# Inhaltsverzeichnis

# 1) Einleitung

## 1.1) Aufgabenstellung (Niclas)

## 1.2 Projektbeschreibung (Niclas)

## 1.3) Projektumfeld (Janfi)

# 2) Projektplanung

## 2.1) Projektphasen (Janfi)

## 2.2) Projektressourcen

### 2.2.1) ESP32-Microcontroller (Niclas)

### 2.2.2) Waveshare E-Paper-Display (Niclas)

## 2.3) Gemeinsames Arbeiten

### 2.3.1) Versionsverwaltung über GitHub (Janfi)

### 2.3.2) Backlogverwaltung über Google Sheets (Niclas)

# 3) Analysephase

## 3.1) Beschreibung des CT-Programms (Janfi)

## 3.2) Beschreibung des CT-Webservers (Janfi)

## 3.3) Anforderungsanalyse (Janfi)

# 4) Entwurfsphase

## 4.1) Entwurf des Anzeigebilds (Niclas)

## 4.2) Entwurf des Webinterfaces (Janfi)

## 4.3) Entwurf des Netzwerks (Niclas)

# 5) Implementierungsphase

## 5.1) PHP-Skript zur Anzeige eines Bildes (Niclas)

## 5.2) Hinzufügen von Texten über Webinterface (Janfi)

# 6) Fazit

## 6.1) Soll/Ist-Vergleich (Janfi)

## 6.2) Ausblick (Niclas)

# 7) Quellen

# 8) Anhang

# Einleitung

## Aufgabenstellung

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.

## Projektbeschreibung

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

Unser Ziel 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 konnten wir den Artikel „Ausdauernde Infotafel“ aus der Zeitschrift c’t im Helft 2/2018 nutzen. Das dort vorgegebene Projekt ließ sich gut für unsere Ansprüche erweitern.

# Projektplanung

## Projektressourcen

### ESP32-Microcontroller

Für unser Projekt brauchten wir einen Microcontroller, der Wifi Funktionalitäten bietet. Außerdem muss er batteriebetrieben arbeiten können. Das Programm der c’t, an dem wir uns orientiert hatten, 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 ein Programm diesen öffnen und eine Benutzeroberflächen 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 „Deep Sleep-Modus“ befinden soll, in welchem er nur auf einer Stromstärke von 2.5µA läuft. Wenn sich der Controller „Deep Sleep-Modus“ befindet gibt es mehrere Möglichkeiten ihn aufzuwecken welche vorher definiert werden müssen. Unteranderem auch eine Möglichkeit, in welcher der Controller seine Wifi Funktionalitäten aufrechterhält und auf einen Wakeup-Call über sein Wifi-Modul reagieren kann, welcher sich für unser Projekt anbieten. Bei einem aktiven Betrieb kann der Battery Charger jedoch nicht mit dem Betriebsverbrauch mithalten. Leider konnten wir während unserer Projektarbeit nicht testen ob der Betrieb über eine Batterie mit unserem endgültigen Programm funktionieren würde, da wir keine Batterie zur Verfügung hatten.

Das einzige Problem, welches wir mit dem ESP32 „Thing“ hatten, war der zu geringe Arbeitsspeicher. Mit 520kB internem SRAM, besitzt der Controller eigentlich sogar recht viel Speicher im vergleich zu anderen Controllern seiner Größe, jedoch sind es leider etwas 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 unseres Projektzeitraums jedoch nicht lösen, da die vom Controller und der GxEPD-Library unterstützte Möglichkeit einer paged-Speicherung und Anzeige nicht mit dem gegebenen Programm der c’t nicht kompatibel war und daher eine sehr aufwändige Änderung wäre, für die uns die Zeit während des Semesters fehlte.

### Waveshare E-Paper-Display

Zur Anzeige des Türschilds brauchten wir ein Display, welches Stromsparend arbeiten kann. 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 unseren spezifischen Fall bedeutet das also, dass ein Türschild einmalig auf das E-Paper Display gezeichnet wurde, kann dieses ohne jeglichen weiteren Stromverbrauch, bis zum nächsten zeichnen des Türschildes, angezeigt werden.

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 weiße Partikel in gefärbten Öl schweben. Für jeden 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.

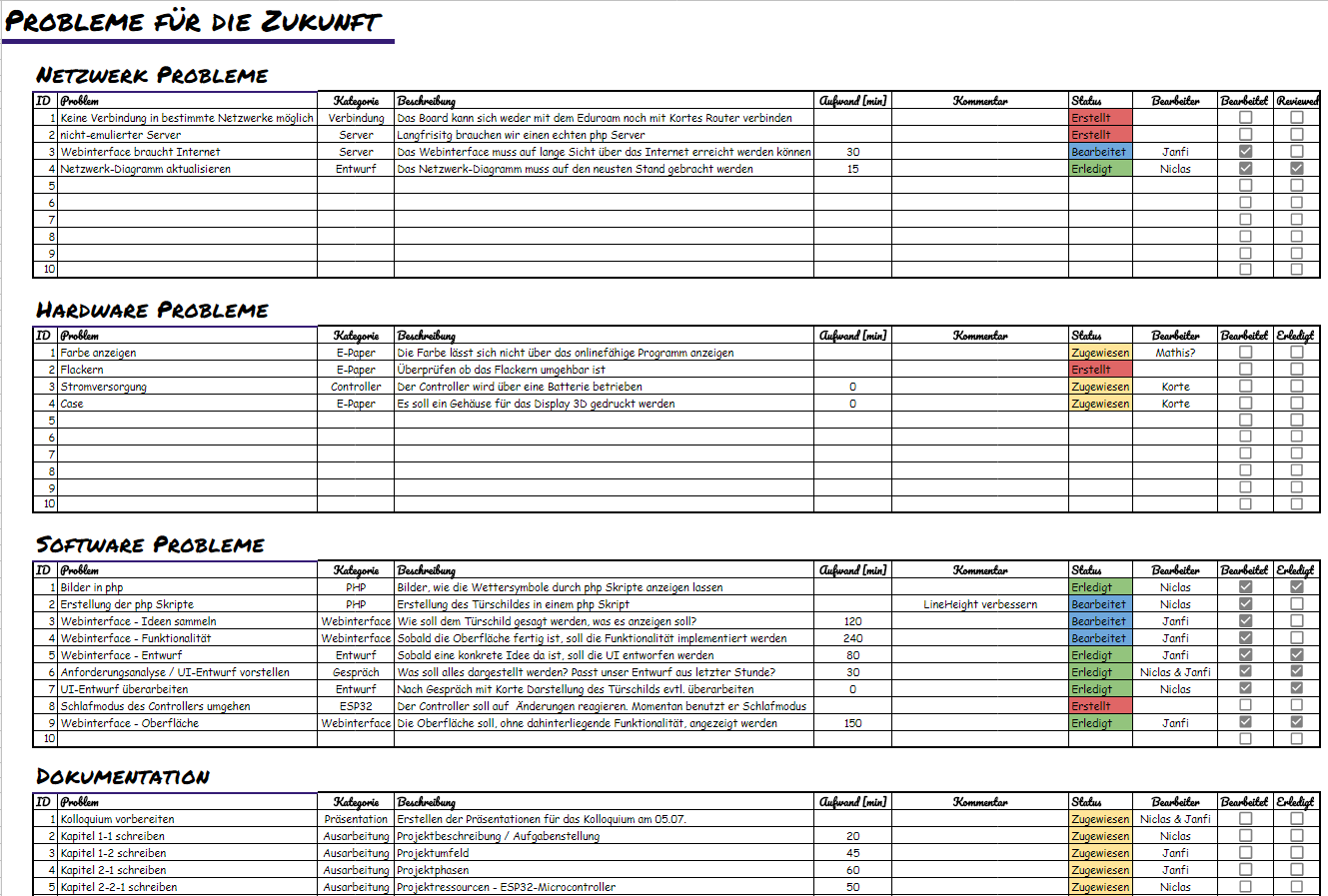
Für unser Projekt benutzen wir die Version des Displays, die neben schwarz und weiß noch eine Farbe anzeigen kann. Die dritte Farbe ist normalerweise Rot, weswegen oft zwischen Schwarz-Weiß und Schwarz-Weiß-Rot unterschieden wird. Die dritte Farbe unsers Displays ist jedoch gelb. Für die Software macht dies jedoch keinen unterscheid, da der Display genau die gleichen Werte übergeben bekommen möchte, 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 Ausgebe gelb sein soll.

## Gemeinsames Arbeiten

### Backlogverwaltung über Google Sheets

Neben GitHub für die Versionsverwaltung, wollten wir noch einen Überblick darüber haben, was wir bereits erledigt haben, und was noch zu tun ist. Wir haben uns für eine tabellarische Übersicht entscheidend, welche wir dann über Googles cloudbasiertes Tabellenbearbeitungsprogramm „Google Sheets“ umgesetzt haben.

Wir haben unsere Backlogs in vier Oberkategorien unterteilt: 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:

* Problem:

Name des Backlogs in wenigen Worten zusammengefasst.

* Kategorie:

Kategorie des Backlogs. Etwas weiter eingeschränkt als die Oberkategorie.

* Beschreibung:

Ausführlichere Beschreibung des Backlogs.

* Aufwand:

Geschätzter Aufwand, der für die Bearbeitung benötigt wird.

Abbildung 1: Überblick über die Backlogverwaltung. Detaillierte Ansicht im Anhang

* Kommentar:

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

* Status: Momentane Status de 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 Status verwalten. Der erste Haken soll gesetzt werden, wenn der Bearbeiter die Bearbeitung abgeschlossen hat. Dieser 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, hatten wir nie Probleme mit verlorengegangenen Änderungen und konnten zu jeder Zeit und auch Ortsunabhängig auf die Backlogs zugreifen. Das Design war übersichtlich genug, damit man problemlos überblicken konnte, was noch zu tun war und wie viel schon erledigt wurde.

# Entwurfsphase

## Entwurf des Anzeigebilds

Für unseren ersten Entwurf des Anzeigebilds wollten wir uns an dem bereits bestehenden nicht-digitalen Türschild orientieren. Da dieses auch nur aus den Farben Schwarz, Weiß und Gelb besteht, war dieses auch ohne weiteres möglich. Mit dem GUI prototyping Programm „Pencil“ erstellten wir unseren ersten Entwurf.

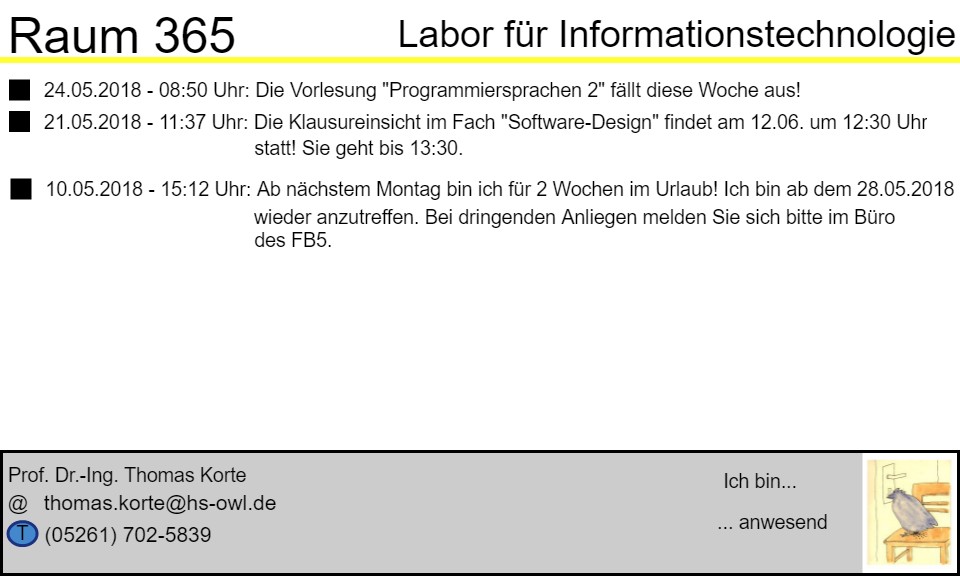
Wir wollten das Türschild in drei Bereiche Aufteilen. Etwas Platz am oberen Ende nach dem Vorbild des bestehenden Türschilds, einen Infomonitor in der Mitte und einen Bereich am unteren Ende für den Anwesenheitsstatus und Informationen über den Professor. Wie in dem bestehenden Türschild, wollten wir am oberen Ende die Raumnummer durch einen gelben Strich vom Rest des Türschilds getrennt haben. Neben der Raumnummer soll auch noch der Raumnamen ausgegeben werden. Um mehr Platz für den Infomonitor zu haben, haben wir uns entschieden die Information über den Fachbereich zu entfernen um mehr Platz für den Infomonitor zu haben. 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 waren unsere ersten Ideen für Professor-spezifische Informationen. Am anderen Ende des Kastens soll der aktuelle Anwesenheitsstatus angezeigt werden. Außerdem haben wir dort noch ein Bild eingefügt um das Türschild etwas anschaulicher zu gestalten. Nachdem genug Platz für den oberen und unteren Bereich geplant haben, war der bleibende Bereich für den Infomonitor vorgesehen.

Abbildung :Erster Entwurf des Anzeigebildes

Für unseren Entwurf haben wir die Informationen von Professor Korte eingetragen und ein paar Beispielmeldungen für den Infomonitor angegeben, sowie einen beispielhaften Anwesenheitsstatus und ein Bild, welches wir aus seinen Vorlesungen entnommen haben.

## Entwurf des Netzwerks

In dem Netzwerk was wir uns bei unserem ersten Entwurf vorgestellt haben sollte Orts- und Geräteunabhängig zu jeder Zeit Informationen, wie der Anwesenheitsstatus und Nachrichten für den Infomonitor aktualisiert werden. Die aktualisierten sollen so gespeichert werden, dass die PHP-Skripte das aktualisierte Türschild „zeichnen“ können, wenn der ESP32 auf den Server zugreift.

Da des E-Paper direkt über ein Kabel mit dem ESP32 verbunden ist, war unsere Erste Hürde die Frage, wie wir des ESP32 mit dem PHP-Server verbinden 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.

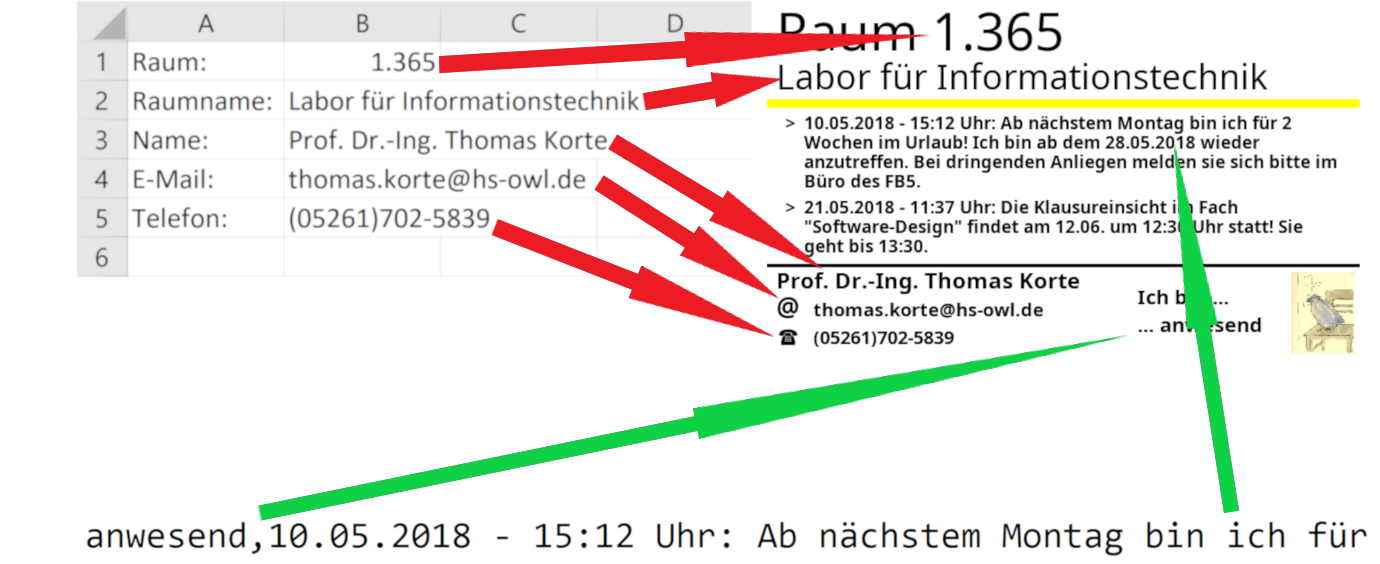
Da wir nicht über unsere eigene Weboberfläche in der Basecamp Weboberfläche Werte ändern können. Mussten wir das aktualisieren des Bildes unabhängig von den Basecamp Funktionalitäten designen. Da Basecamp auf eine spezifizierte PHP-Datei zugreift, müssen wir diesen Pfad statisch belassen, jedoch können wir ändern, 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. 

Abbildung 3: Veranschaulichung wie die csv-Dateien mit dem Anzeigebild zusammenhängen.

# Implementierungsphase

## PHP-Skript zur Anzeige eines Bildes

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“.

### 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 sie versuchen das Skript aufzurufen, müssen sie einige Parameter übergeben, 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 übergeben sie die Größe ihres Displays in Inches und die Information ob er Farben anzeigen kann oder nicht. Die hier möglichen Werte werden in einem Array definiert und betragen 7.5, 7.5bwr, 4.2, 4.2bwr, 2.9, 1.5.
  + Unsere Anzeigeskripte sind jedoch nur für die Bildschirmarten 7.5 und 7.5bwr ausgelegt und skalieren schlecht bis gar nicht auf andere Displaygrößen.
* Content: Name des Anzeigeskriptes. Hier müssen sie Das Anzeigeskript angeben, welches sie ausgeben möchten. Die Datei muss ein 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 sie keinen Wert für debug eingeben wird dieser automatisch auf false gesetzt. Display, Content und Professor jedoch, müssen eingegeben werden, da das Skript ansonsten einen frühzeitig abbricht und eine Fehlermeldung ausgibt. Außerdem überprüft er ob alle wichtigen libraries 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 noch 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 boolscher Wert, der angibt ob der Bildschirm rot, oder in unserem Fall 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, für testzwecke, gelb. Und die Pfade zu den verschiedenen unterstützten Schriftarten in einem Array.

Als nächstes wird das Bild durch das anzeige Skript 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 in nächsten Kapitel genauer erklärt wird, nun die geteilte Bildressource so beschrieben hat, dass sie das Türschild darstellt, liegt es nun wieder 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 Widergabe 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 in als Schwartz ausgebenden und ein helles grau als Weiß. Für ein schwarz-weißes Display ist ein 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.

### 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 sollte. Die GD-Library ist eine Open-Source-Programmbibliothek, zur dynamischen Erzeugung und Manipulation von Grafiken.

Um für eine gute Erweiterbarkeit unseres 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 anzeigen kann, 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 soll kennt das E-Paper Display Softwareseitig nur die Farbe rot für und möchte diese übergeben bekommen. Da aber die endgültige Anzeigefarbe unseres 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.

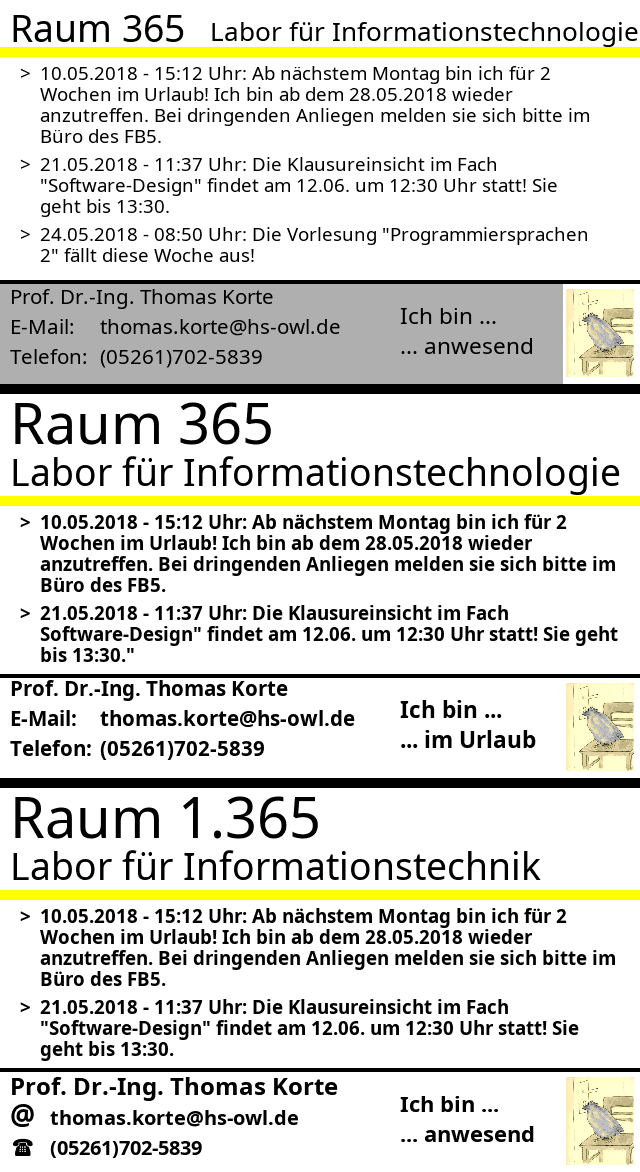
Unser Türschild besteht aus drei Bereichen, wie ich bereits in dem Kapitel „Entwurf des Anzeigebilds“ beschrieben habe. Im oberen Teil soll die Raumnummer und der Name des Büros stehen, welche Daraufhin mit einer gelben Linie vom Rest des Schildes getrennt wird. Zuerst werden die Texte mit der Funktion „imagettftext“ generiert. Schreibt einen Text auf eine Imageressource. 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 bearbeitet 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.

Abbildung 4: Entwicklung des Anzeigebildes über den Projektzeitraum

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 ist uns aufgefallen, 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 Fontart 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 müssen 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 For-Schleife 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 wir noch nicht wissen 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 For-Schleife, 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 sein, in der der Anwesenheitsstatus und einige Informationen zum Professor stehen. Um die Box von dem Infomonitor abzugrenzen, soll das Skript zuerst wieder 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. „imagettftext“ gezeichnet. 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, besten falls, 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 Portable Network Graphics (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 richtig so bemalt worden sein, dass sie das Türschild darstellt.

# Fazit

## Ausblick

Bei der Implementierung des Anzeigeskriptes wurde viel Wert auf die Erweiterbarkeit gelegt. Zusammen mit dem guten Grundgerüst, welches wir von der c’t übernommen haben, 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 Infomonitior einfach ersetzen, ohne dass es einen Einfluss auf den oberen oder den unteren Abschnitt hat.