Entwicklung eines verteilten Anwesenheitsmonitoringsystems

Bachelorarbeit

Bennet Krause

21. März 2018

Christian-Albrechts-Universität zu Kiel
Institut für Informatik
Arbeitsgruppe Programmiersprachen und Übersetzerkonstruktion

Betreut durch: Prof. Dr. Michael Hanus
M.Sc. Marcellus Siegburg

Eidesstattliche Erklärung

Hiermit erkläre ich an Eides statt, dass ich die vorliegende Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

Kiel, 21. März 2018

Zusammenfassung

Besteht innerhalb einer festgelegten Gruppe von Menschen ein allgemeines Interesse an Informationen über die Verfügbarkeit und den (groben) Aufenthaltsort eines jeden Mitglieds, kann der Einsatz eines Systems zur systematischen Erfassung (engl. Monitoring) der Anwesenheit von Gruppenmitgliedern in Erwägung gezogen werden. Speziell bezieht sich das hier entwickelte System auf eine universitäre Arbeitsgruppe mit Professorinnen und Professoren, Doktorandinnen und Doktoranden sowie anderen wissenschaftlichen oder nicht-wissenschaftlichen Mitarbeiterinnen und Mitarbeitern.
Es wurde ein mehrteiliges System entwickelt, das an Türen angebrachte Hardwaremodule, die eine manuelle, physische Eingabe des Anwesenheitsstatus durch das Verschieben eines Magneten über auf Papier gedruckte Statusfelder ermöglicht, zu einem Sensornetzwerk zusammenschließt und die gesammelten Anwesenheitsinformationen der einzelnen Module in einer zentralen Weboberfläche darstellt.

Inhaltsverzeichnis

Abbildungsverzeichnis

Listings

Abkürzungen

BCM Broadcom 51
CE Chip Enable 51
CSN Chip Select Not 51
CSS Cascading Style Sheets 13
DHCP Dynamic Host Configuration Protocol 49
DSL Domain-Specific Language 12
GPIO General Purpose Input/Output 9
HTML Hypertext Markup Language 13
HTTP Hypertext Transfer Protocol 11
IDE Integrated Development Environment 49
OSI Open Systems Interconnection 9
REST Representational State Transfer 9
SPI Serial Peripheral Interface 8
URL Uniform Resource Locator
USB Universal Serial Bus 49
WiF Warantia inter Factiones (zu dt. etwa Garantie zwischen den Parteien) 3

Kapitel 1

Einleitung

Dieses Kapitel soll eine kurze Einführung in das Thema liefern und einen Überblick über die folgenden Kapitel geben.

1.1. Motivation

Arbeitsgruppen im universitären Kontext bestehen nicht selten aus mehr als einem Dutzend Mitgliedern. Neben Professorinnen und Professoren gibt es wissenschaftliche Mitarbeiterinnen und Mitarbeiter, das Sekretariat, sowie weiteres (z.B. technisches) Personal.
Zwischen den Mitgliedern der Arbeitsgruppe besteht eine große Abhängigkeit, gleichzeitig können die der Gruppe zugeordneten Räumlichkeiten groß sein und einige Mitglieder bewegen sich den Tag über nicht nur dort, sondern auf dem gesamten Universitätsgelände. Da der jeweiligen Aufenthaltsort der Mitglieder nicht immer automatisch für alle frei ersichtlich ist, muss dieser vielfach fußläufig ermittelt werden. Wünschenswert wäre eine vom Arbeitsplatz verfügbare Informationsquelle, welche zuverlässig Aufschluss über die An- oder Abwesenheit eines jeden Mitglieds der Arbeitsgruppe gibt.

1.2. Ziel

Das Ziel ist es nun eine solche Informationsquelle in Form eines Anwesenheitsmonitoringsystems zu entwickeln. Das System soll dabei so konzipiert werden, dass es die Vorteile zweier bereits existierender Teillösungen (siehe Kapitel 2) für das selbe Problem kombiniert, um es sowohl für alle Nutzer einfach bedienbar zu machen, als auch einen hohen Informationsgehalt zu liefern.
Dafür soll zur Visualisierung der Anwesenheitszustände aller Nutzer eine Webanwendung entwickelt werden, die sich sowohl für die lokale Nutzung am jeweiligen Arbeitsplatz eignet, als auch zur zentralen Darstellung, beispielsweise auf im Flur oder in Konferenzräumen angebrachten Monitoren.
Jeder Nutzer des Systems soll selbständig seinen aktuellen Anwesenheitsstatus in das System einpflegen. Dafür sollen sowohl die Webanwendung selbst, als auch insbesondere eine physische Eingabemethode an der Tür des Nutzers zur Verfügung stehen. Es soll so genügen, bei Verlassen oder Betreten des eigenen Büros einen Magneten auf das jeweils

passende Statusfeld zu schieben. Der so ausgewählte Anwesenheitsstatus soll automatisch vom System erfasst und sich in der Darstellung der Webanwendung widerspiegeln.

1.3. Aufbau

Zunächst werden in Kapitel 2 zwei sich im Einsatz befindende Systeme vorgestellt, die nach einer ähnlichen Motivation entwickelt wurden wie sie Abschnitt 1.1 beschreibt. Kapitel 3 stellt die verwendete Soft- und Hardware vor, welche für die Entwicklung der Arbeit genutzt wurde. Anschließend folgt in Kapitel 4 die Ermittlung aller funktionalen und nicht-funktionalen Anforderungen des Systems und darauf aufbauend dessen Entwurf. Diesem folgend befasst sich Kapitel 5 mit der Beschreibung der Implementierung der einzelnen Softwarekomponenten. Schließlich wird in Kapitel 6 aufgeführt, welche Weiterentwicklungen für das hier erarbeitete System denkbar wären und in Kapitel 7 werden die Ergebnisse zusammengefasst und bewertet.

Kapitel 2

Bestehende Systeme

In der Arbeitsgruppe Programmiersprachen und Übersetzerkonstruktion der Christian-Albrechts-Universität zu Kiel, existieren bereits zwei unterschiedliche Lösungen, die es Mitarbeiterinnen und Mitarbeitern ermöglichen sollen, sich untereinander über die jeweiligen Anwesenheitsstatus zu informieren.

2.1. Warantia inter Factiones

Warantia inter Factiones (zu dt. etwa Garantie zwischen den Parteien) (WiF) ist eine Webanwendung, die in der Arbeitsgruppe Programmiersprachen und Übersetzerkonstruktion entwickelt wurde und dort auch zum Einsatz kommt (Abbildung 2.1). Grundbestandteil ist eine vertikale Liste aller Mitarbeiterinnen und Mitarbeiter der Arbeitsgruppe mit allgemeinen Informationen wie Name, Titel und Raum, sowie eine Angabe des Anwesenheitsstatus als entweder `Anwesend` (grün markiert) oder `Abwesend` (rot markiert) und gegebenenfalls eine genauere, personalisierte Beschreibung des Status. Mit einem Klick auf den Namen oder das Bild einer Person gelangt man zu einer Detailansicht, die ein größeres Bild sowie weitere Informationen und eine Nachrichtenfunktion beinhaltet (Abbildung 2.2).
Um seinen eigenen Status zu verändern, kann sich eine registrierte Person oben rechts über den „Anmelden“-Button einloggen. Des weiteren können Personen in dem System über ein Suchfeld oben links gesucht werden. Rechts daneben befindet sich ein Textfeld, über das ein zuvor vom System generierter Code eingegeben werden kann, der Zugriff auf ein Formular zur Vereinbarung einer Sprechstunde gewährt.
WiF hat den Vorteil, dass der Status aller Mitarbeiterinnen und Mitarbeiter zentral durch eine überall verfügbare Weboberfläche einsehbar ist. Allerdings muss zur regelmäßigen Änderung des eigenen Status immer die Webanwendung im Browser des Arbeitsplatz-PCs geöffnet bleiben oder neu aufgerufen werden. Zudem muss sich jeder Nutzer selbst daran erinnern, dies auch immer zu tun. Beides könnte dazu führen, dass das System nur unregelmäßig genutzt wird, was es insgesamt zu einer unzuverlässigen Informationsquelle macht und dadurch zu einer kompletten Nichtbenutzung verleiten könnte.

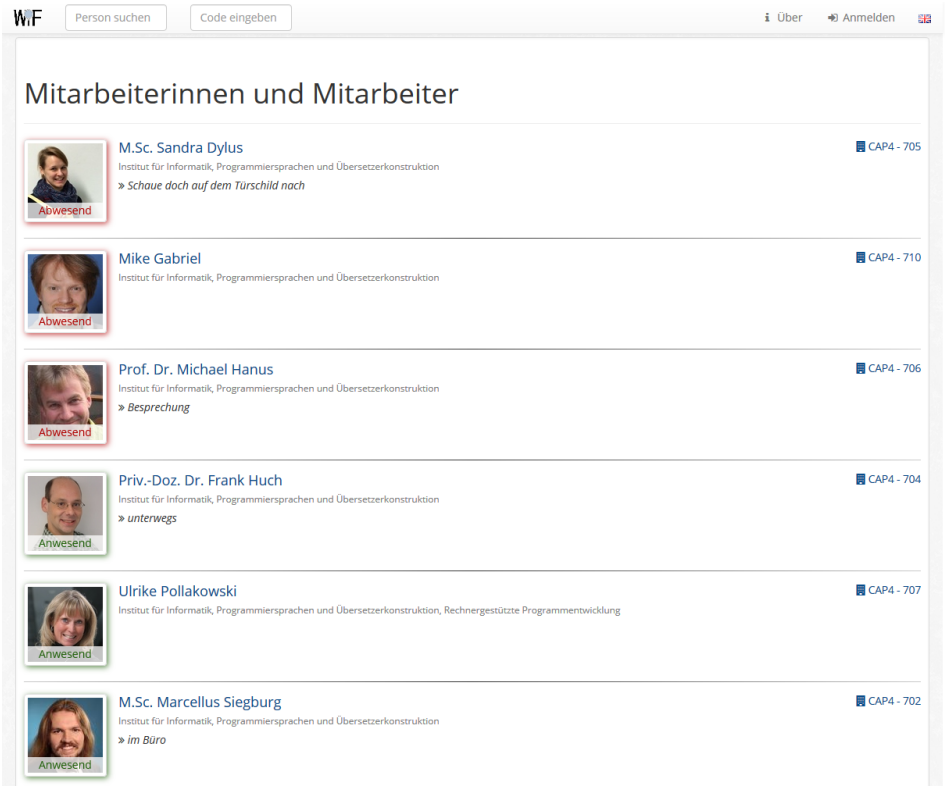


Abbildung 2.1. WiF Weboberfläche

2.2. Magnettafeln

Zusätzlich zu WiF haben viele Mitarbeiterinnen und Mitarbeiter der Arbeitsgruppe an dem Türrahmen ihrer Bürotür eine Magnettafel angebracht, die aus mehreren auf Papier gedruckten Statusfeldern und deren Beschreibung, sowie einem Magneten besteht. Dank des Materials der Türrahmen kann ein Magnet dort direkt haften und so über die Tafel bewegt und auf ihr platziert werden.
Jede Mitarbeiterin und jeder Mitarbeiter ist nun angehalten, jeweils bei Verlassen und Betreten des eigenen Büros, durch Verschieben des Magneten auf ein passendes Feld den gerade geltenden Anwesenheitsstatus anzugeben. Somit können andere Personen beispielsweise bei einem vorgefundenen leeren Büro den ungefähren Grund hierfür erfahren und ob mit einer zeitnahen Rückkehr der Besitzerin oder des Besitzers zu rechnen ist. Bei Besprechungen oder Prüfungen, bei denen die Tür eventuell geschlossenen ist, kann mit einer entsprechenden Angabe auf der Magnettafel ein Klopfen und damit eine Störung verhindert werden.

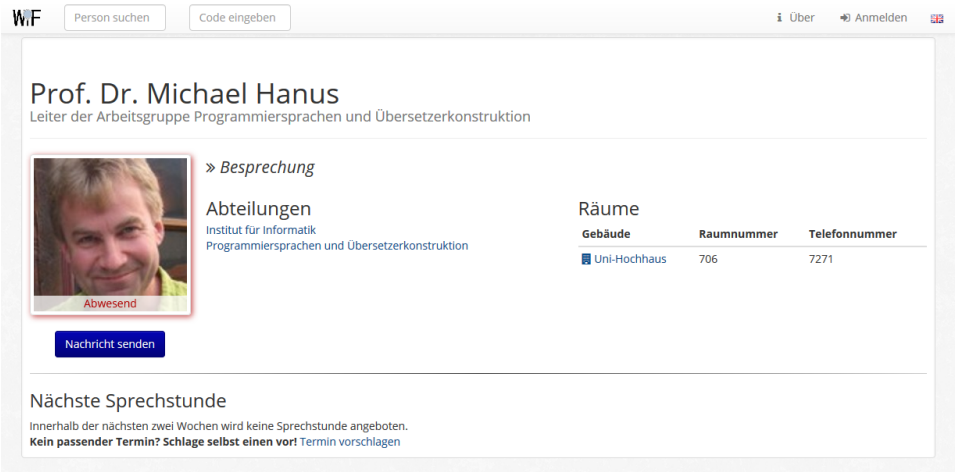


Abbildung 2.2. Detailansicht eines registrierten Nutzers in WiF

Die Magnettafeln haben den klaren Vorteil der einfachen und intuitiven Benutzbarkeit. Das Verschieben des Magneten kostet nur wenig Zeit und der Nutzer wird beim Betreten oder Verlassen des Raumes durch die Platzierung der Tafel automatisch daran erinnert.
Nachteilig im Bezug auf die in Abschnitt 1.1 formulierten Wünsche an ein solches System ist die dezentrale Struktur. Ähnlich wie bei vollständiger Nichtbenutzung eines Anwesenheitsmonitoringsystems müssen gewünschte Informationen fußläufig ermittelt werden, es bleiben primär die oben beispielhaft beschriebenen Vorteile.

Foto einer Magnettafel

Kapitel 3

Verwendete Software und Techniken

In diesem Kapitel werden die wichtigsten Software-Frameworks und -Bibliotheken, sowie Hardware-Elemente vorgestellt, die bei der Entwicklung verwendet wurden. Dort wo es für das Verständnis von Bedeutung ist, werden ausgewählte Konzepte kurz erläutert.

3.1. Arduino

Arduino[1] ist eine 2005 entwickelte Open-Source-Physical-Computing-Plattform, die sich durch einfache Programmierbarkeit und niedrige Anschaffungskosten vor allem an Studierende und Kunstschaffende richtet. Dabei setzen Arduino-Boards größtenteils auf 8-Bit Mikrocontroller der Atmel-AVR-Familie [Hughes 2016, S. 1].

3.1.1. Mikrocontroller

Arduino-Mikrocontroller-Boards werden als Open Hardware veröffentlicht und können daher sowohl in unterschiedlichen Qualitäts- und Preisklassen fertig erworben als auch selbst aus Einzelteilen zusammengebaut werden.
Mikrocontroller sollen vor allem elektronische Abläufe automatisiert steuern, sowie Sensordaten einlesen und verarbeiten. Daher besitzt jedes Arduino-Board einige digitale I/O-Pins, sowie mehrere analoge Ein- und Ausgänge. Viele Arduino-Boards können direkt über eine USB-Verbindung zum PC programmiert und mit Strom versorgt werden, alternativ kann der integrierte Spannungswandler Gleichstrom verschiedener Spannungen entgegennehmen [Banzi 2009, S. 20].
Neben einigen größeren Boards wie dem Arduino Mega oder dem Arduino Uno, deren Ziel vor allem einfaches Prototyping ist, gibt es auch speziell größenreduzierte Boards mit ähnlichem Funktions- und Leistungsumfang. Das kleinste[2] Board der Arduino-Familie ist mit 17,8 × 33 Millimetern der Arduino Pro Mini, bei dem allerdings auf einen USB-Controller verzichten werden muss (Abbildung 3.1).

[1] https://www.arduino.cc/
[2] Quelle: https://en.wikipedia.org/wiki/List_of_Arduino_boards_and_compatible_systems

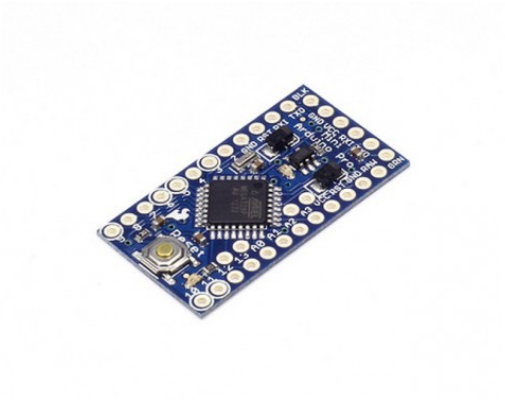


Abbildung 3.1. Arduino Pro Mini Mikrocontroller-Board

3.1.2. Software

Zu der Arduino Plattform gehört auch eine Integrated Development Environment (IDE) welche es ermöglicht die „Sketche“ genannten Programme für Arduino-Boards zu entwickeln und auf die Boards zu übertragen. Die hierfür verwendete Programmiersprache ist eine Variante von C/C++, die auf Processing[3] basiert [Banzi 2009, S. 22]. Über sie und eine Vielzahl von frei verfügbaren Bibliotheken werden technische Details der Mikrocontroller-Programmierung versteckt, damit sich auf das eigentlich zu lösende Problem konzentriert werden kann.

3.2. nRF24 Funkmodule

Nordic Semiconductor[4] bietet verschiedene Transceiver-Schaltkreise an, darunter den nRF24L01+, einen auf geringe Leistungsaufnahme optimierten, im verbreiteten 2,4 GHz Band operierenden Funkchip. Dieser zeichnet sich aus durch einen geringen Spitzenstrom von weniger als 14 Milliampere sowie lediglich 900 Nanoampere im „Power Down“-Modus, ein großes, tief angesiedeltes Eingangsspannungsspektrum von 1,9 bis 3,6 Volt und eine hohe erzielbare Übertragungsrate [Nordic Semiconductor 2008]. Der nRF24L01+ lässt sich über SPI mittels entsprechender Bibliotheken mit einem Arduino nutzen, um verschiedenartige Daten sowohl zu senden als auch zu empfangen.

[3] https://processing.org/
[4] https://www.nordicsemi.com/

3.2.1. Bibliotheken

Unter dem Namen RF24[5] existieren mehrere, aufeinander aufbauende Bibliotheken zur einfachen Nutzung des nRF24L01+ mit einem Arduino.
Die Basisbibliothek ist ein optimierter und in der Schnittstelle vereinfachter Fork des originalen nRF24L01+ Treibers und dient zur einfachen Knoten-zu-Knoten Kommunikation. Darauf aufbauend steht mit RF24Network[6] eine OSI-Netzwerkschicht zur Verfügung, die mittels logischer Adressen und automatischer Nachrichtenweiterleitung eine höhere Abstraktion erlaubt. Wiederum darauf aufbauend kann mit RF24Mesh[7] eine „Mesh"-Schicht konfiguriert werden, die dank automatischer Adresszuweisung eine dynamische Netztopologie erlaubt.

3.3. Raspberry Pi

Raspberry Pi[8] ist eine erstmals 2012 veröffentlichte Serie von Open-Source Einplatinencomputern, die von der Raspberry Pi Foundation entwickelt werden. Auf einer in das Board gesteckten SD-Karte kann eine spezielle Linux-Distribution installiert werden, die den Raspberry Pi zusammen mit seinen USB-, Ethernet-, Audio- und Videoschnittstellen zu einem vollwertigen PC machen kann. Ebenso sind aber über zur Verfügung stehende GPIO-Pins Steuer- und Sensoraufgaben ähnlich wie mit einem Arduino zu realisieren [Dembowski 2013, S. 1-4].
Der aktuellste (Stand 21. März 2018) Raspberry Pi im klassischen Formfaktor ist der Raspberry Pi 3 (Abbildung 3.2), der im Gegensatz zu seinen Vorgängern einen 1,2 GHz Vierkernprozessor von ARM, sowie ein integriertes WLAN- und Bluetooth-Modul besitzt [Engelhardt 2016, S. 1].

3.4. Yesod

Yesod[9] ist ein Haskell basiertes Open-Source Web-Framework zur Entwicklung von typsicheren Webanwendungen nach dem REST-Prinzip.

3.4.1. Haskell

Haskell[10] ist eine rein funktionale, statisch getypte Programmiersprache, die auf dem Lambda-Kalkül basiert. Sie wurde 1990 veröffentlicht und nach dem Logiker Haskell Curry

[5] http://tmrh20.github.io/RF24/
[6] http://tmrh20.github.io/RF24Network/
[7] http://tmrh20.github.io/RF24Mesh/
[8] https://www.raspberrypi.org/
[9] https://www.yesodweb.com/
[10] https://www.haskell.org/

Abbildung 3.2. Raspberry Pi 3 Model B

benannt [O'Sullivan u. a. 2008].
Im Folgenden wird lediglich auf das Konzept der Monaden in Haskell eingegangen, da es für Yesod von großer Bedeutung ist.

3.4.1.1. Monaden

Yesod Anwendungen werden in Haskell geschrieben. Dabei ist das Konzept der sogenannten Monaden essenziell. Rein funktionale Sprachen wie Haskell zeichnen sich normalerweise durch referenzielle Transparenz und Freiheit von Seiteneffekten aus. Bei der Entwicklung einer Webanwendung kann dies aber nicht immer gewährleistet werden. Es müssen beispielsweise Anfrageparameter oder Cookies gelesen und Header oder Session-Informationen geschrieben werden. Solche Aktionen lassen sich nicht einfach in funktionalen Code einbauen, da sie beispielsweise durch äußere Einflüsse eventuell Fehler produzieren können. Um dieses Problem zu lösen wurden Monaden entwickelt.
Eine Monade definiert eine Strategie, Berechnungen zu komplexeren Berechnungen zu kombinieren[11]. Beispielsweise besitzt der Typ `Maybe` (Listing 3.1)

```
1        data Maybe a = Nothing | Just a
```

Listing 3.1. Maybe Datentyp

eine Strategie um solche Berechnungen so kombinieren, die nur eventuell ein Ergebnis haben („Just"): Wenn zwei Berechnungen *A* und *B* z.B. mittels der „bind"-Funktion >>= so kombiniert werden, dass *B* von dem Ergebnis von *A* abhängig ist (Listing 3.2),

```
1        A >>= B
```

Listing 3.2. Binding zweier Berechnungen A und B

[11]Quelle: https://wiki.haskell.org/All_About_Monads

dann bekommt die entstandene kombinierte Berechnung den Wert `Nothing`, falls entweder A oder B zu `Nothing` auswerten und den Wert von B angewandt auf den Ergebniswert von A, wenn beide Berechnungen erfolgreich sind.
Die `Maybe`-Monade kann intern in Funktionen genutzt werden, die normale Werte zurückgeben. Dagegen gibt es sogenannte „Einweg-Monaden“, die es lediglich erlauben Werte in die Monade aufzunehmen und in ihr mittels der „bind“-Funktion Berechnungen auszuführen, es aber verbieten Werte aus der Monade zu entnehmen. Dies bedeutet, dass es nicht möglich ist, eine Funktion zu definieren, die in der Monade eine Berechnung durchführt und deren Rückgabetyp nicht den Typkonstruktor der Monade enthält. Dies stellt sicher, dass mit Seiteneffekten behaftete Operationen in Monaden getrennt von nicht-monadischen Teilen des Programms bleiben. Eine „Einweg-Monade“ ist `IO`, mit der beispielsweise Dateien gelesen oder Texte auf die Konsole ausgegeben werden können.
In Yesod gibt es verschiedene Monaden die auf `IO` aufbauen, darunter `Handler`, die Zugriff auf die HTTP-Anfrage gewährt und es erlaubt Antworten zu senden, `Widget`, über die HTML, CSS und JavaScript generiert wird, sowie `YesodDB` für den Datenbankzugriff [Snoyman 2012, S. 135].

3.4.2. Routen und Handler

Yesod arbeitet nach dem Front-Controller-Prinzip: Anfragen werden automatisch von einem zentralen Punkt zu den passenden Handlern weitergeleitet. Routen werden mittels einer eigenen Domain-Specific Language (DSL) definiert, siehe Listing 5.4 für ein Beispiel. Die zweite Route zeigt, wie typsichere Parameter definiert werden können: „UserId“ kann kein beliebiger Integer-Wert, sondern nur ein gültiger Schlüssel der Datenbanktabelle „User“ sein.

```
1  /              HomeR       GET
2  /user/#UserId  ProfileR    GET POST
```

Listing 3.3. Beispiel der Yesod Routen-Syntax

Für jede unterstütze HTTP-Methode einer definierte Route muss nach einem gewissen Namensschema eine Handler-Funktion existieren. Im obigen Beispiel muss es folglich die Funktionen `getHomeR`, `getUserR` und `postUserR` geben. Die Handler-Funktion `getUserR` könnte zum Beispiel aussehen wie in Listing 3.4:

```
1  getUserR :: UserId -> Handler Html
2  getUserR userId = defaultLayout [whamlet|<h1>Hello World|]
```

Listing 3.4. Handler für die GET-Methode der UserR-Route

Der URL-Parameter wird also zu einem Funktionsparameter des Handlers.

3.4.3. Templates

Yesod benutzt verschiedene Template-Sprachen, deren Namen von Shakespeare inspiriert sind: *Hamlet* für HTML, *Lucius* für CSS und *Julius* für JavaScript [Snoyman 2012, S. 23].

Hamlet ist aufgebaut wie gewöhnliches HTML, setzt aber auf Einrückung statt schließenden Tags. Zudem gibt es erweiterte Konstrukte, die es erlauben Haskell-Code einzubinden. Ein paar relevante werden im Folgenden aufgelistet: Mit `#{x}` greift man auf die Variable *x* in der Umgebung des Template-Aufrufs zu, mit `$if` und `$else` können Bedingungen formuliert werden und `$forall` iteriert über eine Haskell-Liste. Zudem kann beispielsweise mit `@{HomeR}` die zur `HomeR`-Route gehörige URL interpoliert werden.
Ein aus Snoyman [2012, S. 23-24] entnommenes Beispiel für ein *Hamlet*-Template ist in Listing 3.5 aufgeführt.

```
1        $doctype 5
2        <html>
3          <head>
4            <title>#{pageTitle} - My Site
5            <link rel=stylesheet href=@{Stylesheet}>
6          <body>
7            <h1 .page-title>#{pageTitle}
8            <p>Here is a list of your friends:
9            $if null friends
10             <p>Sorry, I lied, you don't have any friends.
11           $else
12             <ul>
13               $forall Friend name age <- friends
14                 <li>#{name} (#{age} years old)
15           <footer>^{copyright}
```

Listing 3.5. Beispiel eines Hamlet HTML-Templates

Lucius ist größtenteils identisch zu CSS, mit dem wichtigsten Unterschied, dass auch hier Variablen- und URL-Interpolation erlaubt ist.

3.4.4. Persistent

Persistent ist eine universelle, typsichere Schnittstelle zur Datenspeicherung [Snoyman 2012, S. 93]. Ihr können unterschiedliche Technologien, wie PostgreSQL, SQLite oder MySQL zugrunde gelegt werden.
Für Datenbankschemata existiert eine eigene DSL, siehe Listing 3.6 für ein Beispiel.

```
1        Person
2          firstname String
```

```
3           lastname String
4           age Int Maybe
```

Listing 3.6. Beispiel eines Persistent Datenbankschemas

Hierbei ist das age-Attribut durch `Maybe` als optional gekennzeichnet. Aus diesen Schemata werden entsprechende Haskell-Typen generiert, mit denen typsicher gearbeitet werden kann. Schlüssel haben einen eigenen Typ und werden von Persistent standardmäßig selbst erzeugt, es existiert also ein implizites Schlüsselattribut.

Aus einer Handler-Monade heraus können nun zum Beispiel Datensätze hinzugefügt und abgefragt werden (siehe Listing 5.1).

```
1       personId <- insert $ Person "Max" "Mustermann" $ Just 42
2       adults <- selectList [PersonAge >. 18]
```

Listing 3.7. Beispiel der Benutzung von Persistent

3.5. Bootstrap

Boostrap[12] ist ein 2011 von Twitter[13] veröffentlichtes Open Source Front-End-Framework zur Entwicklung von Webseiten, die sich automatisch an verschiedene Auflösungen und Displayformate anpassen können. Dafür bietet Bootstrap vorgefertigte HTML- und CSS-Elemente an [Spurlock 2013, S. ix, 1]. Kern von Bootstrap ist ein Grid-System mit 12 Spalten, das sich und seinen Inhalt bei Änderung von Auflösung und Format der Anzeige dynamisch umstrukturieren kann, um möglichst viel Seiteninhalt darstellen zu können.

[12] https://getbootstrap.com/
[13] https://twitter.com

Kapitel 4

Systementwurf

In diesem Kapitel werden zunächst die Anforderungen an die zu entwickelnde Hard- und Software ermittelt. Anschließend wird die Selektion von zu diesen Anforderungen passenden Bauteilen und Softwarekomponenten erläutert sowie die resultierende Struktur und der Aufbau des gesamten Systems beschrieben.

Die Grundlage des hier entwickelten Systems stellen die bestehenden, in Abschnitt 2.2 beschriebenen Magnettafeln dar, die in ein neues, größeres System integriert werden sollen.

4.1. Anforderungen

Als Ausgangspunkt des Systementwurfs werden im Folgenden alle für die Arbeit relevanten funktionalen und nicht-funktionalen Anforderungen ermittelt und beschrieben.

4.1.1. Funktionale Anforderungen

Funktionale Anforderungen legen fest, was ein System tun soll [Robertson und Robertson 2012, S. 10].

4.1.1.1. Erfassung der Magnetposition auf Türschildern

An den Türschildern der Mitarbeiterinnen und Mitarbeiter der Arbeitsgruppe kann bereits jetzt mit einem Magneten der Anwesenheitszustand angegeben werden. Dafür stehen eine variable Anzahl individuell anpassbarer Statusfelder am Türrahmen zur Verfügung, auf denen der Magnet je nach Status platziert werden kann.
Die Position des Magneten, also auf welchem der Statusfelder er sich gerade befindet, soll nun über eine elektronische Komponente an der Tür digital erfasst werden. Dafür soll keine weitere Nutzerinteraktion erforderlich sein, als das Verschieben des Magneten selbst.

4.1.1.2. Unabhängigkeit von Netzstrom

Aufgrund der Montage der Türmodule am Türrahmen und der Position von Steckdosen in den Büros, wird es als nicht praktikabel angesehen, eine Verbindung zum Netzstrom

herzustellen, um die Türmodule zu versorgen. Stattdessen sollen diese mit einer portablen Energiequelle ausgestattet werden, die einen möglichst langen Betriebszeitraum ermöglichen soll.

4.1.1.3. Unabhängigkeit von existierenden Netzwerken

Aufgrund von administrativen Einschränkungen bei der Nutzung von vorhandenen WLAN- und LAN-Netzwerken, sollen die Türmodule in einem autarken Netzwerk operieren, um Daten an die Webanwendung zu senden.

4.1.1.4. Darstellung aller Anwesenheitszustände in einer Webanwendung

Alle erfassten Anwesenheitszustände der installierten Türmodule sollen in einer zentralen Webanwendung zusammengefasst werden. Dabei sind für jedes Modul allgemeine Information der Nutzerin oder des Nutzers, wie Name, Raumbezeichnung und ein Profilbild, aber insbesondere eine Beschreibung des aktuell an der Tür ausgewählten Zustandes anzuzeigen.

4.1.1.5. Konfiguration von personalisierten Beschreibungstexten

Die jedem Zustandsfeld an der Tür zugeordneten Zustandsbeschreibungen, die in der Webanwendung angezeigt werden, sollen für jedes Modul individuell konfigurierbar sein.

4.1.1.6. Verwaltung von Türmodulen

Es muss eine Möglichkeit geben, neue Türmodule in das System zu integrieren und andere zu entfernen. Die Konfiguration eines Moduls muss änderbar sein.

4.1.1.7. Änderung von Anwesenheitszuständen über die Webanwendung

Die Webanwendung soll eine Oberfläche bieten, mit der eine Nutzerin oder ein Nutzer eines Moduls dessen Zustand verändern kann, ohne ein physisches Türmodul zu benutzen, z.B. wenn dieses noch nicht installiert ist. Dafür muss die Webanwendung eine Möglichkeit der Authentifizierung anbieten.

4.1.2. Nicht-funktionale Anforderungen

Nicht-funktionale Anforderungen geben Qualitätseigenschaften des Systems vor [Robertson und Robertson 2012, S. 10].

4.1.2.1. Platzeffiziente Darstellung

Die Übersicht der aktuellen Zustände aller Module in der Webanwendung soll so dargestellt und strukturiert werden, dass sich die Anwendung für eine Art Kiosk-Modus eignet, in der sie auf einem zentral angebrachten Monitor präsentiert wird. Hierfür ist es von großer Bedeutung, dass für eine vollständige Anzeige aller Inhalte die Notwendigkeit von Scroll-Vorgängen minimiert wird.

4.1.2.2. Tastaturlose Bedienung

Die Übersichtsseite aller Zustände soll ohne Tastatur bedienbar sein, damit sie sich für einen Kiosk-Modus auf einem Touchscreen ohne angeschlossene oder eingeblendete Tastatur eignet.

4.1.2.3. Geringe Anschaffungskosten

Da die Türmodule in vielfacher Ausführung angeschafft werden müssen – für jede Nutzerin und jeden Nutzer eines – sollen die Anschaffungskosten der Module gering gehalten werden.

4.1.2.4. Entwicklung in Haskell

Die Webanwendung soll in der Programmiersprache Haskell entwickelt werden, da deren Verwendung bei den wissenschaftlichen Mitarbeiterinnen und Mitarbeitern der Arbeitsgruppe Programmiersprachen und Übersetzerkonstruktion verbreitet ist und somit langfristig Wartung und Betrieb gewährleistet werden kann.

4.2. Entwurf

Eine Betrachtung der Anforderungen aus Abschnitt 4.1 legt eine grobe Einteilung des Systems in zwei Komponenten nahe: Eine Hardwarekomponente zum Einlesen, Verarbeiten und Weiterleiten der Magnetpositionen, sowie eine Softwarekomponente in Form der von Abschnitt 4.1.1.4 geforderten Webanwendung.
Der folgende Abschnitt 4.2.1 beschreibt, wie die Verbindung der Hardware an den Türrahmen zur Webanwendung aufgebaut ist und entwirft die Anwendung selbst. Abschnitt 4.2.2 widmet sich dem Entwurf der benötigten Hardware.

4.2.1. Aufbau der Software

Der Folgende Abschnitt entwickelt ein Konzept, mit dem die an den Türrahmen eingelesenen Sensordaten ohne Nutzung von bestehenden Netzwerken an den Webserver gesendet

werden können. Abschnitt 4.2.1.2 entwirft eine Webanwendung, die diese Sensordaten entgegennimmt und entsprechend den Anforderungen aus Abschnitt 4.1 darstellt.

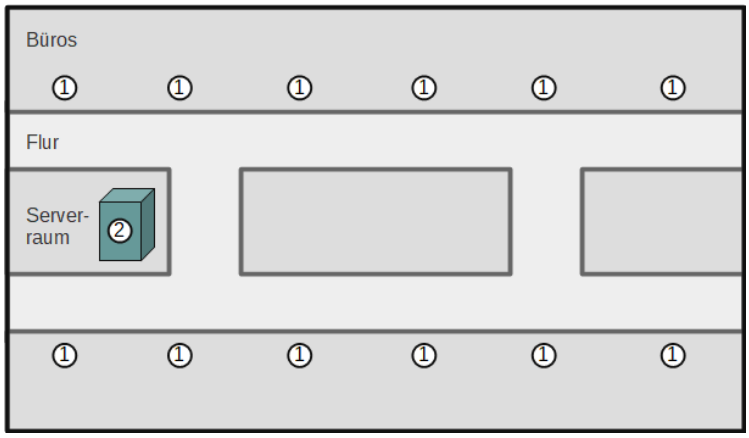
4.2.1.1. Türmodule und Sensornetzwerk

Wie in Abschnitt 4.2.2.1 näher beschrieben, benutzt jedes Türmodul zur Erfassung der aktuellen Magnetposition auf dem Türschild mehrere Hall-Sensoren. Es ist für das Türmodul nicht erforderlich regelmäßig den Zustand der Hall-Sensoren einzulesen, nur wenn sich die Position des Magneten tatsächlich verändert, muss die neue Position bestimmt und an den Server übermittelt werden. Dieser Umstand wird genutzt, um große Mengen Energie zu sparen, indem das Türmodul in einen Energiesparmodus versetzt wird, solange die Magnetposition sich nicht verändert. Die Aktivierung oder Deaktivierung eines Hall-Sensors kann dann verwendet werden, um das Türmodul aus dem Stromsparmodus zu erwecken. Die so erzielte Energieeinsparung ist zwingend erforderlich, da aufgrund von Abschnitt 4.1.1.2 das Türmodul mittels Batterien oder Akkus betrieben werden muss.
Die Arbeit eines Türmoduls kann mit einem zyklischen Ablauf beschrieben werden, der wie folgt aufgebaut ist:

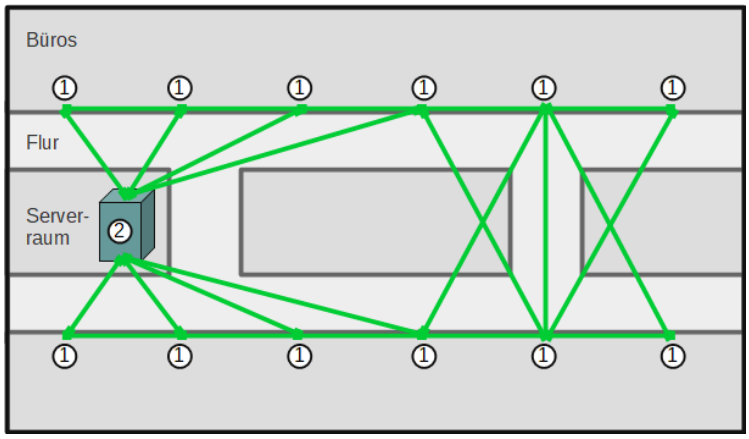
1. Einlesen neuer Sensordaten
2. Aktivieren des Funkmoduls
3. Senden der neuen Daten an den Server
4. Deaktivieren des Funkmoduls
5. Wechsel in den Energiesparmodus
6. Automatisches Erwachen aus Energiesparmodus durch Aktivierung/Deaktivierung eines Sensors
7. Wiederholung des Zyklus bei 1.

Um von den WLAN- und LAN-Netzwerken unabhängig zu sein, muss ein eigenes, vollständig autarkes Netzwerk aufgebaut werden. Abbildung 4.1a stellt die Ausgangssituation beispielhaft dar. Über das gesamte Stockwerk verteilt sind die Türmodule angebracht (1). An Position (2) befindet sich der Server, auf dem die Webanwendung läuft. Alle Türmodule müssen in der Lage sein, eine Verbindung zum Webserver aufzubauen. Da das Verlegen von Kabeln nicht verhältnismäßig erscheint, muss eine Funkverbindung genutzt werden. Der Server kann allerdings keine per Funk übermittelten Datenpakete direkt empfangen, deshalb kommt eine eigene Hardwarekomponente zum Einsatz, die über einen Funkempfänger verfügt. Diese „Mastermodul“ genannte Komponente (beschrieben in Abschnitt 4.2.2.3) empfängt alle von den Türmodulen gesendeten Datenpakete und leitet deren Informationen über eine Ethernet-Schnittstelle an den Server weiter.

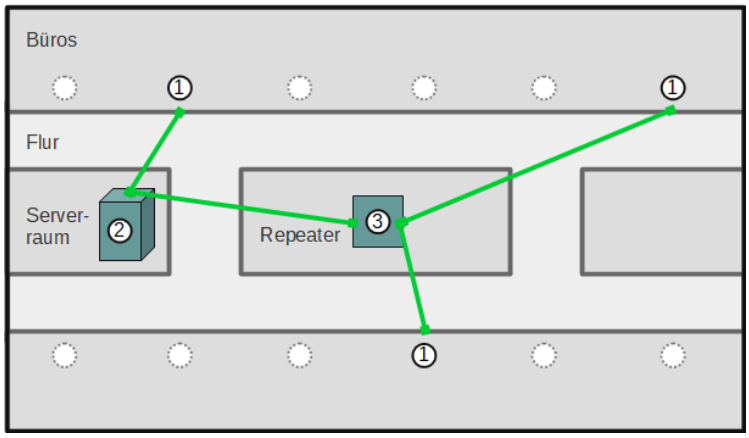
Nicht unbedingt jedes Türmodul kann aufgrund von Entfernung und baulichen Hindernissen eine direkte Verbindung zum Mastermodul und damit zum Server aufbauen.



(a) Die Türmodule (1) müssen eine Verbindung zum Mastermodul des Servers (2) aufbauen.



(b) Mögliche Netztopologie bei einem Mesh-Netzwerk. Alle Türmodule (1) können sich mit dem Master (2) verbinden.



(c) Alle aktiven Module (1) können sich direkt oder über das Repeater-Modul (3) mit dem Master (2) verbinden.

Abbildung 4.1. Beispielhafte Darstellungen des Sensornetzwerkes

Daher liegt es nahe, die jeweils geringen Abstände zwischen den einzelnen Türmodulen auszunutzen, um ein vermaschtes Netz (engl. Mesh Network) aufzubauen (siehe Abbildung 4.1b). Bei dieser Netztopologie sind alle Netzwerkknoten miteinander verbunden, bei denen dies die physikalischen Gegebenheiten zulassen. Pakete werden nicht unbedingt direkt vom Start- zum Zielknoten gesendet, sondern falls nötig von einem oder mehreren anderen Netzwerkknoten weitergeleitet, um die vorhandene Distanz zu überbrücken. Das Problem dieses Lösungsansatzes liegt darin, dass um jederzeit eine problemlose Zustellung von Paketen gewährleisten zu können, alle Netzwerkknoten permanent empfangsbereit sein müssen. Da die Türelemente aber nach Abschnitt 4.1.1.2 keine dauerhafte Stromverbindung besitzen dürfen und deshalb von portablen Stromquellen abhängig sind, ist es nicht praktikabel die Module dauerhaft empfangsbereit zu halten. Um eine lange Betriebszeit zu ermöglichen, müssen die Türmodule so oft und lange wie möglich in einen Energiesparmodus versetzt werden, währendem keine Daten empfangen oder gesendet werden können. Eine Lösung besteht nun darin, dedizierte Knoten für die Weiterleitung von Paketen (engl. Repeater) bereitzustellen, die an Netzstrom angeschlossen sind (siehe Abbildung 4.1c). Jedes Türmodul kann jederzeit und so lange wie nötig in den Energiesparmodus wechseln, ohne die Gesundheit des Netzwerkes zu gefährden. Ein oder mehrere Repeater-Module (3) stehen dank dauerhafter Stromversorgung permanent zur Verfügung, um Pakete von gerade aktiven Türmodulen an den Master weiterzuleiten. In Abbildung 4.1c sind Türmodule, die gerade inaktiv und daher nicht in das Netzwerk eingegliedert sind, gestrichelt dargestellt.
Um ein Sensornetzwerk nach dem in Abbildung 4.1c dargestellten Prinzip aufbauen zu können, wurden nRF24L01+ Funkmodule ausgewählt. Diese Wahl wird in Abschnitt 5.1 näher beschrieben und begründet.
Um eine hohe Skalierbarkeit zu erlangen und wie von Abschnitt 4.1.1.6 gefordert in der Lage zu sein Türmodule hinzuzufügen und zu entfernen, wird auf eine direkte Verwendung der SPI-Schnittstelle des Chips verzichtet. Stattdessen wird auf Bibliotheken zurückgegriffen, die ein höheres Level an Abstraktion bieten.
Die Bibliothek `RF24` für nRF24L01(+) Chips stellt nur eine dünne Abstraktionsschicht über den vom Chip vorgegebenen SPI-Befehlen dar. Sie nutzt Schreib- und Lese-Pipes die individuell geöffnet werden müssen. Dazu ist nach einem Schreibvorgang ein expliziter Wechsel zurück in den Lesemodus nötig, wenn wieder Daten empfangen werden sollen oder umgekehrt. Auch können maximal sechs Lese-Pipes gleichzeitig offen sein, zu wenig um von allen Türmodulen Daten empfangen zu können. `RF24` eignet sich daher vor allem für uni- oder bidirektionale Verbindungen zwischen nur zwei Knoten.
Die auf `RF24` aufbauende `RF24Network`-Bibliothek bietet einige weitergehende Funktionalitäten an. `RF24Network` umgeht die Limitierung von `RF24`, lediglich sechs gleichzeitige Verbindungen nutzen zu können, indem durch logische Adressen Subnetze aufgebaut werden. In der entstehenden Baumstruktur können Pakete mittels einer logischen Zieladresse automatisch durch das Netz bis zum Zielknoten weitergeleitet werden, falls keine direkte Verbindung besteht. In Abbildung 4.2 ist das logische Adressierungsschema veran-

schaulicht[1]. Das Schema verwendet das Oktalsystem für die Repräsentation der Adressen. Der Master-Knoten befindet sich in der Mitte und besitzt immer die logische Adresse $0_{(8)}$. Um ihn herum werden nun – ihrer tatsächlichen physischen Position grob entsprechend – weitere Knoten auf mehreren Ebenen in einer Baumstruktur aufgebaut. Da die Limitierung auf gleichzeitige sechs Pipes bestehen bleibt, hat in dieser Struktur jeder Knoten (bis auf den Master) genau einen Vaterknoten und bis zu fünf Kinderknoten. Dank der Baumstruktur können nun durch Hinzufügen von neuen Teilbäumen beliebig viele Knoten in das Netz aufgenommen werden. Die Struktur des Baumes ist dabei allein durch die logischen Adressen jedes Knotens definiert. Die bis zu fünf Kinder des Master-Knotens ($0_{(8)}$) bekommen die Adressen $1_{(8)}$, $2_{(8)}$, $3_{(8)}$, $4_{(8)}$ und $5_{(8)}$. Kinder von $1_{(8)}$ haben die Adressen $11_{(8)}$, $21_{(8)}$, $31_{(8)}$, $41_{(8)}$, $51_{(8)}$ (vgl. Abbildung 4.2).
Nachrichten von einem Knoten an einen anderen, nicht direkt mit ihm verbundenen Knoten, können nun anhand der logischen Zieladresse automatisch durch die Baumstruktur weitergereicht werden. Einem Zwischenknoten, der die Nachricht zunächst erhält, genügt die im Header gespeicherte Zieladresse, um zu bestimmen, an welchen der mit ihm verbundenen Knoten er die Nachricht weiterreichen muss. In Abbildung 4.2 könnte z.B. Knoten $1_{(8)}$ eine Nachricht an $225_{(8)}$ senden wollen. Durch den Aufbau der Zieladresse ist für den Startknoten $1_{(8)}$ klar, dass er die Nachricht zunächst an den Master schicken muss, da seine eigenen Kindknoten zu einem anderen Subnetz gehören. Vom Master aus kann nun die Zieladresse von rechts angefangen durchlaufen werden und die Nachricht passiert die Zwischenknoten $5_{(8)}$ und $25_{(8)}$, bevor sie den Zielknoten $225_{(8)}$ erreicht.
Ein großer Nachteil des Aufbaus einer baumartigen Netztopologie aus logischen Adressen besteht darin, dass die Struktur des Netzes relativ starr ist. Sie wird in der Regel einmal den Gegebenheiten entsprechend entwickelt und bleibt dann größtenteils bestehen. Zwar können in `RF24Network` neue Knoten ad-hoc dem Netzwerk beitreten, dies erfordert aber eine genaue Kenntnis der vorhandenen Netzstruktur, da für den neuen Knoten genau die passende logische Adresse gefunden werden muss, damit dieser sich korrekt in das Netzwerk einfügt. Beispielsweise kann beim Hinzufügen eines einzelnen Knotens das Anlegen eines komplett neuen Subnetzes nötig sein, wenn im Teilbaum darüber bereits fünf Kinderknoten vorhanden sind. Noch größere Probleme bereitet unter Umständen das Entfernen von Knoten aus dem Netz. Sofern der entfernte Knoten kein Blatt im Baum darstellt, hat dies immer einen dauerhaften Ausfall aller seiner Subnetze zur Folge. Das Entfernen von Knoten $5_{(8)}$ würde beispielsweise in der Nichterreichbarkeit von vier weiteren Knoten resultieren.
Dieses Problem kann mit einer dritten Abstraktionsschicht gelöst werden. `RF24Mesh` basiert auf `RF24Network` und erweitert es um eine dynamische Adressierung. Dafür bekommt jeder Knoten eine eindeutige Zahl zwischen 1 und 255 (`Node ID`) zugewiesen, um von den logischen Adressen, welche die Netztopologie bestimmen, zu abstrahieren. Die Zuweisung der logischen Adressen übernimmt der Master-Knoten, welcher dafür permanent eine Tabelle vorhält, die von `Node IDs` auf logische Netzwerkadressen abbildet. Dabei kann sich

[1] Basierend auf http://tmrh20.github.io/RF24Network/Tuning.html

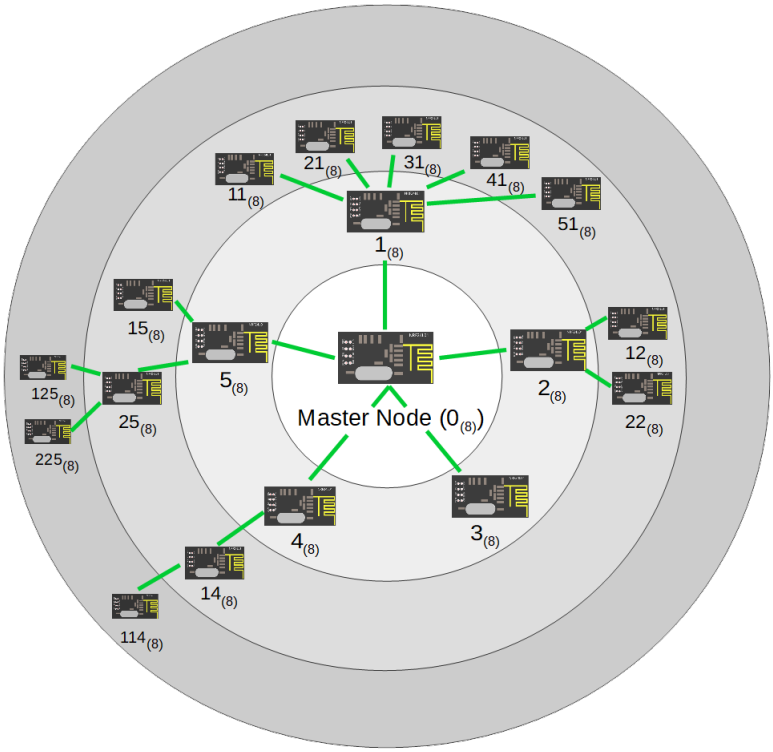


Abbildung 4.2. Baumtopologie durch logische Adressierung in `RF24Network`

die Adresse, welche einer `Node ID` zugeordnet wurde, jederzeit ändern, ein Knoten kann auch zeitweise gar keine Adresse besitzen. Die Zuweisung der Adressen geschieht dabei stets so, dass die Netztopologie möglichst günstig aufgebaut ist und alle Knoten eine Verbindung in das Netz haben. Beim Hinzufügen eines Knotens in das Netz wird seiner `Node ID` eine freie, passende Adresse zugewiesen. Beim Entfernen eines Knotens wird die von ihm gerade belegte Adresse freigegeben und falls nötig die Adressen der anderen Knoten aktualisiert, um die Topologie den neuen Gegebenheiten anzupassen.

4.2.1.2. Webanwendung

Dieses Kapitel befasst sich mit der Konzeption der Webanwendung zur Darstellung aller Anwesenheitszustände.

Model Zur Erfüllung der Anforderungen an die Webanwendung wurde das in Abbildung 4.3 dargestellte Datenbankschema entworfen. Es besteht aus den beiden Entitäten

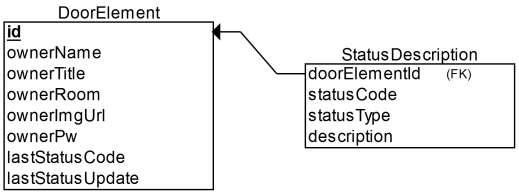


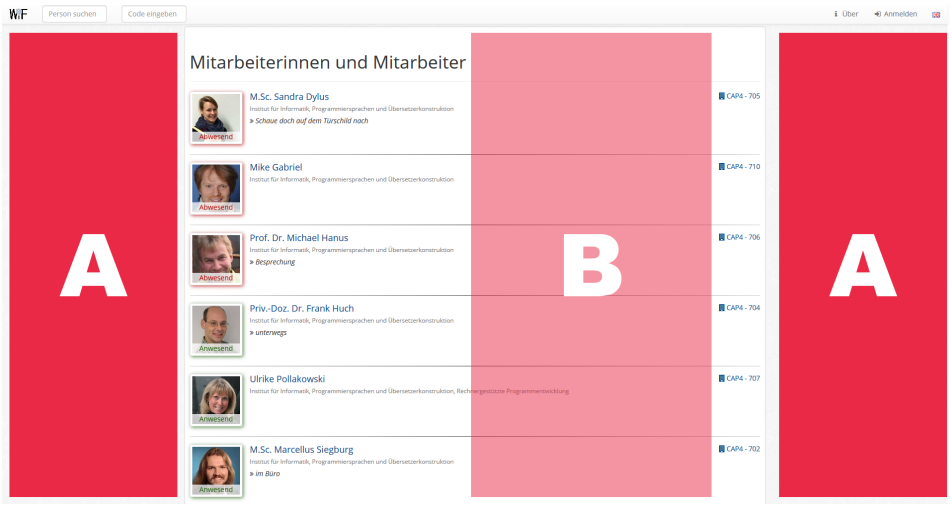
Abbildung 4.3. Entity-Relationship-Modell der Webanwendung

DoorElement und *StatusDescription*.

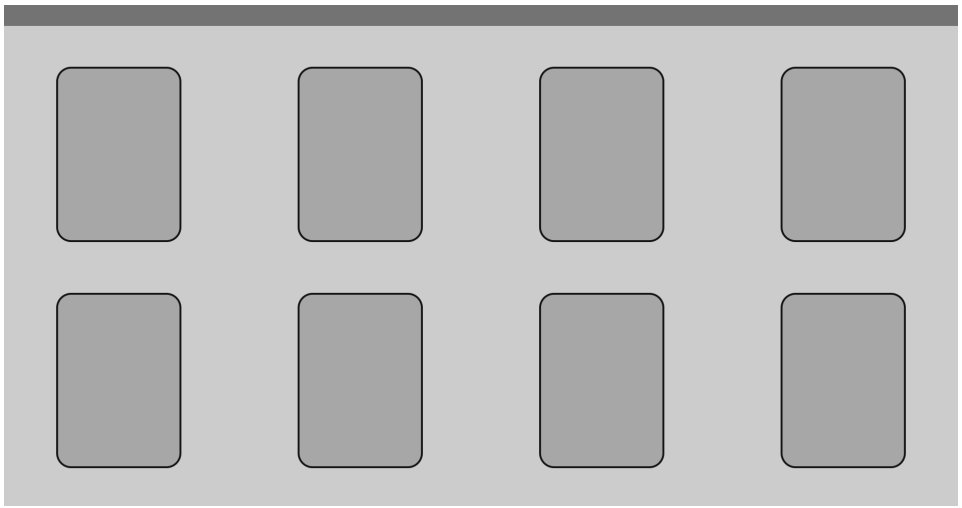
Die *DoorElement*-Entität stellt ein konfiguriertes Türmodul dar. Der Primärschlüssel `id` wird beim Einfügen neuer *DoorElement*s stets als die `Node ID` des zugehörigen Türmoduls gewählt. Dadurch wird implizit eine Verbindung zwischen Hardware-Türmodulen und ihrer Repräsentation in der Webanwendung hergestellt. `ownerName`, `ownerTitle`, `ownerRoom` und `ownerImgUrl` bezeichnen Name, akademischen Titel (falls vorhanden), Raumbezeichnung und die URL eines Profilbildes der Besitzerin oder des Besitzers. `ownerPw` speichert das konfigurierte Passwort, mit dem sich die Nutzerin oder der Nutzer zusammen mit der `Node ID` ihres Türmoduls in der Webanwendung authentifizieren kann. `lastStatusCode` speichert den letzten, vom Türmodul übermittelten Anwesenheitsstatus als Integer-Wert. `lastStatusUpdate` gibt den Zeitpunkt der letzten Aktualisierung an, damit nicht länger antwortende Module identifiziert werden können.

Jedes *DoorElement* besitzt eine gewisse Anzahl von Statusbeschreibungen, die in der Entität *StatusDescription* zu finden sind. Sie ordnet dem in `lastStatusCode` gespeicherten Statuscode eines *DoorElement*s eine Beschreibung (`description`), sowie einen Typ (`statusType`) zu. Der Typ wird genutzt, um in der View verschiedene Klassen von Anwesenheitszuständen unterscheiden zu können und diese unterschiedlich darzustellen.

View Die Oberfläche der Webanwendung orientiert sich grob an dem WiF-System. Die Anforderung aus Abschnitt 4.1.1.4 erlaubt aber eine schlankere Darstellung, da insgesamt weniger Informationen angezeigt werden müssen. Auch eignet sich die bei WiF verwendete vertikale Listenansicht nicht für einen Kiosk-Modus (Abschnitt 4.1.2.1), da bei heute üblichen Breitbildformaten der verfügbare vertikale Platz schnell erschöpft ist (siehe Abbildung 4.4a). Bei Vollbildansicht auf einem Full-HD-Monitor im 16:9-Format können zwei der acht Elemente nicht angezeigt werden. Um auch sie zu sehen muss gescrollt werden, was für ein Kiosk-System nicht praktikabel ist. Dabei bleib in den rot markierten Randbereichen links und rechts (`A`) sowie in der Mitte (`B`) viel Platz ungenutzt. In Abbildung 4.4b wird eine alternative Darstellungsform vorgeschlagen. Statt einer vertikalen Liste kommt hier ein Grid-System zum Einsatz, das die einzelnen kartenartigen Elemente möglichst effizient gruppiert. Dadurch können alle acht Elemente ohne Scrollen angezeigt werden,



(a) Oberfläche von WiF. In den Randbereichen `A` und im Bereich `B` bleibt viel Fläche ungenutzt. Darstellbar sind bei dieser Auflösung 6 Elemente, 2 können nicht angezeigt werden.



(b) Alternative Darstellungsstruktur zu WiF. Alle 8 Elemente können angezeigt werden.

Abbildung 4.4. Vergleich der WiF-Oberfläche mit einer neu entwickelten Struktur

zwei weitere könnten durch Reduzierung der horizontalen Abstände zwischen den Karten Platz finden. Die breite Bildschirmfläche wird hier besser ausgenutzt als in Abbildung 4.4a.

Die Karten sind wie in Abbildung 4.5 aufgebaut. `1` zeigt das von `ownerImgUrl` bestimmte

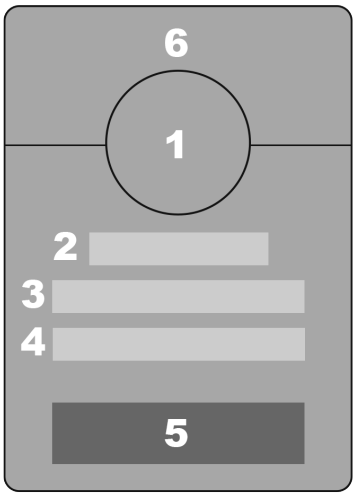


Abbildung 4.5. Struktur einer Kartenansicht

Profilbild an. In `2` wird der Titel (`ownerTitle`), in `3` der Name (`ownerName`) und in `4` die Raumbezeichnung (`ownerRoom`) angezeigt. `5` ist ein Button, dessen Beschriftung der Statusbeschreibung entspricht (`description` aus *StatusDescription*). Der Button wird je nach Wert von `statusType` aus *StatusDescription* unterschiedlich eingefärbt. Zur schnellen optischen Erfassung des aktiven Status hat der Hintergrund in `6` die selbe Farbe wie der Button `5`. Mit einem Klick auf `5` gelangt man zu einer Auflistung aller für dieses Element konfigurierten Statusbeschreibungen als eine vertikale Liste von Buttons. Ein Klick auf einen dieser Buttons hat eine Änderung des aktuellen Status zur Folge (siehe Abschnitt 4.1.1.7).

Zur Authentifizierung wird in der Navigationsleiste ein Login-Button dargestellt, der zu einem einfachen Login-Formular führt. Dieses besteht aus einem Textfeld für die `Node ID` der Nutzerin oder des Nutzers, sowie einem Passwortfeld für das konfigurierte Passwort. Über einen Button kann das Formular abgeschickt werden. Ist die Nutzerin oder der Nutzer bei einem Klick auf den Status-Button (`5`) noch nicht authentifiziert, wird automatisch zu dem Login-Formular weitergeleitet.

Controller Die Übersicht der Zustände aller Türmodule befindet sich unter der Index-Route „`/`“. Der für diese Route zuständige Controller ruft alle *DoorElement*s aus der Datenbank ab, füllt ein entsprechendes HTML-Template und gibt es zurück.

Um den Anwesenheitsstatus eines Türmoduls aktualisieren zu können, existiert die Route „`/status/#DoorElementId/#StatusCode`“, welche lediglich die POST-Methode unterstützt. Der Controller setzt bei einem autorisierten Request an diese Route bei demjenigen

DoorElement, das den Schlüsselwert „`#DoorElementId`“ besitzt, den Wert des `lastStatusCode`-Attributs auf „`#StatusCode`“. Im gleichen Schritt wird zudem `lastStatusUpdate` auf die aktuelle Systemzeit gesetzt.

Die GET-Route „`/status/#DoorElementId/`“ gibt Zugriff auf die Ansicht zur Änderung des Anwesenheitsstatus direkt in der Webanwendung (Abschnitt 4.1.1.7). Dabei werden alle für dieses *DoorElement* in *StatusDescription*s hinterlegten Statusbeschreibungen angezeigt. Diese sind jeweils mit der „`/status/#DoorElementId/#StatusCode`“-Route verlinkt, um eine Änderung des Status per Klick auf eine Beschreibung zu erlauben.

Für die Authentifizierung können eingebaute Routen des Frameworks genutzt werden.

4.2.2. Hardwareentwurf

Zur Erfüllung der Anforderungen muss eine kompakte Hardwarekomponente entwickelt werden, die an einem Türrahmen anzubringen ist, ohne dauerhaften Stromanschluss auskommt und ohne herkömmliche Netzwerkverbindung Daten an den Webserver senden kann.

4.2.2.1. Türmodule

Die zentrale Funktion des Türmoduls besteht darin, die Position des Magneten auf dem Türschild über Sensoren zu erfassen und diese über das in Abschnitt 4.2.1.1 vorgestellte Mesh-Netzwerk an das Mastermodul zu senden.
Aufgrund der Anforderung aus Abschnitt 4.1.1.2 steht dem Türmodul kein Netzstrom zur Verfügung. Daher wird es so konzipiert, dass es mit Batterien oder einem Akku betrieben werden kann. Damit dies möglich wird, müssen alle Komponenten des Moduls unter dem Aspekt des geringen Stromverbrauches ausgewählt werden.

Mikrocontroller Hauptkomponente des Moduls ist das Arduino Pro Mini Mikrocontroller-Board. Die Arduino-Plattform eignet sich durch eine große Unterstützung von Open-Source-Bibliotheken und günstige Hardware besonders für die Entwicklung von Prototypen. Der Arduino Pro Mini zeichnet sich durch einen sehr kleinen Formfaktor und eine geringe typische Leistungsaufnahme aus, die durch verschiedene Maßnahmen weiter reduziert werden kann:

1. Der Pro Mini besitzt eine fest verdrahtete Power-LED, die nicht durch Software deaktiviert werden kann. Wird ihre Verbindung durchtrennt, spart dies ca. 1,5 mA.[2]

[2]Quelle: https://www.mysensors.org/build/battery

2. Der integrierte Spannungswandler des Boards erlaubt ein breites Spektrum an Eingangsspannungen. Wenn direkt eine nutzbare Spannung vorliegt, kann der Wandler abgetrennt werden. Dies spart ca. 220 μA.[3]

3. Wird auf die 3,3 statt der 5 Volt Variante des Pro Mini zurückgegriffen, die mit 8 statt 16 MHz Chipfrequenz arbeitet, reduziert sich die Stromaufnahme von ca. 13,92 mA auf ca. 3,87 mA.[4] Die minimale Betriebspannung des verbauten ATmega328P beträgt 1,8 V.

4. So häufig wie möglich sollte der Chip in den „Power Down“ genannten, sparsamsten Energiesparmodus wechseln, in dem der Arduino ohne Power-LED und Spannungswandler nur noch ca. 4,3 μA benötigt, statt ca. 3,87 mA im aktiven Zustand.[5] In diesem Modus deaktiviert der Chip alle Funktionen, außer einem Timer und Hardware-Interrupts. Somit kann er nach Ablauf einer vorgegebenen Zeit oder durch ein externes Ereignis wieder aufgeweckt werden.

Funkchip Um das in Abschnitt 4.2.1.1 gezeigte Mesh-Netzwerk aufzubauen, wurden Chips des Typs nRF24L01+ von Nordic Semiconductor ausgewählt, die im 2,4 GHz-Band funken. Diese sind günstiger als Module des verbreiteten ZigBee-Protokolls und haben mit maximal 11,3 mA beim Senden eine sehr geringe Leistungsaufnahme, die auf 900 nA im integrierten „Power-Down“-Modus gesenkt werden kann [Nordic Semiconductor 2008]. Das sparsamste XBee-Modul benötigt bis zu 50 mA im aktiven, bzw. 1 μA im „Power-Down“-Modus. Zudem ist die kleinste mögliche Eingangsspannung des nRF24L01+ mit 1,9 V niedriger als die des XBee (2,1 V).[6]

Sensoren Um die Position des Magneten bestimmen zu können, wird für jedes zu detektierende Statusfeld der Türschildes ein Hall-Sensor eingesetzt. Hall-Sensoren liefern ein Signal, wenn sie sich in einem Magnetfeld ausreichender Stärke befinden. Dabei werden zwei Typen von Hall-Sensoren unterschieden: Analoge Hall-Sensoren liefern eine Spannung, die je nach Stärke des Magnetfeldes variiert, digitale Hall-Sensoren geben ein digitales Signal aus, mit dem nur zwischen Präsenz und Abwesenheit eines Magnetfeldes unterschieden werden kann. Um möglichst lange im „Power-Down“-Modus verbringen zu können, kann ein digitaler Hall-Sensor als Interrupt benutzt werden. Somit kann der Mikrocontroller auf unbestimmte Zeit „einschlafen“ und bei Änderung der Magnetposition automatisch erweckt werden. Nach Bestimmung der neuen Position kann diese per Funk übermittelt werden und der Chip sowie das Funkmodul können wieder in den „Power-Down“-Modus übergehen, bis sich die Magnetposition erneut ändert. Bei Verwendung von analogen Hall-Sensoren müsste der Mikrocontroller durch seinen integrierten Timer periodisch erweckt werden, um Messungen durchzuführen. Dies ist weniger energieeffizient

[3] Siehe Fußnote 2
[4] Quelle: https://learn.sparkfun.com/tutorials/reducing-arduino-power-consumption
[5] Siehe Fußnote 4
[6] Quelle: https://en.wikipedia.org/wiki/XBee

und resultiert in einer niedrigeren Aktualisierungsrate.
Nach Betrachtung verschiedener erhältlicher Sensoren wurde sich für den digitalen Hall-Sensor AH3661UA entschieden, da dieser mit einer niedrigen Eingangsspannung von 2,4 V auskommt und im Schnitt lediglich 2,75 μA benötigt. Zudem ist der AH3661UA omnipolar und dank der DIP-Bauweise einfacher zu verwenden als ein SMD-Modell.

Als Energiequelle werden zwei in Serie geschaltete Batterien des Typs Mignon-AA verwendet. Diese sind einfach sowie günstig erhältlich und liefern mit je 1,5 V Spannung insgesamt 3 V an den Arduino und damit genug für alle verbauten Komponenten.
Mit den Informationen über die Leistungsaufnahmen der einzelnen Bauteile kann annäherungsweise berechnet werden, wie lange die maximale Betriebszeit des Türmoduls mit zwei AA-Batterien ist. Dafür werden folgende Annahmen gemacht:

1. Die verwendeten Batterien haben eine Kapazität von $C = 2700\ mAh$.
2. Die Betriebszeit unterteilt sich in zwei scharfe, alternierende Zeitabschnitte: Einen Schlafabschnitt, in der alle Komponenten versuchen so wenig Energie wie möglich zu verbrauchen und einen Wachabschnitt, in dem Mikrocontroller und Funkmodul aktiv sind um Messungen durchzuführen und deren Ergebnisse zu verschicken.
3. Das Funkmodul benötigt den vollständigen Wachabschnitt lang den maximal möglichen Sendestrom.
4. Die summierte Dauer der Wachabschnitte in einem Jahr Betriebszeit wird wie folgt geschätzt: Das Türmodul wird an 255 Tagen im Jahr aktiv genutzt, pro aktivem Tag 20 Mal. Dabei wird das Modul bei jeder Nutzung durchschnittlich für fünf Sekunden aktiv gehalten, dies beinhaltet bereits die maximale Dauer von einigen Sekunden, welche die Software benötigt um vom Master eine neue Netzwerkadresse zugewiesen zu bekommen.[7] Somit ergibt sich eine Gesamtwachzeit von $t_A = 255 \cdot 20 \cdot 5s = 25500\ s \approxeq 7\ h$
5. Nach Punkt 4 beträgt somit die Gesamtschlafzeit $t_S = 8760\ h - 7\ h = 8753\ h$, bei 8760 Stunden im Jahr.

Die Stromaufnahmen der einzelnen Komponenten im Wachzustand (I_{A_x}) und im Schlafzustand (I_{S_x}) sind den obigen Abschnitten nach wie folgt:

1. Mikrocontroller:

$$I_{A_m} = 3,8700\ mA$$
$$I_{S_m} = 0,0043\ mA$$

2. Hall-Sensoren (8 Stück):

$$I_{A_s} = 8 \cdot 0,00275\ \mu A = 0,0220\ mA$$
$$I_{S_s} = 8 \cdot 0,0027\ \mu A = 0,0220\ mA$$

[7] Quelle: http://tmrh20.github.io/RF24Mesh/General-Usage.html

3. nRF24L01+:

$$I_{A_r} = 11,3000\ mA$$
$$I_{S_r} = 0,0009\ mA$$

Die Gesamtströme I_A und I_S betragen somit:

$$I_A = I_{A_m} + I_{A_s} + I_{A_r} = 3,8700\ mA + 0,0220\ mA + 11,3000\ mA = 15,1920\ mA$$
$$I_S = I_{S_m} + I_{S_s} + I_{S_r} = 0,0043\ mA + 0,0220\ mA + 0,0009\ mA = 0,0272\ mA$$

Der durchschnittliche Gesamtstrom I_{avg} ergibt sich folgendermaßen:

$$I_{avg} = \frac{I_A \cdot t_A + I_S \cdot t_S}{t_A + t_S} = \frac{15,1920\ mA \cdot 7\ h + 0,0272\ mA \cdot 8753\ h}{7\ h + 8753\ h} \approxeq 0,0393\ mA$$

Mit der Kapazität der Batterien kann schließlich die Betriebszeit T berechnet werden:

$$T = \frac{2700\ mAh}{0,0393\ mA} \approxeq 68670,85\ h$$

Dies entspricht ca. 7,84 Jahren. Diese geschätzte Betriebszeit kann jedoch nicht ohne weiteres in der Praxis erreicht werden, da die Spannung der Batterien mit der Zeit immer weiter abnimmt. Wird die minimale Betriebsspannung einer der Komponenten unterschritten, ist das Türmodul nicht mehr arbeitsfähig. Die höchste erforderliche Spannung haben mit 2,4 V die Hall-Sensoren. Damit die Kapazität der Batterien trotzdem annähernd voll ausgeschöpft werden kann, muss ein energiesparender Spannungswandler zum Einsatz kommen, der auch die niedrigste Batteriespannung noch auf ein verwendbares Niveau hebt. Ein geeigneter Wandler ist der NCP1402[8] in der Ausführung 30T1. Dieser liefert über das gesamte Spannungsspektrum der Batterien eine konstante Ausgangsspannung von 3 V, bei einem Stromaufnahme von durchschnittlich 42 μA (I_{npc}) und einer Effizienz von 85%. Mit dieser zusätzlichen Komponente erhöhen sich die Gesamtströme auf $I_{A'} = I_A + I_{npc} = 15,1920\ mA + 0,0420\ mA = 15,2340\ mA$ und $I_{S'} = I_S + I_{npc} = 0,0272\ mA + 0,0420\ mA = 0,0692\ mA$. Durch die 15% Verlustleistung des Wandlers reduziert sich die nutzbare Batteriekapazität auf $C' = C \cdot 0,85 = 2700\ mAh \cdot 0,85 = 2295\ mAh$. Diese veränderten Werte ergeben eine neue berechnete Betriebszeit von ca. 3,22 Jahren. Diese beträgt weniger als die Hälfte der theoretischen Betriebszeit, kann aber auch in der Praxis realistisch erreicht werden.

4.2.2.2. Repeatermodule

Um die von den Türmodulen ermittelten Magnetpositionen an den Master zu übermitteln, kommt bei Bedarf ein oder mehrere Repeatermodule zum Einsatz. Diese leiten empfangene Datenpakete an den Master weiter und erhöhen somit die maximale Entfernung von Türmodulen zum Mastermodul.

[8] https://cdn.sparkfun.com/datasheets/BreakoutBoards/NCP1402.pdf

Ein Repeatermodul benötigt lediglich eine Stromversorgung, ein nRF24L01+ Modul, sowie einen kompatiblen Mikrocontroller oder Ein-Platinen-Computer. Hier wurde sich für den Arduino Uno entschieden, da dieser über eine Hohlstecker-Buchse mit einem günstigen 9V-Netzteil betrieben werden kann.

4.2.2.3. Mastermodul

Das Mastermodul ist dafür zuständig alle Daten der Türmodule zu empfangen und diese über Ethernet an den Server weiterzuleiten. Für diese Aufgabe wird ein Raspberry Pi 3 verwendet. Dieser ist durch den vorhandenen Ethernet-Controller und das Linux-Betriebssystem geeignet, um eine Verbindung zu dem Server herzustellen. Zum Empfang der Daten von den Türmodulen benutzt auch der Rasperry Pi ein nRF24L01+ Modul. Die Stromversorgung erfolgt über ein USB-Netzteil.

Kapitel 5

Implementierung

Dieses Kapitel beschreibt die Software-Implementierung der im Entwurf aus Kapitel 4 vorgestellten Komponenten. In Abschnitt 5.1 wird die Software der drei Hardware-Module erläutert. Abschnitt 5.2 befasst sich mit der Umsetzung des Entwurfs im Haskell-Web-Framework Yesod.

5.1. Sensornetzwerk

Dieser Abschnitt erläutert die Implementierungen der drei Komponenten des Sensornetzwerkes: Der Sensor-Node, welcher die Magnetposition bestimmt, der Repeater-Node, welcher Nachrichten zwischen Sensor- und Master-Node weiterleitet, sowie der Master-Node selbst.

5.1.1. Sensor-Node

Arduino-Sketches werden in einem C++-Dialekt entwickelt. Das Arduino-Framework stellt zwei grundlegende Funktionen zur Verfügung, die allen Sketches eine Grundstruktur geben. Die Funktion `setup()` wird einmal bei jeder Ausführung des Sketches automatisch aufgerufen und bietet eine Gelegenheit, um z.B. Variablen zu initialisieren oder GPIO-Pins und andere Peripherie zu konfigurieren. Die Funktion `loop()` wird, nachdem `setup()` abgeschlossen ist, so oft wie möglich wiederholt aufgerufen. Aus ihr wird der Hauptprogrammablauf koordiniert.
Die Software des Sensor-Nodes nutzt die `setup()`-Funktion, um alle acht für das Einlesen von Hall-Sensoren benötigten GPIO-Pins als digitale Eingänge mit aktiviertem Pull-Up-Widerstand zu konfigurieren. Dieser integrierte Widerstand sorgt dafür, dass die Sensoren im inaktiven Zustand einen definierten Spannungspegel am Ausgang liefern, indem dessen Spannung auf den Logikpegel des Mikrocontrollers „hochgezogen“ wird. Weiterhin stellt der Sensor-Node über Funktionen der Bibliothek `RF24Mesh` über seine individuelle, im Code konfigurierte, `NODE ID` eine erste Verbindung zum Mesh-Netzwerk her. Der Inhalt der `loop()`-Funktion stellt eine Implementierung des in Abschnitt 4.2.1.1 beschriebenen Zyklus dar. Zunächst werden über die Arduino-Funktion `digitalRead()` alle Zustände der acht Hall-Sensoren ausgelesen. Im Anschluss wird das Funkmodul, welches zuvor gegebenenfalls in den Energiesparmodus versetzt wurde, mittels `Radio.powerUp()` aktiviert,

sowie eine neue Netzwerkadresse angefordert (`Mesh.renewAddress()`) und die obligatorische `RF24Mesh.update()`-Funktion aufgerufen. Dies müssen für den Betrieb des Mesh-Netzwerkes alle Knoten regelmäßig (einmal pro `loop()`-Durchlauf) tun, damit Nachrichten korrekt weitergeleitet werden können. Über die `RF24Mesh.write()`-Funktion wird nun ein einzelnes Byte an den Master gesendet, das den Wert 0 hat, falls der Magnet des Türschildes von keinem Sensor erkannt wurde, oder die Nummer des Sensors, der den Magnet gemessen hat. Die Nummerierung der Hall-Sensoren beginnt bei 1. Der Master muss nicht explizit adressiert werden, wird in den Parametern von `RF24Mesh.write()` keine Adresse angegeben, wird implizit `0` für den Master-Node angenommen. Als Typ der Nachricht wird `'S'` angegeben, um eine Sensor-Nachricht zu charakterisieren. Nachdem das Messergebnis an den Master gesendet wurde, muss vor Einleitung des „Power-Down"-Modus die Netzwerkadresse des Knoten wieder freigegeben werden (`Mesh.releaseAddress()`), damit die mit ihr verbundenen Pipes in anderen Knoten für neue Verbindungen genutzt werden können. Jetzt kann das Funkmodul wieder über `Radio.powerDown()` in den Energiesparmodus versetzt werden. Der gesamte Mikrocontroller geht anschließend in den „Power Down"-Modus über. Damit er aus diesem wieder erweckt werden kann, wird zuvor mit der Bibliothek `PinChangeInterrupt` für jeden Sensor-Pin ein Interrupt konfiguriert, der bei wechselnder Flanke des Signals ausgelöst wird, also wenn sich das Signal eines der Hall-Sensoren ändert. Dies ist insbesondere dann der Fall, wenn der Magnet von einem Feld auf ein anderes bewegt oder komplett entfernt wird. Löst einer der Sensoren einem Interrupt aus, erwacht der Sensor-Node automatisch aus dem „Power-Down"-Modus und startet eine definierte Interrupt-Routine. Diese deaktiviert zunächst alle zuvor konfigurierten Interrupts, damit weitere Magnetbewegungen den Programmablauf nicht stören, solange der Mikrocontroller noch nicht wieder in den „Power-Down"-Modus versetzt wurde. Anschließend wird der oben beschriebene `loop()`-Zyklus erneut durchlaufen.

5.1.2. Repeater-Node

Der Repeater-Node führt in der `setup()`-Funktion die selbe Initialisierung für das Mesh-Netzwerk durch wie der Sensor-Node. Die `loop()`-Funktion besteht auschließlich aus einem Aufruf von `Mesh.update()`, dies genügt, damit der Repeater-Node kontinuierlich Nachrichten im Netzwerk weiterleitet.

5.1.3. Master-Node

Die Software des Master-Nodes wurde ebenfalls in C++ entwickelt. Die Main-Funktion konfiguriert zunächst `RF24Mesh`. Die `NODE ID` ist hier `0`, da dies den Master des Netzwerkes definiert. Anschließend wird mittels der Bibliothek `restclient-cpp`[1] eine REST-Verbindung zum Server festgelegt. Hierfür wird eine HTTP-Basic-Authentifizierung verwendet. Nach dieser Initialisierungsphase beginnt eine Schleife, welche zunächst die `Mesh.update()`-

[1] https://github.com/mrtazz/restclient-cpp

Funktion aufruft. Für die Rolle als Master ist anschließend der Aufruf von `Mesh.DHCP()` entscheidend. Diese Funktion sorgt dafür, dass die im Netzwerk vorhandenen aktiven Knoten jederzeit eine gültige Netzwerkadresse zugewiesen bekommen. Hierfür wird eine Datei `dhcp.txt` gepflegt, die Zuordnungen von `NODE ID`s zu Netzwerkadressen enthält. Die Netzwerkadresse einer `NODE ID` kann sich bei jedem Aufruf von `Mesh.DHCP()` ändern, wenn damit eine günstigere Netztopologie hergestellt werden kann (siehe Abschnitt 4.2.1.1). Anschließend wird, wenn empfangene Daten vorliegen (ermittelt über `Network.available()`), mittels `Network.peek()` ein `RF24NetworkHeader` ausgelesen, der Aufschluss über den Nachrichtentyp sowie die Absender-ID gibt. Ist der Typ der Nachricht `'S'`, so handelt es sich um eine neue Magnetposition eines Türmoduls. In diesem Fall wird mittels der `post`-Methode des zu Beginn erzeugten `RestClient::Connection`-Objektes der `restclient-cpp`-Bibliothek eine autorisierte HTTP-POST-Anfrage an den Server geschickt. Die neue Magnetpostion sowie die zugehörige `NODE ID` werden dabei als URL-Parameter übergeben.

5.2. Webanwendung

Die Webanwendung des Anwesenheitsmonitoringsystems wurde in Yesod entwickelt. Wie in Abschnitt 3.4 an Minimalbeispielen demonstriert, werden in diesem Haskell-Framework verschiedene DSLs für Routen, Datenbank-Entitäten und View-Templates verwendet. Diese werden im Folgenden genutzt, um zusammen mit Controller-Implementierungen in Haskell den Entwurf aus Abschnitt 4.2.1.2 im Model-View-Controller-Muster umzusetzen.

5.2.1. Model

```
1  DoorElement
2      ownerName         Text
3      ownerTitle        Text
4      ownerRoom         Text
5      ownerImgUrl       Text
6      ownerPw           Text
7      lastStatusCode    Int
8      lastStatusUpdate  UTCTime
9      statusDescs       StatusDescriptions
```

Listing 5.1. Definition der Persistent-Entität eines Türelements auf Basis des Entity-Relationship-Modells aus Abbildung 4.3.

Zunächst wird das in Abbildung 4.3 dargestellte Entity-Relationship-Modell, welches die Daten der Nutzerin oder des Nutzers eines Türmoduls, sowie den aktuellen Status enthält, in die Persistent-DSL übertragen. Das Ergebnis ist in Listing 5.1 zu sehen. Die `DoorElement`-Entität und deren Attribute wurden hier direkt übernommen und um passende Haskell-Datentypen erweitert. Auf ein explizites Schlüsselattribut kann verzichtet werden, da

Persistent dieses automatisch anlegt. Der Status-Code wird, wie in anderen Teilen des Systems auch, als einfache Zahl repräsentiert, in diesem Fall als Integer. Dies lässt die Möglichkeit offen, einen negativen Status-Code zu verwenden, um zu symbolisieren, dass der Sensor-Node dieses `DoorElements` eine gewisse Zeit keine neuen Nachrichten mehr versendet hat und möglicherweise ausgefallen ist.

```
1    type StatusType = Int
2    type StatusDescriptions = [(Int, (Text, StatusType))]
```

Listing 5.2. Der `StatusDescriptions` Datentyp, welcher für das `statusDescs`-Attribut der `DoorElement`-Entität verwendet wird.

Die Entität `StatusDescription` wurde in die `DoorElement`-Entität mit aufgenommen, für sie wurde ein eigener Haskell-Datentyp definiert, siehe Listing 5.2. Für den als Integer angegebenen Statustyp wird zunächst ein einfacher Alias `StatusType` erstellt. Als mögliche Werte von `StatusType` sind `0`, `1` und `2` vorgesehen, welche angeben, dass der Status-Code und die zugehörige Beschreibung einen Anwesenheitszustand bezeichnen, bei dem die Nutzerin oder der Nutzer komplett abwesend ist (`0`), im Büro anwesend (`1`), oder sich im Hause, aber nicht im Büro befindet (`2`). Diese drei Klassen von Anwesenheitszuständen resultieren später in der View in einer deutlich sichtbaren Einfärbung: grün für `1`, gelb für `2` sowie rot für `0`. Damit `StatusDescription` in `DoorElement` aufgenommen werden kann, wird der Typ `StatusDescriptions` definiert. Dieser ist eine Liste von Tupeln, welche einem Status-Code en weiteres Tupel zuordnet, das aus einer Beschreibung dieses Status-Codes, sowie dessen Typ besteht.

Alle Daten der `DoorElements` der einzelnen Nutzerinnen und Nutzer, abgesehen von `lastStatusCode` und `lastStatusUpdate`, die regelmäßig vom Master-Node zu aktualisieren sind, werden bei jedem Start der Anwendung aus Konfigurationsdateien ausgelesen. Für jedes `DoorElement`, das im System verwendet werden soll, muss eine Konfigurationsdatei existieren, deren Dateiname die `NODE ID` des Hardware-Türmoduls ist und deren Dateiendung „`.cfg`“ lautet. Die Webanwendung liest beim Start mit der Funktion `loadConfigs` in `Application.hs` alle im Ordner `config/doorElements/` gespeicherten Konfigurationsdateien mit dieser Dateiendung ein und legt für jede einen `DoorElement`-Record an, dessen Attribute entsprechend der Felder der jeweiligen Konfigurationsdatei initialisiert werden. Somit können neue Nutzer des Systems nur durch das Erstellen einer entsprechenden Konfigurationsdatei in Verbindung mit einem Serverneustart angelegt werden. Genauso können Nutzer durch das Löschen ihrer Konfigurationsdateien aus dem System entfernt werden, da die `DoorElement`-Tabelle bei jedem Anwendungsstart geleert wird.

5.2.2. View

Der HTML-Anteil der View-Implementierung besteht aus drei `.hamlet`-Template-Dateien, sowie zwei zusätzlichen Dateien `default-layout.hamlet` und `default-layout-wrapper.hamlet`,

welche ein Grundgerüst jeder Seite vordefinieren.

homepage.hamlet	Übersicht aller Türelemente
setstatus.hamlet	Manuelle Auswahl der zur Verfügung stehenden Zustände
login.hamlet	Login-Widget für die Authentifizierung

Zusätzlich gibt es zwei `.lucius`-Dateien für das CSS-Styling der `.hamlet`-Templates. Auch für `default-layout.hamlet` gibt es eine vordefinierte Datei `default-layout.lucius`.

homepage.lucius	CSS-Styles für `homepage.hamlet`
login.lucius	CSS-Styles für `login.hamlet`

Die Übersicht aller Zustände in `homepage.hamlet` ist aufgebaut aus zwei übergeordneten Komponenten: Einer `navbar`, die am oberen Rand einen Titel, sowie den Login- bzw. Logout-Button darstellt und einem Div-Container, in dem sich ein Bootstrap Grid mit vier Spalten befindet. Das dynamische Befüllen dieses Grids übernimmt in der Hamlet-Template-Sprache eingebetteter Haskell-Code, wie in Listing 5.3 zu sehen ist:

```
1       $forall Entity id elem <- allDoorElems
2         <div .col-md-3>
3           <div .card>
```

Listing 5.3. Benutzung von Hamlets `forall`-Konstrukt.

Dabei ist `allDoorElems` das Ergebnis einer Datenbankabfrage im Controller. Dieses kann in Hamlet direkt verwendet werden. Für jedes Türelement, das die Anfrage zurückgeliefert hat, wird nun eine Bootstrap-Spalte der Breite drei erzeugt, in der alle Informationen des Türelements, wie in Abbildung 4.5 skizziert, dargestellt werden. Die Breite ist hier drei, da das Bootstrap-Grid zwölf Spalten hat und insgesamt vier Türelement-Karten in einer Zeile angezeigt werden sollen.
Mittels des `case`-Konstrukts von Hamlet wird zwischen den drei verschiedenen Status-Typen unterschieden und die Beschreibung sowie der Hintergrund entweder grün, gelb oder rot eingefärbt.
Die aktuelle Statusbeschreibung wird als Hyperlink eingebaut, der mittels Hamlets URL-Interpolation zur Oberfläche für die manuelle Statusauswahl führt (`setstatus.hamlet`).
Die von dem Template erzeugte HTML-Oberfläche ist in Abbildung 5.1 zu sehen.

`setstatus.hamlet` dient der manuellen Auswahl des aktiven Status eines Türelements, als Alternative zur Benutzung des Hardware-Türmoduls. Das Template ist ähnlich aufgebaut wie `homepage.hamlet`, mit dem Unterschied, dass hier statt Karten Buttons für jeden möglichen Status des betreffenden Türlelements dargestellt werden. Dies geschieht, analog zum Hardware-Türmodul, in einem einspaltigen Grid (siehe Abbildung 5.2).

Das in `login.hamlet` definierte Login-Widget nutzt ein POST-Formular, um die eingegebene Türelement-ID, die gleichzeitig der `Node ID` des zugehörigen Hardware-Moduls entspricht, und das Passwort der Nutzerin oder des Nutzers an die in Yesod integrierte Login-Route zu senden.

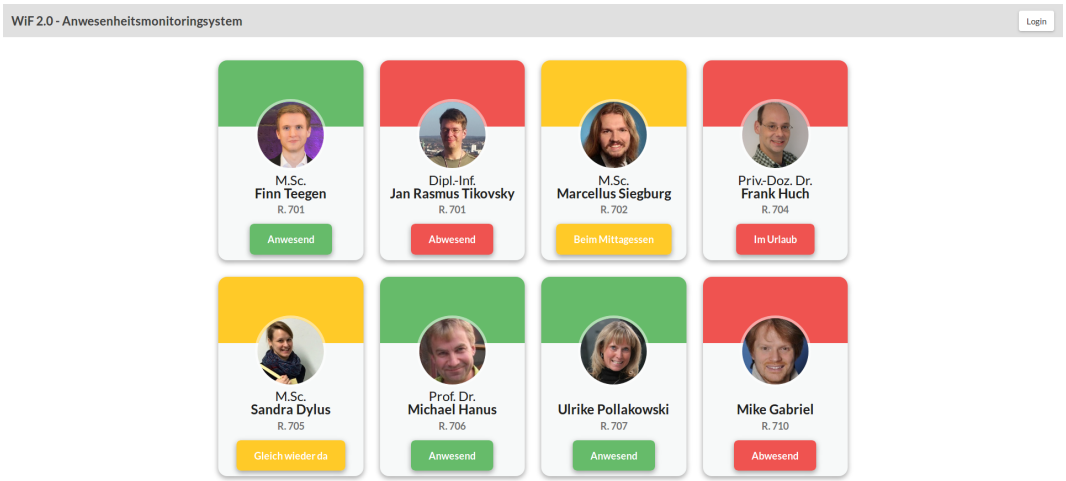


Abbildung 5.1. Von `homepage.hamlet` erzeugte Oberfläche.

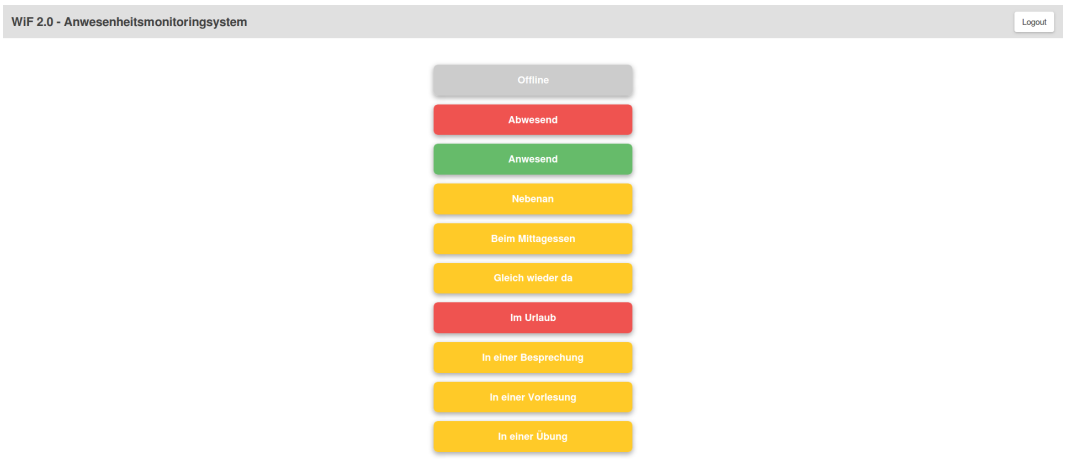


Abbildung 5.2. Von `setstatus.hamlet` erzeugte Oberfläche.

5.2.3. Controller

Die Controller-Implementierung teilt sich auf in eine deklarative Definition der angebotenen REST-Routen in der Datei `routes` (siehe Listing 5.4) und der Haskell-Implementierung der einzelnen Handler für jede Route. Dabei gibt die Datei `routes` an, welche URL welche HTTP-Methoden unterstützt und wie die zugeordneten Handler benannt sein müssen. Für die Indexroute „/“ ist z.B. nur die GET-Methode möglich, der Basisname für alle Handler,

in diesem Fall nur einer für GET, dieser URL ist `HomeR`. Der spezielle Handler für die GET-Methode muss nun `getHomeR` heißen.

```
1  /                                HomeR       GET
2
3  /status/#DoorElementId/#Int      StatusR     POST
4  /status/#DoorElementId           SetStatusR  GET
5
6  /api/status/#DoorElementId/#Int  StatusAPIR  POST
```

Listing 5.4. Definition der angebotenen REST-Routen der Webanwendung.

getHomeR

`getHomeR` ist für die Übersichtsseite zuständig und ist ein Handler mit einem HTML-Rückgabewert. Er lädt zunächst alle `DoorElements` aus der Datenbank und sortiert sie nach der Raumbezeichnung. Auf diese mit `allDoorElems` bezeichnete Liste kann das Hamlet-Template `homepage.hamlet` für die Darstellung zugreifen. Die Funktion `maybeAuthId` liefert ein Maybe-Objekt, das falls die Nutzerin oder der Nutzer eingeloggt ist, die entsprechenden Authentifizierungsinformationen enthält. Dieses Objekt wird als `maid` im Template referenziert, um entweder einen Login oder Logout-Button anzuzeigen.
Zum Schluss wird über die Funktion `defaultLayout` der Titel der vom Handler zurückzuliefernden Webseite gesetzt und mit `widgetFile` das Template `homepage.hamlet` geladen, gerendert und als Ergebnis zurückgegeben.

postStatusR

`StatusR` ist in `Foundation.hs` als autorisierte Route deklariert, somit können nur eingeloggte Nutzer, deren Session-ID als Cookie gespeichert wurde, auf diese Route zugreifen.
Der Handler hat gemäß der URL-Definition in `routes` zwei Funktionsparameter vom Typ `DoorElementId` und `Int` für die ID des Türelements dessen Status-Code geändert werden soll und den neuen Status-Code selbst.
Der Handler ruft zunächst die aktuelle Systemzeit ab und aktualisiert im Datenbank-Record mit der `DoorElementId` des Parameters das Feld `lastStatusUpdate` auf diese Zeit und das Feld `lastStatusCode` auf den ebenfalls als Parameter übergebenen Wert des neuen Status-Codes.
Zum Schluss wird mittels `redirect` auf die Übersichtsseite weitergeleitet.

postStatusAPIR

`postStatusAPIR` hat die selbe Funktion wie `postStatusR`, ist aber für die Nutzung durch den Raspberry Pi gedacht, statt durch die Weboberfläche. Die Aufteilung in zwei Routen wurde vorgenommen, da die Webanwendung eine Authentifizierung über Cookies nutzt,

für den Raspberry Pi jedoch eine Basic-Authentifizierung mit Benutzername und Passwort sinnvoller ist. Diese Daten sind auf dem Raspberry Pi fest hinterlegt und werden an die `postStatusAPIR`-Route im POST-Body mitgesendet. Der Handler gleicht die Daten mit den in der Webanwendung gespeicherten ab und im Erfolgsfall verhält sich der Handler anschließen identisch zu `postStatusR`, mit der Ausnahme dass kein Redirect erfolgt, da dieser für den Raspberry Pi nicht nötig ist.

getSetStatusR

`getSetStatusR` soll die Oberfläche zur manuellen Statusauswahl erzeugen und zurückliefern. Dafür bekommt der Handler die `DoorElementId` des Türelements, für das eine Auswahl erfolgen soll, übergeben.
Der Handler lädt über die `get404`-Funktion, welche im Erfolgsfall einen Datenbank-Record zurückgibt und im Fehlerfall direkt eine 404-Fehlerseite darstellt, das zur `DoorElementId` gehörige `DoorElement` und rendert über `widgetFile` das Template `setstatus.hamlet`.

Kapitel 6

Zukünftige Entwicklung

Die durch das entwickelte Soft- und Hardwaresystem bereitgestellte Funktionalität entspricht den zuvor gestellten Anforderungen (siehe Abschnitt 4.1). Allerdings wurden einige sinnvolle Erweiterungen, die nicht explizit als Anforderung festgelegt wurden, aus Zeitgründen nicht umgesetzt und könnten daher Bestandteil weiterführender Arbeiten sein.

Eine erste Erweiterung wäre das Ersetzen der aktuell zum Einsatz kommenden Konfigurationsdateien durch entsprechende Oberflächen in der Webanwendung. Die Konfigurationsdateien wurden als schnell zu realisierende Möglichkeit zum Hinzufügen und Löschen von Türmodulen und Bearbeiten deren Daten benutzt. Auch die individuellen Statusbeschreibungen und das persönliche Passwort einer Nutzerin oder eines Nutzers müssen in der entsprechenden Datei konfiguriert sein. Praktisch wäre statt der Konfigurationsdateien eine Weboberfläche zur Registrierung neuer Nutzer, zur Änderung des Passworts, sowie aller anderen persönlichen Daten, inklusive der einzelnen Statusbeschreibungen sowie eine Möglichkeit das eigene Türmodul zu löschen. Ein hiervon separater Administratorbereich zur Verwaltung aller Module wäre dann ebenfalls sinnvoll.
Zudem werden die Profilbilder der einzelnen Nutzerinnen und Nutzer derzeit über Links von anderen Webservern geladen, auf Dauer zuverlässiger wäre eine eigene Upload-Funktionalität.

In den HTML-Templates der Webanwendung ist implementiert, dass bei einem Status-Code von `-1`, anstatt der sonst verwendeten Farben grün, gelb und rot, das virtuelle Türmodul grau eingefärbt wird, um zu signalisieren, dass das zugehörige Hardware-Modul nicht aktiv ist. Eine mögliche Erweiterung des Systems könnte sein, dass Türmodule statt durch einfache Trennung der Stromversorgung kontrolliert durch einen Tastendruck abgeschaltet werden können, wobei dies dem Master signalisiert würde, der zunächst die für dieses Modul reservierte Netzwerkadresse freigeben und anschließen dem Webserver über das Setzen des Status-Codes auf `-1` mitteilen könnte, dass das Modul nun inaktiv ist.
Ebenfalls denkbar wäre eine Überwachung der Batteriespannungen an den Türmodulen, diese könnten vor einem Ausfall ihren Status auf „Offline“ setzen, damit Batterien schnell und zielgerichtet ausgetauscht werden können.

Auch wenn das Schadenspotential bei einer vollständigen oder teilweisen Kompromittierung des Systems als gering einzuschätzen ist, wurden einige Sicherheitsmechanismen

implementiert, die jedoch ausbaufähig sind. So werden zur Verhinderung von Session Hijacking Cookies nur über SSL übertragen. Der Master-Node muss sich bei jeder Verbindung zur Webanwendung über SSL mittels Basic-Authentifizierung ausweisen.
Nicht ideal ist die Speicherung der Nutzerpasswörter in Konfigurationsdateien der Webanwendung. Besser wäre das Speichern eines „gesalzenen“ Passwort-Hashes in einer Datenbank.
Die Funkverbindung der Türmodule zum Master findet unverschlüsselt statt. Mittels eines kompatiblen Funkmoduls könnten somit Nachrichten eines im System vorhandenen Türmoduls gefälscht und dadurch dessen Status in der Webanwendung überschrieben werden. Zur Verhinderung eines solchen Angriffes wäre es notwendig eine Art Verschlüsselung der Datenpakete zu implementieren.

Zum Einsatz von mehreren Instanzen des Anwesenheitsmonitoringsystems, die innerhalb der Funkreichweite des jeweils anderen operieren sollen, wäre es notwendig für jedes System unterschiedliche Funkkanäle für die NRF24L01+-Module zu konfigurieren.

Kapitel 7

Fazit

Mit dem hier entwickelten und vorgestellten Anwesenheitsmonitoringsystem ist es einfach möglich, den Anwesenheitszustand von Mitgliedern einer Gruppe systematisch zu erfassen und mittels einer webbasierten Anwendung übersichtlich darzustellen. Die Nutzerinteraktion ist dank den Hardwarekomponenten des Systems besonders intuitiv und praktisch gestaltet, da bereits bestehende Magnettafeln an den Türen zur Statusangabe genutzt werden können.
Man kann sagen, dass das entwickelte System alle im Vorfeld aufgestellten Anforderungen erfüllt. Durch die Auswahl von stromsparenden Hardwarekomponenten und den Aufbau eines eigenen Funk-Sensornetzes können die Türmodule lange Zeit ohne Netzstrom betrieben werden und sind unabhängig von bestehenden Netzwerken. Die Benutzeroberfläche der Webanwendung ist dank der Verwendung von Bootstrap sowohl optisch ansprechend, als auch platzeffizient und variabel in der Darstellung. Zudem wurden alle funktionalen Anforderungen in der Webanwendung implementiert. So gibt es beispielsweise eine Authentifizierung für Nutzerinnen und Nutzer, sowie damit verbunden die Möglichkeit den Anwesenheitszustand über die Oberfläche der Webanwendung, statt über das Türschild zu verändern. Die Konfiguration und Verwaltung von Türmodulen ist über Konfigurationsdateien eher zweckmäßig gestaltet.
Die aus der nicht-funktionalen Anforderung, Haskell zur Entwicklung der Webanwendung zu verwenden, heraus getroffene Wahl des Webframeworks Yesod erwies sich, genauso wie eigentlich alle in Kapitel 3 vorgestellten Technologien – als richtig. Yesod und Haskell konnten in der Entwicklung vor allem durch ihre Typsicherheit – selbst in den Routendefinitionen – und die einfachen sowie ausdrucksstarken Templatesprachen überzeugen.
Mit der Bibliothek RF24Mesh war eine problemlose Implementierung eines vermaschten Sensornetzwerkes möglich, bei dem Knoten jederzeit ohne manuelle Rekonfiguration der Topologie hinzugefügt oder entfernt werden können.
Die Arduino-Plattform bot ein umfangreiches und zugleich erschwingliches Ökosystem zur Konzipierung und Entwicklung der Türmodule.

Anhang A

Installation und Betrieb der Webanwendung

Im Folgenden wird erläutert, wie die in dieser Arbeit entwickelte und präsentierte Webanwendung installiert werden kann.
Für die Entwicklung und das Testen wurde ein Ubuntu 16.04.3 LTS System verwendet.

A.1. Installation

Zunächst sollten alle Pakete und Paketlisten auf den neuesten Stand gebracht werden:

```
sudo apt-get update
sudo apt-get upgrade
```

Anschließend können das Yesod-Framework, sowie die Haskell-Umgebung installiert werden:

```
sudo apt-get install yesod
wget -qO- https://get.haskellstack.org/ | sh
stack install yesod-bin
```

Der zweite Befehl sorgt dafür, dass Stack in der neuesten Version installiert wird, da es mit der Älteren in den Ubuntu-Repositorys später Probleme geben kann.
Jetzt wird die Webanwendung aus dem Repository der Arbeitsgruppe mittels Git heruntergeladen:

```
git clone https://git.ps.informatik.uni-kiel.de/theses/2017/2017-bkrause-ba.
    git
```

Von hier aus wird in das Verzeichnis der Webanwendung gewechselt:

```
cd 2017-bkrause-ba/backend/wif2/
```

Jetzt kann die Anwendung gebaut werden:

```
stack build
```

A.2. Betrieb

Ein Testserver wird folgendermaßen gestartet:

```
stack exec -- yesod devel
```

Nun sollte die Webanwendung unter `https://localhost:3443/` verfügbar sein.
Die Hauptseite der Anwendung stellt die Zustände aller konfigurierten Türmodule dar, Benutzer (Besitzer eines Türmoduls) können sich über den Login-Button oben rechts, oder mit einem Klick auf die aktuell angezeigte Statusbeschreibung ihres Moduls anmelden, in letzterem Fall wird der Benutzer nach erfolgreichem Login direkt auf die Seite zur Änderung des aktuellen Status weitergeleitet.
Anmelden kann sich jeder Benutzer mit seiner Türelement-ID, sowie dem in der zugehörigen Konfigurationsdatei hinterlegten Passwort (siehe Anhang B).

A.3. Deployment mit Keter

Mit dem folgenden Befehl kann ein Keter[1]-Bundle der Webanwendung erzeugt werden:

```
yesod keter
```

Dieses ist als `wif2.keter` im Hauptverzeichnis der Anwendung zu finden.
Auf einem Ubuntu-basierten System kann Keter wie folgt installiert werden:

```
sudo apt-key adv --keyserver keyserver.ubuntu.com --recv-keys 575159689BEFB442
echo "deb http://download.fpcomplete.com/ubuntu \"$(lsb_release -sc)\" main"|
    sudo tee /etc/apt/sources.list.d/fpco.list

sudo apt-get update
sudo apt-get -y install postgresql stack zlib1g-dev

stack update
stack setup
stack install keter

sudo mkdir -p /opt/keter/bin
sudo cp ~/.local/bin/keter /opt/keter/bin

sudo mkdir -p /opt/keter/etc
```

[1] https://github.com/snoyberg/keter

Anschließend muss eine Konfigurationsdatei `/opt/keter/etc/keter-config.yaml` erstellt werden[2].
Keter kann anschließend gestartet werden:

```
/opt/keter/bin/keter /opt/keter/etc/keter-config.yaml
```

Empfehlenswert ist das Anlegen eines Daemons[3].
Für das Deployen genügt es nun, das zuvor erstellte Bundle `wif2.keter` in das Verzeichnis `/opt/keter/incoming` zu kopieren. Keter überwacht diesen Ordner permanent und wird die Anwendung direkt ausführen.

[2]Beipsiel: https://github.com/snoyberg/keter/blob/master/etc/keter-config.yaml
[3]Siehe: https://github.com/snoyberg/keter

Anhang B

Verwaltung der Webanwendung

Die Webanwendung startet das erste Mal mit einer leeren Oberfläche, wenn noch nichts konfiguriert wurde. Um sie sinnvoll zu benutzen, müssen zunächst Konfigurationsdateien erstellt werden, die unter `backend/wif2/config/` abzulegen sind. Dort befindet sich auch eine Beispieldatei (`0.cfg.example`), die kopiert und angepasst werden kann.
Jede Konfigurationsdatei bezieht sich auf ein Türelement und hat über die sogenannte Türelement-ID eine direkte Beziehung zu einem Sensor-Modul, dessen Node-ID (siehe Anhang C) zu dieser identisch sein muss, damit der Status des Sensor-Moduls automatisch auf den Zustand des virtuellen Türelements gemappt werden kann.
Die Türelement-ID wird durch den Namen der Konfigurationsdatei (ohne Endung) definiert und muss eine natürliche Zahl zwischen 1 und 255 (technische Einschränkung der Funkmodule) sein, die bezogen auf alle anderen Konfigurationsdateien eindeutig ist. Sinnvollerweise versucht man hier ein bestehendes Nummerierungsschema aus der realen Welt zu übernehmen, beispielsweise Raumnummern, damit man einfach die gesuchte Datei findet und jeder Benutzer sich leicht seine individuelle Türelement-ID merken kann.
Die Anwendung liest bei jedem Start alle Konfigurationsdateien ein, die unter `backend/wif2/config/` gespeichert sind und die Dateiendung `.cfg` besitzen. Jede korrekt eingelesene Datei resultiert in der Darstellung eines virtuellen Türmoduls in der Weboberfläche, das die in der Datei konfigurierten Daten anzeigt und zunächst den Status „Abwesend“ hat. Dieser kann manuell über die Webanwendung geändert werden (siehe Abschnitt A.2) oder wird regelmäßig automatisch auf den Status des zugehörigen Hardware-Türmoduls gesetzt, sofern eines mit der selben ID konfiguriert und installiert ist.
Die einzelnen Felder der Konfigurationsdateien, die vorhanden sein müssen, sind wie folgt (der Beispieldatei entnommen):

Die Datei beginnt mit der Überschrift

```
[OWNER]
```

In dem darauf folgenden Bereich werden Informationen über den Besitzer des Türmoduls eingetragen.
Name und akademischer Titel (falls vorhanden, ansonsten leer oder beliebige Bezeichnung) des Besitzers:

```
name= Max Mustermann
```

```
title= M.Sc.
```

Raumbezeichnung:

```
room= R. 700
```

Eine URL zu einem (möglichst quadratischen) Bild des Besitzers:

```
imgUrl= https://images.pexels.com/photos/7974/pexels-photo.jpg
```

Anschließen beginnt der Bereich

```
[STATUS]
```

Hier werden die einzelnen Statusbeschreibungen, die in der Weboberfläche für diesen Benutzer angezeigt werden sollen, definiert, sowie eine Einteilung des Statustyps in eine der folgenden drei Klassen:

present	bedeutet „im Büro anwesend“, resultiert in grüner Einfärbung
otp	bedeutet „im Hause“ (on the premises), resultiert in gelber Einfärbung
absent	bedeutet „abwesend“ (z.B. im Urlaub), resultiert in roter Einfärbung

Es müssen insgesamt acht verschiedene Beschreibungen und deren Typ festgelegt sein, sollten so viele nicht benötigt werden, können Beschreibungen leer bleiben. Deren Typ kann dann beispielsweise auf „absent“ gesetzt werden.
Die acht Definitionen von Statusbeschreibungen und Statustypen haben folgende Form:

```
statusX= Anwesend
statusXType= present
```

mit $X \in \{1, 2, 3, 4, 5, 6, 7, 8\}$.
Es existiert ein impliziter neunter Status, der nicht konfiguriert werden kann, als „Abwesend“ festgelegt ist und von den Türmodulen an die Webanwendung übermittelt wird, wenn der Magnet keinen Hall-Sensor aktiviert, zum Beispiel weil er auf ein „Abwesend“ Feld gezogen wurde, das einen gewissen Abstand zu allen Hall-Sensoren hat.

Anhang C

Installation von Türmodulen

Wenn ein neuer Sensor-Node in das Netzwerk integriert werden soll, muss der zugehörige Arduino mit der Datei `client/client.ino` geflasht werden. Hierfür wird die Arduino-IDE[1] und ein USB-Serial-Converter zur Programmierung über USB benötigt.
Vor dem Flash-Vorgang muss die `client.ino`-Datei bearbeitet werden. An folgender Stelle ist für *X* die Türelement-ID einzutragen, die zuvor entsprechend Anhang B auf dem Server der Webanwendung festgelegt wurde:

```
// --- NODE ID ---

#define nodeID X

// --------------
```

Anschließend kann die bearbeitete Datei mittels der Arduino-IDE auf den Arduino übertragen werden. Hierbei ist es wichtig, dass das Modul mittels der Batterien während des gesamten Vorgangs mit Strom versorgt wird. Das Modul ist danach sofort einsatzbereit und wird versuchen sich mit dem Master-Node zu verbinden. Es ist jedoch ratsam den Master-Node erst zum Schluss, wenn alle Türmodule einsatzbereit und an ihrem finalen Platz angebracht sind, einzuschalten, damit die DHCP-Tabelle des Masters keine „toten" Netzwerkadressen beinhaltet, die entstehen würden, wenn der Master eine Adresszuweisung, aber nicht deren Freigabe mitbekommt, zum Beispiel durch ungünstiges Bewegen der Sensor-Nodes oder zwischenzeitliches Abschalten.

[1] https://www.arduino.cc/en/main/software

Anhang D

Installation des Master-Nodes

Die Software des Master-Nodes wurde auf einem Raspberry Pi 3 Model B V1.2 mit Raspbian Stretch Lite in Kernelversion 4.9 vom 29.11.2017 getestet.
Es wird davon ausgegangen, dass der Raspberry Pi bereits korrekt mit einem nRF24L01+-Funkmodul entsprechend seiner Pinbelegung[1] und der Beschriftung der Pins des Funkmoduls verkabelt wurde (hierbei bezieht sich CSN des Funkmoduls auf BCM 8 und CE auf BCM 25 des Raspberry Pis). Außerdem muss der Raspberry Pi über ein Crossover-Kabel mit dem Server verbunden sein, auf dem entsprechend Anhang A die Webanwendung läuft. Der Server muss so konfiguriert sein, dass er eine Verbindung des Raspberry Pis akzeptiert.
Zunächst sollten alle Pakete und Paketlisten auf den neuesten Stand gebracht werden:

```
sudo apt-get update
sudo apt-get upgrade
```

Jetzt muss über das grafische Konfigurationstool des Betriebssystems unter „Interfacing-Options -> SPI“ SPI aktiviert werden:

```
sudo raspi-config
```

Jetzt können die RF24-Bibliotheken installiert werden, dabei sind nur RF24, RF24Mesh und RF24Network notwendig:

```
wget http://tmrh20.github.io/RF24Installer/RPi/install.sh
chmod +x install.sh
./install.sh
```

Weiterhin wird die C++ Bibliothek restclient-cpp benötigt, die folgendermaßen installiert werden kann:

```
git clone https://github.com/mrtazz/restclient-cpp.git
cd restclient-cpp
./autogen.sh
./configure
make install
```

[1] https://de.pinout.xyz/

Gegebenenfalls müssen bei entsprechenden Fehlermeldungen zunächst Abhängigkeiten wie libcurl installiert werden, die sich in den Ubuntu-Repositories befinden.
Den Quellcode für die Master-Software kann aus dem Git-Repository entnommen werden:

```
git clone https://git.ps.informatik.uni-kiel.de/theses/2017/2017-bkrause-ba.git
```

Von hier aus wird in das Verzeichnis des Masters/Gateways gewechselt:

```
cd 2017-bkrause-ba/master_node/
```

Jetzt kann die Software gebaut werden:

```
make install
```

Folgendermaßen wird die Software gestartet:

```
sudo ./master
```

Hierbei sollten einige Informationen über das Funkmodul ausgegeben werden, wird als „Model“ nicht „nRF24L01+“ angezeigt, oder sind einige Werte null oder „PA Power“ wird nicht als „PA_MAX“ angegeben, deutet dies auf ein falsch verkabeltes oder defektes Funkmodul hin.

Literaturverzeichnis

[Banzi 2009] M. Banzi. Getting Started with Arduino. O'Reilly Media, Feb. 2009. (Siehe Seiten 7, 8)

[Dembowski 2013] K. Dembowski. Raspberry Pi - Das Handbuch. Springer-Verlag, Aug. 2013. (Siehe Seite 9)

[Engelhardt 2016] E. F. Engelhardt. Schnelleinstieg Raspberry Pi 3. Franzis Verlag, Juni 2016. (Siehe Seite 9)

[Hughes 2016] J. M. Hughes. Arduino: A Technical Reference. O'Reilly Media, Mai 2016. (Siehe Seite 7)

[Nordic Semiconductor 2008] Nordic Semiconductor. nRF24L01+ Product Specification. 1.0. Sep. 2008. (Siehe Seiten 8, 27)

[O'Sullivan u. a. 2008] B. O'Sullivan, J. Goerzen und D. B. Stewart. Real World Haskell. O'Reilly Media, Nov. 2008. (Siehe Seite 10)

[Robertson und Robertson 2012] S. Robertson und J. Robertson. Mastering the Requirements Process. Addison-Wesley, Aug. 2012. (Siehe Seiten 15, 16)

[Snoyman 2012] M. Snoyman. Developing Web Applications with Haskell and Yesod. O'Reilly Media, Apr. 2012. (Siehe Seiten 11, 12)

[Spurlock 2013] J. Spurlock. Bootstrap. O'Reilly Media, Mai 2013. (Siehe Seite 13)