2023)

Nun müssen die @Tailwind Richtlinien für jede Tailwind-Ebene zur Main CSS hinzugefügt werden.

A screenshot of a code editor with a dark theme. The file is named "src/input.css". It contains three lines of CSS comments starting with "@tailwind":

```
src/input.css
@tailwind base;
@tailwind components;
@tailwind utilities;
```

Abbildung 72 Einfügen Tailwind Richtlinien (Installation - Tailwind CSS, 2023)

Starte den Tailwind CLI build Prozess

A screenshot of a dark-themed terminal window. The title bar says "Terminal". In the main area, there is one command-line entry: "npx tailwindcss -i ./src/input.css -o ./dist/output.css --watch".

Abbildung 73 Befehl zum Starten des CLI-build-Prozess (Installation - Tailwind CSS, 2023)

(Installation - Tailwind CSS, 2023)

10.2.6 React.js

React wird zur Erstellung von Benutzeroberflächen verwendet und ist eine JavaScript-Bibliothek. Die einzelnen Teile der Benutzeroberfläche, jeder React-Webanwendung, bestehen aus wiederverwendbaren Komponenten. Dies ermöglicht eine einfachere und schnellere Entwicklung von Projekten, da separate Komponenten für Navigationsleisten, Fußzeilen und andere wiederkehrende Elemente erstellt werden können, diese müssen lediglich an der benötigten Stelle importiert werden. Zudem ist React eine Single-Page Anwendung, anstatt eine Anfrage an den Server zu senden, jedes Mal, wenn eine Seite neu gerendert werden sollte, wird der Inhalt der Seite direkt von den React-Komponenten geladen, was zu schnelleren Rendering ohne Neuladen der Website führt. Normalerweise wird für die Erstellung von React-Anwendungen die JSX (JavaScript XML) Syntax verwendet. JSX kombiniert die Logik von Js und die einer Benutzeroberfläche, dadurch muss nicht mehr mit dem DOM über Methoden wie document.getElementById oder querySelector interagiert werden.

React wird bei Ipsum IQ verwendet da es eine der beliebtesten Js-Bibliotheken ist, über eine gute Dokumentation verfügt, eine aktive Community sowie viele Erweiterungen besitzt und sehr performant ist. (Was ist React.js? Ein Blick auf die beliebte JavaScript-Bibliothek, 2023)

10.2.6.1 React Hooks

Hooks erlauben es Mechanismen in Function Components zu verwenden, was zuvor nur in Klassen-Komponenten möglich war. Dank Hooks können in Function Components auch Features wie setState oder Lifecycle Methoden benutzt werden. Hooks sind primär nichts anderes als spezielle Funktionen, die sich an ein festes Schema halten. React verfügt über interne Hooks wie zum Beispiel useState, useEffect und useContext. Es ist auch möglich sogenannte Custom Hooks zu erstellen in denen eine eigene Logik gebündelt werden kann, solange der Name mit use beginnt. (Einführung in Hooks - React - lernen und verstehen, 2020) (React Hooks – Eine Einführung - Informatik Aktuell, 2023)

10.2.6.1.1 useState

Durch useState() können komponenteninterne States verwaltete werden. Es werden für die unterschiedlichen Variablen normalerweise auch mehrere useState()-Aufrufe benutzt

10.2.6.1.2 useEffect

Der useEffect Hook dient der Verwirklichung von Seiteneffekten. In React sind zum Beispiel Konsolen-Ausgaben, Netzwerk-Requests und Event-Handler Seiteneffekte, also alles das, das nicht unmittelbar mit dem Rendern von UI Komponenten zusammenhängt.

10.2.6.2 Installation von React und Erstellung eines Projektes

Zuerst muss React mit diesem Befehl installiert werden

npm install -g create-react-app

Nachdem create-react-app installiert wurde, kann im gewünschten Verzeichnis ein Projekt mittels dieses Befehles erstellt werden

create-react-app project-name

Nun kann React-Applikation gestartet werden, indem man mit cd in den Projektordner wechselt und den npm start Befehl ausführt

cd project-name

npm start

(ReactJS installation and setup | ReactJS tutorials for beginners, 2023)

10.2.7 Login

Der Hintergrund des Logins ist eine Farbverlauf zwischen den zwei Hauptfarben von Lorem Limited. *Gradient* definiert die Art des Hintergrundes (Verlauf), *to-br* (bottom-right) schreibt die Fließrichtung des *gradients* vor, wobei der geschriebene Wert immer die Zielrichtung ist. Die Reihenfolge der Farben legt fest welche Farbe sich wo befindet (erster links dann rechts). Dieser Hintergrund wird für alle Unterseiten verwendet.

```
<div className="bg-gradient-to-br from-purple-600 to-blue-500">
```

Abbildung 74 Hintergrund des Logins

Der Gesamte Login Bereich befindet sich in einem sogenannten Grid Container, der über die Eigenschaft *place-items-center* verfügt. Diese Klasse zentriert alles, was sich darin befindet vertikal und horizontal.

```
<div className="grid place-items-center h-screen">
```

Abbildung 75 Flexbox zu zentrierung des Logins

Dem Login Fenster wurde mit *rounded* ein *Border-Radius* von *0.25 rem* zugewiesen, *w-80/h-80* sind für die Breite und Höhe des Elementes verantwortlich (*20rem*).

Die Klasse *scale* verkleinert das Logo auf eine passende Größe und *pb* fügt ein Padding nach unten hinzu.

```
<div className="Login w-80 h-80 shadow rounded bg-white">
| 
```

Abbildung 76 Login Fenster

Das obere Div-Element sorgt mit den Klassen *flex* und *justify-center* für eine zentrierte Position. Das Eingabefeld wird so gestylt, dass es während des fokussierten Zustands keinen speziellen Style wie zum Beispiel einen *Focusing* aufweist.

```
<div className="pt-2 flex justify-center">
  <label className="border-b border-black">
    <input
      className="focus:ring-0 appearance-none bg-transparent border-none
      w-full text-black mr-3 py-1 px-2 leading-tight focus:outline-none"
      type="text"
      onChange={(e)=>
        {
          setUsername(e.target.value);
        }
      }
      placeholder="Username"
      required
    />
  </label>
</div>
```

Abbildung 77 Username Inputfeld

Hier wird ein `useState` erstellt der zur Speicherung des Status des Passwort Inputfelds dient, in der Funktion `pasvis` wird dieser State getoggled. Die Funktion wird durch ein `onClick` Event bei jeweils einem von zwei Buttons aufgerufen. Das Umschalten des State ändert den Typ des Passworteingabefeldes auf den Texttyp, der unzensiert angezeigt wird. Bei den Buttons wird gewechselt welcher sichtbar ist und welcher versteckt ist

```
const [isvis, setvis] = useState<boolean | undefined>(false);
const pasvis = () => {
  setvis(!isvis);
};
```

```
<input
  className="focus:ring-0 appearance-none bg-transparent border border-gray-300 rounded-md py-1 px-2 placeholder-gray-400"
  type={isvis? "text" : "password"}/>
```

Abbildung 78 Passwort Inputfeld

Abbildung 79 `useState isvis/setvis` und die Funktion `pasvis`

```
<button className={isvis ? "hidden absolute -mt-6 ml-1.5": "absolute -mt-6 ml-1.5"} onClick={pasvis}>
```



Abbildung 80 Passwort Inputfeld Vergleich

Mit der Funktion, die sich in `onKeyDown` befindet wird überprüft ob der Benutzer im Inputfeld die Enter-Taste drückt, falls dies der Fall ist wird die `PostLogin()` Funktion ausgeführt.

```
<input
  className="focus:ring-0 appearance-none bg-transparent border border-gray-300 rounded-md py-1 px-2 placeholder-gray-400"
  type={isvis? "text" : "password"}
  onChange={(e)=>
    {
      setPassword(e.target.value);
    }
  }
  onKeyDown={(e)=>{if (e.key === "Enter") {
    PostLogin();
  }
}}
  placeholder="Passwort"
  required
>
```

Abbildung 81 Passwort Inputfeld und Funktionen

Mithilfe von `group-hover:from-purple-600 group-hover:to-blue-500` wird der Button mit einem Farbverlauf gefüllt, sobald der Benutzer den Mauszeiger darüber bewegt

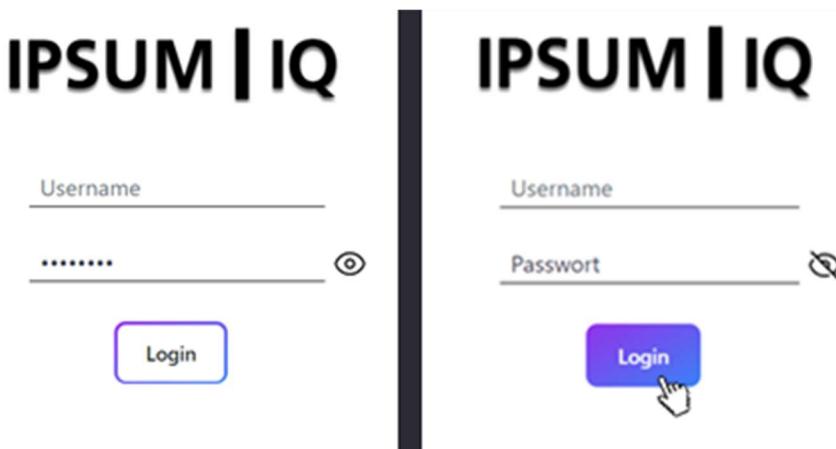


Abbildung 82 Login Button Hover Effekt

10.2.8 Mainpage Komponente

Diese *useState* beinhaltet alle ESPS und deren Daten

```
const [lightstate, lighttoggle] = useState<{ name: string, on: boolean, time: number, futureTime: number }[]>([]);
```

Abbildung 83 ESP useState

Die *map* Funktion erlaubt, durch ein Array zu iterieren und auf die sich darin befindenden Objekte zuzugreifen. Diese Objekte und Daten können manipuliert werden, ohne sie im originalen Array zu verändern. Mit *lightstate.map* wird für jedes Objekt in *lightstate* die Komponente ESP geladen. Mit *light* kann auf Objekte in *lightstate* zugegriffen werden.

```
<div className="grid mt-2 grid-cols-1 sm:grid-cols-2 sm:gap-x-3 gap-y-3 grid-flow-row-dense">
  {lightstate.map((light)=>
    <ESP light={light} />
  )}
</div>
```

Abbildung 84 map Funktion

10.2.9 ESP Komponente

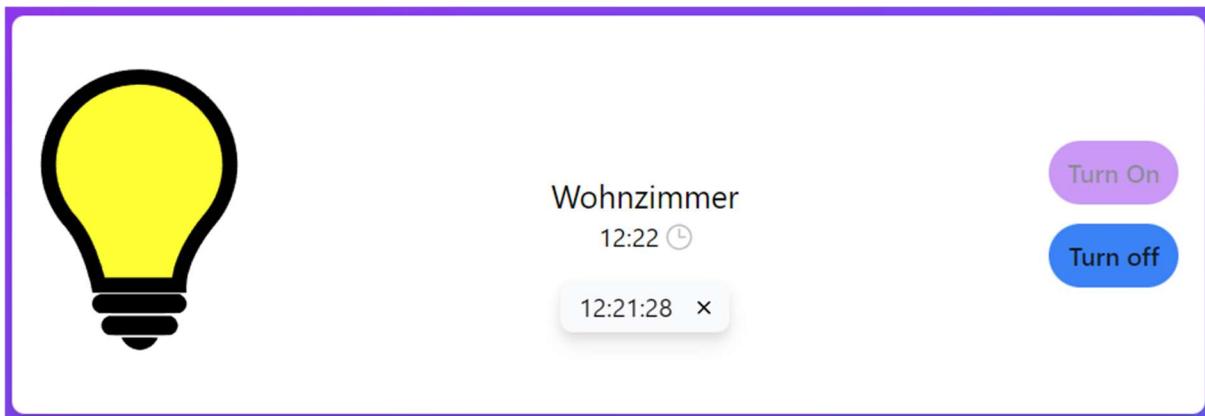


Abbildung 85 ESP Komponente im Frontend

Die Basis der ESP-Komponente bildet ein weißer abgerundeter Hintergrund, indem sich alle weiteren Elemente befinden. Mithilfe von *Flex-Boxen* wurden die inneren Elemente zentriert. Die „Turn on/off“ Buttons wurden in den Beiden Firmenfarben gehalten und verfügen auch über einen Fokusring in diesen Farben. Die Glühbirne auf der linken Seite dient zur Visualisierung des Lichtstatus, zusätzlich kann sie auch als Toggle-Button verwendet werden. Der Anzeige Name (in diesem Fall Wohnzimmer) am ESP selbst definiert und wird auch dementsprechend angezeigt. Darunter sind zwei Inputfelder zusehen, bei denen jeweils höchstens zwei Zahlen eingeben werden können. Nachdem gültige Werte eingegeben, worden sind, die Lampe eingeschalten ist und noch kein Timer läuft kann der Button (Uhr-Symbol) der sich daneben befindet gedrückt werden. Dadurch wird der Timer gestartet und angezeigt, während der Laufzeit kann der X Button gedrückt werden, um den Vorgang zu beenden.

10.2.9.1 Glühbirnen-Button

Der Button wird zum toggeln des Lichtstatus verwendet und ändert dementsprechend das Aussehen zu einer eingeschalteten oder ausgeschalteten Glühbirne. Wenn die Glühbirne vom eingeschalteten Zustand in den eingeschalteten Zustand wechselt, wird die Funktion *ClearTimer* ausgeführt.

```
<button type="button" onClick={()=>{SetLightStatus(light.name); InsertIntoDB(light.name);  
    if(light.on === true){  
        ClearTimer(light.name)  
    }}}>  
<img  
    src={light.on ? "bulb_on.svg" : "bulb_off.svg"}  
    id="bulbbnt"  
    alt={light.on ? "bulb_on" : "bulb_off"}  
    className="sm:h-44"  
/>  
</button>
```

Abbildung 86 Code des Glühbirnen Button

10.2.9.2 Inputfelder

Der Array *allowedNum* beinhaltet die Key-Codes der Zahlen, die in unseren Inputfeldern erlaubt sind (Pfeiltaste links/rechts, Zahlen 0-9, Zahlen 0-9 auf dem Numpad, Backspace, Entfernen und Tab).

```
const allowedNum = [48,49,50,51,52,53,54,54,56,57,96,97,98,99,100,101,102,103,104,105,37,39,8,46,9 ]
```

Abbildung 87 Konstante allowedNum

Mittels einer if-Abfrage wird dann im Inputfeld überprüft ob sich der Key-Code der gedrückten Taste im Array befindet, falls das nicht der Fall ist wird mittels *preventDefault()* [Die *preventDefault()* bricht Events ab die abgebrochen werden können] die Eingabe des Wertes nicht durchgeführt.

```
<input type="number" placeholder="00" max="23" min="0" id="num" name={light.name} maxLength=[2]  
onKeyDown={(evt) => {if(allowedNum.includes(evt.keyCode)){  
} else {  
    evt.preventDefault()  
}}}
```

Abbildung 88 if-Abfrage zur Überprüfung des User-Inputs

Hier wird überprüft ob der Benutzer die Enter-Taste innerhalb des Inputfields benutzt, beim Drücken der Taste wird zuerst kontrolliert ob der Timer überhaupt gestartet werden darf, dannach wird die Funktion *setTime* ausgeführt sofern dies erlaubt ist. Diese Abfrage befindet sich bei beiden Inputfeldern in *onKeyDown*.

```
if(!TimeSubmittable === false && RunningTime() === true){  
    if (evt.key === "Enter") {  
        setTime(light.name, light.on)  
    }  
}
```

Abbildung 89 if-Abfragen um den Timer mit Enter zu starten

Bei jeder Änderung wird der sich im Inputfeld befindende Wert (`event.target.value`) ersetzt durch eine auf zwei Zeichen gekürzte Version des Wertes. Dies wird mit der `slice()` [`slice()`] gibt eine gekürzte Version eines Arrays zurück. In der Klammer werden die Indexe der Elemente aus dem Array als Start- und Endwert angeben, der Endwert wird nicht zurückgegeben] Methode verwirklicht. Die if-Abfrage im ersten Inputfeld überprüft ob, die Länge des sich darin befindenden Wertes größer gleich zwei ist, falls diese Bedingung erfüllt, wird, wird automatisch in das übernächste Element gewechselt (zweites Inputfeld).

```
onChange={(event: any) => {(event.target.value = event.target.value.slice(0, 2));sethours(event.target.value);
  if (event.target.value.length >= 2){
    (event.target.nextElementSibling?.nextElementSibling as HTMLElement)?.focus()
  }
  className="focus:ring-0 appearance-none border-none focus:outline-none p-0 NumericEntry w-5 text-right" required></input>
<span>:</span>
```

Abbildung 90 `slice()` Methode und if-Abfrage Inputfeld-1 ESP

Bei dem zweiten Inputfeld wird das erste Inputfeld ausgewählt, wenn der Wert darin gelöscht wird.

```
onChange={(event:any) => { (event.target.value = event.target.value.slice(0, 2));setminutes(event.target.value);
  if ((event.target as HTMLInputElement).value.length === 0){
    ((event.target as HTMLElement).previousElementsSibling as HTMLElement).previousElementsSibling as HTMLElement)?.
    focus()
  }
  className="NumericEntry focus:ring-0 appearance-none border-none focus:outline-none w-5 p-0" required></input>
```

Abbildung 91 `slice()` Methode und if-Abfrage Inputfeld-2 ESP

Die Funktion `TimeSubmittable` beinhaltet zwei If-Abfragen, die Erste überprüft, ob sich das Licht im ausgeschalteten Zustand befindet, falls das der Fall ist, wird `false` zurückgegeben, die zweite Abfrage überprüft ob bereits ein Timer läuft, wenn die Bedingung erfüllt ist, wird `true` zurückgegeben.

```
function TimeSubmittable():boolean{
  if(!light.on){
    return false;
  };
  if(light.futureTime < Date.now()){
    return true
  }
  return false;
};
```

Abbildung 92 Funktion `TimeSubmittable`

Wenn die Umkehrfunktion `TimeSubmittable` True wiedergibt, ist der Timer-Button deaktiviert und der Cursor sowie die Transparenz verändern sich

```
<button onClick={() => { setTime(light.name, light.on) }} className={TimeSubmittable()?"": "cursor-not-allowed opacity-25"} disabled={!TimeSubmittable()}>
<svg xmlns="http://www.w3.org/2000/svg" fill="none" viewBox="0 0 24 24" strokeWidth={1.5} stroke="currentColor" className="w-5 h-5 -mb-1">
<path strokeLinecap="round" strokeLinejoin="round" d="M12 6v6h4.5m4.5 0a9 9 0 11-18 0 9 9 0 0118 0z" />
</svg>
```

Abbildung 93 Code des Timer Button

In der Funktion `RunningTime` wird kontrolliert, ob der Timer aktiv ist und dem Entsprechendem `true` oder `false` zurückgegebenen.

```
function RunningTime():boolean{
  if(light.futureTime < Date.now()){
    return true
  }
  return false;
};
```

Abbildung 94 Funktion `RunningTime`

Das div-Element, das den Button, der den Timer abbricht und den Timer selbst beinhaltet wird durch das *hidden* Attribut versteckt, wenn *RunningTime* True wiedergibt.

```
<div className="self-center mt-4" hidden={RunningTime()}>
  <div className='rounded-lg bg-gray-50 drop-shadow-lg'>
    <div className='py-1 pl-3 pr-2'>
      <span className='mr-3'><Timer futureTime={light.futureTime} /></span>
      <button onClick={() =>{ClearTimer(light.name)}} className="">
        <svg xmlns="http://www.w3.org/2000/svg" viewBox="0 0 20 14.5" fill="currentColor" className="w-4 h-4">
          <path d="M6.28 5.22a.75.75 0 00-1.06 1.06L8.94 101-3.72 3.72a.75.75 0 101.06 1.06L10 11.06l3.72 3.72a.">
        </svg>
      </button>
    </div>
  </div>
</div>
```

Abbildung 95 Timer und Timerbutton Code

Wenn der Status des Lichtes on ist, wird der Turn On Button deaktiviert und der Cursor sowie die Transparenz verändern sich, beim Status off wird der Turn off Button auf die gleiche Art und Weise verändert.



Abbildung 96 Veränderung der On/Off Buttons

10.2.10 Timer Komponenten

10.2.10.1 Timer.tsx

Hier wird die vom Server in Millisekunden erhaltene Zeit mathematisch in Tage, Stunden, Minuten und Sekunden umgerechnet. Da der Timer der auf der Webseite angezeigt werden soll keine Tage enthält müssen diese zu den Stunden hinzuaddiert werden.

```
const now = new Date().getTime();
// futurtime erhalten vom server
const distance = futureTime - now;

console.log("DISTANCE " + distance);

const days = Math.floor(distance / (24 * 60 * 60 * 1000));

const hours = Math.floor(
  (distance % (24 * 60 * 60 * 1000)) / (1000 * 60 * 60)
) +days * 24;

const minutes = Math.floor((distance % (60 * 60 * 1000)) / (1000 * 60));
const seconds = Math.floor((distance % (60 * 1000)) / 1000);
```

Abbildung 97 Timer Rechnung

Mit diesen if-Abfragen wird verhindert das die Werte in den negativen Zahlenbereich hinunterrechnen

```
timerHours = hours;
timerMinutes= minutes;
timerSeconds= seconds;

if(timerHours < 0) {
| timerHours = 0
}
if(timerMinutes < 0) {
| timerMinutes = 0
}
if(timerSeconds < 0) {
| timerSeconds = 0
}
```

Abbildung 98 if-Abfrage um im positiven Zahlenbereich zu bleiben

Dieser *useEffect* rendert die komplette Komponente jede Sekunde neu und bewirkt damit das die Zeit neu berechnet wird (Eine Sekunde runterzählen).

```
useEffect(() => {
| //rendert die komponente neu => Timer neu berechnet
| const interval = setInterval(() => setTime(Date.now()), 1000);
| return () => {
| | clearInterval(interval);
| };
}, []);
```

Abbildung 99 useEffect zur Neuberechnung

Hier werden die oben errechneten Ergebnisse in die Clock Komponente weitergegeben.

```
return (
|
| <Clock
| | timerHours={timerHours}
| | timerMinutes={timerMinutes}
| | timerSeconds={timerSeconds}
| |
| >
)
```

Abbildung 100 Weitergabe der Ergebnisse zur Clock Komponente

10.2.10.2 Clock.tsx

Hier werden die Eigenschaften der Komponente clock definiert.

```
interface Clockprops {
| | | timerHours: number,
| | | timerMinutes: number,
| | | timerSeconds: number,
| | |
| | &;
```

Abbildung 101 Eigenschaftsdefinition Clock Komponente

Die Stunden/Minuten/Sekunden (timerHours/timerMinutes/timerSeconds) werden in neue Variablen mit dem Typ string (HourString/MinString/SecString) eingesetzt. Die if-Abfrage überprüft, ob die Werte in timerHours/timerMinutes/timerSeconds kleiner 10 sind, falls diese Bedingung erfüllt ist, wird bei jeder der neuen Variablen vorne eine 0 an den string gehängt. Dieser Schritte haben nichts mit der Funktion des Timer zu tun, sondern dienen nur dazu das Aussehen zu verbessern. (vorher 1:3:22 nachher 01:03:22)

```
const Clock: React.FC<Clockprops> = ({ timerHours, timerMinutes, timerSeconds }) => {
  let HourString:string = "" + timerHours;
  let MinString:string = "" + timerMinutes;
  let SecString:string = "" + timerSeconds;
  if(timerHours < 10){
    HourString = "0" + HourString
  }
  if(timerMinutes < 10){
    MinString = "0" + MinString
  }
  if(timerSeconds < 10){
    SecString = "0" + SecString
  }
}
```

Abbildung 102 Umwandlung der Zeit in Strings

Die Werte der Variablen werden nun in einem HTML span Element ausgegeben.

```
return (
  <Fragment>
    | | | <span>{HourString}>:{MinString}>:{SecString}</span>
  </Fragment>
);
```

Abbildung 103 Clock Ausgabe als HTML Element

10.2.11 About Komponente

Der Inhalt der About Seite wird wieder mittels einer *Flexbox* zentriert, zudem ist eine maximale Breite (11/12) vorgegeben. Durch einen sogenannten *iframe* kann ein PDF, in diesem Fall unser Pflichtenheft, auf einer Website dargestellt werden. Der *iframe* befindet sich in einem Element, das vom Benutzer selbst vergrößert oder verkleinert werden kann.

```
<div className=" mt-3 flex justify-center w-max-11/12 ">  
<div className="resize border-2 overflow-auto sm:w-6/12 w-11/12 bigH">  
  
<iframe title="Pflichtenheft" src="Plichtenheft.pdf" className="w-full h-full"></iframe>  
</div>  
</div>  
| | | <div className="flex-grow">  
| | | </div>  
| | | <ShowUser />
```

Abbildung 104 *iframe* Code



Abbildung 105 Darstellung des *iframes* auf der Website

10.2.12 Log Komponenten

Die Tabelle verfügt über einen hellgrauen Hintergrund und abgerundete Ecken. Bei jedem zweiten Eintrag wird der Hintergrund auf ein dunkleres Grau gewechselt. Unter der Tabelle befindet sich ein Button, der zur Löschung der Einträge dient. Dieser Button verändert das Aussehen, wenn der Benutzer sich mit dem Cursor darüber befindet.

Benutzer	Datum	Zeitpunkt	Licht	Status
admin	04-02-2023	17:32	Wohnzimmer	an
admin	04-02-2023	17:32	Wohnzimmer	aus
admin	04-02-2023	17:32	Wohnzimmer	an
admin	04-02-2023	17:32	Wohnzimmer	aus

 Clear Database

Abbildung 106 Tabelle auf der Webseite

10.2.12.1 Querylist.tsx

Im oberen Teil des Codes wurde die Tabelle zentriert und ihr wurde eine Größe zugewiesen. Danach wurde der Tabellenkopf erstellt und gestyliert. Darunter befindet sich eine *map* Funktion, die für jeden Datensatz in der Datenbank-Tabelle „Eintraege“ einen neuen Eintrag in der Log Tabelle generiert.

```
<div className="flex justify-center">
<table className="md:w-9/12 w-11/12 text-sm text-gray-500 bg-gray-50 rounded-t-md rounded-b-md">
  <thead className="text-xs sm:text-base">
    <tr>
      <th scope="col" className="md:px-6 px-2 py-3 text-center border-r-2 border-gray-400 border-b-2">
        Benutzer
      </th>
      <th scope="col" className="md:px-6 px-2 py-3 text-center border-r-2 border-gray-400 border-b-2">
        Datum
      </th>
      <th scope="col" className="md:px-6 px-2 py-3 text-center border-r-2 border-gray-400 border-b-2">
        Zeitpunkt
      </th>
      <th scope="col" className="md:px-6 px-2 py-3 text-center border-r-2 border-gray-400 border-b-2">
        Licht
      </th>
      <th scope="col" className="md:px-6 px-2 py-3 text-center border-r-2 border-gray-400 border-b-2">
        Status
      </th>
    </tr>
  </thead>

  { //griag daten aus setQuerys(Response.data.result);
    querys[0]?.map((entry: any, index: any) =>
      <>
        <tbody className="odd:bg-gray-300">
          <tr className='>
            <td className='text-center border-r-2 border-gray-400'>{entry.user}</td>
            <td className='text-center border-r-2 border-gray-400'>{entry.Datum}</td>
            <td className='text-center border-r-2 border-gray-400'>{entry.Zeitpunkt}</td>
            <td className='text-center border-r-2 border-gray-400'>{entry.Licht}</td>
            <td className='text-center'>{entry.Status}</td>
          </tr>
        </tbody>
    )
  }
</table>
```

Abbildung 107 Code der Tabelle inkl. map Funktion

10.2.12.2 Entries.tsx

Die Komponente

Durch die Klassen `transition`, `ease-in-out`, `hover:scale-110` und `hover:-translate-y-1` wurde der Clear Database Button animiert. Wenn der Benutzer sich mit dem Mauszeiger über den Button befindet, wird die Animation abgespielt.

```
<Querylist />

<div className='mt-4 flex justify-center'>
  <button onClick={clearDB} className='transition ease-in-out h-34 cursor-pointer py-2 px-4 overflow-hidden text-md font-medium text-gray-900 rounded-xl focus:ring-0 focus:outline-none bg-white hover:scale-110 hover:-translate-y-1'>
    <div className='flex'>
      <svg xmlns='http://www.w3.org/2000/svg' fill='none' viewBox='0 0 24 24' strokeWidth={1.5} stroke='currentColor' className='w-6 h-6 mr-1'>
        <path strokeLinecap='round' strokeLinejoin='round' d='M14.74 9l-.346 9m-4.788 0l9.26 9m9.968-3.21c.342.052.682.107 1.022.166m-1.022-.165L18.'/>
      </svg>
      <span className='ml-1'>Clear Database</span>
    </div>
  </button>
</div>
```

Abbildung 108 Clear Database Button Animations-Code

10.2.13 Usershow

Hier wird der zurzeit eingeloggte User angezeigt. Die Komponente wird auf allen Seiten bis auf den Login eingefügt.

```
return(
  <span className='ml-3 text-xs'>Logged in as: {isUser}</span>
)
```

Abbildung 110 UserShow Anzeige auf der Website

Logged in as: admin

Abbildung 109 UserShow Komponente auf der Webseite

10.2.14 React Router

Zuerst muss React-Router installiert werden

```
npm install --save react-router-dom
```

Innerhalb des Routes Tags können die einzelnen Routen via den Route Tag definiert werden.

```
import React from "react";
import "./App.css";
import { BrowserRouter as Router, Route, Link, Routes } from "react-router-dom";
import { Login } from "./components/login";
import Mainpage from "./components/mainpage";
import Layout from "./components/layout";
import Entries from "./components/entries";
import About from "./components/about";

function App() {
  return (
    <div className="">

      <Routes>
        <Route path="/mainpage" element={<Mainpage />} />
        <Route path="/" element={<Layout />}/>
        <Route path="/Mainpage" element={<Mainpage />} />
        <Route path="/entries" element={<Entries />} /></Route>
        <Route path="/Entries" element={<Entries />} /></Route>
        <Route path="/about" element={<About />} /></Route>
        <Route path="/About" element={<About />} /></Route>

        {/* Optional index route if no nested routes match */}
        | <Route index element={<Login />} />
      </Routes>
    </div>
  );
}

export default App;
```

Abbildung 111 Routes in App.tsx

10.2.15 Navbar und Mobile-Menu

Die Navbar besteht aus drei Hauptkomponenten dem Logo rechts das, wenn es gedrückt wird den Benutzer wieder zurück auf die Mainpage oder den Login bringt, falls das Cookie bereits abgelaufen ist. Auf der linken Seite befinden sich die zwei anderen Komponenten, Links zum Navigieren der Webanwendung und ein Logout Button, der in den Firmenfarben gehalten ist.



Abbildung 112 Navbar Desktop

Die Navbar wurde mit der „Mobile-First“-Herangehensweise erstellt. Das heißt es wurde zuerst für Mobile gestylt und danach wurde mit Breakpoints das Design für größere Bildschirme erstellt. Die Mobile Version hat anstelle der Links ein sogenanntes Burger-Menu. In diesem Menu befinden die Links zu den Unterwebseiten und der Logout Button. Der Menu Button führt, wenn er geklickt wird, die Funktion *mobileMenu* aus. Diese Funktion toggelt die [isActive, setActive] useState, wenn diese den Wert false oder true wiedergibt, wird das Menu mittels dem hidden Attribut angezeigt oder versteckt.

```
const [isActive, setActive] = useState<boolean | undefined>(false);
```

Abbildung 115 isActive/setActive useState

```
const mobileMenu = () => { <button  
| setActive(!isActive);  
| onClick={mobileMenu}  
| type="button"  
};
```

Abbildung 114 Funktion mobileMenu

Abbildung 113 Button mobilMenu

```
<div  
| className={  
| isActive  
| ? "mobile-menu w-full md:block md:w-auto"  
| : "hidden mobile-menu w-full md:block md:w-auto"  
| }>
```

Abbildung 116 Style der das Mobilemenu versteckt oder anzeigen

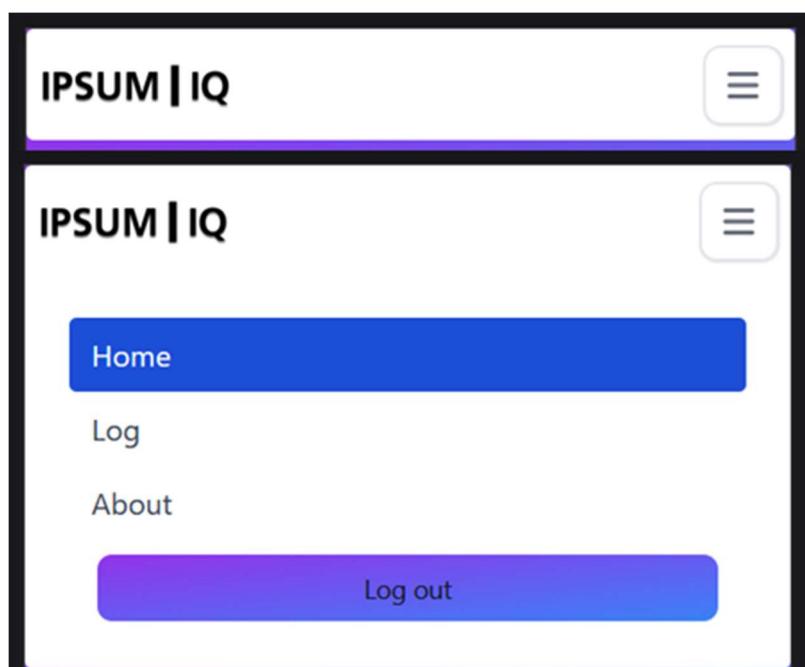


Abbildung 117 Mobile-Menu auf der Webseite

10.2.16 Stylen in verschiedenen Browsern

Hier werden Browser spezifische Styles des Inputfeldes vom Typ number entfernt

```
@layer base {
    input[type="number"]::-webkit-inner-spin-button,
    input[type="number"]::-webkit-outer-spin-button {
        -webkit-appearance: none;
        margin: 0;
    }

    input[type=number] {
        -moz-appearance: textfield; /* Firefox */
    }
}
```

Abbildung 118 Entfernung der browserspezifischen Styles

Mit `@-moz-document url-prefix()` können CSS Klassen erstellt die nur in Firefox funktionieren, dies kann genutzt werden um die Browserkompatibilität zu sichern.

```
@-moz-document url-prefix() {
    .NumericEntry {
        width: 5.7%;

    }

    .GeckoAesthetic {
        ...
    }
}
```

Abbildung 119 Styling in Firefox

11 Webanwendung: Backend (Goran K.)

11.1 Backend

Das Backend ist ein wesentlicher Teil einer jeden Software und Webseite. Ohne das Backend würden die meisten Anwendungen nicht funktionieren. Es beschreibt den Teil einer Software, welcher im Hintergrund abläuft. Der Teil auf den Benutzer keine Sicht und keinen Zugriff haben. Diese Ebene wird auch häufig Serverseite genannt. Im Backend werden administrative Aufgaben erledigt, welche für die Applikation benötigt werden. Beispielsweise Datenbankzugriffe, die Verwaltung von Anfragen, Authentifizierung etc.; Quelle: (Backend und Frontend einfach erklärt, 2023)

11.2 JavaScript

JavaScript ist eine Programmiersprache, welche für flexible Gestalten von Webseiten und die dynamische Anpassung von Webseiten entwickelt wurde. Eine der Hauptaufgaben von JavaScript im Frontend, ist das Manipulieren des HTML DOM. Beispielsweise das hinzufügen einer Klasse zu einem <div> Element. (Was ist eigentlich Node.js?, 2023)

11.3 Node.js

Node.js ist eine von Ryan Dahl entwickelte Laufzeitumgebung für JavaScript, welche es ermöglicht, JavaScript außerhalb der gewohnten Webbrowser Umgebung und stattdessen für serverseitige Anwendungen zu benutzen. Quelle: (Was ist eigentlich Node.js?, 2023)

Node.js bietet einige Vorteile gegenüber der gängigen Backend-Programmiersprache PHP, beispielsweise eine bessere Performance und ein leichteres Arbeiten mit JSON. (Node.js vs. PHP: An In-depth Comparison Guide for Web Development, 2023)

Bevor Node.js benutzt werden kann, muss es installiert werden.

Node.js wurde in diesem Projekt vorwiegend deshalb eingesetzt, da in Front- und Backend dadurch die gleiche Programmiersprache zum Einsatz kommt.

11.4 Node Package Manager (NPM)

NPM ist ein mit Node.js automatisch mitinstallierter Paket Manager. Mit Hilfe von NPM können viele Module und Pakete ganz einfach zu einem Projekt hinzugefügt werden. Quelle: (NPM - Node Package Manager, 2023)

11.5 Ein NPM-Projekt erstellen

Nachdem Node.js installiert worden ist, kann ganz leicht ein Node-Projekt erstellt werden. Als Erstes erstellt man einen Projektordner. Danach muss NPM gestartet werden, indem man mit dem Terminal seiner Wahl (CMD, PowerShell, Git, etc.) ins neu erstellte Verzeichnis navigiert und den Befehl **npm init --yes** eingibt. Dadurch wird eine neue Datei namens **package.json** im Projektordner erstellt. Quelle: (How To Start npm Project?, 2023)

11.6 Package.json

Die Datei **package.json** ist das Herz eines jeden Node.js Projektes. Sie beinhaltet Informationen über das Projekt, wie beispielsweise den Namen, die Beschreibung und die sogenannten **Dependencies** und **DevDependencies**. Dependencies sind die Module, welche zu einem Projekt hinzugefügt wurden und für die Funktionalität der Applikation benötigt werden (beispielsweise das Modul **express** für die Verwendung von Express.js). DevDependencies sind die Module, welche zu einem Projekt hinzugefügt wurden und für die Entwicklung der Applikation benutzt werden. (beispielsweise **nodemon**).

Somit müssen auf anderen Geräten die Module nicht manuell installiert, sondern lediglich der Befehl **npm install** im Verzeichnis der **package.json** ausgeführt werden. Der Befehl

installiert automatisch alle dort gelisteten Module. Quelle: (The Basics of Package.json, 2023)

```
{  
  "name": "ipsum-iq-server",  
  "version": "1.0.0",  
  "description": "Entwicklung eines smarten lichtschalters mit Webseite",  
  "main": "server.js",  
  ▷ Debug  
  "scripts": {  
    "test": "echo \"Error: no test specified\" && exit 1",  
    "start": "nodemon server.js",  
    "wsStart": "nodemon websocketserver.js"  
  },  
  "author": "",  
  "license": "ISC",  
  "dependencies": {  
    "bcrypt": "^5.1.0",  
    "body-parser": "^1.20.1",  
    "cookie-parser": "^1.4.6",  
    "cors": "^2.8.5",  
    "express": "^4.18.1",  
    "express-session": "^1.17.3",  
    "mysql": "^2.18.1",  
    "mysql2": "^2.3.3",  
    "rest": "^2.0.0",  
    "socket.io": "^4.5.4",  
    "websocket": "^1.0.34",  
    "ws": "^8.12.0"  
  },  
  "devDependencies": {  
    "nodemon": "^2.0.20"  
  }  
}
```

Abbildung 120 Package.json des Backends der Webanwendung

11.7 Express.js

Express.js ist ein Framework für Node.js welches für die Erstellung von Webanwendungen benutzt wird. Es bietet viele wichtige Funktionen, wie beispielsweise das Verwalten von HTTP-Anfragen wie GET und POST, welche von hoher Bedeutung in jeder Webanwendung sind. Um Express zu benutzen, muss es zuerst mit NPM installiert werden.

Nachdem die Installation beendet ist, kann Express mit folgenden Zeilen Code hinzugefügt werden:

```
const express = require('express');  
const app = express();
```

Abbildung 121 Implementierung von Express in der Serverdatei

Siehe auch (Was ist Express.js? Alles, was du wissen solltest, 2023)

11.7.1 Express Routing

Unter Routing versteht man, wie eine Applikation auf Anfragen eines Benutzers antwortet. In Express kann mit Hilfe des **app** Objektes auf die gewünschte Anfragemethode (häufig GET und POST) reagiert werden. Im unten abgebildeten Beispiel wird auf eine POST-Anfrage auf den Endpunkt /login gelauscht. Im zweiten Parameter der Funktion ist eine Callback Funktion mit 2 Parametern **req** und **res**. Mit **req** kann auf die vom Benutzer mitgesendeten Daten zugegriffen werden. Mit **res** kann eine Antwort an den Benutzer zurückgesendet werden. (Routing, 2023)

```

app.post('/login', (req, res) => {
  //Code zum Verwalten der Login-Anfrage
})

```

Abbildung 122 Express Routing Beispiel

11.8 Datenbanken

Eine Datenbank ist ein System zur effizienten, schnellen, fehlerfreien und dauerhaften Speicherung von Daten. Die verbreitetste Art von Datenbank ist die relationale Datenbank, welche die Daten in Tabellen in Form von Datensätzen speichert. Zur Erstellung und Verwaltung von Datenbanken benötigt man ein sogenanntes Datenbank-Management-System (kurz DBMS). Bei relationalen Datenbanken gibt es eine große Auswahl an Datenbank-Management-Systemen. Als Datenbanksprache wird in den meisten Fällen die Sprache SQL (Structured Query Language) verwendet. Siehe auch (Datenbank, 2023) Eines der verbreitetsten DBMS ist MySQL. Es wird in vielen Content-Management-Systemen wie WordPress oder TYPO3 zur Speicherung der Daten verwendet. Eine grafische Oberfläche gibt es in MySQL nicht. Dafür muss eine externe Software wie MySQL Workbench oder phpMyAdmin installiert werden. Quelle: (Was ist MySQL?, 2023)

11.9 Projektdatenbank

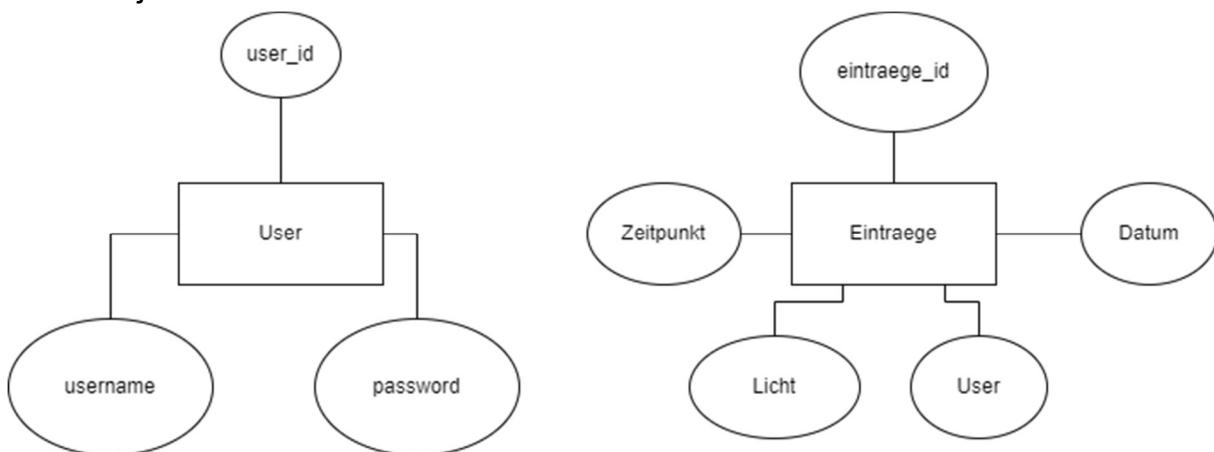


Abbildung 123 ER-Modell der Projektdatenbank

In der oben gezeigten Grafik sieht man die Datenbank, welche für das Projekt erstellt und benutzt worden ist. Sie besteht aus 2 Tabellen, eine Tabelle für die Benutzer und eine Tabelle für die Dokumentation der Veränderung des Lichtstatus. In der User Tabelle sind der Benutzername und das dazugehörige gehashte Passwort gespeichert. In der Einträge Tabelle sind die Zeitpunkte gespeichert, in denen ein Licht über der Webanwendung an- oder ausgeschalten wurde. Ebenso dokumentiert wird, welcher Benutzer welches Licht an- oder ausgeschalten hat. Für die Erstellung der Datenbank wurde phpMyAdmin verwendet.

+ Optionen								
	← T →	▼	eintraege_id	Zeitpunkt	Datum	Status	licht	user
<input type="checkbox"/>				29	16:15	17-01-2023	aus	ESP1 admin
<input type="checkbox"/>				31	16:15	17-01-2023	aus	ESP1 admin
<input type="checkbox"/>				32	8:34	19-01-2023	an	ESP2 user
<input type="checkbox"/>				34	8:34	19-01-2023	an	ESP2 user

Abbildung 124 Einträge Tabelle in phpMyAdmin

			user_id	username	password
<input type="checkbox"/>			6	admin	\$2b\$10\$eX.NrD1O0AQ2f0T9wPAeXeb9qQWN1Lj7QME.fcqrHPn...
<input type="checkbox"/>			7	user	\$2b\$10\$nperthoAOe69oiXkC5ghz.nDaM27PFqtPU0Ryjuixtu...

Abbildung 125 Benutzer Tabelle in phpMyAdmin

Mit folgenden Zeilen Code kann über Node.js auf einen MySQL-Server zugegriffen werden. Als erstes importiert man das Modul seiner Wahl für MySQL, in diesem Fall **mysql2**. Dieses muss vorher mit NPM installiert werden. In der Variable **connInfo** werden alle wichtigen Informationen für die Verbindung mit den MySQL-Server gespeichert. IP-Adresse des Servers, in unserem Fall 127.0.0.1 also der Localhost, der User, das Passwort, der Name der Datenbank und ein Limit für die maximalen Verbindungen, die gleichzeitig auf den Server stattfinden dürfen. In der Variable **connection** wird nun die Funktion **mysql.createConnection()** ausgeführt und als Parameter die Variable **connInfo** benutzt. Dadurch wird eine Verbindung mit dem Datenbankserver hergestellt. Jetzt kann die Datenbank in Node.js nach eigenen Wünschen manipuliert werden.

```
const mysql = require('mysql2/promise');
const connInfo = {
  host: "127.0.0.1",
  user: "root",
  password: "",
  database: "ipsum_iq",
  connectionLimit: 10
}
const connection = mysql.createConnection(connInfo);
```

Abbildung 126 Code für das Hinzufügen einer MySQL Datenbank in Node.js

11.10 Authentifizierung

11.10.1 Allgemein

Der Sinn von einem Authentifizierungsverfahren ist es, unbefugte Zugriffe auf seine Software zu verhindern. Je nachdem um welche Art von Software es sich handelt, kann ein nicht befugter Zugriff fatale Folgen haben. Deshalb wurde für IPSUM IQ ein Login-System entwickelt, welches unbefugte Zugriffe verhindern soll.

11.10.2 Erstellung der Benutzer und Passwörter

Einer der wichtigsten Bereiche in der Authentifizierung ist das Speichern der Passwörter. Denn diese sollten nicht einfach in Klartext in der Datenbank gespeichert werden. Dies bringt viele Sicherheitsrisiken mit sich. Deswegen werden Passwörter als **Hash** in Datenbanken gespeichert. Der große Vorteil an Hashes ist, dass sobald sie einmal generiert wurden, sie nicht mehr zum ursprünglichen Passwort zurückgewandelt werden können.

Beim Registrieren wird das eingegebene Passwort mittels eines Hash-Algorithmus in einen Hash umgewandelt und so in der Datenbank gespeichert. Beim Login wird das eingegebene Passwort mit demselben Algorithmus wieder gehasht und mit dem Hash aus der Datenbank verglichen. Ein kleiner Nachteil bei der Ganzen ist, dass bei 2 gleichen Passwörtern mit dem gleichen Hash-Algorithmus, der Hash gleich ist. Deshalb wird in den meisten Fällen ein sogenannter **Salt** mitgeneriert. Der Salt ist eine eigene Zahlen- und Ziffernkombination, welche für jeden Benutzer unterschiedlich ist. Dieser Salt wird in die Hash-Funktion mit dem

Passwort mitgegeben. Dadurch ist der Hash von 2 gleichen Passwörtern unterschiedlich.
(Passwort Hashing & Salted Password Hashing, 2023)

Für das Hashen der Passwörter wird das Modul Bcrypt verwendet.

```
const bcrypt = require('bcrypt');
```

Abbildung 127 Importieren von Bcrypt in Node.js

```
// Register Code
app.post('/users', async (req, res) => {
  try {

    const salt = await bcrypt.genSalt(10);
    const hashedPassword = await bcrypt.hash(req.body.password, salt);
    console.log(salt);
    console.log(hashedPassword);

    const user = { name: req.body.name, password: hashedPassword };
    const query = `INSERT INTO user (username, password) VALUES ("${user.name}", "${user.password}")`;
    res.status(201).send("Successfully registered");

    connection.pool.query(query, (err, result, fields) => {
      if (result) {
        return console.log(result);
      }
      else {
        return console.log(err);
      }
    });
  } catch {
    res.status(500).send("Fatal Error");
  }
});
```

Abbildung 128 Code für das Erstellen und Einfügen von Benutzer und Passwort in die Datenbank

Der oben gezeigte Code ist für die Erstellung und Speicherung der Benutzer und Passwörter in der Datenbank. Es wird auf eine POST-Anfrage auf den Endpunkt **/users** gehört. In dieser POST-Anfrage werden ein Benutzer und ein Passwort mitgeschickt. In der Variable **salt** wird mit der Funktion **bcyrpt.genSalt()** ein Salt generiert. Der Hash wird durch die Funktion **bcrypt.hash()**, welche als Parameter das Passwort und den Salt nimmt, generiert. Dieser Hash wird in einer Variable gespeichert. Dann wird ein Objekt **user** erstellt, welches die Eigenschaften **name** und **password** trägt. Für das Einfügen in die Datenbank wird in der Variable **query** eine INSERT INTO Abfrage gespeichert, welche als Werte den in der POST-Anfrage gesendeten Benutzernamen und das gehashte Passwort nehmen. Um die Abfrage auszuführen, wird die Funktion **connection.pool.query()** verwendet, welche als Parameter die SQL-Abfrage nimmt.

Ein Registrierfenster gibt es in der Finalen Webanwendung nicht. Der Code wurde nur einmalig zur Erstellung der Benutzer verwendet.

11.10.3 Login

Da nun die Benutzer mit ihren Passwörtern bereit sind, kann mit der Programmierung des Logins begonnen werden. Der Server hört auf eine POST-Anfrage an **/login** und bekommt einen Benutzer und ein Passwort von der Login Seite zugesendet. Im Callback wird als erstes eine SQL-Abfrage, welche alle Benutzer mit ihren Passwörtern aus der Datenbank nimmt, ausgeführt und das Ergebnis in eine Variable gespeichert. Die Abfrage gibt einen Array zurück, in dem die einzelnen Datensätze als Objekte gespeichert werden. Danach werden für alle Benutzer und ihre Passwörter Booleans erstellt, in denen die Benutzer und

Passwörter mit dem Usernamen und Passwort aus der POST-Anfrage verglichen werden. Wenn die Übereinstimmung passt, dann hat der Boolean den Wert True. Da die Passwörter in der Datenbank gehasht sind, muss das eingegebene Passwort ebenfalls gehasht werden, damit sie verglichen werden können. Die Funktion `bcrypt.compare()` erfüllt genau diesen Zweck. Sie vergleicht das zugesendete Passwort mit dem Hash aus der Datenbank und gibt True oder False zurück. Mit den Booleans kann nun mittels IF-Abfragen die gewünschten Szenarien ausprogrammiert werden.

```
app.post('/login', async (req, res) => {

  let an = await (await connection).query("SELECT username, password FROM user");
  let isAdmin = req.body.name === an[0][0].username;
  let isAdminPassword = await bcrypt.compare(req.body.password, an[0][0].password);
  let isUser = req.body.name === an[0][1].username;
  let isUserPassword = await bcrypt.compare(req.body.password, an[0][1].password);

  if (an) {
    if (isAdmin && isAdminPassword) {
      req.session.authenticated = true;
      req.session.user = an[0][0].username;
      res.send({ message: req.session.user });
      console.log("Admin Cookie set");
    }

    else if (isUser && isUserPassword) {
      req.session.authenticated = true;
      req.session.user = an[0][1].username;
      res.send({ message: req.session.user });
      console.log("User Cookie set");
    }

    else {
      res.send({ message: "Wrong Username or Password" });
    }
  }
  else {
    res.status(500).send("Fatal error :(");
  }
});

});
```

Abbildung 129 Server Login Code

Dieser Code funktioniert nur, wenn insgesamt nicht mehr als 2 Benutzer in der Datenbank existieren. Im Falle des Projektes werden nicht mehr als 2 Benutzer benötigt, weshalb sich für diese Lösung entschieden wurde.

11.10.4 Login in React

Um die eingegebenen Daten vom Benutzer an den Server weiterzuleiten, muss im Frontend die POST-Anfrage initialisiert werden. Dafür wird das Modul Axios verwendet.

```
import Axios from 'axios';
```

Abbildung 130 Axios importieren in React

Damit Axios funktioniert muss folgender Code eingegeben werden:

```
Axios.defaults.withCredentials = true;
```

Abbildung 131 Axios Konfiguration

Ebenso gibt es 3 useStateS, einmal für den Benutzernamen, einmal für das Passwort und einmal für den Login Status.

```
const [username, setUsername] = useState("");
const [password, setPassword] = useState("");
const [LoginStatus, setLoginStatus] = useState("");
```

Abbildung 132 2 useStateS für Login

Die useStateS bekommen ihre Werte von den Input Feldern des Logins.

```
<input
  className="ttit focus:ring-0 appearance-none"
  type="text"
  onChange={(e)=>
  {
    setUsername(e.target.value);
  }}
  placeholder="Username"
  required
/>
```

Abbildung 133 Eingegebenen Wert im Input-Feld der useState zuweisen

Die Funktion **PostLogin()** ist für die Login POST-Anfrage an den Server zuständig. Sie sendet in der Anfrage die Benutzernamen und Passwort useStateS an den Server. Der Server sendet, je nachdem ob Benutzer und Passwort stimmen, eine bestimmte Antwort an den Benutzer zurück. Wenn die Überprüfung stimmt, dann wird der Benutzer an die Hauptseite weitergeleitet. Wenn nicht, dann sendet der Server eine Nachricht zurück, welche den String „**Wrong Username or Password**“ beinhaltet. Dieser String wird der useState LoginStatus zugewiesen. PostLogin() wird beim Drücken des Login-Buttons ausgeführt.

```
function PostLogin() {
  Axios.post('http://localhost:3001/login', {
    name: username,
    password: password
  }).then((Response) => {
    if(Response.data.message) {
      setLoginStatus(Response.data.message);
      if(Response.data.message == "admin" || Response.data.message == "user") {
        navigate("/Mainpage");
      }
    }
  });
}
```

Abbildung 134 Funktion PostLogin()



Abbildung 135 Falscher Benutzer und Passwort auf Login-Seite

11.10.5 Session Cookies

Da nun der Benutzer eingeloggt ist, muss der Server sich diesen Benutzer merken. Dafür gibt es mehrere Optionen. Die aktuelle Weise, die von den meisten Applikationen verwendet wird, ist das JSON Web Token. Die veraltete Art ist der sogenannte Session Cookie. Session bedeutet auf Deutsch Sitzung. Eine Sitzung beginnt, sobald ein Benutzer eine Webseite aufruft. Auf dem Server wird eine Sitzung erstellt und eine Session ID wird an den Client gesendet. Die Session ID dient dazu, den Benutzer der richtigen Sitzung zuzuordnen. (Was sind Session-Cookies?, 2023)

Name	Wert
connect.sid	s%3AOtnsjOMd2yh7ReIHly1-18Mxbf3245LQ.Hq%2F0j%2BuJ4Kt35Bc4dyPAs...

Abbildung 136 Session Cookie ID

Um Session Cookies in Node.js zu benutzen, benötigen wir die Module Cookie-Parser und Express-Session.

```
const session = require("express-session");
const cookieParser = require("cookie-parser");
```

Abbildung 137 Importieren von Express-Session und Cookie-Parser

```
app.use(session({
  secret: 'aSAieSJfieimnfsiosI',
  cookie: { maxAge: 1000 * 60 * 10 },
  saveUninitialized: false,
  resave: false,
  store,
}));

});
```

Abbildung 138 Erstellung und Konfiguration der Session

Für die Sitzung können eine Reihe von Einstellungen konfiguriert werden. Einer dieser Einstellungen ist die **maxAge**. Sie gibt an, wie lange der Session Cookie gültig ist. Im Bild darüber sieht man, dass der Session Cookie auf 10 Minuten eingestellt wurde. Eine sehr

wichtige Einstellung ist das **Secret**. Das Secret wird, wie der Salt beim Hashen eines Passwortes, zum Hashen der Sitzung verwendet. Das Secret sollte wegen Sicherheitsgründen immer eine Kette aus zufälligen Buchstaben sein und nicht weitergegeben werden.

Eine wichtige Frage, die man sich bei Sessions stellen muss, ist, wo die Session gespeichert wird. Im Falle des Projektes wird jede Sitzung im Arbeitsspeicher des Servers gespeichert. (How do Express Sessions work?, 2023) Dafür verwendet man das Modul **Express-Session MemoryStore**.

```
const store = new session.MemoryStore();
```

Damit **MemoryStore** verwendet wird, muss man den Variablenamen, in dem Fall **store**, in die Session Konfiguration reinschreiben.

11.10.6 Prüfen, ob der User eingeloggt ist

Damit die ganze Arbeit für den Login und den Session Cookie nicht um sonst war, muss auch geschaut werden, dass der Benutzer nicht einfach /Mainpage in die Suchzeile vom Browser eingeben kann und somit den ganzen Authentifizierungsprozess überspringt. Dies wird folgendermaßen gelöst:

Auf der Serverseite wird beim Login, wenn Benutzer und Passwort übereinstimmen 2 Eigenschaften **authenticated** und **user** zur Session hinzugefügt. **Authenticated** wird auf true gesetzt und **user** ist gleich der Benutzername.

```
if (an) {
  if (isAdmin && isAdminPassword) {
    req.session.authenticated = true;
    req.session.user = an[0][0].username;
    res.send({ message: req.session.user });
    console.log("Admin Cookie set");
  }
}
```

Abbildung 139 Login Code, wenn Username und Passwort übereinstimmen

Auf der Clientseite benötigen wir UseEffect. Da UseEffect bei jedem Aufruf von der Webseite ausgeführt wird, eignet es sich perfekt zum Überprüfen, ob der User eingeloggt ist. Es wird eine GET-Anfrage an den Server gesendet. Auf dem Server wird überprüft, ob die Eigenschaft **req.session.user** in der Session des Benutzers existiert. Wenn sie existiert, heißt das, dass der Benutzer eingeloggt ist und der Server sendet einen Boolean **LoggedIn** mit dem Wert true zurück. Wenn nicht, dann sendet er ebenfalls LoggedIn aber mit dem Wert false zurück. Das Frontend schaut dann mittels IF-Abfragen, welches der beiden der Fall ist und navigiert den Benutzer entweder auf die gewünschte Seite oder auf die Login Seite zurück.

```

useEffect(() => {
  Axios.get("http://localhost:3001/Mainpage").then((Response) => {
    if (Response.data.LoggedIn) {
      console.log("LOGGED IN");
      navigate("/Mainpage");
    } else if (!Response.data.LoggedIn) {
      console.log("LOGGED out");
      navigate("/");
    }
  });
  ws();
}, []);

```

Abbildung 140 Überprüfung, ob der Benutzer eingeloggt ist (Frontend)

```

app.get('/Mainpage', (req, res) => {
  if (req.session.user) {
    res.send({ LoggedIn: true });
  } else {
    res.send({ LoggedIn: false });
  }
});

```

Abbildung 141 23 Überprüfung, ob der Benutzer eingeloggt ist (Backend)

Diese Überprüfung muss für jede Unterseite extra programmiert werden.

11.10.7 Logout Button

Zum Ausloggen gibt es eine Funktion **Logout()** welche eine POST-Anfrage an /logout sendet. Auf der Serverseite wird, nach einer Überprüfung, ob man eingeloggt ist, die Sitzung mit **req.session.destroy()** ungültig gemacht und im Callback eine Antwort an den Benutzer geschickt. Wenn der Benutzer diese Antwort erhält wird er auf die Login Seite zurückgeleitet. Diese Funktion wird beim Klicken des Logout Button ausgeführt.

```

function Logout() {
  Axios.post("http://localhost:3001/logout").then((Response) => {
    if (Response.data.LoggedOut == true) {
      navigate("/");
    }
  });
}

```

Abbildung 142 Logout Code Clientseite

```

app.post('/logout', (req, res) => {
  if (req.session.user) {
    req.session.destroy(() => {
      res.send({ LoggedOut: true });
    })
  } else {
    res.send({ LoggedOut: false });
  }
});

```

Abbildung 143 Logout Code Serverseite

11.11 WebSocket Server

Für das Konfigurieren eines Websocket Servers gibt es ein paar Module zur Auswahl. Im Projekt fiel die Entscheidung auf die Module **ws** und **SocketIO**. Ws für die Verbindung mit den Microcontrollern und SocketIO für die Verbindung mit dem Frontend.

```

const Websocket = require("ws");
const wss = new Websocket.Server({ port: 8080, clientTracking: true, });
const { Server } = require("socket.io");
const IO = new Server(server, {
  cors: {
    origin: ["http://localhost:3000", "ws://localhost:3000"]
  }
});

```

Abbildung 144 Importieren von ws und SocketIO

Sobald der Websocket Server läuft, hört er mit **wss.on(„connection“)** darauf, ob ein Websocket Client sich verbunden hat. Sobald ein Client sich verbindet, werden dem **ws** Objekt einige Eigenschaften hinzugefügt. Das **ws** Objekt ist für jeden Client einzigartig.

ws.isAlive	Zum Checken, ob der Client noch verbunden ist
ws.id	Name des Clients
ws.status	Status des Lichtes
ws.time	Zum Ausrechnen des Timers
ws.futureTime	Zum Ausrechnen des Timers
ws.timer	Variable für SetTimeout() Funktion

Mit **ws.on(„message“)** kann man auf Nachrichten, die ein Client sendet, hören. Der Client sendet je nachdem ob das Licht an oder aus ist, eine Statusmeldung, welche entweder 0 oder 1 ist. Wenn sie 0 ist, wird **ws.status** auf false gesetzt. Wenn sie 1 ist, wird **ws.status** auf true gesetzt.

```

wss.on('connection', (ws, req) => {

    let pathname = url.parse(req.url);
    ws.isAlive = true;
    ws.id = pathname.path.substring(1);
    ws.status = false;
    ws.time;
    ws.timer;
    ws.futureTime = 0;

    IO.emit("ledstate", { Message: ESPArray() });

    ws.on('pong', heartbeat);

    ws.on('message', message => {
        let msg;
        try {
            msg = JSON.parse(message)
        }
        catch (e) {
            console.log(e);
            return
        }

        if (msg.status == 0) {
            ws.status = false;
            ws.futureTime = 0;
            clearTimeout(ws.timer);

        }
        else if (msg.status == 1) {
            ws.status = true;

        }
        else {
            ws.send("Invalid message");
        }
        IO.emit("ledstate", { Message: ESPArray() });
    });
});

```

Abbildung 145 Websocket wss.on("connection") Code

11.12 Gruppierung aller Clients in einen Array

Die Funktion **ESPArray()** dient dazu, alle verbundenen Microcontroller in einen Array zu bringen, damit dieser dann ans Frontend gesendet werden kann. Es wird ein leerer Array erstellt. Mit **wss.clients.forEach()** kann durch alle Websocket Clients durchiteriert werden. Mit **ws.isAlive** wird geschaut, ob der Client verbunden ist. Für jeden Websocket Client werden die oben definierten Eigenschaften in den Array reingegeben. Zu guter Letzt gibt die Funktion **ESPArray** zurück.

```

let ESPArray = () => {
  let ESPArray = [];
  wss.clients.forEach(ws => {
    if (ws.isAlive) {
      ESPArray.push({ name: ws.id, on: ws.status, time: ws.time, futureTime: ws.futureTime });
    }
  })
  return ESPArray;
}

```

Abbildung 146 Funktion **ESPArray()**

11.12.1 SocketIO Server

SocketIO funktioniert im Grunde gleich wie jede andere Websocket Bibliothek, indem auf gewisse Events gelauscht wird. Der große Vorteil von SocketIO gegenüber anderen Modulen ist, dass man mit SocketIO eigene Events definieren kann. In anderen Websocket Modulen gibt es nur das **message** Event, um Nachrichten abzufangen. In SocketIO kann mit der **emit** Funktion ein eigenes Event ausgestrahlt werden. Im unten gezeigten Code wird für jede neue Verbindung mit dem SocketIO Server das Event **ledstate** ausgestrahlt und als Nachrichteninhalt die Funktion **ESPArray()** mitgegeben.

```

IO.on('connection', (socket) => {

  console.log('a user connected');

  socket.emit("ledstate", { Message: ESPArray() });

  socket.on('disconnect', () => {
    console.log('user disconnected');
  });

});

```

Abbildung 147 SocketIO `IO.on("connection")` Code

Im Frontend muss die Verbindung mit dem SocketIO Server hergestellt werden. Als erstes muss ein extra Modul für den SocketIO Client installiert werden.

```
import { io } from "socket.io-client";
```

Die Verbindung wurde folgendermaßen gelöst:

In der Datei der Hauptseite stellt eine Funktion **ws** eine Verbindung mit dem SocketIO Server her. Dann lauscht die Funktion mit `socket.on(“ledstate”)` auf das Event **ledstate**. Wenn das Event antrifft, dann wird die Nachricht in eine Variable gespeichert und in eine useState übertragen. Die Funktion wird mittels useEffect bei jedem neu laden der Webseite ausgeführt. Somit hat man beim Laden der Webseite sofort alle verbundenen Microcontroller und ihre aktuellen Stände.

```

const ws = () => {
  const socket = io("ws://localhost:3001");

  socket.on("ledstate", (data) => {

    let test = data.Message;
    lighttoggle(test);

  });
}

```

Abbildung 148 Funktion für die Verbindung zum SocketIO Server

11.13 Steuerung des Lichtes

Um das Licht nun an- oder auszuschalten, wird die Funktion **SetLightStatus()** verwendet. Sie hat einen Parameter, in dem der Name des betroffenen ESP reinkommt. Es wird eine POST-Anfrage an /state mit dem Licht Namen als Nachricht gesendet. Auf dem Server wird zuerst überprüft, ob der Benutzer eingeloggt ist. Dann wird die Nachricht des Benutzers in eine Variable gegeben. Anschließend wird mit **wss.clients.forEach()** durch alle Clients durchiteriert. Wenn der Name gleich der **ws.id** eines Clients ist, bedeutet das, dass der richtige Client gefunden wurde. Daraufhin wird die Nachricht „toggle“ an den ESP gesendet. SetLightStatus() wird bei jedem Drücken des An- oder Ausschalters ausgeführt.

```

function SetLightStatus(name: string) {
  Axios.post("http://localhost:3001/state", { ledname: name }).then((Response) => {

    if (Response.data.LoggedIn) {

    }

    else if (!Response.data.LoggedIn) {
      console.log("LOGGED out");
      navigate("/");
    }
  });
}

app.post('/state', (req, res) => {
  if (req.session.user) {
    let name = req.body.ledname;

    wss.clients.forEach(ws => {
      if (name == ws.id) {
        ws.send("toggle");
      }
    })

    res.send({ LoggedIn: true });
  }
  else {

    res.send({ LoggedIn: false });
  }
});

```

11.14 Automatische Dokumentation

Damit jedes An- und Ausschalten dokumentiert wird, wurde eine Funktion **InsertIntoDB()** erstellt. Gleich wie bei **SetLightStatus()** wird auch hier der Name des ESP als Parameter mitgegeben. Eine POST-Anfrage an /entries wird an den Server gesendet. Als Nachricht wird wieder der ESP Name mitgegeben. Auf dem Server wird ein neues **Date Object** erstellt, um das jetzige Datum mit dem jetzigen Zeitpunkt zu bekommen. In den Variablen drunter werden die einzelnen Teile des Zeitpunktes rausgeholt und dann zu einer **date** Variable und **time** Variable zusammengefügt. Ebenso werden Variablen für den User, für den Namen des Lichtes und für den Status des Lichtes definiert.

```
function InsertIntoDB(name: string) {
  Axios.post("http://localhost:3001/entries", { ledname: name }).then((Response) => {

    if (Response.data.LoggedIn) {

    }
    else if (!Response.data.LoggedIn) {
      console.log("LOGGED out");
      navigate("/");
    }
  });
}
```

Abbildung 149 Dokumentation Code Clientseite

```
app.post('/entries', async (req, res) => {
  if (req.session.user) {

    let date_ob = new Date();
    let day = ("0" + date_ob.getDate()).slice(-2);
    let month = ("0" + (date_ob.getMonth() + 1)).slice(-2);
    let year = date_ob.getFullYear();
    let hours = date_ob.getHours();
    let minutes = ("0" + (date_ob.getMinutes() + 1)).slice(-2);
    let seconds = date_ob.getSeconds();
    let date = day + "-" + month + "-" + year;
    let time = hours + ":" + minutes;

    let user = req.session.user;
    let licht = req.body.ledname;
    let status;
```

Abbildung 150 Dokumentation Code Serverseite Teil 1

Um den richtigen Status zu erhalten, wird mit **wss.clients.foreach()** durch alle Microcontroller durchiteriert und wenn der Name des Microcontroller mit dem Namen aus der POST-Anfrage übereinstimmt, wird geschaut, ob **ws.status** true oder false ist. Wenn es true ist, bekommt **status** den Wert „an“. Wenn nicht dann bekommt sie den Wert „aus“. Zu guter Letzt wird in einer Variable die INSERT INTO Abfrage mit den ganzen Variablen gespeichert und in die Funktion **connection.query()** übergeben.

```

        wss.clients.forEach(ws=> {
            if (req.body.ledname == ws.id) {
                if (ws.status) {
                    status = "an";
                }
                else if (ws.status == false) {
                    status = "aus";
                }
            }
        });

        const InsertQuery = `INSERT INTO eintraege (Datum, Zeitpunkt, user, licht, Status) VALUES ("${date}", "${time}", "${user}", "${licht}", "${status}")`;
        (await connection).query(InsertQuery);
        res.send({ LoggedIn: true });
    }
    else {
        res.send({ LoggedIn: false });
    }
});

});

```

Abbildung 151 Dokumentation Code Serverseite Teil 2

```
<button type="button" onClick={()=>{SetLightStatus(light.name); InsertIntoDB(light.name)}>
```

Abbildung 152 Aufrufen der Funktionen im Button

11.14.1 Anzeigen der Einträge

Um die Datenbankeinträge anzuzeigen und aktuell zu halten, wird im UseEffect der Komponente **querylist.tsx** eine GET-Anfrage an /entries gesendet. Auf der Serverseite wird eine SELECT Abfrage, welche alle Einträge aus der Tabelle **eintraege** holt, ausgeführt und in einer Variable gespeichert. Nach einer Überprüfung ob man eingeloggt ist und ob der eingeloggte Benutzer der Admin ist, wird das Ergebnis der Datenbankabfrage an den Client zurückgesendet. Im Frontend wird das Abfragen Ergebnis dann in die useState **querys** übergeben.

```

useEffect(() => {

    Axios.get('http://localhost:3001/entries').then((Response) => {

        if (Response.data.result) {
            setQuerys(Response.data.result);
        }

        else {
            console.log("error");
        }
    });
}, [querys]);

```

Abbildung 153 UseEffect Funktion für das Anzeigen der Datenbankeinträge

```

app.get('/entries', async (req, res) => {
  let queryresult = await (await connection).query("SELECT user, Zeitpunkt, Licht, Status, Datum FROM eintraege ORDER BY eintraege_id DESC LIMIT 15;");

  if (req.session.user) {
    if (req.session.user == "admin") {
      res.send({ LoggedIn: true, result: queryresult, isAdmin: true });
    }
    else {
      res.send({ LoggedIn: true, isAdmin: false });
    }
  }
  else {
    res.send({ LoggedIn: false });
  }
});

```

Abbildung 154 Servercode für Anzeige der Datenbankeinträge

11.15 Automatisches Ausschalten / Berechnung des Timers

Damit der Timer und dessen Anzeige richtig für jedes Licht angezeigt werden, müssen die benötigten Zahlen auf der Serverseite berechnet werden. Im Frontend wird aus den beiden Inputfeldern die eingegebene Stunde und Minute rausgeholt und in 2 useState **hours** und **minutes** übertragen.

```

onChange={(event: any) => {(event.target.value = event.target.value.slice(0, 2));sethours(event.target.value);

```

Abbildung 155 Übertragung der Werte aus dem Inputfeld in eine useState

Mit der Funktion **setTime()**, welche als Parameter den Namen des Lichtes und den Status nimmt, wird eine POST-Anfrage an /time gesendet. Als Nachricht werden die eingegebenen Stunden und Minuten, der Name des Lichtes und der Status mitgesendet.

```

function setTime(ESP: string, statusled: boolean) {
  Axios.post("http://localhost:3001/time", { ledhours: hours, ledminutes: minutes, ESPName: ESP, status: statusled }).then((Response) => {
    if(Response.data.futureTime) {
    }
  });
}

```

Abbildung 156 Funktion zum Senden der Stunden und Minuten an den Server

Auf der Serverseite werden als erstes die Stunden und Minuten in eigene Variablen gegeben. Ebenso wird eine neue Variable **date** erstellt, welche ein **Date Object** speichert. Mit der Funktion **getTime()** wird das jetzige Datum in Millisekunden umgerechnet. Nach einer Überprüfung, ob der Benutzer eingeloggt ist, wird mit **wss.clients.forEach()** für jeden ESP geschaut, ob er den gleichen Namen, wie der Name aus der Anfrage hat. Wenn dies der Fall ist, wird die Dauer des Timers in Millisekunden umgerechnet. Ebenso wird die „zukünftige Zeit“ ausgerechnet, indem man die **date** Variable mit **ws.time** addiert. Die zukünftige Zeit ist das jetzige Datum mit den eingegebenen Stunden und Minuten addiert. Mit diesem Wert kann nun im Frontend die Timer Anzeige berechnet werden.

Danach wird überprüft, ob **ws.status** auf false gesetzt ist. Da das automatische Ausschalten nur funktionieren kann, wenn das Licht auch wirklich an ist, wird beim Fall von **ws.status == false ws.futureTime** auf 0 gesetzt und das SocketIO Event **ledstate** gerufen, um die neuen Eigenschaften der Lichter dem Frontend zu schicken.

Wenn **ws.status** nicht false ist, dann wird in **ws.timer** eine **setTimeout** Funktion gestartet. SetTimeout ist eine in JavaScript eingebaute Funktion, welche nach einer angegebenen Zeit den Code in der Callback Funktion ausführt. In **setTimeout** wird lediglich die Nachricht „off“ an den ESP gesendet. Als Zeitangabe wird **ws.time** verwendet. Danach wird das Event **ledstate** an das Frontend geschickt.

```

app.post('/time', (req, res) => {

    let hours = +req.body.ledhours;
    let minutes = +req.body.ledminutes;
    let date = new Date().getTime();

    if (req.session.user) {
        wss.clients.forEach(ws => {
            if (ws.id == req.body.ESPName) {

                ws.time = (hours * 60 + minutes) * 60 * 1000
                console.log("HOURS " + hours);

                ws.futureTime = date + ws.time;

                if(ws.status == false) {
                    ws.futureTime = 0;
                    IO.emit("ledstate", { Message: ESPArray() })
                }
                else {
                    ws.timer = setTimeout(() => {

                        ws.send("off");

                    }, ws.time)
                    IO.emit("ledstate", { Message: ESPArray() })
                }
            }
        })
        res.send({ LoggedIn: true })
    }
    else {
        res.send({ LoggedIn: false });
    }
});

```

Abbildung 157 Server Code zum Berechnen des Timers

11.15.1 Abbrechen des Timers

Um den Timer abzubrechen, gibt es die Funktion **ClearTimer()** in der **esp.tsx** Komponente, welche durch das Drücken des Timer-abbrechen Buttons ausgeführt wird.

```

function ClearTimer(ESP: string) {
    Axios.post("http://localhost:3001/timeclear", {ESPName: ESP}).then((Response) => {
    })
}

```

Abbildung 158 Funktion zum Timer abbrechen Frontend

Sie sendet eine POST-Anfrage an /timeclear und sendet den Namen des Lichtes mit. Auf der Serverseite regelt die Funktion **ClearTime()** das Löschen des Timers. Sie hat einen Parameter für den Namen des Lichtes. Es wird für jeden Microcontroller geschaut, ob dieser denselben Namen wie der Parameter **name** hat. Wenn dies der Fall ist, dann wird die Funktion **clearTimeout()** aufgerufen. clearTimeout() wird verwendet, um laufende Timeout Funktionen zu stoppen. Die **setTimeout()** Funktion, welche in **ws.timer** gespeichert ist, wird dadurch gestoppt. Damit im Frontend die Anzeige des Timers ebenso stoppt, wird **ws.futureTime** auf 0 gesetzt und das Event **ledstate** wird aufgerufen.

```
function ClearTime(name) {  
    wss.clients.forEach(ws => {  
        if (ws.id == name) {  
            clearTimeout(ws.timer);  
            ws.futureTime = 0;  
            IO.emit("ledstate", { Message: ESPArray() })  
        }  
    })  
}
```

Abbildung 159 Funktion zum Timer abbrechen Backend

Diese Funktion wird, wenn eine POST-Anfrage auf /timeclear ankommt, ausgeführt. Als Parameter wird der mitgesendete Name des Lichtes dazugegeben.

```
app.post('/timeclear', (req, res) => {  
    if (req.session.user) {  
        ClearTime(req.body.ESPName);  
        res.send({ LoggedIn: true });  
    }  
    else {  
        res.send({ LoggedIn: false });  
    }  
})
```

Abbildung 160 Ausführen der ClearTime() Funktion

12 Literaturverzeichnis

- Backend und Frontend einfach erklärt.* (25. 1 2023). Von
<https://www.ionos.de/digitalguide/websites/webseiten-erstellen/backend-frontend/>
abgerufen
- Bacon, R. S. (21. 01 2023). #224  STOP using Serial.print in your Arduino code! THIS is better.. Von
Youtube: <https://www.youtube.com/watch?v=--KxxMaiwSE> abgerufen
- Bedeutung der Farben ["Farbsymbolik und Farben in der Psychologie"].* (21. Jänner 2023). Von
Designer in Action: <https://www.designeraction.de/design-wissen/bedeutung-farben>
abgerufen
- Buy a Raspberry Pi 4 Model B – Raspberry Pi.* (21. 01 2023). Von raspberrypi.com/:
<https://www.raspberrypi.com/products/raspberry-pi-4-model-b/> abgerufen
- Configuring MariaDB for Remote Client Access - MariaDB Knowledge Base.* (2. Februar 2023). Von
mariadb: <https://mariadb.com/kb/en/configuring-mariadb-for-remote-client-access/>
abgerufen
- Corporate Design – Definition, Beispiele und Tipps.* (20. Jänner 2023). Von [Arbeitstipps.de:](https://arbeitstipps.de/corporate-design-beispiele-tipps.html)
<https://www.arbeitstipps.de/corporate-design-beispiele-tipps.html> abgerufen
- Corporate Design – Wie gut präsentiert sich Ihr Unternehmen?* (20. Jänner 2023). Von [inconcepts:](https://inconcepts.at/agentur/blog/blog-corporate-design/)
<https://www.inconcepts.at/agentur/blog/blog-corporate-design/> abgerufen
- CSS - Ausführliche Erklärung aus dem Hosting-Lexikon.* (22. Jänner 2023). Von [checkdomain:](https://www.checkdomain.de/hosting/lexikon/css/)
<https://www.checkdomain.de/hosting/lexikon/css/> abgerufen
- CSS Frameworks - mfg.* (22. Jänner 2023). Von mfg: <https://mfg.fhstp.ac.at/allgemein/css-frameworks/> abgerufen
- Datasheet: FTR-F1R relay - ftr_f1r-944176.pdf.* (30. Jänner 2023). Von Mouser Electronics:
https://www.mouser.at/datasheet/2/164/ftr_f1r-944176.pdf abgerufen
- Datenbank.* (19. 1 2023). Von <https://www.weclapp.com/de/lexikon/datenbank/> abgerufen
- Einführung in Hooks - React - lernen und verstehen.* (23. Jänner 20203). Von [React - lernen und verstehen:](https://lernen.react.js.dev/hooks/einfuehrung) <https://lernen.react.js.dev/hooks/einfuehrung> abgerufen
- ENG_DS_OJ_OJE_series_relay_data_sheet_E_1120.pdf.* (30. Jänner 2023). Von TE Connectivity:
https://www.te.com/commerce/DocumentDelivery/DDEController?Action=showdoc&DocId=Data+Sheet%7FOJ_OJE_series_relay_data_sheet_E%7F1120%7Fpdf%7FEnglish%7FENG_DS_OJ_OJE_series_relay_data_sheet_E_1120.pdf%7F1-1419129-8 abgerufen
- ESP8266 Wikipedia.* (30. Jänner 2023). Von Wikipedia: <https://de.wikipedia.org/wiki/ESP8266>
abgerufen
- Farblehre - Bedeutung der Farbe Schwarz / Polster Fischer.* (21. Jänner 2023). Von Polster-Fischer:
<https://www.polster-fischer.de/servicewelt/farbenlehre/was-bedeutet-die-farbe-schwarz/>
abgerufen
- Favicon - Was ist das - Seobility Wiki.* (22. Jänner 2023). Von seobility wiki:
<https://www.seobility.net/de/wiki/Favicon> abgerufen
- Heroicons.* (20. Dezember 2022). Von Heroicons: <https://heroicons.com> abgerufen

How do Express Sessions work? (2023, 2 1). Retrieved from <https://codeburst.io/how-do-express-sessions-work-c465e0488af4>

How To Start npm Project? (22. 1 2023). Von <https://dev.to/moek/how-to-start-npm-project-2bga> abgerufen

Html einfach und verständlich erklärt - SEO-Küche. (22. Jänner 2023). Von seo-kueche: <https://www.seo-kueche.de/lexikon/html/> abgerufen

<https://cplusplus.com/reference/cstdlib/atoi/>. (2. 2 2023). Von cplusplus.com: <https://cplusplus.com/reference/cstdlib/atoi/> abgerufen

Installation - Tailwind CSS. (22. Jänner 2023). Von Tailwind CSS: <https://tailwindcss.com/docs/installation> abgerufen

IRLZ44N - Infineon Technologies. (30. Jänner 2023). Von Infineon Technologies: <https://www.infineon.com/cms/de/product/power/mosfet/n-channel/irlz44n/> abgerufen

Learning Electronics with Fritzing. (31. Jänner 2023). Von Fritzing: <https://fritzing.org/learning/> abgerufen

LM1117 800-mA, Low-Dropout Linear Regulator datasheet (Rev. Q). (30. Jänner 2023). Von Texas Instruments: https://www.ti.com/lit/ds/symlink/lm1117.pdf?ts=1675043333270&ref_url=https%253A%252F%252Fwww.google.com%252F abgerufen

Node.js vs. PHP: An In-depth Comparison Guide for Web Development. (2023, 1 19). Retrieved from <https://distantjob.com/blog/nodejs-vs-php>

NPM - Node Package Manager. (2023, 1 19). Retrieved from <https://www.tutorialsteacher.com/nodejs/what-is-node-package-manager>

Passwort Hashing & Salted Password Hashing. (26. 1 2023). Von <https://www.datenschutzexperte.de/blog/datenschutz-im-unternehmen/passwort-hashing-salted-password-hashing/> abgerufen

Preprocessor - cppreference.com. (21. 01 2023). Von [cppreference.com: https://en.cppreference.com/w/cpp/preprocessor](https://en.cppreference.com/w/cpp/preprocessor) abgerufen

Raspberry Pi 4 Model B kaufen. (30. Jänner 2023). Von [raspberrypi.com: https://www.raspberrypi.com/products/raspberry-pi-4-model-b/](https://www.raspberrypi.com/products/raspberry-pi-4-model-b/) abgerufen

React Hooks – Eine Einführung - Informatik Aktuell. (23. Jänner 2023). Von Informatik Aktuell: <https://www.informatik-aktuell.de/entwicklung/programmiersprachen/react-hooks-eine-einfuehrung.html> abgerufen

ReactJS installation and setup / ReactJS tutorials for beginners. (23. Jänner 2023). Von Pragim Technologies: <https://www.pragimtech.com/blog/reactjs/install-reactjs/> abgerufen

Routing. (2023, 1 29). Retrieved from <https://expressjs.com/en/guide/routing.html>

Schriftportrait Frutiger - Frutiger kaufen. (21. Jänner 2023). Von [Schriftgestaltung.com: https://schriftgestaltung.com/schriftlexikon/schriftportrait/frutiger.html](https://schriftgestaltung.com/schriftlexikon/schriftportrait/frutiger.html) abgerufen

SVG-Dateien | Adobe. (22. Jänner 2023). Von Adobe: <https://www.adobe.com/at/creativecloud/file-types/image/vector/svg-file.html> abgerufen

Tailwind CSS - Einfach erklärt | Twigbit Technologies Digitale Lösung nach Maß. (22. Jänner 2023).

Von Twigbit: <https://www.twigbit.com/glossar/tailwind> abgerufen

Tailwind CSS. (22. Dezember 2022). Von Tailwind CSS: <https://tailwindcss.com/> abgerufen

Tech Stack | Überblick der Top-Technologien | 2022. (22. Jänner 2023). Von freelancermap:

<https://www.freelancermap.de/blog/tech-stack/> abgerufen

The Basics of Package.json. (2023, 1 19). Retrieved from <https://nodesource.com/blog/the-basics-of-package-json/>

tzapu. (30. 01 2023). *tzapu/WiFiManager: ESP8266 WiFi Connection manager with web captive portal.* Von [github.com: https://github.com/tzapu/WiFiManager](https://github.com/tzapu/WiFiManager) abgerufen

Was ist eigentlich Node.js? (19. 1 2023). Von <https://www.mittwald.de/blog/hosting/was-ist-eigentlich-node-js> abgerufen

Was ist Express.js? Alles, was du wissen solltest. (19. 1 2023). Von <https://kinsta.com/de/wissensdatenbank/was-ist-express-js/> abgerufen

Was ist Frontend — was Backend? (20. Jänner 2023). Von [Frontend-gmbh: https://www.frontend-gmbh.de/blog/frontend-und-backend/](https://www.frontend-gmbh.de/blog/frontend-und-backend/) abgerufen

Was ist MySQL? (19. 1 2023). Von <https://www.bigdata-insider.de/was-ist-mysql-a-614184/> abgerufen

Was ist React.js? Ein Blick auf die beliebte JavaScript-Bibliothek. (23. Jänner 2023). Von Kinsta: <https://kinsta.com/de/wissensdatenbank/was-ist-react-js/#was-ist-react> abgerufen

Was sind Session-Cookies? (1. 2 2023). Von <https://www.ionos.de/digitalguide/hosting/hosting-technik/was-sind-session-cookies/> abgerufen

13 Abbildungsverzeichnis

Abbildung 1: Organigramm der Hardware	14
Abbildung 2 ESP8266-01S / eigene Aufnahme.....	16
Abbildung 3 Raspberry PI 4 2Gb / eigene Aufnahme.....	16
Abbildung 4 3.3V Stepdown Board / eigene Aufnahme	16
Abbildung 5 IOR IRLZ44N FET/ eigene Aufnahme.....	16
Abbildung 6 FUJITSU F1CL012R 12V Relais / eigene Aufnahme	17
Abbildung 7 Fritzing Bauteile Menü / eigene Aufnahme	18
Abbildung 8 Fritzing Beispieldschaltplan / eigene Aufnahme.....	18
Abbildung 9 Schaltplan IPSUM IQ / eigene Aufnahme	19
Abbildung 10 Steckboard Beispiel / eigene Aufnahme	20
Abbildung 11 Steckboard IPSUM IQ halbfertig / eigene Aufnahme	20
Abbildung 12 Steckboard IPSUM IQ fertig / eigene Aufnahme	21
Abbildung 13 Beispiel Fädeldraht / eigene Aufnahme.....	22
Abbildung 14 Lochraster Bild 1 / eigene Aufnahme.....	22
Abbildung 15 Lochraster Bild 2 / eigene Aufnahme.....	23
Abbildung 16 Lochraster Bild 3 / eigene Aufnahme.....	23
Abbildung 17 Lochraster Bild 4 / eigene Aufnahme.....	24
Abbildung 18 Lochrasterplatine IPSUM IQ Rückseite / eigene Aufnahme	24
Abbildung 19 Lochrasterplatine IPSUM IQ Vorderseite / eigene Aufnahme	24
Abbildung 20 IPSUM IQ Versuchsaufbau leer / eigene Aufnahme	25
Abbildung 21 IPSUM IQ Versuchsaufbau Vorderseite / eigene Aufnahme	25
Abbildung 22 IPSUM IQ Versuchsaufbau Rückseite / eigene Aufnahme.....	26
Abbildung 23 Jumper auf 1.5mm Draht Adapterkabel / eigene Aufnahme	26
Abbildung 24 Raspberry Pi Imager Hauptseite / eigene Aufnahme.....	28
Abbildung 25 Raspberry Pi Imager Erweiterte Optionen / eigene Aufnahme.....	28
Abbildung 26 Raspberry Pi Imager Schreiben / eigene Aufnahme	29
Abbildung 27 IPSUM IQ Server Rückseite / eigene Aufnahme	29
Abbildung 28 IPSUM IQ Server Vorderansicht / eigene Aufnahme	29
Abbildung 29 Colasoft MAC Scanner / eigene Aufnahme.....	30
Abbildung 30 Windows Terminal SSH / eigene Aufnahme	30
Abbildung 31 raspi-config / eigene Aufnahme.....	31
Abbildung 32 VNC Viewer Verbinden / eigene Aufnahme.....	31
Abbildung 33 Raspberry Pi Desktop / eigene Aufnahme	31
Abbildung 34 Terminal apt-get update / eigene Aufnahme	32
Abbildung 35 apt-get install npm / eigene Aufnahme	32
Abbildung 36 git clone / eigene Aufnahme.....	33
Abbildung 37 npm i / eigene Aufnahme	33
Abbildung 38 apt-get install mariadb-server / eigene Aufnahme.....	34
Abbildung 39 MySQL Shell / eigene Aufnahme.....	34
Abbildung 40 create database / eigene Aufnahme.....	35
Abbildung 41 Importieren der SQL Datei / eigene Aufnahme	35
Abbildung 42 MySQL Berechtigungen erteilen / eigene Aufnahme	36
Abbildung 43 bind-address / eigene Aufnahme.....	36
Abbildung 44 EPS-01S USB Programmierer; Selbstaufnahme	37
Abbildung 45 EPS-01S USB Programmer mit Widerstand zwischen GPIO0 und GND; Selbstaufnahme	37
Abbildung 46 C++ Debugging Makro; Quelle: (Bacon, 2023).....	37
Abbildung 47 git clone befehl Selbstaufnahme.....	38
Abbildung 48 WifiManager minimal code; Selbstaufnahme	38

Abbildung 49 WiFiManager Gesamtcode; Selbstaufnahme	39
Abbildung 50 WiFiManagers Konfigurationsseite; Selbstaufnahme	40
Abbildung 51 Hauptmenu der WiFiManager Konfigurationsseite; Selbstaufnahme	40
Abbildung 52 EEPROM Code; Selbstaufnahme	41
Abbildung 53 WebSocket Code; Selbstaufnahme	43
Abbildung 54 Hardware-Interaktionscode; Selbstaufnahme	44
Abbildung 55 Pindefinition; Selbstaufnahme	45
Abbildung 56 Timer-Code; Selbstaufnahme	45
Abbildung 57 Reset Timer; Selbstaufnahme	46
Abbildung 58 rising edge detector; Selbstaufnahme	46
Abbildung 59 setLed Funktion; Selbstaufnahme	47
Abbildung 60 Grafik Frontend (Was ist Frontend – was Backend?, 2023)	48
Abbildung 61 Grafik Corporate Design (Corporate Design – Definition, Beispiele und Tipps, 2023)	48
Abbildung 62 Logos - Ipsum IQ	48
Abbildung 63 Farben - Ipsum IQ	49
Abbildung 64 Favicon - Ipsum IQ	49
Abbildung 65 Icons (Heroicons, 2022)	49
Abbildung 66 Glühbirne aus	50
Abbildung 67 Glühbirne an	50
Abbildung 68 Techstack - Ipsum IQ	50
Abbildung 69 Tailwind Logo (Tailwind CSS, 2022)	51
Abbildung 70 Befehl zur Installation von Tailwind (Installation - Tailwind CSS, 2023)	51
Abbildung 71 Tailwind.config.js (Installation - Tailwind CSS, 2023)	51
Abbildung 72 Einfügen Tailwind Richtlinien (Installation - Tailwind CSS, 2023)	51
Abbildung 73 Befehl zum Starten des CLI-build-Prozess (Installation - Tailwind CSS, 2023)	51
Abbildung 74 Hintergrund des Logins	53
Abbildung 75 Flexbox zu zentrierung des Logins	53
Abbildung 76 Login Fenster	53
Abbildung 77 Username Inputfeld	53
Abbildung 78 Passwort Inputfeld	54
Abbildung 79 useState isvis/setvis und die Funktion pasvis	54
Abbildung 80 Passwort Inputfeld Vergleich	54
Abbildung 81 Passwort Inputfeld und Funktionen	54
Abbildung 82 Login Button Hover Effekt	54
Abbildung 83 ESP useState	55
Abbildung 84 map Funktion	55
Abbildung 85 ESP Komponente im Frontend	55
Abbildung 86 Code des Glühbirnen Button	56
Abbildung 87 Konstante allowedNum	56
Abbildung 88 if-Abfrage zur Überprüfung des User-Inputs	56
Abbildung 89 if-Abfragen um den Timer mit Enter zu starten	56
Abbildung slice() Methode und if-Abfrage Inputfeld-1 ESP	57
Abbildung slice() Methode und if-Abfrage Inputfeld-2 ESP	57
Abbildung Funktion TimeSubmittable	57
Abbildung Code des Timer Button	57
Abbildung Funktion RunningTime	57
Abbildung Timer und Timerbutton Code	58
Abbildung Veränderung der On/Off Buttons	58
Abbildung Timer Rechnung	58
Abbildung if-Abfrage um im positiven Zahlenbereich zu bleiben	59

Abbildung useEffect zur Neuberechnung.....	59
Abbildung Weitergabe der Ergebnisse zur Clock Komponente	59
Abbildung Eigenschaftsdefinition Clock Komponente	59
Abbildung Umwandlung der Zeit in Strings.....	60
Abbildung Clock Ausgabe als HTML Element	60
Abbildung iframe Code.....	61
Abbildung Darstellung des iframes auf der Website.....	61
Abbildung Tabelle auf der Webseite	62
Abbildung Code der Tabelle inkl. map Funktion	62
Abbildung Clear Database Button Animations-Code	63
Abbildung UserShow Komponente auf der Webseite	63
Abbildung UserShow Anzeige auf der Website.....	63
Abbildung Routes in App.tsx	63
Abbildung Navbar Desktop.....	64
Abbildung Button mobilMenu.....	64
Abbildung isActive/setActive useState.....	64
Abbildung Funktion mobileMenu.....	64
Abbildung Style der das Mobilemenu versteckt oder anzeigt	64
Abbildung Mobile-Menu auf der Webseite.....	64
Abbildung Entfernung der browserspezifischen Styles.....	65
Abbildung Styling in Firefox.....	65
Abbildung Package.json des Backends der Webanwendung.....	67
Abbildung Implementierung von Express in der Serverdatei.....	67
Abbildung Express Routing Beispiel	68
Abbildung ER-Modell der Projektdatenbank.....	68
Abbildung Einträge Tabelle in phpMyAdmin.....	68
Abbildung Benutzer Tabelle in phpMyAdmin	69
Abbildung Code für das Hinzufügen einer MySQL Datenbank in Node.js.....	69
Abbildung Importieren von Bcrypt in Node.js.....	70
Abbildung Code für das Erstellen und Einfügen von Benutzer und Passwort in die Datenbank	70
Abbildung Server Login Code	71
Abbildung Axios importieren in React.....	71
Abbildung Axios Konfiguration.....	72
Abbildung 2 UseStates für Login	72
Abbildung Eingegebenen Wert im Input-Feld der useState zuweisen	72
Abbildung Funktion PostLogin()	72
Abbildung Falscher Benutzer und Passwort auf Login-Seite	73
Abbildung Session Cookie ID	73
Abbildung Importieren von Express-Session und Cookie-Parser	73
Abbildung Erstellung und Konfiguration der Session	73
Abbildung Login Code, wenn Username und Passwort übereinstimmen.....	74
Abbildung Überprüfung, ob der Benutzer eingeloggt ist (Frontend).....	75
Abbildung 23 Überprüfung, ob der Benutzer eingeloggt ist (Backend)	75
Abbildung Logout Code Clientseite	75
Abbildung Logout Code Serverseite	76
Abbildung Importieren von ws und SocketIO	76
Abbildung Websocket wss.on("connection") Code.....	77
Abbildung Funktion ESPArray()	78
Abbildung SocketIO IO.on("connection") Code.....	78
Abbildung Funktion für die Verbindung zum SocketIO Server.....	79

Abbildung Dokumentation Code Clientseite	80
Abbildung Dokumentation Code Serverseite Teil 1	80
Abbildung Dokumentation Code Serverseite Teil 2	81
Abbildung Aufrufen der Funktionen im Button.....	81
Abbildung UseEffect Funktion für das Anzeigen der Datenbankeinträge.....	81
Abbildung Servercode für Anzeige der Datenbankeinträge.....	82
Abbildung Übertragung der Werte aus dem Inputfeld in eine useState	82
Abbildung Funktion zum Senden der Stunden und Minuten an den Server.....	82
Abbildung Server Code zum Berechnen des Timers	83
Abbildung Funktion zum Timer abbrechen Frontend	83
Abbildung Funktion zum Timer abbrechen Backend	84
Abbildung Ausführen der ClearTime() Funktion.....	84