|  |
| --- |
|  |
| Drahtloses Türschild mit E-Paper-Display |
| Projektarbeit |
|  |
| **Autoren:**  Niclas Muss &  Jan-Philipp Töberg |
| **Studiengang:**  Technische Informatik  **Projektbetreuer:**  Prof. Dr. Thomas Korte  **Abgabedatum:**  02.07.2018 |

|  |
| --- |
|  |

# Inhaltsverzeichnis

[Inhaltsverzeichnis I](#_Toc518256361)

[Tabellenverzeichnis II](#_Toc518256362)

[Abbildungsverzeichnis II](#_Toc518256363)

[1. Einleitung 1](#_Toc518256364)

[1.1 Aufgabenstellung (Muss) 1](#_Toc518256365)

[1.2 Projektbeschreibung (Muss) 1](#_Toc518256366)

[1.3 Projektumfeld (Töberg) 1](#_Toc518256367)

[2. Projektplanung 2](#_Toc518256368)

[2.1 Projektphasen (Töberg) 2](#_Toc518256369)

[2.2 Projektressourcen (Muss) 2](#_Toc518256370)

[2.2.1 ESP32-Microcontroller 2](#_Toc518256371)

[2.2.2 Waveshare E-Paper-Display 3](#_Toc518256372)

[2.3 Gemeinsames Arbeiten 3](#_Toc518256373)

[2.3.1 Versionsverwaltung über GitHub (Töberg) 3](#_Toc518256374)

[2.3.2 Backlogverwaltung über Google Sheets (Muss) 4](#_Toc518256375)

[3. Analysephase (Töberg) 5](#_Toc518256376)

[3.1 Der c’t-Micro-Controller 5](#_Toc518256377)

[3.2 Der c’t-Webserver 6](#_Toc518256378)

[3.3 Anforderungsanalyse 7](#_Toc518256379)

[4. Entwurfsphase 7](#_Toc518256380)

[4.1 Entwurf des Anzeigebilds (Muss) 7](#_Toc518256381)

[4.2 Entwurf des Webinterface (Töberg) 8](#_Toc518256382)

[4.3 Entwurf des Netzwerks (Muss) 10](#_Toc518256383)

[5. Implementierungsphase 10](#_Toc518256384)

[5.1 Bildanzeige über PHP (Muss) 10](#_Toc518256385)

[5.1.1 index.php 11](#_Toc518256386)

[5.1.2 türschild.php 12](#_Toc518256387)

[5.2 Das Webinterface (Töberg) 15](#_Toc518256388)

[5.2.1 HTML und JavaScript auf Client-Seite 16](#_Toc518256389)

[5.2.2 PHP auf Server-Seite 17](#_Toc518256390)

[6. Fazit 18](#_Toc518256391)

[6.1 Soll-/Ist-Vergleich (Töberg) 18](#_Toc518256392)

[6.2 Ausblick (Muss) 18](#_Toc518256393)

[7. Quellen 19](#_Toc518256394)

[A Anhang A](#_Toc518256395)

[A1 – Netzwerkdiagramme A](#_Toc518256396)

[A2 – Das Webinterface A](#_Toc518256397)

[A3 – Fotos der Türschilder (vorher) B](#_Toc518256398)

[A4 – Fotos der Türschilder (nachher) C](#_Toc518256399)

[A5 – Abkürzungsverzeichnis D](#_Toc518256400)

[A6 – QR-Code zum GitHub-Repository D](#_Toc518256401)

[A7 – Artikel aus der c’t E](#_Toc518256402)

[A8 – Die „index.php“-Datei H](#_Toc518256403)

[A9 – Die „turschild\_V1.php“-Datei K](#_Toc518256404)

[A10 – Die „webinterface.html“-Datei N](#_Toc518256405)

[A11 – Die “getDataFromCSV.php“-Datei Q](#_Toc518256406)

[A12 – Die “saveDataToCSV.php“-Datei R](#_Toc518256407)

[A13 – Die Anschlüsse des SparkFun ESP32 Thing T](#_Toc518256408)

# Tabellenverzeichnis

[Tabelle 1 - Vergleich der Webserver-Schnittstellen 9](#_Toc518256409)

[Tabelle 2 - Aufbau der professor.csv mit zwei Beispieleintragungen 19](#_Toc518256410)

# Abbildungsverzeichnis

[Abbildung 1 - Überblick über die Backlogverwaltung (detaillierte Ansicht im Anhang …) 5](file:///C:\Users\Janfi\Documents\Projektarbeit\Drahtloses-Tuerschild-PA\Dokumentation\Ausarbeitung%20komplett.docx#_Toc518256411)

[Abbildung 2 - Entwurf des Anzeigebildes (Version 01) 8](file:///C:\Users\Janfi\Documents\Projektarbeit\Drahtloses-Tuerschild-PA\Dokumentation\Ausarbeitung%20komplett.docx#_Toc518256412)

[Abbildung 3 - Entwurf des Webinterface (Version 01) 9](file:///C:\Users\Janfi\Documents\Projektarbeit\Drahtloses-Tuerschild-PA\Dokumentation\Ausarbeitung%20komplett.docx#_Toc518256413)

[Abbildung 4 - Entwurf des Webinterface (Version 02) 9](file:///C:\Users\Janfi\Documents\Projektarbeit\Drahtloses-Tuerschild-PA\Dokumentation\Ausarbeitung%20komplett.docx#_Toc518256414)

[Abbildung 5 - Veranschaulichung des Zusammenhangs von csv-Datei und Anzeige 10](#_Toc518256415)

[Abbildung 6 - Entwicklung des Anzeigebildes über den Projektzeitraum 13](file:///C:\Users\Janfi\Documents\Projektarbeit\Drahtloses-Tuerschild-PA\Dokumentation\Ausarbeitung%20komplett.docx#_Toc518256416)

[Abbildung 7 - Kommunikation zwischen Server und Client 16](file:///C:\Users\Janfi\Documents\Projektarbeit\Drahtloses-Tuerschild-PA\Dokumentation\Ausarbeitung%20komplett.docx#_Toc518256417)

[Abbildung 8 - Entwurf des Netzwerks (Version 01) A](file:///C:\Users\Janfi\Documents\Projektarbeit\Drahtloses-Tuerschild-PA\Dokumentation\Ausarbeitung%20komplett.docx#_Toc518256418)

[Abbildung 9 - Entwurf des Netzwerks (Version 02) A](file:///C:\Users\Janfi\Documents\Projektarbeit\Drahtloses-Tuerschild-PA\Dokumentation\Ausarbeitung%20komplett.docx#_Toc518256419)

[Abbildung 10 - Das aktuelle Webinterface (Version 02) A](file:///C:\Users\Janfi\Documents\Projektarbeit\Drahtloses-Tuerschild-PA\Dokumentation\Ausarbeitung%20komplett.docx#_Toc518256420)

[Abbildung 11 - Türschild von Prof. Hausdörfer (vorher) B](file:///C:\Users\Janfi\Documents\Projektarbeit\Drahtloses-Tuerschild-PA\Dokumentation\Ausarbeitung%20komplett.docx#_Toc518256421)

[Abbildung 12 - Türschild von Prof. Korte (vorher) B](file:///C:\Users\Janfi\Documents\Projektarbeit\Drahtloses-Tuerschild-PA\Dokumentation\Ausarbeitung%20komplett.docx#_Toc518256422)

[Abbildung 13 - Türschild von Prof. Hausdörfer (nachher) C](file:///C:\Users\Janfi\Documents\Projektarbeit\Drahtloses-Tuerschild-PA\Dokumentation\Ausarbeitung%20komplett.docx#_Toc518256423)

[Abbildung 14 - Türschild von Prof. Korte (nachher) C](file:///C:\Users\Janfi\Documents\Projektarbeit\Drahtloses-Tuerschild-PA\Dokumentation\Ausarbeitung%20komplett.docx#_Toc518256424)

[Abbildung 15 - QR-Code für GitHub D](file:///C:\Users\Janfi\Documents\Projektarbeit\Drahtloses-Tuerschild-PA\Dokumentation\Ausarbeitung%20komplett.docx#_Toc518256425)

# 1. Einleitung

## 1.1 Aufgabenstellung (Muss)

Im Heft 2/2018 der Zeitschrift c't wird unter dem Titel Ausdauernde Infotafel der Aufbau und Programmierung eines stromsparenden drahtlosen Türschilds mit E-Paper-Display beschrieben. So ein Türschild soll für das Büro im Raum 1.365 aufgebaut und mit erweiterten Funktionsumfang programmiert werden. So soll es neben der Übertragung von Bildern auch möglich sein, schnell einen Ankündigungstext - so wie auf dem Info-Monitor auf der 4. Etage - aus dem Büro heraus auf das Türschild via WLAN zu übertragen. Es muss dazu nicht nur ein Mikrocontroller und ein Web Server programmiert, sondern auch eine sichere Befestigung für Display, Controller und Batterien konstruiert werden. In der c`t ist dazu ein Beispiel angegeben. Dies ist ein Team-Projekt für zwei Personen. Alle benötigten Komponenten bis auf die Befestigungsmaterialien sind bereits vorhanden.

## 1.2 Projektbeschreibung (Muss)

Im Rahmen der Projektarbeit im Sommersemester 2018 soll das in der Aufgabenstellung beschriebene Türschild implementiert werden. Dafür wurde mehrere Wochen lang mit dem Mikrocontroller „SparkFun ESP32 Thing“ und dem „Waveshare 7.5inch e-Paper HAT“ arbeiten können.

Das Ziel des Projekts war es, alle in der Aufgabenstellung genannten Anforderungen zu erfüllen und, wenn möglich, noch einige weitere Features hinzuzufügen Neben dem geforderten Ankündigungstext soll auch ein Anwesenheitsstatus und einige Informationen über den Professor, welchem das Büro gehört, angezeigt werden. Sowohl der ESP32, als auch der E-Paper Display sollen außerdem über eine Batterie betrieben werden und müssen dementsprechend stromsparend betrieben werden. Alle angesprochenen Einzelteile sollen sicher an der Wand vor dem Raum 1.365 befestigt werden mit einer Wandhalterung, welche mit einem 3D-Drucker hergestellt werden soll.

Als Referenz wurde der Artikel „Ausdauernde Infotafel“ aus der Zeitschrift c’t im Helft 2/2018 nutzen, das dort vorgegebene Projekt ließ sich gut für die Ansprüche des Projekts erweitern.

## 1.3 Projektumfeld (Töberg)

Dieses Projekt ist Teil des Moduls „Projektarbeit“ im Bachelor-Studiengang „Technische Informatik“ an der Hochschule Ostwestfalen-Lippe in Lemgo. Bei diesem Modul handelt es sich um ein Pflichtmodul des vierten Semesters, welches von den Studierenden fordert die Inhalte ihres bisherigen Studienverlaufs in einem gewählten Themengebiet anzuwenden und zu vertiefen. Dabei steht vor allem eine praxisnahe und konzentrierte Bearbeitung der gestellten Aufgabenstellung im Vordergrund. [[1]](#footnote-1)

Dieses Projekt ist in der Projektgruppe von Professor Doktor Thomas Korte entstanden, welcher mehrere Projekte für ein bis zwei Personen zum Thema „Physical Computing“ anbietet. So wurde ein zweites Projekt, mit einem ähnlichen Thema angeboten, welches von Mathis Mohr bearbeitet wurde und sich mit der Programmierung des Türschildes mit einem anderen Microcontroller auseinandersetzte. Des Weiteren wurde ein Projekt für die Entwicklung einer ToDo-Liste für den Schreibtisch von Ka Yung Cheng umgesetzt, welches eine kleinere Version des hier genutzten e-Paper Displays verwendet hat.

Eine weitere Schnittstelle dieses Projekt ist, wie in Abschnitt „1.2 Projektbeschreibung“ bereits angesprochen, das Projekt aus der Computerzeitschrift c’t (Ausgabe 02/2018), welches eine Grundlage für die Ansteuerung / Darstellung des Türschilds mit Hilfe des ESP32-Controllers bildet[[2]](#footnote-2). Dieses Projekt setzt selbst auf der sogenannten „Basecamp“-Bibliothek, welche von der c’t entwickelt wurde und genutzt wird, um auf leichte und modifizierbare Art und Weise aus Microcontrollern IoT-Geräte zu machen, auf[[3]](#footnote-3). Die letzte Bibliothek, die von diesem Projekt verwendet wird, ist die sogenannte „Espressif“-Bibliothek, welche dazu verwendet wird, den SparkFun ESP32 Thing über die Arduino-IDE verwenden zu können[[4]](#footnote-4).

# 2. Projektplanung

## 2.1 Projektphasen (Töberg)

Die einzelnen Abschnitte des Projekts lassen sich in vier unterschiedliche Phasen einteilen: Planung, Analyse, Entwurf und die Umsetzung. Begonnen wurde dabei logischerweise mit der Planungsphase, in welcher das weitere Vorgehen während des Projektes überlegt und geplant wurde. Dabei wurde sich einmal mit der Hardware vertraut gemacht, aber auch Lösungen für die Probleme des Versionsmanagements und der Aufgabenverfolgung gesucht & gefunden.

In der darauffolgenden Analysephase wurde sich intensiv mit dem Programm der Zeitschrift c’t auseinandergesetzt. Dabei wurde dessen Ablauf und grundlegende Struktur verinnerlicht, sowie notwendige Anpassungen vorgenommen. Insgesamt wurde während dieser Phase ein gutes Verständnis der möglichen Ansatzpunkte für dieses Projekt gesammelt, was das Entwerfen von eigenen, ergänzenden Funktionalitäten, vereinfacht.

Nachdem das grundlegende Programm analysiert und verstanden wurde, wurde sich mit dem Entwurf der neuen Möglichkeiten, welche das Projekt in Zukunft bieten soll, auseinandergesetzt. Dabei wurden Prototypen der graphischen Oberflächen angefertigt und auch im Nachhinein iterativ überarbeitet.

Zu guter Letzt wurden die geplanten Funktionen gemäß den Entwürfen umgesetzt. Die Vorgehensweise und die daraus resultierenden Ergebnisse werden im Abschnitt „5. Implementierungsphase“ detaillierter beschrieben.

Insgesamt wurde mit einer angepassten Form des Wasserfallmodells gearbeitet, da, im Gegensatz zum herkömmlichen Wasserfallmodell, Prozessphasen mehrmals durchlaufen wurden. So wurden die Planungs- und die Entwurfsphase im Nachhinein noch angepasst, wenn Änderungen in der Implementierungsphase aufgetreten sind. Generell wurden die einzelnen Phasen aber erst komplett abgearbeitet, bevor mit der Arbeit in der nächsten Phase begonnen wurde.

## 2.2 Projektressourcen (Muss)

### 2.2.1 ESP32-Microcontroller

Für das Projekt wurde ein Microcontroller verwendet, der Wifi Funktionalitäten bietet und außerdem batteriebetrieben arbeiten kann. Das Programm der c’t, welches als Grundlage dient, nutzte den ESP32. Der ESP32 „Thing“ von der Firma SparkFun ist ein Microcontroller aus der gleichen Familie, wie der, welcher von der c’t benutzt wurde und erfüllt, auf dem ersten Blick, alle erweiterten Anforderungen, die wir zusätzlich zur c’t hatten.

Der Controller besitzt einen integrierten 802.11 BGN WiFi Transceiver, mit dem die geforderten Wifi Funktionalitäten problemlos abgedeckt werden können. Der ESP32 kann seinen eigenen Wifi Access Point aufbauen, über den man eine Verbindung direkt zum Controller herstellen kann, als auch sich mit anderen Routern verbinden, wodurch alle Geräte im gleichen Netzwerk sich mit dem Controller verbinden können. Um diesen Wifi Access Point jedoch zielführend nutzend zu können, muss ein Programm diesen öffnen und eine Benutzeroberfläche anbieten. Für unsere Zwecke benutzen wir die Basecamp Library, welche auch von der c’t genutzt wurde.

Durch einen integrierten „LiPo Battery Charger“ ist der langfristige Batteriebetrieb möglich, vor allem da sich der Controller, die meiste Zeit im Tiefschlafmodus befinden soll, in welchem er nur auf einer Stromstärke von 2.5 µA läuft. Wenn sich der Controller im Tiefschlafmodus befindet, gibt es mehrere Möglichkeiten ihn aufzuwecken, welche vorher definiert werden müssen. Unter anderem auch eine Möglichkeit, in welcher der Controller seine Wifi Funktionalitäten aufrechterhält und auf einen Wakeup-Call über sein Wifi-Modul reagieren kann, welche sich für das Projekt anbietet. Bei einem aktiven Betrieb kann der Battery Charger jedoch nicht mit dem Betriebsverbrauch mithalten. Leider konnte während der Projektarbeit nicht getestet werden, ob der Betrieb über eine Batterie mit dem endgültigen Programm funktionieren würde, da keine Batterie zur Verfügung gestanden hat.

Das einzige Problem, welches sich im Umgang mit dem ESP32 „Thing“ gezeigt hat, war der zu geringe Arbeitsspeicher. Mit 520 kB internem SRAM besitzt der Controller recht viel Speicher im Vergleich zu anderen Controllern seiner Größe, jedoch sind es leider zu wenig um all die Daten zu bearbeiten, die das Waveshare Display braucht um Farbe anzuzeigen. Das c’t Programm selbst war auf ein schwarz-weißes Display ausgelegt und das Hinzufügen einer weiteren Farbe verdoppelt die Anzahl an Daten, die an das Waveshare Display gesendet werden müssen, mehr dazu in der Beschreibung des Waveshare Displays. Dieses Problem ließ sich innerhalb des Projektzeitraums nicht lösen, da die vom Controller und der GD-Bibliothek unterstützte Möglichkeit einer paged-Speicherung und Anzeige nicht mit dem gegebenen Programm der c’t kompatibel war und daher eine sehr aufwändige Änderung wäre

### 2.2.2 Waveshare E-Paper-Display

Zur Anzeige des Türschilds wurde ein Display, welches Stromsparend arbeiten kann benötigt. E-Paper Displays erfüllen dieses Kriterium, da sie nur beim erstmaligen „Zeichnen“ eines Bildes Strom verbrauchen und dieses Bild ohne weiteren Stromverbrauch „halten“ können. Für den spezifischen Fall dieses Projekts bedeutet das, dass ein Türschild, welches einmalig auf das E-Paper Display gezeichnet wurde, von diesem ohne jeglichen weiteren Stromverbrauch bis zum nächsten Zeichnen des Türschildes, angezeigt werden kann.

Der “7.5inch e-Paper HAT” der Firma Waveshare benutzt die Anzeigetechnik „Microencapsulated Electrophoretic Display (MED)“. Diese Technologie lässt Micro-Kapseln gefüllt mit elektrisch geladenen weißen Partikel in gefärbten Öl schweben. Für jede einzelne Kapsel wird elektronisch festgelegt, ob die weißen Partikel oben oder unten in der Kapsel schweben sollen, so dass der Betrachter des Displays entweder die weißen Partikel oder das farbige Öl wahrnimmt. Sobald dieser Zustand einmal festgelegt ist, ändert sich der Zustand nicht von selbst, weswegen das angezeigte Bild für eine lange Zeit angezeigt werden kann, ohne weiter Strom zu benötigen.

Für das Projekt wird die Version des Displays benutzt, die neben schwarz und weiß noch eine Farbe anzeigen kann. Die dritte Farbe ist normalerweise Rot, weswegen oft zwischen Schwarz-Weiß und Schwarz-Weiß-Rot unterschieden wird. Die dritte Farbe unsers Displays ist jedoch Gelb. Für die Software macht dies jedoch keinen Unterschied, da der Display genau die gleichen Werte übergeben bekommet, wie an einem Display mit Rot als dritter Farbe. Aus diesem Grund sind viele der Variablen innerhalb des Skriptes auf die Farbe Rot bezogen, obwohl die endgültige Ausgabe Gelb sein soll.

## 2.3 Gemeinsames Arbeiten

### 2.3.1 Versionsverwaltung über GitHub (Töberg)

Da bei Softwareprojekten an denen mehr als eine Person beteiligt ist, sehr schnell Probleme bezüglich der Versionsverwaltung auftreten, haben wir uns entschieden ein Tool für die Versionsverwaltung zu verwenden. Unsere Entscheidung fiel auf Grund von persönlichen Erfahrungen auf GitHub welches von der GitHub Inc. entwickelt wird, welche im Laufe des Projekts von Microsoft aufgekauft wurden.

Der Name GitHub leitet sich von dem Versionsverwaltungssystem Git ab, für welches GitHub eine Online-Plattform mit den Aspekten der sozialen Medien verknüpft. So kann jeder Nutzer hier eigene, sogenannte Repositories *(zu Deutsch: Ablage / Archiv)* erstellen um seine Projekte zu verwalten. Dabei können die einzelnen Projekte sehr einfach mit anderen geteilt oder von anderen favorisiert werden. Dieser soziale Aspekt erleichtert das Finden von interessanten Projekten, an welchen man mitarbeiten kann. Außerdem behält man so stets den Überblick über die Projekte von befreundeten Entwicklern und kann diesen bei eventuellen Problemen leicht aushelfen.

Dabei verwendet GitHub die, oben bereits genannten, Repositories. Dabei handelt es sich um eine Art Ordner, welcher alle Dateien eines bestimmten Software-Projekts zentriert sammelt und immer einem Benutzerkonto zugeordnet ist. Dieser Ordner kann nun lokal kopiert werden um Dateien hinzuzufügen oder zu ändern. Sobald Änderungen vorgenommen wurden, kann der Nutzer diese „committen“. Dabei werden seine Änderungen mit dem aktuellen Stand auf dem Server verglichen und danach dort hochgeladen. Jedem Commit wird eine bestimmte Beschreibung und eine eindeutige Nummer zugeordnet, was dafür sorgt, dass jeder Commit als eine Version der Software gesehen werden kann. GitHub gibt dem Entwickler nun die Möglichkeit, bei Problemen oder Bugs alte Versionen der Software wieder zu laden um mit diesen weiterzuarbeiten.

Wenn nun mehrere Entwickler an einem Projekt arbeiten wollen, so muss der Besitzer des Repositories die anderen einladen, bevor sie als sogenannte „Contributors“ *(zu Deutsch: Mitwirkender)* den vollen Zugriff auf den gesammelten Quellcode haben. Um allen Entwicklern die Möglichkeit zu geben, nicht ständig die Arbeit der anderen während der Entwicklung durch regelmäßige Commits zu behindern, können sogenannte „Branches“ (zu Deutsch: Äste) angelegt werden. Diese erlauben die parallele Entwicklung ab einem bestimmten Startpunkt, bis zu welchem alle Branches auf dem gleichen Stand sind. Nach der Trennung kann ein Entwickler seinen Ast bearbeiten, bis er sein gewünschtes Feature umgesetzt hat. Daraufhin kann er seinen Ast wieder mit dem Hauptast (hier: master, häufig aber auch „Trunk“ genannt) verknüpfen, um allen anderen Entwicklern sein Ergebnis zur Verfügung zu stellen.

Wenn jedoch zwei Entwickler so zeitnah Änderungen an der gleichen Datei vornehmen, dass sich beide Änderungen überschneiden würden, so gilt das Prinzip „Wer zuerst kommt, malt zuerst“. Das heißt der Entwickler, welcher zuerst committed, kann seine Änderungen problemlos hochladen. Wenn der andere Entwickler dann seine Ergebnisse committen möchte, so wird er von GitHub darauf hingewiesen, dass seine Datei von der Server-Datei abweicht. Dann muss ein manuelles Abgleichen der Änderungen geschehen, um zu entscheiden, wie beide Dateien verknüpft werden sollen.

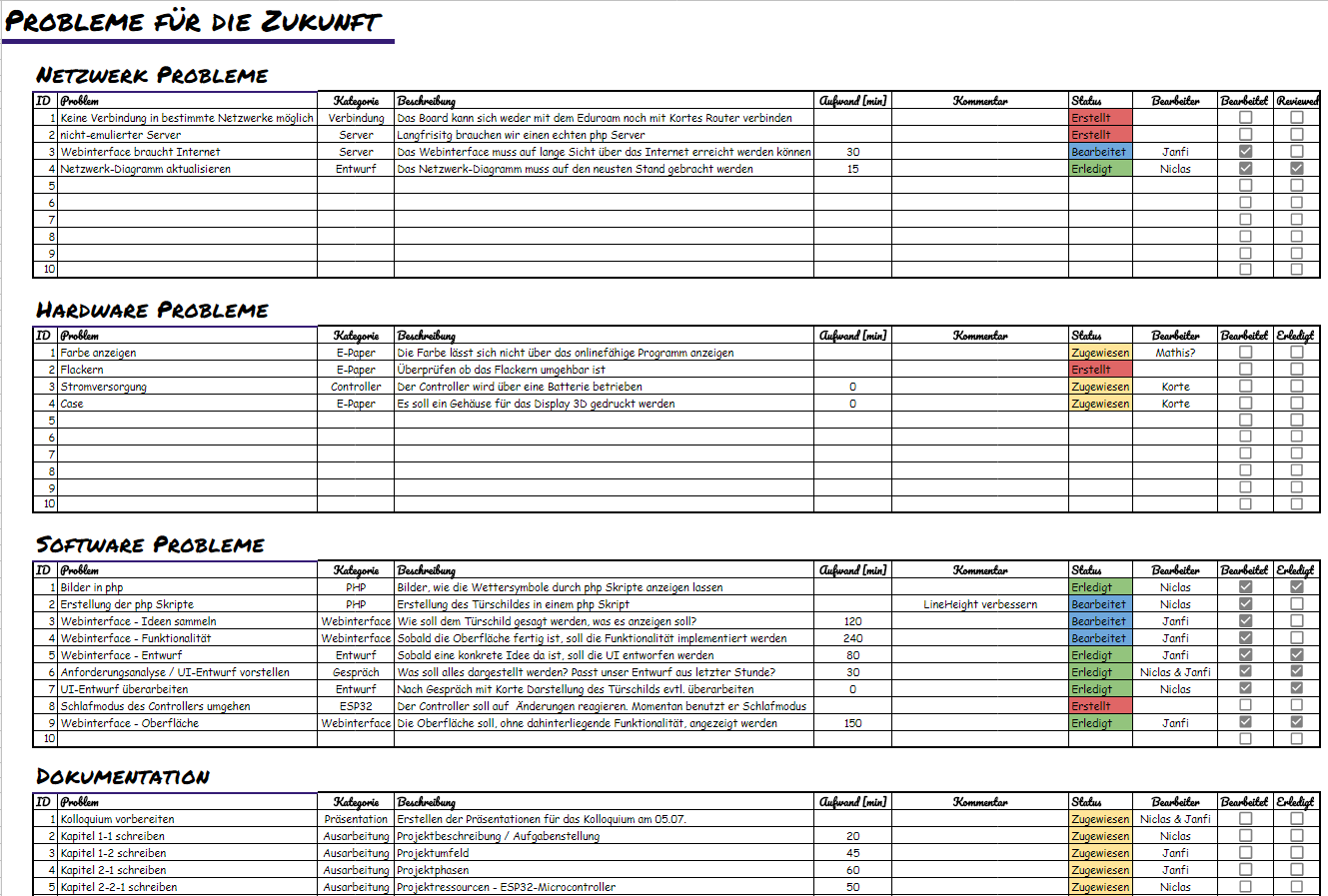
### 2.3.2 Backlogverwaltung über Google Sheets (Muss)

Neben GitHub für die Versionsverwaltung sollte ein Überblick darüber existieren, was wir bereits erledigt haben und was noch zu tun ist. Im Team wurdesich aufeine tabellarische Übersicht geeinigt, welche dann über Googles cloudbasiertes Tabellenbearbeitungsprogramm „Google Sheets“ umgesetzt wurde.

Diesogenannten Backlogs lassen sich in vier Oberkategorien unterteilen: Netzwerk, Hardware, Software und Dokumentation. Jede der Oberkategorien hat zur besseren Übersicht eine eigene Tabelle bekommen, in welcher wiederrum mehrere einzelne Backlogs verwaltet werden.

Ein Backlog besteht aus mehreren Punkten:

Abbildung 1 - Überblick über die Backlogverwaltung (detaillierte Ansicht im Anhang …)



* Problem:

Name des Backlogs in wenigen Worten zusammengefasst.

* Kategorie:

Kategorie des Backlogs. Etwas weiter eingeschränkt als die Oberkategorie.

* Beschreibung:

ausführliche Beschreibung des Backlogs.

* Aufwand:

Geschätzter Aufwand, der für die Bearbeitung benötigt wird.

* Kommentar:

Platz für Kommentare. Um den Status etwas genauer zu beschreiben.

* Status: Momentane Status des Backlogs (automatisch generiert).
* Bearbeiter: Name der Person, die diesen Backlog bearbeitet.

Der Status wird basierend auf den Eingaben in den anderen Feldern des Backlogs berechnet. Sobald etwas in der Spalte „Problem“ eingetragen wird, wird der Status auf „Erstellt“ gesetzt. Der Status wird automatisch auf „Zugewiesen“ gesetzt, wenn das Feld „Bearbeiter“ gefüllt wird. Zwei Checkboxen sollen nun die übrigen Stati verwalten. Der erste Haken soll gesetzt werden, wenn der Bearbeiter die Bearbeitung abgeschlossen hat. Dieser Vorgang setzt den Status auf “Bearbeitet“. Nach einer Überprüfung durch die zweite Person darf diese den zweiten Haken setzen, und somit den Status auf „Erledigt“ setzen. Die Statusspalte an sich ist jedoch zu eingeschränkt um alleine anzeigen zu können, wie der Backlog bearbeitet wurde oder wird. Zur beschreibbaren Erweiterung der Statusspalte soll die Beschreibung genutzt werden um, wenn nötig, Einzelheiten zu Bearbeitung des Backlogs angeben zu können.

Die Backlogverwaltung auf diese Art hat sehr gut funktioniert. Da Google Sheets cloudbasiert ist und auf die gleichzeitige Bearbeitung durch mehrere Leute ausgelegt ist, sind keine Probleme wieverlorengegangene Änderungen oder ähnliches aufgetreten und es konnte zu jeder Zeit auch ortsunabhängig auf die Backlogs zugegriffen werden. Das Design ist übersichtlich genug, damit man problemlos überblicken kann, was noch zu tun ist und wie viel schon erledigt wurde.

# 3. Analysephase (Töberg)

Da diese Projektarbeit auf einem, bereits abgeschlossenem, Projekt der Computerzeitschrift c’t basiert, wurde dieser Quellcode in der Analysephase analysiert und nachvollzogen, um auf optimale Art und Weise darauf aufsetzen zu können.

## 3.1 Der c’t-Micro-Controller

Der grundlegende Teil der c’t-Unterlagen, welcher in diesem Projekt auch nur minimal angepasst werden musste, besteht aus der „tuerschild.ino“-Datei. Dabei handelt es sich um eine Quellcodedatei, welche in der Programmiersprache C++ in der Arduino-IDE geschrieben wurde und auf dem ESP32 Controller ausgeführt wird.

Das Programm beginnt dabei mit dem einbinden von unterschiedlichen Bibliotheken und definieren von diversen Konstanten. Dabei wird auch die Größe des verwendeten e-Papers festgelegt damit die dazugehörige Bibliothek des Herstellers geladen werden kann. Daraufhin wird die begin()-Methode gestartet, welche einmalig beim Start des Controllers ausgeführt wird und ein paar grundlegende Einstellungen vornimmt. So wird die Kommunikation mit dem Webserver über TCP/IP eingerichtet und getestet.

Daraufhin beginnt die loop()-Methode, welche von dem Controller in Endlosschleife ausgeführt wird, bis dieser ausgeschaltet wird oder ein neues Programm übertragen bekommt. Da wird zu Beginn einmal das aktuelle Bild vom Webserver angefragt und ausgedruckt bevor der Controller entweder in den Tiefschlaf übergeht oder, wenn er noch nicht im produktiven Modus betrieben wird, eine vom Anwender definierte Zeit wartet. Nach dieser Wartezeit wird mit der Ausführung der loop()-Methode weitergemacht.

Zusätzlich zu diesen beiden Methoden, welche als grundlegende Methoden bei einem Großteil von Microcontrollern eingesetzt werden, besteht dieses Programm noch aus sieben, teilweise sehr kurzen, Methoden, welche für das Senden und Empfangen http-Anfragen über TCP/IP verantwortlich sind und einer Methoden, welche für das Darstellen auf dem e-Paper verantwortlich sind. Dieser Methode wird ein Array von Character-Variablen übergeben, welche die einzelnen Pixel des Displays darstellen. Bei der Auswertung dieses Arrays werden nun einzelne Bytes anstatt Bits ausgewertet und zu Beginn wird überprüft, ob das genutzte e-Paper nur zwei Farben (schwarz & weiß) oder noch eine dritte Farbe (rot oder gelb) anzeigen kann. Abhängig davon finden unterschiedliche Auswertungen statt.

Bei der ersten, zweifarbigen Auswertung wird jeder Pixel entsprechend der Stelle in jedem Byte geschaltet. So bedeutet eine 1, dass der Pixel schwarz und eine 0 das er weiß gefärbt wird. Bei der zweiten Abfrage gestaltet sich diese Abfrage nicht mehr so leicht, da eine dritte Möglichkeit eingeführt wird. So werden bei dieser Auswertung immer ein Bit und sein Vorgänger betrachtet. Wenn beide den Wert 1 besitzen, dann wird der Pixel schwarz. Wenn nur der Vorgänger den Wert 1 besitzt, so wird der Pixel rot (bzw. gelb, je nach e-Paper) geschaltet. Sollte keines der beiden Pixel den Wert 1 besitzen, so wird der Pixel weiß gefärbt. Im Gegensatz zu der anderen Auswertung werden hier auch immer zwei Pixel weitergesprungen, da immer zwei Pixel zum Übertragen der Farbinformation gebraucht werden.

Wenn der Controller nun gestartet wird, so lässt sich dieser über ein eigenes Webinterface einrichten. Diese Funktionalität kommt aus der (in Abschnitt „1.3 Projektumfeld“ bereits erwähnten) Basecamp-Bibliothek. Dabei lässt sich einstellen, welchen Server der Controller anfragen soll, welche Datei und nach welcher Zeit der Controller neue Anfragen senden soll.

## 3.2 Der c’t-Webserver

Mit dem c’t-Programm mitgeliefert werden mehrere PHP-Skripte, welche die Funktionalität des Webservers abbilden. So besitzt jedes Bild, was das Türschild theoretisch anzeigen kann, ein eigenes PHP-Skript, welche jedoch alle von einem zentralen Skript, der „index.php“-Datei, gestartet werden.

Diese Datei beginnt mit dem Laden von mehreren Schriftarten, welche von den anderen Skripten danach verwendet werden können, und dem Einstellen der Displaygröße. Diese wird dabei aus den Parametern beim Aufruf der Datei ausgelesen. Über diese Parameter wird auch der Inhalt angegeben, der angezeigt werden soll. Dieser wird, nachdem das Skript überprüft, ob der Server eine bestimmte Erweiterung namens GD, welche als Bibliothek für graphische Bearbeitung fungiert, geladen hat, in einen Dateinamen übersetzt und die zugehörige Datei wird eingebettet und ausgeführt.

Nach der Ausführung dieser Datei wird mit der „index.php“ weitergemacht. Diese übersetzt zu guter Letzt dass erzeugte PNG-Bild der anderen Datei in ein Array aus einzelnen Byte, welches dann an den Microcontroller gesendet wird, damit dieser es ausgeben kann.

Bei den Beispieldateien aus der c’t wurden die naheliegendsten Anwendungsfälle des Projektes bereits berücksichtigt. So fanden sich unter den Beispielen eine Wetteranzeige, ein Tagesplan für einen Gesprächsraum und ein Türschild. Da dieses Türschild jedoch nicht die gewünschten Anforderungen erfüllt hat, wurde in diesem Projekt ein eigenes Skript für die Darstellung erstellt, welches in Abschnitt „5.1 Bildanzeige über PHP“ genauer behandelt wird.

Ein weiteres, letztes PHP-Skript erlaubte die Anzeige eines statischen Bildes aus dem „static images“ Ordner. Dieses Bild muss dabei im PNG-Dateiformat vorhanden sein. Sollten in dem Ordner mehrere Bilder liegen, so wählt das Skript bei jeder Anfrage des Controllers zufällig eines der passenden aus und nimmt die Umwandlung vor.

## 3.3 Anforderungsanalyse

Zum Abschluss der Analysephase wurden die Anforderungen des Anwenders analysiert um die wichtigsten Anforderungen an das fertige Projekt herauszuarbeiten. Dabei konnte zum einen auf mündliche Aussagen, aber auch auf die formulierte Aufgabenstellung zurückgegriffen werden (siehe Abschnitt „1.1 Aufgabenstellung“). Insgesamt wurden folgende Anforderungen ausgearbeitet:

* Projekt aus der c’t lauffähig machen
* eigene Texte hinzufügen können (ähnlich dem Infomonitor des FB5)
* schnelle Verarbeitung der eingegebenen Texte
* umgesetzt in C oder Java (optional)

Während der Planung und der Analyse des bisherigen Projekts wurden weitere, interne Anforderungen spezifiziert. Diese sollen vor allem klare technische Anforderungen beschreiben, da die Anforderungen des Anwenders bewusst technisch unabhängig formuliert sind:

* eigenes PHP-Skript für Türschild-Anzeige schreiben
* Schnittstelle zum Webserver schreiben
* Anzeigen von Bildern und Text gemeinsam
* Programme möglichst modular entwickeln
* ausführliche Kommentierung des Quellcodes

Nachdem alle Anforderungen definiert wurden, wird im Folgenden mit der Umsetzung dieser begonnen. Wie erfolgreich diese Umsetzung verlaufen ist, wird im Abschnitt „6.1 Soll-/Ist-Vergleich“ dargestellt.

# 4. Entwurfsphase

## 4.1 Entwurf des Anzeigebilds (Muss)

Für den ersten Entwurf des Anzeigebilds wurde sich an dem bereits bestehenden, nicht-digitalen Türschild orientieren. Da dieses auch nur aus den Farben Schwarz, Weiß und Gelb besteht, war dies auch ohne Weiteres möglich. Mit dem kostenlosen GUI-Prototyping Programm „Pencil“ wurde der erste Entwurf erstellt.

Das Türschild sollte in drei Bereiche aufgeteilt werden: Etwas Platz am oberen Ende nach dem Vorbild des bestehenden Türschilds, einen Infomonitor in der Mitte und einen Bereich am unteren Ende für den Anwesenheitsstatus und Informationen über den Professor. Wie in dem bestehenden Türschild (siehe Anhang A3), wollten wir am oberen Ende die Raumnummer durch einen gelben Strich vom Rest des Türschilds getrennt haben. Neben der Raumnummer soll auch noch der Raumname ausgegeben werden. Um mehr Platz für den Infomonitor zu haben, wurden die Informationen über den Fachbereich entfernt. Am unteren Ende des E-Papers sollen Infos zum Professor an der linken Seite angezeigt werden. Der Name, die E-Mail-Adresse und die Telefonnummer wurden als erste Ideen für Professor-spezifische Informationen umgesetzt. Am anderen Ende des Kastens soll der aktuelle Anwesenheitsstatus angezeigt werden. Außerdem ist dort noch ein Bild eingefügt um das Türschild etwas anschaulicher zu gestalten. Nachdem genug Platz für den oberen und unteren Bereich geplant wurde, war der bleibende Bereich für den Infomonitor vorgesehen.

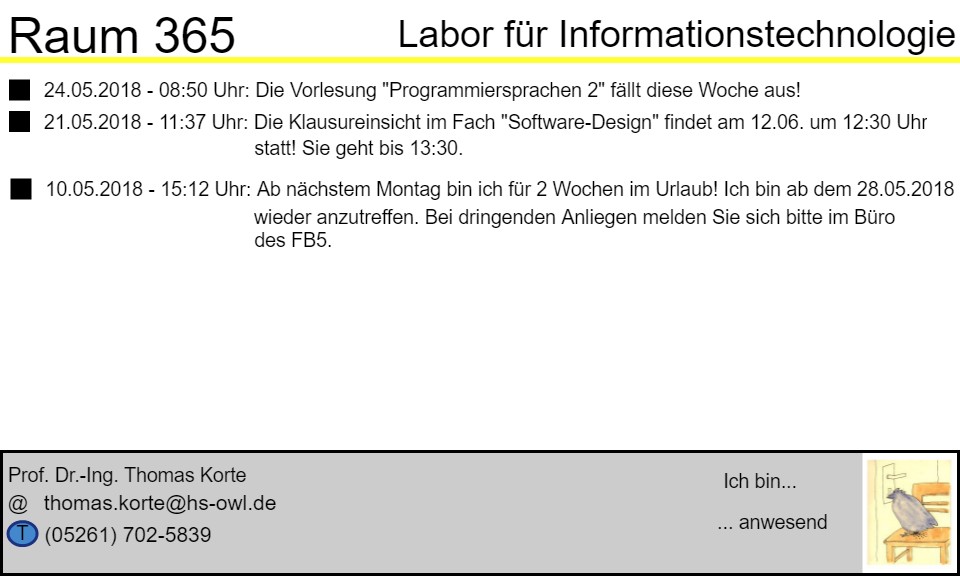


Abbildung - Entwurf des Anzeigebildes (Version 01)

Für den Entwurf wurden die Informationen von Professor Korte eingetragen und ein paar Beispielmeldungen für den Infomonitor angegeben, sowie einen beispielhaften Anwesenheitsstatus und ein Bild, welches aus seinen Vorlesungen zu Sofware-Design entnommen wurde.

## 4.2 Entwurf des Webinterface (Töberg)

Bei der Entwicklung des Webinterface war anfangs gar nicht klar, wie die eigentliche Implementierung aussehen sollte. Auch die genutzte Programmiersprache und die Software standen noch nicht fest. So wurden erst einmal die Anforderungen an die Schnittstelle zum Webserver formuliert:

* Unabhängig vom Benutzersystem
* Zugriff von mehr als einem Rechner / Gerät
* Umsetzbar in C oder Java
* Möglichkeit CSV-Dateien zu bearbeiten

Innerhalb des Teams wurden dann mehrere mögliche Umsetzungen gesammelt und ihre Vor- und Nachteile abgeglichen:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Möglichkeit | Unabhängig? | Mehrfachzugriff? | Programmiersprachen | CSV-Dateien |
| HTML Webinterface | benötigt nur Browser | Ja | HTML, JavaScript und PHP | Ja, über PHP |
| Java Desktopanwendung | benötigt JRE | Nein, nur Zugriff von einem Rechner auf lokale Dateien | Java | Ja |
| Java-Webserver (Java Servlet) | Server als auch Client benötigen JRE | Muss programmiert werden | Java | Ja |

Tabelle - Vergleich der Webserver-Schnittstellen

Wie man der obigen Tabelle entnehmen kann, bietet die Entwicklung eines Webinterfaces mit HTML, JavaScript und PHP die Umsetzung der meisten geforderten Aspekte, weshalb dieser Ansatz verfolgt wurde. Bevor jedoch mit der eigentlichen Programmierung und Implementierung begonnen wurde, wurde ein Entwurf des Interface erstellt, welcher als Vorlage für die nachfolgende Umsetzung verwendet wurde. Dieser wurde, nach ein paar ersten Tests, iterativ überarbeitet und den Anforderungen noch besser angepasst.

So besteht der erste Entwurf aus zwei Hälften, welche die zwei Fragestellungen repräsentieren, welche der Anwender beantworten kann um seine Eingabe zu tätigen. Im ersten Teil soll der Anwender seinen aktuellen Zustand auswählen. Dabei hat er per RadioButton die Auswahl zwischen fünf unterschiedlichen Zuständen, welche mal mehr, mal weniger häufig Verwendung finden. Die fünf Zuständen handelt es sich um anwesend, abwesend, krank, im Urlaub und in der Pause.



Abbildung - Entwurf des Webinterface (Version 01)

Nachdem der Anwender sich für einen dieser fünf Zustände entschieden hat, kann im unteren Bereich der Seite in einem Textfeld eine Meldung eingegeben werden, welche dann, bei der nächsten Aktualisierung auf dem Türschild angezeigt werden soll. Zur Unterstützung des Anwenders werden die zwei aktuell angezeigten Meldungen, mitsamt dem Datum und der Uhrzeit der Erstellung, unter dem Textfeld angezeigt. Diese Anordnung verhindert, dass der Nutzer manuell auf seinem Türschild nachsehen muss um zu wissen, was die aktuellen Meldungen sind. Zu guter Letzt finden sich auf dem Entwurf noch zwei reguläre Buttons, welche einmal die Änderungen verwerfen („Abbrechen“) oder die Änderungen bestätigen und die Hintergrundprozesse starten können.

Der zweite Entwurf führt kleine, aber sinnvolle Verbesserungen am bisherigen Design ein. So wird ein Dropdown-Menü hinzugefügt, welches in der oberen rechten Ecke des Bildschirms angeordnet ist und die Auswahl zwischen mehreren Professoren bzw. Anwendern des Türschilds ermöglicht. Über diesen Umstand können mehrere Anwender das gleiche Webinterface zur Einstellung ihrer jeweiligen Türschilder nutzen. Die anderen Änderungen sind mehrere kleine Textfelder, welche hinzugefügt werden um dem Anwender eine Hilfestellung bei der Bedienung des Webinterface zu bieten.



Abbildung - Entwurf des Webinterface (Version 02)

## 4.3 Entwurf des Netzwerks (Muss)

In dem ersten Entwurf des Netzwerks sollte Orts- und Geräteunabhängig zu jeder Zeit Informationen, wie der Anwesenheitsstatus und Meldungen für den Infomonitor aktualisiert werden. Die aktualisierten Meldungen sollen so gespeichert werden, dass die PHP-Skripte das aktualisierte Türschild „zeichnen“ können, wenn der ESP32 auf den Server zugreift.

Da das E-Paper direkt über ein Kabel mit dem ESP32 verbunden ist, war die erste Hürde die Frage, wie der ESP32 mit dem PHP-Server verbunden wird. Das Programm der c’t hat die Netzwerkfunktionalitäten über die „Basecamp“-Library gelöst. Hier noch mal mit Jani klären, wie viel im Bereich c’t Programmerklärung oder der Projektumgebung schon geschrieben wurde.

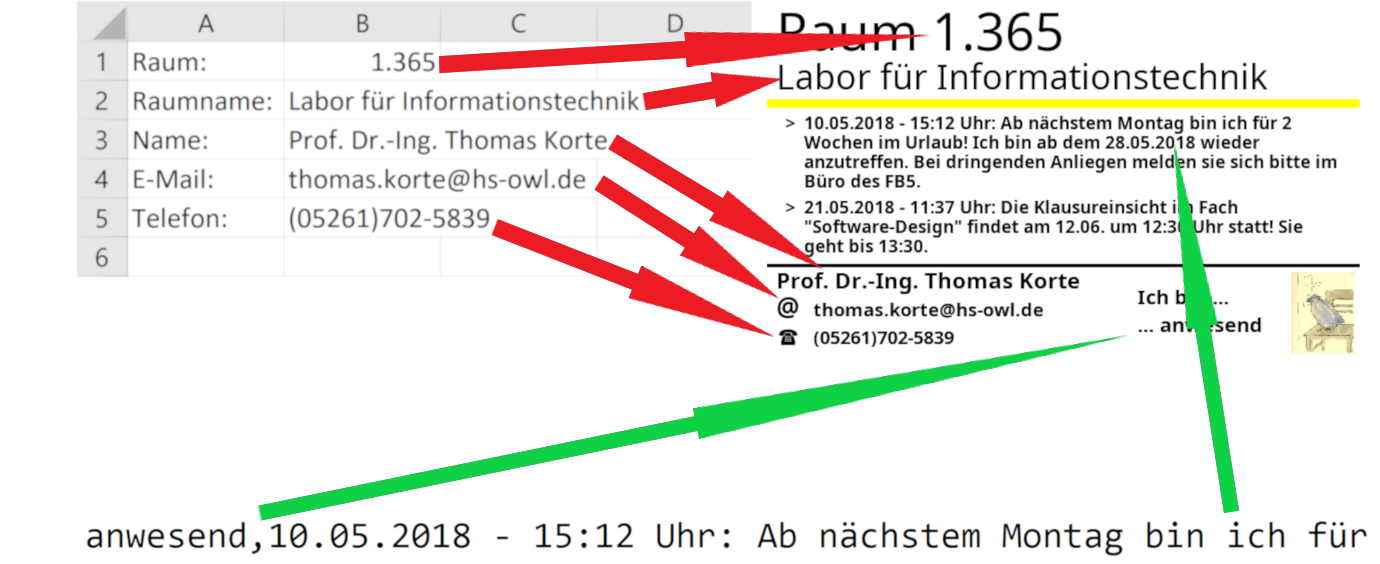


Abbildung - Veranschaulichung des Zusammenhangs von csv-Datei und Anzeige

# Da nicht über die entworfene Weboberfläche in der Basecamp Weboberfläche Werte geändert werden können, musste das Aktualisieren des Bildes unabhängig von den Basecamp Funktionalitäten designt werden. Da Basecamp auf eine spezifizierte PHP-Datei zugreift, muss diesen Pfad statisch belassen werden, jedoch kann geändert werden, was in der Datei steht. Da die Anzeigebilder durch PHP-Skripte „gezeichnet“ werden und diese Skripte wiederrum auf CSV-Dateien zugreife, soll die Weboberfläche die CSV-Dateien verändern, so dass beim neuen „zeichnen“ die aktualisierten Werte genutzt werden. 5. Implementierungsphase

## 5.1 Bildanzeige über PHP (Muss)

Das Bild, welches auf dem Bildschirm angezeigt werden soll, wird über zwei PHP-Skripte erstellt. Das Index Skript wird direkt vom Controller aufgerufen. Durch Spezifikation in der angegebenen Serveradresse wird das zweite Skript angegeben, welches von der index.php aufgerufen wird um das auszugebende Bild zu „zeichnen“.

### 5.1.1 index.php

Die index.php ist die Schnittstelle zwischen dem ESP32 und dem Anzeigeskript. Es ruft ein anderes Skript auf und gibt dessen Inhalt, in einer für den Controller verständlichen Form, zurück.

Wenn versucht wird, das Skript aufzurufen, müssen einige Parameter übergeben werden, damit das das Skript richtig funktionieren kann.

* Debug: Ein boolescher Wert, der festlegt, wie das Bild zurückgegeben werden soll:
  + True: Gibt das Bild als Bilddatei zurück. Lesbar von Webbrowsern und daher gut um zu testen, wie das gezeichnete Bild aussehen wird.
  + False: Gibt das Bild als Byte-String zurück. Lesbar von dem Controller und daher nötig um das Bild auf dem E-Paper anzeigen zu können.
* Display: Bei diesem Wert wird die Größe des Displays in Inches und die Information, ob er Farben anzeigen kann oder nicht übergeben. Die hier möglichen Werte werden in einem Array definiert und betragen 7.5, 7.5bwr, 4.2, 4.2bwr, 2.9 & 1.5.
  + Die Anzeigeskripte sind nur für die Bildschirmarten 7.5 und 7.5bwr ausgelegt und skalieren schlecht bis gar nicht auf andere Displaygrößen.
* Content: Name des Anzeigeskriptes. Hier muss das Anzeigeskript angegeben werden, welches ausgegeben werden soll. Die Datei muss die Endung .php besitzen und im Ordner „contents“ liegen, damit es akzeptiert wird.
* Professor: Auswahl welche Daten über das Anzeigeskript ausgegeben werden. Genauere Erläuterung folgt im nächsten Kapitel.

Wenn keine Wert für Debug eingeben wird, dann wird dieser automatisch auf false gesetzt. Display, Content und Professor müssen jedoch eingegeben werden, da das Skript ansonsten frühzeitig abbricht und eine Fehlermeldung ausgibt. Außerdem überprüft es, ob alle wichtigen Bibliotheken installiert sind und bricht die Bearbeitung ab, falls sie es nicht sind. Im nächsten Schritt werden, basierend auf den Parametern, einige Variablen definiert, die auch für das Anzeigeskript relevant sind. Diese Variablen sind die Dimensionen des Displays in Pixeln, eine Bildressource mit den Ausmaßen des Displays, die zuerst mit der Farbe Weiß gefüllt wird, und ein boolescher Wert, der angibt ob der Bildschirm rot, oder in diesem Projekt gelb, anzeigen kann oder nicht, welcher sowohl im Anzeigeskript als auch bei der Übergabe der Daten an den ESP32 benutzt wird. Neben den Variablen, die von den Paramatern abhängig sind, werden auch noch parameterunabhängige Variablen festgelegt. Dazu gehören die Farbcodes für Schwarz, Rot und, zu Testzwecken, gelb. Außerdem werden die Pfade zu den verschiedenen unterstützten Schriftarten in einem Array definiert.

Als nächstes wird das Bild durch das Anzeigeskript gezeichnet. Durch den „include“-Befehl kann eine andere PHP-Datei eingebunden werden. Die eingebundene Datei ist das über den content-Parameter angegebene Skript. Da das andere Skript innerhalb des index-Skriptes ausgeführt wird, teilen sich die Skripte die Variablen. Das andere Skript kann also in die gleiche Bildressource, die vorher bereits mit der Größe des Bildschirms initialisiert wurde, hineinschreiben.

Nachdem das Anzeigeskript, welches im nächsten Kapitel genauer erklärt wird, nun die geteilte Bildressource so beschrieben hat, dass das Türschild darstellt wird, liegt es nun an dem Index-Skript diese Ressource angemessen weiterzugeben. Je nach Einstellung der Debug-Variable wird die Ressource anders übergeben. Wenn Debug auf „True“ gesetzt ist, muss die Ressource für die Wiedergabe über einen Webbrowser aufbereitet und letztendlich angezeigt werden. Sollte Debug jedoch auf „False“ eingestellt sein, wird die Funktion rawImage() aufgerufen um die Ressource in eine Form zu bringen, die der Controller verarbeiten kann, um das gezeichnete Bild auf dem E-Paper Display anzeigen zu können. Diese Funktion überprüft für jeden Pixel einzeln die vorhandene Farbe und versucht, den vorhandenen Wert in Schwarz, Weiß oder Rot einzuteilen. So würde er also ein dunkles Grau als Schwarz ausgeben und ein helles grau als Weiß. Für ein schwarz-weißes Display steht ein Bit stellvertretend für Schwarz oder Weiß, wobei eine 1 für Schwarz und eine 0 für Weiß steht. Für ein mehrfarbiges Display werden zwei Bit für einen Pixel benötigt, wobei „11“ für Schwarz, „00“ für Weiß und „01“ für Rot steht. Die einzelnen Bits werden noch in Bytes verpackt und am Ende des Skriptes wird der Byte-String zurückgegeben.

### 5.1.2 türschild.php

Das Türschild-Skript dient für unsere Zwecke als das Anzeigeskript, welches von dem Index-Skript eingebunden wird. In dem Index-Skript wurde bereits eine Bildressource initialisiert, die jetzt von dem Anzeigeskript über Funktionen aus der GD-Library „bemalt“ werden soll. Die GD-Library ist eine Open-Source-Programmbibliothek, zur dynamischen Erzeugung und Manipulation von Grafiken.

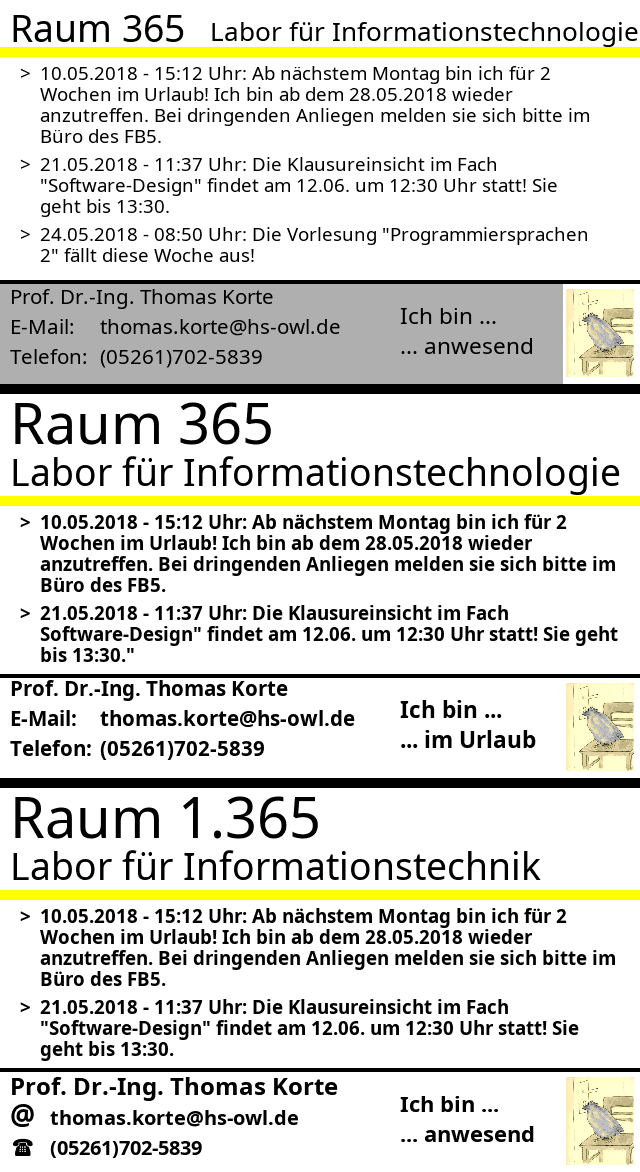
Um für eine gute Erweiterbarkeit des Türschilds zu sorgen, werden fast alle der angezeigten Werte aus externen CSV-Dateien eingelesen. In der Datei „Professor.csv“ stehen Werte, die voraussichtlich für längere Zeit konstant bleiben, wie die Raumnummer und Informationen über den Professor, welchem das Büro gehört. In der Datei „Professor\_status.csv“ hingegen werden die Werte abgelegt, die häufiger durch die Weboberfläche geändert werden können, wie der Anwesenheitsstatus und die Infomonitormeldungen. Diese Daten werden am Anfang des Skriptes eingelesen und in zwei verschiedene Arrays geschrieben.

Außerdem soll es möglich sein, zu den Informationen eines anderen Professors zu wechseln, wenn diese bereits eingetragen wurden. Für diesen Zweck gibt es den Parameter „professor“, in welchem man den Namen des Professors eingeben kann, dessen Türschild man anzeigen möchte. Die csv-Dateien müssen Namenskonventionen folgen, damit dieser Parameter richtig funktionieren kann. Die Datei, die im vorherigen Kapitel „professor.csv“ genannt wurde, muss den gleichen Namen haben wie der Parameter und die „status.csv“ Muss einen Namen der Form „Parameter\_status.csv“ haben.

Neben dem Parameter für den Professor ist auch der im vorherigen Kapitel erwähnte Parameter des Displays wichtig, um genau zu sein, die Variable $hasred, welche aussagt, ob der Display Farben oder nur schwarz anzeigen kann. Die Variable wurde bereits im Index-Skript definiert und ist abhängig von dem Parameter Display. Wenn man also einen Bildschirm als Parameter übergibt, der keine Farben anzeigen kann, soll auch in der Webansicht die Farbe gelb durch schwarz ersetzt werden.

Wie schon erwähnt kennt das E-Paper Display softwareseitig nur die Farbe Rot und möchte diese übergeben bekommen. Da aber die endgültige Anzeigefarbe des Displays gelb ist, wäre es schön, wenn die Webansicht gelb anzeigt, obwohl der Controller rot übergeben bekommt. Dafür benutzen wir die bereits definierte Variable „debug“, welche, wie im letzten Kapitel erläutert, angibt, ob die fertige Bilddatei in einem Webbrowser ausgegeben werden soll, oder ob sie für den Controller in Pixelarrays aufbereitet werden soll. In dem Skript wird nur die Variable $red benutz, die je nach Wert des debug-Parameters entweder den Farbcode der Farbe Rot enthält, wenn die Bildressource an den Controller übergeben werden soll, oder den Farbcode der Farbe Gelb, falls die Webansicht benutzt werden soll.

Abbildung 6 - Entwicklung des Anzeigebildes über den Projektzeitraum



Unser Türschild besteht aus drei Bereichen, wie bereits in dem Kapitel „4.1 Entwurf des Anzeigebilds“ beschrieben. Im oberen Teil soll die Raumnummer und der Name des Büros stehen, welche daraufhin mit einer gelben Linie vom Rest des Schildes getrennt werden. Zuerst werden die Texte mit der Funktion „imagettftext“ generiert, welcheeinen Text auf eine Imageressource schreibt. Dazu müssen die x und y-Koordinaten angegeben werden, sowie den Pfad zu einer Schriftart, die Schriftfarbe und eine Schriftgröße. Sowohl die Schriftgröße und die y-Koordinate werden über Variablen realisiert und die Schriftart kann über ein Array, welches in dem Index-Skript initialisiert wurde, ausgewählt werden Die x-Koordinate jedoch hat keine eigene Variable und muss manuell für jeden Text eingegeben werden. Das liegt daran, dass die Variable für die y-Koordinate hochgezählt werden kann, da das Skript die Bildressource von oben nach unten berabeiten wird. Daher kann eine Variable für die Texthöhe während des Skriptes hochgezählt werden, während die x-Koordinaten für jeden Text unabhängig von der Stelle im Skript sind. Die beiden ersten Werte des Arrays, welches aus der Professor.csv erzeugt wurde, sind die Raumnummer und der Raumname, die beide als Text auf die Bildressource gezeichnet werden. Für die gelbe Trennlinie wird die Funktion „imageline“ genutzt. Auch diese Funktion „zeichnet“ auf die Bildressource aus dem Index-Skript. Diese Funktion zeichnet jedoch keinen Text, sondern eine Linie. Für diese Funktion müssen der Start und der Endpunkt der Linie festgelegt werden. Damit also eine horizontale Linie über den gesamten Bildschirm angezeigt wird, müssen die y-Koordinaten am Startpunkt und Endpunkt gleichbleiben. Die y-Koordinaten des Startpunktes muss 0 sein und für den Endpunkt der Linie bietet sich die bereits definierte Variable über die Bildschirmweite in Pixeln an. Um festzulegen, wie Fett die Linie sein soll muss vorher die Funktion „imagesetthickness“ aufgegeben werden, welcher man eine Zahl übergibt, die die Dicke alle weiteren Linien, die durch GD-Library-Funktionen gezeichnet werden, angibt. Dabei ist zu beachten, dass wenn wir die Zahl vier übergeben die Linie acht Pixel breit ist, da sich die Dicke in beide Seiten erstreckt.

Der Bereich in der Mitte ist reserviert für den Infomonitor. Dieser war die größte Herausforderung in der Implementierung des Anzeigeskriptes. Schon in der ersten Implementierung wurde klar, dass die Ausgabe über „imagettftext“ keine automatischen Zeilenumbrüche unterstützt, da sie nur Text auf eine Bildressource „zeichnet“ ohne dessen Aufmaße zu kennen. Dadurch wurde der Text einfach in einer Zeile gedruckt, bis er am rechten Bildschirmrand abgeschnitten wurde. Zeilenumbrüche mussten also manuell in das Anzeigeskript implementiert werden. Da die Länge eines Textes nicht von der Anzahl Buchstaben abhängig ist, weil Buchstaben unterschiedliche Breiten haben können, musste die Länge des Textes dynamisch bestimmt werden. Die GD-Library bietet mit „imagettfbbox“ eine Funktion an, die genau das ermöglicht. Wenn man ihr den Text zusammen mit der benutzen Schriftart und Größe übergibt, bekommt man ein Array zurückgegeben, welches die Eckpunkte einer box angibt, welches den Text umgeben könnte. Die Differenz zwischen einem Punkt am rechten Rand und einem Punkt am linken Rand ergibt die Länge des Textes in Pixeln, die wir mit der Variable $displayWidth, welche die Breite des Displays in Pixeln angibt, vergleichen können. Um zu wissen an welcher Stelle Zeilenumbrüche eingefügt werden müssen muss die Textlänge jedoch nach jedem einzelnen Wort überprüft werden. Dazu muss im ersten Schritt eine Schleife gestartet werden, die über die Anzahl der Nachrichten iteriert, damit jede Nachricht einzeln bearbeitet und ausgegeben werden kann. Die Anzahl der Nachrichten ist die Größe des Arrays welche aus der Status-Datei ausgelesen wurde, abzüglich des Anwesenheitsstatus. Dazu müssen die Nachrichten zuerst durch die „explode“-Funktion in Arrays aus Wörtern aufgebrochen werden. Nun wird mit einer Zählschleife über die Anzahl an Wörtern in dem Array iteriert und in jedem Schritt wird ein weiteres Wort dem Teststring $zeile hinzugefügt. Mit „imagettfbbox“ wird die Länge des Strings zu diesem Zeitpunkt, also immer nach dem Hinzufügen eines neuen Wortes zu dem Teststring, überprüft und mit der Bildschirmweite abgeglichen. Falls die Breitenbegrenzung noch nicht überschritten wurde, wird das Wort was zuletzt dem $zeile hinzugefügt wurde auch dem String $text hinzugefügt, der später ausgegeben werden soll. Falls die Breitenbegrenzung überschritten wurde, wird das letzte Wort zuerst zurückgehalten. In dem String $text steht nun genau so viel Text, wie in eine Zeile passt, daher kann dieser ausgegeben werden. Da jedoch eine Nachricht nur als Ganzes ausgegeben werden soll und noch nicht bekannt ist, ob die Nachricht nicht vielleicht noch die Höhenbegrenzung überschreitet, muss der $text String zuerst noch einmal zwischengespeichert werden. Dies geschieht im Array $nachricht in dem jedes Element eine Zeile einer Nachricht sein soll. Nachdem die Schleife abgearbeitet ist und die Nachricht zeilenweise in ein Array verpackt wurde, wird einmal überprüft, ob die Nachricht die Höhenbeschränkung des Infomonitors überschreitet. Falls sie zu groß ist und in den dritten Teil der Anzeige hineinschreiben würde, wird diese Nachricht nicht mehr ausgegeben und auch keine weiteren. Die Schleife über die Anzahl der Nachrichten wird daraufhin mit einem break-Befehl vorzeitig beendet. Falls die Höhenbegrenzung nicht überschritten wurde, kann die Nachricht ganz normal ausgegeben werden. Dazu wird zuerst das Zeichen „>“ über „imagettftext“ gedruckt werden, um den Beginn einer neuen Nachricht zu kennzeichnen. Daraufhin werden alle Zeilen aus dem $nachricht Array untereinander mit „imagettftext“ ausgegeben. Danach springt das Skript wieder zum Anfang der äußeren Zählschleife, die über die Anzahl der Nachrichten iteriert, falls noch weitere Nachrichten vorliegen. Dabei behält es jedoch die Variable, welche die y-Koordinate angibt, bei um testen zu können, ob die nächste Nachricht die Höhenbeschränkung überschreitet, da man ab der zweiten Nachricht einen Startpunkt der Nachricht hat, die von der Größe der vorherigen Nachricht abhängt. Die weiteren Nachrichten werden dabei auf dieselbe Art bearbeitet, wie die erste. Sobald es keine Nachrichten mehr gibt, oder die Schleife aufgrund der Höhenbegrenzung frühzeitig abbricht, ist die Bearbeitung des Infomonitors beendet und das Skript kann zum nächsten Teil der Anzeige übergehen.

Am unteren Ende des Bildschirms soll eine Box angezeigt werden, in der der Anwesenheitsstatus und einige Informationen zum Professor stehen. Um die Box von dem Infomonitor abzugrenzen, soll das Skript zuerst eine Linie mit „imageline“ zeichnen, wobei auch hier vorher mit „imagesetthickness“ die Dicke der Linie angegeben werden. Zuerst wird der Name des Professors ausgegeben, welchem dieses Büro gehört. Diese Information kann man aus der Einrichtungs-Datei lesen. Direkt darunter sollen die Telefonnummer und die E-Mailadresse des Professors angegeben werden. Vor diesen Informationen soll ein einzelnes Zeichen gedruckt werden, das anzeigen soll, ob es sich um die E-Mailadresse oder die Telefonnummer handelt. Für die E-Mailadresse wird ein einzelnes „@“ mit einer leicht erhöhten Schriftgröße über „imagettftext“ gedruckt werden und für die Telefonnummer wird durch das Telefonzeichen aus der Schriftart „NotoEmoji-Regular“ dargestellt. Jeweils hinter diesen Zeichen sollen die dazugehörigen Werte geschrieben werden, die auch aus der zugehörigen csv-Datei eingelesen werden. Auch in der Box am unteren Ende soll der Anwesenheitsstatus stehen für welchen wir erst den festen String „Ich bin …“ mit Hilfe von „imagettftext“ drucken und darunter den Anwesenheitsstatus zeichnen, welcher das erste Element in dem status-Array ist. Als letztes angezeigtes Element gibt es noch ein Bild, welches rechts-unten in der Ecke angezeigt werden soll. Der Pfad des Bildes ist abhängig von dem Professor-Parameter. Um ein Bild anzeigen zu können muss ein Bild mit dem Namen des Parameters im Ordner Bilder auf dem Webserver liegen. An dieser Stelle wird auch noch unterschieden ob der Bildschirm Farbe angeben kann. Falls er es kann muss noch „\_bwr“ an den Namen der Bilddatei angehangen werden. So gibt es ein Bild für farbfähige Bildschirme und ein schwarz-weißes Bild für Bildschirme, die keine Farben anzeigen können. Damit das Bild richtig angezeigt werden kann, müssen die Dimensionen so geändert werden, dass es in die Ecke passt, dabei kann es jedoch das ursprüngliches Seitenverhältnis des Bildes auf 3:4 gestreckt oder gestaucht wird. Das Bild kann vom Typ PNG oder vom Typ JPEG sein, falls kein Bild angegeben ist oder die Datei von einem nicht unterstützen Typ sein sollte, wird kein Bild angezeigt.

Am Ende des Anzeigeskripts sollte die Bildressource so bemalt worden sein, dass sie das Türschild darstellt.

## 5.2 Das Webinterface (Töberg)

Parallel zu der Entwicklung des PHP-Skriptes, welches sich mit der Anzeige des Türschilds auf dem e-Paper befasst, wurde das Webinterface entwickelt. Dabei besteht das Webinterface aus mehreren Abschnitten und Dateien, welche dem Nutzer ermöglichen sollen, über eine Webschnittstelle seinen Zustand einzustellen und Meldungen hinzuzufügen. Der Austausch zwischen diesen unterschiedlichen Dateien und den Skripten der Anzeige erfolgt dabei über eine CSV-Datei, welche beschrieben und gelesen werden kann. (siehe Anhang?)

Bei der Entwicklung des Webinterface wurde mit dem Erstellen eines HTML-Skriptes begonnen, welches die Darstellung der verschiedenen graphischen Elemente ermöglicht. Innerhalb dieses HTML-Skriptes findet sich ein Abschnitt in welchem die dahinterliegende Funktionalität mit Hilfe von JavaScript entwickelt wurde. Beide Aspekte, welche im Folgenden noch genauer beschrieben werden, werden jedoch auf Clientseite ausgeführt, weshalb es einen Gegenpart auf der Serverseite geben muss, welcher Anfragen bearbeiten und beantworten kann. Diese Funktionalität wurde mit zwei unterschiedlichen PHP-Skripten realisiert, welche in Abschnitt „5.2.2 PHP auf Server-Seite“ genauer beschrieben werden.

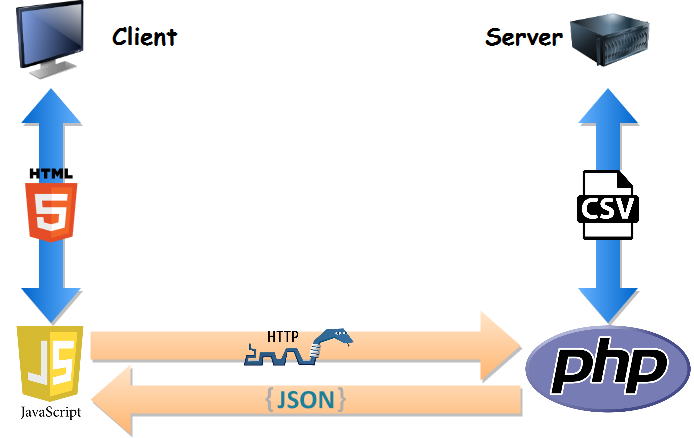


Abbildung - Kommunikation zwischen Server und Client

### 5.2.1 HTML und JavaScript auf Client-Seite

Im Folgenden wird die clientseitige Implementierung dargestellt werden. So beginnt das Dokument mit ein paar allgemeinen Festlegungen (z.B.: Sprache des Dokuments ist deutsch, Zeichensatz ist UTF-8, usw.) gefolgt von der Definition der unterschiedlichen Interaktionselemente mit ihren jeweiligen Attributen. Besonders hervorheben möchte ich dabei die Attribute „id“ und „style“, welche bei vielen (jedoch nicht allen) Elementen verwendet werden. Über das id-Attribut wird dem Element eine eindeutige Identifikation zugewiesen, welche das Auffinden und Bearbeiten des Elements über JavaScript stark vereinfacht. Mit Hilfe des style-Attribut können unterschiedliche Einstellungen bezüglich des Designs des Elements gemacht werden. Dabei wird die gleiche Schreibweise wie bei CSS-Dateien verwendet, dies jedoch nicht über den expliziten Import einer passenden Datei sondern über eine eingebettete Art und Weise.

Nachdem alle Elemente definiert wurden, werden bestimmte Elemente mit dazugehörigen JavaScript-Funktionen verknüpft, sodass die Funktion bei einem bestimmten Ereignis ausgeführt wird. Die drei Ereignisse auf die reagiert wird sind der Klick auf jeweils einen der Buttons, der Wechsel des Professors über das Dropdown-Menü und das Aktualisieren der Website.

Die Funktion mit dem geringsten Umfang wird aufgerufen, wenn entweder der Professor gewechselt wird oder der „Abbrechen“-Button betätigt wird. Die sogenannte „reloadTheWholePage()“-Funktion tut was der Name verspricht und lädt einmal die Seite neu, was automatisch die „reloadAndLoadData()“-Funktion ausführt. Diese Funktion erstellt eine neue http-Request an den Webserver, um genau zu sein an die „GetDataFromCSV.php“-Datei. In dieser Anfrage sendet der Client die Bezeichnung des aktuell ausgewählten Professors mit, damit der Webserver weiß welche Daten er laden & zurücksenden muss. Nachdem die Anfrage gesendet wurde, wartet die Funktion auf eine Antwort (Anhang A10, ab Zeile 57) und wertet, bei Erhalt der Antwort, die Antwort im JSON-Format aus. Die Antwort enthält dabei den aktuell gespeicherten Zustand und die zwei aktuellsten Meldungen des Professors. Diese werden direkt auf den Inhalt der zugehörigen Elemente der Weboberfläche geschrieben. Dadurch wird garantiert, dass nach jeder Aktualisierung der Seite die aktuellsten Daten angezeigt werden.

Die letzte Funktion („confirmAndStartSending()“) befasst sich mit dem Senden der neuen Daten an den Webserver. Diese Funktion wird immer ausgeführt, wenn der „Bestätigen“-Button geklickt wird. Die Funktion beginnt damit, die unterschiedlichen Eingaben des Anwenders auszuwerten. Dafür werden die oben bereits angesprochenen eindeutigen Bezeichner der Elemente verwendet, um mit Hilfe des *document.getElementById("ID").innerHTML*-Aufrufs den Inhalt des Elements zu bekommen. Mit dem Inhalt ist in diesem Kontext der Text angezeigt, welchen das Element anzeigt. Innerhalb dieser Funktion werden auch die unterschiedlichen RadioButtons innerhalb einer Zählschleife ausgewertet, um zu bestimmen, welcher der fünf Buttons ausgewählt ist. Nachdem alle relevanten Informationen ausgewertet wurden, werden diese per Methodenaufruf an die „sendToPHPAndSave()“-Methode übertragen, welche ebenfalls eine http-Request formuliert. In diesem Fall wird jedoch die „saveDataToCSV.php“-Datei angesprochen und es werden auch mehr Parameter übergeben. Diese werden in der Funktion vor dem Senden zu einem String mit passenden Stichworten konkateniert, was die Auswertung durch das PHP-Skript erleichtert. Danach wird dieser String gesendet und die Antwort des Servers auf der Konsole ausgegeben.

### 5.2.2 PHP auf Server-Seite

Nachdem die clientseitige Kommunikation beschrieben wurde, werden im Folgenden die beiden, bereits angesprochenen, PHP-Dateien behandelt, welche den Zugriff auf die CSV-Datei auf dem Webserver steuern. Beide Dateien besitzen dabei gewisse Gemeinsamkeiten. Zum einen nutzen beide einen vordefinierten Header, welcher die aktuellen Informationen zum Dokument bündelt. Des Weiteren legen beide über den Befehl „mb\_internal\_encoding("UTF-8")“ ihre interne Kodierung auf UTF-8 fest, was das Verarbeiten von Sonderzeichen in den Strings, welche per http-Anfrage gesendet werden, ermöglicht. Zu guter Letzt teilen sich beide Dateien eine Mehrfachverzweigung, welche, abhängig vom gesendeten Professor, den Dateinamen festlegt, aus welchem gelesen werden soll. Dadurch wird garantiert, dass mehrere Professoren unabhängig voneinander ihre Meldungen bearbeiten können.

Die „getDataFromCSV.php“-Datei verwaltet den Lesezugriff auf die CSV-Datei. Dabei wird, nachdem der passende Dateiname festgelegt ist, aus dieser Datei alles ausgelesen und in ein Array geschrieben. Die Elemente dieses Arrays werden dann ins JSON-Format überführt und als Antwort an den Client gesendet.

Wenn der Client jedoch Daten sendet, welche in einer Datei gespeichert werden sollen, so wird die „saveDataToCSV.php“-Datei angefragt, welche mit den oben bereits beschriebenen Formatierungen und der Dateinamen-Abzweigung beginnt. Nachdem diese Aspekte festgelegt wurden, werden die gesendeten Daten des Clients ausgelesen. Diese werden bei einer http-Anfrage in ein Array auf dem Server geschrieben, welches dieser einfach auslesen kann. Lediglich die Formatierung muss bei einigen Werten angepasst werden, da die alten Meldungen, welche mitgeschickt werden, ein vorangestelltes Datum mitsamt Uhrzeit besitzen. Dieses wird am Anfang entfernt um spätere Vergleiche zu erleichtern. Daraufhin wird in einer verschachtelten bedingten Anweisung festgelegt, welche Daten in die Datei geschrieben werden sollen. Dabei werden neue, vom Anwender geschriebene Meldungen nicht gespeichert, wenn Sie entweder leer sind, aus der Standard-Nachricht bestehen oder die aktuell angezeigte Meldung sind. Nachdem diese Festlegung passiert, werden die Daten über einen einfachen Methodenaufruf in die passende CSV-Datei geschrieben und über den „echo“-Befehl wird eine Antwort an den Client gesendet, welche kurz zusammenfasst, was geschrieben wurde.

# 6. Fazit

## 6.1 Soll-/Ist-Vergleich (Töberg)

Bevor im nächsten Abschnitt beschrieben wird, welche möglichen Erweiterungen der aktuelle Stand dieses Projekts bietet, wird im Folgenden darauf eingegangen ob die in Abschnitt „3.3 Anforderungsanalyse“ beschriebenen Anforderungen umgesetzt werden konnten.

So wurden die grundsätzlichen Anforderungen des Anwenders erfolgreich umgesetzt. Das Türschild erlaubt das flexible Hinzufügen von Texten, welche auf dem Display angezeigt werden. Lediglich die Geschwindigkeit bietet Verbesserungspotential, da das Türschild aktuell nicht auf Veränderungen durch den Anwender reagiert, sondern nach jeder Aktualisierung in einen Tiefschlaf übergeht. Sobald es aus diesem erwacht, überprüft es, ob andere Meldungen angezeigt werden müssen. Ist dies nicht der Fall, so ändert sich nichts und es wird wieder in den Tiefschlaf gegangen. Da die Tiefschlafzeit aktuell zehn Minuten beträgt, werden im schlimmsten Fall neue Meldungen erst nach zehn Minuten angezeigt.

Leider konnte die optionale Anforderung das Projekt in der Programmiersprache Java oder C umzusetzen nicht umgesetzt werden. Das lag zum einen daran, dass der C/C++-Teil des Projekts bereits abgeschlossen war und keiner Erweiterung bedurfte, aber auch daran, dass es bessere Alternativen gab (wie in Abschnitt „4.2 Entwurf des Webinterface“ dargestellt).

Die Anforderungen, die das Projektteam ergänzend formuliert hat, konnten überzeugend umgesetzt werden. Insbesondere auf die Kommentierung und die modulare Entwicklung wurde geachtet. Durch die modulare Entwicklung ist ein Grundstein für die Weiterentwicklung des Projektergebnisses eventuell sogar durch ein anderes Projektteam gelegt.

## 6.2 Ausblick (Muss)

Bei der Implementierung des Anzeigeskriptes wurde viel Wert auf die Erweiterbarkeit gelegt. Zusammen mit dem guten Grundgerüst, welches von der c’t übernommen werden konnte, ist es sehr leicht das Türschild mit den Werten eines anderen Professors zu füllen. Alles was getan werden muss um dies zu ermöglichen, ist das Anlegen von zwei csv-Dateien nach dem Vorbild der bereits vorhandenen und gegebenenfalls das Hinzufügen eines Bildes an die richtige Stelle.

Die erste csv-Datei, die angelegt werden muss, ist die Datei, die Konstanten über den Professor und den Raum beinhalten soll. Diese muss den Namen tragen, welcher später als Parameter übergeben werden muss um sich die Daten anzeigen zu lassen. Um diese Dateien auseinanderhalten zu können, haben wir in unseren Beispielen den Namen des Professors als Parameter benutzt. Die csv-Datei ist benutzerfreundlich aufgebaut, da diese von Benutzern geändert werden soll. Alle Werte der Tabelle müssen gefüllt sein, damit das das Anzeigeskript funktioniert.

|  |  |  |  |
| --- | --- | --- | --- |
| professor.csv | Eintrag | korte.csv | hausdoerfer.csv |
| Raum: | Nummer des Raumes | 1.365 | 1.328 |
| Raumname: | Name oder Titel des Raumes | Labor für Informationstechnik | Echtzeitsysteme |
| Name: | Name des Professors | Prof. Dr.-Ing. Thomas Korte | Prof. Dr.-Ing. Rolf Hausdörfer |
| E-Mail: | E-Mail-Adresse des Professors | thomas.korte@hs-owl.de | rolf.hausdoerfer@hs-owl.de |
| Telefon | Telefonnummer des Professors | (05261)702-5839 | (05261) 702-3562 |

Tabelle - Aufbau der professor.csv mit zwei Beispieleintragungen

Die zweite csv-Datei ist die Datei, die regelmäßig von der Weboberfläche geändert wird. Da unterschiedliche Professoren auch unterschiedliche Infomonitormeldungen haben können, muss das Skript neben einer anderen professor.csv auch eine andere status.csv benutzen. Auch dieser lässt sich durch den Parameter „professor“ auswählen, wenn die csv-Datei nach dem Muster professorname\_status.csv benannt wurde. Die momentane Weboberfläche kann auch zwischen mehreren verschiedenen Professoren differenzieren und in die zugehörigen csv-Dateien schreiben.

Auch das Bild, welches rechts-unten in der Ecke angezeigt wird kann von Professor zu Professor unterschiedlich sein. Auch hier müssen die Bilder einen Namen tragen, der später durch den Parameter zugeordnet werden kann, nach dem Muster „*parameter*.png“ und „*parameter*\_bwr.png“. Die Endung muss dabei nicht zwangsläufig „.png“ sein, da „.jpeg“-Dateien auch unterstützt werden

Durch Professor-abhängige csv-Dateien, und die Auswahl dieser über Parameter ist gewährleistet, dass mehrere Professoren das gleiche Skript und den gleichen Server benutzen können, wenn alle ihren eigenen Controller und ihr eigenes E-Paper Display besitzen. Es muss nur der richtige Parameter bei der Ersteinrichtung in der Basecamp Weboberfläche übergeben werden.

Da die drei Bereiche des Türschildes im Skript komplett unabhängig voneinander sind, lassen sich Änderungen, die nur einen der drei Teile betreffen gut durchführen, ohne die anderen zu beeinflussen. Falls ein anderer Professor anstelle des Infomonitors beispielsweise eine Art Kalender haben möchte. Könnte man den Infomonitor einfach ersetzen, ohne dass es einen Einfluss auf den oberen oder den unteren Abschnitt hat.

# 7. Quellen

1) Modulhandbuch Fachbereich 5

Letzter Aufruf am: 28.06.2018)

[Modulhandbuch FB5](https://www.hs-owl.de/fb5/fileadmin/download/pdf/Studiengaenge/Modulhandbuch/Modulhandbuch_Ba-E-TI_4-10.pdf)

2) Doorsign EPD Repository by jamct on GitHub

Letzter Aufruf am: 28.06.2018)

[DoorsignEPD](https://github.com/jamct/DoorsignEPD)

3) Basecamp by merlinschuhmacher on GitHub

Letzter Aufruf am: 29.06.2018)

[Basecamp](https://github.com/merlinschumacher/Basecamp)

4) arduino-esp32 by espressif

Letzter Aufruf am: 30.06.2018)

[Espressiff](https://github.com/espressif/arduino-esp32)

# A Anhang

## A1 – Netzwerkdiagramme

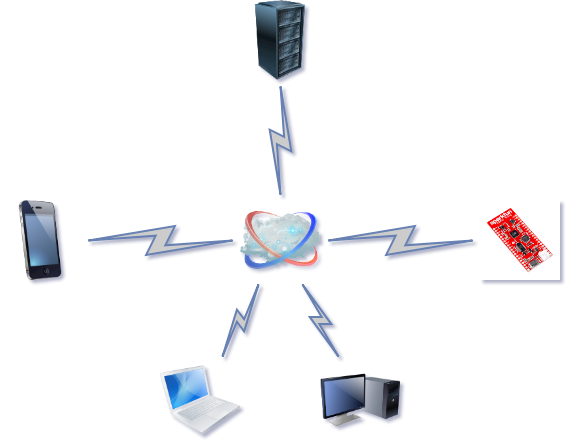


Abbildung - Entwurf des Netzwerks (Version 01)



Abbildung - Entwurf des Netzwerks (Version 02)

## A2 – Das Webinterface



Abbildung - Das aktuelle Webinterface (Version 02)

## A3 – Fotos der Türschilder (vorher)

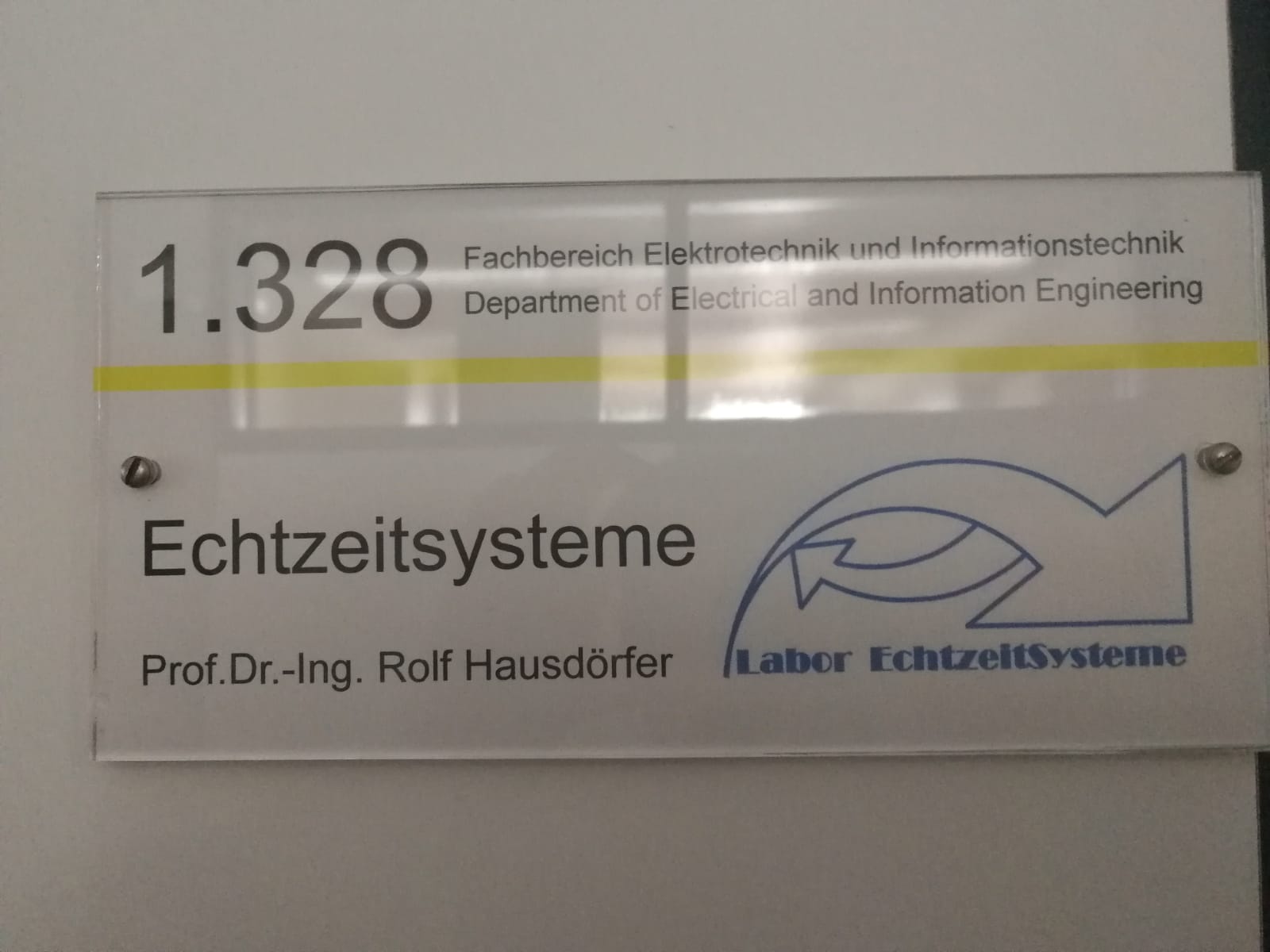


Abbildung - Türschild von Prof. Hausdörfer (vorher)

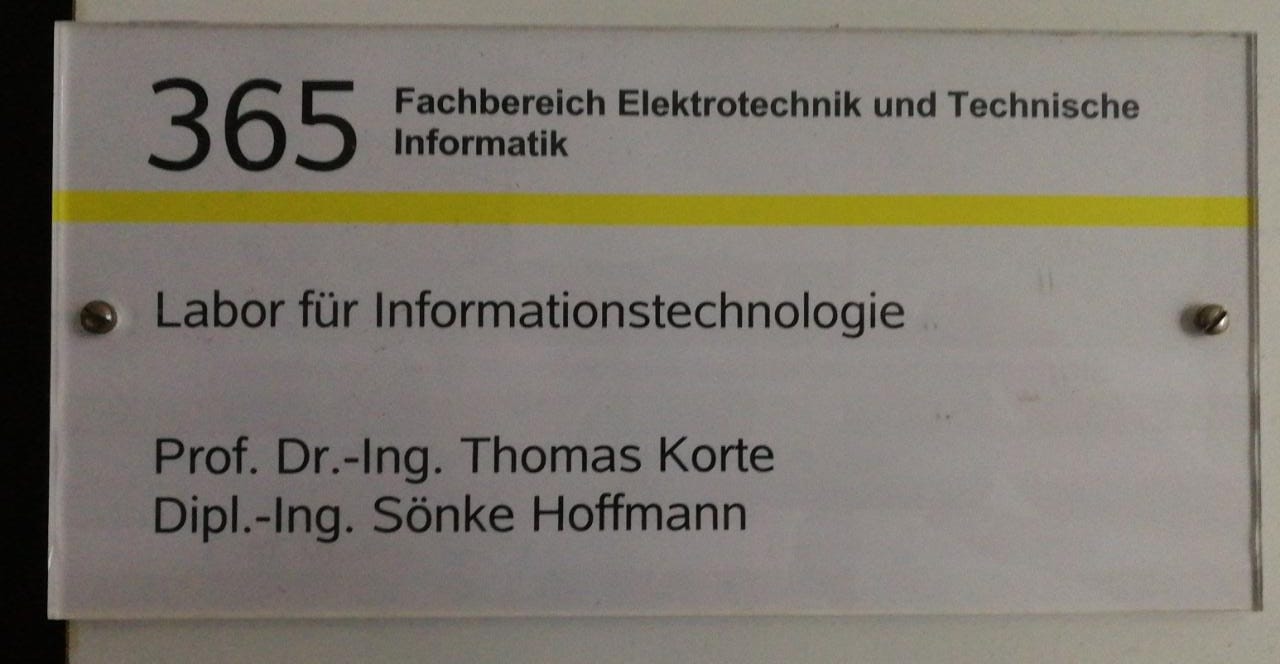


Abbildung - Türschild von Prof. Korte (vorher)

## A4 – Fotos der Türschilder (nachher)

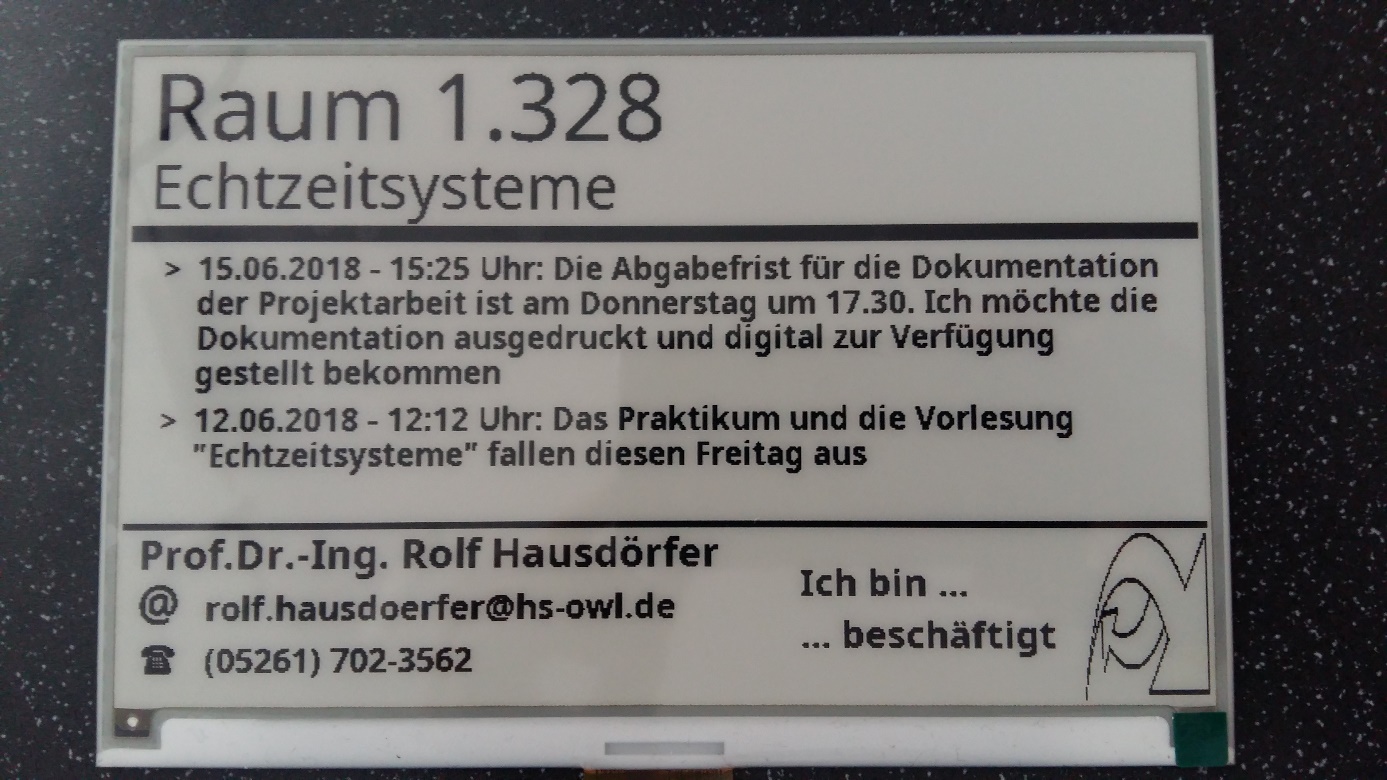


Abbildung - Türschild von Prof. Hausdörfer (nachher)

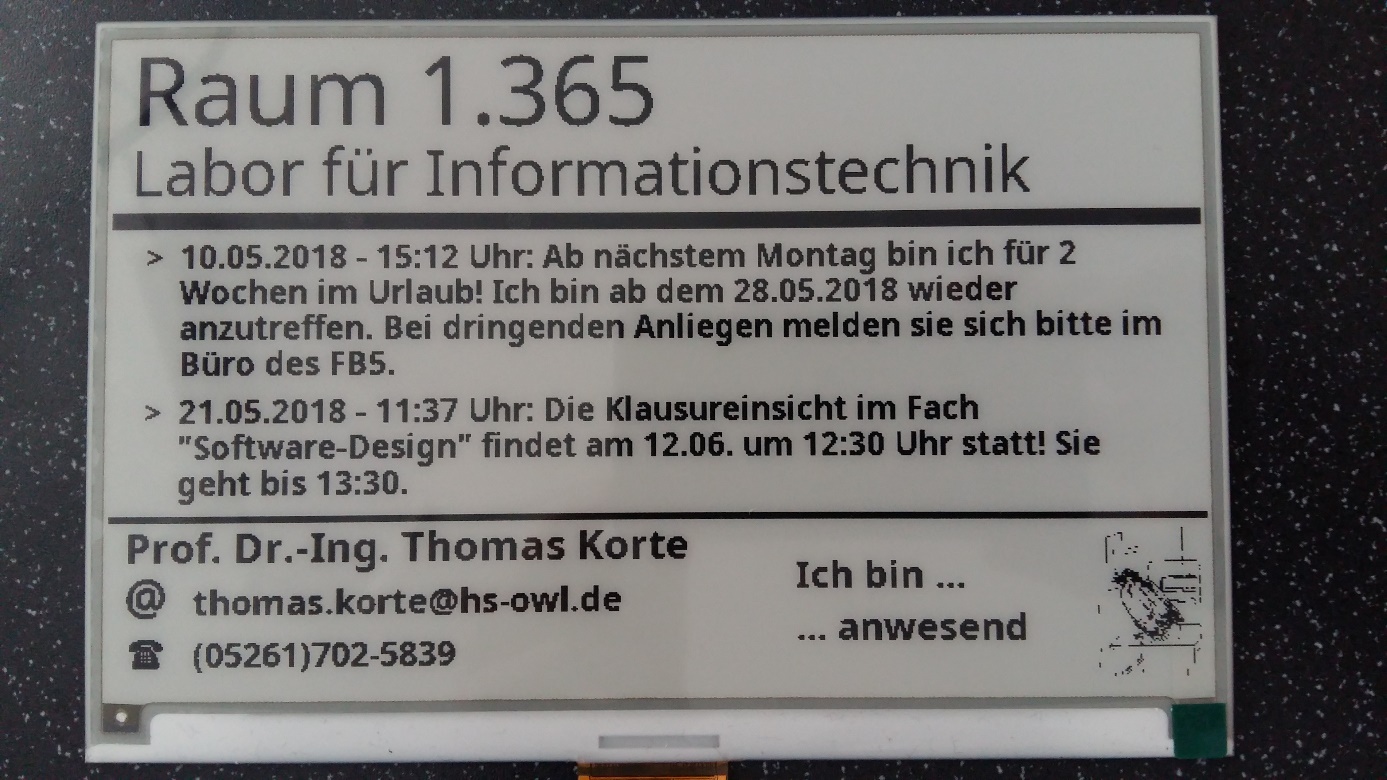


Abbildung - Türschild von Prof. Korte (nachher)

## A5 – Abkürzungsverzeichnis

WLAN Wireless Local Area Network

HAT Hardware Attached on Top

IoT Internet of Things

IDE Integrated Development Environment

LiPo Lithium-Polymer

SRAM Static Random-Access Memory

TCP/IP Transmission Control Protocol/Internet Protocol

http Hypertext Transfer Protocol

PHP Hypertext Preprocessor

GD Gif Draw

PNG Portable Network Graphics

HTML Hypertext Markup Language

CSV Comma-separated values

UTF Unicode Transformation Format

JSON JavaScript Object Notation

JPEG Joint Photographic Experts Group

## A6 – QR-Code zum GitHub-Repository



Abbildung - QR-Code für GitHub

## A7 – Artikel aus der c’t

## A8 – Die „index.php“-Datei

## A9 – Die „turschild\_V1.php“-Datei

## A10 – Die „webinterface.html“-Datei

## A11 – Die “getDataFromCSV.php“-Datei

## A12 – Die “saveDataToCSV.php“-Datei

## A13 – Die Anschlüsse des SparkFun ESP32 Thing

1. Siehe Quelle #1 [↑](#footnote-ref-1)
2. Siehe Quelle #2 [↑](#footnote-ref-2)
3. Siehe Quelle #3 [↑](#footnote-ref-3)
4. Siehe Quelle #4 [↑](#footnote-ref-4)