Westfälische Hochschule

Gelsenkirchen Bocholt Recklinghausen
University of Applied Sciences

Keyless Entry

Projektdokumentation

Modul: Internetanwendungen 2

Studiengang: Informatik.Softwaresysteme

Westfälische Hochschule Gelsenkirchen,

Standort Bocholt

Wintersemester 2014/15

Projektmitglieder: Thomas Bohn, Sven Ehmer, Fabian Künne,
Timo Matuszewski, Jan Rathmer

Dozent: Prof. Dr. Martin Schulten

Inhaltsverzeichnis

1 Einleitung

In den folgenden Abschnitten wird die Projektarbeit „Keyless Entry“ vorgestellt. Es werden verschiedene Themenbereiche des Projekts beleuchtet, die zur Lösung der Projektaufgabe geführt haben. Dabei wird auf die Anforderungen an das Projekt, die technische Umsetzung, sowie den Entwicklungsprozess der Projektarbeit eingegangen.

Dieses Projekt entstand an der Westfälischen Hochschule Gelsenkirchen, Standort Bocholt im Rahmen des Wintersemesters 2014/15 im Modul Internetanwendungen 2. Die Gruppenteilnehmer sind Thomas Bohn, Sven Ehmer, Fabian Künne, Timo Matuszewski und Jan Rathmer und studieren alle im 5. Semester des Studiengangs „Informatik.Softwaresysteme“. Der begleitende Professor ist Prof. Dr. Martin Schulten.

1.1 Motivation

Die Idee dieses Projekts ist entstanden als Analogie zu den „Keyless“ Zugangssystemen vieler aktueller Automodelle, die einem ermöglichen ohne aktive Handlung sein Auto zu entriegeln und zu starten. Dieses Konzept wurde in Verbindung mit Heimautomation auf Haustüren übertragen. Es sollte ein System entstehen, dass einen schlüssellosen Zugang zum Haus/zur Wohnung ermöglicht. Das ursprüngliche Konzept sah vor, dieses System mit Hilfe der bereits vorhandenen Zugangssysteme der Autohersteller zu realisieren. Es war jedoch relativ schnell absehbar, dass aufgrund der Verschlüsselung der Autohersteller, diese Zugangssysteme dafür ungeeignet sind. Das Herausfinden oder Umgehen der Verschlüsselung stellt einen erheblichen Aufwand dar und ist vermutlich sogar strafbar, sodass wir uns für eine andere Möglichkeit entschieden haben.

1.2 Konzeption

Innerhalb der Projektgruppe haben wir nach Alternativen für die schlüssellosen Systeme der Autohersteller ausschau gehalten, so kam die Idee auf, mobile Geräte als Schlüssel nutzbar zu machen. Da die meisten technikaffinen Menschen heutzutage immer ihr Smartphone dabei haben, schien dieses Medium ideal als Schlüsselsystem. Das Smartphone soll sich ähnlich wie die Systeme der Autohersteller verhalten: mit ihm in der Hosentasche nähert man sich seinem Haus und sobald es sich gegenüber der Haustür authentifiziert hat, wird die Haustür automatisch entriegelt. Die Kommunikation zwischen Haustür und Smartphone findet per Bluetooth statt. Die Gegenstelle für das Bluetoothsignal ist ein kleiner ARM-Rechner, in unserem Fall ein Raspberry Pi B+. Dieser ist gekoppelt mit dem Schließmechanismus der Tür und einem Bluetooth-Modul. Die Verwaltung der Tür zum hinzufügen und entfernen berechtigter Smartphones ist über ein Webinterface möglich, welches ebenfalls vom

Raspberry Pi ausgeliefert wird.

1.2 Übersicht der technischen Umsetzung

Unser System arbeitet mit Hilfe eines Smartphones, dass die mobile Plattform Android und den Bluetooth-Standard 4.0 LE (Low Energy) unterstützt, sowie eines Bluetooth 4.0 LE USB-Sticks verbunden mit dem Raspberry Pi. Da unser System auch verkauft werden soll, mussten wir uns für einen möglichst kleinen Computer entscheiden, der unsere Anforderungen erfüllt, sodass wir verschiedene Kleinrechner zur Auswahl hatten: Raspberry Pi, Banana Pi. Im Laufe des Projekts kam außerdem der Intel Edison mit in die Auswahl, der eine sehr attraktive Alternative bot, da er bereits über integriertes Bluetooth 4.0, sowie WLAN verfügt. Ein solches Setup ließe sich leicht in bestehenden Häusern/Wohnungen nachrüsten, da keine Kabel o.Ä. (außer der Stromversorgung) gelegt werden müssten. Die Entwicklung des Projektes geschah komplett mit dem Raspberry Pi Modell B+. Anschließend sollte das System ebenfalls auf den Banana Pi, sowie den Intel Edison portiert werden, was aber aus Zeitgründen nicht mehr umgesetzt wurde. Auf dem Kleinrechner läuft eine Linux-Distribution mit Node.js, sowie einer kleinen MongoDB-Datenbank. Mehr zum Thema in Kapitel 3 “Server”.

Für die Entwicklung der Android-App standen uns verschiedene Smartphone's zur Verfügung: Samsung Galaxy S3, OnePlus One, Motorola Razr HD & Google Nexus 7. Das Android-Betriebssystem der Smartphones muss jedoch mindestens in Version 4.3 installiert sein, da erst ab Android Version 4.3 Bluetooth 4.0 unterstützt wird. Als Mini-Datenbank kam auf den Smartphones SQLite zum Einsatz. Mehr zum Thema Android in Kapitel 2 “Mobile Anwendung”.

1.3 Projektumfeld

Nachfolgend werden die eingesetzten Technologien & Werkzeuge erläutert.

Git: Als zentrale Softwareverwaltung haben wir Git eingesetzt. Das zentrale Repository ist auf der Website Github.com gehostet unter: https://github.com/Almewty/keylessentry/. Dort kann man auch eine Kopie der aktuell existierenden Lösung herunterladen und selbst zum Einsatz bringen.

Adobe Brackets: Brackets ist ein Editor für das Entwickeln von Webanwendungen. Wir nutzen ihn für die Erstellung der Serveranwendung in Node.js. Node.js nutzt die Programmiersprache JavaScript, für die Brackets Syntax-Highlighting beherrscht. Außerdem ist es möglich mit Brackets direkt mit Github zu kommunizieren & Quellcode runter- und Quellcode-Änderungen hochzuladen.

Android Studio: Android Studio ist die integrierte Entwicklungsumgebung von Google und offizielle Entwicklungsumgebung von Android. Wir haben Android Studio für die Erstellung der App genutzt. Android nutzt als Programmiersprache Java, beherrscht für dieses Syntax-Highlighting und ermöglicht es ebenfalls direkt mit Github zu kommunizieren.

Es wurde kein besonderes **Vorgehensmodell** bei der Softwareprogrammierung genutzt. Die Gruppe wurde allerdings in 2 Arbeitsgruppen unterteilt:

- **Android-Arbeitsgruppe:** Bestehend aus: Thomas Bohn, Sven Ehmer, Timo Matuszewski
- **Server-Arbeitsgruppe:** Bestehend aus: Fabian Künne, Jan Rathmer

Die Arbeitsgruppen hatten zum Ziel die jeweiligen Anforderungen an das Gesamtergebnis in dem ihnen zugeteilten Umfeld zu bewerkstelligen.

Die **Android-Arbeitsgruppe** hat sich um die Entwicklung der App und des Webinterface gekümmert. Dazu gehörte:

- Entwicklung der Grundfunktionalität (Kommunikation per Bluetooth)
- Entwicklung eines Hintergrundservices, der kontinuierlich nach vorhandenen Bluetooth-Türen sucht
- Entwicklung einer Oberfläche zum einfachen hinzufügen und entfernen von Türen innerhalb der App
- Entwicklung eines Webinterface zum hinzufügen und entfernen von Benutzern auf dem Server

Die **Server-Arbeitsgruppe** hat sich um die Erstellung der Serveranwendung und den Bau eines “Vorführkoffers” gekümmert. Dazu gehörte:

- Entwicklung der Grundfunktionalität (Kommunikation per Bluetooth)
- Konsistente Speicherung der authentifizierten Benutzer
- Hinzufügen und löschen eines Nutzers
- Authentifizierung eines anfragenden Clients
- ver-, entriegeln der Türfalle, bzw. Ansteuerung des Schließmechanismus
- Bau eines Vorführkoffers

Der Entwicklungszeitraum der Projektarbeit war vom 07. Oktober 2014 – 03. Februar 2015.

2 Mobile Anwendung

Die Android-Anwendung enthält mehreren Funktionalitäten. Die folgende Auflistung zeigt die Kernmechanismen der Anwendung. Diese werden in den anschließenden Kapiteln von ihrer technischen Seite erläutert.

- Kommunikation mit dem Server
- Registrierung von Türen

2.1 Kommunikation mit dem Server

Um mit dem Server kommunizieren zu können, benutzen wir den in Android mitgelieferten Bluetooth Stack. Ab Android Version 4.3 werden API's zur Verfügung gestellt, um mit Bluetooth 4.0 Low Energy arbeiten zu können, welches wir für unser Vorhaben nutzen. Wie in Kapitel 3.3 beschrieben wird, war es zur Zeit der Entwicklung noch nicht möglich, dass Smartphone als Clientgerät für die Bluetooth-Kommunikation zu implementieren. Daher stellt das Smartphone aktuell das zentrale Gerät einer Bluetooth-Verbindung dar.

Die Kommunikation mit Geräten innerhalb von Bluetooth 4.0 geschieht mit Hilfe so genannter Services. Hier stellt Bluetooth vorgefertigte Services bereit, die unterschiedlichen Zwecken dienen. Da keiner dieser Services unseren Vorgaben bezüglich der zu übermittelnden Daten entsprach, haben wir einen eigenen Service entwickelt. Dieser Service besteht aus einem, nur zum Schreiben freigegebenen, Characteristic. Die Daten, die vom Android-Gerät in dieses Characteristic geschrieben werden, bestehen aus einer 16 Byte langen UUID des Android-Geräts und eines 4 Byte langen One-Time-Password das vom Smartphone generiert wurde (näheres zum Thema *One-Time-Password* in Kapitel 3.7 “Authentifizierung anfragender Nutzer”).

2.2 Registrierung von Türen

Damit das Smartphone eine Tür öffnen kann muss diese erst innerhalb der Android-App registriert werden, sodass die jeweilige UUID und das secret der Tür vorliegt (näheres zum Thema *secret* in Kapitel 3.7 “Authentifizierung anfragender Nutzer”). Dies geschieht über einen vom Server generierten QR-Code der einen, auf die Android-App zeigenden, Link enthält.

Der Benutzer scannt diesen QR-Code mithilfe einer beliebigen QR-Code Scanner App und wird daraufhin gefragt, ob er diesen Link mit unserer App öffnen möchte. Wenn er dies bestätigt, dekodiert die App die Daten die in dem QR-Code enthalten sind und speichert anhand dieser die dazugehörige Tür in die interne SQLite Datenbank.

3 Server

Im folgenden Kapitel wird auf die Funktionalitäten des Servers eingegangen. Dabei wird die Funktionsweise und kurz die technische Umsetzung beschrieben.

Die Funktionalitäten des Servers umfassen:

- Kommunikation mit Bluetooth-Geräten
- Konsistente Datenhaltung
- Registrieren neuer Nutzer
- Löschen vorhandener Nutzer
- Authentifizieren anfragender Nutzer
- Öffnen/Schließen des Türmechanismus

3.1 Serverumgebung

Der Server besteht aus einer Linux-Umgebung mit installiertem BlueZ, Node.js und MongoDB. Die verwendete Linux-Distribution ist theoretisch irrelevant, sie muss jedoch die genannten Softwarekomponenten (BlueZ, Node.js, MongoDB) unterstützen.

Da wir uns aufgrund des guten Preis-Leistungs-Verhältnisses für den Raspberry Pi entschieden haben, müssen wir auf ein ARM-kompatibles Linux zurückgreifen. Auf dem Raspberry Pi verwenden wir das angepasste Debian-System „Raspbian“, welches speziell für die Raspberry Pi Hardware angepasst ist, um bestmögliche Performance und Stabilität zu gewährleisten.

Des weiteren wird das Paket **BlueZ** benötigt, dass unter Linux den offiziellen Bluetooth-Stack darstellt. Da BlueZ praktisch die einzige Alternative unter Linux ist, lässt sich über die Motivation BlueZ zu verwenden nicht viel sagen.

Außerdem verwenden wir **MongoDB**, eine Schema-freie, dokumentenorientierte, NoSQL-Datenbank, der Firma „MongoDB Inc.”. MongoDB gehört zu dem so genannten MEAN-Stack (MongoDB, ExpressJS, AngularJS, NodeJS), was bedeutet das es besonders gut mit den genannten Komponenten zusammenarbeitet. In Bezug auf die Datenbank ist vor allem die Zusammenarbeit mit Node.js gemeint. Des weiteren ermöglicht MongoDB uns die Daten auf “natürlichere Weise” zu modellieren, da eine Objektrelationale Abbildung für MongoDB mit Namen “Mongoose” existiert, die den Datenbankinhalt auf Objekte in Node.js abbildet.

Node.js verwenden wir, um die zentralen Aufgaben des Servers auszuführen. Node.js ist ein Framework zum Betrieb von Netzwerkanwendungen auf Basis der JavaScript-Engine V8, die ursprünglich von Google für den Browser Google Chrome

entwickelt wurde. Wir haben Node.js verwendet, da es die asynchrone Ausführung von Aufgaben nicht nur erlaubt, sondern als Programmierkonzept quasi darauf baut. Außerdem bringt Node.js direkt viele nützliche Funktionen mit, wie JSON-Export/Import oder Webschnittstellen (für das Verwaltungs-Webinterface).

Das von uns verwendete Webframework **AngularJS** setzt, genau wie Node.js auf JavaScript. JavaScript ermögicht eine schnelle Entwicklung (es muss nicht kompiliert werden, etc.) von Anwendungen, sowie gute Debugging-Möglichkeiten. JavaScript-Code ist einfach zu lesen und so auch für spätere Projektnachfolger einfach nachzuvollziehen. AngularJS ist ein Open-Source MVC Framework von Google. Es unterstützt asynchrone Aufrufe und manipuliert das DOM daraufhin in Echtzeit. Eines der Hauptziele von AngularJS ist sogenannte Single-Page-Apps zu erstellen, also Anwendungen die nur einmal vom Server geladen werden und anschließend ohne erneutes Laden der Seite Funktionalitäten bereitstellen. (näheres zum Thema *AngularJS* in Kapitel 5.1 “AngularJS”)

3.2 Node.js

Da wir hauptsächlich mit Node.js zu tun hatten, wird nachfolgend nochmals genauer darauf eingegangen. Wie bereits erwähnt, greift Node.js auf JavaScript als Programmiersprache zurück und da dieses eine ereignisgesteuerte Architektur vorgibt, ist Node.js ebenfalls so aufgebaut. Ereignisorientiert bedeutet, dass ein von Außen herbeigeführtes Event eine Aktion auslöst. Es gibt keine feste Reihenfolge in der verschiedene voneinander unabhängige Aktionen ausgeführt werden müssen, jedoch muss nahezu jede Aktion zuvor „ausgelöst“ werden. Der große Vorteil darin ist das „non-blocking“-Verhalten von Node.js, es gibt also fast keine Aktion, die eine andere blockiert und den Server zum Stillstand bringen könnte. Das “non-blocking”-Verhalten hat allerdings auch seine Nachteile. Datenbankanwendungen zum Beispiel müssen in bestimmten Situationen nämlich blockend sein, um die Datenkonsistenz zu erhalten. Es sollte zum Beispiel nicht möglich sein einen Tabelleneintrag zu löschen, während dieser gerade neu beschrieben wird. Diese beiden Aktionen müssen zwingend nacheinander ausgeführt werden, um Dateninkonsistenz zu vermeiden. Daher gibt es in Node.js dennoch Möglichkeiten „blockend“ zu programmieren, aber die grundsätzliche Architektur ist auf „non-blocking“ ausgelegt.

Als JavaScript-Engine nutzt Node.js die aus dem Browser “Google Chrome” bekannte Engine V8. Die JavaScript-Engine V8 ist komplett in C++ geschrieben, um JavaScript möglichst schnell auszuführen. Außerdem enthält V8 verschiedene Techniken, wie „Just-in-time-Kompilierung“, „Inline Caching“, „exakte automatische Speicherbereinigung“ und „Snapshots“, die die Ausführung von JavaScript-Code nochmals beschleunigen.

Die Laufzeitumgebung von V8, bzw. Node.js ist Single-Threaded, was bedeutet dass

jede Aufgabe auf dem Host-Rechner in einem einzigen Thread ausgeführt wird. Diese Eigenschaft ist auf unserem Raspberry Pi besonders von Vorteil, da dieser nur einen Prozessorkern besitzt und Threads daher nicht auf verschiedene Kerne auslagern kann.

3.3 Kommunikation mit Bluetooth-Geräten

Die Kommunikation von Node.js mit Bluetooth-Geräten erfolgt über das Node.js Modul bleno (https://github.com/sandeepmistry/bleno), welches unter Linux auf BlueZ zugreift. Bleno ist als „peripheral“-Module gedacht, was bedeutet, dass es sich wie ein Bluetooth-Gerät verhält mit dem sich verbunden werden kann. Bleno ist nicht in der Lage andere Bluetooth-Geräte zu kontaktieren, es kann lediglich auf eingehende Verbindungen reagieren. Diese Wahl mussten wir treffen, da sich Android nur als „central“-Modul verhalten kann, es kann also nur Bluetooth-Geräte kontaktieren, indem es versucht sich mit diesen zu verbinden. Android ist nur in der Lage andere Bluetooth-Geräte zu kontaktieren. Die einzige Möglichkeit mit einem Android-Smartphone zu kommunizieren ist also den Server als „peripheral“-Modul zu implementieren.

Damit bleno auf eingehende Bluetooth-Verbindungen reagieren kann, sendet es so genannte „advertisement“-Packages, die im Grunde nichts anderes tun als „ich bin da und ich heiße xy“ zu verschicken. Android kann diese Pakete empfangen und dem Server (dessen Namen es dann ja kennt) eine Nachricht schicken.

Wenn bleno auf eine eingehende Bluetooth-Nachricht reagiert, gibt es dessen Inhalt an die Funktion „checkSmartphone“ weiter, die dann die Authentifizierung der eingehenden Nachricht überprüft. Der Inhalt der Nachricht muss ein 20Byte großes Datenpaket sein.

3.4 Datenhaltung

Damit die Legitimität anfragender Android-Clients überprüft werden kann, müssen diese im Server hinterlegt sein. Der Kern dieser Aufgabe wird durch das Node.js Modul „MongoDB“ übernommen. Dieses stellt eine Schnittstelle zum Erstellen, Ändern und Löschen von Tabellen und Tabelleneinträgen bereit. MongoDB stellt bei allen Transaktionen mit der Datenbank sicher, dass die Datenkonsistenz erhalten bleibt. Es setzt dabei vor allem die AKID-Eigenschaften (Atomarität, Konsistenz, Isolation und Dauerhaftigkeit) um, sodass sichergestellt ist, dass jede Transaktion die Datenbank von einem konsistenten in einen anderen konsistenten Zustand überführt. So wird sichergestellt, dass die Datenhaltung jederzeit konsistent (also „richtig“) ist und keine fehlerhaften Datensätze gespeichert/erzeugt werden.

3.5 Registrierung neuer Nutzer

Um sich gegenüber dem Server zu Authentifizieren muss ein entsprechender Nutzereintrag des anfragenden Client in der Datenbank vorhanden sein. Diesen kann der Server durch das registrieren eines neuen Nutzers mit Hilfe der Methode „registerSmartphone“ erstellen. Dabei wird ein zufälliger UUID (Universally Unique Identifier) erstellt, in base64 konvertiert und in der Datenbank zusammen mit einem ebenfalls zufällig generierten und in base64 konvertierten „secret“ gespeichert.

UUID ist ein Standard für Identifikatoren, dessen Absicht es ist Informationen ohne zentrale Koordination eindeutig kennzeichnen zu können. Da ein UUID aus einer 16-Byte Zahl besteht und somit 2^{122} verschiedene UUID existieren, ist eine Eindeutigkeit gewährleistet. Die Eindeutigkeit ist **nicht garantiert**, da theoretisch zwei gleiche UUID existieren könnten, allerdings gehen wir von einer sehr geringen Anzahl von UUID innerhalb unserer Datenbank (1-30) aus, sodass eine Pseudo-Sicherheit gewährleistet ist.

Das „secret“ besteht aus 256 zufällig generierten Bytes, die in base64 konvertiert wurden. Es wird bei der Authentifizierung der Nutzer für die Erstellung eines OTP (One-Time-Password) benötigt. Näheres zum OTP wird in Kapitel 3.7 beschrieben.

Zusätzlich zum generieren und speichern von UUID und secret wird ein QR-Code erzeugt, der diese Informationen ebenfalls enthält. Im QR-Code ist ein Weblink mit den Attributen „UUID“, „secret“, „name der Tür“ und „UUID der Tür“ kodiert. Dieser QR-Code kann vom Smartphone mit einem beliebigen QR-Code Scanner dekodiert und, von unserer App, verarbeitet werden.

3.6 Löschen vorhandener Nutzer

Falls ein Nutzer aus der Liste der autorisierten Personen ausgetragen werden soll, kann der Server dies durch die Methode „deleteSmartphone“ bewerkstelligen. Dabei wird dem Server der UUID der zu löschenden Person übergeben. Dieser stellt dann eine Anfrage an die Datenbank und diese entfernt dann den Eintrag mit dem UUID und dem secret.

3.7 Authentifizierung anfragender Nutzer

Die Authentifizierung eines anfragenden Nutzers gegenüber dem Client geschieht in der Methode „checkSmartphone“. Dieser wird ein Bytecode übergeben, der in den ersten 16Byte den UUID und in den folgenden 4 Byte das OTP (One-Time-Password) enthält. Wenn der übergebene Bytecode keine 20 Byte groß ist, wird die Authentifizierung verweigert und die Nachricht verworfen.

Der gesamte Authentifizierungsprozess wird durch eine so genannte

„One-Time-Authorization“ realisiert. Dabei versucht sich eine Person einer anderen gegenüber als „legitim“ auszuweisen. Zuvor haben beide Personen ein so genanntes „secret“ ausgetauscht (bei uns mittels QR-Code), dass nur sie alleine kennen. Um sich nun zu authentifizieren erzeugen beide Seiten ein One-Time-Password (Abkürzung: OTP). Die Person, die sich ausweisen möchte schickt der anderen das OTP, sodass diese das empfangene OTP mit dem selbst erzeugten OTP vergleichen und damit die Legitimität der Person bestätigen oder verweigern kann.

Grundsätzlich gibt es 2 Möglichkeiten ein OTP zu erzeugen: Durch Kennwortlisten, oder durch Kennwortgeneratoren. Da wir nur von der zweiten Möglichkeit Gebrauch machen, wird im folgenden nur darauf eingegangen.

Kennwortgeneratoren sind in der Lage auf Basis des secret ein OTP zu erzeugen, dass entweder

- Zeitgesteuert
- Ereignisgesteuert
- oder Challenge-Response-gesteuert

generiert wird. Wir nutzen von diesen Verfahren die zeitgesteuerte und erzeugen das OTP auf Basis der aktuellen Zeit +- 30 Sekunden. Die +- 30 Sekunden sind ein Toleranzbereich innerhalb dessen die Authentifizierung als legitim erachtet wird. Dies ist notwendig, da nicht sichergestellt werden kann, dass Client und Server exakt die gleiche Zeit haben, es gibt immer etwas Abweichung. Es ist jedoch möglich diesen Toleranzbereich frei einzustellen.

Der Server erstellt auf Basis der aktuellen Zeit durch das Node.js Modul „speakeasy“ (https://github.com/markbao/speakeasy) und dem in der Datenbank zum UUID hinterlegten secret 3 OTP's (+- 30 Sekunden), vergleicht diese mit dem empfangenen OTP und öffnet bei Übereinstimmung eines OTP's den Türmechanismus.

3.8 Öffnen/Schließen des Türmechanismus

Das Öffnen oder Schließen des Türmechanismus geschieht auf dem Raspberry Pi innerhalb der Methode „checkSmartphone“ mit Hilfe des Node.js Moduls „pi-gpio“ (https://www.npmjs.com/package/pi-gpio). Um den Status eines GPIO-Pins des Raspberry Pi zu ändern ist es notwendig den fraglichen Pin zuerst in einen bestimmten Modus zu versetzen. Es ist möglich den Pin als Output, -oder als Input-Pin zu konfigurieren. Als Output Pin ist es möglich den Pinzustand zu definieren, als Input Pin ist es möglich den Zustand des Pin's abzufragen.

Unter einem GPIO-Pin (General Purpose Input/Output) versteht man einen Kontaktstift, dessen Verhalten durch logische Programmierung frei bestimmbar ist. Da GPIO-Pins

standardmäßig kein Zweck vorgegeben ist, sind sie frei verwendbar.

Wenn der Server die Tür schließen soll, öffnet er den Pin 7 im Output-Modus, schaltet auf diesen eine 0, welche einem Low-Signal entspricht. Wenn der Server die Tür öffnen soll, öffnet er den Pin 7 im Output-Modus und schaltet auf diesen eine 1, welche einem High-Signal entspricht. Gleichzeitig startet ein Timer, der nach einer Minute den Pin 7 erneut im Output-Modus öffnet und auf diesen eine 0 schaltet, damit die Tür nicht unendlich offen bleibt. Es könnte ja sein, dass man zufällig an seiner Haustür vorbeifährt, der Server eine legitime Authentifizierung registriert und die Tür öffnet. Die Tür schwingt dann zwar nicht von alleine auf, jedoch sollte die Türfalle nur geöffnet werden, falls man wirklich vor hat das Haus zu betreten.

4 Türfalle

Da der Kleinrechner in der Haustür nur die Fähigkeit hat einen bestimmten Pin High oder Low zu schalten, muss noch eine Möglichkeit geschaffen werden die Türfalle in der Tür zu aktivieren oder zu deaktivieren. Der GPIO-Pin des Raspberry Pi selbst stellt nicht genügend Strom zur Verfügung, um die Türfalle direkt zu steuern. Der Versuch dies zu tun, würde die integrierte GPIO-Bank auf dem Raspberry Pi zerstören, denn da sie nicht genügend Strom zur Verfügung stellen kann, würde sie einfach durchbrennen. Des weiteren ist die Spannung am GPIO-Pin (0-3,3V) nicht ausreichend für die Türfalle, die 12V benötigt. Daher ist es notwendig eine zwischengeschaltete Verstärkerschaltung mit dem GPIO-Pin zu verbinden, die dann die Türfalle schaltet.

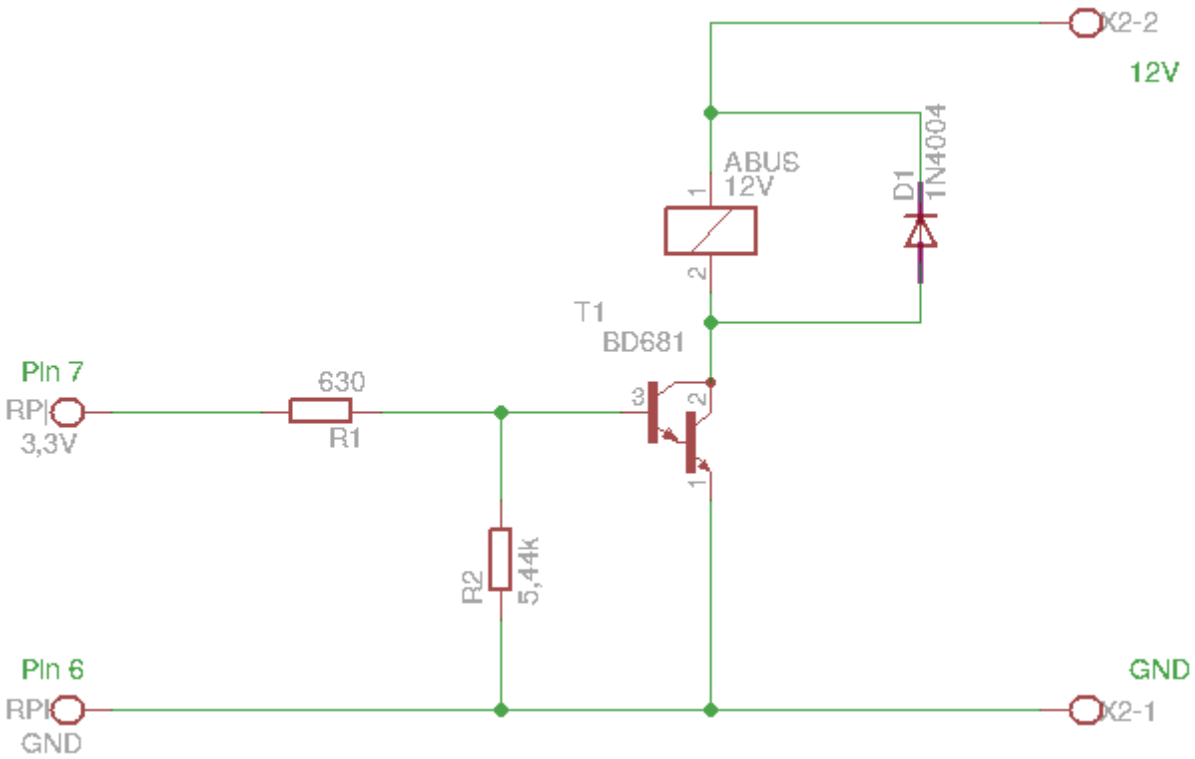
Die folgende Tabelle gibt die Spannungspotentiale am GPIO-Pin 7 des Raspberry Pi im High und Low-Zustand wieder:

Zustand	0	1
Logischer Zustand	Low	High
Spannungspotential	0V	3.3V

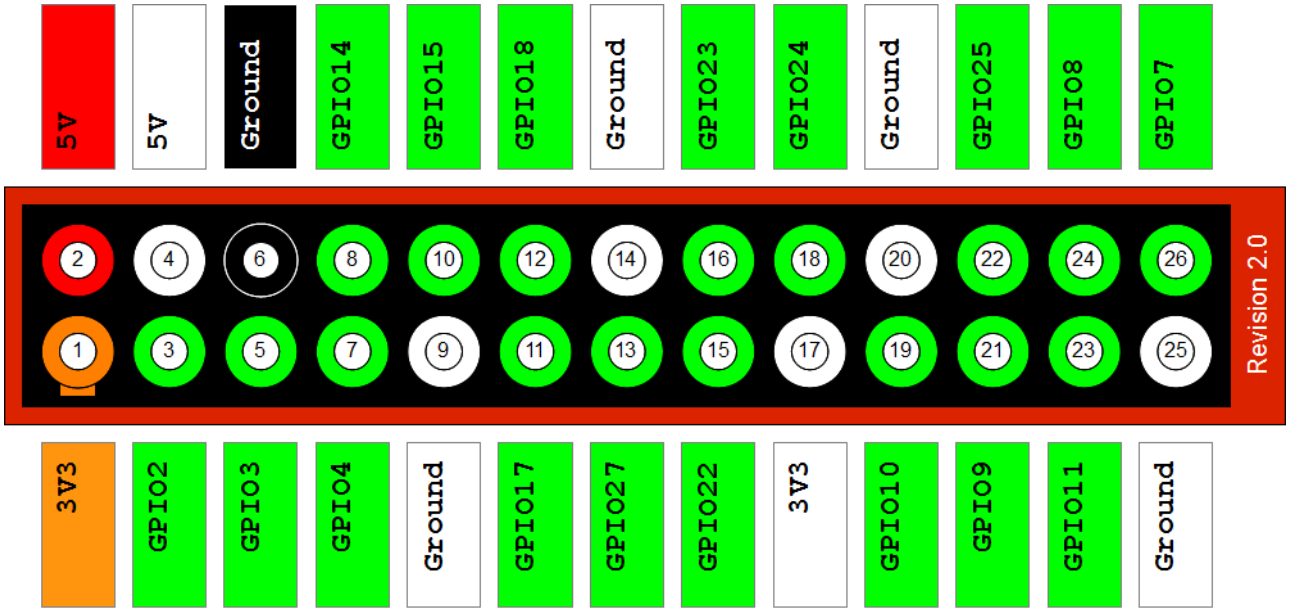
Die Verstärkerschaltung besteht aus einem Darlington-Transistor des Typs BD681. Dieser Transistor vereint in seinem Inneren eigentlich 2 Transistoren. Jeder Transistor hat einen eigenen Verstärkungsfaktor mit dem er den eingehenden Strom am Ausgang verstärkt. Durch die Verschaltung von 2 Transistoren mit Vor- und Hauptverstärkung kann eine deutlich höhere Stromverstärkung erreicht werden, als mit einem einzelnen Transistor. Dieser Transistortyp wird auch als Darlington-Transistor bezeichnet.

4.1 Schaltplan

Transistoren haben üblicherweise 3 Kontaktbeine, die mit Basis, Kollektor und Emitter bezeichnet werden. Der Pin 7 des Raspberry Pi wird mit der Basis des Darlington-Transistors verbunden, der Kollektor mit der Türfalle und der Emitter mit der Masse (GND). Das andere Ende der Türfalle wird mit einer 12V liefernden Stromversorgung verbunden, die mindestens 0,7A (besser 1A) Strom liefern muss. Zusätzlich werden 2 Widerstände mit den Transistor-Kontakten verbunden, um ein spezielles Spannungsverhältnis am Transistor zu erzeugen. Durch diese Widerstände wird gesteuert, wie der Transistor sich verhalten soll (wie viel Strom durch den Transistor und wie viel durch die Widerstände fließen soll). Wichtig ist dabei zu beachten, dass die Schaltung sowohl im Low-Zustand des Pin 7, als auch im High-Zustand nicht durchbrennen darf und der Transistor eindeutig die Türfalle an,- oder ausschaltet, der Transistor darf also nicht “flimmern”. Wenn der Pin 7 im Low-Zustand ist, „sperrt“ der Transistor. Der Strom zur Türfalle kommt zum erliegen, da der Transistor den Kontakt zur Türfalle nicht auf GND schalten kann. Die Türfalle wird deaktiviert. Ist der Pin 7 im High-Zustand „öffnet“ der Transistor. Der Strom zur Türfalle kann durch den Transistor zur Masse fließen und die Türfalle wird aktiviert. Es ergibt sich das folgende Schaltbild:



Die Pin-Belegung des Raspberry Pi B+ sieht wie folgt aus:



4.2 Widerstandswerte

Die Widerstände errechnen sich aus den Anforderungen an den Transistor. Die gegebenen Umstände sind:

Transistorstrom I_C = 1A

Verstärkungsfaktor des Transistors h_{FE} = 750

Basis-Emitter-Spannung U_{BE} = 1,45V

Gesucht wird der Widerstandswert von R1 und R2. Diese Werte lassen sich wie folgt errechnen:

Spannung über Widerstand 2 $U_{R2} = U_{BE} = 1,45V$

Spannung über Widerstand 1 $U_{R1} = 3,3V - U_{R2} = 3,3V - 1,45V = 1,85V$

Basisstrom Transistor $I_B = 2 * \frac{I_C}{h_{FE}} = \frac{1A}{750} = 2,6mA$

Der Faktor 2 gilt hier als “Sicherheitswert”, der dafür sorge trägt, dass der Transistor **definitiv** durchschaltet, oder sperrt. Dies verhindert ein “flimmern” des Transistors, wenn der Strom des Raspberry Pi absinkt.

Strom durch Widerstand 2 $I_{R2} = \frac{1}{10} * I_b = \frac{1}{10} * 2,6mA = 0,26mA$

Die $\frac{1}{10}$ sind dabei keine feste Größe, sondern eher ein Schätzwert. Der Strom durch R_2, soll etwa $\frac{1}{10}$ des Basisstroms betragen.

Gesamtstrom des Raspberry Pi $I_G = I_B + I_{R2} = 2,6mA + 0,26mA = 2,93mA$

Widerstand $R_1 = \frac{Ur1}{Ig} = \frac{1,85V}{2,93mA} = 630,68\Omega$

Widerstand $R_2 = \frac{Ur2}{Iq} = \frac{1,45V}{0,26mA} = 5,44k\Omega$

5. Verwaltungsseite

Zur Verwaltung der Smartphones die auf eine Tür Zugriff haben sollen, gibt es ein Webinterface. Dieses ermöglicht es neue Geräte zu erstellen und diese beim Server zu registrieren, sowie bereits bestehende Geräte zu bearbeiten oder zu löschen.

Die Verwaltungsseite wird vom Webserver des Raspberry Pi ausgeliefert. Der Webserver selbst wird mit Hilfe des Node.js Modul ExpressJS (http://expressjs.com/) implementiert. Express erleichtert im wesentlichen die Entwicklung von Webanwendungen auf Serverseite. Speziell werden hier die Routen definiert, also Ziele die auf dem Server über “Unterseiten” (localhost/unterseite1/unterseite2) erreichbar sind. Zur Entwicklung des Webinterface haben wir AngularJS genutzt, das im folgenden Kapitel 5.1 näher erläutert wird. Die Datenhaltung der Verwaltungsseite wird ebenfalls über die MongoDB-Datenbank abgewickelt. Die Kommunikation zwischen der Datenbank und dem Webinterface findet über eine REST-API statt. Die dafür nötigen Routen, sind in ExpressJS definiert.

5.1 AngularJS

Zur Entwicklung der Verwaltungsseite haben wir auf das Framework AngularJS zurückgegriffen. AngularJS ist auf die Entwicklung von Single-page-Webanwendungen ausgelegt, die mittels HTML und JavaScript erstellt werden. Im wesentlichen bietet AngularJS eine Erweiterung des HTML-Funktionssatzes. Darüber hinaus ist es möglich statisches HTML auszuliefern und Daten dynamisch nachzuladen. Ziel des Frameworks ist es, Webanwendungen möglichst einfach zu erstellen und zu warten. Das wird unter anderem dadurch erreicht, dass die Implementierung dem MVC-Pattern folgt. Dies ermöglicht außerdem erleichtertes Testen von Einzelkomponenten. Außer der Tatsache, dass wir bereits andere Komponenten des MEAN-Stacks (Kapitel 3.1) nutzen, ist AngularJS sehr ressourcenschonend. Das ist wichtig, da die Leistungsfähigkeit des Raspberry Pi eingeschränkt ist. Durch AngularJS wird das Webinterface nur einmal bei dem Client ausgeliefert bzw. nur einmal vom Server geladen. Alle weiteren Änderungen, die der Nutzer über das Webinterface tätigt, lösen kein komplettes Neuladen der Seite aus, sondern es werden nur die geänderten Daten vom Server geladen. Diese Tatsache schont den Raspberry Pi und ermöglicht ein angenehmeres Nutzererlebnis, denn es muss nicht nach jeder Aktion die komplette

Seite geladen werden, was durchaus 2-3 Sekunden dauern kann.

5.2 Anlegen eines neuen Nutzers

Damit sich ein Nutzer mit seinem Smartphone bei einer Tür anmelden kann, benötigt er ein secret und ein UUID. Dies wird in Kaptiel 3.5 näher erläutert. Diese beiden Informationen erhält der Nutzer über das Webinterface. Dazu erstellt er auf der Verwaltungsseite einen neuen Nutzer. Anschließend öffnet sich ein neues Fenster innerhalb der Seite, dass einen QR-Code anzeigt. Der QR-Code beinhaltet das secret und den generierten UUID. Den QR-Code kann der Nutzer scannen und unsere Android-Anwendung verarbeitet die Informationen daraus und speichert im Anschluss die Tür in der SQLite-Datenbank des Smartphones. Der QR-Code hat den Vorteil, dass die Nutzer innerhalb der App keinen UUID und secret eintippen müssen.

6. Zusammenfassung

Die Grundziele des Projektes wurden umgesetzt. Es ist möglich Smartphones im Server zu hinterlegen und zu entfernen, sowie die Tür von registrierten Smartphones zu öffnen. Die App kann die Tür nicht ohne Nutzerinteraktion, automatisch im Hintergrund, öffnen. Dies wurde aufgrund der drastisch kürzeren Akkulaufzeit des Smartphones veworfen, bis eine geeignete Möglichkeit gefunden wurde den Akku zu schonen. Das Webinterface und die API haben bisher keine Authentifizierung und es ist nicht möglich zu sehen, wer wann die Tür geöffnet hat. Ein weiteres Sekundärziel ist die Möglichkeit Nutzern zeitlich beschränkt Zugang zu gewähren. Dieses Ziel wurde aufgrund von Zeitmangel nicht erreicht.

Das Projekt funktioniert momentan als Proof-of-concept aber es muss noch einiges getan werden, um es reif für den Massenmarkt zu machen.

7. Projektziele und Verwirklichung

Ziel	Erfüllt	Umsetzung
Öffnen der Tür	Ja	Die Tür und das Smartphone verbinden und authentifizieren sich über Bluetooth
Sicherheit der Authentifizierung	Ja	Die Sicherheit ist durch die Nutzung eines One-Time-Password gewährleistet
Einfache Bedienung	Ja	Die App öffnet per Druck auf einen Button die Tür. Das Webinterface ist schlicht gehalten und intuitiv zu bedienen

Nutzerinteraktionen minimieren	teil-weise	Der Benutzer muss mit dem Handy interagieren, da eine dauerhafte Bluetoothverbindung zu stromintensiv ist
Stromsparend	nein	Der Stromverbrauch des Bluetooth-Suchvorgangs ließ sich nicht weit genug reduzieren um dauerhaft aktiv zu sein

8. Ausblick

Viele der erreichten Ziele lassen sich deutlich verfeinern und optimieren. Es gibt außerdem noch Potential, weitere Erweiterungen zu implementieren. Einige dieser möglichen Erweiterungen sind:

- Die Tür erhält einen Sensor, der registriert, ob sich der Nutzer innerhalb oder außerhalb des Hauses befindet. Je nach Standort oder Absicht des Nutzers öffnet sich die Tür oder nicht
- Die Tür sendet eine Benachrichtigung an ein Smartphone, wenn jemand die Tür öffnet mit der Information wer die Tür gerade öffnet.
- Die Tür lässt sich nicht nur mit Bluetooth-Smartphones öffnen, sondern anderer Hardware, z.B. mit einem smarten Schlüsselanhänger, oder Finger-Ring.
- Die Tür erkennt durch einen RFID-Chip in der Kleidung den Benutzer und öffnet. Vorteil dieser Methode wäre das passive Verhalten des RFID-Chips, der durch die Tür selbst mit Strom versorgt wird. Ein “Akku-leer-Problem” würde so nicht auftreten.
- Die Tür verbindet sich mit einer bestehenden Hausautomation und schaltet bei bestimmten Benutzern Licht oder Heizung an.

Das größte Problem, der Energieverbrauch des Smartphones resultiert aus dem hohen Stromverbrauch von Bluetooth. Selbst langfristig kann man nicht davon ausgehen, dass sich dieses Problem lösen lässt ohne das Bluetooth in einer verbesserten noch energieeffizienteren Version erscheint. Durch Android 5 wird dieses Problem bereits etwas verbessert, da Bluetooth LE durch das sogenannte “Project Volta” effizienter genutzt wird.