Waking Wind

Thorsten Schaeff Daniel Grießhaber

25. April 2014

Inhaltsverzeichnis

Zusammenfassung

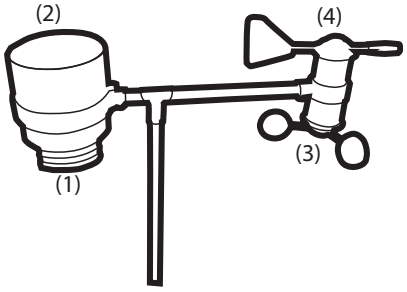
„Waking Wind“ ist ein Webservice der Windsportler mit Echtzeit Wetterdaten versorgt. Der Slogan „real-time wind data from anywhere to everywhere“ beschreibt unseren Fokus auf die möglichst einfache Erfassung von Daten selbst von abgelegenen Orten sowie das möglichst einfache Abrufen der Daten auch von unterwegs.

In dieser Dokumentation soll der Prozess von der Idee zum fertigen Produkt aufgezeigt werden.

1 Die Wetterstation

Bei der verwendeten Wetterstation handelt es sich um eine CTW-600 der Firma Chillitec GmbH[**CTW-600**]. Diese wurde wegen ihres geringen Preises und der Vielzahl an Messgeräten die sie zur Verfügung stellt ausgewählt. Sie besteht aus 2 Komponenten, dem Außensensor und der Basisstation. Der Außensensor bietet folgende Sensoren:

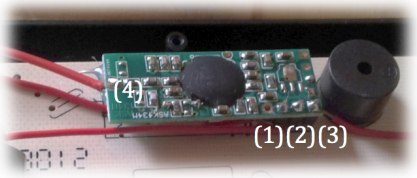
- (1) Temperatursensor
- (1) Luftfeuchtigkeitssensor
- (2) Regenfallsensor
- (3) Anemometer
- (4) Windrichtungssensor



Ein weiterer Vorteil dieser Wetterstation war die Kabellose Übertragung der Daten über das 433MHz Band. Dies ermöglichte das einfache Abgreifen der Daten.

2 Das Protokoll der Wetterstation

Der erste Schritt bestand in der Demodulation des 433MHz, amplitudenmodulierten Signals. Dieser Schritt gestaltete sich deutlich einfacher als zunächst angenommen, da die Basisstation ein, auf die Frequenz abgestimmtes, Empfangsmodul auf seine Hauptplatine aufgelötet hatte. Da die Basisstation nicht weiter benötigt wurde, konnte dieses Modul abgelötet und als Demodulator für das Signal verwendet werden. Das Modul bietet 3 Pins in Form einer Stiftleiste. Diese Pins haben folgende Funktion:



- (1) Vcc (Positive Stromversorgung)
- (2) GND (Masse)
- (3) SIG (Daten Pin)

Zusätzlich wurde noch hinzugefügt:

- (4) ANT (Verbindung zur Antenne)

Mithilfe eines Voltmeters war die Pinbelegung leicht herauszufinden.

2.1 Abgreifen der Daten

Zum Abgreifen der Daten wurden zunächst mehrere Möglichkeiten evaluiert. Die erste Variante bestand darin ein Arduino Leonardo[**ArduinoLeonardo**] mit einer speziellen Implementierung des offenen SUMP Logic Analyzer Protocols[**SUMP**] zu verwenden. Nach einigen Testversuchen zeigte sich, dass die Buffergröße des Leonardos zu klein war und die serielle Übertragungsrate des virtuellen USB Ports zu gering war um die Datenrahmen der Wetterstation aufzunehmen.

Eine naheliegende Alternative wären professionelle Logik Analysatoren, welche allerdings preislich ein vielfaches eines Arduino Leonardo kosten.

Durch die fehlerhaften Mitschnitte der Daten hat sich jedoch gezeigt, dass die einzelnen Signale in einer Abfolge von mindestens $400\mu s$ aufeinander folgen. Daraus ergab sich eine nötige Abtastrate bei der Analyse nach dem Abtasttheorem von Nyquist[**Nyquist1928**] und Shannon von:

$$\frac{1}{400\mu s} * 2 = 5kHz$$

Durch dieses Wissen war es möglich die Daten mithilfe einer Soundkarte, welche im Normalfall über eine Samplingrate von $44.1kHz$ verfügt, aufzuzeichnen.

Da das Empfangsmodul mit einer Spannung von $5V$ arbeitet, der Line Anschluss einer Soundkarte allerdings einen Bezugspegel von $0.5V$ benötigt, wurde ein Transistor zur Entkopplung eingefügt.

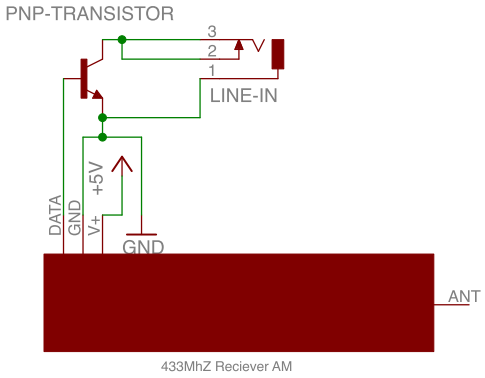


Abbildung 1: Entkopplung des Signals mithilfe eines Transistors

Die Datenrahmen wurden mit der freien Audio Software Audacity[**Audacity**] aufgezeichnet und ausgewertet.

2.2 Signal

Bei Auswertung des Signals fiel auf, dass das Signal zusätzlich mit einer Pulsweitenmodulation codiert war. Dabei repräsentiert ein hohes, kurzes Signal eine

logische 1 und ein hohes, langes Signal eine logische 0. Die Pause zwischen den Signalen ist immer gleich lang und dient lediglich zur Abgrenzung der logischen Pegel. Durch Vergleich mehrerer aufgezeichneter Datenrahmen konnte folgendes Wissen über die Pulslängen hergeleitet werden:

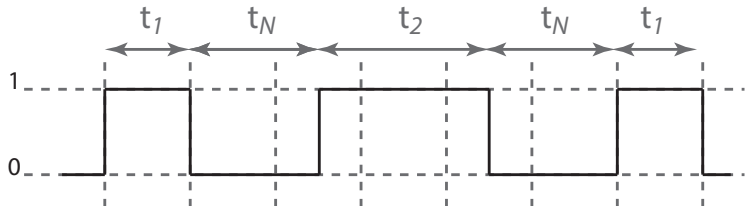


Abbildung 2: Illustration eines PWM Signals

Ausdruck	Dauer $t_i[\mu s]$
1	400 - 621
N (Pause)	805 - 1150
0	1495 - 1840

Tabelle 1: Pulslängen der logischen Werte

Dieses Wissen Abbildung war auch für das spätere Design des Empfängers notwenig, da pulsweitenmodulierte Signale normalerweise mit einem Tiefpassfilter gefiltert werden. Durch die diskrete Anzahl an möglichen Pulslängen ist dies allerdings nicht nötig, sondern das Signal kann auch in Echtzeit durch ein diskretes Programm demoduliert werden.

2.3 Protokoll

Die Datenrahmen besitzen eine Länge von 88 Bit. Diese Rahmen mussten den folgenden Informationen zugeordnet werden:

- Temperatur
- Luftfeuchtigkeit
- Regenfall
- Windgeschwindigkeit
- Windrichtung

Außerdem hat sich im Verlauf des Reverse Engineerings gezeigt dass zusätzlich noch folgende Informationen von der Wetterstation übertragen werden:

- Eindeutige ID des Außensensors

- Batterieladungszustand des Außensensors
- Prüfsumme aller Daten

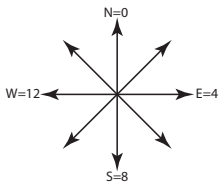
Die Zuordnung der Daten erfolgte iterativ in folgenden Schritten

1. Aufzeichnung und Demodulation eines Datenrahmens
2. Aufzeichnung der Daten an der Wetterstation
3. Einen gezielten Stimuli auf einen der Sensoren der Wettertstion geben
4. Aufzeichnen eines weiteren Datenrahmens und der Daten der Basisstation
5. Finden der unterschiedlichen Stellen zwischen den beiden Datenrahmen
6. Assoziationsanalyse mit der gefundenen Differenz der Datenpakete und den Daten der Basisstation

Letztlich ergab sich folgende Tabelle, mit deren Hilfe das Protokoll beschrieben werden kann. Dabei werden die Bits in ihrer chronologischen Reihenfolge in der sie von dem Außensensor gesendet werden durchnummeriert.

Bits	Bedeutung	Umrechnung	Default Wert
0 - 9	Synchronisation	-	1111111110_2
11 - 13	Batterieladungszustand	$Wert_{10} = [0 - 15]$	-
14 - 19	Eindeutige ID des Ausßensensors	-	-
20 - 21	Keine Bedeutung	-	00_2
22 - 31	Temperatur	$T[\,^\circ\mathrm{C}] = \frac{Wert_{10}-400}{10}$	-
32 - 39	Luftfeuchtugkeit	$\varphi[\%] = Wert_{10}$	
40 - 55	Windgeschwindigkeit	$v[\frac{km}{h}] = \frac{Wert_{10}}{240}$	-
56 - 71	Regenfall	$V[\frac{mm}{m^2}] = Wert_{10} \cdot 0.3$	-
72 - 75	Keine Bedeutung	-	00_2
76 - 79	Windrichtung	$Wert_{10}$Anordnung wie Windrose, 0=Norden	-
80 - 88	Prüfsumme	*unbekannt*	-

Die Windrichtung ist im Uhrezeigersinn durchnummeriert und in 8 Teile unterteilt. Für die 16tel Teile wurden Platzhalter freigelassen, jedoch werden diese nicht von der Wetterstation genutzt.



Um herauszufinden wie sich die Prüfsumme berechnet wurden folgende Algorithmen auf mehreren Datenrahmen getestet:

- CRC-8
- Summe über alle Bytes (Überlaufe ignorieren)
- Verschiedene Kombinationen von XOR Gattern auf Byte Basis

Obwohl es nicht gelang den tatsächlich verwendeten Algorithmus zu identifizieren, ist die Lösung der XOR Gatter - aufgrund des sehr geringen Berechnungsaufwandes und der einfachen Realisierung in Hardware - die Naheliegendste.

3 Empfänger

Die Aufgabe des Empfängers ist es, das Signal der Wetterstation zu empfangen, zu demodulieren, zu interpretieren und in der Datenbank persistent zu speichern. Sekundäre Ziele waren eine möglichst günstige Fertigung, sowie die Unabhängigkeit von der Umgebung und ggf. der Infrastruktur.

Für die Umsetzung der Ziele wurde der Empfänger in 2 Module aufgeteilt; das Empfangsmodul, dessen Aufgabe der Empfang und die Demodulation des Signals ist, und einer weiteren aktiven Komponente für die Interpretation und Übertragung der Daten.

Für den Prototyp nahm ein Raspberry PI[**RasPi**] die Rolle der zweiten Komponente ein, da dieser durch seinen geringen Stromverbrauch von ca. $2W$ und den geringen Anschaffungspreis von ca. €35 die Kosten für die Inbetriebnahme und den Betrieb einer Station nicht unnötig in die Höhe treibt.

Das Empfangsmodul musste aufgrund der speziellen Anforderungen selbst entwickelt werden. Dabei stand auch die Minimierung der Kosten im Vordergrund.

3.1 Empfangsmodul

Das Herz des Empfangsmoduls bildet ein ATTiny 45 Mikrocontroller[**ATtiny45**] der mit einem internen $8MHz$ Quarzoszillator arbeitet. Auf einen externen Quarz wurde, aus Kostengründen und weil der ATTiny nur 5 Input/Output Pins hat, verzichtet. Ein externer Quarz hätte 2 dieser Pins für die Taktversorgung bereits belegt. Für die Einspeisung des Signals wurde der $433MHz$ Empfänger der Basisstation verwendet und an `PB2` des Mikrocontrollers angeschlossen.

Aufgrund des fehlenden Hardware UART für die serielle Übertragung musste ein minimaler Software UART (siehe Seite 8) implementiert werden. Die Daten werden über diesen an einen USB auf UART Adapter gesendet, der auf seiner USB Seite einen virtuellen, seriellen Port zur Verfügung stellt, auf dem die Daten nach dem Empfang ausgelesen werden können.

Für die Funktionsanzeige wurde an den noch übrigen `PB1` und `PB3` eine zweifarbige LED angebracht. Diese leuchtet beim Empfang eines $433MHz$ Signals gelb und blinkt bei erfolgreichem Empfang eines Datenrahmens rot.

Verbesserung der Empfangsqualität

Für die Verbesserung der Empfangsqualität wurde die alte Drahtantenne des Basisstationsempfängers durch einen SMA-Anschluss ersetzt. Da diese der Standart für WLAN Antennen sind, haben sie den Vorteil, dass sie sehr preiswert in der Anschaffung sind. Zusätzlich wurde eine alte WLAN Antenne für den Empfang von $433MHz$ Signalen umgerüstet. Es gibt zwar auch spezielle $433MHz$ Antennen, jedoch sind diese wegen ihrer geringen Verbreitung sehr teuer.

Für die Umrüstung wurde die Antenne geöffnet und auf die passende Länge gekürzt. Die Wellenlänge eines $433MHz$ Signals berechnet sich wie folgt:

$$\lambda = \frac{c}{f} = \frac{3 \cdot 10^8 \frac{m}{s}}{433MHz} \approx 69.23cm$$

Aufgrund des Platzmangels im Gehäuse der WLAN Antenne wurde die Antenne noch auf $\frac{\lambda}{8} \approx 8.65cm$ gekürzt.

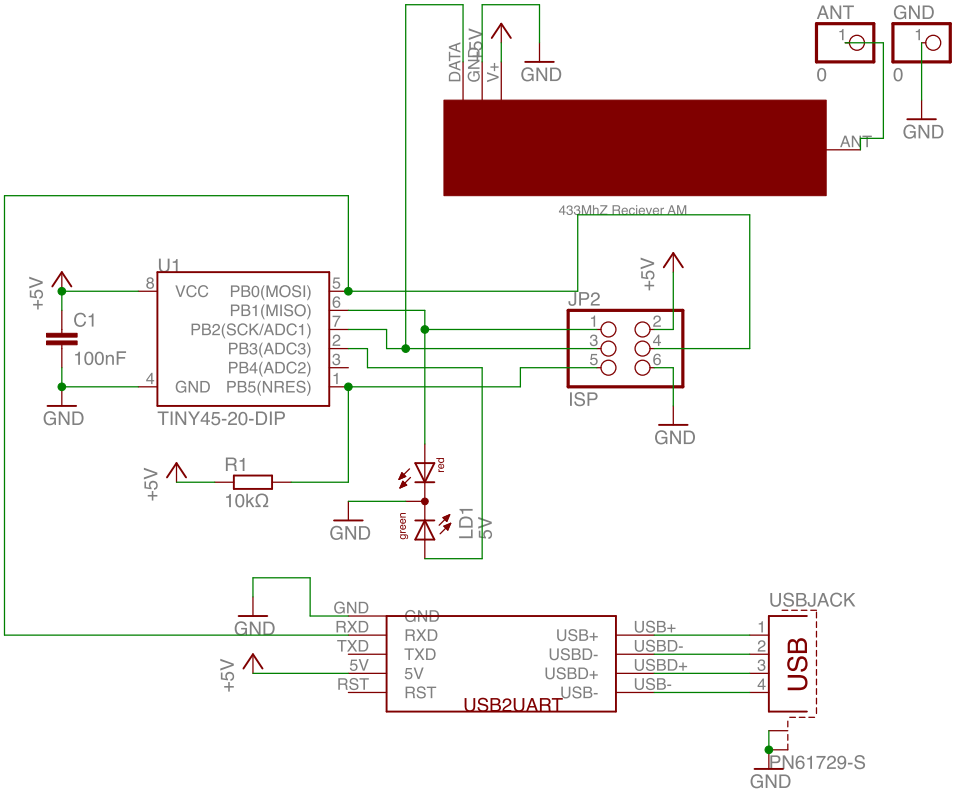


Abbildung 3: Schematik des Empfängers

Das Empfangen der Daten

Das Hauptprogramm, das auf dem Mikrocontroller läuft arbeitet nach dem Schema in [Abbildung 4].

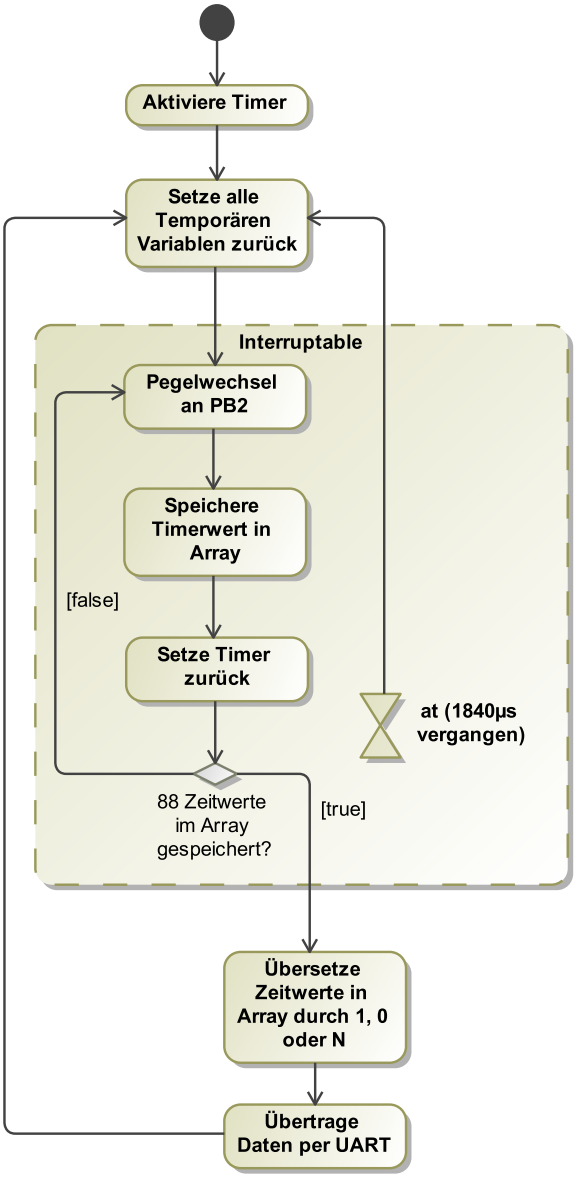


Abbildung 4: Ablaufdiagramm des Hauptprogramms

Dabei ist anzumerken, dass es durch das kritische Timing beim Empfang des Signals nicht möglich war, die ankommenden Bits direkt zu übersetzen. Deshalb werden zuerst die Timerwerte zwischengespeichert und nach erfolgreichem Empfang eines Rahmens diese Werte mit Hilfe der [Tabelle 1] übersetzt.

Wird beim Empfang festgestellt, dass zwischen zwei Pegelwechseln die Zeit größer der maximalen Länge für eine logische 0 überschritten wird, wird der Empfang abgebrochen, da es sich nicht um das Signal der Wetterstation handelt.

Der Software UART

Für die Übertragung der Daten vom Mikrocontroller zum USB auf UART Adapter war eine eigene Implementierung nötig, da der Mikrocontroller keine Hardware Implementierung des **U**niversal **A**synchronus **R**eciever and **T**ransmitter bereitstellt.

Mikrocontroller, die diese Schnittstelle bereitstellen, sind mit deutlich mehr Funktionen und Input/Output Ports ausgestattet. Das erhöht sowohl den Preis, als auch die Baugröße dieser Chips.

Die eigene Implementierung beschränkt sich auf das Senden der Daten, da nur die Simplex Kommunikation in Richtung der Schnittstelle vorgesehen ist. Dabei wird die Datenleitung, solange keine Übertragung stattfindet, durch den internen PullUp Widerstand des ATTiny auf einem Pegel von $+5V$ gehalten. Die Übertragung erfolgt Byteweise. Dabei wird am Anfang jedes Bytes die Leitung auf logisch 0 ($0V$) gezogen. Danach erfolgt die Übertragung der Bits durch entprechendes senken auf $0V$ oder halten auf $+5V$. Am Ende des Bytes wird die Leitung für das Stoppbit wieder auf $+5V$ gehalten.

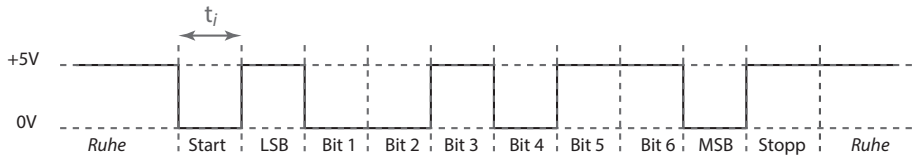


Abbildung 5: Beispielverlauf der Übertragung über UART

Da bei dieser Art der Übertragung das Timing wichtig ist, muss im Vorfeld geklärt werden, wielange die Dauer t_i eines Bits (Symbol) ist. Durch Versuche zeigte sich, dass die höchste Symbolrate (Baudrate) bei $38400Baud$ liegt, ohne dass es zu Übertragungsfehlern durch Überlastung des Mikrocontrollers kommt. Dadurch ließ sich die Anzahl der Takte zwischen den Flankenwechseln (t_i) mit folgender Formel berechnen:

$$\frac{IOClock}{Prescaler \cdot 1MHz} \cdot \frac{1MHz}{BAUD} = \frac{8MHz}{1 \cdot 1MHz} \cdot \frac{1MHz}{38400Baud} \approx 208$$

3.2 Herstellung des Prototypen

Während der Evaluations- und Implementierungsphase wurde die Schaltung lediglich auf einem Breadboard (Steckbrett) betrieben. Für den produktiven Einsatz musste aber eine dauerhafte und stabilere Lösung gefunden werden.

Aus diesem Grund wurde zusätzlich zur Schematik des Empfängers ein **P**rinted **C**ircuid **B**oard Layout erstellt. Dabei wurde für günstige Konstruktionspreise darauf geachtet, dass das Layout der Leiterplatte nur auf einer Seite bearbeitet werden muss.

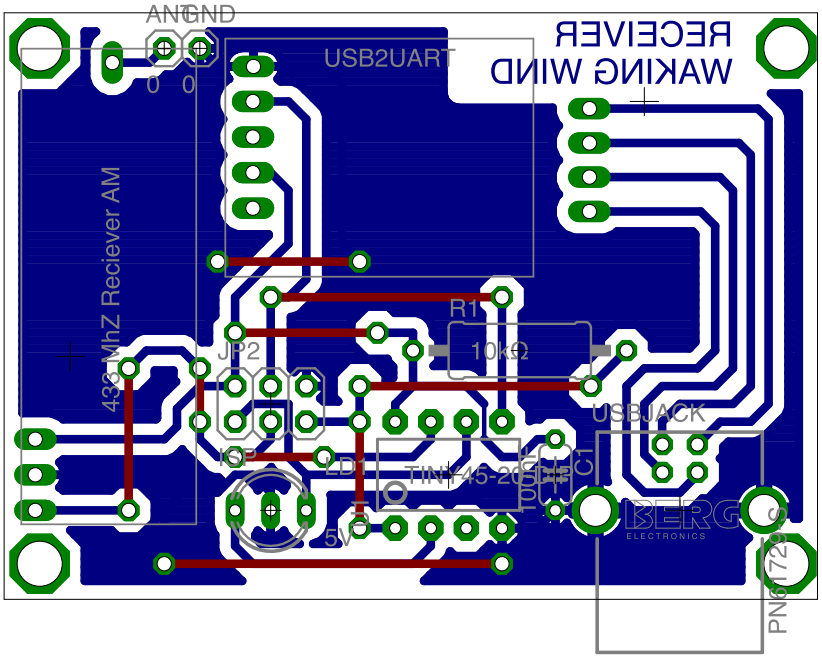


Abbildung 6: Das PCB Layout

Bei weiteren Versionen des Layouts könnte noch die Komponente `JP2` in der Mitte entfernt werden, da diese nur zur Programmierung des Chips gebraucht wird. Ausserdem könnte der Anschluss für die Antenne oben links (`ANT` und `GND`) weiter in die Mitte verschoben werden, so dass dieser besser gegen elekromagnetische Störerscheinungen durch die Masseplatte abgeschirmt wird.

Als nächstes wurde das zweidimensionale Layout in einer 3D Software modelliert um eine bessere Vorstellung der Maße zu bekommen und um ein passendes Gehäuse auswählen zu können (Abbildung 7).

Zur Herstellung der Prototyp Platine wurde das Tonertransferverfahren auf ein zugeschnittenes Stück einseitig mit Kupfer beschichtete Epoxydplatte angewendet. Dabei geht man in folgenden Schritten vor:

1. Reinigen der Kupferfläche von Fetten mithilfe von Ethanol
2. Druck einer Seite des Layouts (Blau) mit einem Laserdrucker in Schwarz auf ein Blatt Hochglanzpapier (Dabei wird der Toner nicht ganz auf das Papier gebrannt)
3. Positionierung der gedruckten Schablone auf der Kupferfläche
4. Erhitzen der Schablone für ca. 5 Minuten mit einem Bügeleisen
5. Einlegen der Kupferfläche in kalten Wasser
6. Entfernen der Papierreste
7. Ätzen der Kupferfläche (dabei werden die Stellen die vom Toner geschützt sind nicht angegriffen)

Zum Ätzen wurde eine 20% ige Natriumpersulfatlösung ($Na_2S_2O_8$) verwendet. Diese ist nicht so aggressiv wie Ätzmittel auf Salzsäurebasis (HCL) und verträglicher mit der Umwelt.

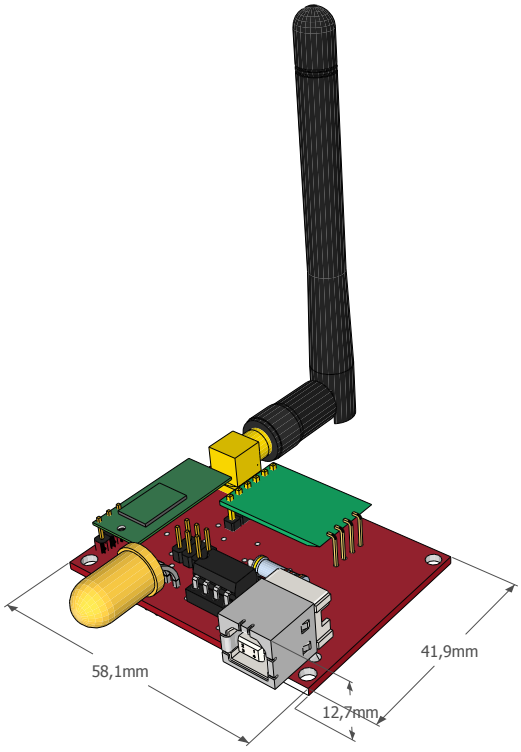


Abbildung 7: 3D Illustration mit Bemaßung

3.3 Interpretation und Übertragung der Daten

Für das Interpretieren und Übertragen der Daten wurde eine Plattformunabhängige Applikation in dem C++ Framework Qt[**QtProject**] geschrieben. Dies ermöglicht den Einsatz mit folgenden Betriebssystemen

- Windows
- Intel Based Mac OSX
- Linux (x68/x64/ARM)

Dadurch kann das Empfangsmodul über USB an alle, bereits vor Ort befindlichen Computersysteme angeschlossen werden, wodurch der Raspberry PI der im Prototyp eingesetzt wurde obsolet wird.

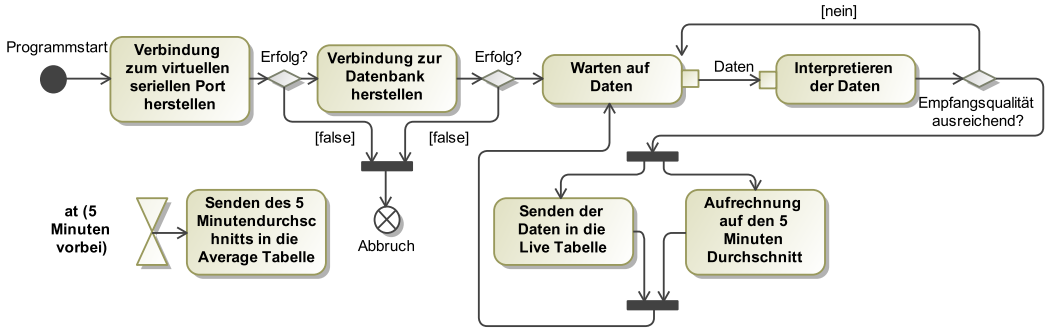


Abbildung 8: Ablauf der Interpretation und Übertragung

Dabei wird die Empfangsqualität aufgrund einfacher Algorithmen berechnet. So wird zum Beispiel geprüft ob die Synchronisationssequenz vollständig empfangen wurde. Dies stellte die einzige Möglichkeit der Validierung dar, bis die Prüfsumme der Wetterstation berechnet werden kann.

Die Übertragung der Daten an den Datenbankserver erfolgt dabei über eine native, unverschlüsselte MySQL Socket Verbindung. Auf eine Verschlüsselung wurde verzichtet, da die Daten keinerlei Nutzen für mithörende Angreifer hätten.

4 Speichern der Daten

4.1 Die Infrastruktur

Auf serverseite haben wir uns für eine LAMP Architektur entschieden:

- **L**inux als Betriebssystem
- **A**pache als Webserver
- **M**ySQL als Datenbankmanagementsystem

- **P**HP als serverseitige Skriptsprache

Installiert auf einem virtuellen Server im Rechenzentrum der Hochschule, bildete dies die Grundlage für den Webservice.

4.2 Die Datenbankstruktur

Tabellenname	Funktion
Locations	Enthält alle Wetterstationen mit genauem Standort sowie Meta-Daten für die Suchfunktion.
realtime_data	Enthält immer den neusten Satz an Wetterdaten zu jeder Location. Hier wird kein Verlauf gespeichert sondern die Wetterdaten pro Location aktualisiert.
average_data_loc[x]	Enthält den Verlauf der Wetterdaten für die Statistik der Location mit der ID x
Users	Enthält alle registrierten Nutzer inklusive E-Mailadresse und md5-Hash des Passwortes.
gcm_users	Enthält alle Registration-IDs (Erklärung s. Phonegap App) der Nutzer, die sich auf ihrem Smartphone mit unserer App eingeloggt haben.
fav_spots	Enthält für jeden Nutzer die favorisierten Locations sowie die Alarmeinstellungen zu diesen Locations.

5 Auslesen der Daten

Für die Präsentation der Daten haben wir uns, um möglichst viele Endgeräte unterstützen zu können, für zwei Varianten entschieden:

5.1 Webseite mit responsive Design

Funktionalität im Überblick:

- Responsive Design
- Abruf / Aktualisierung der Daten durch Server-Sent-Events
- Ajax Fallback für ältere Browser
- Location Online
- Location Suche
- User-Registrierung

- Login / Sessions
- Locations favorisieren
- Alarm für Locations stellen
- Statistik-Charts der einzelnen Locations
- Lokalisierung der Locations mit Hilfe der Google Maps API

Funktionalität im Detail:

Responsive Design

Um die Website möglichst für alle Endgeräte optimal darstellen zu können wurden verschiedene CSS-Dateien angefertigt, welche abhängig von der Auflösung bzw. der Abmessung der Anzeigefläche über sogenannte Media Queries geladen werden.

```
1 <link rel="stylesheet" media='screen' href="css/main.css" type="
    text/css" />
2   <link rel="stylesheet" media='screen and (max-width:950px) and (
    min-width: 750px)' href="css/med.css" type="text/css" />
3   <link rel="stylesheet" media='screen and (max-width:750px)' href=
    "css/small.css" type="text/css" />
4   <link rel="stylesheet" media='screen and (max-device-width:480px)
    ' href="css/mobile.css" type="text/css" />
```

Codebeispiel 1: Designanpassung je nach Endgerät

Server-Sent-Events (SSE) – „Die Schlanke Variante“

SSE ist eine HTML5-Technologie, die es dem Client ermöglicht automatisch Aktualisierungen von einem Server zu empfangen, ohne Fragen zu müssen, ob neue Daten vorhanden sind. Im Gegensatz zu Websockets stellt SSE keine Full-Duplex-Verbindung bereit, sondern lediglich einen Kanal über den der Server Updates zum Client pushen kann. Für unseren Anwendungsfall war dies die optimale Technologie, da sich die Wetterdaten sehr häufig ändern und wir diese in Echtzeit dem Nutzer zur Verfügung stellen wollen.

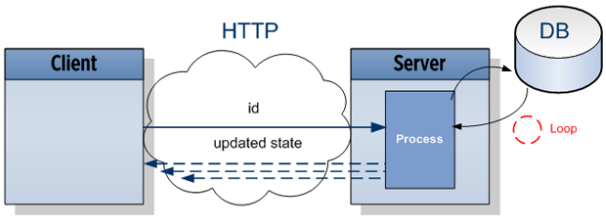


Abbildung 9: SSE Architektur [**sse-architecture**]

Ein weiterer Vorteil der SSE im Gegensatz zu Technologien wie Ajax, welche früher eingesetzt wurden um Echtzeitanwendungen zu realisieren, ist die Einsparung von viel HTTP-Overhead. Bei der Messung mit Wireshark zeigte sich

anhand der Übertragung von neun JSON-Objekten ein über 3 mal so großes Datebvolumen (Tabelle 2).

Übertragungsmethode	Bytes
Ajax	9.855
SSE	2.645

Tabelle 2: Test des HTTP Overhead in Wireshark

Dies liegt daran, dass SSE lediglich eine HTTP-Verbindung aufbaut und den Datenkanal offen hält, über welchen dann alle weiteren Datenpakete vom Server gesendet werden. Da unser Webservice mit Fokus auf mobile Endgeräte entwickelt wurde stellt dies, vor allem in mobilen Netzen einen Vorteil dar. Die Funktionalität der Server-Sent-Events haben wir mithilfe von JavaScript (auf Clientseite) und PHP (auf Serverseite) implementiert.

Clientseite:

```
evtSource = new EventSource("getwinddata.php?locid=" + locid);
evtSource.addEventListener("winddata", function(e) {
    //Aktualisieren der Werte
}, false);
```

Codebeispiel 2: Der Client horcht auf „winddata“-Events

Serverseite:

```
while(1){
  //read winddata from table
    $result = mysql_query($sql);
    $data = @mysql_fetch_assoc($result);
    $wspeed = $data['windspeed'];
    $wdir = $data['winddir'];
  //generate timestamp
    $curTime = date('H:i:s');
    if($prev != $wspeed){
      $prev = $wspeed;
      echo "event: winddata\n";
  //build json object
      echo 'data: {"wspeed": "'.$wspeed.'", "wdir": "'.$wdir.'"}';
      echo "\n\n";

      ob_flush();
      flush();
    }
    sleep($update_frequenz);
  }
```

Codebeispiel 3: Server prüft auf neue Werte und pusht diese an den Client

Ajax-Fallback

Da SSE nur von neueren HTML5 Browsern, welche zwar auf den mobilen Endgeräten zu einem Großteil schon vorhanden sind, unterstützt werden, musste

zusätzlich ein Ajax-Fallback für alle älteren Browser implementiert werden. Das Ajax-Fallback klopft alle 30 Sekunden bei unserem Server an und fordert ein neues Datenpaket an. Hierzu wird jedes Mal eine neue HTTP-Verbindung aufgebaut.

```
if(typeof(EventSource)!=="undefined"){
    // Yes! Server-sent events support.
    sse();
}else{
    //Sorry! No Server-Sent Events Support...
    sse_fallback();
}
```

Codebeispiel 4: Werden SSE unterstützt?

Location Online

Eine PHP-Funktion, welche überprüft, ob für die jeweilige Location innerhalb der letzten fünf Minuten ein neuer Datensatz eingetragen wurde.

```
function loc_online ($locid){
  //built sql query
  $sql = "SELECT timestamp FROM realtime_data WHERE timestamp > (
    DATE_SUB(NOW(), INTERVAL 5 MINUTE)) AND loc_id='".$locid."'";
  $result = mysql_query($sql) OR die( mysql_error() );
  if( mysql_num_rows( $result ) )
  {
    $row = mysql_fetch_assoc( $result );
    return $row['timestamp'];
  }
  else
  {
    return '';
  }
}
```

Codebeispiel 5: Ist die Location online?

Diese Funktion dient dazu dem Nutzer anzuzeigen, welche Wetterstationen zum aktuellen Zeitpunkt Echtzeitdaten liefern.

Location Suche

Für die Suche der Locations wurde das Autocomplete Widget des jQuery UI Frameworks verwendet. Durch ein eigenes Renderingscript passt die Ausgabe optisch zum Rest der Seite. Auf eine Serverseitige Suche wurde aufgrund der hohen Komplexität und der geringen Anzahl an Locations verzichtet.

User Registrierung

Bei der Registrierung wird lediglich zuerst geprüft, ob die EMail Adresse bereits registriert ist. Ist dies nicht der Fall, wird ein neuer Datensatz in der *Users* Tabelle angelegt. Das Passwort wird aus Sicherheitsgründen nur als gehashter Wert gespeichert.

Login/Sessions

Loggt sich ein User erfolgreich ein, wird in seiner Session dieser Zustand gespeichert. Zusätzlich zu dieser Information werden in der Session häufig benötigte Daten des Users gespeichert (z.B. seine ID und sein Username). Diese Informationen ermöglichen eine elegante Programmierung zahlreicher Ajax features wie beispielsweise das favorisieren von Locations.

Sessions sind im Normalfall 24 Stunden gültig, bevor sie automatisch gelöscht werden. Loggt sich der User manuell aus wird die Session sofort zerstört.

Location favorisieren

Nach erfolgreicher Anmeldung kann der User beliebige Locations favorisieren. Dabei wird nach klick auf den grauen Stern der Location ein Ajax-Request mit der ID der Location an den Server gesendet. Dieser holt sich die User ID über die Sessionvariable und speichert diese Kombination in der *fav_spots* Tabelle

Alarm für Locations stellen

Das stellen eines Alarms ist nur für favorisierte Spots möglich und erfolgt über die Seitenleiste nach dem erfolgreichen Login.

Auch hier wird wieder ein Ajax-Request in Kombination mit den Session Parametern verwendet.

Statistik-Charts

Die Statistiken basieren auf der Highcharts API [**highcharts**] und werden ausschließlich mit MySQL befehlen generiert. Durch die Trennung der Daten in persistente Statistikdaten pro Location und flüchtige Live Daten für alle Locations ist die Aggregation immer über eine ganze Tabelle möglich. Das verhindert rechenintensive `WHERE` Abfragen in MySQL Befehlen.

Highcharts rendert dabei die Statistiken in SVG Dateien die erst vom Browser gerendert werden. Das nimmt zusätzliche Last vom Server, welche bei serverseitigen Lösungen anfallen würden. Zusätzlich minimiert es die erforderliche Datenmenge für die Übertragung.

Google Maps API

Um dem Nutzer eine Vorstellung vom Standort der Wetterstation geben zu können haben wir auf der Website sowie in der App die Google Maps API implementiert. Auf unserer Datenbank sind zu jeder Location die Geokoordinaten hinterlegt.

```
1  function geolocMap(locname, lati, longi){
2    var position = new google.maps.LatLng(lati, longi);
3
4    var mapOptions = {
5            center: position,
6            zoom: 8,
```

```
7             mapTypeId: google.maps.MapTypeId.ROADMAP
8       };
9       var map = new google.maps.Map(document.getElementById("
        map_canvas"), mapOptions);
10    //marker
11    var marker = new google.maps.Marker({
12      position: position,
13      map: map,
14      title: locname
15    });
16  }
```

Codebeispiel 6: Aus den Geokoordinaten wird eine Karte generiert

Um unnötigen Traffic zu verhindern wird die API nur eingebunden, wenn der User tatsächlich die Kartenfunktionalität aufruft:

```
1  function loadMapsAPI(){
2      var script = document.createElement("script");
3      script.type = "text/javascript";
4      script.src = "http://maps.googleapis.com/maps/api/js?sensor=
       false&callback=geolocMap_helper";
5      document.body.appendChild(script);
6  }
```

Codebeispiel 7: Lade die Google Maps API

5.2 Phonegap App

Das freie Framework Phonegap stellt Webapplikationen native APIs mobiler Geräte zur Verfügung, so dass sich installierbare Anwendungen mit Webtechniken wie HTML5, Javascript und CSS entwickeln lassen. Phonegap unterstützt die mobilen Betriebssysteme iOS, Android, Blackberry OS, WebOS, Symbian, Bada sowie Windows Phone.

Hybride App in HTML5, CSS und JavaScript geschrieben mit jQueryMobile als unterstützendes Framework. Anschließend mit Phonegap in einen nativen Container verpackt um native Smartphone Funktionen nutzen zu können:

- Notification: Vibration, native Alert/Confirm
- Storage: Local & Session Storage
- Events: Backbutton event
- Native Push Notifications

Die Struktur

Die App ist so aufgebaut, dass alle Kontextseiten in einer HTML-Datei deklariert sind. Dies hat den Vorteil, dass die DOM Manipulation aller Unterseiten sehr leicht durchgeführt werden kann und sich die Ladezeiten der Unterseiten verringern. Hierzu wird jede Seite in einem DIV-Container deklariert.

```
<div data-role="page" id="map">
  <div data-role="header" data-position="fixed" data-theme="d">
    <h1 class="location"></h1>
  </div>
  <div data-role="content">
    <div id="map_canvas" style="width:100%; height:100%"></div>
  </div><!--content-->
</div><!--page-->
```

Codebeispiel 8: Deklarierung einer Seite

Das Laden sowie das Rendering der einzelnen Seiten werden von dem jQuery-Mobile Framework über Ajax realisiert.

Mit Hilfe der nativen Funktionalität, die die Phonegap API bietet, wurde das Herzstück der Mobile App realisiert:

Der intelligente Wecker

Die Idee

Der Benutzer soll einen Wecker stellen können, der ihn abhängig von der zum Weckzeitpunkt herrschenden Wettersituation alarmiert. Diese Weckfunktion soll selbst funktionieren, wenn die App zum Weckzeitpunkt nicht läuft und das Smartphone sich im Standby-Modus befindet.

Die Realisierung

Um den Nutzer unabhängig vom Zustand der App Benachrichtigungen zustellen zu können nutzen wir die Nativen Push Notifications. Auf Grundlage des Cordova Push Notifications Plugins for Android and iOS [**cordova-pp**] wurde die Funktionalität zum Empfangen der Notifications implementiert.

Das Zusammenspiel der einzelnen Komponenten soll im Folgenden am Beispiel des Android OS beschrieben werden.

Um einem Android Device Notifications schicken zu können benötigt man die Google Cloud Messaging API. Um diese verwenden zu können benötigt man eine Project-ID sowie einen Server-Key, welche nach Registrierung als Developer bei Google beantragt werden können.

Push Notifications mithilfe des Google Cloud Messeging Services

1. Android Device sendet Project-ID und Application-ID zum GCM server zur Registrierung.

2. Nach erfolgreicher Registrierung generiert GCM eine Registration-ID, welche unsere App auf dem Device des Nutzers eindeutig identifiziert, und schickt diese an das Android Device zurück.

3. Nach Erhalt dieser schickt das Device die Registration-ID an unseren Server.

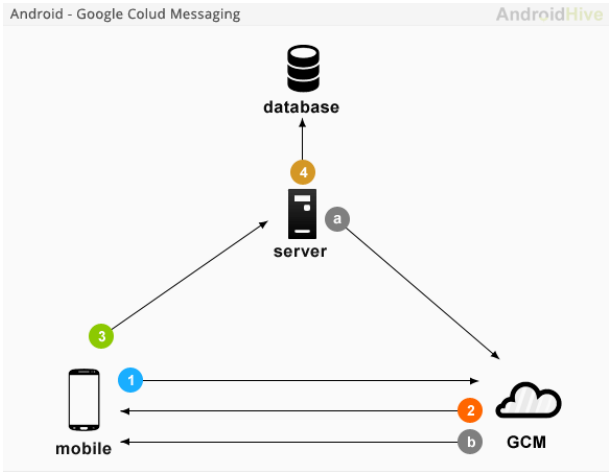


Abbildung 10: Kommunikation mit den Endgeräten über GCM [**gcm-architecture**]

4. Unser Server speichert die Registration-ID in unserer Datenbank. Wir benötigen sie um eine Notification an dieses Device zu senden.
   (a) Immer wenn wir eine Push-Notification versenden möchten startet unser Server mit PHP cURL einen POST-Request an den GCM-Dienst mit der Registration-ID, dem Server-Key und einem JSON-Objekt, das den Daten-Payload unserer Notification erhält.
   (b) Wenn der Server-Key uns dazu autorisiert an die Registration-ID eine Notification zu senden, leitet der GCM-Server den Daten-Payload an das zugehörige Device weiter.

```
1  public function send_notification($registatoin_ids, $message) {
2      // include config
3      include_once './config.php';
4      // Set POST variables
5      $url = 'https://android.googleapis.com/gcm/send';
6
7      $fields = array(
8          'registration_ids' => $registatoin_ids,
9          'data' => $message,
10     );
11
12     $headers = array(
13         'Authorization: key=' . GOOGLE_API_KEY,
14         'Content-Type: application/json'
15     );
16     // Open connection
17     $ch = curl_init();
18
19     // Set the url, number of POST vars, POST data
```

```
20      curl_setopt($ch, CURLOPT_URL, $url);
21
22      curl_setopt($ch, CURLOPT_POST, true);
23      curl_setopt($ch, CURLOPT_HTTPHEADER, $headers);
24      curl_setopt($ch, CURLOPT_RETURNTRANSFER, true);
25
26      // Disabling SSL Certificate support temporarly
27      curl_setopt($ch, CURLOPT_SSL_VERIFYPEER, false);
28
29      curl_setopt($ch, CURLOPT_POSTFIELDS, json_encode($fields));
30
31      // Execute post
32      $result = curl_exec($ch);
33      if ($result === FALSE) {
34          die('Curl failed: ' . curl_error($ch));
35      }
36
37      // Close connection
38      curl_close($ch);
39  }
```

Codebeispiel 9: PHP Funktion sendet POST-Request via cURL an GCM-Server

Erhält das Device eine Notification wird unsere App auf diesem gestartet um die Notification zu verarbeiten. Um die gewünschte Funktionalität, nämlich das aufwecken des Nutzers zu erreichen, musste in den nativen Java-Code des Push-Plugins eingegriffen werden.

```
1   private void tryPlayRingtone()
2   {
3       try {
4           Uri alert =  RingtoneManager.getDefaultUri(RingtoneManager.
        TYPE_ALARM);
5           mMediaPlayer = new MediaPlayer();
6           mMediaPlayer.setDataSource(this, alert);
7           final AudioManager audioManager = (AudioManager)
        getSystemService(Context.AUDIO_SERVICE);
8           if (audioManager.getStreamVolume(AudioManager.STREAM_RING)
        != 0) {
9               mMediaPlayer.setAudioStreamType(AudioManager.
        STREAM_RING);
10              mMediaPlayer.setLooping(true);
11              mMediaPlayer.prepare();
12              mMediaPlayer.start();
13          }
14      }
15      catch(Exception e) {
16      }
17  }
```

Codebeispiel 10: Java Code der Weckfunktion

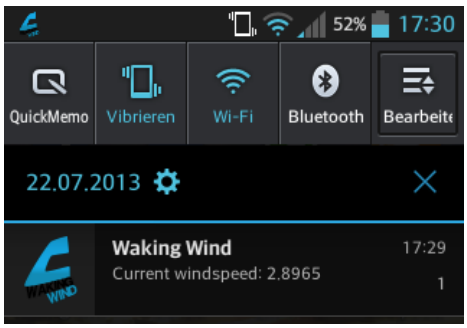
Hier ist verankert, dass bei eintreffen einer Notification ein Klingelton sowie ein Vibrationsalarm so lange ausgeführt werden, bis der Nutzer unsere Notification antippt.

```
1   public static void cancelNotification(Context context)
2   {
```

```
3       NotificationManager mNotificationManager = (NotificationManager
        )context.getSystemService(Context.NOTIFICATION_SERVICE);
4       mNotificationManager.cancel((String)getAppName(context),
        NOTIFICATION_ID);
5       //cancel vibration and alarm
6       v.cancel();
7       mMediaPlayer.stop();
8       notofied=false;
9   }
```

Codebeispiel 11: Beim Antippen der Norification

Ist dies geschehen öffnet sich unsere App im Kontext der Notification.



Nach Erhalt der Notification wird der Alarm ausgelöst und diese in der Taskbar angezeigt.

Nach antippen der Notification wird die App im Kontext dieser geöffnet und gibt dem Nutzer Auskunft über Location und Windgeschwindigkeit.

Das Senden der Notifications geschieht durch unseren Server. Hier ist ein Cronjob angelegt, der alle 15min ein PHP-Script ausführt, welches in der Datenbank prüft, ob ein Nutzer zu einer Location einen Alarm gestellt hat und wenn ja,

ob die gewünschte Windgeschwindigkeit im Durchschnitt innerhalb der letzten fünf Minuten überschritten wurde. Ist dies der Fall wird dem Nutzer über seine Registration-ID eine Notification versendet.

Automatischer Login mit der Registration-ID

id	gcm_regid
3	APA91bEtXhZXnI78a08X714opPhU-u-Opcv-wFTtWB3T7YBhsK...
1	APA91bEE_EdM4hhjVoSNHXZw2s6uUBYDdbrHaOHGHAsC1p9XEv...
11	APA91bFlyt5W38sGsEF85KAz8ivXGr0WBzF9CQGOuHtqRh460N...
12	APA91bH7j7IuLOMzPpv2UPR6Uyi5WmLy7nTVeRluoHDWh6Oyle...
8	APA91bFnLLuWUmtJGM-KwB-8SqoJQXBqt0HKsgWpK0-EUEp8I1...
10	APA91bHAlsjFGtfDjAy3B5GfTPYv-LGCQG0BCxioKqWu1gXmYQ...

Abbildung 11: Die Registration-IDs werden auf unserer Datenbank gespeichert

Da die Registration-ID ein Device eindeutig identifiziert lag die Entscheidung nahe damit ein automatisches Login Verfahren zu implementieren. Wenn der Nutzer auf einen geschützten Bereich zugreifen möchte, klopft das Device beim GCM-Server an und lässt sich seine Registration-ID geben. Jetzt wird überprüft, ob diese Registration-ID schon auf unserer Datenbank hinterlegt wurde. Ist dies der Fall wird der Nutzer automatisch eingeloggt. Andernfalls muss er sich mit E-Mail-Adresse und Passwort anmelden. Ist die Kombination korrekt, wird die Registration-ID auf unserer Datenbank gespeichert. Fortan wird der Nutzer automatisch eingeloggt.