AIR ZERMATT AG
RAPHAEL ANDRES

Dokumentation Air Zermatt App

INHALT

I. VORWORT / EINLEITUNG

1 DIE IDEE

Wie im Pflichtenheft bereits erwähnt, existiert bereits eine Applikation für Apple-Geräte. Da diese Applikation aber erstens nicht viel zu bieten hat, zweitens zwei Franken kostet und drittens nur für iOS verfügbar ist, war es bereits lange ein Wunsch der Air Zermatt eine neue, überarbeitete Version anzubieten. Diese sollte auf den gängigsten Plattformen verfügbar und vor allem kostenlos sein. Ausschlaggebend war ein Input eines Mitarbeiters an einer Marketing-Sitzung. Man müsste die Kunden-Daten unserer Rundflug-Gäste besser abholen, war sein Vorschlag. Wir sahen darin eine mögliche Projektarbeit für mich und besprachen diese Idee. Schlussendlich kamen wir zum Entscheid, dass ich mich darüber informieren soll, ob ich so eine Applikation realisieren kann.

2 DIE VORBEREITUNG

Ich sah dieses Vorhaben, mit Hilfe von Cordova, als realistisches Projekt an, wobei ich die Voraussetzung hatte, dass ich mich für solch ein Projekt ausreichend vorbereiten kann. Ich informierte mich im Internet, nach passenden Lösungen und gängigen Praktiken. Da ich bisher nur mit Webtechnologien Erfahrung hatte, musste ich ein Tool suchen, das mir erlaubt mit meinen Fähigkeiten eine Mobile Applikation zu erstellen. Mit Cordova, oder auch PhoneGap genannt, habe ich die ideale Lösung gefunden.

3 PERSÖNLICHE ZIELE

Ich wollte mit diesem Projekt herausfinden, wo die Grenzen der Webtechnologien sind. Ich habe bisher während meiner Lehre zwei grössere Webapplikationen realisiert. Dabei habe ich gemerkt, dass einem Web Developer mächtige Werkzeuge in die Hände gedrückt werden. Die Fantasie kann noch so gross sein, irgendwie kann mit HTML, CSS, JavaScript, PHP und MySQL alles realisiert werden. Da ich mit diesen beiden Projekten einige Erfahrungen mit der Programmiersprache JavaScript gemacht habe, wollte ich noch mehr über diese Sprache lernen.

Weiter interessierte ich mich sehr für die Entwicklung von Mobilen Applikationen und die mobilen Betriebssysteme allgemein. Da war es für mich eine willkommene Einladung in diesem Gebiet weitere Erfahrungen zu sammeln.

4 ZIELE DES PROJEKTS

Von diesem Projekt erwarten wir eine funktionierende, mobile Applikation, welche auf den drei Plattformen Android, iOS und Windows Phone läuft. Diese App soll ein weiteres Element der Kundendatensammlung darstellen.

Weiter soll sich die App optisch am Corporate Design der Air Zermatt orientieren aber auch die Looks and Feel der spezifischen Plattform berücksichtigen.

Es soll den Zeitaufwand für Mitarbeiter der Einsatzleitung beim Benachrichtigen von Kunden verkürzen. Durch das Back-End wird eine Benachrichtigung einmalig erstellt und über den Benachrichtigungsdienst an alle Interessenten gesendet.

Diese App soll in Zukunft die alte, funktionsarme Applikation für iOS-Geräte ersetzen

5 AUFBAU DIESER DOKUMENTATION

Da dieses Projekt eine lange und wichtige Vorbereitungs-Phase hatte, habe ich mich vorgängig dazu entschieden diese Phase ebenfalls zu dokumentieren. In dieser Phase werden grundlegende Sachverhalte erklärt. So zum Bespiel mit welchen Frameworks gearbeitet wird und wie diese aufgebaut sind und funktionieren. Weiter werden die Schritte beim Vorbereiten der Programmier- und Testumgebung beschrieben. Dieser erste Teil befindet sich im Kapitel Vorarbeit.

6 VORAUSSETZUNGEN

Dieses Projekt setzt gute Kenntnisse mit Webtechnologien voraus. Serverseitig, beim Back-End sind vorwiegend PHP und MySQL angewendet worden. Für die App wurde vor allem mit JavaScript, aber auch mit HTML und CSS gearbeitet.

Damit mit Cordova gearbeitet werden kann, werden spezielle Programme gebraucht. Wie man diese Programme bereitstellt ist auf den nächsten Seiten beschrieben.

Weiter werden echte Smartphones vorausgesetzt, auf denen man die App laufend testen kann. Mit dem Emulator lassen sich nicht alle Funktionen real testen.

II. VORARBEIT

1 INTERNE BESPRECHUNGEN / PROJEKT-IDEEN

Nach dem Informationsabend über die IPA für die Kandidaten habe ich mit den involvierten Personen eine Besprechung abgehalten, ob eine Mobile Applikation für den Betrieb einen Nutzen darstellt. Wir haben Vor- und Nachteile diskutiert, mögliche Funktionen und erste Grobziele definiert. Schlussendlich sind wir zum Entschluss gekommen, dass wir, wenn es auch von den Experten so genehmigt wird, das Projekt Air Zermatt App in Angriff nehmen.

Am selben Tag erfolgte der Informationsabend für die Fachverantwortlichen.

2 ZEITMANAGEMENT

Um die App in der kurzen Zeit realisieren zu können (max. 90 Std.), möchten wir die Planung / Konzeption als Vorarbeit einreichen. Dazu zählt auch das Kennenlernen mit AngularJS und Cordova, sowie für eine praktische Übung eine Vorab-Applikation, die z.B. Daten lokal speichert, und Push Benachrichtigungen erhält. Das eigentliche Projekt wäre dann die Realisierung dieser plattformübergreifenden App.

3 STARTSCHUSS IDEEN-POOL

Ich werde mich über die gängige Praxis und einen effizienten Workflows informieren, in einem Pflichtenheft festhalten, was die Applikation alles draufhaben und was der Nutzen für den Betrieb darstellten soll. Zudem informiere ich mich durch Dokumentationen und Tutorial (video- und textbasiert) über die Funktionen und deren Erstellung, da ich zum Ersten Mal eine solche App für das Smartphone kreieren werde.

4 ERKENNTNISSE DER RECHERCHEN

4.1 Frameworks & Programmierumgebung

Ein geeignetes **Framework** zur Erstellung der Applikation ist Cordova. Bei diesem Framework kann man mit Hilfe von Standard Web Technologien, also von HTML, CSS und JavaScript, eine "Website" entwickeln, welche dann in einem Container umgewandelt und für die native Datenverarbeitung bereitgestellt wird. Durch sogenannte Plugins können auf native Gerätefunktionen zugegriffen werden. Durch die **Programmierschnittstelle** des Frameworks lässt sich so eine plattformunabhängige Applikation erstellen.

Programmieren kann man diese "Website" auf dem lokalen Rechner. Auf dem Windows-Rechner lässt sich das App für Android und Windows Phone, auf einem Mac das App für Android

und iOS programmieren. Dazu werden einige Programme benötigt, wie z.B. das SDK (**Software Development Kit**) der betreffenden Plattform, aber auch plattformunabhängige Programme, wie das **Java Development Kit** von Oracle oder ein **git** client.

Idealerweise möchte ich eine minimale Applikation im Voraus erstellen, um den Umgang mit dem Framework und den dazugehörigen Programme zu üben.

4.2 Weitere Frameworks

Cordova lässt sich mit weiteren Frameworks erweitern. So kann z.B. eine an das Ausgabegerät angepasst Oberfläche gewährleistet werden, so dass der Benutzer die App über das ihm bekannte Interface bedienen kann. Dazu könnte man das Framework **ionic** benutzen. Um das Design an die **Look and Feels** der verschiedenen Plattformen anzupassen, gibt es auch verschiedene Frameworks wie Angular **Material** für Android oder Chocolate Chip für iOS und Windows Phone.

Update: Das Framework Chocolate Chip wurde aufgekauft und ist nicht mehr öffentlich zugänglich.

4.3 Unterstütze Plattformen

In erster Linie wollen wir die Applikation für die Plattformen Android, iOS und Windows Phone anbieten. Eine wichtige Erkenntnis hierbei ist sicherlich, dass sich die Plattformen auch weiterentwickeln. Ende 2015 sind zum Beispiel Android 5 Lollipop, iOS 9 und Windows Phone 8.1 noch aktuell. Im Sommer des nächsten Jahres werden diese Versionen vielleicht schon wieder veraltet sein. Windows 10 wurde herausgegeben, Android 6 Marshmallow wird bald erhältlich sein, etc.

4.4 Programmiersprachen

Um diese App zu entwickeln, haben wir uns extra ein Framework herausgesucht, das auf meinem jetzigen Wissen aufbaut, also auf die Standard Web Technologien HTML, CSS, ein wenig JavaScript und im weitesten Sinne PHP. Allerdings wird zur Entwicklung von einer solchen App **AngularJS** (ein ähnliches JavaScript Framework wie z.B. jQuery) empfohlen, weil sich damit hervorragend eine sogenannte **Single Page Application** (SPA) entwickeln lässt. Also eine Applikation, die auf einem einzigen Script bleibt und die benötigten Daten asynchron (siehe AJAX) in dieses Script ladet. Deshalb möchte ich vor dem Projekt die Grundlagen von AngularJS kennenlernen.

4.5 Third-Party Logistik

Weitere Anbieter (Drittanbieter) werden beispielsweise gebraucht, um eine Benachrichtigung zum Smartphone des Kunden zu senden. Dazu stellen die Hersteller von den Smartphones eigene Dienste / Sever an, die die sogenannten **Push-Notification** managen.

4.6 Publizieren der App

4.6.1 bei Android

Um die fertige App auf dem **Play Store** von Google zu veröffentlichen, braucht man einen Developer-Account. Unter diesem **Link** findet man eine Checkliste von Google, in der die Schritte zum Publizieren der App aufgeführt sind. Eine sehr nützliche **Checkliste** vor dem finalen Publizieren gibt es auf der Website von Google.

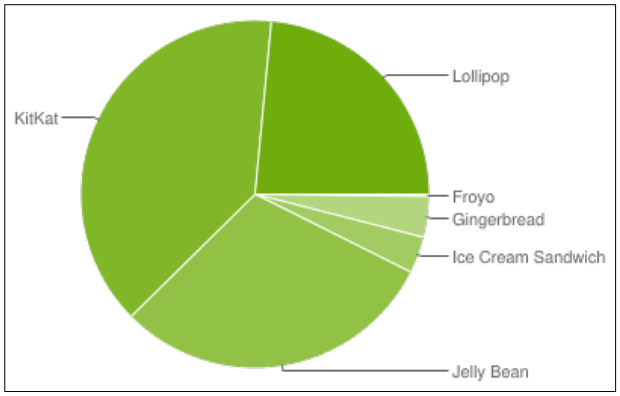


Abbildung 1 Kreisdiagramm Verbreitung der Android-Versionen

Quelle: **http://developer.android.com/about/dashboards/index.html**, Stand: 05.10.2015

4.6.2 bei Apple

Auch zum Publizieren von Apps im Apple App Store gibt es eine **Anleitung**.

4.6.3 bei Windows

Bei Windows findet man die Anleitung zum Publizieren der App **hier**.

5 ZIELGRUPPE

Als Zielgruppe sehe ich alle Kunden der Air Zermatt, welche über aktuelle Angebote und Neuigkeiten informiert werden wollen.

6 FUNKTIONSUMFANG DER APP

Eine primäre Funktion der App soll dem Kunden das Organisieren eines Rundflugs erleichtern. So wäre ein erstes Ziel, dass ein Kunde, der sich für einen oder mehrere bestimmten Rundflug-Routen interessiert, eine Benachrichtigung erhält, wenn es noch freie Plätze für diesen Rundflug gibt.

6.1 Befragung der betreffenden Personen (EL, Kunden)

Um Ideen zu weiteren nützlichen Funktionen der App zu erhalten, erstelle ich einen Ideen-Pool. Zudem befrage ich Mitarbeiter, welche Funktionen für Sie einen Vorteil darstellen würden. Aus all diesen Vorschlägen entnehmen wir dann die besten Ideen und sortieren sie nach Priorität.

7 ERSTE APPLIKATION ZUM TESTEN

Ich habe mich dazu entschieden, vorab schon mal eine einfache App zu entwickeln, um den Umgang mit den ganzen Komponenten (**SDK**, **CLI**, **AngularJS**, etc.) kennenzulernen. Ziel dieser ersten App soll es sein, gewisse Termine für ein Anlass oder Event festzulegen und andere Nutzer der App via Push-Nachricht darüber zu informieren.

7.1 Vorbereitung

Als Vorbereitung für die erste App installiere ich die notwendigen Tools zum Entwickeln für Windows Phone und Android. Diese sind: Das JDK 8 (Java Development Kit) von **Oracle**, Die Stand-alone SDK Tools (Software Development Kit) von **Android**, sowie das Visual Studio in der Express Version von **Windows**. Die Bereitstellung dieser Tools nimmt eine gewisse Zeit in Anspruch. Besonders die Installation der Android SDKs kann Stunden dauern. Ich entschliesse mich für die Test-App eine Unterstützung für Android bis Version 4.3 zu gewährleisten. Die APIs von früheren Versionen installiere ich nicht mehr. Eine gute Referenz, welche Plattformversionen aktuell am meisten genutzt werden, hat man beim **Dashboard** von Google. Diese Arbeitsschritte habe ich aus dem Videotraining **Mobile Apps mit Cordova** von video2brain.

Sind diese Tools installiert kann man an die Installation von Cordova gehen. Hierzu werden auch nochmals gewisse Programme benötigt: **node.js**, einen git-Client und speziell für Android **Apache Ant**. Wichtig ist, dass nach der Installation dieser Programme, die Umgebungsvariablen erweitert werden. So muss eine Variable *JAVA_HOME* hinzugefügt werden mit dem

Pfad zur JDK-Installation (z.B.: C:\Program Files\Java\jdk1.8.0_60). Darüber hinaus muss der Installationspfad von git und von Apache Ant zu der Variable *path* hinzugefügt werden.

Möchte man für Android entwickeln, so müssen darüber hinaus noch drei weitere Pfade zu den Umgebungsvariablen hinzugefügt werden. Diese sind der Ordner tools in der SDK-Installation von Android und der Ordner plattform-tools. Weiter sollte eine Systemvariable namens *ANDROID_HOME* und dem Pfad zum Ordner tools der SDK-Installation hinzugefügt werden.

7.2 Installation von Cordova

Die eigentliche Installation von Cordova geht via CLI (Command Line Interface) ganz schnell. Der Aufwendige Teil der Installation ist die Vorbereitung der ganzen Tools und das Setzten der Umgebungsvariablen. Um Cordova zu installieren öffnet man bei Windows die Eingabeaufforderung (cmd) und gibt folgender Befehl ein:

```
1  npm install -g cordova
```

Cordova wird nun unter den AppData des aktuellen Benutzers installiert. Neben Cordova selbst installiere ich ngCordova. Das ist ein Framework, welches Cordova und AngularJS verbindet, so dass direkt und einfach entwickelt werden kann.

(Quelle: **http://ngcordova.com/**)

7.3 Anlegen der Projektdateien und Plattformen

Sind alle benötigten Frameworks installiert, kann mit dem Erstellen der Projektdateien begonnen werden. Dazu öffnet man wiederum das CMD bzw. Terminal im gewünschten Ordner und führt folgenden Befehl aus:

```
1  cordova create app com.example.app MyApp
```

Dieser Befehl erstellt im gewünschten Ordner einen Unterordner namens "app" indem die Grunddateien hinterlegt werden. "com.example.app" bezeichnet den Herausgeber und "MyApp" ist der sichtbare Name der App. Anschliessend habe ich benötigte Dateien, wie die von Angular oder ngCordova in einen Unterordner "/lib" hinzugefügt. Dieses Grundgerüst findet man auf der beiliegenden CD.

Beim Einbilden der verschiedenen Frameworks dürfen die JavaScript-Files am Schluss der index.html eingefügt werden. Es muss nur die richtige Reihenfolge beachtet werden:

angular.js

ng-cordova.js

cordova.js (File wird beim builden hinzugefügt)

weitere Komponente von AngularJS

app.js

controller.js

Nun fügt man die Plattformen hinzu, für die man die Applikation realisieren will:

```
1  cordova platform add android
```

Sobald eine Plattform erstellt ist, kann man die aktuelle App in einer Testumgebung ausprobieren. Das Setup der App-Umgebung ist nicht so einfach.

7.4 Vorbereitung der Testumgebung

Um die entstehende App laufend zu Testen sollte man sie direkt auf richtigen Smartphones testen. Damit man nicht ständig händisch die App generieren, diese mühselig auf das Gerät kopieren und dort installieren muss, kann man dank den Entwickleroptionen dieser Prozess vereinfachen und automatisieren:

7.4.1 Testen bei Android

Dazu muss bei Android das USB-Debugging auf dem Device eingeschaltet sein, und die richtigen adb-Treiber auf dem Computer installiert sein. Ob das System das Smartphone erkennt, kann man mit folgendem Befehl in der Eingabeaufforderung überprüfen:

```
1  adb devices
```

Wird dort nun mindestens ein Device angezeigt, kann Cordova die erste App direkt auf das Gerät kopieren, installieren und ausführen.

Wird das verbundene Gerät nicht gefunden, kann es sein, das nicht alle benötigten Treiber installiert sind. In diesem Fall wird im Geräte-Manager das Gerät mit einem gelben Ausrufenzeichen dargestellt. Je nach Gerät reicht es schon aus, den Google Universal Treiber zu installieren.

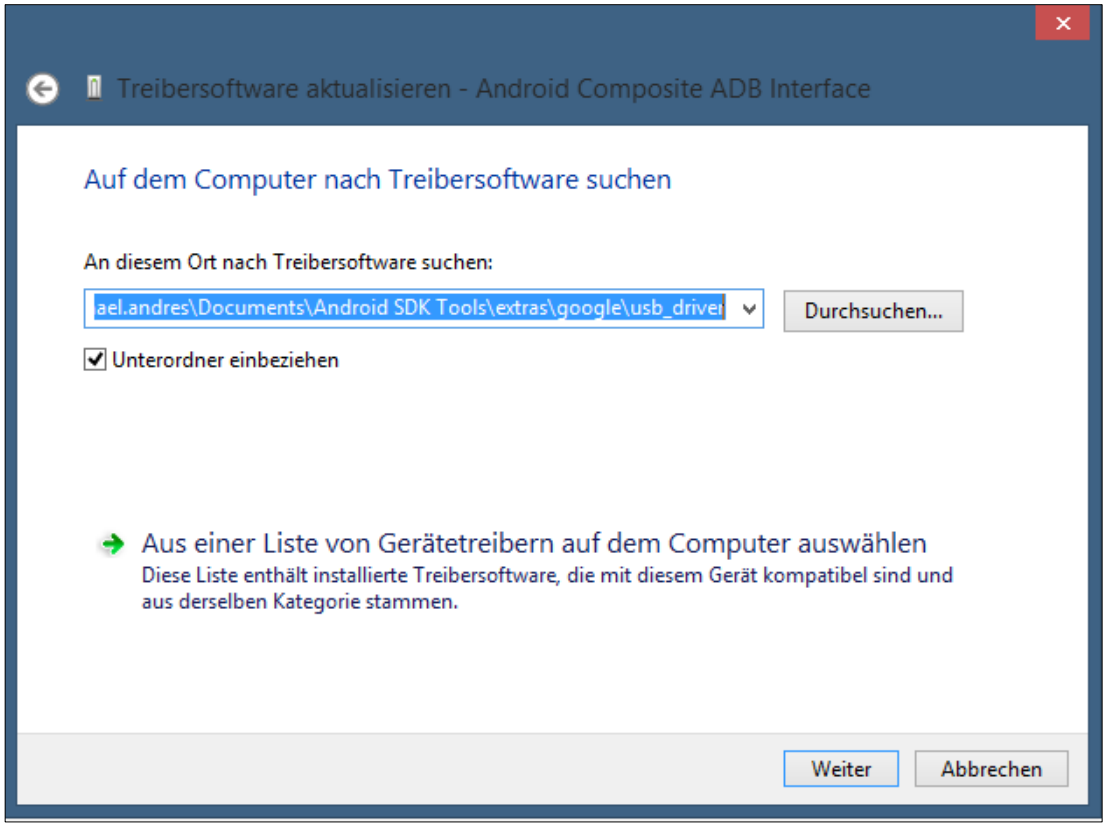


Abbildung 2 Google Universal ADB Treiber installieren

Manchmal reicht das schon aus und der Treiber wird erfolgreich installiert. Andernfalls muss ein etwas aufwändiger Workaround getätigt werden. Eine ausführliche Anleitung, welche ich auch schon mal benutzt habe, ist in diesem **Forum-Beitrag** zu finden, wobei mir die ersten beiden Arbeitsschritte gereicht haben, um den Treiber erfolgreich zu installieren.

Auch bei Cordova kann man nochmals überprüfen ob das gewünschte Smartphone erkannt wurde. Hier ein Android Gerät:

```
1  cordova run android --list
```

und Ausführen kann man die App via:

```
1  cordova run android --device
```

7.4.2 Testen bei iOS

Wenn man für iOS entwickelt, war jahrelang eine Registrierung beim Apple iOS Developer Program zwingend um die App auf dem iPhone zu testen. Dort fielen bereits die 99$ pro Jahr an. Mit der neuen Version von Xcode 7 (Stand: Ende 2015) ist das Testen nun auch ohne Online-Registrierung möglich. Das iPhone muss am Mac angeschlossen werden. Wird das Gerät erkannt, wird dieses bei Xcode direkt als Zieldevice erkannt und man kann den Build-Prozess starten. Allfällige Fehler werden automatisch behoben, man muss sich lediglich erneut mit einer Apple-ID anmelden, dann wird das Gerät automatisch als Entwickler-Gerät registriert.

7.4.3 Testen bei Windows Phone

Um eine App auf einem Windows Phone zu testen muss dieses Gerät registriert werden. Ein Device kann über das Windows Phone SDK mit einem Developer-Account zum Testen freigeschaltet werden. **Anleitung**

7.5 Google Play Publisher Account

Viel einfacher und schneller geht die Registration bei Google. Man braucht dazu eine Google-Konto, welches dann mit der Developer Console verknüpft wird.

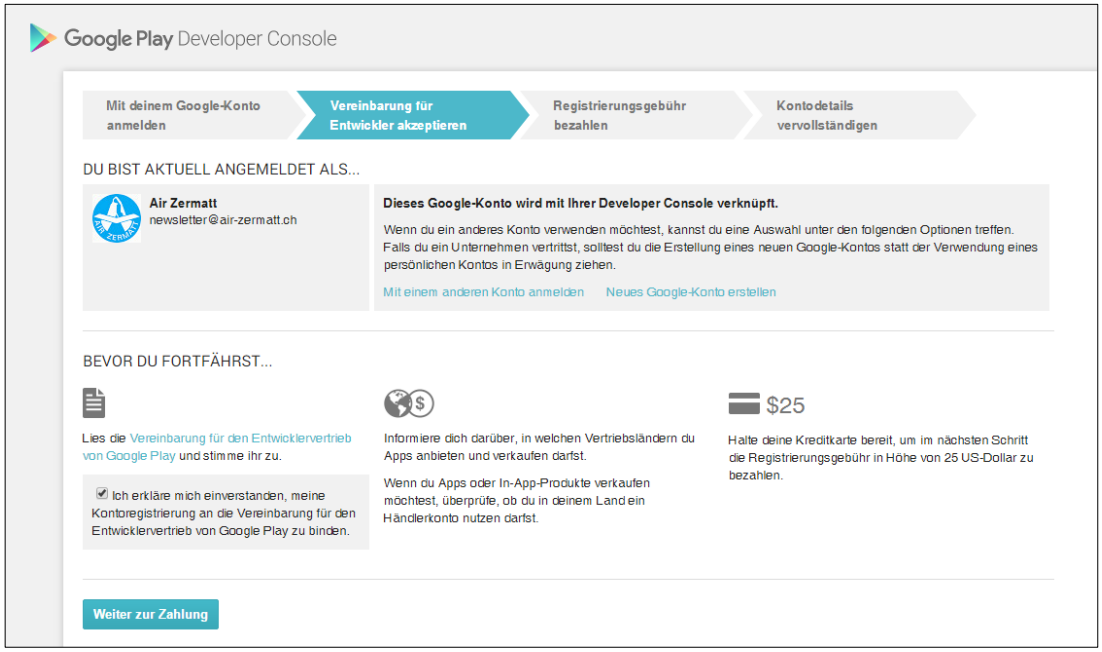


Abbildung 3 Schritte zum Google Developer Account

Mit einem Google Payment Account, werden dann die einmaligen Kosten von 25$ bezahlt. Ist das passiert, gelangt man direkt zur Developer Console und könnte die erste APK-Datei hochladen. Viel komplizierter funktioniert der Anmeldeprozess bei Apple:

7.6 Apple Developer Programm

Damit man die Applikation auf einem Apple-Gerät testen und später auch auf dem Apple Store publizieren kann, braucht es ein Developer-Account. Es gibt einen Account für Einzelpersonen und einen für Unternehmen. Um den Anmeldeprozess für einen "Company-Account" künstlich in die Länge zu ziehen, braucht man zusätzlichen eine sogenannte D-U-N-S Nummer. Dies ist eine neunstellige Nummer, mit der man jedes Unternehmen eines Landes eindeutig identifizieren kann. Hat man noch keine solche Nummer kann man diese kostenlos registrieren. Nach zirka einem Monat erhält man dann die D-U-N-S Nummer und nach weiteren zwei Wochen kann man sich mit dieser Nummer zum Apple Developer Programm anmelden, (man verliert also etwa anderthalb Monate für die Registration). Glücklicherweise hatte die Air Zermatt AG bereits so eine D-U-N-S Nummer. Jedoch stimmte die Anschrift des Hauptsitzes nicht mehr. Ich musste via E-Mailverkehr die Änderung der Adresse beantragen. Die Änderung sollte dann innert 2 Wochen durchgeführt werden.

7.7 Registration zum Windows App Developer

Bei Windows ist der Anmeldeprozess ähnlich trivial wie bei Google. Man braucht ein Microsoft-Konto welches dann mit dem Dev Center verknüpft wird. Nach dem Angeben der notwendigen Informationen und der Zahlung von CHF 90.- gelangt man direkt zum Dashboard. Dort kann man direkt den Namen einer ersten Applikation reservieren.

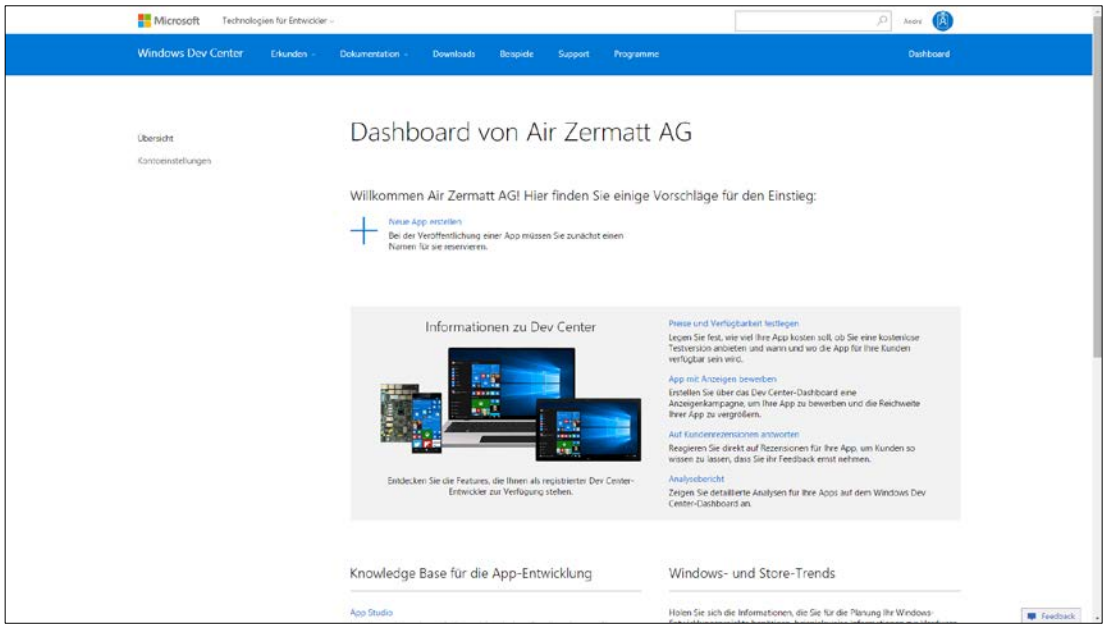


Abbildung 4 Dashboard Windows Dev Center

7.8 Push Notifications einrichten

Bei der Test-App möchte ich mir schon mal das Einrichten des Push-Nachrichten-Dienst genauer ansehen. Bei Android-Geräten wird dieser Dienst von Google angeboten und heisst

Google Cloud Messaging, kurz GCM. Bei Apple-Geräten heisst der Dienst Apple's Push Notification Service, kurz APN und bei Windows: Microsoft Push Notification Service (MPNS).

Grundsätzlich lässt sich der Ablauf einer Benachrichtigung folgendermassen erklären:

1. Der User installiert die App, dabei kann er auswählen, ob er Benachrichtigungen von dieser Applikation erhalten will oder nicht.

2. Wenn der User Benachrichtigungen erhalten möchte, baut das Smartphone eine Verbindung mit dem Push-Notification-Dienst auf, welcher mit einem einmaligen Key antwortet.

3. Dieser Key kann dann zum Entwickler gesendet werden. Der Entwickler speichert diese Keys beispielsweise in einer Datenbank.

4. Um nun dem User eine Benachrichtigung zu senden, muss der Entwickler die Daten (Message Payload genannt) zum Push-Notification-Dienst senden mit den betreffenden Key des Users. Neben der eigentlichen Nachricht können auch andere nützliche Variablen übermittelt werden.

5. Der Dienst sendet dann diese Daten zum Smartphone und übergibt sie der betreffenden App. Läuft die Applikation in dem Moment nicht, so wird diese automatisch gestartet.

6. Die App erstellt dann die Push-Benachrichtigung in der Statuszeile oder via alert-Box auf dem Bildschirm.

7.8.1 Einrichtung GCM-Account

Um Push Nachrichten an ein Android-Gerät zu senden, braucht man einen Account beim **Google Cloud Messaging**. Man richtet sich ein Projekt an und erhält eine Projekt-Nummer. Zusätzlich wird dort ein sogenannter Server-Key generiert, welcher bei der Applikation hinterlegt werden muss. So wird durch eine zweifache Sicherheitsüberprüfung ein Missbrauch verhindert.

Man kann Nachrichten an bis zu 1000 registrierte Token auf einmal senden. Sind es mehr, so müssen die Token auf mehrere Sendungen verteilt werden. Dieses Limit muss in die Planung des Back-Ends einbezogen werden.

Jetzt müssen die Daten (Payload) zum Google Cloud Messaging-Dienst sendet werden. Dazu eignet sich PHP, da man mit cURL die Informationen zum betreffenden Server von Google senden kann. Folgender Code kann in einem PHP-File auf dem Server gespeichert und ausgeführt werden.

```
1  <?php
2  define( 'API_ACCESS_KEY', '[API_Schlüssel]');
3  $registrationIds = array('[id1]','[id2]', ...);
```

```
4  // prep the bundle
5  $msg = array(
6        'message'   => 'Das ist meine Nachricht!',
7        'title'     => 'Das ist ein Titel',
8        'style'     => 'inbox',
9        'summaryText'     => '%n% neue Nachrichten',
10       ...
11 );
12 $fields = array(
13       'registration_ids'      => $registrationIds,
14       'data'                  => $msg
15 );
16
17 $headers = array(
18       'Authorization: key=' . API_ACCESS_KEY,
19       'Content-Type: application/json'
20 );
21 $ch = curl_init();
22 curl_setopt( $ch,CURLOPT_URL, 'https://android.goog-
   leapis.com/gcm/send');
23 curl_setopt($ch,CURLOPT_POST, true);
24 curl_setopt($ch,CURLOPT_HTTPAUTH, CURLAUTH_BASIC);
25 curl_setopt($ch,CURLOPT_HTTPHEADER, $headers);
26 curl_setopt($ch,CURLOPT_IPRESOLVE, CURL_IPRESOLVE_V4);
27 curl_setopt($ch,CURLOPT_RETURNTRANSFER, true);
28 curl_setopt($ch,CURLOPT_SSL_VERIFYPEER, 0);
29 curl_setopt($ch,CURLOPT_POSTFIELDS, json_en-
   code($fields));
30 $result = curl_exec($ch);
31
32 curl_close($ch);
33 echo $result;
34 ?>
```

8 LOKALER SPEICHER / DATENBANK

Ein weiteres Gebiet, welches ich bei der ersten Testapp auch unbedingt kennenlernen will, ist die Speicherung von Informationen auf dem lokalen Gerät. Also zum Bespiel in einer Datenbank, die auf dem Smartphone des Kunden gespeichert wird. Dazu bietet Cordova verschiedene Möglichkeiten mit den unterschiedlichsten Vor- und Nachteilen. Ein Plugin, das alle drei Plattformen Android, iOS und Windows Phone unterstützt wird von ngCordova angeboten. Dieses Plugin ermöglicht das Erstellen und Nutzen von SQLite Datenbanken mit der HTML5/Web SQL API.

https://github.com/litehelpers/Cordova-sqlite-storage

9 SINGLE PAGE APPLICATION / TEMPLATES

AngularJS ist prädestiniert für sogenannte Single Page Applications, kurz SPA. Das heisst, eine einzige Seite wird einmalig geladen und die weiteren Unterseiten und sonstige Inhalte werden dynamisch geladen und ersetzen den vorherigen Inhalt. Dies hat den Vorteil, dass Ressourcen wie JS-Files, oder Icons bereits geladen sind, und nur noch der veränderte Inhalt geladen werden muss. Dieses Verfahren nennt sich bei AngularJS *angular-route* und ist ein JS-File welches in die index.html eingebunden wird. Die index.html kann nun als Grundgerüst verwendet werden. Man erstellt ein Container, in den dann die unterschiedlichen Seiten hineingeladen werden. Die Templates können in einen gleichnamigen Ordner gespeichert werden. Der Container braucht das Angular-Attribut *ng-view*. AngularJS erkennt dann den Container und fügt der Inhalt aus dem Template dort ein. Doch wie weiss Angular welches Template geladen werden muss? Dazu gibt es den *$routeProvider*. Damit kann man einem gewissen String im Pfad ein gewisses Template bzw. die dazugehörige Template-URL zuweisen.

```
1  app.config(function($routeProvider) {
2  $routeProvider.when('/second', {
3  templateUrl : 'templates/secondpage.html',
4  }).when('/', {
5  templateUrl : 'templates/startpage.html',
6  }).otherwise({
7  redirectTo: '/'
8  })
9  });
```

10CONTENT SECURITY POLICY

Content Security Policy ist ein wichtiges Sicherheitswerkzeug um XSS-Attacken abzuwehren. Da bei Cordova die Applikation mit Web-Technologien realisiert wird, ist auch Content Security Policy (CSP) ein zu beachtender Punkt. Damit kann definiert werden, welche Quellen

als sicher angesehen werden. Von anderen Quellen werden gar keine Daten entgegengenommen. Will man z.B. über eine AJAX-Abfrage Inhalte laden, muss der Server, an den die Anfrage geht als sichere Quelle angegeben werden, ansonsten erhält man keine Antwort. Diese "Whitelist" wird im HTML-Header als Meta-Tag hinterlegt:

```
10 <meta http-equiv="Content-Security-Policy"
11 content="default-src 'self'; style-src 'self'
12 'unsafe-inline'; connect-src http://www.air-zermatt.ch
13 'self'; ">
```

Diese Zeilen sagen dem Browser, dass grundsätzlich alle Elemente mit derselben Herkunft, wie das File selbst erlaubt sind. Zudem kann eine Verbindung (connect-src) mit dem Server **http://www.air-zermatt.ch** gemacht werden. Eine umfangreiche Dokumentation über CSP findet man unter **http://content-security-policy.com/**.

11 NATIVES LOOK & FEEL

Ein wichtiger Punkt der auch in das Konzept einfliessen muss, ist das Look and Feel der jeweiligen Plattformen. Bei jedem Hersteller gibt es dazu eine mehr oder weniger grosse Sammlung von Richtlinien in Sachen User Experience. Das Material Design wird von Google selbst angeboten. Mit Hilfe von CSS und JS-Files lassen sich damit diverse Webapplikationen gestalten. Auf ihrer Website **https://material.angularjs.org/** gibt es eine ausführliche Dokumentation über alle gestalterischen Elemente des Material Designs und jeweils einige Beispiele samt Beispiel-Code. Bei iOS und Windows Phone habe ich mich für das Framework **Chocolate Chips** entschieden. Das heisst, dass es einige Änderungen am Code geben wird.

Abbildung 5 Chocolate Chips

12ERFAHRUNGEN BEI DER VORAB-APPLIKATION

12.1 Emulator vs. Device

Das Starten des Emulators, besonders des Android-Emulators kann ewig dauern. Da ist es viel interessanter die App direkt auf das Gerät zu installieren und dort zu überprüfen. Durch den folgenden Befehl wird die App gebuildet und direkt auf dem angeschlossenen Gerät ausgeführt:

```
1  cordova run android --device
```

Zusätzlich kann man über den Browser Google Chrome auf Entwicklertools zugreifen und, da die App ja mit HTML, CSS und Co. programmiert ist, den Code validieren und etwaige Fehler anzeigen lassen:

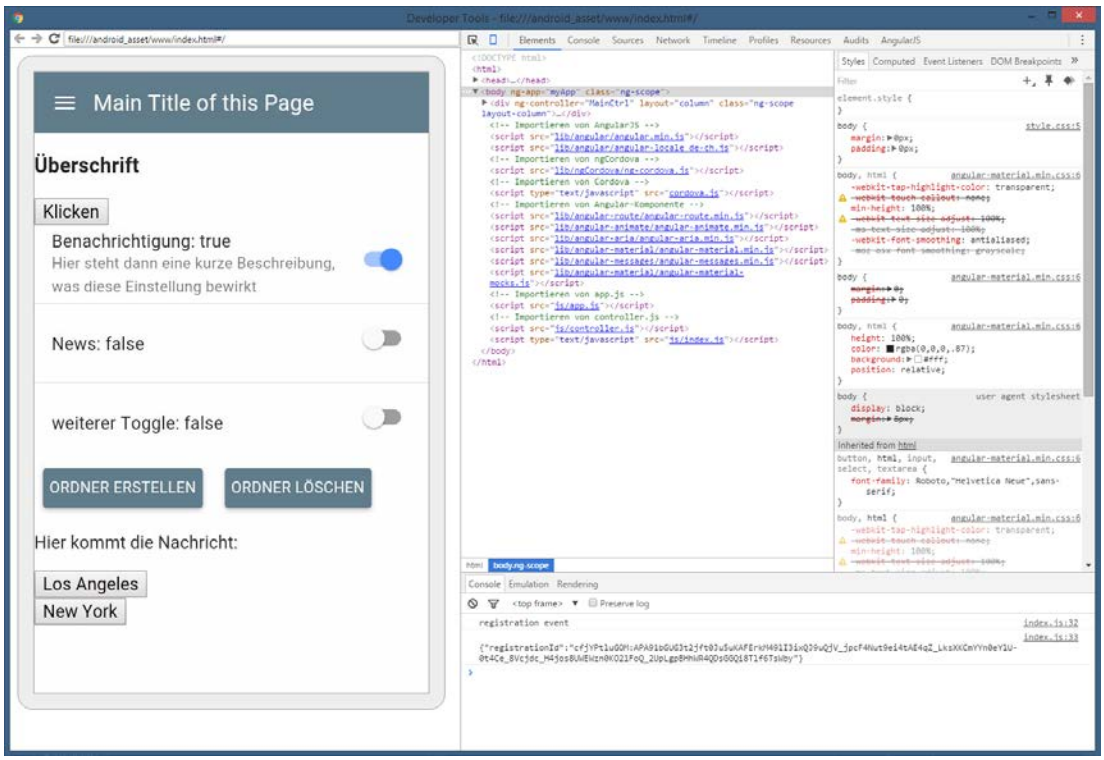


Abbildung 6 Screenshot Chrome Inspect

12.2 Berechtigungsprobleme bei OS X und cordova

Über lange Zeit hatte ich Probleme beim Builden der App unter OS X. Zuerst dachte ich, es liege an der fehlenden Apple Developer Lizenz, aber dann stellte sich heraus, dass es "lediglich" ein Berechtigungsproblem war. Zum lösen dieses Problems musste man den Benutzer Schreib- und Leserechte auf den gesamten Benutzerordner geben:

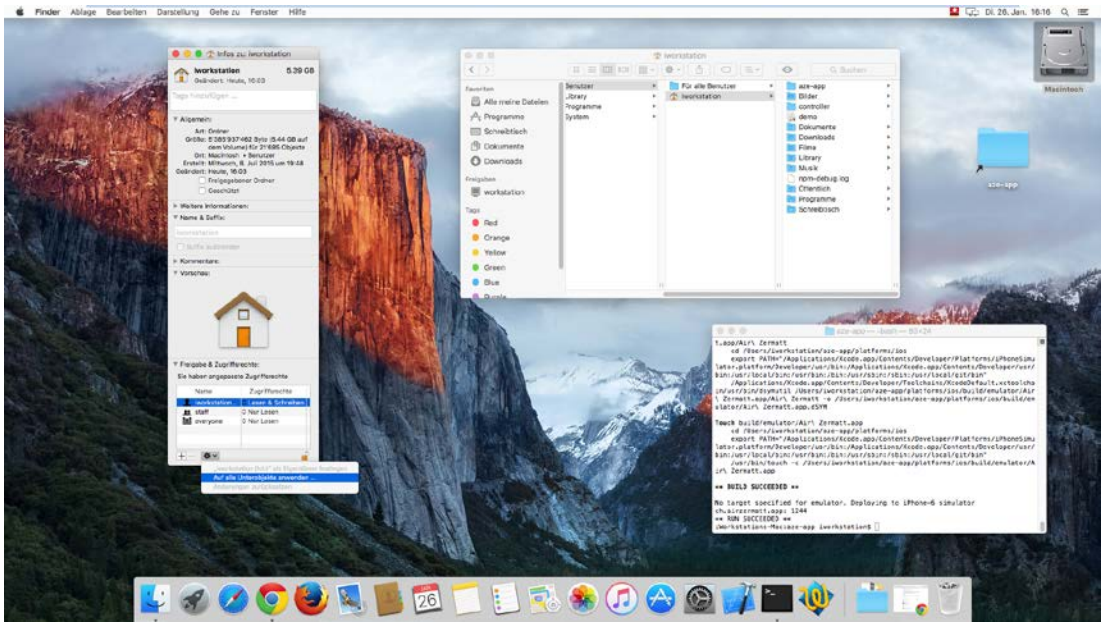


Abbildung 7 Berechtigungen vergeben

Diesen Lösungsansatz habe ich von folgendem Blog:

http://zacvineyard.com/blog/2015/11/ionic-invalid-device-state-error-with-xcode-7-solved

12.3 Nicht vertrauenswürdiger Entwickler bei iOS

Versucht man das erste Mal ein App auf einem iPhone zu testen, kann es sein, das dem Entwickler der App nicht vertraut wird. Falls die Fehlermeldung wie auf **Fehler! Verweisquelle konnte nicht gefunden werden.Fehler! Verweisquelle konnte nicht gefunden werden.** erscheint, so muss in den Einstellungen folgende Änderung durchgeführt werden. Unter Allgemein >> Geräteverwaltung erscheint ein neuer App-Entwickler. Dort kann dann ausgewählt werden, dass man diesem Entwickler vertrauen möchte. Danach kann man die App wie gewünscht öffnen.

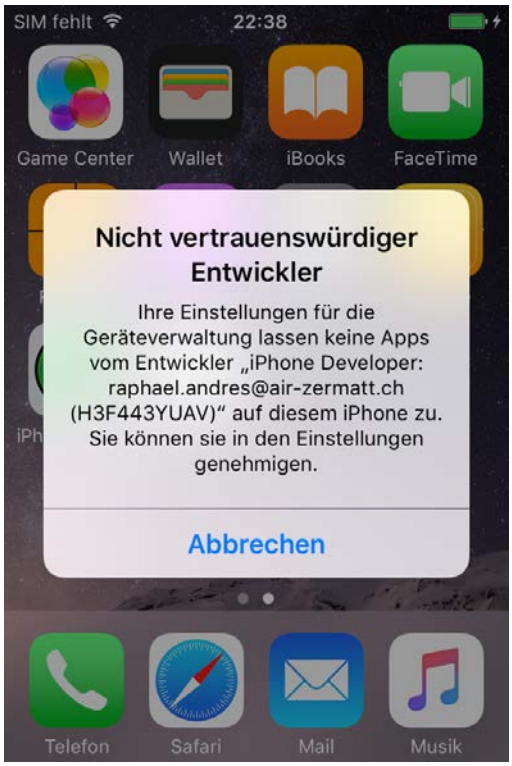


Abbildung 8 Nicht vertrauenswürdiger Entwickler

Abbildung 9 Entwickler vertrauen

12.4 Keine Push-Benachrichtigung für Windows Phone

Bei der genaueren Recherche hat sich ergeben, dass die Funktion der Push-Benachrichtigung für Windows Phone 8.0 Geräte nicht möglich ist. Das dazu benötigte Plugin phonegap-plugin-push unterstützt die Plattform WP8 nicht (Stand: 25.01.2016):

https://github.com/phonegap/phonegap-plugin-push/blob/master/docs/PLATFORM_SUPPORT.md

Wir müssen uns nun überlegen, ob wir auf die Plattform Windows Universal wechseln, da die Plattform WP8 mit dem **Update von Cordova zur Version 6** als Veraltet gilt. Zudem unterstützt die neue Version 6 nun offiziell Android 6 Marshmallow. (Stand: 28.01.2016)

III. REALISIERUNG

1 PROJEKT ERSTELLEN

Der schwierige Part beim Erstellen des Cordova-Projekts, ist die Bereitstellung von allen notwendigen Programmen und deren Konfiguration. Genauere Beschreibungen finden Sie im Teil der **Vorbereitung**. Um das Programm Cordova zu installieren oder zu aktualisieren, kann dieser Befehl in der Eingabeaufforderung ausgeführt werden:

```
1  npm install -g cordova
```

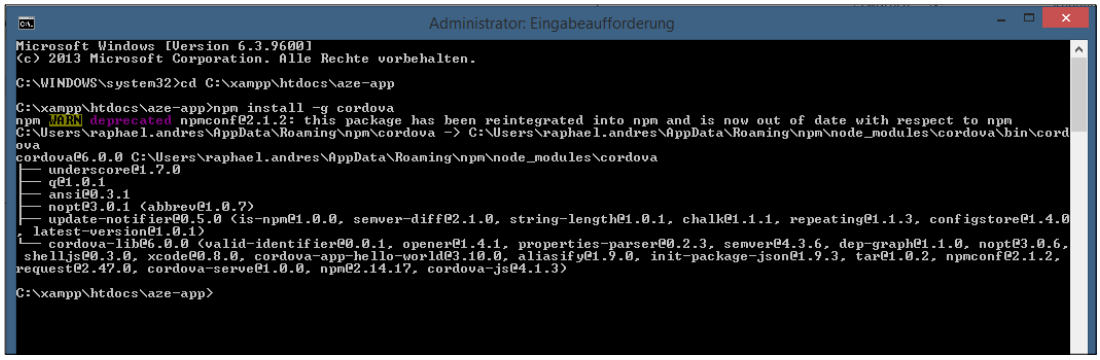


Abbildung 10 Eingabeauffordeurng

1.1 Cordova-Projekt erstellen

Um ein neues Cordova-Projekt anzulegen, öffnet man die Eingabeaufforderung und navigiert zum gewünschten Ordner. Hier kann folgender Befehl ausgeführt werden:

```
1  cordova create projekt ch.beispiel.app "App"
```

Mit diesem Befehl erstellt Cordova im aktuellen Ordner einen Unterordner namens *projekt*. Ich habe für dieses Projekt ein Ordner in *xampp/htdocs* gewählt. Damit ich die Website so bereits begutachten kann. Zusätzlich wird dieses Verzeichnis in unsere Creative Cloud gesichert. So kann ich jederzeit auf ein Backup zurückgreifen. In diesem Unterordner wird die Ordnerstruktur erstellt, in der alle für das Projekt relevanten Elemente gespeichert. Der Begriff *ch.beispiel.app* ist die **App-ID** und "*App*" ist der Name der App, welche dann auf dem Display des Users angezeigt wird. Ist dieser Vorgang abgeschlossen kann man via cmd-Befehl zum Unterordner wechseln:

```
2  cd project
```

1.2 Plattformen und Plugins installieren

Über folgenden Befehl können nun Plattformen hinzugefügt werden, für die man eine App realisieren will:

```
1  cordova platform add [plattform_name]
2  cordova platform ls
```

Mit dem Befehl in der zweiten Zeile können die installierten und noch verfügbaren Plattformen abgerufen werden.

Analog funktioniert das Installieren von Plugins:

```
1  cordova plugin add [plugin_name]
2  cordova plugin ls
```

Der genaue Plugin-Name kann beim Herausgeber des Plugins (Cordova, ngCordova, git) nachgeschlagen werden. Der Befehl in der zweiten Zeile zweigt wiederum die bereits installieren Plugins mit deren Version an.

1.3 Weitere Frameworks einbeziehen

Für dieses Projekt beziehe ich noch einige weite Frameworks ein:

1.3.1 AngularJS

AngularJS ist ein JavaScript-**Framework** und wurde von Google entwickelt. Es ist prädestiniert für sogenannte **SPA** (Single Page Applications). Entnehmen Sie bitte genauere Informationen über AngularJS im **Glossar** am Ende dieser Dokumentation.

Das Framework kann entweder auf der offiziellen Website von AngularJS (**https://angularjs.org/**) heruntergeladen werden, oder via CLI. Ich lade das Paket von der Website. Beim Download sollte man darauf achten, dass die stabile Version ausgewählt wird:

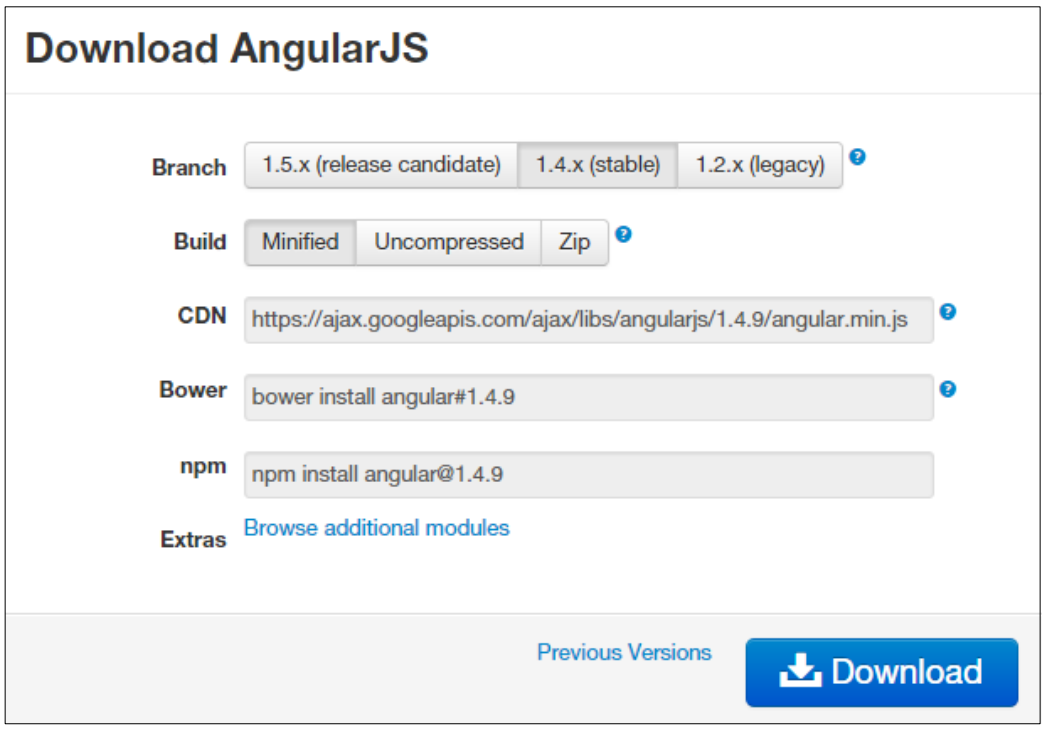


Abbildung 11 Download von AngularJS

Beim Download-Fenster wird zudem einen Link für CDN angezeigt. Diese Variante des Verknüpfens empfiehlt sich jedoch nicht, da der Benutzer vielleicht nicht immer mit seinen Smartphone Zugang zum Internet hat. Darunter findet man die Befehle, wenn man die Pakete über ein CLI installieren möchte. Die heruntergeladenen Pakete speichere ich im Verzeichnis *www/lib*. Dieses Verzeichnis sieht dann folgendermassen aus:

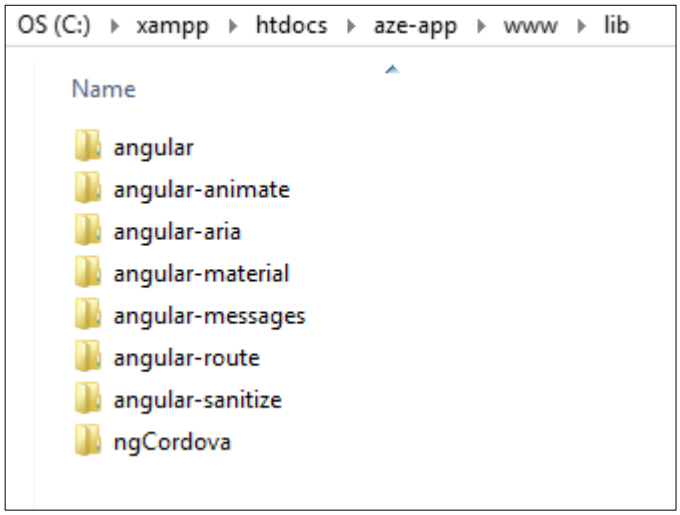


Abbildung 12 lib-Verzeichnis des Projektordners

Grundsätzlich installiere ich Frameworks und weitere Pakete, die ich vom Internet herunterlade unter dieses Verzeichnis. Deshalb sind auch die Pakete von ngCordova dort aufgelistet.

1.3.2 ngCordova

ngCordova ist ein Framework, das Cordova mit AngularJS verbindet und dadurch zu einem mächtigen Tool wird, für die Entwicklung von Mobilen Applikationen. Weitere Informationen über Cordova finden Sie im **Glossar**. Wie bei AngularJS lassen sich auch bei ngCordova die benötigten Pakete direkt von der Website oder via Terminal herunterladen.

1.3.3 Angular Material

Neben AngularJS und ngCordova wird für Android das Paket von Angular Material installiert. Damit kommt man sehr nahe an den nativen Look einer Android-App und doch kann vieles frei gestaltet werden.

1.3.4 Chocolate Chips UI

Für Windows & iOS wird das Framework Chocolate Chips UI verwendet. Auch hiermit kommt man sehr nahe an den Nativen Look einer iOS-App. Bei der Plattform Windows gibt es noch das düstere Design von Windows 8 bzw. Windows 8.1. Da ich selbst kein Windows Phone habe und selten eins in den Händen halte, weiss ich nicht, wie das Design neu bei Windows 10 aussieht, bzw. ob es da auch gewisse Anforderungen an das Design gibt.

1.4 Das index.html File

Wie in der Webumgebung üblich befindet sich im www-Verzeichnis eine index.html-Datei. Dies ist die wichtigste (und, wenn die App als Single Page Applikation realisiert wird, die einzige) Datei, die bewusst aufgerufen wird. Die anderen Inhalte wie CSS, JS und HTML-Inhalte werden innerhalb dieser Datei nachgeladen.

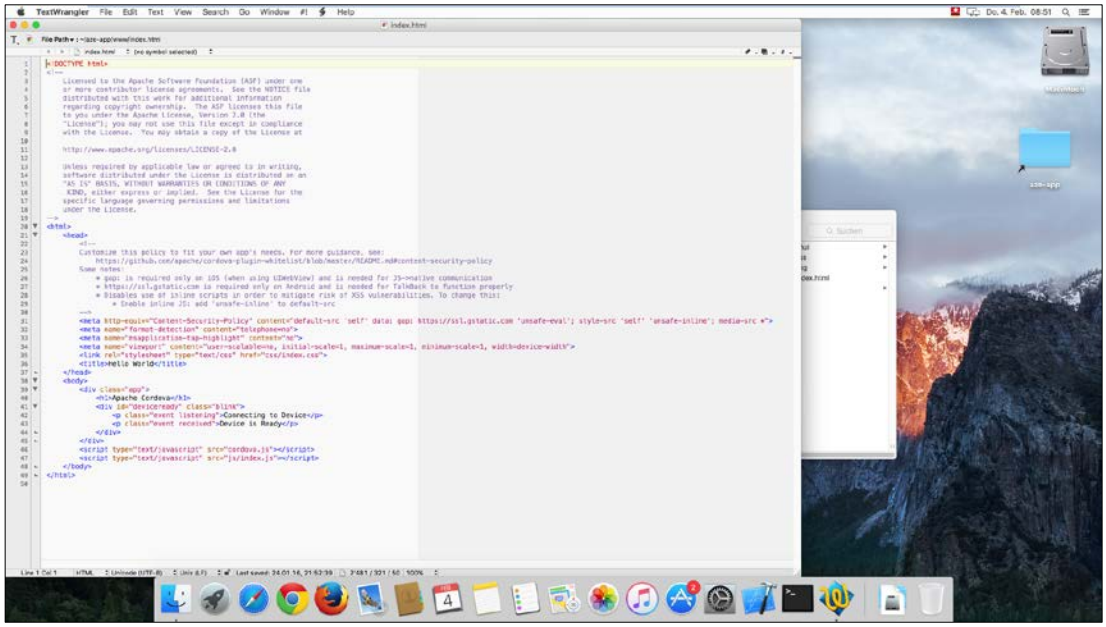


Abbildung 13 index.html-File

Öffnet man die index.html sieht man, dass es sich dabei um eine ganz normale Webpage handelt. Wenn man diese Webpage im Browser anschaut, fällt auf, dass in der Konsole eine Fehlermeldung erscheint, dass die Komponente cordova.js nicht gefunden werden konnte. Dies ist nicht weiter schlimm, da diese Datei tatsächlich noch nicht im Verzeichnis vorhanden ist. Diese wird erst beim Builden der App in das Verzeichnis kopiert.

1.4.1 Frameworks einbeziehen

Je nachdem, für welche Plattform gearbeitet wird, werden unterschiedliche Frameworks gebraucht. AngularJS wird für alle Plattformen verwendet. Bei Angular gibt es eine Hauptkomponente und weiter kleinere Komponente, mit denen weitere Funktionen hinzugefügt werden können. Damit das Rendern der Seite nicht unterbrochen wird, währendem die benötigten Dateien geladen werden, kann man die folgenden JavaScript-Files zuunterst der index.html-Datei eingefügt werden. Die Hauptkomponente wird folgendermassen importiert:

```
1  <!-- Importieren von AngularJS -->
2  <script src="lib/angular/angular.min.js"></script>
3  <script src="lib/angular/angular-locale_de-
   ch.js"></script>
```

Die zweite Datei *angular-locale_de-ch.js* ist eine Lokalisierungsdatei. Standardmässig hat Angular amerikanische Einheiten, Zahlen-, Zeit- und Datumsformate. Mit dieser Datei werden diese zu unseren schweizer Formate geändert. Zudem wird die Sprache von englisch auf Deutsch geändert.

Weitere relevante Komponente werden dann laufend hinzugefügt.

Weiter kann bereits ngCordova in die index.html importiert werden:

```
1  <!-- Importieren von ngCordova -->
2  <script src="lib/ngCordova/ng-cordova.js"></script>
```

Hierbei ist wichtig, dass *ng-cordova.js* nach *angular.js* aber vor *cordova.js* importiert wird. Der bisherige Block der Komponenten sieht also so aus:

```
1  <!-- Importieren von AngularJS -->
2  <script src="lib/angular/angular.min.js"></script>
3  <script src="lib/angular/angular-locale_de-
   ch.js"></script>
4
5  <!-- Importieren von ngCordova -->
6  <script src="lib/ngCordova/ng-cordova.js"></script>
7
8  <!-- Importieren von Cordova -->
9  <script type="text/javascript" src="cordova.js"></script>
```

Damit AngularJS problemlos funktioniert, braucht es einige Attribute, die wir noch hinzufügen müssen. Eine detaillierte Einführung in das Gebiet von Angular findet man z.B. auf **www.angularjs.org**.

1.4.2 Das Prinzip von AngularJS

Abbildung 14 Logo von AngularJS

Angular funktioniert nach dem **Model View ViewModel**-Muster. Mit dem Attribut *ng-app* kann man ein Angular Modul aktivieren, das dann innerhalb des ausgewählten Elements funktioniert. In meinem Projekt befindet sich dieses Attribut im *body*-Tag:

```
1  <body ng-app="myApp" >
```

Es können weiter eigene Controller erstellt werden. Innerhalb eines Controllers können dann Informationen gespeichert werden. Um einen bestimmten Controller zu aktivieren gibt es das Attribut *ng-controller*:

```
1  <div ng-controller="MainCtrl" >
```

Diese beiden Module müssen irgendwo hinterlegt sein. Dazu laden wir zwei weiter Files in die index.html:

```
<!-- Importieren von app.js -->
<script src="js/app.js"></script>

<!-- Importieren von controller.js -->
<script src="js/controller.js"></script>
```

In der ersten Datei app.js wird als erstes das Modul erstellt:

```
var app = angular.module('myApp', ['ngMaterial']);
```

Hier erscheint wieder der Name *myApp*, den wir zuvor als Wert des Attributes *ng-app* gebraucht haben. In den eckigen Klammern können weitere Module hinzugefügt werden, die für die Verarbeitung gebraucht werden. Man redet hier von der **Dependency Injection**.

In der zweiten Daten controller.js werden die Controller bereitgestellt:

```
app.controller('MainCtrl',['$scope', function($scope) {
$scope.text = 'Hello World!';
}]);
```

Auch hier wird zuerst der Name des Controllers angegeben. Dann folgt die Dependency Injection mit den ganzen Funktionen. $scope ist ein Objekt, in das Ausdrücke gespeichert werden können. Im obigen Beispiel wird der Ausdruck *Hello World!* in dieses Objekt gespeichert. Neben einfachen Ausdrücken können auch Arrays und Funktionen darin gespeichert werden.

Nun sind die Informationen auf Seiten Model gespeichert. Jetzt wollen wir den gespeicherten Ausdruck auf der Seite (View) anzeigen lassen. Dazu kann man innerhalb des Controllers die Ausgabe des Ausdruckes deklarieren. Dazu fügt man folgender Befehl ein:

```
{{text}}
```

Angular erkennt diesen Befehl mit den doppelt geschweiften Klammern und sucht im Controller *MainCtrl* nach einem Objekt *text* und gibt dieses zurück. Ein vollständiges Beispiel kann so aussehen:

```
<body ng-app="myApp">
<div ng-controller="MainCtrl">
<h1>{{text}}</h1>
</div>
</body>
```

Kommt man mit AngularJS zu ersten Mal in Kontakt kann dieser Arbeitsablauf recht kompliziert wirken. Zumindest traf das auf mich zu. Arbeitet man allerdings ein paar Mal mit Angular, so werden einem diese Schritte klarer und verständlicher.

Ein riesiger Vorteil von AngularJS gegenüber jQuery ist die Datenbindung. Diese funktioniert nämlich bei AngularJS bidirektional. Das heisst, dass die Werte in der View automatisch und direkt aktualisiert werden, wenn sie im Model geändert werden und umgekehrt werden die Daten im Model direkt angepasst, wenn sie in der View durch den Besucher geändert werden.

AngularJS besitzt weitere Attribute, wie z.B: ng-model, ng-repeat, ng-click, ng-if, etc. auf die ich zurückkommen werden, wenn wir sie konkret verwenden.

2 BACK-END & DATENBANK ERSTELLEN

Als nächstes wird das Back-End erstellt und die Datenbank bereitgestellt. Wie im Pflichtenheft erwähnt gab es bereits eine App für Apple-Geräte und dementsprechend auch ein Back-End für Mitarbeiter. Ich habe im Vorfeld einen Einblick in dieses Back-End erhalten, um zu sehen, wie es aufgebaut ist und wie der Workflow für die Mitarbeiter aussieht.

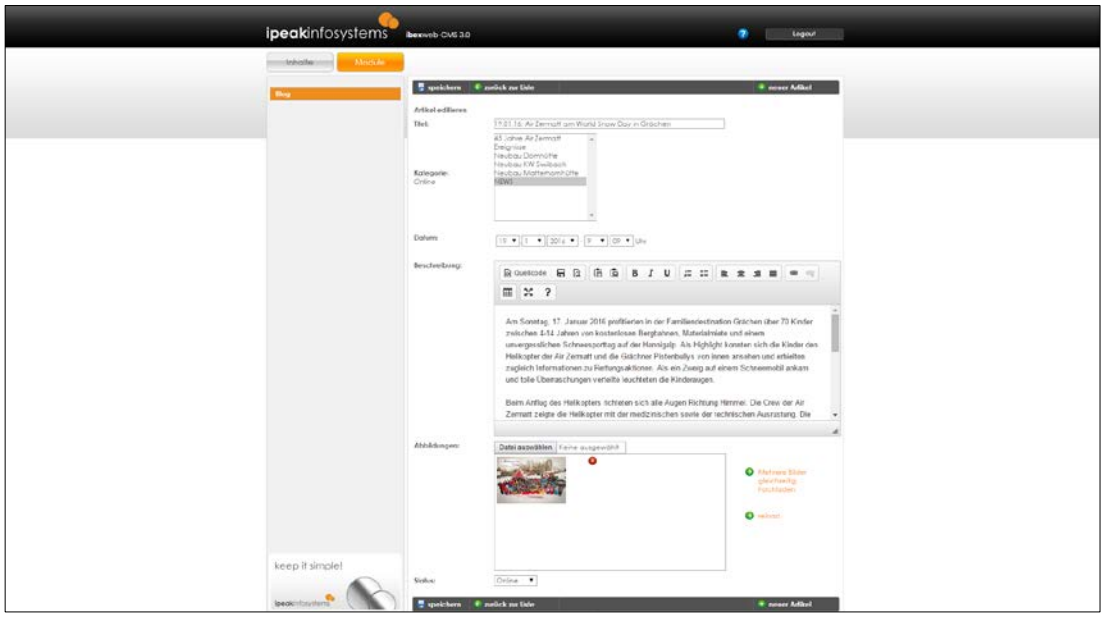


Abbildung 15 Früheres Back-End Newsticker

Weil mir für dieses Projekt nur eine begrenzte Zeit zur Verfügung steht, muss ich das Back-End fürs Erste etwas einfacher gestalten. Dieses kann zu einem späteren Zeitpunkt, nach der IPA, ausgebaut werden.

Das Back-End richte ich in einem Unterordner auf unserer Domain air-zermatt.ch ein. Auf dieser Ebene befinden sich dann die Eingabemasken für den Newsticker, sowie für die Angebote und den Push-Benachrichtigungen. In einem weiteren Unterordner soll der Login-Bereich sein, der bei erfolgreichem Anmelden auf die obere Ebene verlinkt und den Inhalt freigibt.

Wichtig: Für diese Dokumentation werden bei sämtlichen (Benutzer-)Namen und Passwörter Beispieldaten verwendet.

2.1 Login-Bereich erstellen

Was nicht explizit auf dem Projektplan aufgeführt, aber für die Sicherheit von fundamentaler Bedeutung ist, ist ein passwortgeschützter Login-Bereich. Da ist in einer früheren Arbeit für die Schule bereits ein solches Login erstellt habe, benütze ich dieses als Vorlage für dieses

Projekt. Als erstes erstelle ich eine PHP-Datei, die eine Verbindung zur MySQL-Server herstellt, um die Kommunikation zwischen Datenbank und Script zu ermöglichen:

```
1  <?php
2
3  $host = "localhost";
4  $user = "[benutzername]";
5  $password = "[passwort]";
6
7  mysql_connect($host, $user, $password) or die("Verbindung
   konnte nicht hergestellt werden!");
8  mysql_select_db("[datenbank]");
9  ?>
```

Grundsätzlich ist es aus Datenschutz-Gründen wichtig, dass der Benutzer nicht zu viele Berechtigungen erhält. Will man beispielsweise in einem PHP-Script lediglich einige Datensätze auslesen, sollte man dringend darauf achten, dass sich ein Benutzer mit dem MySQL-Server verbindet, der nur die Berechtigung zum auslesen (konkret der SELECT-Befehl) berechtigt ist! Am Schlimmsten wäre, wenn man solche Abfragen mit den root-Benutzer tätigt. Die Gefahr von **PHP Injection** oder ähnlichen Angriffen ist gross.

Diese Datei wird später von verschiedenen Scripts aufgerufen, um die Verbindung aufzubauen.

Im Unterordner befindet sich eine index.html-Datei, mit einem Formular zur Eingabe des Passworts. Dieses wird an ein zweites File im selben Ordner gesendet, welches das Passwort überprüft. Konkret wird das Passwort entgegen genommen und verschlüsselt. Dann wird die oben genannte Datei zum Verbindung zum MySQL-Server bereits das erste Mal angefordert, um die Verbindung herzustellen. In der Datenbank wird überprüft, ob das verschlüsselte Passwort hinterlegt ist und falls ja, wird eine PHP-Session gestartet. Ist das eingegebene Passwort nicht vorhanden, springt der Benutzer zurück zum Formular und erhält den Hinweis, dass sein Passwort falsch ist.

2.2 Eingabeformular Newsticker

Das erste Eingabeformular wird vorerst, relativ einfach gestaltet. Wichtig ist, dass die Eingabe und die Speicherung in die Datenbank reibungslos funktioniert. Auf den gestalterischen Aspekt komme ich später in diesem Projekt zurück.

Als erstes überprüfe ich wiederum, ob eine Session mit dem vergebenen Session-Name existiert. Ist dies nicht der Fall wird der Besucher zum Login-Formular weitergeleitet. Ist die Session vorhanden, wird der Wert überprüft und der restliche Inhalt des Files wird ausgegeben. Die Eingabemaske besteht aus einem normalen HTML-Formular.

2.2.1 Hochladen von Bildern

Da auch Bilder hochgeladen werden können, braucht das HTML-Tag *form* ein spezielles Attribut:

```
<form method="post" enctype="multipart/form-data">
```

Dieses Attribut sagt aus, dass die Formulardaten nicht codiert werden, wie das normalerweise der Fall ist. Das gesamte Formular sieht wie folgt aus:

```
<form method="post" enctype="multipart/form-data">
Titel: <input type="text" name="titel" /></br>
Kategorie: <select name="kategorie">
<option value="">Kategorie w&auml;hlen</option>
</select></br>
Datum: <input type="date" name="datum" /></br>
Beschreibung: <textarea name="beschrei-
bung"></textarea></br>
Bilder: <input type="file" name="bild" id="upload"></br>
Status: <select name="status">
<option value="">Status w&auml;hlen</option>
</select></br>
<input type="submit" value="Speichern" name="submit">

</form>
```

Bei den Dropdown-Feldern werden die Optionen später via PHP dynamisch geladen.

Nun müssen diese Formulardaten vom selben File entgegengenommen werden. Dies darum, weil im HTML-Tag *form* das Attribut *action* nicht definiert wird.

Bei Dateien ist die Verarbeitung etwas anders als bei normalen Textfeldern. Hier werden die Informationen in einer Variable *$_FILES* gespeichert. Ich habe mich vorgängig über diese Vorgang auf **w3schools.com** informiert. Dort wurde das Prinzip anhand eines Beispiels erklärt, welches ich als Vorlage gebraucht habe:

```
$target_dir = "bilder/";
$file_name = basename($_FILES["bild"]["name"]);
$target_file = $target_dir . $file_name;
$uploadOk = 1;
$imageFileType = pathinfo($target_file, PATHINFO_EXTEN-
SION);
$check = getimagesize($_FILES["bild"]["tmp_name"]);
if($check !== false) {
echo "Datei ist ein Bild.</br>";
```

```
9  $uploadOk = 1;
10 } else {
11 echo "Datei ist kein Bild!</br>";
12 $uploadOk = 0;
13 }
```

In der Variable *$target_dir* gibt man das Verzeichnis an, wo das hochgeladene File gespeichert werden soll. In meinem Fall ist das der Unterordner *bilder*. Wird nichts in die Anführungszeichen geschrieben, wird das File auf derselben Ebene gespeichert, wie das Script. In die Variable *$file_nam*e wird der Name des Files gespeichert, z.B. *bild.jpg*. Mit dem Befehl pathinfo kann von einer Datei gewisse Informationen ausgelesen werden. Hier ist es die Extension, also die Dateiendung. Damit kann später zum Beispiel überprüft werden, ob es sich bei der Datei wirklich um eine Bild-Datei handelt. Wie der Name des Befehls *getimagesize()* bereits verrät, kann damit die Abmessung des Bildes herausgelesen werden.

```
1  if (file_exists($target_file)) {
2  echo "Sorry, die Datei existiert bereits!</br>";
3  $uploadOk = 0;
4  }
```

In dieser if-Schlaufe wird überprüft, ob die Datei bereits vorhanden ist.

```
1  if ($_FILES["bild"]["size"] > 1000000) {
2      echo "Sorry, die Datei ist zu gross!</br>";
3      $uploadOk = 0;
4  }
```

Mit der nächsten if-Schlaufe wird überprüft, ob die Datei eine gegebene Dateigrösse, hier von einem Megabyte, nicht überschreitet.

Wie oben bereits erwähnt lässt sich auch die Dateiendung überprüfen. Dazu kann folgender Code verwendet werden:

```
1  if($imageFileType != "jpg" && $imageFileType != "png" &&
   $imageFileType != "jpeg" && $imageFileType != "gif" ) {
2  echo "Sorry, nur .jpg, .png, .jpeg und .gif sind er-
   laubt.</br>";
3  $uploadOk = 0;
4  }
```

Zum Schluss muss die Datei nur noch am richtigen Ort abgespeichert werden. Dazu wird mit der Variable *$uploadOK* nochmals untersucht, ob alle Vorgaben erfüllt sind und verschiebt folglich die hochgeladene Datei an den richtigen Platz:

```
1  if ($uploadOk == 0) {
```

```
2  echo "Sorry, die Datei konnte nicht hochgeladen wer-
   den!</br>";
3  } else {
4  if (move_uploaded_file($_FILES["bild"]["tmp_name"], $tar-
   get_file)) {
5  echo "Die Datei ". $file_name . " wurde hochgela-
   den.</br>";
6  } else {
7  echo "Sorry, es gab ein Problem beim Hochladen der Da-
   tei.</br>";
8  }
9  }
```

Funktioniert alles ordentlich, so sollte nach dem Upload diese Meldung erscheinen:

Abbildung 16 Screenshot Meldung nach Upload eines Bilder

Und auch das Bild wurde erfolgreich am richtigen Ort gespeichert:

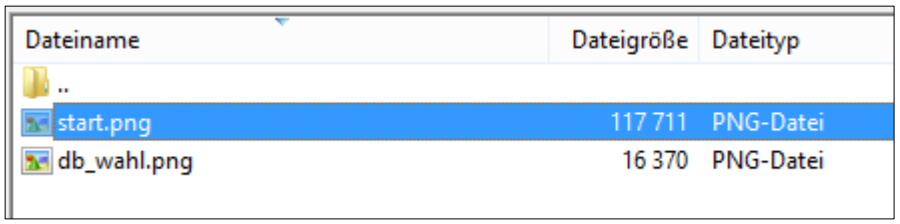


Abbildung 17 Hochgeladenes Bild befindet sich auf dem Server

Neben dem physischen Speichern der Datei werden einige Informationen über die Datei auch in eine Tabelle der Datenbank gespeichert. Primär soll der Name der Datei samt Dateiendung in die Datenbank gespeichert werden, damit das Bild später wieder als Ressource geladen werden kann. Dazu verwende ich die Variable *$file_name*, in welcher der Name noch gespeichert ist und führe damit ein Query aus:

```
require("connection.php");
$query = mysql_query("INSERT INTO tbl_bild(name)
VALUES('$file_name')");
if($query) {
$id_bild = mysql_insert_id();
} else {
echo "Bildinformationen wurden nicht in DB gespeichert";
}
```

Zugleich speichere ich die ID des gerade eingefügten Datensatzes in der Variable *$id_bild*. Diese Variable werden wir später noch brauchen.

2.2.2 Speichern der Daten in der Datenbank

Die anderen Daten, wie Titel, Beschreibung, Kategorie, etc. werden normal in eine Tabelle gespeichert. Die Werte aus den Textfeldern werden in eine Variable gespeichert:

```
$titel = htmlspecialchars($_POST["titel"]);
$kategorie = htmlspecialchars($_POST["kategorie"]);
$datum = htmlspecialchars($_POST["datum"]);
$beschreibung = htmlspecialchars($_POST["beschreibung"]);
```

Diese Variablen werden nachher mit einem Query gespeichert:

```
$query = mysql_query("INSERT INTO tbl_news(titel, datum,
beschreibung, status, FK_kategorie) VALUES('$titel',
'$datum', '$beschreibung', '$status', '$kategorie')");
if($query) {
$id_news = mysql_insert_id();
}
```

Ausserdem wird wieder die ID des eingefügten Datensatzes gespeichert. Unter Anbetracht, dass wir später einmal mehrere Bilder zu einem Newsbeitrag hinzufügen möchten, ergibt das eine Mehrfach zu Mehrfach-Relation zwischen den beiden Tabellen tbl_news und tbl_bild. Daher braucht man eine Tabelle, die diese Beziehungsform ermöglicht. In dieser Tabelle wird die ID der beiden eingefügt Datensätze gespeichert. Diese beiden IDs sind in je einer Variable gespeichert, weshalb man dieses Query ausführen kann:

```
$query = mysql_query("INSERT INTO tbl_news_bild(FK_news,
FK_bild) VALUES('$id_news', '$id_bild')");
if($query) {
echo "Bild und News verkn&uuml;pft!";
}
```

2.3 Eingabeformular Push-Benachrichtigung

Nachdem nun ein Eingabeformular für den Newsticker besteht, brauchen wir ein ähnliches Formular für die Push-Benachrichtigungen. Dieses erstelle ich in einem zweiten File. Wie beim ersten Eingabeformular, wird auch hier zuerst überprüft, ob die Session besteht. Ziel dieser Seite ist, dass man die notwendigen Informationen angeben kann, über die der Smartphone-Besitzer informiert wird. Dazu gehört die Variante des Rundflugs, das vorhergesehene Datum, die Uhr- oder Tageszeit und natürlich die Anzahl freien Plätze, die es für diesen Rundflug gibt. Die Varianten des Rundflugs werden in einer separaten Tabelle hinterlegt und können dort verwaltet werden. Zu einem späteren Zeitpunkt, könnte die Verwaltung ausgebaut werden, so dass der angemeldete Mitarbeiter die Varianten auf einer weiteren Seite des Back-Ends selbst verwalten kann. Ebenso kann der Helikopter ausgewählt werden, mit dem der Rundflug voraussichtlich durchgeführt wird.

2.3.1 Dynamisches Laden des Inhalts via AJAX

Gewisse Inhalte in den beiden Eingabefeldern, werden dynamisch geladen, ja nachdem welche Auswahl bei den Dropdown-Felder gewählt wurde. Der Inhalt dieser Auswahlfelder, wird via PHP geladen. Grundsätzlich sieht so ein Feld mit dynamischen Inhalt folgendermassen aus:

```
1  Kategorie: <select name="kategorie">
2  <option value="">Kategorie w&auml;hlen</option>
3
4  <?php
5  require("connection.php");
6  $query = mysql_query("SELECT * FROM tbl_kategorie");
7  while($row = mysql_fetch_array($query)) {
8  echo "<option value=\"" . $row["PK_kategorie"] . "\">" .
   $row["kategorie"] . "</option>";
9  }
10 ?>
11
12 </select>
```

Um nun den Inhalt eines anderen Feldes entsprechend der gewählten Option anzupassen, braucht es JavaScript. Genauer müssen wir einen xmlhttp-Request, kurz **XHR**, ausführen. Mit dem Attribute *onchange* kann eine JavaScript-Funktion ausgeführt werden. Dazu fügt man dieses Attribut und die aufzurufende Funktion ein:

```
1  <select name="basis" onchange="edit_base(this)">
```

Die Funktion hat den Namen *edit_base()*. Mit dem Parameter *this* geben wir der Funktion die Eigenschaften des aktuellen Elements mit. Diese Funktion ist ein einer JavaScript-Datei gespeichert Sie ermittelt den Wert der ausgewählten Option. Weiter wird eine asynchrone Abfrage der Datei *get_variante.php* eröffnet und durchgeführt. Die Antwort dieses Files wird dann in das zweite Dropdown-Feld eingefügt. Mithilfe des Attributs *disabled* kann das zweite Auswahlfeld vorläufig zum Bearbeiten deaktiviert werden, solange bis, im konkreten Fall, keine Basis ausgewählt wurde:

```
1  function edit_base(basis) {
2  var date_field = document.getElementById('variante');
3  if(basis.value == "") {
4  date_field.innerHTML = '<option value="">Variante
   w&auml;hlen</option>';
5  date_field.setAttribute('disabled', 'disabled');
6  } else {
7  var xmlhttp;
8  if (window.XMLHttpRequest) {
9  xmlhttp=new XMLHttpRequest();
10 } else {
11 xmlhttp=new ActiveXObject("Microsoft.XMLHTTP");
12 }
13 xmlhttp.onreadystatechange=function() {
14 if (xmlhttp.readyState==4 && xmlhttp.status==200) {
15 date_field.innerHTML=xmlhttp.responseText;
16 date_field.removeAttribute('disabled');
17 }
18 }
19 xmlhttp.open("GET","ajax/get_variante.php?b=" + basis.va-
   lue,true);
20 xmlhttp.send();
21 }
22 }
```

Wie werden nun die Optionen für das zweite Auswahlfeld generiert? Dies geschieht in der *get_variante.php*-Datei. Der übergebene Get-Parameter *b* beinhaltet den Fremdschlüssel der Basis und wird ausgelesen sowie in einen Abfrage integriert:

```
1  if(isset($_GET["b"])) {
2  $basis = htmlspecialchars($_GET["b"]);
3  require("../connection.php");
4  $query = mysql_query("SELECT * FROM tbl_variante WHERE
   FK_basis = '$basis'");
5  echo "<option value=\"\">Variante w&auml;hlen</option>";
6  while($row = mysql_fetch_array($query)) {
```

```
7  echo "<option value=\"" . $row["PK_variante"] . "\">" .
   $row["variante"] . " " . $row["dauer"] . "min.</option>";
8  }
9  }
```

Die zurückerhaltenen Datensätze werden formatiert und dann zum Client zurückgesendet. Dort fügt JavaScript die Optionen in das Dropdown-Feld ein und entfernt das Attribut *disabled* vom *select*-Element.

Um auch diese AJAX-Abfragen vor Missbrauch zu schützen, überprüfe ich ebenfalls vor jeder Verbindung mit der Datenbank, ob eine Session existiert. Ist dies nicht der Fall, antwortet das File mit dem Fehler-Code 404 - Datei nicht gefunden. Mit Javascript kann ich nun bewirken, dass die Seite neu geladen wird und dass der Benutzer sein Passwort erneut eingeben muss:

```
1  if(xmlhttp.readyState!==4 && xmlhttp.status==404) {
2  window.location.href = window.location.href;
3  }
```

2.3.2 Speichern der Daten in die Datenbank

Als nächstens müssen auch diese Formulardaten in die Datenbank gespeichert werden. Erneut werden die eingegeben Informationen in Variablen gespeichert und mithilfe eines MySQL-Querys der Datenbank übergegeben.

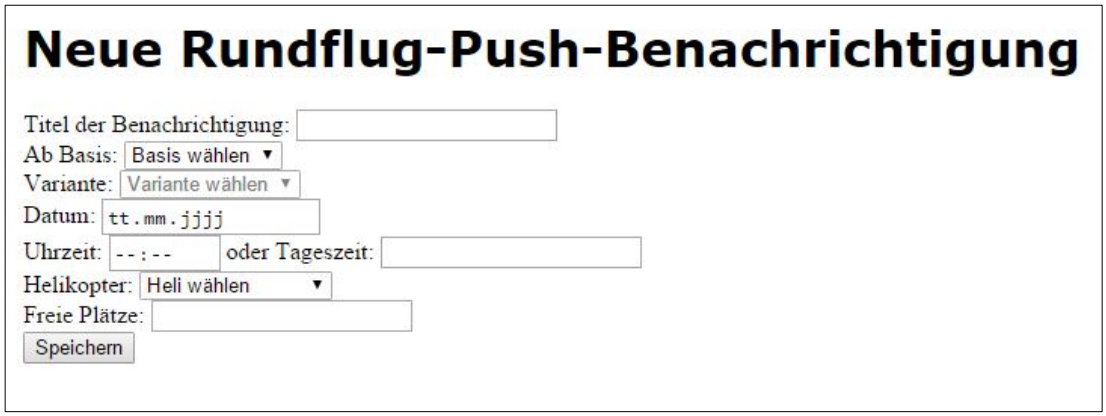


Abbildung 18 Eingabeformular für die Push-Benachrichtigung

Im Formular kann entweder die Uhrzeit oder die Tageszeit des geplanten Rundflugs abgegeben werden. Ist beides noch nicht bekannt, können beide Felder weggelassen werden. Um diese verschiedenen Szenarien zu handhaben werden diese Formulardaten folgendermassen verarbeitet:

```
1  if(isset($_POST["uhrzeit"]) && !empty($_POST["uhrzeit"]))
   {
```

```
2  $uhrzeit = "'" . htmlspecialchars($_POST["uhrzeit"]) .
   "'";
3  } else {
4  $uhrzeit = "NULL";
5  }
```

Zuerst wird überprüft, ob der Parameter *$_POST["uhrzeit"]* existiert und ob er leer ist. Ist ein Wert gegeben, wird dieser mit Anführungszeichen in die Variable gespeichert. Gibt es keinen Wert, so erhält die Variable den Wert *"NULL"*.

```
1  require("connection.php");
2  $query = mysql_query("INSERT INTO tbl_rundflug(titel, da-
   tum, uhrzeit, tageszeit, freie_platze, FK_variante,
   FK_helikopter)
3  VALUES('$titel', '$datum', $uhrzeit, $tageszeit,
   '$freie_platze', '$variante', '$helikopter')");
4  if($query) {
5  echo "Daten gespeichert!";
6  } else {
7  echo "Fehler: " . mysql_error();
8  }
```

Auffallend ist, dass *$uhrzeit* und *$tageszeit* nicht in einfachen Anführungszeichen stehen. Dies ist deshalb der Fall, weil wir den Wert *NULL* übergeben, wenn im Formular keine Zeit angegeben wurde und der darf nicht in Anführungszeichen stehen.

2.4 Senden der relevanten Daten an die Smartphones

Nun sollen diese Daten aber nicht nur in die Datenbank gespeichert werden, sondern auch an die Smartphones der registrierten Benutzer gesendet werden. Dazu müssen sogenannte **Payloads** an die Push Benachrichtigungs-Dienstleister gesendet werden. Bei Android ist dieser Dienstleister Google, bei iOS ist es Apple und bei Windows stellt dies Microsoft zur Verfügung.

3 SEITE HOME ERSTELLEN

Kommen wir nun von Back-End zum "Front-End", also zu diesem Teil der Applikation, den jeder einzelne Benutzer sieht. Speziell in diesem Projekt ist, dass dieser Teil der Applikation nicht auf demselben Server aufgerufen wird, wie das Back-End. Das Front-End befindet sich auf dem Smartphone des Benutzers. Dort werden alle Ressourcen hinterlegt. Fixe Dateien, wie HTML-, CSS- und JS-Files oder auch Icons kann man im Cordova-Ordner unter dem www-Verzeichnis speichern. Dateien, die auf das mobile Gerät heruntergeladen werden, wie zum Beispiel Bilder und auch die sqlite-Datenbanken können in einen für die App vorgesehenen Ordner gespeichert werden. Dieser Ordner befindet sich je nach Plattform an einem anderen Ort. Bei allen Plattformen ist dieses Verzeichnis aber nicht ohne weiteres für den Nutzer zugänglich. Mit einem **root**-Benutzer mit Administrator-Berechtigung und einem passenden **Datei Manager** kann man jedoch auf diese Verzeichnisse zugreifen:

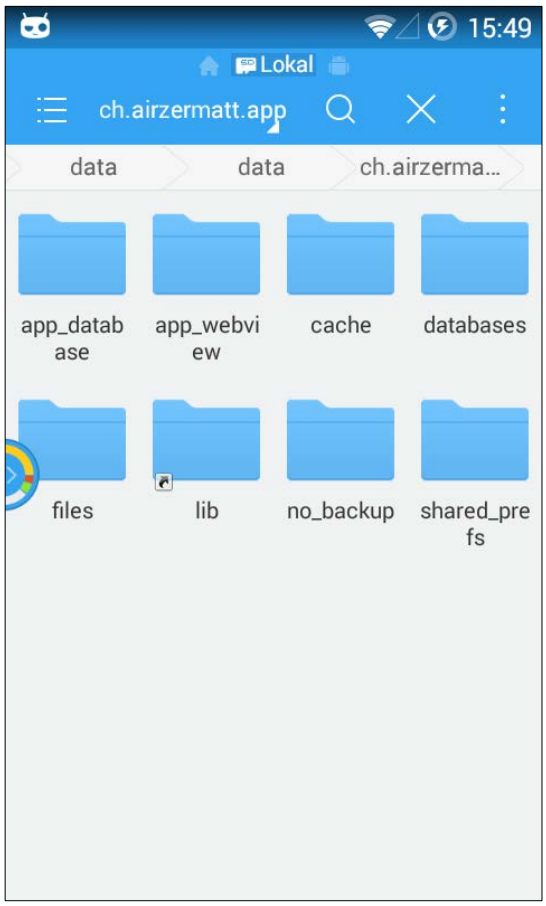


Abbildung 19 Root-Zugriff auf das Datei System

Wie im Pflichtenheft beschrieben, beginne ich mit der Entwicklung für die Plattform Android. Es gibt noch einige Vorbereitungen zu erledigen.

3.1 Vorbereitungen für die Plattform Android

In der Vorbereitungs-Phase dieses Projektes hat sich gezeigt, dass es von Vorteil sein kann, wenn zuerst nochmals auf der Website von Cordova über die aktuellen Versionen informiert und gegebenenfalls Cordova und die Plattformen sowie Plugins aktualisiert. Informationen über die installierten Versionen kann man auch mit diesem CLI-Befehl abrufen:

```
1  cordova info
```

Daraufhin erscheinen Informationen zu NodeJS, Cordova, die config.xml-Datei wird dargestellt, die installieren Plugins werden aufgelistet sowie die verfügbaren Ziel-Plattformen.

Weiter kann man überprüfen, ob es Updates beim Android SDK Manager gibt. Öffnet man den SDK Manager erscheint dieses Fenster und es wird automatisch nach Updates gesucht:

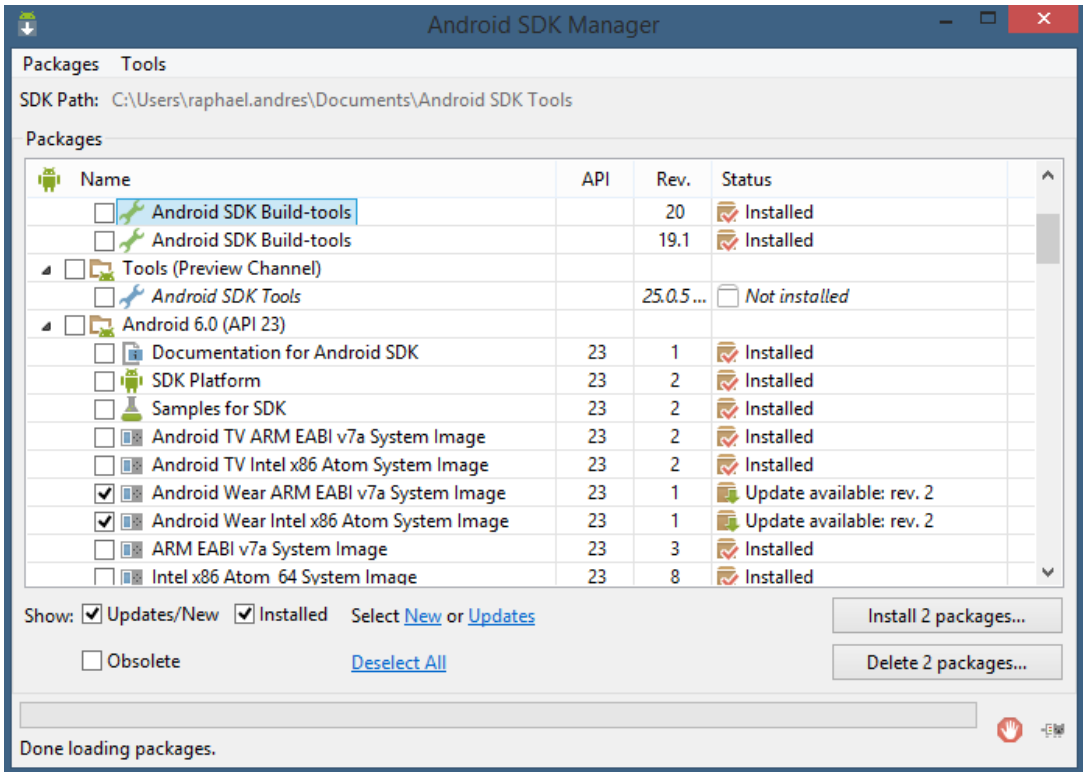


Abbildung 20 Android SDK Manager

3.1.1 Farbpalette für Angular Material

Damit die App schlussendlich im Air-Zermatt-typischen Design erscheint, kann bei Angular Material eine eigene Farbpalette definiert werden. Wie auf der Website von **Angular Material** ersichtlich, geht man von einer Grundfarbe aus und fügt dann weitere Farben hinzu, die heller als die Grundfarben sind bzw. dunkler. Im unterstehenden Beispiel ist die Grundfarbe dem Wert 500 hinterlegt. Je tiefer der Wert umso heller die Farbe und umgekehrt. Diese

Farbreihe kann man zum Beispiel mithilfe der Website http://www.color-hex.com/ anfertigen. Diese Palette wird in der Datei app.js definiert:

```
app.config(function($mdThemingProvider) {

$mdThemingProvider.definePalette('aze_primary', {
'50': 'e999a2',
'100': 'de6674',
'200': 'd84c5d',
'300': 'd33246',
'400': 'cd192f',
'500': 'c80018',
'600': 'e53935',
'700': 'b40015',
'800': 'a00013',
'900': '78000e',
'A100': 'de6674',
'A200': 'd33246',
'A400': 'c80018',
'A700': 'a00013',

'contrastDefaultColor': 'light',
'contrastDarkColor': ['50', '100', '200', '300', '400',
'A100'],
'contrastLightColors': undefined
});

$mdThemingProvider.theme('default')
.primaryPalette('aze_primary')
.accentPalette('blue');
});
```

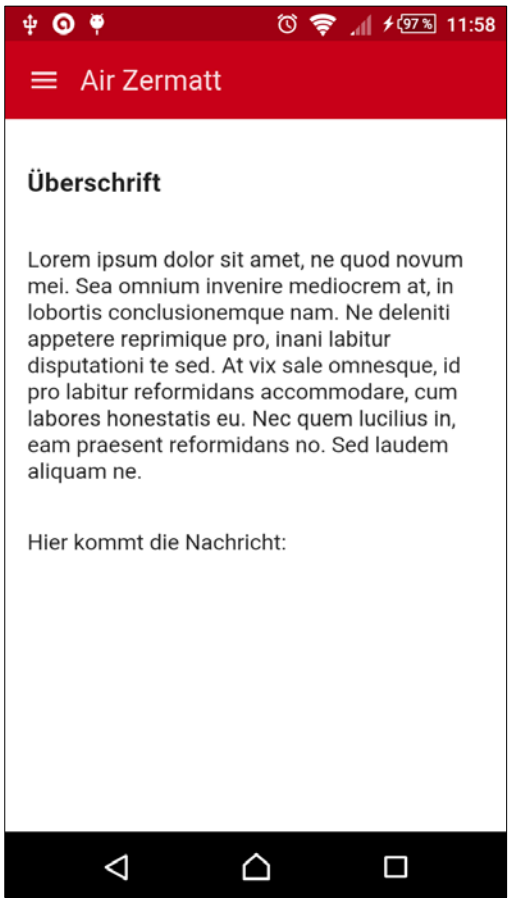


Abbildung 21 Screenshot mit neuer Farbpalette

Analog zu dieser roten Farbe, welche als primäre Farbe gewählt wurde, kommt eine zweite Palette hinzu, die das Air Zermatt-Blau enthält. Die blaue Farbe wird als Akzentfarbe verwendet. Ist die Farbpalette definiert, so kann man sich dem Inhalt der Seiten widmen.

3.2 Templates und Angular Route

Ein mächtiges Tool, das mit Angular kommt, ist Angular Route. Dieses Tool ermöglicht das Laden von Templates je nach Adresspfad. Das heisst, Angular schaut sich die URL an und entscheidet entsprechend, welches Template nun geladen werden muss. Um Angular Route zu verwenden, muss die dazugehörige JavaScript-Datei in die index.html importiert werden. Dazu fügt man diese Zeilen hinzu:

```
1  <!-- Importieren von Angular-Komponente -->
2  <script src="lib/angular-route/angular-
   route.min.js"></script>
```

Die index.html dient als Grundgerüst für die ganzen Seiten. In meinem Fall erstelle ich auf dieser index-Seite das Menü, das für alle Seiten dasselbe ist. Innerhalb des Body-Tags, das das *ng-app*-Attribut enthält, füge ich einen Wrapper ein. Dieser Wrapper erhält den Controller *MainCtrl*. In diesem Wrapper füge ich das Menü ein. Da dieses Menü viele wiederkehrende Elemente aufweist, erzeuge ich dieses Menü mit einem Attribut von Angular, das sich *ng-*

repeat nennt. Die unterschiedlichen Elemente, wie Name und Icon speichere ich in einem $scope-Element eines neuen Controllers, der nur für das Menü gilt:

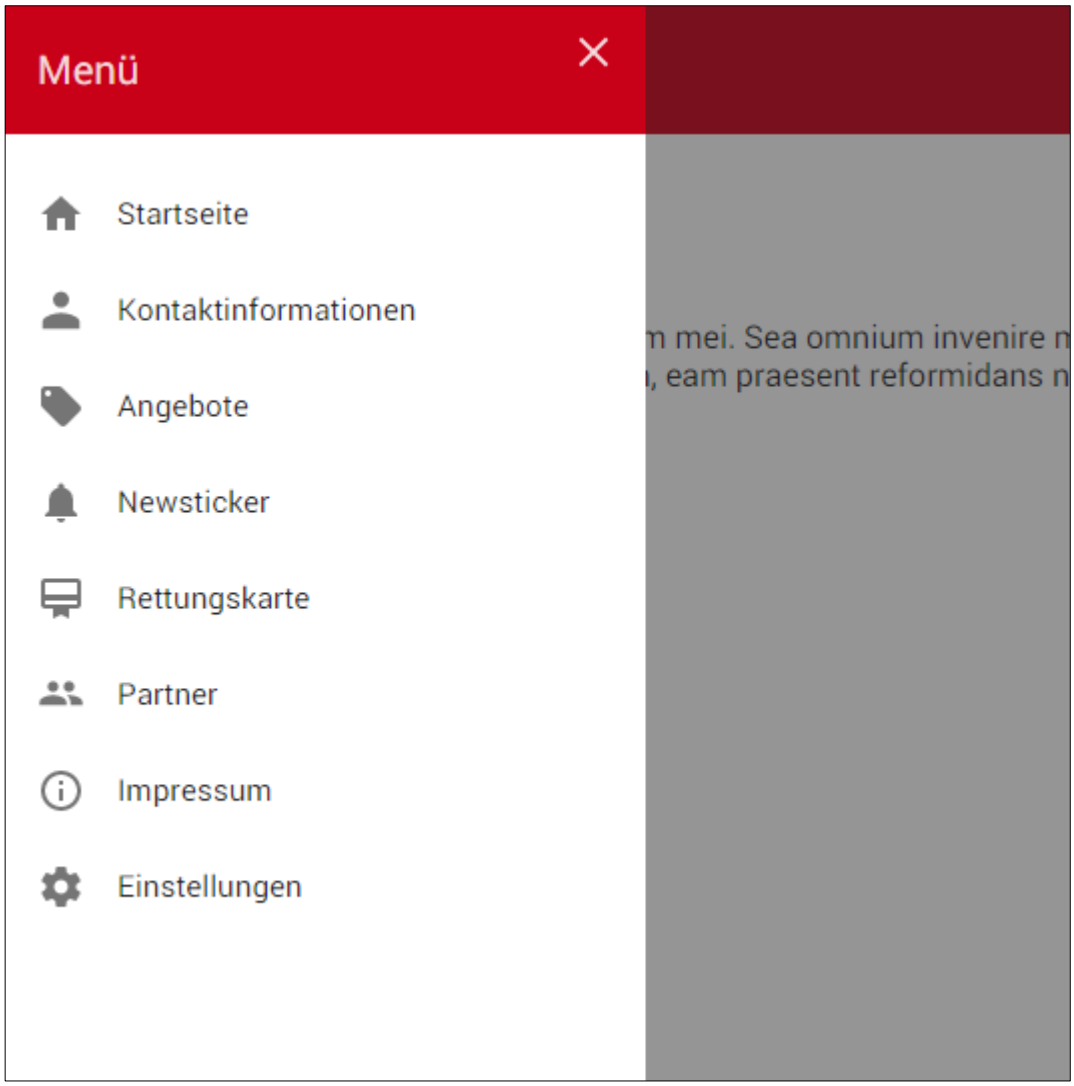
```
app.controller('MenuCtrl', ['$scope', function($scope) {
$scope.menu = [
{name: 'Startseite', icon: 'home'},
{name: 'Kontaktinformationen', icon: 'person'},
{name: 'Angebote', icon: 'local_offer'},
{name: 'Newsticker', icon: 'notifications'},
{name: 'Rettungskarte', icon: 'card_membership'},
{name: 'Partner', icon: 'people'},
{name: 'Impressum', icon: 'info_outline'},
{name: 'Einstellungen', icon: 'settings'}
];
}]);
```

Dieses Array kann nun mit *ng-repeat* in der View aufgelistet werden. Die Funktion *ng-repeat* kann man vergleichen mit der PHP-Funktion *foreach*. Dort gibt man dem einzelnen Eintrag einen neuen Namen und kann deren Werte beliebig formatieren und ausgeben und zwar solange, wie es Einträge im Array gibt. Bei AngularJS funktioniert dies nach dem gleichen Prinzip:

```
<md-menu-sidenav ng-controller="MenuCtrl">
<md-menu-sidenav-content>
<md-menu-item ng-repeat="link in menu">
<md-button href="#/{{link.name | lowercase}}" ng-
click="toggleLeft()" aria-label="{{link.name}}">
<md-icon md-menu-align-target md-svg-
icon="img/icons/{{link.icon}}.svg"></md-icon>
<span>{{link.name}}</span></md-button>
</md-menu-item>
</md-menu-sidenav-content>
</md-menu-sidenav>
```

Das Element *md-menu-sidenav* erhält den Controller *MenuCtrl*. Jeder Link wird mit dem Element *md-menu-item* ummantelt. Dieses Element erhält also das Attribut *ng-repeat*. Damit sagen wir Angular, dass es dieses Element so oft wiederholen soll, wie es Einträge im Array gibt. Für jeden Eintrag wird das Array *menu* in *link* umbenannt. Jetzt können der Name und der Name des Icons herausgelesen werden. Dies geschieht mit der Variable *{{link.name}}* und *{{link.icon}}*. Beim Element *md-button* kommt zusätzlich ein Filter zum Zug. Es gibt eine ganze Liste dieser Filter, die vor allem mit der Darstellung der übermittelten Informationen zu tun haben. Der Filter *lowercase* bewirkt, dass der übergebene Ausdruck in Kleinbuchstaben geschrieben wird. Eine Liste aller Filter gibt es auf der **Website von AngularJS**. Der Pfad

zum Icon wird durch die Variable *{{link.icon}}* vervollständigt. Diese Icons sind dementsprechend im Unterordner *img/icons/* gespeichert. Schaut man sich das Ergebnis an, so sollte das Menu problemlos angezeigt werden:



Der Link eines Menüpunktes sieht zum Beispiel so aus:

```
1  <a href="#/newsticker">Newsticker</a>
```

Wird auf diesen Link geklickt, wird die URL **http://localhost/aze-app/www/#/newsticker** geöffnet. Die Raute gibt Angular an, dass ich gerne mit dem *routeProvider* arbeiten möchte. Angular liest nun was hinter der Raute steht und ändert dem entsprechend die Template-URL. Folgender Code fügt man in die Datei *app.js* hinzu:

```
1  app.config(function($routeProvider) {
2  $routeProvider.when('/angebote', {
3  templateUrl : 'templates/angebote.html',
```

```
4  }).when('/einstellungen', {
5  templateUrl : 'templates/einstellungen.html',
6  }).when('/impressum', {
7  templateUrl : 'templates/impressum.html',
8  }).when('/kontaktinformationen', {
9  templateUrl : 'templates/kontaktinformationen.html',
10 }).when('/newsticker', {
11 templateUrl : 'templates/newsticker.html',
12 }).when('/partner', {
13 templateUrl : 'templates/partner.html',
14 }).when('/rettungskarte', {
15 templateUrl : 'templates/rettungskarte.html',
16 }).when('/', {
17 templateUrl : 'templates/homepage.html',
18 }).otherwise({
19 redirectTo: '/'
20 })
21 });
```

Diese Angular-Funktion ist vergleichbar mit der *switch* -Funktion bei PHP. Der übergebene Parameter wird mit den angegebenen Zuständen verglichen. Stimmt einer davon überein, wird templateUrl angewendet, ansonsten wird die Standardfunktion ausgeführt, in diesem Fall die Weiterleitung zur Startseite. Weiter muss das Modul *ngRoute* in die Dependency Injection hinzugefügt werden:

```
1  angular.module('myApp', ['ngMaterial', 'ngRoute']);
```

Damit angular auch weiss an welchen Ort das Template hereingeladen werden soll, verwendet man das Attribut *ng-view*. Dieses Attribut braucht keinen Wert zu haben:

```
1  <div ng-view="">
2  </div>
```

Das Template wird innerhalb des *div*-Containers geladen.

4 SEITE ANMELDEFORMULAR ERSTELLEN

Mithilfe von Angular Material lassen sich schöne und funktionale Eingabefelder, im Android-typischen Material Design erstellen. In der Vorbereitungs-Phase habe ich bereits ein kleines Formular damit kreiert:

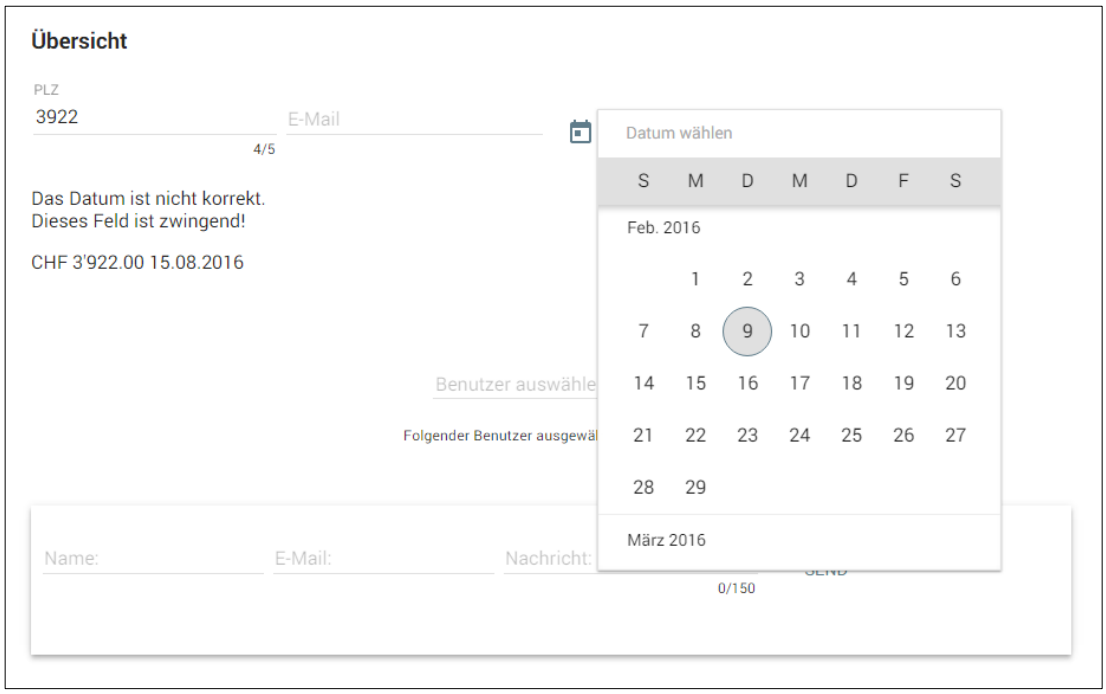


Abbildung 22 Test-Anmeldeformular aus der Vorbereitungs-Phase

Nun gilt es auf der Template-Seite Kontaktinformationen ein neues Formular zu erstellen und diese eingegeben Daten zu speichern.

4.1 Eingabefelder darstellen

Die Aufstellung eines Formulars funktioniert im Zusammenhang mit Angular Material etwas anders, als beim normalen Gebrauch. Was identisch bleibt ist das HTML-Tag *form*:

```
1  <form name="kontaktinformationen">
2  </form>
```

Es werden keine Attribute *method* und *action* gebraucht, weil die Daten nicht wirklich versendet werden, sondern von JavaScript aufgenommen werden und so weiterverarbeitet werden. Dafür erhält das Element einen eindeutigen Name, was wichtig für die später Validierung der Daten ist. Grundsätzlich wird jedes Feld in einem Container eingefügt:

```
1  <md-input-container>
2  <label>Vorname</label>
```

```
3  <input type="text" ng-modul="contact.vorname" required>
4  </md-input-container>
```

Mit dem Element *label* wird dem Feld den anzuzeigenden Namen mitgeteilt. Im *input*-Element stossen wir auf ein neues Attribut von AngularJS: *ng-modul*. Dieses Attribut bewirkt dasselbe, wie die Variablen in den doppelt geschweiften Klammern. Der hinterlegte Ausdruck wird ausgegeben. Einziger Unterschied ist, dass *{{ausdruck}}* im Inhalt eines Elements eingefügt werden kann, zusammen mit anderen Inhalten. Wird das Attribut auf ein Element gesetzt, wird ausschliesslich der hinterlegte Ausdruck angezeigt.

Die anderen Felder des Formulars sind auf dieselbe Weise aufgebaut, mit Ausnahme eines Drop-Down-Feldes:

```
1  <md-input-container>
2  <label>Anrede</label>
3  <md-select ng-model="contact.anrede" placeholder="Anrede
   w&auml;hlen">
4  <md-option value="herr">Herr</md-option>
5  <md-option value="frau">Frau</md-option>
6  </md-select>
7  </md-input-container>
```

Es gibt wiederum ein *label*-Tag, dass den Namen definiert. Neu ist das Element <*md-select*>. Daraufhin folgen die Auswahlmöglichkeiten. Der Aufbau des Drop-Down-Feldes ist derselbe wie beim HTML-Element *select*. Das Element erhält lediglich noch das Angular Material-typische Präfix *md-*. Nun braucht es noch einen Button, mit dem der Benutzer das Speichern der Daten bestätigen kann:

```
1  <md-button type="submit" class="md-primary" ng-
   click="save()" aria-label="Speichern">Speichern</md-but-
   ton>
```

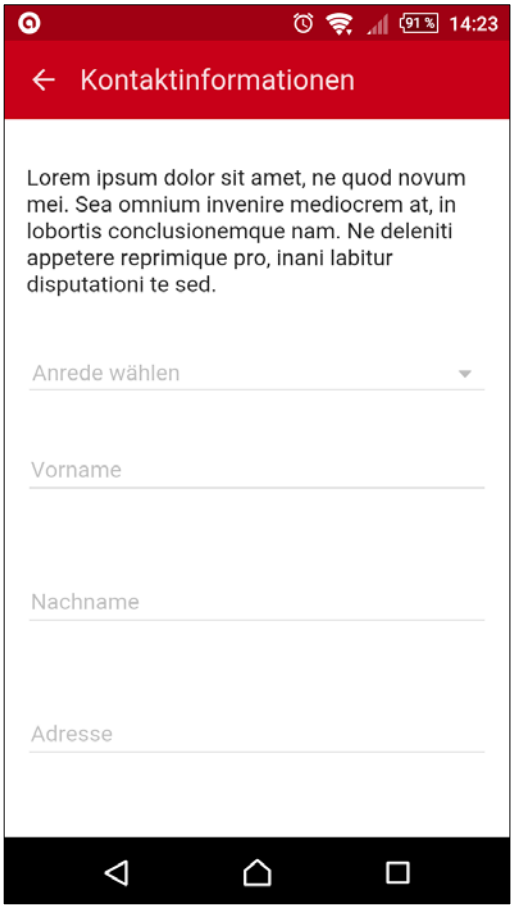


Abbildung 23 Screenshot Kontaktinformationen

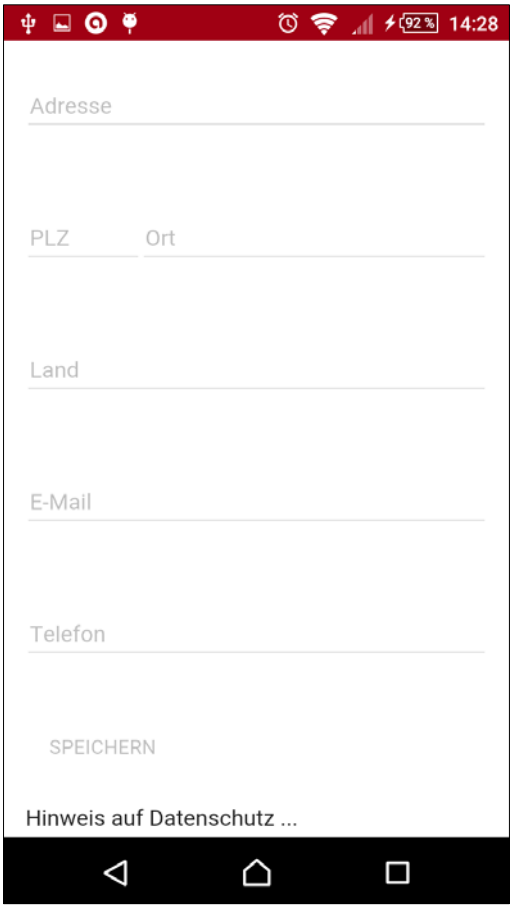


Abbildung 24 Screenshot Kontaktinformationen

4.2 Validierung der eingegebenen Daten

Bevor die Daten in die Datenbank gespeichert oder zu unserem Server gesendet werden, müssen diese kontrolliert werden. Mit Angular Material kann bereits bei der Eingabe der Daten überprüft werden, ob die Eingabe dem gewünschten Muster entspricht, oder ob der Besucher etwas anderes eintippt. Auch hier hilft Angular gewaltig und zwar mit der Direktiven *ngMessages*. Dieses Tool ermöglicht es, unterhalb des Eingabefeldes eine Nachricht erscheinen zu lassen, wenn die Eingabe nicht korrekt ist. Diese Direktive muss ebenfalls als Dependency Injection geladen werden:

```
1  var app = angular.module('myApp', ['ngMaterial',
   'ngRoute', 'ngMessages']);
```

Ist dies erledigt, kann innerhalb eines *input-containers* folgende Elemente hinzugefügt werden:

```
1  <md-input-container>
2  <label>E-Mail</label>
```

```
3  <input type="email" name="email" ng-model="contact.email"
   required>
4  <div ng-messages="kontaktinformationen.email.$error"
   role="alert">
5  <div ng-message="required">Die E-Mailadresse ist erfor-
   derlich.</div>
6  <div ng-message="email">Die Eingabe ist keine E-
   Mailadresse</div>
7  </div>
8  </md-input-container>
```

Unter dem *input*-Element ist neu ein *div*-Container mit dem Attribut *ng-messages*. Als Wert wird *kontaktinformationen.email.$error* angegeben, wobei *kontaktinformationen* der Name des Formulars ist und *email* der Name des Input-Feldes. In die Variable *$error* schriebt Angular selbstständig die fehlerhaften Angaben. In den beiden *div*-Container innerhalb des ersten Divs sind die Fehlermeldungen untergebracht. Zudem besitzen beide Container ein Attribut *ng-message*, in dem der fehlerhafte Zustand hinterlegt ist. Wurde zum Beispiel nicht eine korrekte E-Mailadresse in das Feld geschrieben, erscheint die Fehlermeldung:

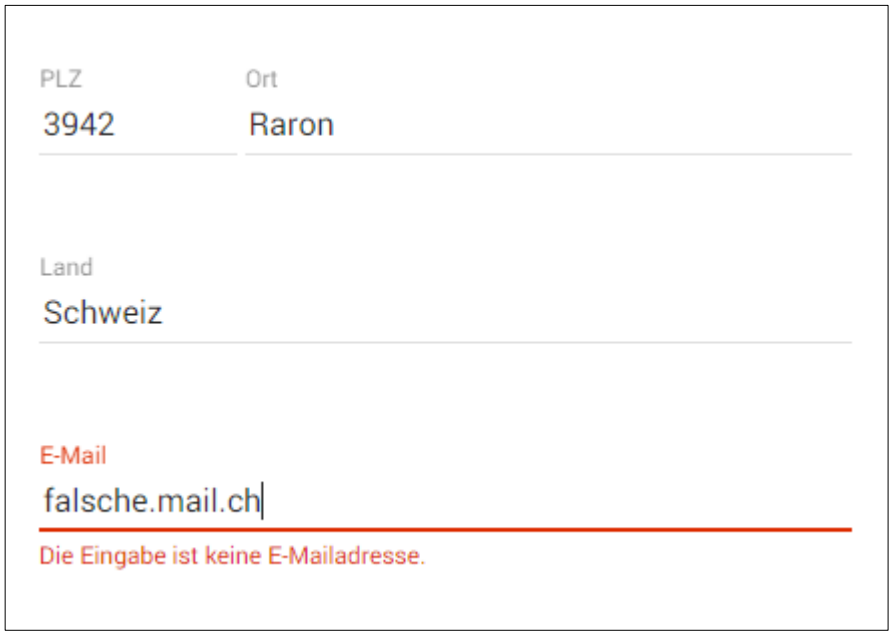


Abbildung 25 Screenshot Fehlermeldung bei inkorrekter Eingabe

4.3 Speichern der Daten in die Datenbank

Sind die Daten validiert, so folg die Speicherung der Daten in die Datenbank. Mit dem Plugin **cordova-plugin-dbcopy** für Cordova hat man die Möglichkeit, eine sqlite-Datenbank in Form eines Files in den Projektordner zu kopieren, welche dann in die Applikation eingebunden wird. Beim ersten Start der App wird die Datenbank in das dafür vorgesehene Verzeichnis auf

dem lokalen Gerät gespeichert. Von dort aus kann mit der Datenbank agiert werden, wie auf einem normalen Server. Doch bevor wir dies tun können, müssen wir zuerst einmal die Datenbank erstellen und mit Werten füttern.

4.3.1 Vorbereitung der Datenbank

Der Umgang mit **SQLite** ist für Beginner nicht gerade einfach. Auf der **Website des Anbieters** gibt es eine ausführliche Dokumentation mit vielen praktischen Beispielen. SQLite eignet sich sehr gut für eingebettete Datenbanksysteme. Es gibt ein Frontend, welches über die Konsole, bzw. Eingabeaufforderung verwendet werden kann. Zudem hat man mit dem Programm **sqlitebrowser** ein einfaches und vor allem grafisches Frontend. Aus Zeitgründen entscheide ich mich, bei der Vorbereitung der Datenbank, für diese grafische Lösung.

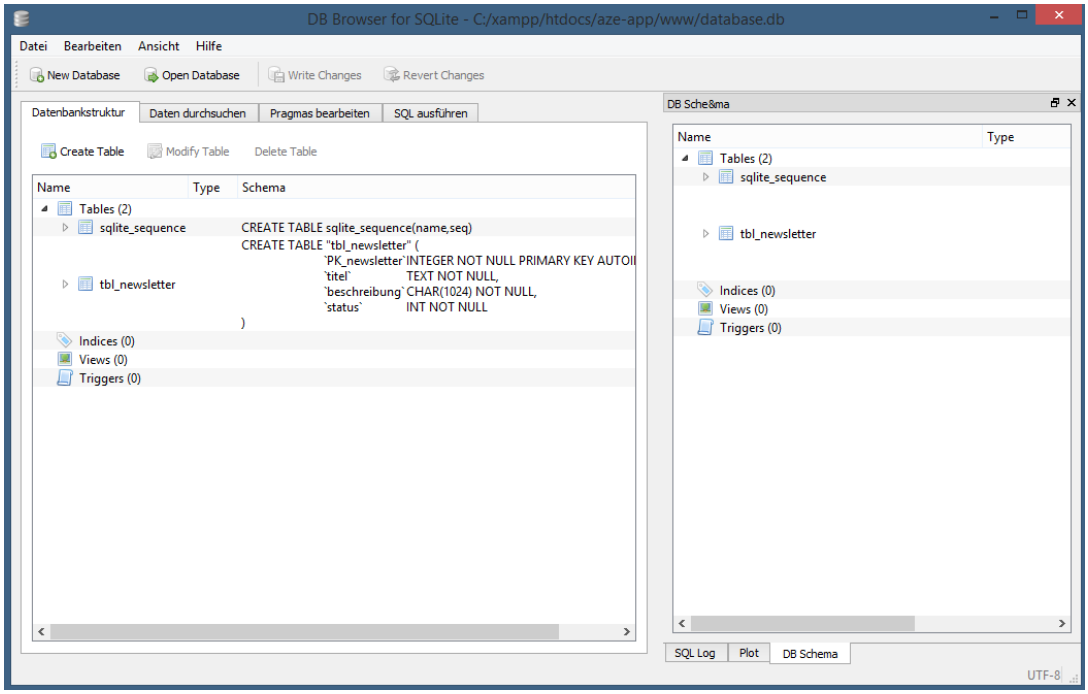


Abbildung 26 Screenshot SQLiteBrowser

Mit diesem Programm erstelle ich schon mal eine Tabelle, in die die Informationen zum Benutzer gespeichert werden. Diese noch leere Tabelle kopiere ich nun in das www-Verzeichnis meines Cordova-Projekts:

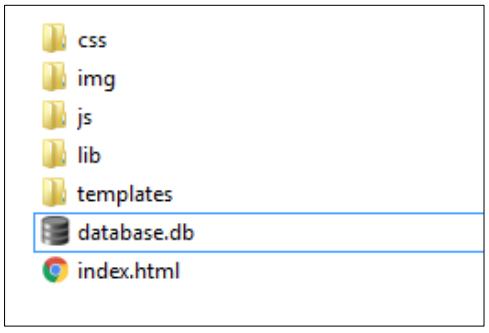


Abbildung 27 Screenshot www-Verzeichnis

4.3.2 Datenbank beim ersten Start verschieben

Die ausführliche Erklärung zum Plugin zum Kopieren der Datenbank gibt es beim **Anbieter des Plugins**. Um dieses Plugin zu installieren führt man im CMD diesen Befehl aus:

```
cordova plugin add https://github.com/an-rahulpandey/cordova-plugin-dbcopy.git
```

Ist das Plugin installiert, folgt die Implementierung im JS-File app.js:

```
// Kopiert die Datenbank in das Standardverzeichnis
window.plugins.sqlDB.copy("database.db", 0, function () {
console.log("Datenbank erfolgreich kopiert!");
}, function(e) {
 console.log("Error Code = " + JSON.stringify(e));
});
```

Im Befehl *window.plugins.sqlDB.copy()* wird der exakte Name der Datenbank samt Dateiendung angegeben. Dann, an welchen Ort dieses File kopiert werden soll, wobei *0* die Standardeinstellung ist. Danach folgen die Funktion bei erfolgreicher Durchführung und die Funktion wenn die Datenbank nicht kopiert werden konnte. Damit eine bestehende Datenbank nicht überschrieben und damit zurückgesetzt wird, wird der Vorgang nur ausgeführt, wenn im Zielverzeichnis keine Datenbank mit demselben Namen existiert.

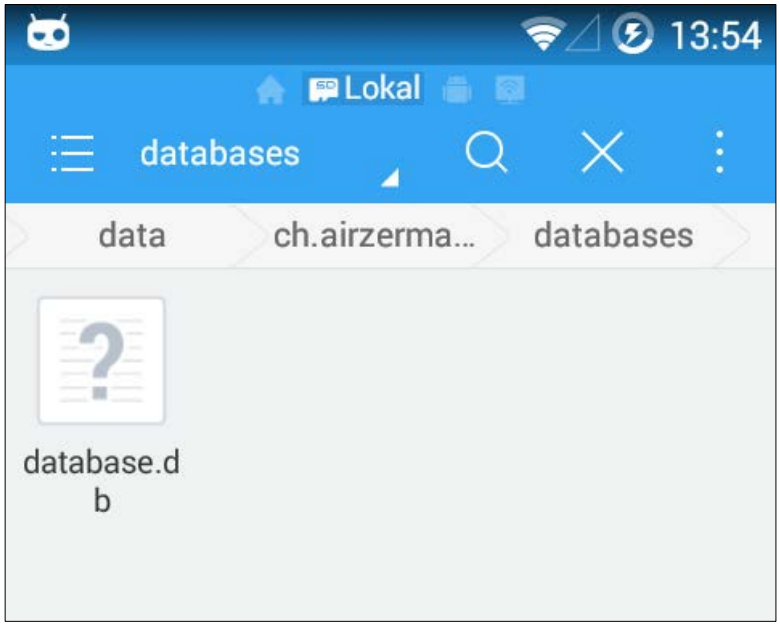


Abbildung 28 Screenshot kopierte Datenbank im Verzeichnis databases

Nun, da die Datenbank an ihrem Platz ist, können wir die Verbindung dazu aufbauen. Dafür wechsle ich zu den Controller und erstelle einen neuen Scope:

```
1  $scope.contact = {};
```

In diesen Scope sollen später die Informationen gespeichert werden, die durch eine Abfrage von der Datenbank resultieren. (Dieses Scope wird später umbenannt!)

```
2  var query = "SELECT * FROM tbl_benutzer";
3  var db = $cordovaSQLite.openDB({name: "database.db"});
4  $cordovaSQLite.execute(db, query, []).then(function(res)
   {
5  console.log("Output: " + JSON.strin-
   gify(res.rows.item(0)));
6
7  $scope.contact = ({
8  anrede: res.rows.item(0).anrede,
9  vorname: res.rows.item(0).vorname,
10 nachname: res.rows.item(0).nachname,
11 adresse: res.rows.item(0).adresse,
12 plz: res.rows.item(0).plz,
13 ort: res.rows.item(0).ort,
14 land: res.rows.item(0).land,
15 email: res.rows.item(0).email,
16 telefon: res.rows.item(0).telefon
17 });
18
19 }, function (err) {
20 console.error(err);
21 $scope.contact = undefined;
22 });
```

In der Variable *query* wird die Abfrage gespeichert, in der Variable *db* der Befehl zum Öffnen der Datenbank. Danach folgt der Befehl zur Abfrage. Der Befehl $cordovaSQLite.execute() erwartet immer drei Parameter: den Aufruf der Datenbank, die Abfrage selbst, und optional ein Array mit den Werten, wenn z.B. ein INSERT- oder ein UPDATE-Befehl ausgeführt wird. Kann die Abfrage durchgeführt werden, so erscheint in der Konsole die entsprechende Nachricht. Weiter werden die abgefragten Informationen ausgegeben und gleichzeitig in den zuvor erstellten Scope gespeichert. Falls die Funktion nicht durchgeführt werden konnte erscheint eine Fehlermeldung in der Konsole. Dies passiert auch bei uns, weil wir noch keinen Eintrag in der Tabelle haben. Sind später einmal Werte gespeichert, werden diese an das Scope übergeben, welches die View aktualisiert und dem Benutzer die Angaben zeigt.

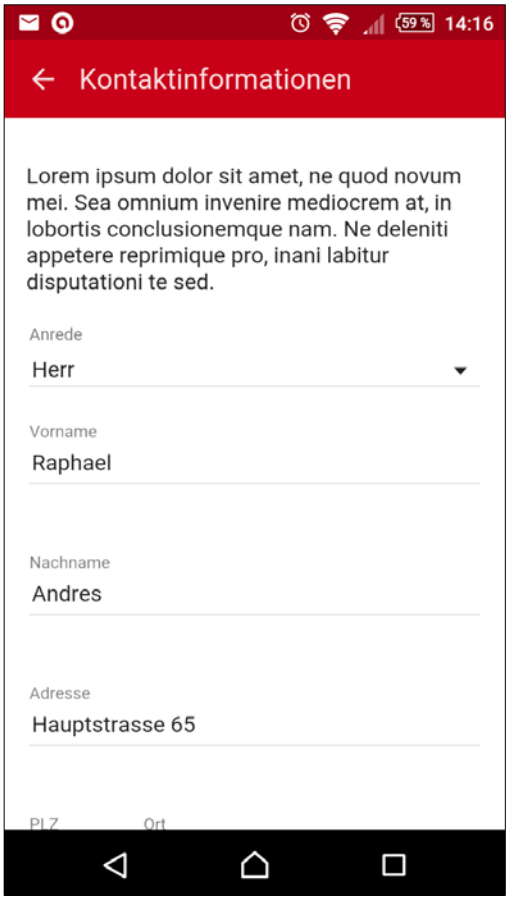


Abbildung 29 Screenshot Kontaktinformationen

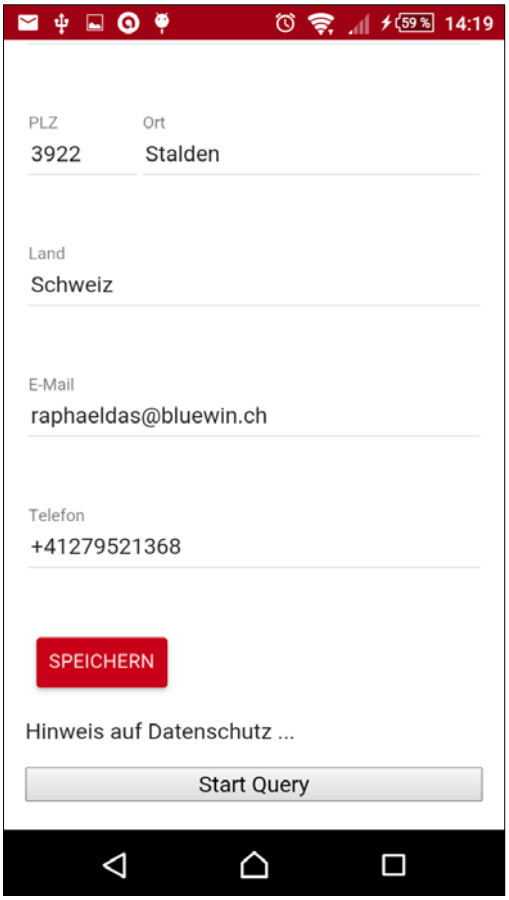


Abbildung 30 Screenshot Kontaktinformationen

Wenn nun der Benutzer die Informationen angibt oder sie ändert und auf Speichern drückt wird eine Funktion gestartet, welche die Daten wieder zurück in die Datenbank speichert. Der Button erhält das Attribut *ng-click="save()"*, was den Befehl auslöst. Die Funktion sieht im File controller.js folgendermassen aus:

```
1  $scope.save = function () {
2  var db = $cordovaSQLite.openDB({name: "database.db"});
3  db.transaction(function(tx) {
4  tx.executeSql("INSERT OR REPLACE INTO tbl_benutzer(PK_be-
   nutzer, anrede, vorname, nachname, adresse, plz, ort,
   land, email, telefon) VALUES(?,?,?,?,?,?,?,?,?,?)", ['',
   $scope.contact.anrede, $scope.contact.vorname,
   $scope.contact.nachname, $scope.contact.adresse,
   $scope.contact.plz, $scope.contact.ort,
   $scope.contact.land, $scope.contact.email,
   $scope.contact.telefon], function() {
5  console.log("INSERT OR REPLACE durchgeführt");
6  });
```

```
7  }, function(err) {
8  console.error(err);
9  }, function(res) {
10 console.log("Gespeichert: " + res.rowsAffected);
11
12 });
13 };
```

In der Variable *db* wird wiederum die zu öffnende Datenbank gespeichert. Danach wird die Transaktion eingeleitet und der SQL-Befehl aufgeführt. *INSERT OR REPLACE INTO* ist ein spezieller Befehl, den es so nur bei SQLite gibt. Ein Datensatz wird entweder eingefügt, oder überschrieben, wenn es ihn bereits gibt. Dazu muss mindestens eine Spalte der Tabelle das Attribut *UNIQUE* enthalten:

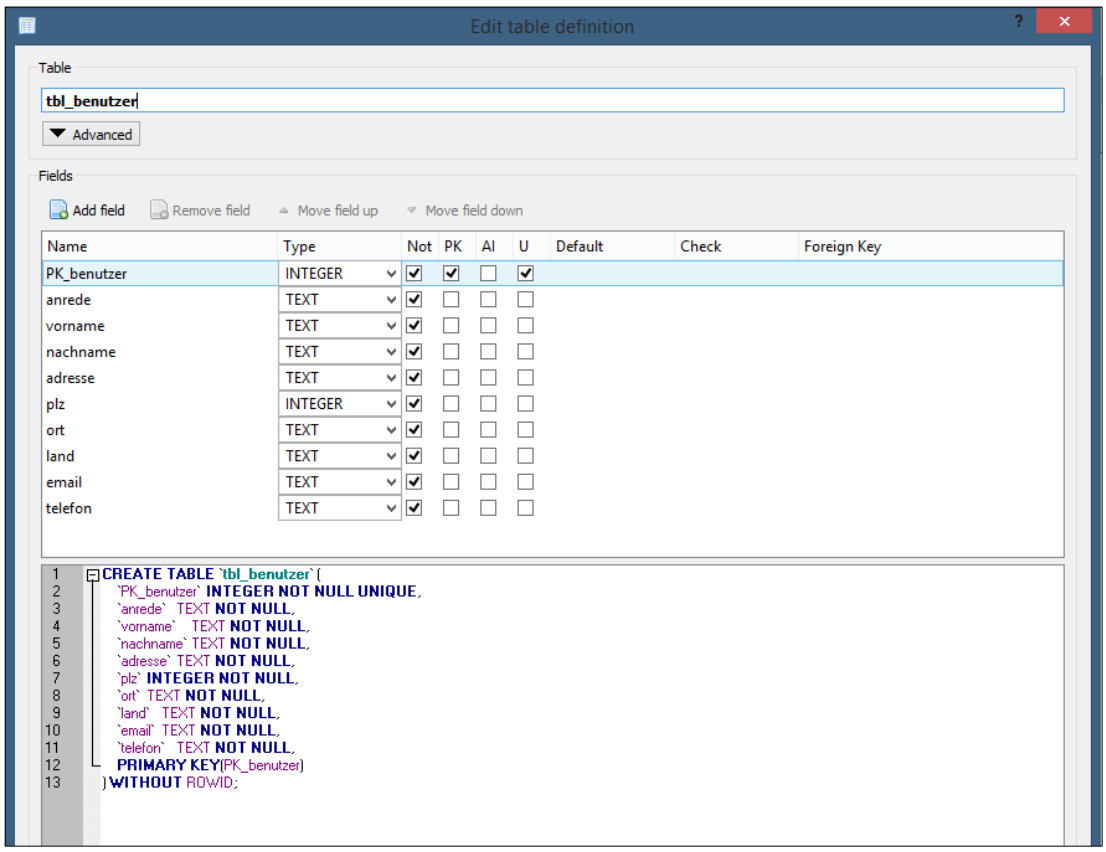


Abbildung 31 Struktur der Tabelle tbl_benutzer

Da in dieser Tabelle sowieso nur ein Datensatz geplant ist, kann die Spalte PK_benutzer dieses Attribut erhalten.

Auffallend ist, dass die Werte nicht direkt in VALUES() geschrieben wird, sondern erst später in den eckigen Klammern. In VALUES() befindet sich nur ein Fragezeichen, als Platzhalter

für den Wert. Da neun Werte folgen, sind es neun Fragezeichen. Dies wird zur Sicherheit gegen SQL Injection so praktiziert.

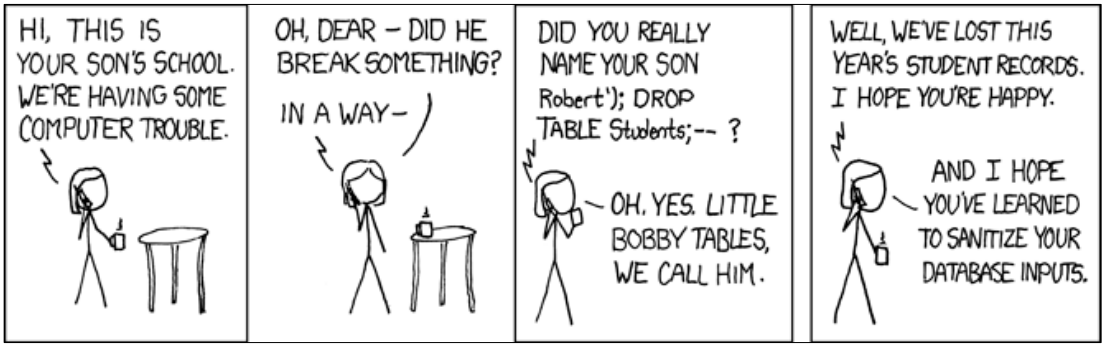


Abbildung 32 Little Bobby Tables

(Quelle: **http://xkcd.com/327/**)

Ist der Aufruft erfolgreich, wird dies in der Konsole durch einen *console.log* bestätigt. Gab es ein Fehler wird auch dieser in der Konsole ausgegeben.

4.4 Überprüfen, ob das Smartphone online ist

Zur Überprüfung, ob das Gerät aktuell mit einem Netzwerk verbunden ist oder nicht, gibt es ein weiteres Plugin von Cordova. Es nennt sich **Network Information** und lässt sich so via Eingabeaufforderung installieren:

```
1  cordova plugin add cordova-plugin-network-information
```

Um nun den Status zu überprüfen, bevor die Daten versendet werden, füge ich eine neue Tabelle namens *tbl_networkstatus* hinzu. In dieser Tabelle gibt es eine einzige Spalte, die *online* heisst. Sie wird entweder den Wert *1* für online oder den Wert *0* für Offline haben. Wenn sich das Gerät mit dem Internet verbindet, sendet es ein Event an Cordova. Mit diesem Event, kann ich nun den Eintrag der Spalte online anpassen. Dasselbe passiert, wenn das Gerät die Verbindung zum Internet verliert. Daher ist es wichtig, dass die App nicht zu sehr abhängig von Internet ist, dass sie auch im Offline-Modul funktioniert. Ein guter Bericht darüber gibt es auf der Website von **Cordova**.

Zuerst erstelle ich das Szenario, dass sich der Status Verändert, währendem die App läuft. Dazu dient dieser Code im File controller.js:

```
1  // Handle Offline Mode
2  document.addEventListener("deviceready", function () {
3  var db = $cordovaSQLite.openDB({name: "database.db"});
4  // listen for Online event
```

```
5  $rootScope.$on('$cordovaNetwork:online', function(event,
   networkState){
6  console.log("Neuer Online-Status: " + networkState);
7  db.transaction(function(tx) {
8  tx.executeSql("UPDATE tbl_networkstatus SET online = ?",
   [1], function() {
9  });
10 }, function(err) {
11 console.error(err);
12 }, function(res) {
13 console.log("Status auf online geändert");
14 });
15 })
16
17 // listen for Offline event
18 $rootScope.$on('$cordovaNetwork:offline', function(event,
   networkState){
19 console.log("Neuer Offline-Status: " + networkState);
20 db.transaction(function(tx) {
21 tx.executeSql("UPDATE tbl_networkstatus SET online = ?",
   [0], function() {
22 });
23 }, function(err) {
24 console.error(err);
25 }, function(res) {
26 console.log("Status auf offline geändert");
27 });
28 })
29
30 }, false);
```

Die beiden Direktiven *$rootScope* und *$cordovSQLite* müssen via Dependency Injection in den Controller eingebettet werden. Die Funktion wird, wie jede andere auch innerhalb des EventListeners ausgeführt. Der **Eventlistener** wartet bis die gesamte Website geladen ist und startet die Funktionen erst, wenn er den Event *deviceready* erhalten hat. Dieser Eventhandler braucht es nicht nur im MainController sondern auch auf allen Unter-Controller. Der *rootScope* erhält eine Meldung, wenn sich der Netzwerkstatus von Online auf Offline ändert oder umgekehrt. Wenn das der Fall ist, gibt er eine Nachricht über die Konsole aus und Ändert den Eintrag in der Tabelle tbl_networkstatus. Da die Tabelle nur eine Zelle besitzt kann diese Abfrage durchgeführt werden:

```
1  tx.executeSql("UPDATE tbl_networkstatus SET online = ?",
   [0], function() {
```

```
2  });
```

Dieser Wert ist beim Starten der App auf 1 also auf Online gestellt. Was ist aber, wenn das Gerät beim Start der App offline ist?

Wir müssen also bereits beim Initialisieren der App überprüfen, ob das Gerät jetzt online oder offline ist:

```
1  var db = $cordovaSQLite.openDB({name: "database.db"});
2
3  if($cordovaNetwork.isOffline()) {
4
5  db.transaction(function(tx) {
6  tx.executeSql("UPDATE tbl_networkstatus SET online = ?",
   [0], function() {
7  });
8  }, function(err) {
9  console.error(err);
10 }, function(res) {
11 console.log("Status auf offline geändert");
12 });
13
14 }
```

Da der Wert standardmässig 1 ist, muss nur untersuchen, ob das Smartphone offline ist. Ist es das, wird wieder dieselbe Abfrage ausgeführt wie schon oben.

4.5 Daten zum Server senden

Um Daten zu transportieren, bietet AngularJS die Funktion *$http* an. Es gibt verschiedene Varianten, wie Daten transportiert werden. Sie werden entweder via **XMLHttpRequest** oder via *JSONP* versendet und entweder mit der Methode *get* oder *post*. Man kann nun Daten mit folgendem Befehl an einen Host versenden:

```
1  var data = $scope.contact;
2  $http.post('http://www.air-zermatt.ch/app/contact.php',
   data)
```

In die Variable *data* wird das Scope *contact* gespeichert. Dieses Scope ist ein Objekt mit den gesamten Kontaktinformationen. Nun nimmt die contact.php die Daten entgegen und verarbeitet sie. Da die Daten in einem JSON-Objekt gespeichert sind, müssen die Daten zuerst decodiert werden. Dann werden die einzelnen Informationen in eine Variable gespeichert:

```
1  $postdata = file_get_contents("php://input");
2  $request = json_decode($postdata);
```

```
3
4  require("connection.php");
5
6  $anrede = mysql_real_escape_string($request->anrede);
7  $vorname = mysql_real_escape_string($request->vorname);
8  $nachname = mysql_real_escape_string($request->nachname);
9  $adresse = mysql_real_escape_string($request->adresse);
10 $plz = mysql_real_escape_string($request->plz);
11 $ort = mysql_real_escape_string($request->ort);
12 $land = mysql_real_escape_string($request->land);
13 $email = mysql_real_escape_string($request->email);
14 $telefon = mysql_real_escape_string($request->telefon);
15 if(isset($request->FK_benutzer)) {
16 $FK_benutzer = mysql_real_escape_string($request->FK_be-
   nutzer);
17 }
```

Als nächstes werden die Variablen in die Datenbank gespeichert. Mit dem Befehl mysql_real_escape_string(); werden Sonderzeichen ausser Gefecht gesetzt, die ansonsten von MySQL falsch interpretiert werden, Stichwort **SQL Injection**. Dies ist ein sehr wichtiger Befehl, der bei jedem String angewendet werden sollte, der von einem Besucher eingegeben wird.

Nun muss unterschieden werden, ob der Benutzer die Kontaktangaben zum ersten Mal speichert, oder ob es nur einen Änderung ist. Speichert er die Daten zum ersten Mal muss der neue Datensatz via INSERT eingefügt, ansonsten via UPDATE aktualisiert werden. Um dies unterscheiden zu können, füge ich in der lokalen Datenbank auf dem Smartphone eine weiter Spalte ein. Diese Spalte heisst FK_benutzer und dient quasi als Fremdschlüssel für den Eintrag in der AZE-Datenbank. Ist dieser Schlüssel nicht definiert, wissen wir, dass der Benutzer die Infos zum ersten Mal speichert und der INSERT-Befehl wird dementsprechend angewendet. Mit der Funktion mysql_insert_id() erhalten wir dann die ID des gerade eingefügten Datensatzes zurück. Diese ID senden wir wieder zurück zum Gerät. Nach dem obigen Code kann also folgendes angefügt werden:

```
1  require("../connection.php");
2
3  if(isset($FK_benutzer)) {
4  $query = mysql_query("UPDATE tbl_benutzer SET anrede =
   '$anrede', vorname = '$vorname', nachname = '$nachname',
   adresse = '$adresse', plz = '$plz', ort = '$ort', land =
   '$land', email = '$email', telefon = '$telefon' WHERE
   PK_benutzer = '$FK_benutzer'");
5  if($query) {
```

```
6  $response = array('response' => 'Daten geändert!',
   'FK_benutzer' => $FK_benutzer);
7  echo json_encode($response);
8  } else {
9  $response = array('response' => 'Daten nicht geändert: '
   . mysql_error(), 'FK_benutzer' => $FK_benutzer);
10 echo json_encode($response);
11 }
12
13 } else {
14 $query = mysql_query("INSERT INTO tbl_benutzer(anrede,
   vorname, nachname, adresse, plz, ort, land, email, tele-
   fon)
15 VALUES('$anrede', '$vorname', '$nachname', '$adresse',
   '$plz', '$ort', '$land', '$email', '$telefon')");
16 if($query) {
17 $response = array('response' => 'Daten gespeichert!',
   'FK_benutzer' => mysql_insert_id());
18 echo json_encode($response);
19 } else {
20 $response = array('response' => 'Daten nicht geändert: '
   . mysql_error(), 'FK_benutzer' => 'null');
21 echo json_encode($response);
22 }
23 }
```

Am Anfang wird die Verbindung mit der Datenbank aufgebaut. Dann wird überprüft, ob die Variable FK_benutzer definiert ist. Ist er das, so wird der Befehl zum Aktualisieren des Datensatzes ausgeführt. Wenn die Abfrage erfolgreich durchgeführt wurde, wird eine Antwort ausgegeben und der Fremdschlüssel FK_benutzer. Da das Smartphone ein JSON-Objekt als Antwort erwartet, wird die Ausgabe wieder codiert. Dasselbe passiert, wenn die Abfrage nicht erfolgreich war.

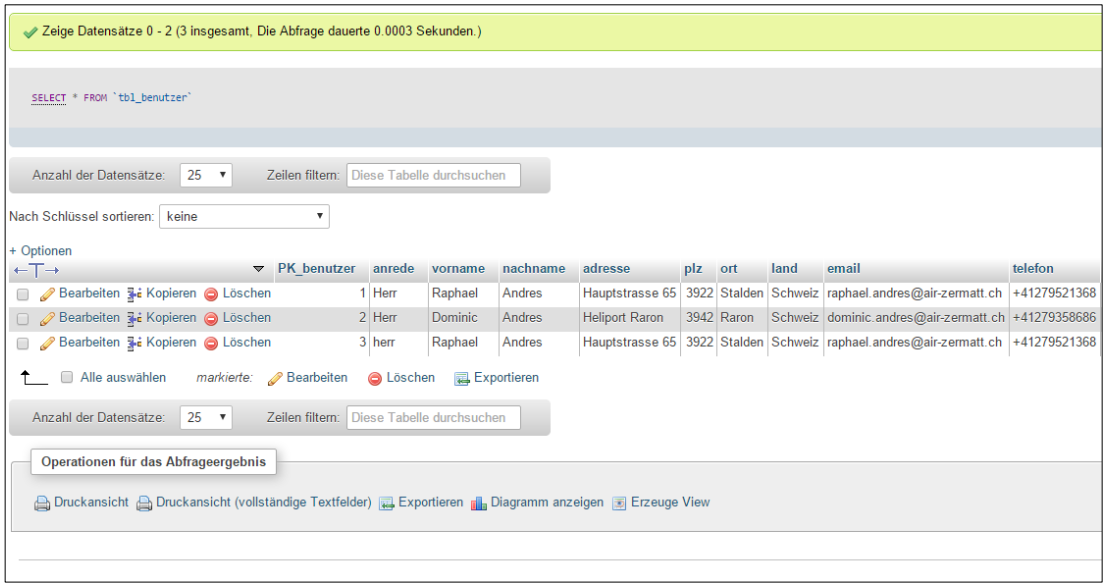


Abbildung 33 Screenshot Tabelle tl_benutzer auf dem AZE-Server

Ist die Variable FK_benutzer nicht definiert, so wird ein INSERT-Befehl gestartet und die Informationen werden eingefügt. Als Antwort wird die ID des gerade eingefügten Datensatzes herausgegeben und codiert. War die Abfrage nicht erfolgreich wird mit einer Fehlermeldung geantwortet.

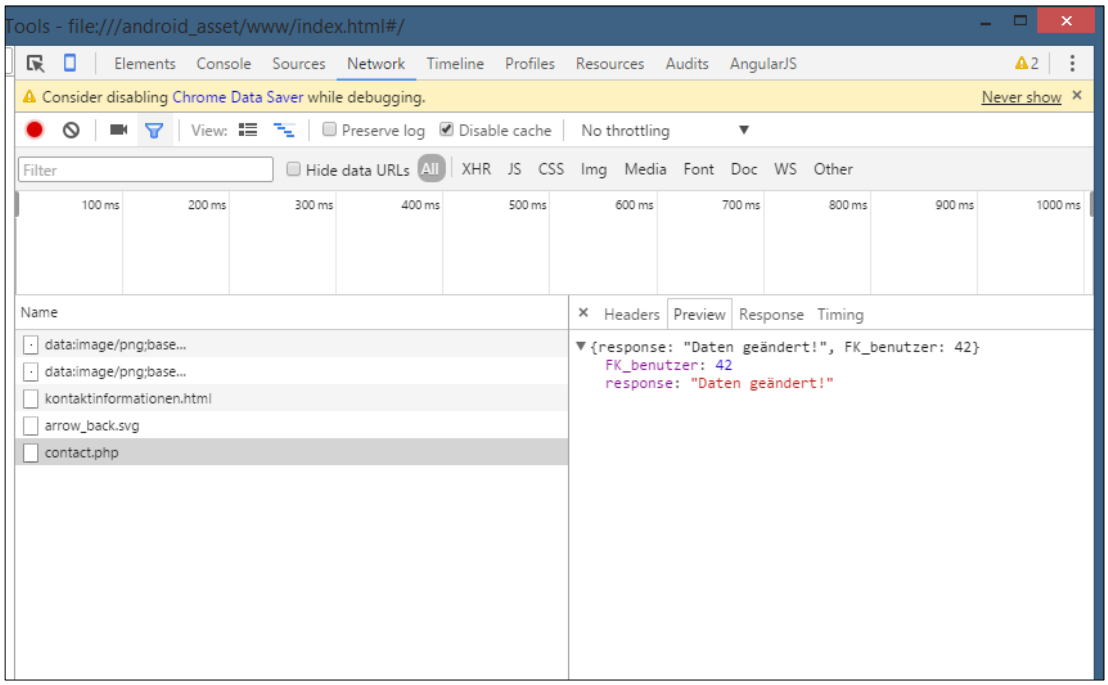


Abbildung 34 Screenshot Google Developer Tools auf dem Smartphone

Und so sieht dann die Antwort des Files auf dem Gerät aus. Rechts im Register *Preview* sieht man das Objekt mit dem Fremdschlüssel des Eintrages. Um nun dieser Fremdschlüssel lokal

zu speichern und dem Benutzer eine Rückmeldung zu geben, dass seine Daten gespeichert wurden, wechseln wir wieder zu den Controller und fügen weiteren Code hinzu. Wir haben bereits die Funktion $http zum Senden der Daten erstellt. Nun wird dieser Befehl noch erweitert:

```
1  db.transaction(function(tx) {
2  tx.executeSql("SELECT online FROM tbl_networkstatus", [],
   function(tx, res) {
3  if(res.rows.item(0).online) {
4  var data = $scope.contact;
5  $http.post('http://www.air-zermatt.ch/app/contact.php',
   data).then(function(response) {
6  $scope.contact.FK_benutzer = response.data.FK_benutzer;
7  db.transaction(function(tx) {
8  tx.executeSql("UPDATE tbl_benutzer SET FK_benutzer = ?",
   [response.data.FK_benutzer], function() {
9  console.log("FK gespeichert!");
10
11 }, function(err) {
12 console.error(err);
13 }, function(res) {
14 console.log("Transaktion durchgeführt");
15 console.log($scope.contact.FK_benutzer);
16 });
17
18 }, function(error) {
19 console.log(error);
20 });
21 });
22 }
23 });
24 }, function(err) {
25 console.error(err);
26 }, function(res) {
27 // console.log("Status ermittelt");
28 });
29 if(info !== 'no_info') {
30 $scope.InfoToast();
31 }
```

Als erstes wird überprüft, ob der Eintrag online in der Tabelle tbl_networkstatus auf 1 gestellt ist, also, ob das Gerät mit dem Internet verbunden ist. Dann werden die Daten ausgelesen

und an die angegeben Daten auf dem AZE-Server gesendet. Danach wird die Antwort dieses Scripts verarbeitet. Mit der Zeile

```
1  $scope.contact.FK_benutzer = response.data.FK_benutzer;
```

speichern wir den Fremdschlüssel im Scope *contact*. Weiter wird ein SQL-Befehl ausgeführt, der den Fremdschlüssel auch in die Datenbank speichert. Es folgen mehrere Funktionen, die bei fehlerhaften Ausführung aufgerufen werden. Zum Schluss wird die Funktion InfoToast() aufgerufen. Diese Funktion lässt eine Nachricht auf dem Bildschirm erscheinen:

```
1  // Toast Daten gespeichert
2  $scope.InfoToast = function($event) {
3  $mdToast.show($mdToast.simple()
4  .textContent('Informationen gespeichert')
5  .action('Ok')
6  .hideDelay(4000)
7  );
```

Diese Nachricht wird mithilfe von Angular Material ausgegeben. Weitere Informationen findet man auf deren **Website**.

Was jetzt noch beachtet werden muss, ist das Szenario, wenn das Gerät offline ist, der Benutzer die Daten angibt und speichert. Die Daten werden nun zwar auf die lokale Datenbank gespeichert, aber nicht auf der Datenbank unseres Servers. Dazu erstelle ich eine weitere Funktion im MainCtrl, genauer in der rootScope-Funktion, die aufgerufen wird, wenn der Netzwerkstatus auf online wechselt:

```
1  $rootScope.$on('$cordovaNetwork:online', function(event,
   networkState){
2  console.log("Neuer Online-Status: " + networkState);
3  $rootScope.$emit('wentOnline', {});
4  })
```

Da sich diese Funktion in einem anderen Controller befindet muss ich mithilfe von $emit und $on einen eigenen Event kreieren, der dann die Funktion zum Speichert der Daten auslöst. Dieser Code-Schnipsel gehört in den Controller ContactCtrl:

```
1  $rootScope.$on('wentOnline', function() {
2  // Check if something has changed!
3  $scope.save('no_info');
4  });
```

Die Funktion $scope.save() wird aufgerufen und der String *'no_info'* wird übergeben. Das daher, weil wir keine grafische Benachrichtigung ausgeben wollen. Zudem habe ich einen

Platzhalter eingefügt. Später sollte zuerst überprüft werden, ob sich seit dem letzten Speichern etwas an den Kontaktinformationen geändert hat. Damit kann der Datenverkehr minimiert werden.

5 SEITE NEWSTICKER ERSTELLEN

5.1 Verbindung zum GCM aufbauen

Damit man einem Smartphone Informationen zusenden kann, muss sich das Gerät bei einem Push-Notification-Dienstleister, im konkreten Fall mit dem von Google, registrieren. Dieses Verfahren habe ich bereits in der Vorbereitungs-Phase getestet und meine Fortschritte dort **dokumentiert**.

Konkret füge ich nun ein weiteres Plugin hinzu:

```
cordova plugin add phonegap-plugin-push
```

Mit dem Phonegap Plugin Push lässt sich sehr einfach eine Verbindung zum jeweiligen Push Message-Provider herstellen.

```
var push = PushNotification.init({
android: {
senderID: "12345679",
"sound": "true",
"icon": "phonegap",
"iconColor": "#29d",
"forceShow": "true"
},
ios: {
alert: "true",
badge: true,
sound: 'false'
},
windows: {}
});
```

Als erstes muss für jede Plattform gewisse Informationen angegeben werden. Bei Android ist dies die senderID und weitere optionale Angaben. Nachdem diese Angaben definiert sind, folgt das erste Szenario: Die Registration beim Provider:

```
push.on('registration', function(data) {
console.log("registration event");
// console.log(JSON.stringify(data));
angular.element(document.getElementById('wrap-
perCtrl')).scope().save_regId(data);
});
```

Auf die Funktion *registration* antwortet das Plugin mit der erhaltenen **Registration-ID**. Diese kann nun weiterverwendet werden. In diesem Beispiel wird sie erstmal in der Konsole ausgegeben und dann weitergegeben an folgende Funktion in den Controller:

```
1  $scope.save_regId = function(data) {
2  db.transaction(function(tx) {
3  tx.executeSql("UPDATE tbl_benutzer SET registration_id =
   ?", [data.registrationId], function() {});
4  }, function(err) {
5  console.error(err);
6  }, function(res) {
7  console.log("RegistrationId gespeichert!");
8  });
9  }
```

Hier wird die Registration-ID in die lokale Datenbank gespeichert.

Neben der Registration-ID werden später auch die Informationen zum Newsticker lokal gespeichert. Dazu erstelle ich eine weitere Tabelle mit dem Namen *tbl_newsticker*:

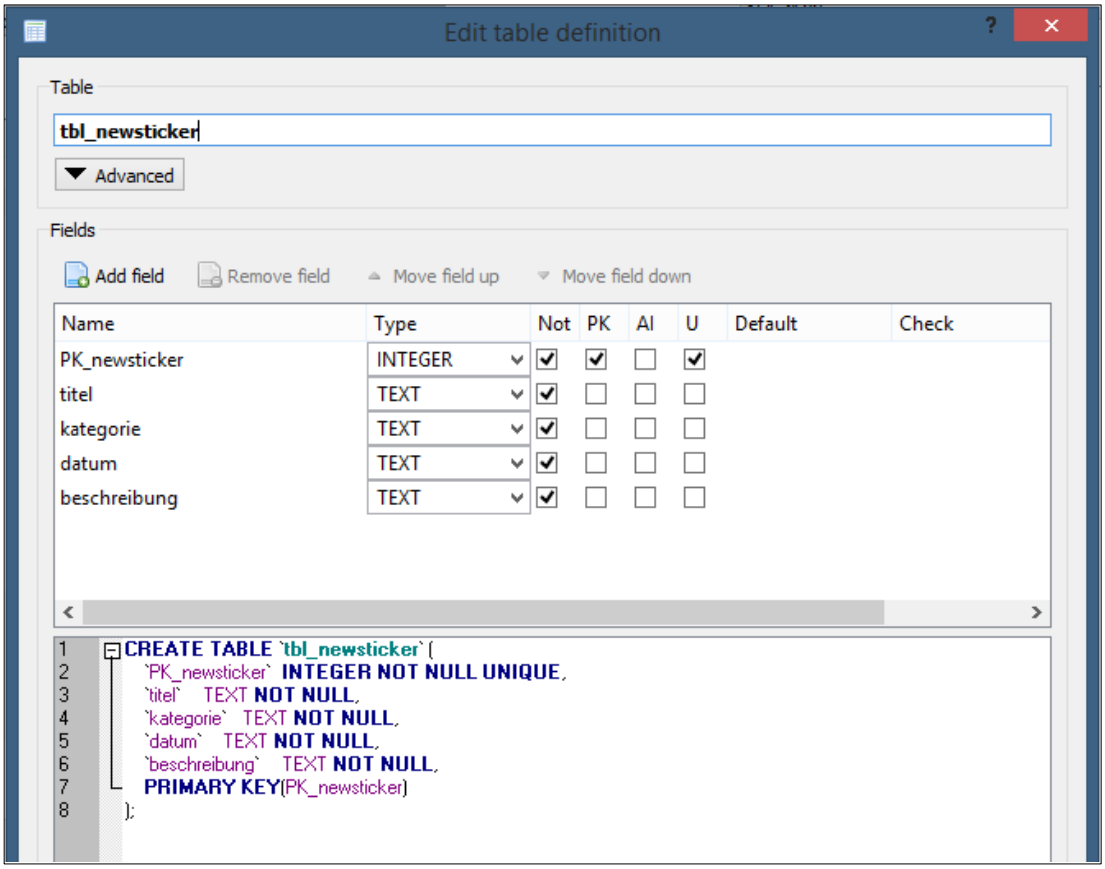


Abbildung 35 Definition Tabelle Newsticker

In diese Tabellen werden die Details zum Newsticker gespeichert. Damit sind diese Informationen für den Benutzter auch verfügbar, wenn er offline ist. Die Bilder werden ebenfalls auf das Gerät geladen und gespeichert.

5.2 Payload entgegen nehmen und speichern

Wenn das Gerät eine Benachrichtigung erhält, wird ein Event gestartet. Mit dem Event *notification* erhält man das Payload vom Push-Benachrichtigungs-Provider und kann diese verarbeiten:

```
1  push.on('notification', function(data) {
2  console.log("notification event");
3  angular.element(document.getElementById('wrap-
   perCtrl')).scope().push(data);
4  });
```

Mit diesem Code-Schnipsel wird das Payload in der Konsole ausgegeben. Weiter werden die Daten an die Funktion push() im Controller MainCtrl gesendet. Diese Funktion verwertet dann diese Daten weiter. Als erstes wird der Typ der Benachrichtigung festgestellt:

```
1  if(data.additionalData.push_type == 'newsticker') {
2  }
```

Der dazugehörige Wert wird beim serverseitigen Versenden des Payloads definiert. Handelt es sich bei der Nachricht um einen Newsticker-Beitrag werden die erhaltenen Daten in Variablen zwischengespeichert:

```
1  // store values in variable
2  var titel = data.title;
3  var kategorie = data.additionalData.kategorie;
4  var datum = data.additionalData.datum;
5  var beschreibung = data.message;
6  var bild = data.additionalData.file;
```

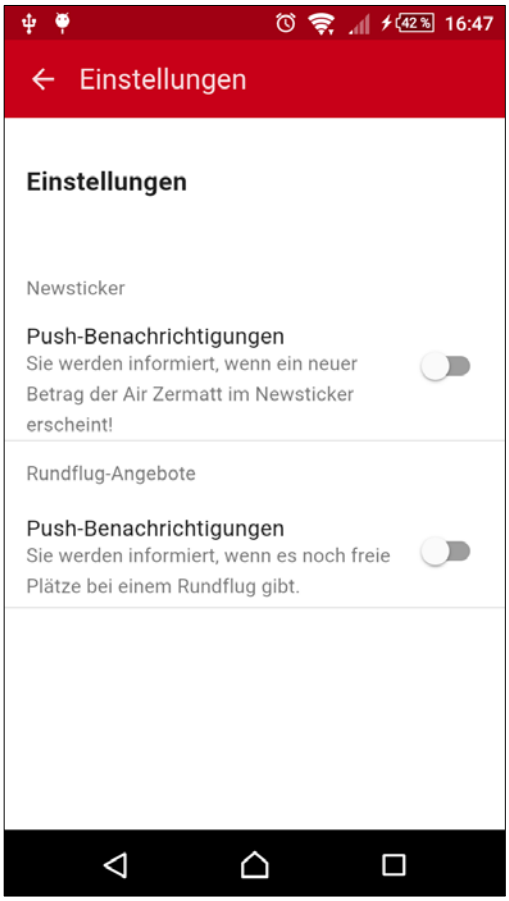
Nachfolgend werden die Daten in die Datenbank übernommen:

```
1  // store in database
2  db.transaction(function(tx) {
3  tx.executeSql("INSERT INTO tbl_newsticker(titel, katego-
   rie, datum, beschreibung, bild) VALUES(?,?,?,?,?)", [ti-
   tel, kategorie, datum, beschreibung, bild], function()
   {});
4  }, function(err) {
5  console.error(err);
6  }, function(res) {
```

```
7  console.log("Neuer Beitrag gespeichert!");
8  });
9
10 window.location.href = '#/newsticker';
```

Weiter wird mithilfe von *window.location.href* auf die Unterseite Newsticker verlinkt, damit dem Benutzer der neue Beitrag direkt angezeigt wird.

5.3 Einstellungen der Push-Benachrichtigungen



Vielleicht gibt es Benutzer der App, die keine Push-Benachrichtigungen erhalten möchten. Deshalb erstelle ich eine Seite Einstellungen, in der die Benutzer diese Funktion deaktivieren können. Zu Beginn gibt es zwei Arten von Push-Benachrichtigungen, eine für den Newsticker und eine für die Rundflug-Angebote. Ich erstelle also mithilfe von Angular Material zwei Felder mit Schalter, zum ein- bzw. ausschalten der Push-Funktion:

Abbildung 36 Screenshot Seite Einstellungen

Diese Darstellung erreicht man mit diesem Code:

```
1  <md-list>
2  <md-subheader class="">Newsticker</md-subheader>
3  <md-list-item class="md-2-line">
4  <div class="md-list-item-text">
5  <h3>Push-Benachrichtigungen</h3>
```

```
6  <p>Sie werden informiert, wenn ein neuer Betrag der Air
   Zermatt im Newsticker erscheint!</p>
7  </div>
8  <div><md-switch ng-model="einstellungen.newsticker_push"
   aria-label="Push Newsticker">
9  </md-switch>
10 </div>
11 <md-divider ng-if="!$last"></md-divider>
12 </md-list-item>
13 <md-subheader class="">Rundflug-Angebote</md-subheader>
14 <md-list-item class="md-2-line">
15 <div class="md-list-item-text">
16 <h3>Push-Benachrichtigungen</h3>
17 <p>Sie werden informiert, wenn es noch freie Plätze bei
   einem Rundflug gibt.</p>
18 </div>
19 <div><md-switch ng-model="einstellungen.rundflug_push" a-
   ria-label="Push Rundflug">
20 </md-switch>
21 </div>
22 <md-divider ng-if="!$last"></md-divider>
23 </md-list-item>
24 </md-list>
```

Die jeweiligen Einstellungen sind im *$scope.einstellungen.newsticker_push* bzw. *$scope.einstellungen.rundflug_push* gespeichert. Diese beiden Werte werden beim Laden der Unterseite aus der Datenbank ausgelesen. Sie werden direkt beim Ändern wieder gespeichert, weshalb es hier keinen Speichern-Button gibt. Nun müssen diese beiden Werte noch auf unserem Server gespeichert werden. Dazu fügen wir die lokale Tabelle mit den beiden Einstellungen zur Abfrage hinzu:

```
1  var query = "SELECT * FROM tbl_benutzer, tbl_einstel-
   lung";
2  $cordovaSQLite.execute(db, query, []).then(function(res)
   {
3  console.log("Output: " + JSON.strin-
   gify(res.rows.item(0)));
4
5  $scope.infos = ({
6  anrede: res.rows.item(0).anrede,
7  vorname: res.rows.item(0).vorname,
8  nachname: res.rows.item(0).nachname,
9  adresse: res.rows.item(0).adresse,
10 plz: res.rows.item(0).plz,
```

```
ort: res.rows.item(0).ort,
land: res.rows.item(0).land,
email: res.rows.item(0).email,
telefon: res.rows.item(0).telefon,
FK_benutzer: res.rows.item(0).FK_benutzer,
registration_id: res.rows.item(0).registration_id,
newsticker_push: res.rows.item(0).newsticker_push,
rundflug_push: res.rows.item(0).rundflug_push
});
```

Zur besseren Verständnis habe ich das $scope-Element zu *infos* umbenannt! Serverseitig werden die beiden Daten entgegengenommen:

```
$newsticker_push = mysql_real_escape_string($request->newsticker_push);
$rundflug_push = mysql_real_escape_string($request->rundflug_push);
```

Und fliessen in die Abfrage ein. Die Einstellungen des Benutzers sind nun in unserer Datenbank gespeichert:

	newsticker_push	rundflug_push
)CPjO-f3I1Nj...	0	0
O1dQ3Dq...	0	0
)Ns60Mg5oY...	0	0
qkN_oNv...	0	0
	0	0
ZTKIE9OQs...	0	0
8Itmjy_d2...	0	0
onD0-M31t...	0	0
x5IazIVNP...	0	0
qE2R7...	0	0
6_aT38I...	0	0
53H8kV...	0	0
	0	0
53H8kV...	0	0
IDP_-ZOA...	1	0
8Itmjy_d2...	1	0

Abbildung 37 Screenshot Einstellungen in der Tabelle tbl_benutzer

5.4 Push-Benachrichtigung serverseitig ausgeben

Damit der Benutzer eine Push-Benachrichtigung empfangen kann, müssen wir auch dafür sorgen, dass eine Payload von unserem Server aus versendet wird. Deshalb passen wir nun das Eingabeformular für den Newsticker etwas an. Unterhalb der Abfrage, die die ID des Beitrags und die ID des Bildes in der Tabellen tbl_news_bild speichert, fügen wir folgenden Code hinzu:

```
1  if(isset($id_bild)) {
2  $query = mysql_query("INSERT INTO tbl_news_bild(FK_news,
   FK_bild)
3  VALUES('$id_news', '$id_bild')");
4  if($query) {
5  echo "Bild und News verkn&uuml;pft!";
6
7  // API access key from Google API's Console
8  define( 'API_ACCESS_KEY', '[API_ACCESS_KEY hier
   einfügen]' );
9
10 $query = mysql_query("SELECT registration_id FROM tbl_be-
   nutzer WHERE newsticker_push = 'true'");
11 while($row = mysql_fetch_array($query)) {
12 $registrationIds[] = $row["registration_id"];
13 }
14
15 // prep the bundle
16 $msg = array(
17       'file'      => $file_name,
18       'dasundas'  => '987 und 22der',
19       'message'   => $beschreibung,
20       'title'           => $titel,
21       'subtitle'  => 'und das der Untertitel',
22       'kategorie' => $kategorie,
23       'vibrate'   => 1,
24       'sound'           => 'default',
25       'largeIcon' => 'large_icon',
26       'smallIcon' => 'small_icon',
27       'style'     => 'inbox',
28       'summaryText'     => '%n% neue Nachrichten',
29 );
30
31 $fields = array(
32       'registration_ids'      => $registrationIds,
```

```
33      'data'                  => $msg
34 );
35
36 $headers = array(
37      'Authorization: key=' . API_ACCESS_KEY,
38      'Content-Type: application/json'
39 );
40
41 $ch = curl_init();
42 curl_setopt($ch,CURLOPT_URL, 'https://android.goog-
   leapis.com/gcm/send');
43 curl_setopt($ch,CURLOPT_POST, true);
44 curl_setopt($ch,CURLOPT_HTTPAUTH, CURLAUTH_BASIC);
45 curl_setopt($ch,CURLOPT_HTTPHEADER, $headers);
46 curl_setopt($ch,CURLOPT_IPRESOLVE, CURL_IPRESOLVE_V4);
47 curl_setopt($ch,CURLOPT_RETURNTRANSFER, true);
48 curl_setopt($ch,CURLOPT_SSL_VERIFYPEER, 0);
49 curl_setopt($ch,CURLOPT_POSTFIELDS, json_en-
   code($fields));
50 $result = curl_exec($ch);
51
52 curl_close($ch);
53 echo $result;
54 }
55 }
```

Dieser Code habe ich von der Website **https://gist.github.com/prime31/5675017** entnommen und für meinen Gebrauch angepasst. Als erstes wird der notwendige API Access Key definiert. Mit diesem Verifizieren wir uns bei Google. Dann wählen wir alle Registration-IDs aus der Datenbank, deren Wert von *newsticker_push 1*, also *true*, ist und fügen sie in ein Array ein. Dann wird das eigentliche Payload definiert mit den verschiedenen Informationen. Diese werden wieder in ein Array gespeichert. Schliesslich werden diese beiden Arrays wieder ein Array verpackt und die notwendigen Header werden angegeben. Dann wird die PHP-Funktion CURL initialisiert. Mit diesem Befehl können Informationen versendet werden, das Prinzip ist dasselbe wie bei einem Formular. Zum Schluss wird dieses ganze Paket ausgeführt, die Daten werden an die Adresse *https://android.googleapis.com/gcm/send* gesendet und die Antwort vom Google-Server wird entgegengenommen und ausgegeben:

Abbildung 38 Bestätigung nach erfolgreichem Versenden eines Payloads

Nun gibt es beim GCM ein Limit wie viele Geräte pro Push-Benachrichtigung angegeben werden können. Dieses Limit liegt aktuell bei 1000 registrierten Geräten. (**https://developers.google.com/cloud-messaging/http-server-ref**, Stand: Februar 2016).

Zu Beginn wird dieses Limit vielleicht noch nicht erreicht. Mit der Zeit aber könnte dieses Limit aber erreicht werden. Zu Vergleich: Die Facebook-Seite von der Air Zermatt hat aktuell um die 11'000 Likes, der Newsticker wird pro Woche etwa 1200 Mal aufgerufen und da die App gratis sein wird, kann erwartet werden, das dieses Limit mit der Zeit überschritten wird. Dementsprechend dürfen auch nicht mehr als 1000 Einträge gleichzeitig in das Array geladen werden. Dafür splitte ich den Sendeprozess auf. Ich erstelle eine While-Schlaufe und frage die Einträge in Blöcke von 950 Stück ab. Etwas Reserve schadet bestimmt nicht. Um diese Funktion zu testen erstellte ich zuerst Blöcke von 5 Einträgen:

```
1  $total = mysql_num_rows(mysql_query("SELECT registra-
   tion_id FROM tbl_benutzer WHERE newsticker_push = '1'"));
2  // echo $total;
3  $limit = 5;
4  $offset = 0;
5
6  while($offset <= $total) {
7
8  $query = mysql_query("SELECT registration_id FROM tbl_be-
   nutzer WHERE newsticker_push = 'true' LIMIT $offset,
   $limit");
9
10 while($row = mysql_fetch_array($query)) {
11 $registrationIds[] = $row["registration_id"];
12 }
13
```

```
14 $fields = array(
15       'registration_ids'      => $registrationIds,
16       'data'                  => $msg
17 );
18
19 $ch = curl_init();
20 curl_setopt($ch,CURLOPT_URL, 'https://android.goog-
   leapis.com/gcm/send');
21 curl_setopt($ch,CURLOPT_POST, true);
22 curl_setopt($ch,CURLOPT_HTTPAUTH, CURLAUTH_BASIC);
23 curl_setopt($ch,CURLOPT_HTTPHEADER, $headers);
24 curl_setopt($ch,CURLOPT_IPRESOLVE, CURL_IPRESOLVE_V4);
25 curl_setopt($ch,CURLOPT_RETURNTRANSFER, true);
26 curl_setopt($ch,CURLOPT_SSL_VERIFYPEER, 0);
27 curl_setopt($ch,CURLOPT_POSTFIELDS, json_en-
   code($fields));
28 $result = curl_exec($ch);
29
30 curl_close($ch);
31 echo $result;
32 $offset = $offset + $limit;
33 unset($ch);
34 unset($registrationIds);
35 }
```

Als erstes führe eine Abfrage durch, die mir die Anzahl Einträge ausgibt. Dann definiere ich in der Variable *$limit* das Limit der Abfrage. Die Variable $offset brauche ich, damit ich nach der ersten Serie der Abfrage einen Wiedereinstiegspunkt habe und die nächsten paar Datensätze geladen werden. Die Schlaufe wird solange ausgeführt, bis der Wert des Offsets grösser ist, als die totale Anzahl Datensätze, wobei das Offset nach jeder Abfrage um den Wert des Limits erhöht wird. Die Abfrage selbst ist in der Variable $query gespeichert:

```
1  mysql_query("SELECT registration_id FROM tbl_benutzer
   WHERE newsticker_push = 'true' LIMIT $offset, $limit");
```

Danach wird in einer weiteren While-Schlaufe die ausgelesenen Datensätze in das Array *$registrationIds* gespeichert. Der weitere Prozess blieb unverändert. Zu Schluss werden die Variablen wieder geleert, das Offset wird um den Wert des Limits erhöht und die Schlaufe beginnt wieder von vorne.

5.5 Newsticker-Beitrag auflisten

Als nächstes möchten wir die Beiträge des Newsticker auflisten. Dazu erstelle ich einen neuen Controller, der auf der Seite Newsticker implementiert wird. Den Controller nenne ich *NewsCtrl*. Darin frage ich die Datensätze der Tabelle *tbl_newsticker* ab:

```
1  var db = $cordovaSQLite.openDB({name: "database.db"});
2  $scope.newsticker = [];
3  var query = "SELECT * FROM tbl_newsticker";
4  $cordovaSQLite.execute(db, query, []).then(function(res)
   {
5  count = res.rows.length;
6  for(i=0;i<count;i++) {
7  $scope.newsticker.push({
8  titel: res.rows.item(i).titel,
9  kategorie: res.rows.item(i).kategorie,
10 datum: res.rows.item(i).datum,
11 beschreibung: res.rows.item(i).beschreibung,
12 bild: cordova.file.dataDirectory + '/bilder/' +
   res.rows.item(i).bild });
13 };
14
15 bild = res.rows.item(i).bild;
```

Ich definiere den Scope $scope.newsticker als Objekt. Danach folgt die Abfrage. Mit der for-Schlaufe speichere ich jeden einzelnen Datensatz in den Scope.

In den Key *bild* speichere ich direkt den Link zum heruntergeladenen Bild. Dazu brauche ich das Plugin *$cordovaFile* von ngCordova:

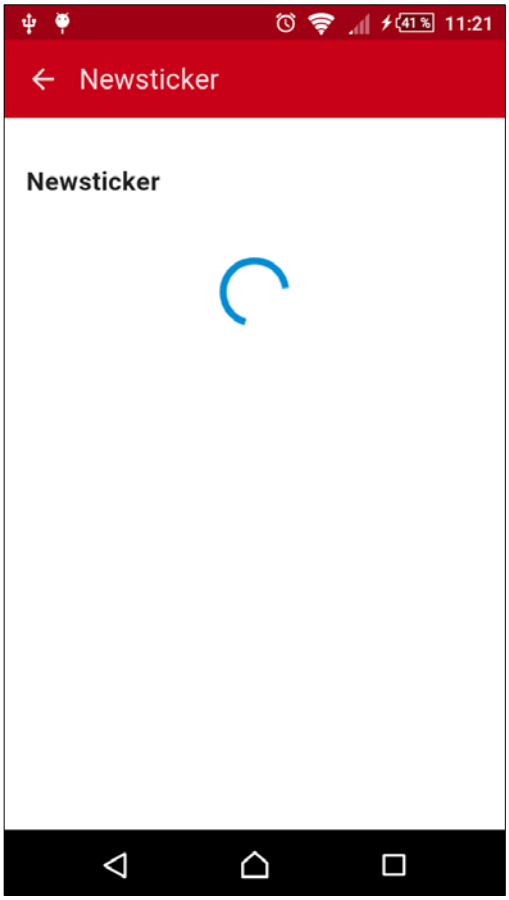
```
1  cordova plugin add cordova-plugin-file
```

Der Begriff *cordova.file.dataDirectory* bezeichnet den Dateipfad, der der Applikation zur Verfügung steht. Danach gebe ich den Unterordner an, indem die Bilder gespeichert sind. Zusätzlich speichere ich den Wert für das Bild in eine separate Variable ab. Diese brauche ich später zum Überprüfen, ob das Bild überhaupt auf dem Gerät gespeichert ist, oder ob es zuerst noch heruntergeladen werden muss. Um dies zu überprüfen, kann folgender Code verwendet werden:

```
1  $cordovaFile.checkFile(cordova.file.dataDirectory +
   "/bilder/", bild)
2  .then(function (success) {
3  console.log('Bild existiert!');
4  }, function (error) {
```

```
5
6  var url = "http://www.air-zermatt.ch/app/bilder/" + bild;
7  var targetPath = cordova.file.dataDirectory + '/bilder/'
   + bild;
8  var trustHosts = true;
9  var options = {};
10
11 $cordovaFileTransfer.download(url, targetPath, options,
   trustHosts)
12 .then(function(result) {
13 console.log('Bilder heruntergeladen');
14 }, function(err) {
15 console.log('Bild konnte nicht heruntergeladen werden:' +
   JSON.stringify(err));
16 });
17
18 }, false);
```

checkFile() ist der Befehl zum Überprüfen, ob eine gewisse Datei vorhanden ist. Als erster Parameter wird der Pfad angegeben, der durchsucht werden soll. Der zweite Parameter ist der Name der Datei. Hier kann ich die Variable von oben nehmen. Ist die Datei vorhanden, so wird dies in der Konsole ausgegeben. Ist die Daten nicht vorhanden, starte ich eine neue Funktion, die das fehlende Bilder von unserem Server herunterladet. Die URL und der Pfad, in den die Datei gespeichert werden soll, wird in der Variable gespeichert. Die beiden anderen Variablen sind weniger relevant. Dann wird mit *download()* der Download durchgeführt. Danach folgt die Ausgabe des Ergebnisses.



Je nachdem wie viele Datensätze gespeichert sind und wie viele Bilder noch heruntergeladen werden müssen, kann dies doch eine Weile dauern. Damit der Benutzer während dieser Zeit nicht denkt, die App sei abgestürzt, blende ich ein sich drehendes Rädchen ein. Dies wird ebenfalls von Angular Material bereitgestellt und sieht so aus, wie ein bei Android üblich. Dieser Prozess Indikator wird ausgeblendet, sobald der Scope *newsticker* seine Daten erhalten hat und bereit ist, diese auszugeben. Wenn die Daten vollständig geladen sind, wir ein Scope definiert. Ist dieser definiert, verschwindet der Indikator und das div-Element mit den Beiträgen wird angezeigt. Auf der Template-Seite newsticker.html sieht dies folgendermassen aus:

Abbildung 39 Android Prozess Indikator

```
1  <div flex layout="row" layout-align="center center" ng-
   hide="loaded">
2  <md-progress-circular class="md-accent" md-mode="indeter-
   minate" md-diameter="100"></md-progress-circular>
3  </div>
4
5  <div ng-show="loaded" ng-repeat="beitrag in newsticker |
   orderBy: '-datum'" layout="column">
6  <md-card>
7  <img ng-src="{{::beitrag.bild}}" class="md-card-image"
   alt="Beitragsbild">
8  <md-card-title>
9  <md-card-title-text>
10 <span class="md-headline">{{::beitrag.titel}}</span>
11 </md-card-title-text>
12 </md-card-title>
13 <md-card-content>
14 <div ng-bind-html="beitrag.beschreibung"></div>
15 </md-card-content>
```

```
16 <!--<md-card-actions layout="row" layout-align="end cen-
   ter">
17 <md-button class="md-icon-button" aria-label="Favorite">
18 <md-icon md-svg-icon="img/icons/favorite.svg"></md-icon>
19 </md-button>
20 <md-button class="md-icon-button" aria-label="Settings">
21 <md-icon md-svg-icon="img/icons/menu.svg"></md-icon>
22 </md-button>
23 <md-button class="md-icon-button" aria-label="Share">
24 <md-icon md-svg-icon="img/icons/share-arrow.svg"></md-
   icon>
25 </md-button>
26 </md-card-actions>-->
27 </md-card>
28 </div>
```

Mit ng-hide wird das Element ausgeblendet, wenn die Bedingung darin erfüllt ist, also wenn der Scope *loaded* definiert ist. Im Element selbst befindet sich Prozess Indikator. Dieser befindet sich im Modus *indeterminate*, also fortlaufend. Mit *md-diameter* wird die Grösse des Indikators definiert. Das div-Element, in dem die Beiträge aufgelistet werden, enthält das Attribut *ng-repeat*. Mit diesem Attribut lässt sich der Inhalt des Elements so lange wiederholen, bis es keine Einträge mehr in einem Scope hat:

```
1  <div ng-show="loaded" ng-repeat="beitrag in newsticker |
   orderBy: '-datum'" layout="column">
```

Jeder einzelne Datensatz kann mit *beitrag* angesprochen werden, und deren Unterelement ausgegeben werden. Zudem wird ein Angular Filter angewendet, der die Beiträge chronologisch ordnet. Der jüngste Beitrag erscheint also als erstes. Falls noch kein Beitrag gespeichert sein sollte, kann dem Benutzer eine Meldung ausgegeben werden:

```
1  <div ng-show="empty">
2  Keine Beitr&auml;ge gefunden!
3  </div>
```

Auch hier wird ein Scope *empty* definiert, wenn bei SQLite-Abfrage keine Datensätze resultieren. Im Controller sieht das dann so aus:

```
4  count = res.rows.length;
5
6  if(count > 0) {
7  // Speichern der Daten im Scope
8  } else {
9  $scope.loaded = true;
```

```
10 $scope.empty = true;
11 }
```

Zum Testen senden wir nun eine Payload von unserem Server zum Smartphone:

Abbildung 40 Newsticker Eingabemaske Test

In der Statusleiste des Smartphones erscheint nun die Benachrichtigung. Wenn man diese Nachricht antippt, öffnet sich die Air Zermatt App und es erscheint direkt die Seite Newsticker, mit dem neusten Beitrag:

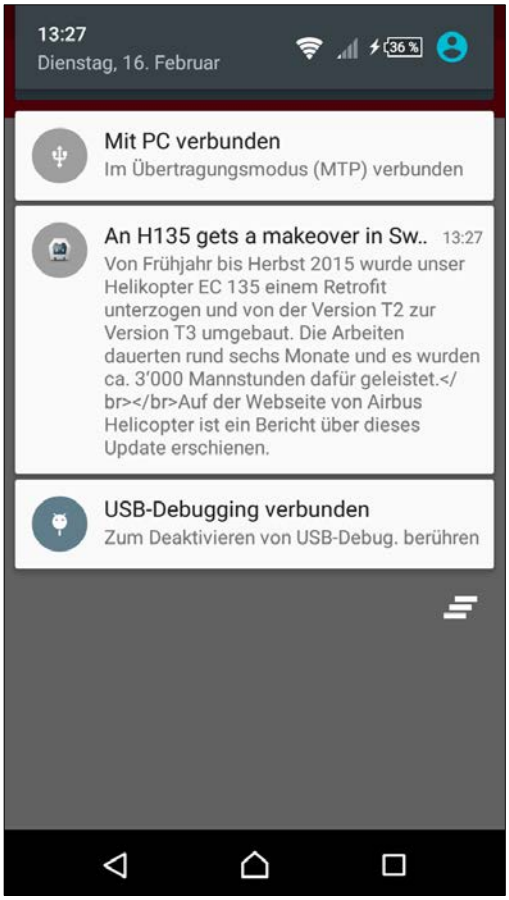


Abbildung 41 Benachrichtigung auf dem Smartphone

Abbildung 42 Beitrag auf der Seite Newsticker

6 SEITE ÜBERSICHT ÜBER DIE ANGEBOTE

Analog zur Seite Newsticker wird die Seite Angebote gestaltet. Die Seite erhält zwei Tabs. Eines mit dem Label Angebote, auf der allgemeine Informationen zu den Angeboten der Air Zermatt erscheinen. Das zweite Tab zeigt offene Rundflugangebote an. Bei Aufruf werden vorbereitete Daten aus der Datenbank gelesen, weshalb es für diese Seite wieder ein eigener Controller mit dem Namen *AngebotCtrl* gibt. Zu Beginn werden die notwendigen Daten aus der Datenbank ausgelesen und in das Scope *angebote* gespeichert:

```
1  var db = $cordovaSQLite.openDB({name: "database.db"});
2
3  $scope.angebote = [];
4
5  var query = "SELECT * FROM tbl_angebot, tbl_basis WHERE
   PK_basis = FK_basis";
6  $cordovaSQLite.execute(db, query, []).then(function(res)
   {
7  count = res.rows.length;
8  if(count > 0) {
9  console.log(count + ' ' + JSON.stringify(res));
10 for(i=0;i<count;i++) {
11
12 $scope.angebote.push({
13 titel: res.rows.item(i).angebot,
14 beschreibung: res.rows.item(i).beschreibung,
15 basis: res.rows.item(i).basis,
16 telefon: res.rows.item(i).telefon,
17 bild: cordova.file.dataDirectory + 'bilder/' +
   res.rows.item(i).bild
18 });
19
20 bild = res.rows.item(i).bild;
21
22 $cordovaFile.checkFile(cordova.file.dataDirectory +
   "/bilder/", bild)
23 .then(function (success) {
24 console.log('Bild existiert!');
25 }, function (error) {
26
27 var url = "http://www.air-zermatt.ch/app/bilder/" + bild;
28 var targetPath = cordova.file.dataDirectory + '/bilder/'
   + bild;
29 var trustHosts = true;
```

```
30 var options = {};
31
32 $cordovaFileTransfer.download(url, targetPath, options,
   trustHosts)
33 .then(function(result) {
34 console.log('Bilder heruntergeladen');
35 }, function(err) {
36 console.log('Bild konnte nicht heruntergeladen werden:' +
   JSON.stringify(err));
37 });
38
39 }, false);
40 };
41
42 console.log(JSON.stringify($scope.angebote));
43
44 } else {
45 $scope.loaded = true;
46 $scope.empty = true;
47 }
48
49
50 }, function (err) {
51 console.error(err);
52 }).then(function() {
53 $scope.loaded = true;
54 });
```

Dieser Code wurde bereits beim Aufrufen der Newsticker-Beiträge verwendet und kann deshalb kopiert und etwas abgeändert werden. Neben dem Speichern im Scope wird wieder überprüft, ob die dazugehörigen Bilder existieren. Existieren sie nicht, werden sie heruntergeladen. Weiter wird auch hier ein Prozess Indikator angezeigt, bis die Daten vollständig geladen sind. Daraufhin werden die Daten der offenen Rundflug-Angebote geladen und gespeichert.

Das Template-File der Seite Angebote sieht wie folgt aus:

```
1  <div ng-controller="AngebotCtrl">
2  <md-tabs md-stretch-tabs="auto" md-selected="selectedIn-
   dex" md-dynamic-height md-border-bottom>
3  <md-tab id="allgemein" class="md-accent" label="Allge-
   mein">
4  <md-content class="md-padding">
5  <h1>Tab One</h1>
6  </div>
```

```
7
8  </md-content>
9  </md-tab>
10 <md-tab id="rundfluge" class="md-accent" label="Rund-
   flüge">
11 <md-content class="md-padding">
12 <h1>Tab Two</h1>
13 </md-content>
14 </md-tab>
15 </md-tabs>
```

Abbildung 43 Seite Angebote Tab Allgemein

Abbildung 44 Seite Angebote Tab Rundflüge

Nun fügen wir in beiden Tabs die geladenen Inhalte ein. Als erstes der Prozess Indikator, der wieder verschwindet, wenn alle Daten geladen sind. Dann die einzelnen Card-Elemente mit Bild und Text und am Schluss der Hinweis, wenn keine Einträge gefunden wurden:

```
1  <div flex layout="row" layout-align="center center" ng-
   hide="loaded">
2  <md-progress-circular class="md-accent" md-mode="indeter-
   minate" md-diameter="100"></md-progress-circular>
```

```
3  </div>
4
5  <div ng-show="loaded" ng-repeat="angebot in angebote"
   layout="column">
6  <md-card>
7  <img ng-src="{{::angebot.bild}}" class="md-card-image"
   alt="Angebotsbild">
8  <md-card-title>
9  <md-card-title-text>
10 <span class="md-headline">{{::angebot.titel}}</span>
11 </md-card-title-text>
12 </md-card-title>
13 <md-card-content>
14 <div ng-bind-html="angebot.beschreibung"></div>
15 </md-card-content>
16 </md-card>
17 </div>
18
19 <div ng-show="empty">
20 Keine Beitr&auml;ge gefunden!
21 </div>
```

Diese Auflistung wird wiederum mit *ng-repeat* erreicht. In der Tabelle tbl_angebote befinden sich die notwendigen Informationen:

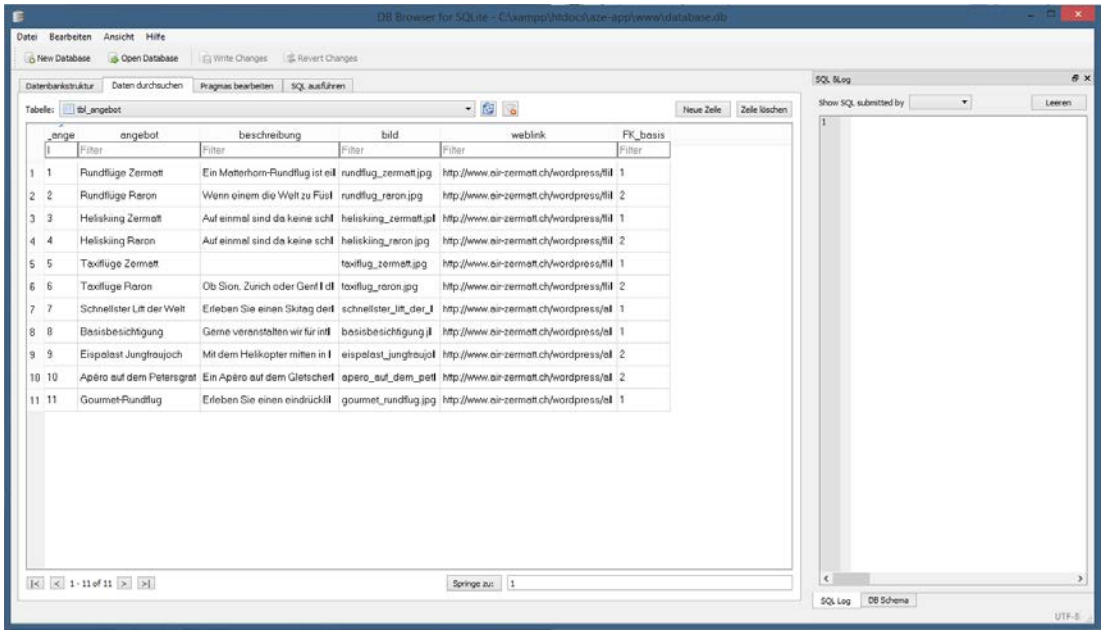


Abbildung 45 Tabelle tbl_angebote

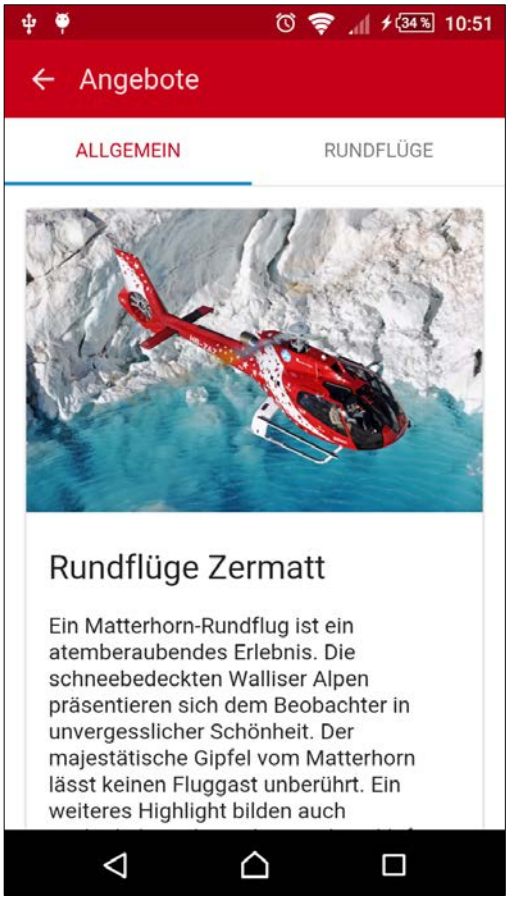


Abbildung 46 Seite Angebote mit Inhalt

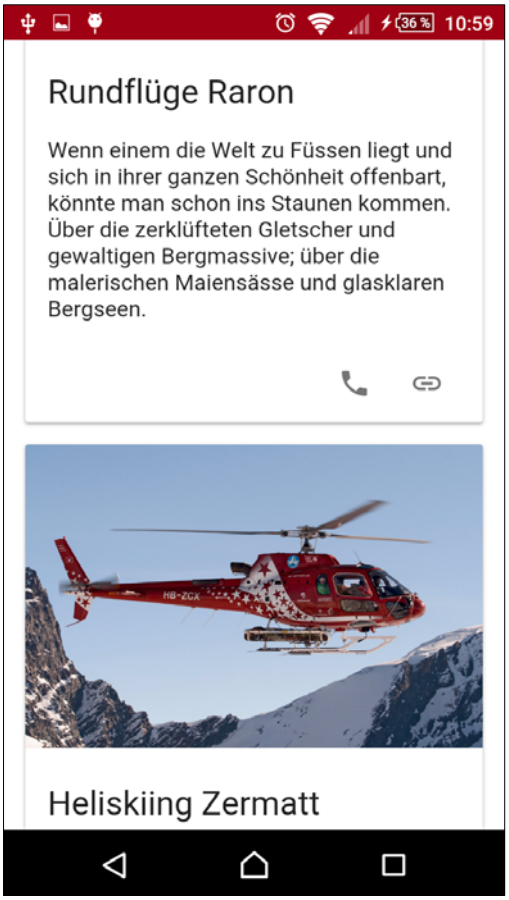


Abbildung 47 Seite Angebot mit Schaltflächen

Nun soll unterhalb des Beschreibungs-Textes noch ein kleiner Bereich folgen, mit zwei Schaltflächen. Mit der einen Schaltfläche kann ein direkt ein Anruf zur jeweiligen Basis tätigen. Mit der anderen gelangt er auf unsere Website, auf der es weitere Informationen zum Angebot gibt. Der dazugehörige Code kann zuunterst hinzugefügt werden:

```
1  <md-card-actions layout="row" layout-align="end center">
2  <md-button href="tel:{{angebot.telefon}}" class="md-icon-
   button" aria-label="Anfragen">
3  <md-icon md-svg-icon="img/icons/phone.svg"></md-icon>
4  </md-button>
5  <md-button href="{{angebot.weblink}}" class="md-icon-but-
   ton" aria-label="Link">
6  <md-icon md-svg-icon="img/icons/link.svg"></md-icon>
7  </md-button>
8  </md-card-actions>
9  </md-card>
10 </div>
```

Nun muss noch sichergestellt werden, dass die Bilder auf dem lokalen Gerät gespeichert sind. Dazu füge ich im File *app.js* weitere Befehle hinzu. Es werden alle Dateinamen der Bilder aus der Datenbank gelesen und bei jedem einzelnen überprüft, ob es vorhanden ist. Existiert es auf dem lokalen Gerät noch nicht, wird es vom Server heruntergeladen:

```
1  db.transaction(function(tx) {
2  tx.executeSql("SELECT bild FROM tbl_variante", [], func-
   tion(tx, res) {
3  count = res.rows.length;
4  console.log(count + ' Bilder: ' + JSON.stringify(res));
5  for(i=0; i < count; i++) {
6  var bild = res.rows.item(i).bild;
7  download_pic(bild);
8  }
9  });
10 }, function(err) {
11 console.error(err);
12 }, function(res) {
13 console.log("Bilder nicht geladen");
14 });
15
16 function download_pic(bild) {
17 $cordovaFile.checkFile(cordova.file.dataDirectory +
   "bilder/", bild)
18 .then(function (success) {
19 console.log('Bild existiert!');
20 }, function (error) {
21 var url = "http://www.air-zermatt.ch/app/bilder/" + bild;
22 var targetPath = cordova.file.dataDirectory + 'bilder/' +
   bild;
23 var trustHosts = true;
24 var options = {};
25 $cordovaFileTransfer.download(url, targetPath, options,
   trustHosts)
26 .then(function(result) {
27 console.log('Bilder heruntergeladen');
28 }, function(err) {
29 console.log('Bild konnte nicht heruntergeladen werden:' +
   JSON.stringify(err));
30 });
31 }, false);
32 }
```

Die ausgelesenen Daten werden in der Variable *res* gespeichert. Danach folgt eine for-Schlaufe, in der der Dateiname des einzelnen Bildes in einer Variable gespeichert wird. Diese Variable wird dann als Parameter an die Funktion *download_pic()* gesendet.

7 SEITE RETTUNGSKARTE

Die Seite Rettungskarte gestaltet sich etwas einfacher als die vorherigen Seiten. Diese Unterseite hat einen statischen Inhalt. Das heisst, die Informationen werden nicht aus einer Datenbank geladen, sondern befinden sich direkt auf der Template-Seite. Höchstens die Bild-Informationen werden in eine Datenbank-Tabelle gespeichert, damit auch hier sichergestellt werden kann, dass die Bilder auf dem Gerät vorhanden sind.

Konkret erstellen wir einen neuen Controller, der auf der Template-Seite implementiert wird. Dieser Controller nenne ich *RettungskarteCtrl*:

```
1  app.controller('RettungskarteCtrl', ['$scope',
   '$cordovaSQLite', '$cordovaFile', '$cordovaFileTransfer',
   function($scope, $cordovaSQLite, $cordovaFile,
   $cordovaFileTransfer) {
2  var db = $cordovaSQLite.openDB({name: "database.db"});
3  $scope.rettungskarte = [];
4  var query = "SELECT bild FROM tbl_rettungskarte";
5  $cordovaSQLite.execute(db, query, []).then(function(res)
   {
6  count = res.rows.length;
7
8  if(count > 0) {
9  console.log(count + ' ' + JSON.stringify(res));
10 for(i=0;i < count;i++) {
11
12 $scope.rettungskarte.push({
13 bild: cordova.file.dataDirectory + 'bilder/' +
   res.rows.item(i).bild
14 });
15
16 var bild = res.rows.item(i).bild;
17 console.log(bild);
18 download_pic(bild);
19 delete bild;
20 };
21
22 function download_pic(bild) {
23 $cordovaFile.checkFile(cordova.file.dataDirectory +
   "bilder/", bild)
24 .then(function (success) {
25 console.log('Bild existiert!');
26 }, function (error) {
```

```
27
28 var url = "http://www.air-zermatt.ch/app/bilder/" + bild;
29 var targetPath = cordova.file.dataDirectory + 'bilder/' +
   bild;
30 var trustHosts = true;
31 var options = {};
32
33 $cordovaFileTransfer.download(url, targetPath, options,
   trustHosts)
34 .then(function(result) {
35 console.log('Bilder heruntergeladen');
36 }, function(err) {
37 console.log('Bild konnte nicht heruntergeladen werden:' +
   JSON.stringify(err));
38 });
39
40 }, false);
41 }
42
43 }
44 }, function (err) {
45 console.error(err);
46 });
47
48 }]);
```

Analog zu den vorherigen Controller befindet sich in diesem Controller wiederum die Funktion, welche sicherstellt, dass sich die notwendigen Bilder auf dem Smartphone befinden.

Abbildung 48 Seite Rettungskarte

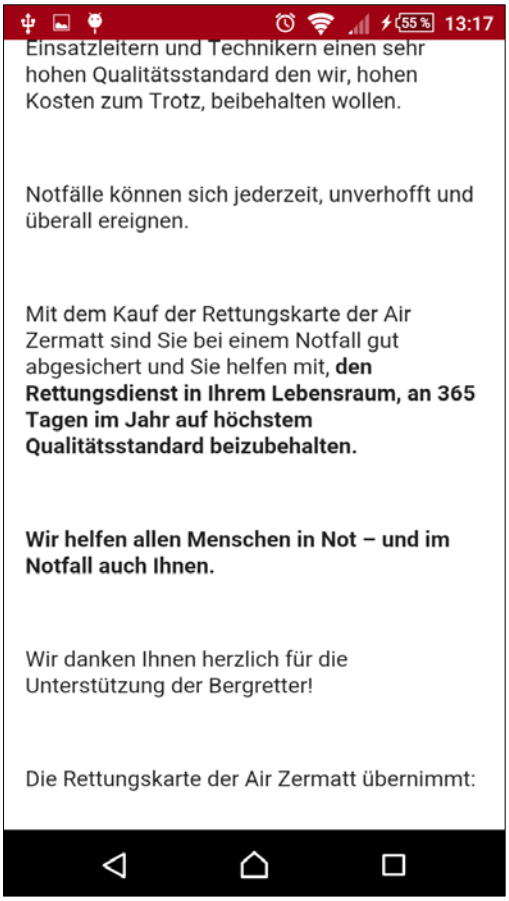


Abbildung 49 Seite Rettungskarte

8 SEITE PARTNER

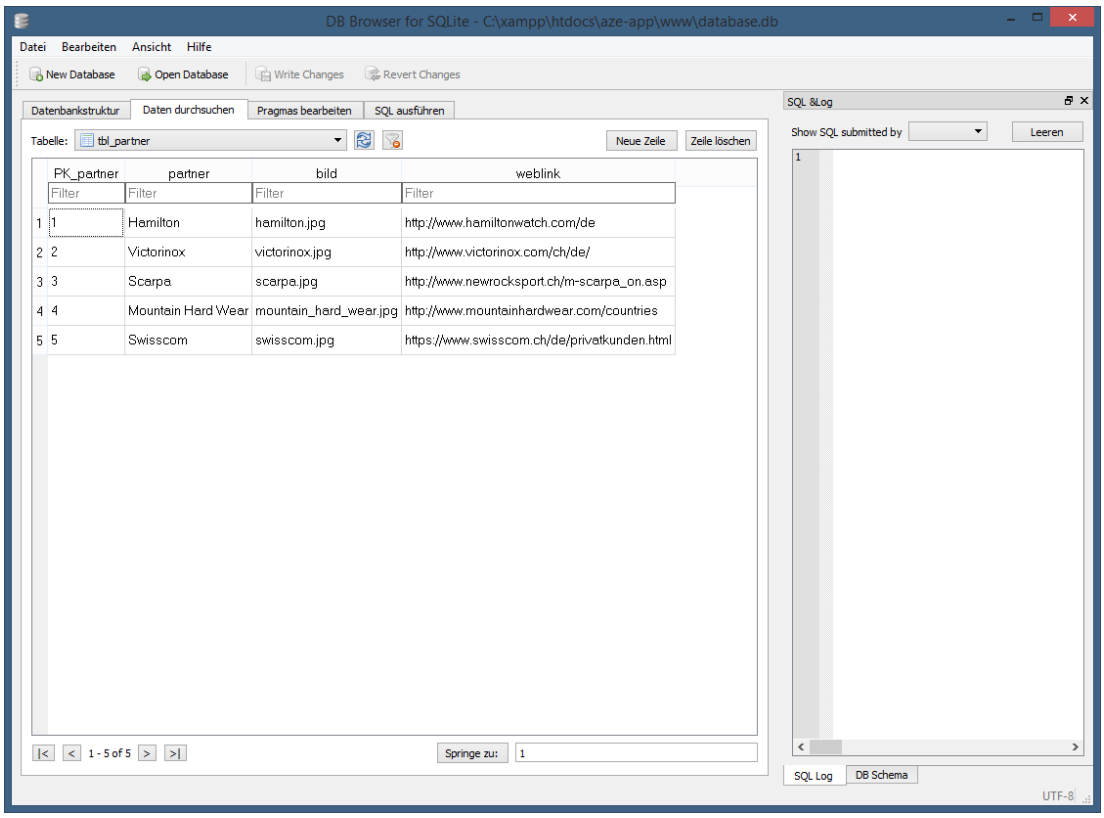


Abbildung 50 Tabelle tbl_partner

Als erstes bereite ich die Datenbank für die Seite Partner vor. Dann erstelle ich wiederum ein Controller mit dem Namen *PartnerCtrl*:

```
1  app.controller('PartnerCtrl', ['$scope', '$cordo-
   vaSQLite', '$cordovaFile', '$cordovaFileTransfer', func-
   tion($scope,  $cordovaSQLite, $cordovaFile, $cordovaFi-
   leTransfer) {
2  var db = $cordovaSQLite.openDB({name: "database.db"});
3  $scope.partner = [];
4
5  var query = "SELECT * FROM tbl_partner";
6  $cordovaSQLite.execute(db, query, []).then(function(res)
   {
7  count = res.rows.length;
8  if(count > 0) {
9  console.log(count + ' ' + JSON.stringify(res));
10 for(i=0;i < count;i++) {
11 $scope.partner.push({
12 partner: res.rows.item(i).partner,
```

```
13 weblink: res.rows.item(i).weblink,
14 bild: cordova.file.dataDirectory + 'bilder/' +
   res.rows.item(i).bild
15 });
16
17 var bild = res.rows.item(i).bild;
18
19 download_pic(bild);
20 delete bild;
21 };
22
23 function download_pic(bild) {
24 $cordovaFile.checkFile(cordova.file.dataDirectory +
   "bilder/", bild)
25 .then(function (success) {
26 console.log('Bild existiert!');
27 }, function (error) {
28
29 var url = "http://www.air-zermatt.ch/app/bilder/" + bild;
30 var targetPath = cordova.file.dataDirectory + 'bilder/' +
   bild;
31 var trustHosts = true;
32 var options = {};
33
34 $cordovaFileTransfer.download(url, targetPath, options,
   trustHosts)
35 .then(function(result) {
36 console.log('Bilder heruntergeladen');
37 }, function(err) {
38 console.log('Bild konnte nicht heruntergeladen werden:' +
   JSON.stringify(err));
39 });
40
41 }, false);
42 }
43
44 console.log(JSON.stringify($scope.partner));
45
46 }
47 }, function (err) {
48 console.error(err);
49 });
50
```

```
51 }]);
```

Auch hier wird wieder überprüft, ob die Bilder existieren und werden heruntergeladen, falls sie noch nicht gespeichert sind.

Folglich wird auf der Unterseite partner.html die in den Scope geladenen Daten ausgegeben:

```
1  <div ng-repeat="partner in partner">
2  <md-card style="min-height: 100px">
3  <a href="{{::partner.weblink}}">
4  <img ng-src="{{::partner.bild}}" class="md-card-image"
   alt="{{::patner.partner}}" />
5  </a>
6  </md-card>
7  </div>
```

Abbildung 51 Seite Partner

9 SEITE IMPRESSUM

Die Seite Impressum ist ebenfalls eine statische Seite, weshalb sie keinen Controller braucht. Auf dieser Seite wird der Herausgeber der App angegeben.

Abbildung 52 Seite Impressum

Wie man anhand der Template-Seite sieht, steht der Inhalt der Seite direkt im Code:

```
1  <md-content layout="column">
2  <h3>Herausgeber:</h3>
3  <p>Air Zermatt AG</br>
4  André Zenhäusern (Webmaster)</br>
5  Postfach 1</br>
6  CH-3942 Raron</p>
7  <p>Telefon: <a href="tel:+41279358686">+41 (0)27 935 86
   86</a></br>
8  E-Mail: <a href="mailto:webmaster@air-zermatt.ch">webmas-
   ter@air-zermatt.ch</a></p>
9  </md-content>
```

10ICON UND SPLASHSCREEN

10.1 Icon hinzufügen

Um unsere Applikation von anderen abzuheben, kann jetzt noch ein eigenes Icon bestimmt werden. Da das Logo der Air Zermatt prädestiniert für dieses Icon ist, habe ich mich für das entschieden. Bei Android kann dieses Logo für mehrere Auflösungen erstellt und hinterlegt werden. Google nennt die 4 Hauptauflösungen *ldpi, mdpi, hdpi, xhdpi*, wobei ein Icon bei der Auflösung ldpi eine Abmessung von *36 x 36 Px*, bei mdpi eine Auflösung von *48 x 48 Px*, bei hdpi *72 x 72 Px* und bei xhdpi *96 x 96 Px* haben sollte. Diese Icons werden im Projektverzeichnis gespeichert und müssen in der *config.xml*-Datei angegeben werden:

```
1  <platform name="android">
2  <icon src="www/res/android/ldpi/icon.png" density="ldpi"
   />
3  <icon src="www/res/android/mdpi/icon.png" density="mdpi"
   />
4  <icon src="www/res/android/hdpi/icon.png" density="hdpi"
   />
5  <icon src="www/res/android/xhdpi/icon.png" den-
   sity="xhdpi" />
6  </platform>
```

Wichtig ist hierbei, dass der Pfad nicht relativ zur *config*-Datei ist, sondern relativ zum Projektordner, weshalb das Verzeichnis *www* ebenfalls angegeben ist.

10.2 Splashscreen hinzufügen

Etwas schwieriger gestaltet sich das Hinzufügen eines Splashscreens. Dieser Splashscreen beim Starten der App angezeigt. Wiederum werden vier verschieden grosse Bilder verwendet, um die unterschiedlichen Auflösungen der Geräte zu unterstützten. Für die Auflösung ldpi wird ein Bild verwendet mit der Abmessung 320 x 480 Px, für mdpi wird dasselbe Bild verwendet, bei hdpi braucht es eine Abmessung von 480 x 720 Px und bei xhdpi 720 x 1080 Px. Diese Bilder müssen nun in einem Unterverzeichnis der Plattform Android gespeichert werden. Im Unterordner *platforms/android/res* befinden sich die verschiedenen Ordner, in denen das Betriebssystem dann nach den Icons und Splashscreen sucht. Wichtig hierbei ist der Dateiname. Die Bilder müssen zwingend den Namen *screen.png* haben, ansonsten werden sie nicht als Splashscreen erkannt. In jedem Ordner befindet sich also eine solche Datei:

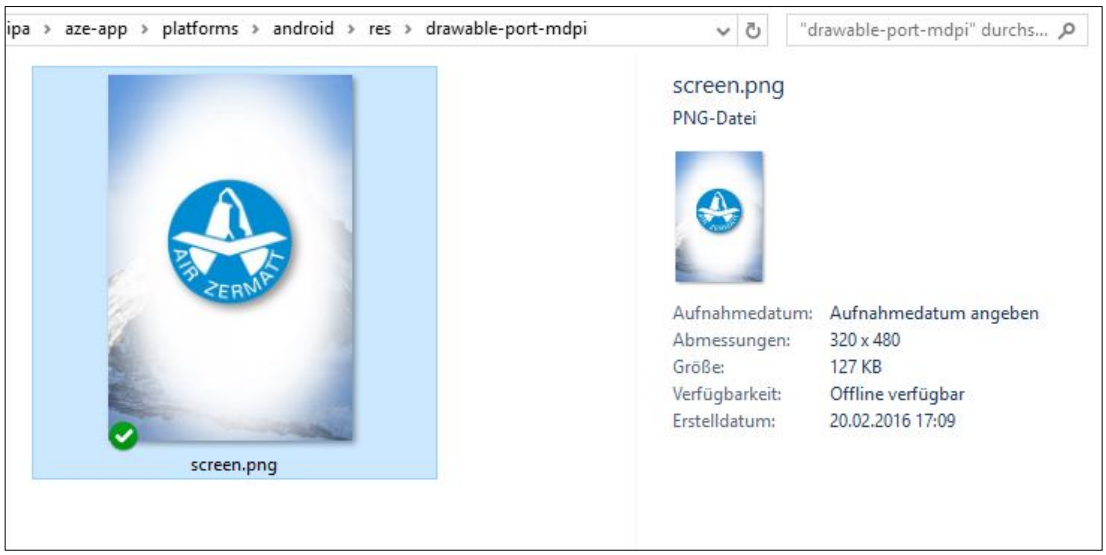


Abbildung 53 Ordnerstruktur Splashscreen

In der config-Datei können einige Einstellungen, bezüglich des Splashscreens unternommen werden:

```
7  <preference name="Splashscreen" value="screen" />
8  <preference name="AutoHideSplashScreen" value="true" />
9  <preference name="SplashscreenDelay" value="4500" />
10 <preference name="SplashMaintainAspectRatio" value="true"
   />
11 <preference name="FadeSplashScreenDuration" value="800"/>
```

Obwohl es die Einstellung gibt, dass der Splashscreen automatisch verschwindet, wenn die App vollständig geladen ist, kann man angeben, nach maximal wie vielen Millisekunden das Bild verschwinden soll. Mit *SplashMaintainAspectRatio* kann sichergestellt werden, dass das Bild nicht gestaucht wird, sondern höchstens abgeschnitten wird, wenn es nicht dieselben Abmessungen hat wie das Display des Gerätes. Mit dem runden Logo der Air Zermatt macht diese Einstellung sicherlich Sinn. Zusätzlich kann noch ein *FadeOut* definiert werden, was ein weicher Übergang zur Hautseite bedeutet.

Testet man nun die App nochmals auf dem Gerät, wird der Splashscreen angezeigt.

Abbildung 54 Splashscreen beim Starten der App

11 ENDE DER IPA

Bis zu diesem Punkt wurde die Dokumentation anlässlich der IPA geschrieben. Was nun folgt wurde nach abgeschlossener IPA-Arbeit ergänzt.

12AUFBESSERUNG DES BACK-ENDS

Da das Back-End während der IPA nur oberflächlich gestaltet wurde, wird dieses zuerst etwas aufgebessert, damit der Umgang damit übersichtlicher und einfacher wird. Zudem wird die Sicherheit erhöht, damit auch nur jene Zugriff auf das Back-End haben, die dürfen.

12.1 MySQLi anstatt MySQL

Als erstes werden die veralteten MySQL-Befehle durch die etwas sichereren MySQLi-Befehle ersetzt. Grosse Unterschiede gibt es zwischen diesen beiden Sprachen nicht. Einige kleinere Anpassungen an der Syntax sind aber nötig. Meistens muss der mysql-Präfix mit einem *i* ergänzt werden und als ersten Parameter wird meistens die Verbindung mit der Datenbank erfordert. Eine einfache MySQL-Abfrage:

```
1  mysql_query("SELECT * FROM tbl_table");
```

sieht bei MySQLi so aus:

```
1  mysqli_query($con, "SELECT * FROM tbl_table");
```

wobei in der Variable *$con* die Verbindung zur Datenbank gespeichert ist:

```
1  $host = "localhost";
2  $user = "[Benutzername]";
3  $password = "[Passwort]";
4  $db = "[Datenbank]";
5
6  $con = mysqli_connect($host, $user, $password, $db);
```

Eine ausführliche Auflistung der Befehle gibt es auf der Website von **w3schools**.

12.2 Verbesserung der Sicherheit beim Login

Während der IPA wurde nicht ein sehr sicherer Login-Bereich erstellt. Dieser wurde nach abgeschlossenem Projekt nochmals überarbeitet. Grundsätzlich wurde mit einer PHP-Funktion gearbeitet, die PHP seit Version 5.5 anbietet. Diese **Password Hashing Funktionen** bieten eine einfache API an, mit dem relativ simple Passwörter verifiziert und damit ein Passwortschutz erstellt werden können. Einen sehr nützlichen Beitrag über die sichere Speicherung von Passwörter und deren Hashing gibt es hier: **https://crackstation.net/hashing-security.htm**

12.2.1 Passwörter hashen

Mit dem Befehl *password_hash()* kann ein Passwort gehasht werden. Der folgende Befehl gibt ein Zeichenkette mit 60 Zeichen aus:

```
1  echo password_hash("MeinPW123", PASSWORD_BCRYPT);
```

Die Ausgabe kann zum Beispiel so aussehen:

```
2  $2y$10$ikk/IiZuN56h7OT-
   qHr6JJO3xLm4zZhiZx5iFkw3aIs1PuS4MsDhza
```

Interessant wird es erst, wenn man den Befehl nochmals ausführt. Es ergibt sich eine total andere Zeichenkette. Dies deshalb, weil nicht nur das Passwort selbst verschlüsselt wird, sondern zusätzlich zum Passwort noch ein automatisch generierter Salt. Dieser Salt wird bei jedem Aufruf von neuem generiert. Der Vorteil dieser Methode ist, dass diese Zeichenkette nun in der Datenbank gespeichert werden kann.

Als erster Parameter erwartet die Funktion das Passwort, das vom Benutzer eingegeben wurde. Der zweite Parameter definiert den Algorithmus mit dem das Passwort verschlüsselt wird. Als dritten, optionalen Parameter kann ein Array angeben werden, in dem der Cost angegeben werden kann. Zudem könnte man eine eigene Funktion zum Erstellen des Salts angeben, davon wird aber abgeraten!

```
1  $option = ['cost' => 13];
2  echo password_hash($password, PASSWORD_BCRYPT, $option);
```

Mithilfe von der Cost-Angabe werden der Zeitaufwand und die aufzuwendende Power zum Hashen erhöht. Das mag paradox klingen, doch so kann die Gefahr vor Brut Force-Attacken verringert werden, weil ein allfälliger Bot bei jedem Versuch mehr Zeit braucht.

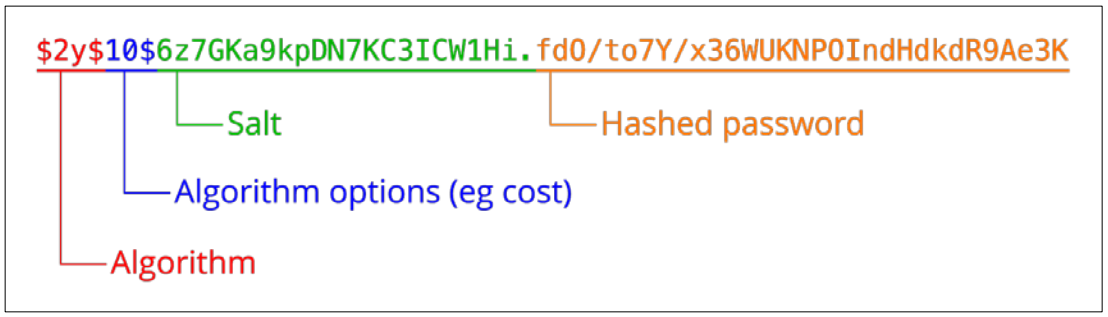


Abbildung 55 Aufteilung eines gehashten Passworts

12.2.2 Passwort vergleichen

Um nun das eingegebene Passwort mit demjenigen in der Datenbank zu vergleichen, kann mit die Funktion *password_verify()* verwenden:

```
1  password_verify($password, $hash)
```

Als erster Parameter wird das eingegebene Passwort verlangt. Der zweite Wert ist das gehashte Passwort aus der Datenbank, welches vorgängig ausgelesen wurde. Stimmt das Passwort überein, gibt die Funktion ein *true* zurück, andernfalls *false*.

12.2.3 Konkrete Anwendung im Backend

Konkret wird beim Login-Prozess als erstes geprüft, ob alle Felder ausgefühlt wurden. Trifft dies zu, wird eine Abfrage ausgeführt, die das gehashte Passwort des angegebenen Benutzers herausgibt. Existiert zum Benutzer einen Datensatz wird das Passwort schliesslich in einer if-Klausel auf seine Richtigkeit überprüft. Erst wenn das Passwort stimmt, wird eine Session eröffnet und die anderen Informationen darin gespeichert.

Auf jeder Seite des Backends wird zuerst überprüft ob eine PHP-Session läuft und ob alle relevanten Informationen darin gespeichert sind. Wenn das nicht zutrifft wird zur Login-Seite weitergeleitet und der Benutzer muss sein Passwort angeben.

12.3 Benutzer und deren Berechtigungen

Ein weiter Punkt, der verändert wurde ist das System der Berechtigung. Jeder Mitarbeiter der befugt ist, auf das Backend zugreifen zu können, hat seinen eigenen Benutzer. Einerseits kann dadurch die Berechtigung verwaltet werden. Andererseits kann nachvollzogen werden, welche Person welche Änderung an den Daten gemacht hat. D.h. bei jeder Änderung eines Datensatzes wird der Benutzer, welcher die Änderung vollzogen hat in der Datenbank vermerkt.

In der Tabelle tbl_benutzer ist neben dem gehashten Passwort des Benutzers zusätzlich die Information gespeichert, ob er nur Zugriff auf die Seite Rundflug-Push hat oder zusätzlich auch auf die Seite Newsticker. Beim Login werden diese beiden Spalten auch abgefragt. Die beiden erhaltenen Zahlen werden zusammen addiert. Wenn die Zahl 1 resultiert, hat er nur Zugriff auf die Seite Rundflug-Push, wenn aber die Zahl 2 resultiert, kann er auch auf der Seite Newsticker Änderungen vollziehen. Folglich wird auf diesen beiden Seiten auch die Rolle des Benutzers kontrolliert.

Diese Methode sollte die Verwaltung der Benutzer und deren Berechtigungen sehr einfach machen. Auch das Hinzufügen und Löschen eines Benutzers ist dadurch trivial.

12.4 Kleine Änderungen an den Scripten

Neben diesen grösseren Änderungen wurden auch noch kleine Änderungen an diversen Scripten durchgeführt. Vor allem Vereinfachungen am Code wurden gemacht, damit dieser einfacher zu bearbeiten ist. Diese Änderungen wurden sowohl bei der Applikation als auch serverseitig gemacht.

Bei der App wurden wiederkehrende Funktionen, die auf mehreren Seiten ausgeführt werden, vereinheitlicht und in den MainCtrl eingefügt. Von den anderen Controllern wird dann auf die Funktion im MainCtrl verwiesen. Teilweise auch mit Parameter, die von der Funktion verarbeitet werden.

Auf dem Server wurden redundante Funktionen ebenfalls vereinheitlicht. Solche Snippets wurde im Unterordner *templates* abgelegt und werden an den betreffenden Stellen via *require* in das Script geladen.

Dies macht zwar das Arbeiten an spezifischen Scripts etwas unübersichtlicher, allerdings lassen sich so Änderungen in den Snippets schnell und global durchführen.

12.5 Annahme und Verarbeitung mehrerer Bilder

Weil wir sicherlich mehrere Bilder in einem Newsticker-Beitrag einfügen möchten, muss der Workflow für die Bilder auch angepasst werden. Im Back-End kann man mehrere Bilder gleichzeitig hochladen. Da man nie weiss, wie viele Bilder hochgeladen werden, muss mit Schlaufen gearbeitet werden.

12.5.1 Bilder samt Vorschaubild speichern

Im Template-File *bilder_speichern.php* werden die Bilder aus der POST-Übergabe validiert und in zwei Versionen gespeichert. Dies passiert in einer Schlaufe. Zudem wird der Datenname in einem Array gespeichert, das später verwendet wird. Wieder in der index.php wird der Beitrag in der Datenbank gespeichert und die ID des Beitrags wird in einer Variable gespeichert. Diese Variable wird in der nächsten Schlaufe verwendet, um in der Beziehungsmengen-Tabelle *tbl_news_bild* die Bildinformationen zu speichern:

```
1  foreach($id_bild as $bild) {
2  if($i++ === $count) {
3  $query_text .= "('$id', '$bild');";
4  } else {
5  $query_text .= "('$id', '$bild'),";
6  }
7  }
8
9  $query = mysqli_query($con, $query_text);
```

In dieser foreach-Schlaufe wird der Text der Abfrage vorbereitet. Dieser Text wird in der Variable *$query_text* zwischengespeichert und schlussendlich ausgeführt.

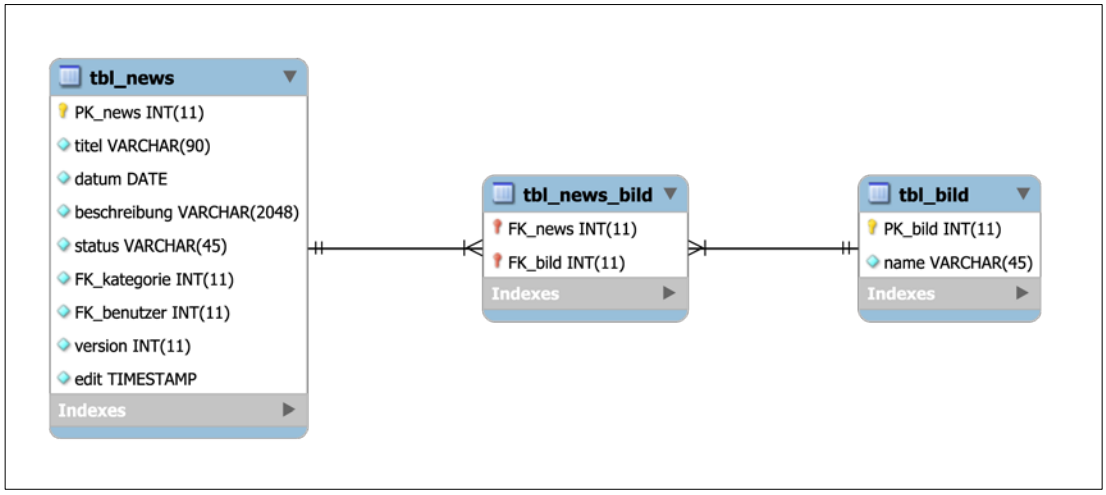


Abbildung 56 Datenmodell der Relation zwischen tbl_news und tbl_bild

Das obige Datenmodell besteht sowohl serverseitig als auch auf der App. Dies ist deshalb nötig, weil wir zwischen der Tabelle *tbl_news* und der Tabelle *tbl_bild* eine Mehrfach-zu-Merfach-Verbindung haben. Es können also kein, einer oder mehrere Beiträge existieren, die ein, oder mehrere Bild(er) enthalten. In der Tabelle tbl_news_bild werden jeweils die Fremdschlüssel gespeichert der beiden anderen Tabellen gespeichert.

12.5.2 Alle Bildinformationen im Payload

Nun müssen die Bildinformationen via Payload an die Smartphones geladen werden. Dazu werden die Dateinamen der Bilder, die zum Beitrag gehören, in ein Array geladen:

```
1  $query = mysqli_query($con, "SELECT name FROM tbl_bild,
   tbl_news_bild WHERE FK_bild = PK_bild AND FK_news =
   '$id'");
2  $pictures = array();
3  while($row = mysqli_fetch_array($query)) {
4  $pictures[] = $row["name"];
5  }
```

Das Array $pictures wird nun in den Payload übernommen und versendet.

12.5.3 Annahme der Bilder durch das Smartphone

Als nächstes muss Cordova beim erhalten des Pushs die Bilder verarbeiten. Hier passiert folgendes. Als erstes werden der Text des Beitrages und weitere Informationen durch einen *INSERT OR REPLACE*-Befehl eingefügt. *INSERT OR REPLACE* ist ein spezieller Befehl, den es bei SQLite gibt. Dabei wird darauf geachtet, ob in der Tabellen ein *UNIQUE*-Attribut auf einer Spalte liegt. Dann darf nämlich kein zweiter Datensatz mit demselben Wert eingefügt werden. Da wir in der Tabelle *tbl_news* dieses Attribut der Spalte FK_news zugewiesen haben, wird

beim Einfügen also überprüft, ob bereits ein Datensatz mit derselben ID existiert. Ist das der Fall wird dieser Datensatz ersetzt. Falls nicht wird er hinzugefügt.

```
1  tx.executeSql("INSERT OR REPLACE INTO tbl_newsticker(ti-
   tel, kategorie, datum, beschreibung, FK_newsticker) VA-
   LUES(?,?,?,?,?)", [titel, kategorie, datum, beschreibung,
   id], function() {},
2  function(error) {
3  console.log(error);
4  });
```

Danach wird jeder allfällige Eintrag in der Tabelle *tbl_news_bild* mit derselben ID, wie aktuelle gelöscht. Dies ist aber nur der Fall, wenn es sich um ein Update des Beitrags handelt.

```
1  tx.executeSql("DELETE FROM tbl_news_bild WHERE FK_news =
   ?", [id], function() {},
2  function(error) {
3  console.log(error);
4  });
```

Nun kommt der etwas schwierigere Teil. Nun wird wiederum in einer Schlaufe jeder einzelne Name der Bilder aufgerufen und überprüft, ob dieses Bild bereits auf dem Smartphone existiert. Dann wird der Name in die Tabelle *tbl_bild* gespeichert, bzw. ignoriert, wenn dasselbe Bild bereits existiert. Anschliessend wird die ID des gerade eingefügt Bild ausgegeben und in der Variable *bild_id* gespeichert. Dies deshalb, weil wir diese ID in die Tabelle *tbl_news_bild* einfügen müssen. Mit einen *INSERT*-Befehl wird dieser zusammen mit der ID des Beitrags hinzugefügt.

```
1  var count = bild.length;
2  for(i=0;i<count;i++) {
3  var single_pic = bild[i];
4  $scope.check_pic(single_pic);
5
6  tx.executeSql("INSERT OR IGNORE INTO tbl_bild(bild) VAL-
   UES(?)", [single_pic]);
7
8  tx.executeSql("SELECT PK_bild FROM tbl_bild WHERE bild =
   ?", [single_pic], function(tx, result) {
9  var bild_id = result.rows.item(0).PK_bild;
10 tx.executeSql("INSERT INTO tbl_news_bild(FK_news,
   FK_bild) VALUES(?,?)", [id, bild_id], function() {},
11 function(error) {
12 console.log(error);
13 });
```

```
14
15 }, function(err) {
16 console.error("Neues Bild wurde nicht verknüpft: " +
   JSON.stringify(err));
17 }, function(res) {
18 console.log(single_pic + " wurde mit Beitrag ver-
   knüpft!");
19 });
20 }
```

12.5.4 Ausgabe der Beiträge auf der Seite Newsticker

Was jetzt noch fehlt, ist die Anpassung bei der Ausgabe der Beiträge auf der Seite Newsticker. Dazu passen wir die Abfrage etwas an:

```
1  tx.executeSql("SELECT titel, kategorie, datum, beschrei-
   bung, bild, min(FK_bild) FROM tbl_newsticker, tbl_bild,
   tbl_news_bild WHERE FK_bild = PK_bild AND FK_newsticker =
   FK_news GROUP BY FK_news", [], function(tx, res) {
2
3  count = res.rows.length;
4
5  if(count > 0) {
6
7  // Speichern der geladenen Daten im Scope
8  for(i=0;i<count;i++) {
9
10 $scope.newsticker.push({
11 titel: res.rows.item(i).titel,
12 kategorie: res.rows.item(i).kategorie,
13 datum: res.rows.item(i).datum,
14 beschreibung: res.rows.item(i).beschreibung,
15 bild: cordova.file.dataDirectory + 'bilder/' +
   res.rows.item(i).bild
16 });
17 };
18
19 console.log(JSON.stringify($scope.newsticker));
20
21 } else {
22 $scope.loaded = true;
23 $scope.empty = true;
24 }
25
```

```
26 },
27 function(error) {
28 console.error(error);
29 });
```

In dieser Abfrage gibt es zwei wichtige Punkte. Erstens der Zusatz *GROUP BY FK_news*, der dafür sorgt, dass nur ein Bild des Beitrages geladen wird. Zweitens der Zusatz min(FK_bild), der die tiefste ID und damit das erste Bild auswählt, das mit der Beitrag-ID verknüpft ist.

12.6 Synchronisierung und Versionierung von Beiträgen

Da ein Nutzer auf seinem Smartphone die Einstellung zur Push-Benachrichtigung abschalten kann, muss er trotzdem die neusten Beiträge im Newsticker erhalten. Dazu muss das Gerät beim Aufrufen der Seite unseren AZE-Server anfragen, ob es neue, geänderte oder gelöschte Beiträge gibt.

12.6.1 Laden der Versionsnummer vom Server

Als erstes möchten wir die Versionsnummer von jedem Beitrag, der auf dem Server gespeichert ist, laden. Dazu erstelle ich eine Funktion, die eine POST-Abfrage beim Server durchführt und die jeweiligen IDs und die dazugehörigen Versionen ladet:

```
1  $rootScope.$on('sync_posts', function(event, data) {
2  $http.post('http://www.air-zermatt.ch/app/updater.php',
   data[0]).then(function(response) {
3  count = response.data.length;
4
5  for(i=0;i<count;i++) {
6  var post = response.data[i]['PK_' + data];
7  $scope.compare_post(data, post, serversion);
8  };
9
10 }, function (response) {
11 console.log("Etwas hat nicht geklappt!");
12 });
13 });
```

Über den Parameter data erhalten wir die Kategorie der Beiträge (z.B.: news oder rundflug). Die erhaltenen Informationen werden später weitergeleitet. Im File updater.php passiert nun folgendes:

```
1  $request = file_get_contents("php://input");
2
3  require("connection.php");
4  $tabelle = mysqli_real_escape_string($con, $request);
```

```
5
6  $query = mysqli_query($con, "SELECT PK_$tabelle, version
   FROM tbl_$tabelle WHERE status = 'publish'");
7  $response = array();
8
9  $i = 0;
10 while($row = mysqli_fetch_array($query)) {
11 $response[$i]["PK_$tabelle"] = $row["PK_$tabelle"];
12 $response[$i]["version"] = $row["version"];
13 $i++;
14 }
15
16 echo json_encode($response);
```

Neben der Überprüfung der erhaltenen Daten und Request Headers wird mit einer Abfrage die IDs und Versionen aus der spezifischen Tabelle geladen, danach im einem Array gespeichert und als JSON-Objekt codiert. Dieses JSON-Objekt wird als Antwort zurück an das Smartphone gesendet.

12.6.2 Vergleich der Informationen auf dem lokalen Gerät

An die Funktion *$scope.compare_post()* werden 3 Parameter gesendet. Der erste ist *data*, welche die Kategorie der Beiträge beinhaltet. Der zweite heisst *post* und enthält die ID des Beitrags. Als drittes wird der Parameter *serversion* (Abk. von Server-Version) übermittelt in welchem die Versionsnummer des Beitrags auf dem Server gespeichert ist.

Serverseitig wird die Versionsnummer um eins erhöht, wenn der Beitrag bearbeitet wird. Ein neuer Beitrag hat standardmässig die Versionsnummer 1. Wenn ein Beitrag gelöscht wird, erhält der Beitrag die Version 0.

Anhand dieser Informationen kann man die Beiträge vergleichen und dementsprechend handeln. Als erstes überprüfen wir, ob ein Beitrag mit der Übermittelten ID in der lokalen Datenbank vorhanden ist. Wenn das nicht der Fall ist und die Versionsnummer des Beitrags auf dem AZE-Server höher als Null ist, wird der Beitrag heruntergeladen. Ist ein Beitrag mit dieser ID aber vorhanden beginnt eine if-else-Klausel. Wenn der Versionsnummer (serversion) gleich Null ist, so wird der Beitrag gelöscht. Wenn es sich um einen Beitrag mit der Kategorie News handelt, so wird zusätzlich die Verknüpfung zwischen dem Beitrag und den Bilder in der Tabelle *tbl_news_bild* gelöscht. Wenn die lokale Version mit der serversion übereinstimmt, so wird nichts unternommen. Weichen die beiden Versionen aber ab, wird der Beitrag vom Server geladen und überschrieben. Diese ganze Funktion sieht dann folgendermassen aus:

```
1  $scope.compare_post = function(data, post, serversion) {
2  db.transaction(function(tx) {
```

```
3  tx.executeSql("SELECT FK_" + data + ", version FROM tbl_"
   + data + " WHERE FK_" + data + " = ?", [post], func-
   tion(tx, res) {
4  if(res.rows.length == 1) {
5  if(serversion == 0) {
6  tx.executeSql("DELETE FROM tbl_" + data + " WHERE FK_" +
   data + " = ?", [post], function() {},
7  function(error) {
8  console.log(JSON.stringify(error));
9  });
10 if(data == 'news') {
11 tx.executeSql("DELETE FROM tbl_news_bild WHERE FK_news =
   ?", [post], function() {},
12 function(error) {
13 console.log(JSON.stringify(error));
14 });
15 }
16 console.log("Beitrag gelöscht!");
17 } else if(serversion == res.rows.item(0).version) {
18 console.log("Version ist identisch");
19 } else {
20 var options = {
21 'data': data[0],
22 'post_id': post
23 };
24
25 $http.post('http://www.air-zermatt.ch/app/posts.php', op-
   tions).then(function(response) {
26 var news = response.data;
27 if(data == 'news') {
28 $scope.save_news(news);
29 } else if(data == 'rundflug') {
30 $scope.save_rundflug(news);
31 }
32 });
33 console.log("Version weicht ab!");
34 }
35
36 } else if(res.rows.length == 0 && serversion > 0) {
37
38 var options = {
39 'data': data[0],
40 'post_id': post
```

```
41 };
42
43 $http.post('http://www.air-zermatt.ch/app/posts.php', op-
   tions).then(function(response) {
44 var news = response.data;
45
46 if(data == 'news') {
47 $scope.save_news(news);
48 } else if(data == 'rundflug') {
49 $scope.save_rundflug(news);
50 }
51 });
52 }
53
54 },
55 function(error) {
56 console.log(JSON.stringify(error));
57 });
58 }, function(error) {
59 console.log("Error: " + error);
60 }, function() {
61 // console.log("Erfolgreich!");
62 });
63 }
```

Diese Funktion kann nun durch diesen Befehl jederzeit ausgeführt werden:

```
1  $rootScope.$broadcast('sync_posts', ['news']);
```

Wobei man in der eckigen Klammern die Kategorie des Beitrags übermittelt.

12.7 WYSIWYG-Editor

Des Weiteren habe ich einen WYSIWYG-Editor hinzugefügt, der das Formatieren der Nachrichten für die Mitarbeiter vereinfachen soll. Ich entschied mich für das 3rd-Party Plugin **tinymce**. Dieser bietet viele Funktionen und ist einfach zu Implementieren. Zudem ist er vollkommen kostenlos erhältlich. Eine ausführliche Dokumentation für dieses Plugin findet man hier: **https://www.tinymce.com/docs/**

Via JavaScript lässt sich der Editor beliebig konfigurieren:

```
1  tinymce.init({
2  selector: '#newsticker_beschreibung',
3  language_url: '../js/tinymce_de.js',
4  plugins: 'link preview table autolink code contextmenu
   fullscreen',
5  menu: {
6  edit: {title: 'Edit', items: 'undo redo | cut copy paste
   | selectall'},
7  format: {title: 'Format', items: 'bold italic underline
   strikethrough superscript subscript | removeformat'},
8  table: {title: 'Table', items: 'inserttable tableprops
   deletetable | cell row column'},
9  tools: {title: 'Tools', items: 'code'}
10 },
11 default_link_target: "_blank",
12 table_class_list: [
13 {title: 'None', value: ''},
14 {title: 'Erste Zeile', value: 'first_row'},
15 {title: 'Erste Spalte', value: 'first_column'}
16 ],
17 toolbar: 'paste | undo redo | fontsizeselect | bold
   italic underline | alignleft aligncenter alignright |
   numlist bullist | outdent indent | link unlink | full-
   screen preview code',
18 fontsize_formats: '8pt 10pt 12pt 14pt 18pt 24pt 36pt',
19 min_height: 200,
20
21 });
```

Dieser Editor ist vorläufig nur für die Newsticker-Beitrage vorgesehen.

12.8 DataTables für interaktive Kontrolle über Tabellen

Damit die ganzen Daten in den Tabellen des Back-Ends nicht in ein Chaos ausufern, möchte ich einige Optionen zu den Tabellen, die der Benutzer interaktiv benutzen kann. So möchte ich die Tabellen durchsuchen können, sortieren und vor allem auf mehrere Seiten aufteilen. Zu Beginn des Projekts wird die Menge an Informationen wohl noch nicht ein Problem sein. Doch nach einigen Monaten sammeln sich sicherlich viele Informationen an. Solche Einstellungen, zur besseren Übersicht der Tabelle, bietet das Plugin **DataTables** an. Auch dieses Plugin ist sehr mächtig und hat einige gute Funktionen im Angebot. Zur Verwendung dieses Plugins wird jQuery benötigt. Nachfolgend werden die Installation und einzelne Funktionen erklärt.

12.8.1 Installation des Plugins

Wie bereits erwähnt, ist das Plugin auf dem JavaScript-Framework jQuery aufgebaut. Daher binde ich eine aktuelle Version von jQuery auf die betreffende Seite ein:

```
1  <!-- Laden des 3rd-Party Plugin jQuery -->
2  <script type="text/javascript"
   src='../js/jquery.js'></script>
```

Zudem kann ich auf der Website von DataTables den **Download Builder** verwenden, um meine benötigten Datei herunterzuladen. Neben dem Plugin selbst kann auch eine aktuelle Version von jQuery und weitere Extensions heruntergeladen werden. Dann werden auch diese Datei in die betreffende Seite implementiert:

```
1  <!-- Laden des 3rd-Party Plugin Datatables -->
2  <script type="text/javascript"
   src='../js/datatables.min.js'></script>
```

Zusätzlich könnte man auch noch eine CSS-Datei hinzufügen. Da ich die Tabelle aber schon gestaltet habe, verzichte ich auf diese Datei.

Nun steht das Plugin für den Gebrauch bereit. Nun kann in einem separaten JS-Script die erste Tabelle und somit das Plugin initialisiert werden:

```
1  $(document).ready(function(){
2  $('#myTable').DataTable();
3  });
```

Anstelle von *#myTable* fügt man die ID der gewünschten Tabelle ein.

12.8.2 Konfiguration der Tabelle

Für unser Back-End habe ich nun einige nützliche Funktion konfiguriert. Am wichtigsten schien mir das Aufteilen der Datensätze auf mehrere Seiten. Dazu bietet DataTables die **Page lenth Option** an. Die Konfigurationen werden als Objekt definiert. Dazu ändern wir im JS-Script den Init-Befehl für das Plugin:

```
1  $('#posts_news').DataTable({
2  "lengthMenu": [[10, 20, 50, -1], [10, 20, 50, "alle"]],
3  });
```

Es kann ein Array mit der gewünschten Anzahl Einträge pro Seite definiert werden. In einem zweiten Array kann optional der anzuzeigende Name im Dropdown-Menü angegeben werden. Der Wert *-1* bedeutet, das alle Einträge auf derselben Seite angezeigt werden.

Natürlich möchten wir die englischen Strings übersetzen. Auch dazu bietet das Plugin eine **Konfigurationsmöglichkeit**. Es kann ein weiterer Key mit dem Name "language" hinzugefügt werden:

```
1  "language": {
2  "decimal": ".",
3  "emptyTable": "Keine Daten verf&uuml;gbar",
4  "info": "Zeige _START_ bis _END_ von _TOTAL_
   Eintr&auml;gen",
5  "infoEmpty": "Zeige keine Eintr&auml;ge",
6  "infoFiltered": "(gefiltert von Total _MAX_
   Eintr&auml;gen)",
7  "infoPostFix": "",
8  "thousands": "'",
9  "lengthMenu": "Zeige _MENU_ Eintr&auml;ge",
10 "loadingRecords": "Laden...",
11 "processing": "Laden...",
12 "search": "Suchen: ",
13 "zeroRecords": "Keine &uuml;bereinstimmenden
   Eintr&auml;ge gefunden",
14 "paginate": {
15 "first": "Erste",
16 "last": "Letzte",
17 "next": "&gt;",
18 "previous": "&lt;"
19 }
20 },
```

Weil die Einträge beim Generieren der Tabelle nach dem Datum sortiert werden, kann man dies auch mit dem Plugin anzeigen:

```
1  "order": [2, 'desc'],
```

Der erste Wert gibt an, welche Spalte sortiert sein soll. Die erste Spalte hat den Wert 1. Als zweites gibt man die Richtung an, nach der sortiert werden soll. (ASC = aufsteigend, DESC = absteigend) So werden auch die unterschiedlichen Pfeile neben jedem Spaltenkopf angezeigt:

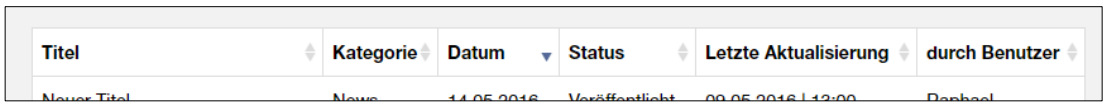


Abbildung 57 Tabellenkopf im Back-End

Diese Bilddateien habe ich ebenfalls heruntergeladen und zum Projekt hinzugefügt. Mit folgenden CSS-Regeln, die von der bereitgestellten CSS-Datei stammen, werden sie eingebunden:

```
1  table.dataTable thead .sorting,
2  table.dataTable thead .sorting_asc,
3  table.dataTable thead .sorting_desc {
4  cursor: pointer;
5  *cursor: hand;
6  }
7
8  table.dataTable thead .sorting,
9  table.dataTable thead .sorting_asc,
10 table.dataTable thead .sorting_desc,
11 table.dataTable thead .sorting_asc_disabled,
12 table.dataTable thead .sorting_desc_disabled {
13 background-repeat: no-repeat;
14 background-position: center right;
15 }
16
17 table.dataTable thead .sorting {
18 background-image: url("../img/sort_both.png");
19 }
20
21 table.dataTable thead .sorting_asc {
22 background-image: url("../img/sort_asc.png");
23 }
24
25 table.dataTable thead .sorting_desc {
```

```
26 background-image: url("../img/sort_desc.png");
27 }
28
29 table.dataTable thead .sorting_asc_disabled {
30 background-image: url("../img/sort_asc_disabled.png");
31 }
32
33 table.dataTable thead .sorting_desc_disabled {
34 background-image: url("../img/sort_desc_disabled.png");
35 }
```

Was man hier beachten muss, ist, dass Spalten mit einem Datum oder einer Uhrzeit nicht ohne weiteres chronologisch geordnet werden kann. Dazu benötigen wir eine Extension von der DataTable-Website. Zu diesem Thema gibt es im Forum eine **Beitrag**, in dem erklärt wird, wie Datums- und Zeitformate richtig sortiert werden können. Es wird auf ein weiteres Plugin zurückgegriffen, dass **moment.js** heisst. Auch diese Datei wird in die Seite importiert:

```
1  <!-- Laden des 3rd-Party Plugin Moment.js -->
2  <script type="text/javascript" src='../js/mo-
   ment.min.js'></script>
```

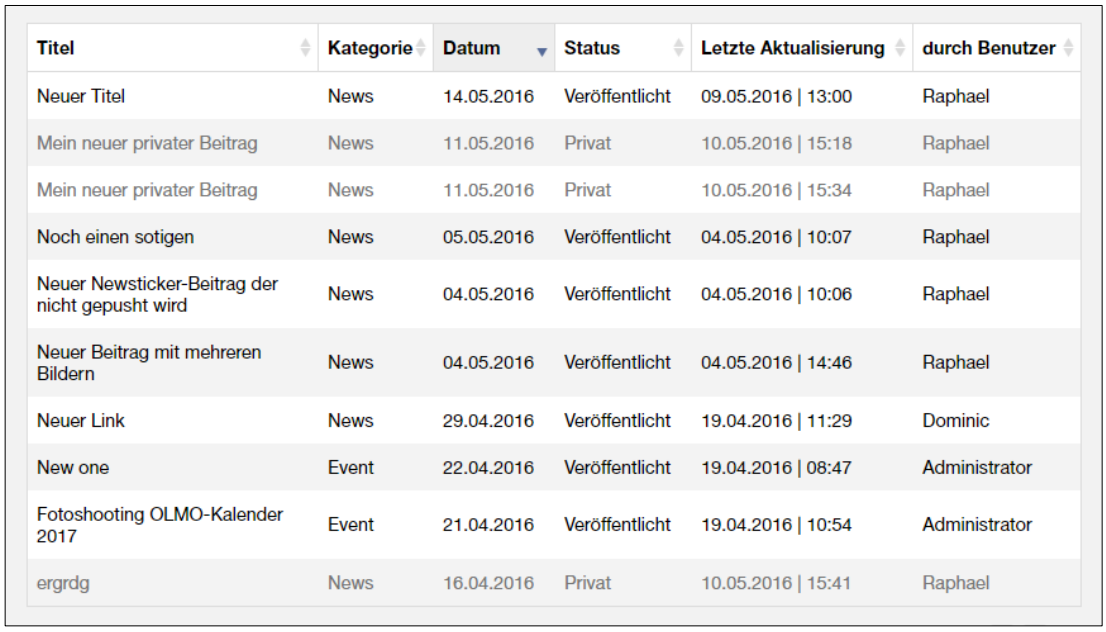
Weiter erhalten wir in Code-Snippet, das ich in ein neues JS-File einfüge. Dieses File nenne ich *datatables-extensions.js*. In diese Datei füge ich alle Snippets ein, damit nur eine JS-Datei geladen werden muss. Hat man diese Datei importiert, kann man eine Zeile zur Tabellen-Konfiguration hinzufügen:

```
1  $(document).ready(function() {
2  $.fn.dataTable.moment('DD.MM.YYYY');
3  $.fn.dataTable.moment('DD.MM.YYYY | HH:mm');
4  $('#posts_news').DataTable();
5  });
```

Der Befehl lautet *$.fn.dataTable.moment()* und in die Klammer wird das Datums- bzw. Zeitformat geschrieben. Dieses Format lautet in diesem Beispiel *DD.MM.YYYY*. Diese Formate können mithilfe dieser **Übersichtstabelle** erstellt werden. In der Tabelle der Newsticker-Beiträge befinden sich zwei Spalten mit einem Datum. Deshalb wird der obige Befehl zweimal aufgeführt mit beidem Datums- und Zeitformate die in der Tabelle auftauchen. Wichtig ist auch, dass man die spezifischen Spalten in der Konfiguration als Datum definiert:

```
1  columnDefs : [{type: 'date', targets: [3,5]}],
```

Wird dies nicht angegeben, funktioniert die Sortierung nicht richtig. In diesem Array wird zuerst der Typ der Spalte definiert und dann auf welche Spalten dieser Typ übernommen werden soll. Hier ist es die dritte und fünfte Spalte:



Titel	Kategorie	Datum	Status	Letzte Aktualisierung	durch Benutzer
Neuer Titel	News	14.05.2016	Veröffentlicht	09.05.2016 | 13:00	Raphael
Mein neuer privater Beitrag	News	11.05.2016	Privat	10.05.2016 | 15:18	Raphael
Mein neuer privater Beitrag	News	11.05.2016	Privat	10.05.2016 | 15:34	Raphael
Noch einen sotigen	News	05.05.2016	Veröffentlicht	04.05.2016 | 10:07	Raphael
Neuer Newsticker-Beitrag der nicht gepusht wird	News	04.05.2016	Veröffentlicht	04.05.2016 | 10:06	Raphael
Neuer Beitrag mit mehreren Bildern	News	04.05.2016	Veröffentlicht	04.05.2016 | 14:46	Raphael
Neuer Link	News	29.04.2016	Veröffentlicht	19.04.2016 | 11:29	Dominic
New one	Event	22.04.2016	Veröffentlicht	19.04.2016 | 08:47	Administrator
Fotoshooting OLMO-Kalender 2017	Event	21.04.2016	Veröffentlicht	19.04.2016 | 10:54	Administrator
ergrdg	News	16.04.2016	Privat	10.05.2016 | 15:41	Raphael

Abbildung 58 Tabelle mit den Newsticker-Beiträgen, sortiert nach Datum

Was mir auch noch nützlich erschien, war die Option **stateSave**. Damit werden die aktuellen Einstellungen des Benutzers im localStorage gespeichert und beim nächsten Aufruf der Seite wieder eingelesen:

```
1  stateSave: true,
```

Zusätzlich kann dem Item im localStorage ein eigener Namen gegeben werden:

```
1  stateSaveCallback: function(settings, data) {
2  localStorage.setItem('view_options_news',
   JSON.stringify(data));
3  },
4  stateLoadCallback: function ( ) {
5  try {
6  return JSON.parse(localStorage.getItem('view_op-
   tions_news'));
7  } catch (e) {}
8  },
```

Überprüft man den LocalStorage des Browsers so sieht man den gerade definierten Namen mit den Konfigurationen:

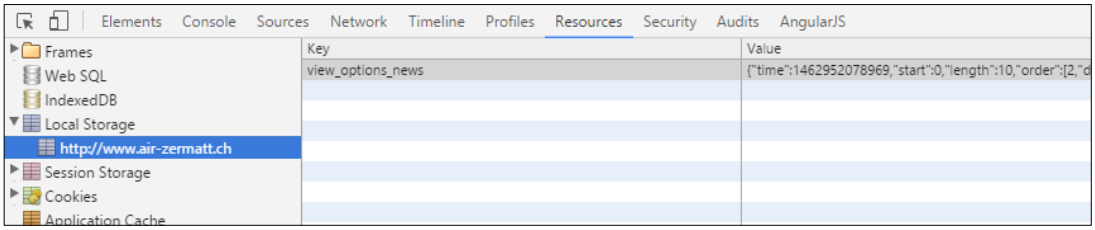


Abbildung 59 Local Storage im Browser Chrome

Mit den Optionen, welche **dom** zur Verfügung stellt, kann man die einzelnen Kontrollelement, die mit dem Plugin kommen beliebig positionieren. Für unsere Bedürfnisse habe ich die Element wie folgt angeordnet:

```
1  "dom": '<"#view_config"<"left"lf>>tip',
```

Wird die Seite aufgerufen, so stellt PHP lediglich die Tabelle bereit. jQuery manipuliert dann dieses DOM-Element und fügt weitere Elemente ein. Zum einen fügt er ein div-Element ein mit der ID *view_config*. In dieses div folgt ein weiters div mit der Klasse *left*. In dieses zweite div für das Plugin dann das Dropdown-Feld zum Ändern der anzuzeigenden Einträge pro Seite und die Suchfunktion ein. Nach diesen beiden divs folgt die eigentliche Tabelle. Anschliessend erscheint eine Info über die aktuelle Anzeige und zum Schluss kommen die Schaltflächen zum Navigieren auf die benachbarten Seiten. Alles zusammen sieht dann so aus:

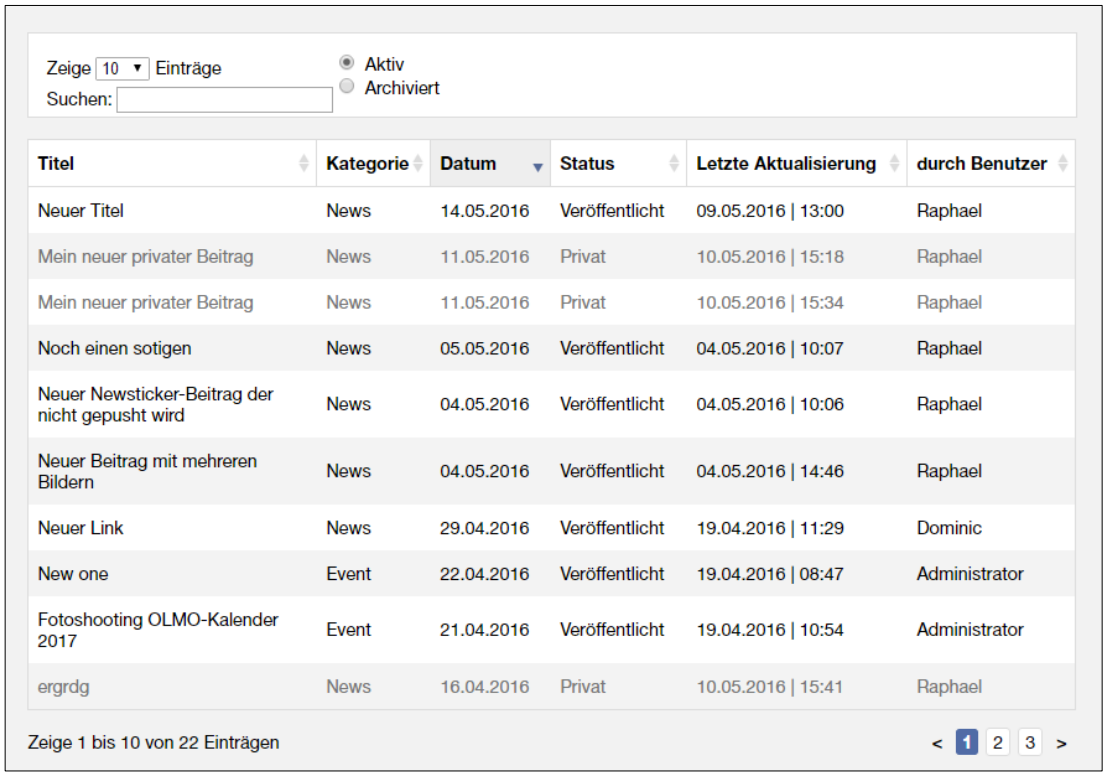


Abbildung 60 Gesamt Tabelle mit den Kontrollelementen

Über die beiden Radio-Buttons wird die Tabelle via AJAX manipuliert. Es wird demnach unterschieden zwischen aktiven und archivierten Beiträgen. In der Datenbank haben archivierte Beitrag die Version 0. Diese beiden Radio-Buttons werde ebenfalls via jQuery hinzugefügt:

```
$("div#view_config").append('<div id="post_archiv"><input
type="radio" class="view_options" name="where_version"
id="active" value="0" alt=">" on-
change="change_view_news(this)" checked> Aktiv</br><input
type="radio" class="view_options" name="where_version"
id="archive" value="0" alt="=" on-
change="change_view_news(this)"> Archiviert</br></div>');
```

Wie man sieht haben beide Elemente das HTML-Attribut *onchange* mit dem Aufruf der JS-Funktion *change_view_news(this)*. Diese Funktion ist in der Datei script.js hinterlegt:

```
function change_view_news(edit) {

edit['alt'] = edit.getAttribute('alt');

var xmlhttp;
if (window.XMLHttpRequest) {
xmlhttp=new XMLHttpRequest();
} else {
xmlhttp=new ActiveXObject("Microsoft.XMLHTTP");
}
xmlhttp.onreadystatechange=function() {
if (xmlhttp.readyState==4 && xmlhttp.status==200) {

document.getElementById('display_table').in-
nerHTML=xmlhttp.responseText;

}
}
xmlhttp.open("GET","../ajax/view_posts_news.php?a=" +
edit.name + "&v=" + edit.value + "&o=" + edit.alt,true);
xmlhttp.send();

}
```

Die Funktion nimmt den Parameter *edit* entgegen. Als erstes liest die Funktion den Wert heraus der beim Attribut *alt* steht. Dieses Attribut brauche ich, um der Funktion den gewünschten Operator mitzugeben. Andere relevante Informationen sind im Namen der Element gespeichert und im value-Attribut des Input-Feldes. Diese Informationen werden benötigt, um später die SQL-Abfrage abzuändern.

Die PHP-Datei *view_posts_news.php* nimmt die übermittelten get-Parameter auf und verarbeitet sie weiter:

```
1  if(isset($_GET["a"]) && isset($_GET["v"]) && is-
   set($_GET["o"])) {
2  $attribute = htmlspecialchars($_GET["a"]);
3  $value = htmlspecialchars($_GET["v"]);
4  $operator = $_GET["o"];
5
6  require("../connection.php");
7  $attribute = mysqli_real_escape_string($con, $attribute);
8  $value = mysqli_real_escape_string($con, $value);
9  $operator = mysqli_real_escape_string($con, $operator);
10
11 $condition = explode("_", $attribute);
12
13 $query_text = "SELECT PK_news, titel, datum, status, kat-
   egorie, edit, benutzer
14 FROM tbl_news, tbl_kategorie, tbl_benutzer
15 WHERE FK_kategorie = PK_kategorie AND FK_benutzer =
   PK_benutzer";
16
17 if($condition[0] == "where") {
18 $query_text .= " AND " . $condition[1] . " $operator
   '$value'";
19 }
20
21 $query_text .= " ORDER BY datum DESC";
22
23 $query = mysqli_query($con, $query_text);
24
25 // Laden des Templates zum Anzeigen der Tabelle auf der
   Seite Newsticker
26 // Erwartet die Variable $query
27 require("../templates/display_table_newsticker.php");
28 }
```

Der PHP-Befehl explode() splittet ein String anhand einer Zeichenkette in mehrere Strings auf, In diesem Fall ist die Zeichenkette ein Underline:

```
1  $condition = explode("_", $attribute);
```

Die aufgeteilten Strings sind dann über die Variable $condition und dem passenden Index aufrufbar. z.B:

```
1  echo $condition[0];
```

Es wird also der Name des Radio-Buttons aufgeteilt in *where* und *version*. Die Variable *$query_text* stellt den ersten Teil der Abfrage bereit. Dann wird überprüft, ob es sich beim ersten aufgeteilten String um das Wort *where* handelt. Ist dies der Fall, so wird der Variable *$query_text* der folgende Text hinzugefügt:

```
1  $query_text .= " AND " . $condition[1] . " $operator
   '$value'";
```

Diese Zeile fügt, je nach Auswahl des Radio-Buttons, die Bedingung " *AND version > 0*" hinzu. Es werden also beim Ausführen des Querys alle Einträge ausgewählt, die eine höhere Version haben als 0. Also werden alle aktiven Beiträge ausgegeben. Die andere mögliche Bedingung ist: " *AND version = 0*". Diese Abfrage würde alle archivierten Beiträge auslesen.

Dann wird die Template-Datei geladen, um die Abfrage zu formatieren. Anschliessend wird die Antwort an die JS-Funktion zurückgesendet und der Inhalt der Antwort wird in die Tabelle eingefügt:

```
1  document.getElementById('display_table').inner-
   HTML=xmlhttp.responseText;
```

13SERVERSEITIGE DATENBANK AUFRÄUMEN

Wenn beispielsweise ein Benutzer die App löscht, so merkt man dies nicht und der Eintrag des Benutzers bleibt in unserer Datenbank gespeichert. Es wird also auch bei jedem Push eine Nachricht an das Gerät gesendet, auf dem die App nicht mehr existiert. Dies kann mit der Zeit das ganze System ausbremsen. Um dies zu verhindern, wird in der Antwort des GCM-Servers aufgeführt, welche RegistrationIDs nicht mehr aktuell sind. Anhand von dieser Rückmeldung können die betreffenden Einträge auf unserem Server gelöscht oder immerhin angepasst werden. Bei Apple gibt es einen eigenen Feedback-Service zur Eliminierung von solchen veralteten RegistrationIDs.

13.1 Anpassungen für Android

Bisher erhalten wir vom GCM-Server eine Antwort, die in die Variable *$result* gespeichert wird. Nun decodieren wir diese Antwort, damit wir deren Informationen abrufen können. Dann zählt man die Fehlermeldungen, damit man danach mit einer Schlaufe die jeweiligen, fehlerhaften Einträge anzupassen.

Wir unterscheiden hier zusätzlich, ob im veralteten Eintrag die Kontaktinformationen des Kunden gespeichert sind oder nicht. Sind sie nicht gespeichert, wird der Eintrag gelöscht. Sind die Informationen vorhanden, werden lediglich die beiden Push-Benachrichtigungen auf *false* geändert. Somit werden keine weiteren Payloads an das spezifische Gerät gesendet. Der Code dazu sieht folgendermassen aus:

```
1  // Parsen der Antwort des GCM-Servers für weiteren Ge-
   brauch
2  $response = json_decode($result, true);
3  // print_r($response);
4
5  // Zählen der Resultate
6  $length = count($response["results"]);
7
8  // For-Schlaufe zum Überprüfen, ob ein Fehler aufgetreten
   ist
9  for($i=0;$i<$length;$i++) {
10 if($response["results"][$i]["error"]) {
11
12 // Speichern der RegistrationID bei welcher ein Fehler
   aufgetreten ist
13 $id = $registrationIds[$i];
14
15 // Überprüfung, ob von Kunden die Kontaktinformationen
   gespeichert sind
```

```
16 $query = mysqli_query($con, "SELECT email FROM tbl_kunde
   WHERE registration_id = '$id'");
17 while($row = mysqli_fetch_array($query)) {
18
19 // Falls E-Mailadresse nicht gespeichert, lösche den Da-
   tensatz
20 if($row["email"] == "") {
21 $query_delete = mysqli_query($con, "DELETE FROM tbl_kunde
   WHERE registration_id = '$id' AND email = ''");
22
23 if($query_delete) {
24 // echo "Eintrag gel&ouml;scht!";
25 }
26
27 } else {
28 // Falls E-Mailadresse gespeichert, setzte beide Push zu
   false
29 // damit keine neuen Push an das Gerät gesendet wird
30 $query_update = mysqli_query($con, "UPDATE tbl_kunde SET
   rundflug_push = 'false', newsticker_push = 'false' WHERE
   registration_id = '$id'");
31 if($query_update) {
32 // echo "Eintrag ge&auml;ndert";
33 }
34 }
35 }
36
37 }
38 }
```

Weiter kann man nun die Anzahl der erfolgreichen Übertragungen, sowie der fehlerhaften auslesen:

```
1  $result = $response["success"];
2  $failure = $response["failure"];
```

13.2 Anpassungen für iOS

Wie üblich sieht das bei Apple etwas anders aus. Der APNS verfügt über einen eigenen Feedback Service, der regelmässig aufgerufen werden muss, um veraltete RegistrationIDs zu erhalten.

13.2.1 Das Script für den APNS Feedback Service

Ähnlich wie beim Push-Service wird eine Verbildung zum Apple-Server hergestellt:

```
1  // Put your private key's passphrase here:
2  $passphrase = '';
3
4  $ctx = stream_context_create();
5  stream_context_set_option($ctx, 'ssl', 'local_cert',
   'cert/apns_dist_cert.pem');
6  stream_context_set_option($ctx, 'ssl', 'passphrase',
   $passphrase);
7
8  // Open a connection to the APNS server
9  $fp = stream_socket_client(
10 'ssl://feedback.push.apple.com:2196', $err, $errstr, 60,
   STREAM_CLIENT_CONNECT|STREAM_CLIENT_PERSISTENT, $ctx);
11
12 $feedback_tokens = array();
13 //and read the data on the connection:
14 while(!feof($fp)) {
15 $data = fread($fp, 38);
16 if(strlen($data)) {
17 $feedback_tokens[] = unpack("N1timestamp/n1length/H*dev-
   token", $data);
18 }
19 }
20
21 // Zählen der Resultate
22 $length = count($feedback_tokens);
23
24 if($length > 0) {
25
26 require("connection.php");
27
28 // For-Schlaufe zum Überprüfen, ob ein Fehler aufgetreten
   ist
29 for($i=0;$i<$length;$i++) {
30
31 // Speichern der RegistrationID bei welcher ein Fehler
   aufgetreten ist
32 $id = $feedback_tokens[$i]["devtoken"];
33
34 // Überprüfung, ob von Kunden die Kontaktinformationen
   gespeichert sind
35 $query = mysqli_query($con, "SELECT email FROM tbl_kunde
   WHERE registration_id = '$id'");
```

```
36 while($row = mysqli_fetch_array($query)) {
37
38 // Falls E-Mailadresse nicht gespeichert, lösche den Da-
   tensatz
39 if($row["email"] == "") {
40 $query_delete = mysqli_query($con, "DELETE FROM tbl_kunde
   WHERE registration_id = '$id' AND email = ''");
41
42 if($query_delete) {
43 // echo "Eintrag gel&ouml;scht!";
44 }
45
46 } else {
47 // Falls E-Mailadresse gespeichert, setzte beide Push zu
   false
48 // damit keine neuen Push an das Gerät gesendet wird
49 $query_update = mysqli_query($con, "UPDATE tbl_kunde SET
   rundflug_push = 'false', newsticker_push = 'false' WHERE
   registration_id = '$id'");
50 if($query_update) {
51 // echo "Eintrag ge&auml;ndert";
52 }
53 }
54 }
55
56 }
57
58 }
59
60 // Close the connection to the server
61 fclose($fp);
```

Als erstes werden das Zertifikat und dessen Passwort angegeben. Dann folgt der Verbindungsaufbau zum Feedback-Server. Der Port ist nicht derselbe, wie beim Verbindungaufbau für einen Push.

Als nächstes wird ein Array vorbreitet, in dem die fehlerhaften Token gespeichert werden. Danach wird überprüft, ob mindestens ein Eintrag existiert und eine Schlaufe ausgeführt. Diese Schlaufe bewirkt das gleiche, wie die obere Schlaufe bei Android. Sind die Kontaktinformationen des Benutzers vorhanden, wird der Eintrag geändert, ansonsten wird er gelöscht. Zum Schluss wird die Verbindung zum Server wieder geschlossen. Meistens wird diese Abfrage beim Server keine DeviceToken liefern. Die Antwort des Servers ist in diesem Fall *null*.

13.2.2 Cronjob einrichten

Damit dieses Script nicht von Hand oder nach jedem Push ausgeführt werden muss, kann beim Host ein Cronjob eingerichtet werden. Mithilfe des Cronjob kann ein spezifisches File mit einem definierbaren Abstand ausgeführt werden:

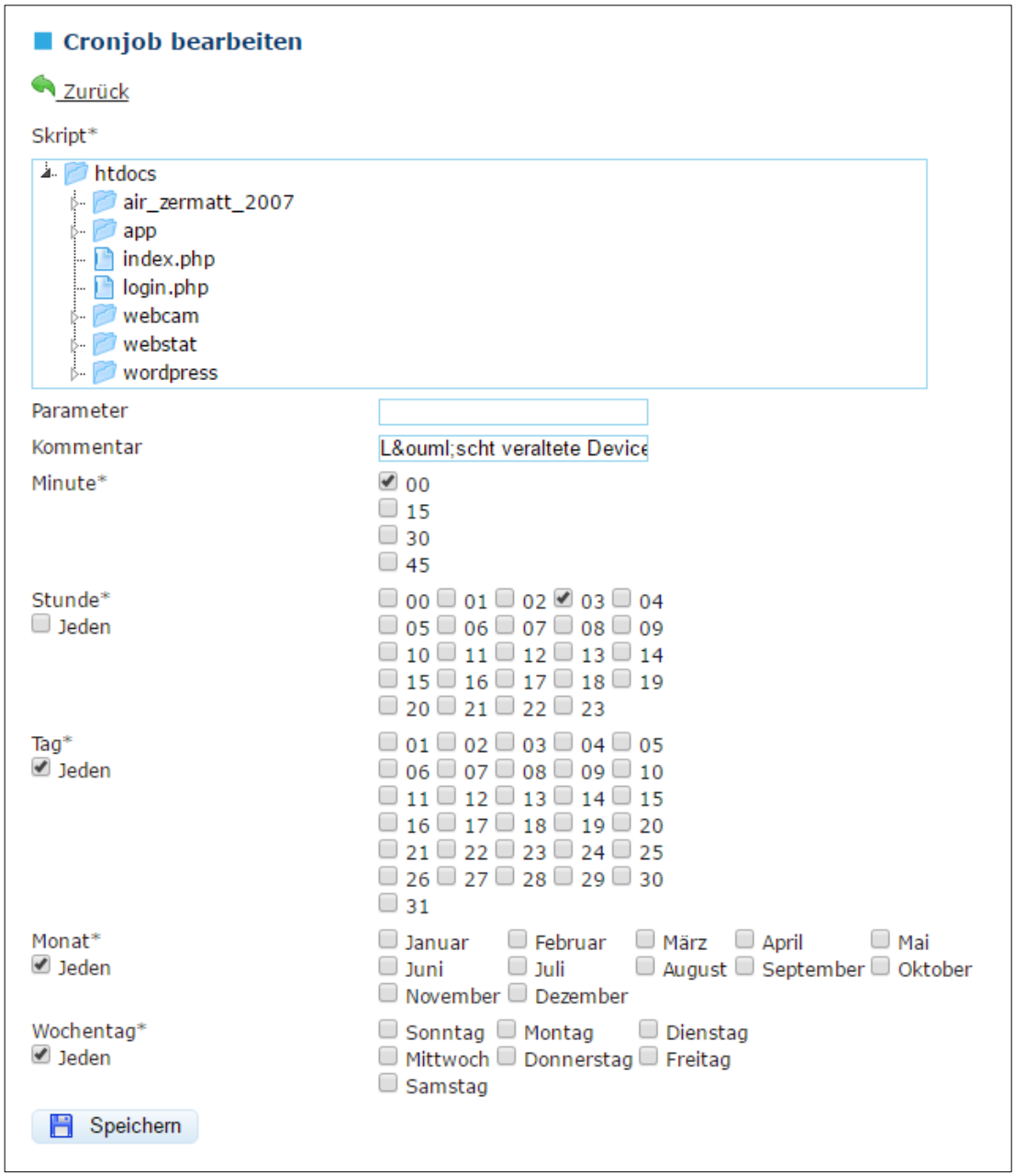


Abbildung 61 Cronjob für das Script update_apns.php beim Host Rhone

Diese Einstellungen können im **Back-End des Hosts** geändert werden.

13.3 Serverseitige Anpassungen

13.3.1 Alte Rundflug-Angebote automatisch archivieren

Nicht nur die RegistrationIDs muss man von Zeit zu Zeit säubern, sondern auch die Rundflug-Angebote. Diese kann man zwar im Back-End manuell archivieren, aber dies lässt sich auch automatisieren. Dazu habe ich folgede Zeilen Code an das obige Script, das vom Cronjob regelmässig ausgeführt wird, hinzugefügt:

```
1  // Automatisches Archivieren von alten Rundflug-Angebote
2  require("connection.php");
3
4  // Wie lange sollen alte Einträge angezeigt werden?
5  $time = 604800; // Eine Woche in Sekunden
6
7  $query = mysqli_query($con, "SELECT PK_rundflug, datum
   FROM tbl_rundflug WHERE version > '0'");
8
9  while($row = mysqli_fetch_array($query)) {
10 $id = $row["PK_rundflug"];
11 $post_time = strtotime($row["datum"]);
12 $diff = time() - $post_time;
13
14 if($diff > $time) {
15 mysqli_query($con, "UPDATE tbl_rundflug SET version = '0'
   WHERE PK_rundflug = '$id'");
16
17 }
18 }
```

Mit diesem Code-Snippets werden die RF-Angebote aus der Datenbank gelesen, welche nicht archiviert sind (*version != 0*) und liest das angegebene Datum aus. Dieses Datum wird mit dem aktuellen Datum verglichen. Ist die Differenz zwischen diesen beiden Daten grösser als der Wert in der Variable *$time*, so wird der Eintrag archiviert. Aktuell ist in dieser Variable eine Woche in Form von Sekunden angegeben. D.h. spätestens eine Woche nach dem das Angebot "abgelaufen" ist, wird der Eintrag archiviert.

14VERHALTEN BEIM UPDATE DER APP

14.1 App-Version anpassen

Damit eine neue Version in den App Store / Play Store hochgeladen werden kann, muss vor dem Builden des Release-APK die Versionennummer in der config.xml angepasst werden. Bei der Versionennummer gibt es mehrere Theorien, wie diese bei einer neuen Version angepasst werden soll. Ich entschied mich für eine 3-geteilte Versionierung entschieden.

```
<?xml version='1.0' encoding='utf-8'?>
<widget id="ch.air-zermatt.app"
version="0.1.6"
xmlns="http://www.w3.org/ns/widgets"
xmlns:cdv="http://cordova.apache.org/ns/1.0
">
```

Abbildung 62 Versionierung in der config.xml-Datei

Die erste Zahl steht für ein Major-Update. Diese Zahl wird um eins erhöht, wenn ein neuer Entwicklungs-Meilenstein erreicht wird. Zum Beispiel wird die Version, mit welcher wir die neue App ankündigen werden, die Versionennummer 1.0.0 tragen. In der Entwicklungsphase beginnt die Versionennummer aber mit einer Null.

Die zweite Zahl stehlt ein Minor-Update dar. Diese Zahl wird um eins erhöht, wenn neue, kleinere Funktionen hinzugekommen sind und wenn mittelschwere Fehler korrigiert wurden.

Die dritte Zahl bezeichnet die Build-Version. Diese Zahl wird wohl am häufigsten geändert und zwar, wenn kleinere Fehler behoben wurden (Bug Fixes) oder wenn Änderungen am statischen Inhalt gemacht wurden.

Wichtig ist, zu wissen, dass jede Version, die man hochladen will, eine einmalige Versionennummer haben muss. Zudem darf diese Nummer immer nur erhöht werden. Ein "Downgrade" ist nicht möglich!

Beim Testen auf dem Smartphone kann die App-Versionennummer beliebig erhöht werden. Wenn die Version für die Veröffentlichung bereit ist, kann sie wieder herabgesetzt werden. Man muss dann die auf dem Smartphone installierte Debug-App löschen, damit man die richtige App wieder installieren kann.

14.2 Änderungen an der Datenbank

Normalerweise wird beim Update die eigentliche App durch die neue Version ersetzt. Die Datenbank wird jedoch nicht verändert. Dies ist von Vorteil, da die gespeicherten Informationen in der Datenbank nicht verloren gehen. Wenn nun aber mit sqliteBrowser Änderungen an der Datenbank gemacht werden müssen, werden diese deshalb nicht übernommen.

Um diese Änderungen an den bestehenden, lokalen Datenbanken zu tätigen, überprüfe ich mit folgendem Code die Version der Applikation.

```
1  $cordovaAppVersion.getVersionNumber().then(function (ver-
   sion) {
2  console.log("App-Version: " + version);
3  }
```

Diese Zeile gibt die App-Version aus, die in der config.xml-Datei hinterlegt wurde. Diese Versionsnummer wird bei jedem Update angepasst.

Diese Versionsnummer ist auch in der Datenbank hinterlegt. Beim Ausführen der App werden die beiden Werte verglichen. Weichen die Werte voneinander ab, so handelt es sich um ein Update und man kann daraufhin spezifische Funktionen ausführen. Unter anderem auch Änderungen an der Datenbank. Hierzu ein Bespiel:

```
1  $cordovaAppVersion.getVersionNumber().then(function (ver-
   sion) {
2  var app_version = version;
3  db.transaction(function(tx) {
4  tx.executeSql("SELECT app_version FROM tbl_benutzer", [],
   function(tx, res) {
5  console.log("Aktuelle Version: " + app_version + ", ge-
   speicherte Version: " + res.rows.item(0).app_version +
   ",");
6
7  if(res.rows.item(0).app_version != app_version) {
8
9  ////////////////////////////////////////////////
10 // Änderungen an der Datenbank bei Update
11
12 // Änderungen bei Version 0.1.13
13 var db_version = $scope.compare_db_version(local_version,
   "0.1.13");
14 if(db_version < 0) {
15
16 tx.executeSql("ALTER TABLE tbl_basis ADD email TEXT", [],
   function() {
17 console.log("Spalte hinzugefügt!");
```

```
18 },
19 function(error) {
20 console.error("Fehler: " + JSON.stringify(error));
21 });
22
23 tx.executeSql("UPDATE tbl_basis SET mail = ? WHERE PK_ba-
   sis = '1'", ['zermatt@air-zermatt.ch'], function() {
24 console.log("E-Mail zermatt hinzugefügt!");
25 },
26 function(error) {
27 console.error("Fehler: " + JSON.stringify(error));
28 });
29
30 tx.executeSql("UPDATE tbl_basis SET mail = ? WHERE PK_ba-
   sis = '2'", ['raron@air-zermatt.ch'], function() {
31 console.log("E-Mail raron hinzugefügt!");
32 },
33 function(error) {
34 console.error("Fehler: " + JSON.stringify(error));
35 });
36
37
38 }
39
40  // Ende Änderungen an der Datenbank bei Update
41 ////////////////////////////////////////////////
42
43 tx.executeSql("UPDATE tbl_benutzer SET app_version = ?;",
   [app_version], function() {
44 $rootScope.$broadcast('saveInfos', []);
45 },
46 function(error) {
47 console.log('Error: ' + JSON.stringify(error));
48 });
49
50 push.on('registration', function(data) {
51 console.log('new registration event!');
52 $scope.save_regId(data);
53 });
54
55 }
56
57 },
```

```
58 function(error) {
59 // error
60 });
61 }, function(error) {
62 // error
63 }, function() {
64 // erfolgreich
65 });
66
67 });
```

In diesen Zeilen werden die aktuelle, sowie die gespeicherte Versionennummer der App geladen und schon mal in der Konsole ausgegeben. Dann wird mit der Funktion *$scope.compare_db_version* überprüft, ob die beiden Werte unterschiedlich sind. Wenn dies der Fall ist, werden drei SQL-Abfragen ausgeführt. Die erste Funktion fügt die Spalte *mail* zu der Tabelle *tbl_basis* hinzu. Die beiden anderen Funktionen fügen die jeweilige E-Mailadresse der beiden Basen in die neu erstellte Spalte ein. So kann man Änderungen an der Datenbank bei einem Update erzielen. ~~Nachdem hochladen der neuen, signierten APK-Datei, können diese Funktionen wieder entfernt werden.~~

Update: Da man nie sicher weiss, was für eine Version auf dem Smartphone installiert ist, wenn ein Update durchgeführt wird, müssen die Änderungen an der Datenbank zwischen der installierten und der neuesten veröffentlichten Version im Script bleiben! Ansonsten würden nicht alle Änderungen ausgeführt und es könnte Fehler auftauchen.

nanufacturer	model	platform	os_version	app_version	registration_id
amsung	SM-G925F	Android	6.0.1	0.1.13	ft0GkI4viGY:AP
amsung	SM-G935F	Android	6.0.1	0.1.13	eK_5mgonc-g:A
cer	A3-A10	Android	4.4.2	0.1.13	dww2gUk4usU:A
pple	iPhone4,1	iOS	9.3.2	0.1.7	450ee4e53cb5c
ony	E2303	Android	5.0	0.1.16	eLfJueympLQ:A
ony	E2303	Android	5.0	0.1.16	c7iwq6Bh6LM:A

Abbildung 63 Unterschiedliche App-Versionen machen Änderungen an der DB komplizierter

Wie man im obigen Bild sieht, kann es gut sein, dass ein Benutzer nicht immer direkt auf die neuste Version updatet. Änderungen zwischen diesen Versionen müssen also theoretisch gespeichert werden, bis jeder seine App aktualisiert.

Damit doch von Zeit zu Zeit die Änderungen aus den MainCtrl gelöscht werden können, kann man in der Datenbank in der Tabelle tbl_kunde die App-Versionen überprüfen.

Danach wird die gespeicherte App-Versionennummer aktualisiert, damit die obigen Funktionen beim nächsten Aufstarten nicht noch einmal ausgeführt werden. Ist die neue Versionennummer in der Datenbank gespeichert, werden die gesamten Informationen auf unseren Server gespeichert.

cordova	manufacturer	model	platform	os_version	app_version
5.1.1	Sony	E2303	Android	5.0	0.1.10
4.1.1	Apple	iPhone4,1	iOS	9.3.2	0.1.6
5.1.1	Sony	E2303	Android	5.0	0.1.10
5.1.1	samsung	SM-G925F	Android	6.0.1	0.1.10
5.1.1	samsung	SM-G935F	Android	6.0.1	0.1.10
5.1.1	Acer	A3-A10	Android	4.4.2	0.1.10
5.1.1	Sony	E2303	Android	5.0	0.1.10
5.1.1	Sony	E2303	Android	5.0	0.1.12
5.1.1	Sony	E2303	Android	5.0	0.1.11
					0.1.11
5.1.1	asus	Nexus 7	Android	5.0	0.1.11
5.1.1	asus	Nexus 7	Android	5.0	0.1.11
5.1.1	LGE	Nexus 5	Android	5.1.1	0.1.11
5.1.1	samsung	SM-G920F	Android	5.1.1	0.1.11
5.1.1	samsung	SM-G920F	Android	5.1.1	0.1.11
5.1.1	samsung	SM-N9005	Android	4.4.2	0.1.11
5.1.1	samsung	SM-N9005	Android	4.4.2	0.1.11
5.1.1	motorola	XT1092	Android	4.4.4	0.1.11
5.1.1	motorola	XT1092	Android	4.4.4	0.1.11
5.1.1	LGE	Nexus 4	Android	5.1	0.1.11
5.1.1	LGE	Nexus 4	Android	5.1	0.1.11
5.1.1	motorola	XT1064	Android	4.4.4	0.1.11
5.1.1	motorola	XT1064	Android	4.4.4	0.1.11

Abbildung 64 Auszug der Informationen über die Smartphones

14.3 Neuladen von gespeicherten Bildern

Um Bilder, welche auf dem Gerät des Benutzers gespeichert sind zu erneuern, habe ich eine Funktion hinzugefügt, die das alte Bild löscht und das neue vom Server herunterläd. Beim Überprüfen der App Version erstelle ich ein Array in der Variable *reload_pics*. In dieses Array werden später die Namen der Datei gespeichert, die aktuallisert werden sollen. Dazu

vergleichen wir zuerst wieder die gespeicherte und die aktuelle Versionennummer. Bei einer Abweichung fügen wir über die *push()*-Funktion die Namen der Datei hinzu:

```
1  // Änderungen bei Version 0.2.5
2  var db_version = $scope.compare_db_version(local_version,
   "0.2.5");
3  if(db_version < 0) {
4  console.log("Änderungen von 0.2.5 an DB");
5  reload_pics.push("alpen-panorama.jpg",
   "gourmet_rundflug.jpg", "matterhorn.jpg",
   "eispalast_jungfraujoch.jpg", "taxiflug_raron.jpg",
   "taxiflug_zermatt.jpg", "saas-fee_mittellalin.jpg",
   "schnellster_lift_der_welt.jpg");
6  }
```

Somit wird auch hier sichergestellt, dass die Bilder nicht bei jedem Start von neuem heruntergeladen werden. Die eigentliche Funktion zum Aktuallisieren folgt weiter unten im Script. In einer for-Schlaufe wird dort ein Bild nach dem anderen gelöscht und erneut von Server heruntergeladen:

```
1  // Erneutes Herunterladen der angegebenen Bilder vom
   Server
2  var count_reload_pic = reload_pics.length;
3  if(count_reload_pic != 0) {
4  for(i=0;i<count_reload_pic;i++) {
5
6  $cordovaFile.removeFile(cordova.file.dataDirectory +
   'bilder/', reload_pics[i])
7  .then(function (success) {
8  console.log("Success: " + success.fileRemoved.name);
9  $scope.download_pic(success.fileRemoved.name);
10 }, function (error) {
11 console.log("Löschen fehlgeschlagen: " +
   JSON.stringify(error));
12 });
13
14 }
15 }
```

Zum Herunterladen wird die bereits existiertende Funktion *download_pic()* verwendet.

14.4 Updaten von Cordova / Plattformen und Plugins

Von Zeit zu Zeit kann ein Update von den Core-Files, also Cordova selbst, die Plattformen und die Plugins, gemacht werden. Es empfiehlt sich immer eher diese Versionen zu installieren, die vom Hersteller als stabil bezeichnet werden.

Bei Apple kann es sein, das nach einem Update der Plattform oder von Cordova gar nichts mehr geht. In diesem Fall helfen die Debugger Outputs von Xcode weiter. Ich hatte einmal das Problem, dass Xcode nicht alle Files zu bestimmten Plugins gefunden hat, was zur Folge hatte, dass der Event *deviceready* nicht ausgeführt wurde.

In **diesem Forum** habe ich herausgefunden, dass beim Aktualisieren der Plattform die Liste der Plugins nicht gelöscht wird. Daher wird diese Liste bei der neuen Plattform nicht erneut erstellt, was bedeutet, dass sie veraltet ist. Löscht man die Daten *ios.json* im Ordner *Plugins* bevor die neue Version der Plattform installiert wird, wird die Liste neu erstellt und Xcode findet alle benötigten Files zum Kompilieren.

15ALPHA-TESTPHASE BEI PLATTFORM ANDROID

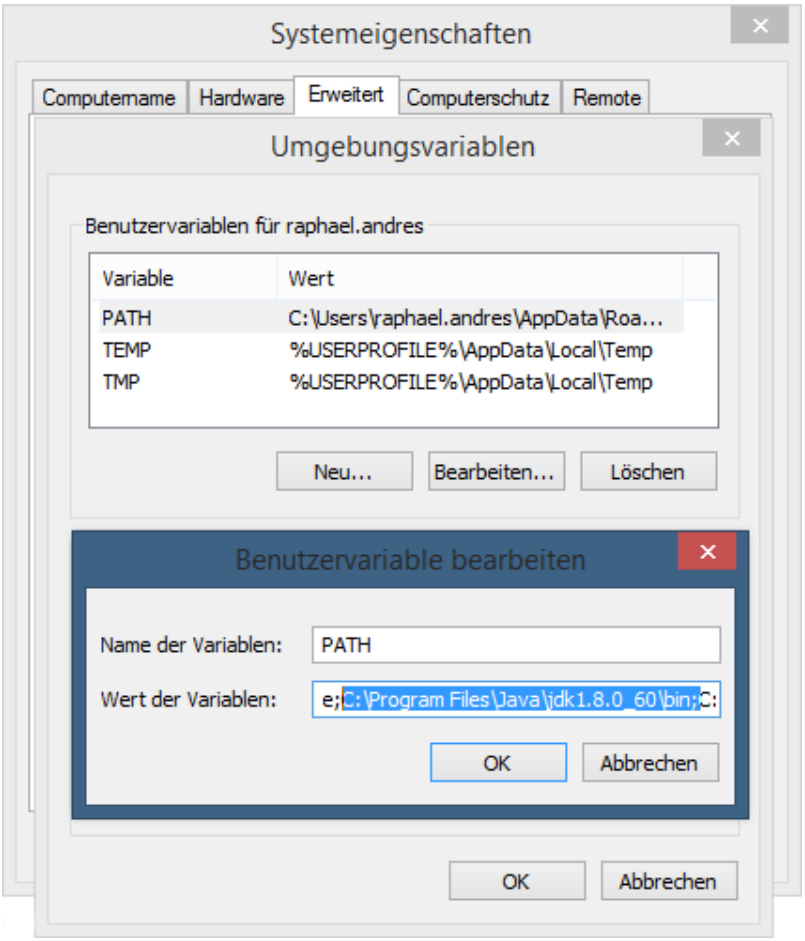
Um die App nun für Android zu testen, kann man eine Release-Version der App schon mal in der Google Developer Console in Alphaphase hochladen. Das heisst die App muss noch nicht fertig sein und auch noch nicht vollumfänglich funktionieren. Die App wird deshalb auch noch nicht öffentlich erreichbar sein. Man kann aber die E-Mailadresse von auserwählten Android-Benutzer angeben, welche die App zu Testzwecken bereits installieren dürfen.

15.1 APK-Datei signieren

Um eine unsignierte Release-Version der App zu signieren braucht man das **JDK**. Als erstes muss eine Datei erstellt werden, in der mehrere Schlüssel (Private Key) und Informationen des Unternehmens gespeichert werden. Diese Datei nennt man Keystore. Damit man so ein File generieren lassen kann muss sichergestellt sein, das der Pfad zur folgenden Anwendung in den Umgebungsvariablen gespeichert ist:

C:\Program Files\Java\jdk1.8.0_60\bin

Dieser Pfad muss man in die Variable PATH hinzufügen werden:



Zudem kann bereits ein weiterer Pfad eingefügt werden und zwar der zu den Build-Tools von Android SDK:

C:\Users\raphael.andres\Documents\Android SDK Tools\build-tools\23.0.2

Dort wird später die Anwendung zipalign gebraucht.

15.1.1 Keystore-File generieren

Weil ich dieses Signieren zum ersten Mal gemacht habe, musste ich mich in Internet informieren. Ich fand im Forum Stackoverflow eine gute Anleitung:

http://stackoverflow.com/questions/35173542/unable-to-sign-the-unsigned-apk

Das File kann wiederum über die Eingabeaufforderung erstellt werden. Es empfiehlt sich, diese einmalige Schritte genau und vorsichtig auszuführen:

```
1  keytool -genkey -v -keystore certificates/my-release-
   key.keystore -alias alias_name -keyalg RSA -keysize 2048
   -validity 10000
```

Mit *keytool* wird die Anwendung mit folgenden Parameter aufgerufen. Der Paramter *–genkey* sagt dem Programm, dass ich einen Schlüssel generieren möchte. Mit dem Paramter *–v* werden ausführliche Informationen angezeigt. Via *–keystore* kann der Speicherort des generierten Files angegeben werden. In diesem Fall wird die Datei in einem Unterordner *certificates* und mit dem Namen *my-release-key.keystore* abgespeichert. Als Alias kann man dem Schlüssel einen Namen geben, dieser Namen ist wichtig und wird später gebraucht, um die App zu signieren! Mit *–keyalg* gibt man den Algorithmus an, mit dem die Datei verschlüsselt wird. *–keysize* bezeichnet die Grösse des Schlüssels und *–validity* besagt, wie lange dieser Schlüssel gültig sein soll im Tagen. In diesem Beispiel sind dies 10'000 Tage, also etwa 27 Jahre.

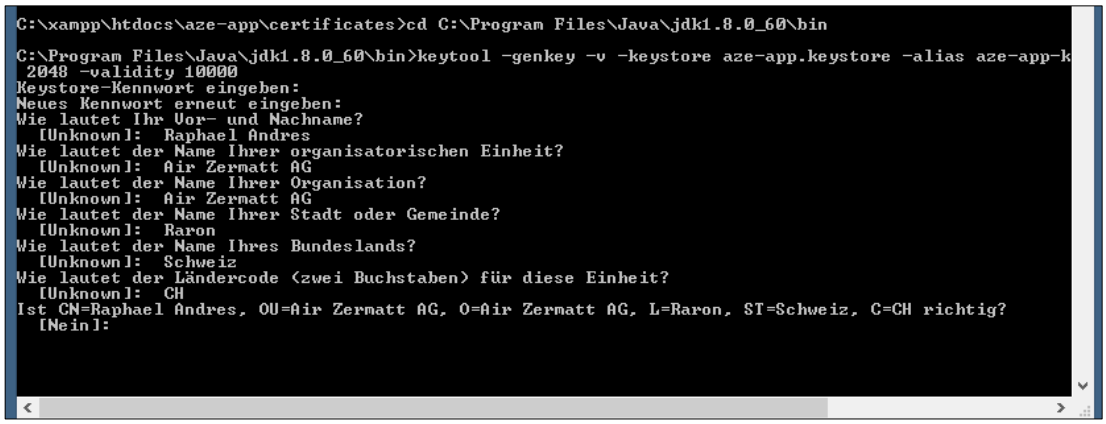


Abbildung 65 Generieren einer Keystore-Datei

Führt man diesen Befehl aus, muss man gewisse Angaben über die eigene Person und über das Unternehmen angeben. Man gibt das Keystore-Kennwort an, das später erneut angegeben werden muss, wenn die App signiert werden soll. Später wird ein zweites Mal nach einem Passwort gefragt. Hier ist wichtig, dass man dasselbe Passwort angibt. Mit Enter kann das Passwort von oben übernommen werden.

Es ist sehr wichtig, dass man diese Datei nicht verliert! Will man ein Update der App publizieren, so muss diese APK mit demselben Key signiert werden, ansonsten wird die APK nicht als Update angenommen!

15.1.2 Unsignierte APK-Datei signieren

Als nächstes kann man die APK mit zuvor erstelltem Key signieren. Dies funktioniert wiederum mit folgendem Befehl in der Eingabeaufforderung:

```
1  jarsigner -verbose -sigalg SHA1withRSA -digestalg SHA1 -
   keystore certificates/aze-app.keystore platforms/an-
   droid/build/outputs/apk/android-release-unsigned.apk aze-
   app-keystore
```

jarsigner ist die Anwendung, welche die Signierung durchführt. *–verbose* gibt an, das ausführliche Informationen über den Prozess angezeigt werden soll. Mit *–sigalg* und *–digestalg* wird wieder dem Algorithmus angegeben, wie Verschlüsselt werden soll. Mit *–keystore* gibt man an, wo die keystore-Datei zu finden ist. In diesem Fall wieder im Unterordner *certificates*. Als nächstes wird der Speicherort der unsignierten APK-Datei angegeben und zum Schluss der Alias, der keystore-Datei. Führt man den Befehl aus, so muss zweimal das Passwort angegeben werden. Die bestehende unsignierte App wird durch die signierte ersetzt.

In einem zweiten Befehl kann überprüft werden, ob alle Komponenten der App signiert wurden. Dieser Befehl ist aber optional:

```
1  jarsigner -verify -verbose -certs platforms/an-
   droid/build/outputs/apk/android-release-unsigned.apk
```

Mit dem Parameter *–verify* wird der Anwendung gesagt, dass man die Signatur überprüfen möchte. Nach *–certs* wird der Pfad zur nun signierten App angegeben.

Als letzten Schritt wird noch ein Befehl ausgeführt, der vom Android SDK angeboten wird:

```
1  zipalign -v 4 platforms/android/build/outputs/apk/an-
   droid-release-unsigned.apk platforms/android/build/out-
   puts/apk/aze-app.apk
```

zipalign bezeichnet die Anwendung. Wiederum gibt man der Speicherort der signierten App an und weiter der Speichertort, in dem die endgültige APK-Datei gespeichert wird.

15.2 APK-Datei zum Google Play Store hochladen

Will man die signierte APK zum **Google Play Store** hochladen, kann man entscheiden, ob man die App direkt veröffentlichen will, oder für den Alpha- bzw. Betatest bereitstellen möchte. Der Alpha-Test ist vorgesehen, wenn die App noch nicht vollumfänglich funktioniert. Durch eine E-Mail-Liste kann man Google-Konten angeben, die berechtigt sind, die App herunterzuladen und zu testen. Der Beta-Test funktioniert identisch. Nur sollten hier alle Funktionen der App funktionieren und die App wird an ein etwas breiteres Publikum gerichtet. Damit können noch kleinere Bugs vor dem Release gefunden und behoben werden.

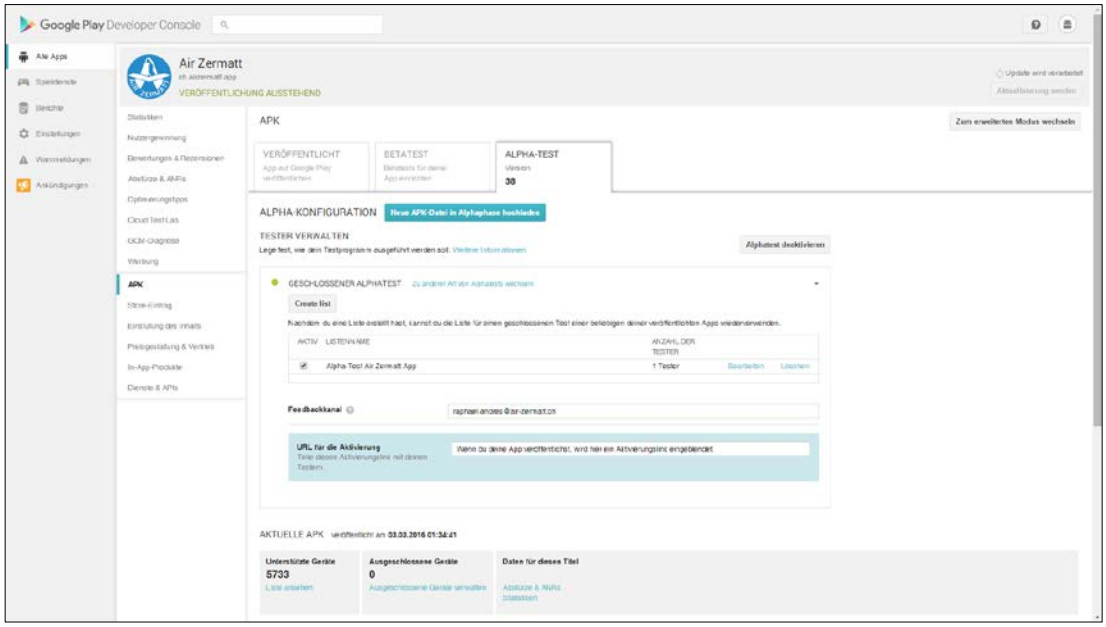


Abbildung 66 Alpha-Test-Bereich im Google Play Store

Neben der APK-Datei müssen noch weitere Informationen zur App angegeben werden. Dies sind z.B. Informationen und Bilder die im Store angezeigt werden, die Einstufung des Inhalts, Kontaktinformationen und die Preisgestaltung. Erst wenn all diese Informationen angegeben wurden, kann die App "veröffentlicht" werden. Veröffentlich ist beim Alpha-Test nicht das richtige Wort, die App wird vielmehr auf die Google Play Plattform bereitgestellt und kann wie definiert nur von Personen gefunden und heruntergeladen werden, welche sich in der Liste befinden und dazu berechtigt sind. Nach einigen Stunden ist die App dann bereit für die Installation.

16VIRTUELLES OS X

Da mir persönlich kein Mac zur Verfügung steht, musste ich auf eine andere Variante zurückgreifen. Ich entschied mich, ein OS X auf eine virtuelle Harddisk unter Windows zu installieren. Dazu brauche ich das Programm VMware, mit welchem man dem Betriebssystem eine weitere Festplatte vorgaukeln kann, auf dem dann ein weiteres Betriebssystem installiert werden kann.

Abbildung 67 Betriebssystem OS X in VMware

Obwohl das Betriebssystem OS X nur für Apple-Produkte vorgesehen ist, funktioniert diese Lösung überraschend gut. Einzig die Tastatur ist eine Umgewöhnung. Die Programme, wie Xcode oder Google Chrome funktionieren einwandfrei. Alle Bedingungen zum Entwickeln für iOS können in der **Dokumentation** von Cordova angesehen werden.

17ÜBERNAME DES ANDROID-PROJEKTS AUF IOS

Nun, da die Version für Android mehr oder weniger fertig ist, können wir die bisherigen Projekt-Files auf die Plattform iOS übernehmen. Bevor wir aber die App auf einem Apple-Gerät testen können muss noch einige Einstellungen gemacht werden.

17.1 Provisioning Profile vorbereiten und installieren

Wie bereits angemerkt wird die Sicherheit bei Apple sehr gross geschrieben. Daher gibt es auch einen Mehraufwand bei der Vorbereitung für diese Plattform.

Bisher haben wir einen Account beim Apple Developer Program erstellt. Jetzt folgen noch einige weitere Schritte, bis man die App auf einem Gerät installieren oder auf dem Apple Store veröffentlichen kann.

17.1.1 Ein iOS Development -Zertifikat anfordern

Um ein iOS Development-Zertifikat anzufordern, muss das iPhone mit dem Mac verbunden sein. Nachdem Xcode startet, gelangt man über das Menu >> Preferences >> Account zu den angemeldeten Accounts bei Xcode.

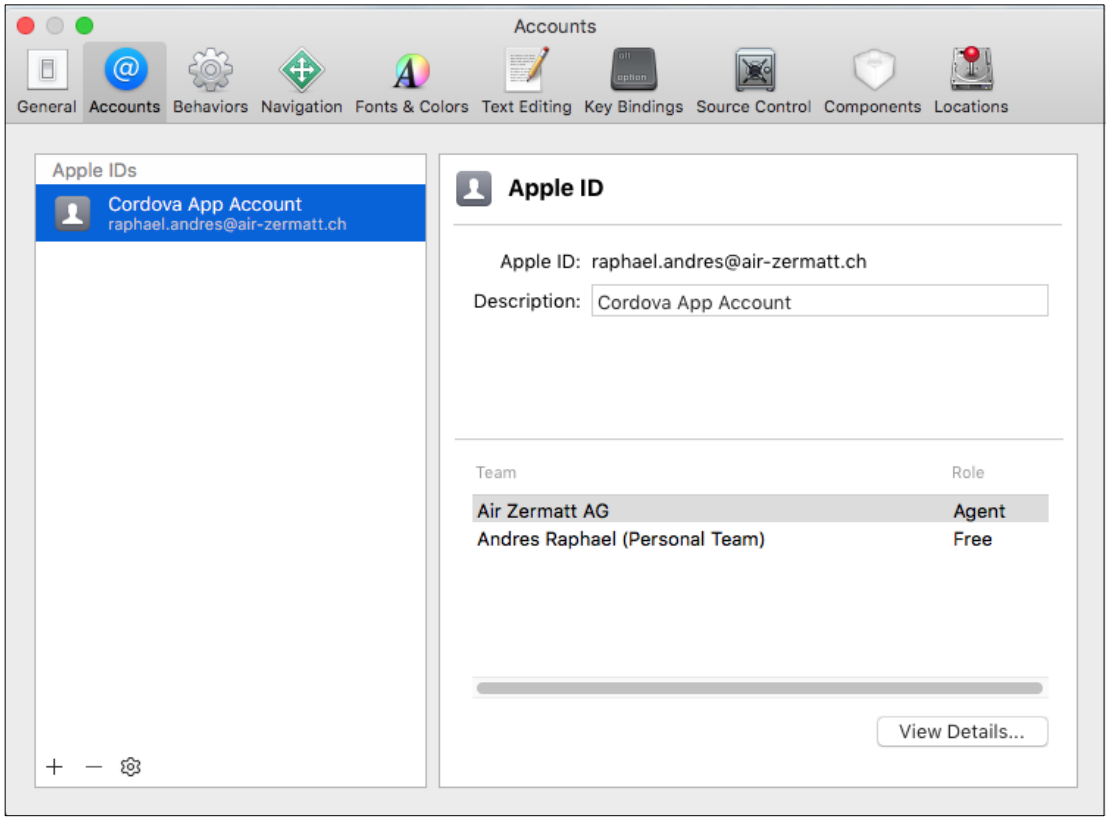


Abbildung 68 Xcode Einstellung Accounts

Über den Button View Details findet man die hinterlegten Signierungs-Identitäten, die beim ausgewählten Account erstellt werden können. Hier erstellt man ein Zertifikat für iOS Development.

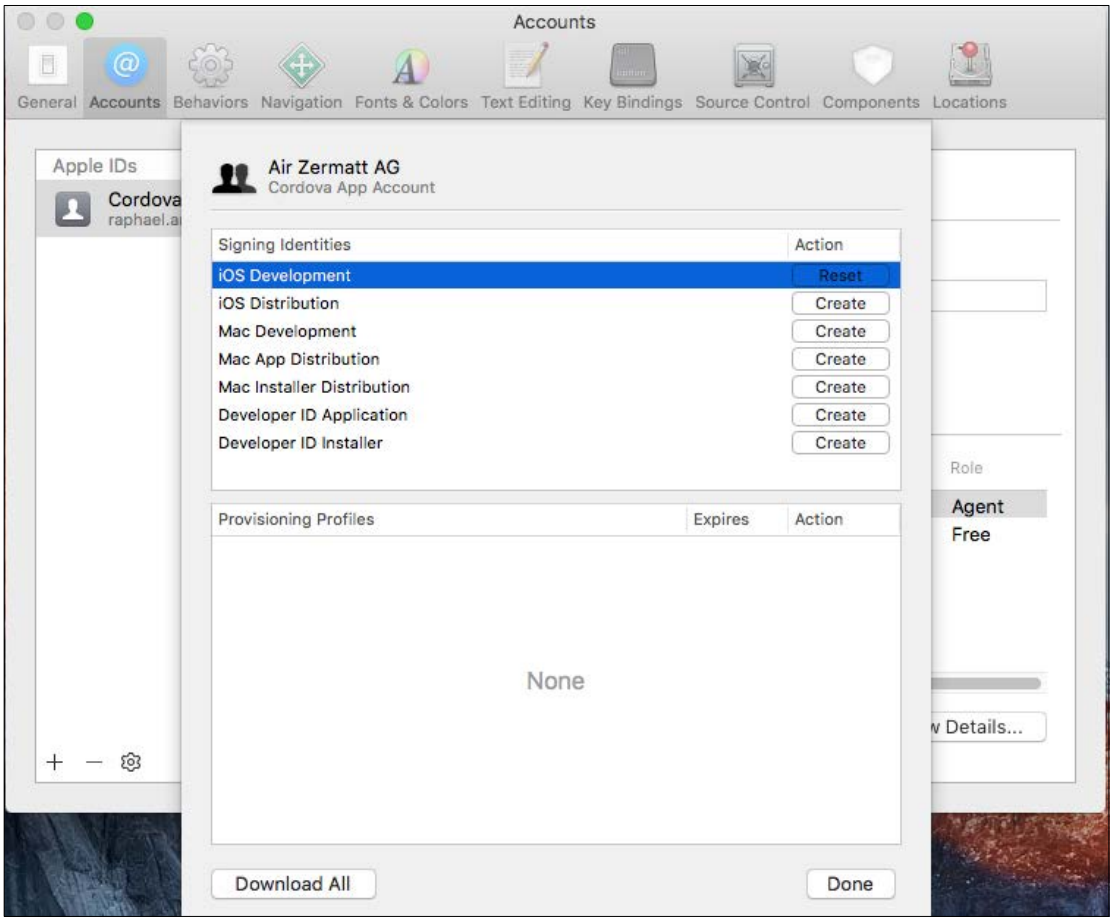


Abbildung 69 Signierungsidentitäten des Accounts

17.1.2 Erstellen einer App ID

Diese App ID welche nachfolgend erstellt wird, habe ich wieder gelöscht, weil ich später eine neue App ID erstellen werde, die für Push Benachrichtigungen geeignet ist. Siehe dazu Kapitel Zertifikate für Apple Push Notification Service erstellen.

Nun loggt man sich online bei Apple Developer an und erstellt, falls dies nicht automatisch geschehen ist, eine Explicit App ID. Wählt man eine Wildcard App ID ist man später nicht in der Lage, Push Benachrichtigungen zu versenden!

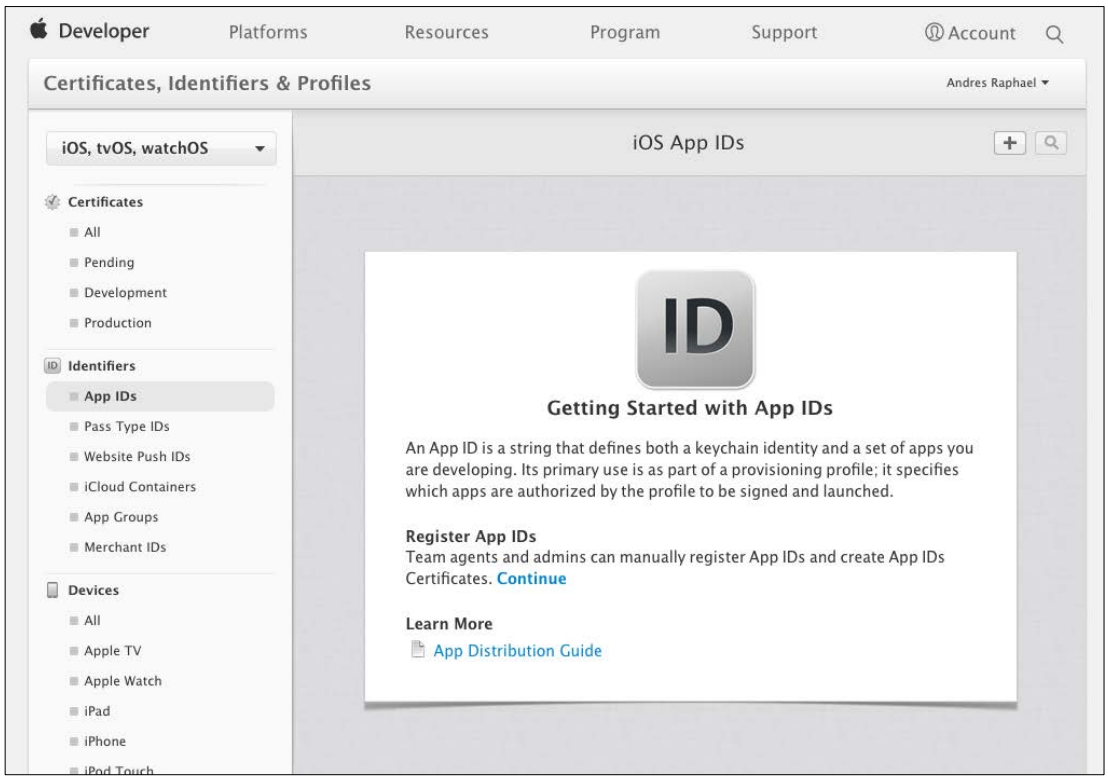


Abbildung 70 Apple Developer App IDs

Mit dem Plus auf der oberen rechten Seite kann eine neue App ID hinzugefügt werden. Als nächstes gibt man einen Namen für die App ID ein und wählt ein Suffix. Eine Wildcard App ID ist zwar flexibler, wenn man mehre Apps erstellen will. Allerdings kann man damit keine Push-Benachrichtigungen versenden. In diesem Beispiel habe ich mich für eine Windcard entschieden. In einem folgenden Kapitel erstelle ich eine Explicit App ID, die zum Versenden der Push-Benachrichtigung, geeignet ist.

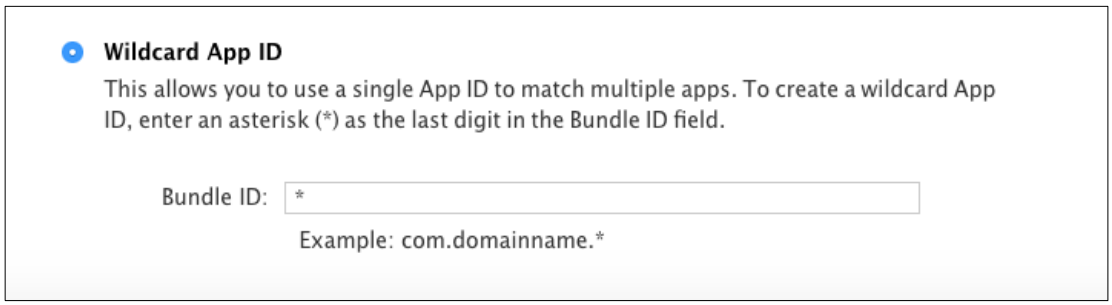


Abbildung 71 Auszufüllende Felder App ID

Nach erfolgreichem Registrieren der App ID erscheint der Eintrag in der Übersicht.

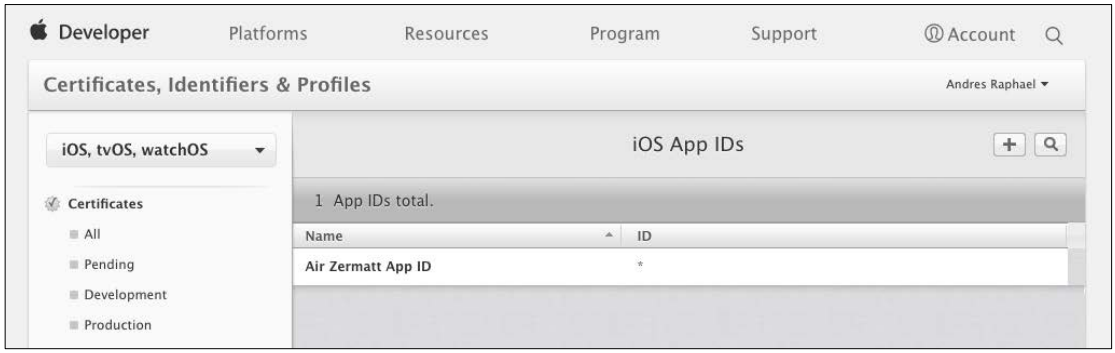


Abbildung 72 Übersicht über alle App IDs

17.1.3 Ein Apple-Gerät registrieren

Analog zur App ID kann auch ein Apple-Gerät registriert werden.

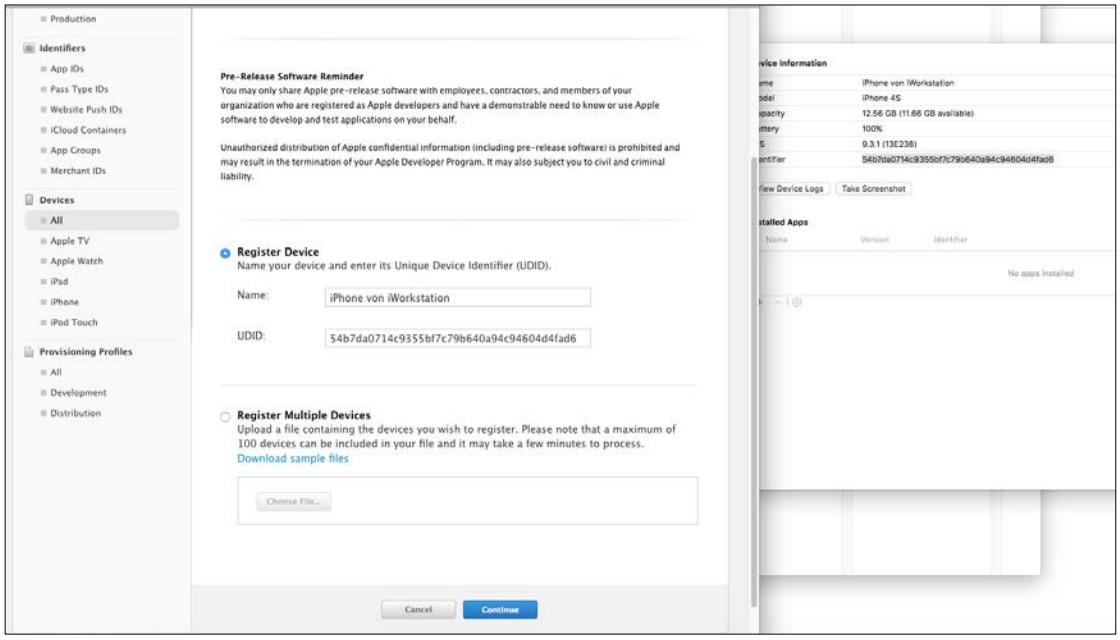


Abbildung 73 Registation eines Apple-Gerätes

17.1.4 Provisioning Profile erstellen

Um ein Provisioning Profile zu erstellen muss man nun lediglich das Profile in der Schritt-für-Schritt-Anleitung konfigurieren. Dann wird das Profil generiert und steht bereit zum Download.

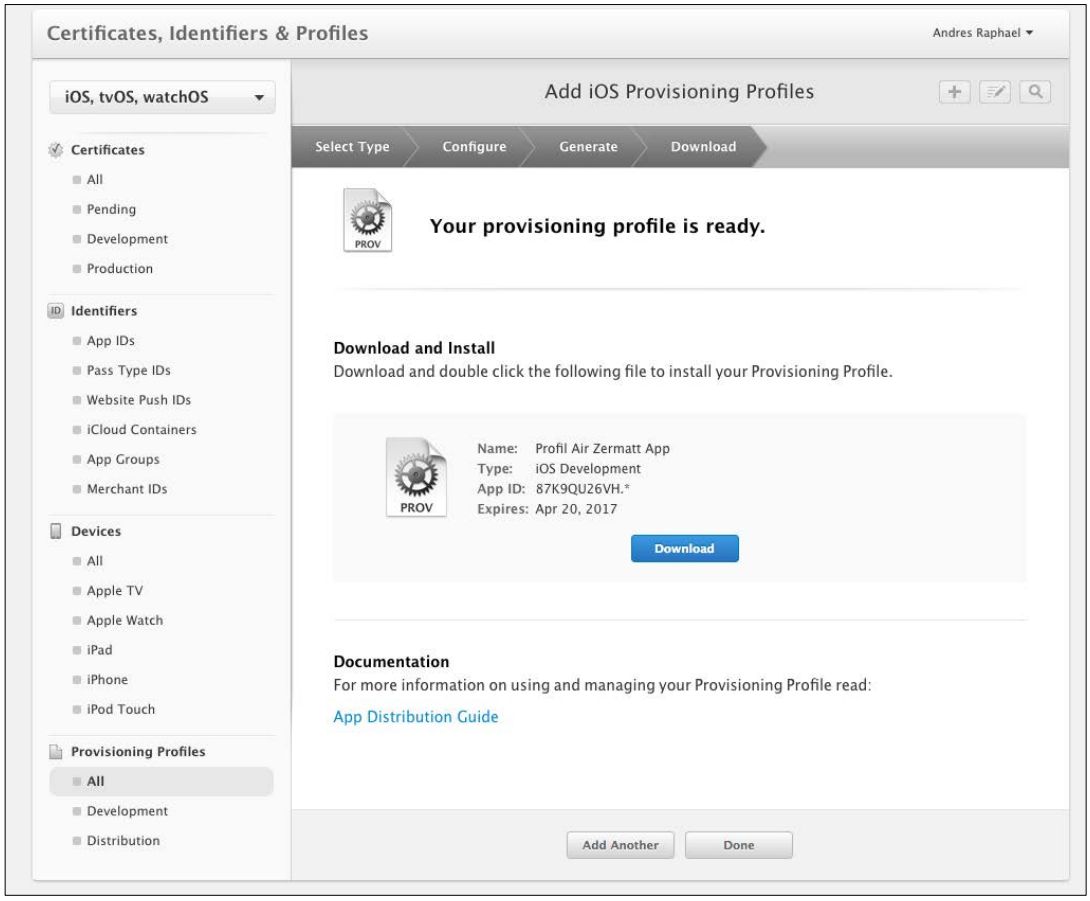


Abbildung 74 Provisioning Profile zum Download bereit

17.1.5 Download und Installation auf dem Gerät

Vielleicht ist es notwendig, dieses Profil auf einem registrierten Apple-Gerät zu installieren. Dann kann man folgende Schritte tätigen. Meistens sollte dies aber automatisch beim Installieren der App passieren.

Wie schon zuvor öffnen wir die Einstellungen von Xcode und wählen die Accounts aus. Unter View Details wird nun bei Provisioning Profile das gerade erstellte Profil angezeigt und zum Download angeboten.

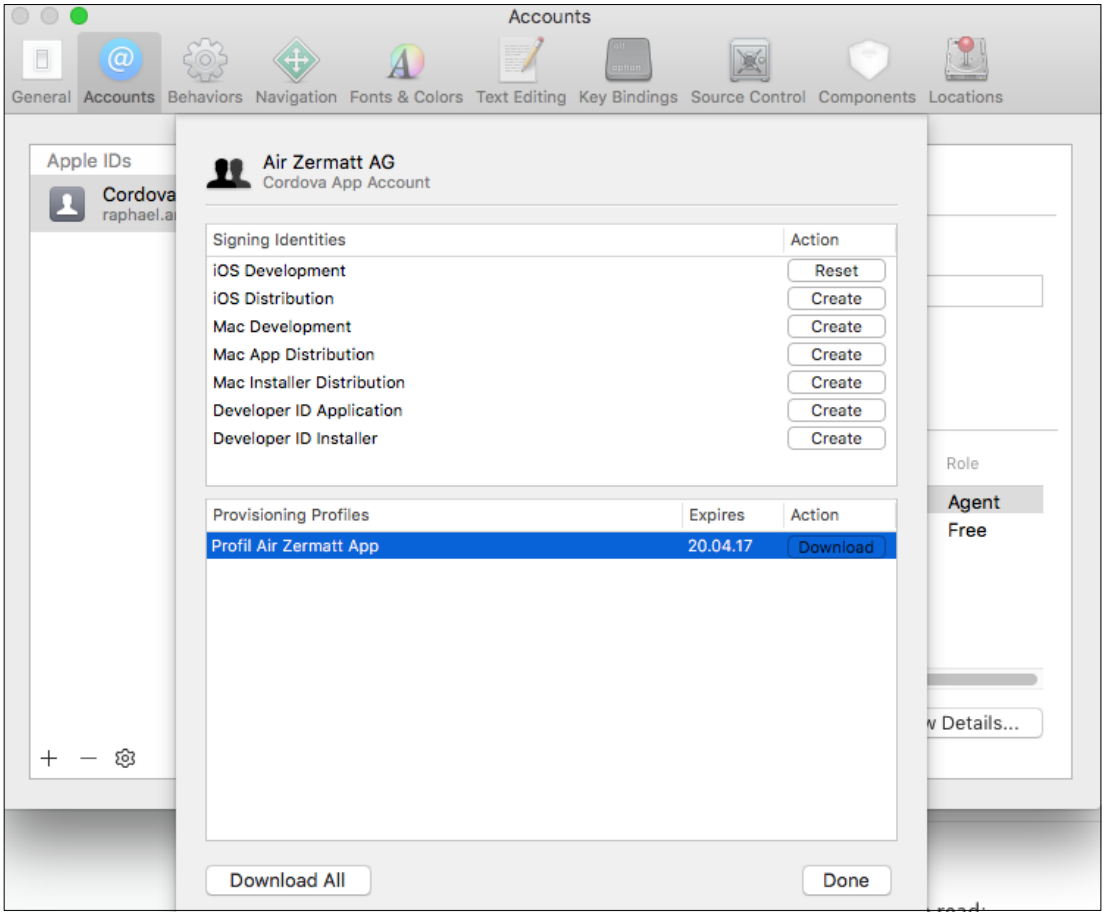


Abbildung 75 Download von Provisioning Profile

Danach kann unter Window >> Devices das angeschlossene Gerät ausgewählt werden. Darunter können die Profile aufgelistet werden, die auf dem ausgewählten Gerät installiert sind.

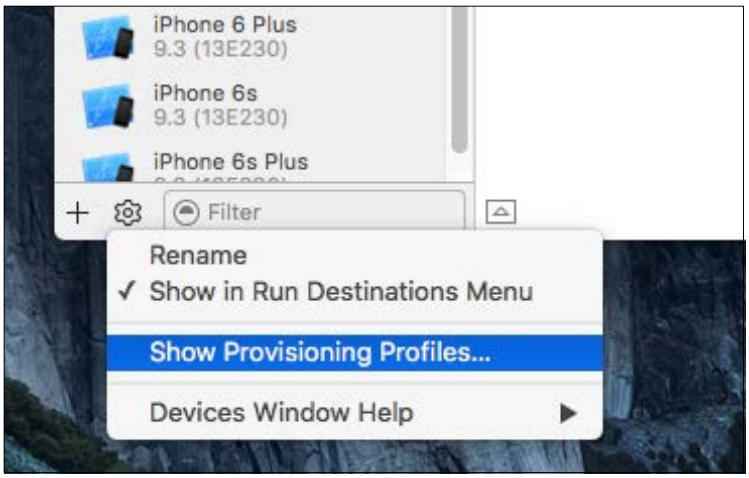


Abbildung 76 Anzeigen der Provisioning Profiles

Über das Plus auf der linken unteren Seite kann das heruntergeladene Profil ausgewählt und installiert werden.

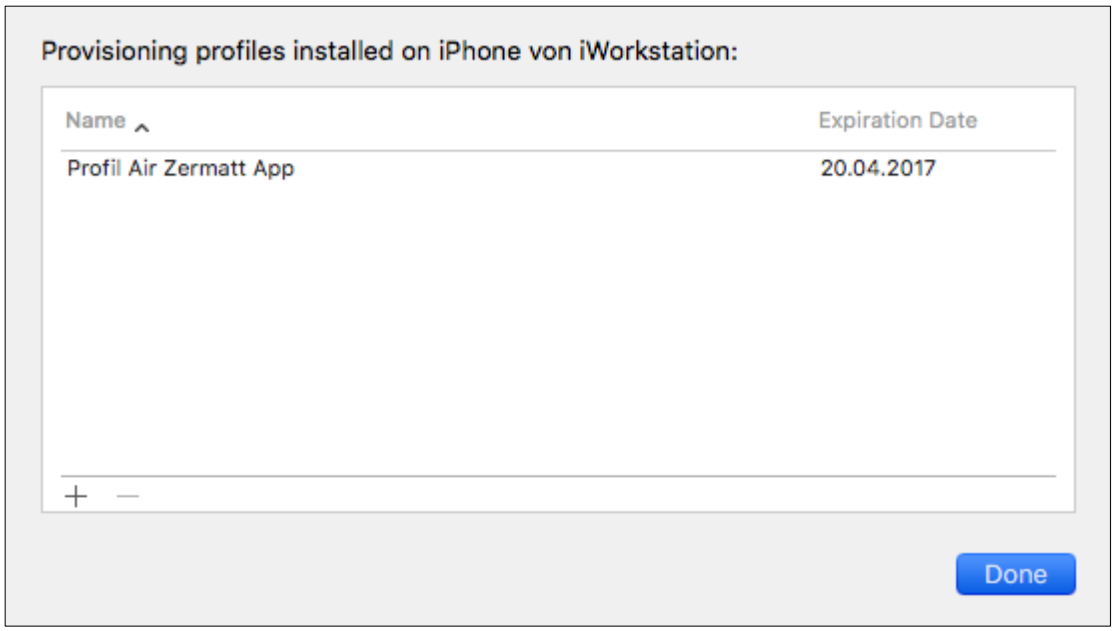


Abbildung 77 Installierte Profile auf einem Gerät

17.1.6 Ein Apple-Gerät zum Profil hinzufügen

Möchte man ein weiteres Gerät zum bestehenden Profil hinzufügen, muss das Gerät registriert werden und kann dann durch Bearbeitung des Profils hinzugefügt werden. Anschliessend kann die erneuerte Version heruntergeladen und auf dem Gerät installiert werden.

17.2 Zum Apples Push-Provider wechseln

Nachdem ich das www-Verzeichnis von Android zu iOS kopiert habe, wollte ich schon mal versuchen, die App für iOS zu builden und auf einem iPhone zu installieren. Zu meinem Verblüffen stellte ich fest, dass fast alle Funktionen einwandfrei funktionierten. Lediglich die Push-Benachrichtigung funktionierte nicht, was nicht weiter verwunderlich war, da bisher der Payload serverseitig lediglich nur an Android-Geräte versendet wurde.

17.2.1 Zwischen Android und iOS unterscheiden

Damit wir beim Versenden des Payload zwischen der Registration-ID von Android und deren von Apple unterscheiden können, speichern wir zusätzlich das Betriebssystem des Gerätes, das sich anmeldet. Dazu benötigt wird das Cordova-Plugin **Device**. Damit lesen wird relevante Informationen des Gerätes und speichern sie auf unserem Server. Installiert wird das Plugin mit dem Befehl:

```
1  cordova plugin add cordova-plugin-device
```

Beim ersten Start werden nun die Informationen zum Gerät herausgelesen und gespeichert.

```
1  db.transaction(function(tx) {
2  tx.executeSql("SELECT platform FROM tbl_benutzer", [],
   function(tx, res) {
3  if(res.rows.item(0).platform == '') {
4
5  var device = $cordovaDevice.getDevice();
6  console.log(JSON.stringify(device));
7
8  tx.executeSql("UPDATE tbl_benutzer SET cordova = ?, manu-
   facturer = ?, model = ?, platform = ?, version = ?;",
   [device.cordova, device.manufacturer, device.model, de-
   vice.platform, device.version], function() {
9  $rootScope.$broadcast('saveInfos', []);
10 },
11 function(error) {
12 console.log('Error: ' + JSON.stringify(error));
13 });
14
15 } else {
16 console.log("Device Infos bereits gespeichert: " +
   res.rows.item(0).platform);
17 }
18
19 },
20 function(error) {
21 console.log(error);
22 });
23 });
```

Danach werden die Informationen zu unserem Server übermittelt. Nun wissen wir also, mit was für einem Gerät wir kommunizieren möchten.

17.2.2 Zertifikate für Apple Push Notification Service erstellen

Wie wir das bei Apple mittlerweile gewohnt sind, brauchen wir zum Versenden von Push-Benachrichtigungen einige Zertifikate. Dieser Vorgang ist im Vergleich zur Plattform Android relativ kompliziert. Es ist also wichtig, dass man eine gut verständliche Anleitung findet. Ich habe diese auf der Seite **quickblox.com** gefunden. Diese Anleitung ist schon etwas älter, und das Layout des Apple Developer Center hat sich mittlerweile geändert, aber die Arbeitsschritte sind noch immer dieselben. Die Dienstleistungen von Quickblox werden zwar nicht verwendet, trotzdem empfehle ich diese Anleitung zu verwenden.

Als erstes muss sichergestellt sein, dass die Geräte, auf dem man die App später testen will, registriert sind. Siehe dazu Kapitel Ein Apple-Gerät registrieren. Dann erstellt man die App ID. Dazu öffnet man online das Apple Developer Portal und navigiert zur erstellen App ID.

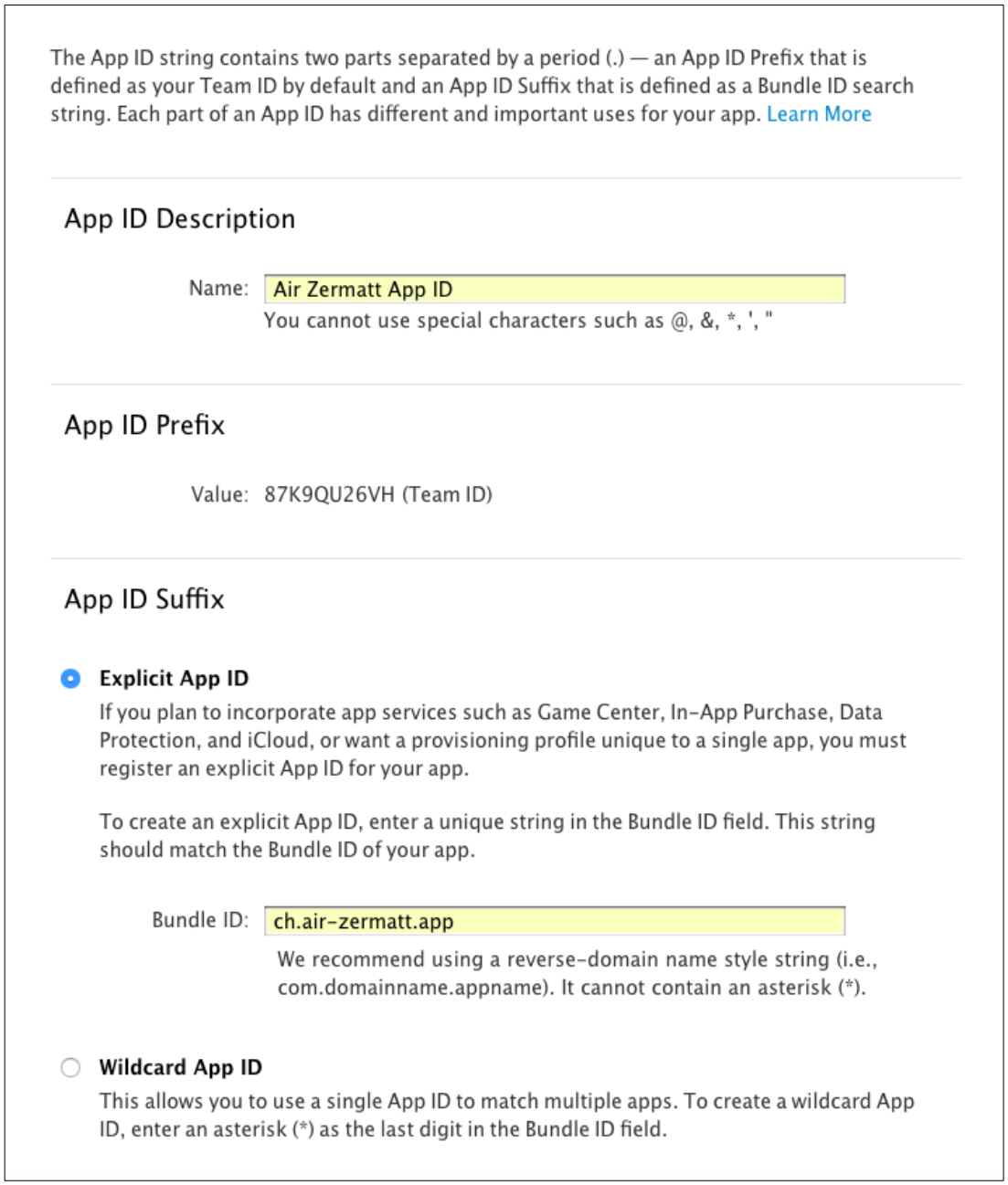


Abbildung 78 Erstellen einer Explicit App ID

Wichtig ist hier, dass man *Explicit App ID* auswählt und dass die Bundle ID, wie beschrieben, mit der Bundle ID der App übereinstimmt. In diesem Fall ist die Bundle ID *ch.air-zermatt.app*.

Danach kann man fortfahren und die App ID in der Übersicht ansehen. Nun bearbeiten wir die gerade erstellte App ID, um die Push Benachrichtigung zu aktivieren.

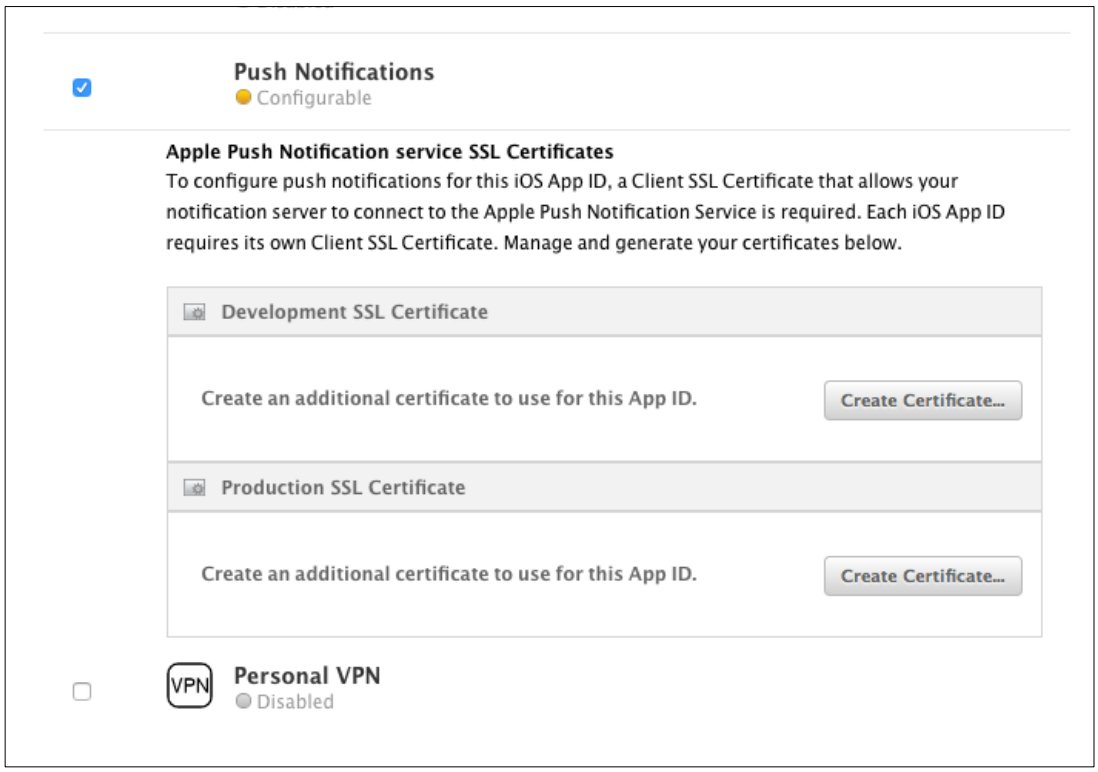


Abbildung 79 Development SSL Certificate erstellen

Beim Apple Push Notification Service wird zwischen Testphase und Produktionsphase unterschieden. Daher können auch zwei separate Zertifikate dazu erstellt werden. Zum Testen erstellen wir ein *Development SSL Certifikate*.

Drückt man den Button *Create Certificate* gelangt man zu einer Anleitung, wie man auf seinem Mac ein neues Zertifikat anfordern kann. Dazu öffnet man das Dienstprogramm *Schlüsselbundverwaltung* auf dem Mac. Im Menü klickt man auf Schlüsselbundverwaltung >> Zertifikatsassistent und dort wählt man *Zertifikat einer Zertifizierungsinstanz anfordern* aus.

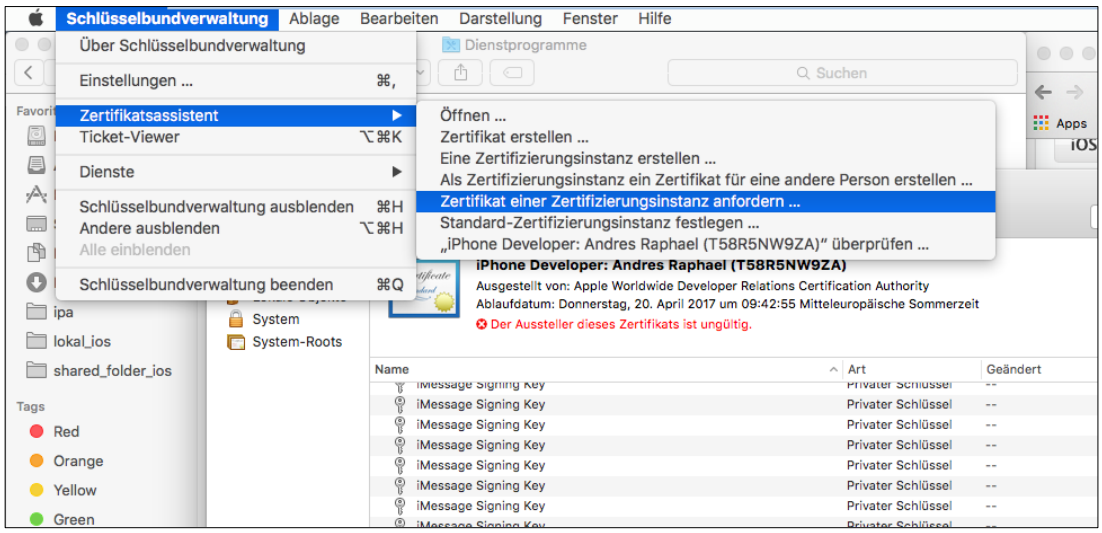


Abbildung 80 Zertifikat anfordern

Als nächstes öffnet sich ein Fenster in dem man Informationen zum Zertifikat hinterlegen muss.

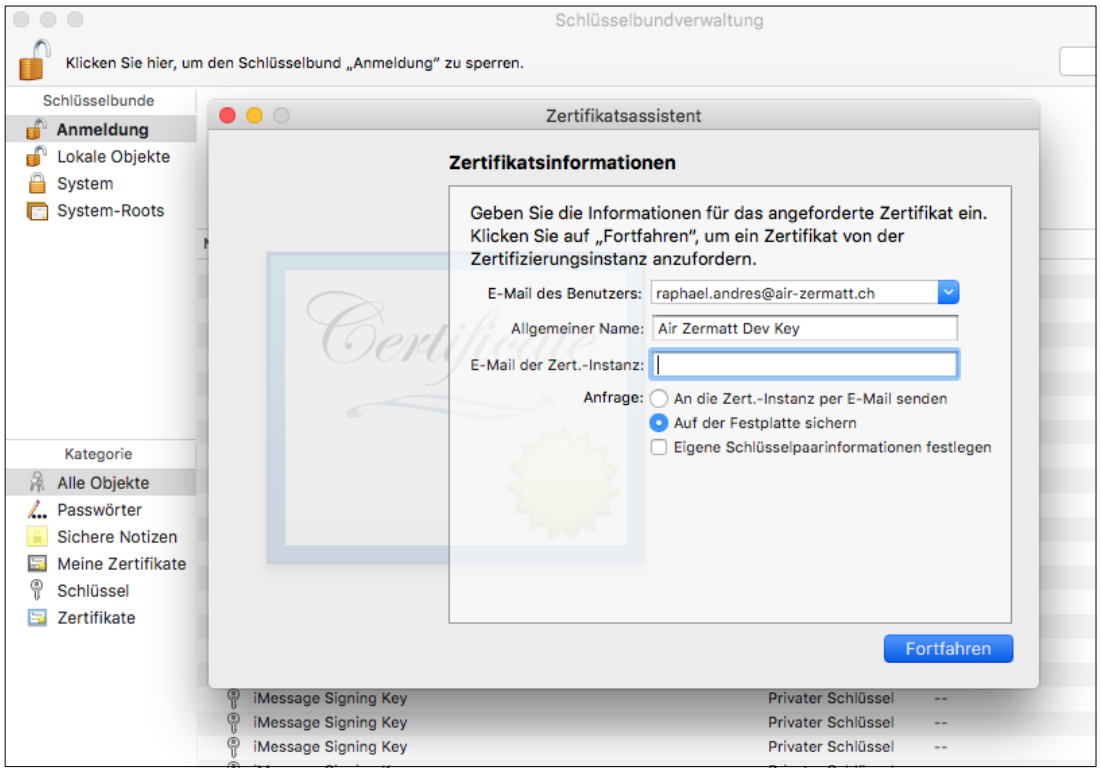


Abbildung 81 Zertifikatsinformationen hinterlegen

Dieses Zertifikat kann auf der lokalen Festplatte gespeichert werden. Klickt man auf Fortfahren wird eine Datei mit dem Namen *CertificateSigningRequest.certSigningRequest* gespeichert.

Zurück im Apple Developer Portal wird diese CSR-Datei hochgeladen. Wurde die Datei hochgeladen, wird das eigentliche Apple Development iOS Push Services-Zertifikat von Apple generiert und steht dann zum Download bereit.

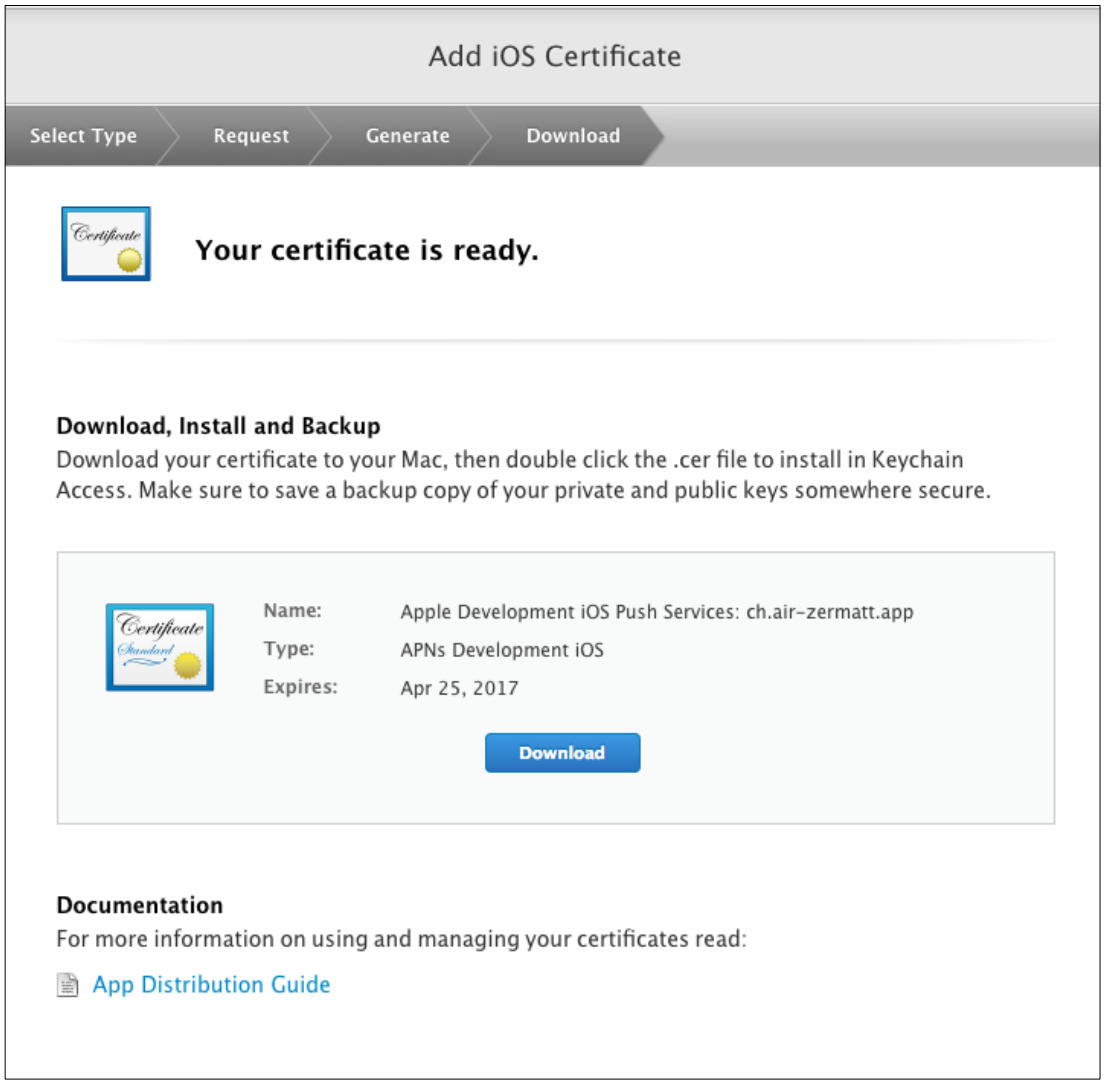


Abbildung 82 Download des Apple Development iOS Push Services-Zertifikat

Als nächster Schritt öffnet man das eben heruntergeladene Zertifikat wiederum in der Schlüsselbundverwaltung. Unter dem Punkt *Meine Zertifikate* sollte nun das installierte Zertifikat erscheinen. Mit dem kleinen Pfeil an der linken Seite kann man den Eintrag erweitern. Der Private Schlüssel mit dem zuvor vergebenen Namen wird sichtbar. Nun muss man beide Elemente markieren und via Rechts-Klick exportieren. Diese Datei nenne ich *apns_dev_cert.p12* und speichere sie in den Dokumenten.

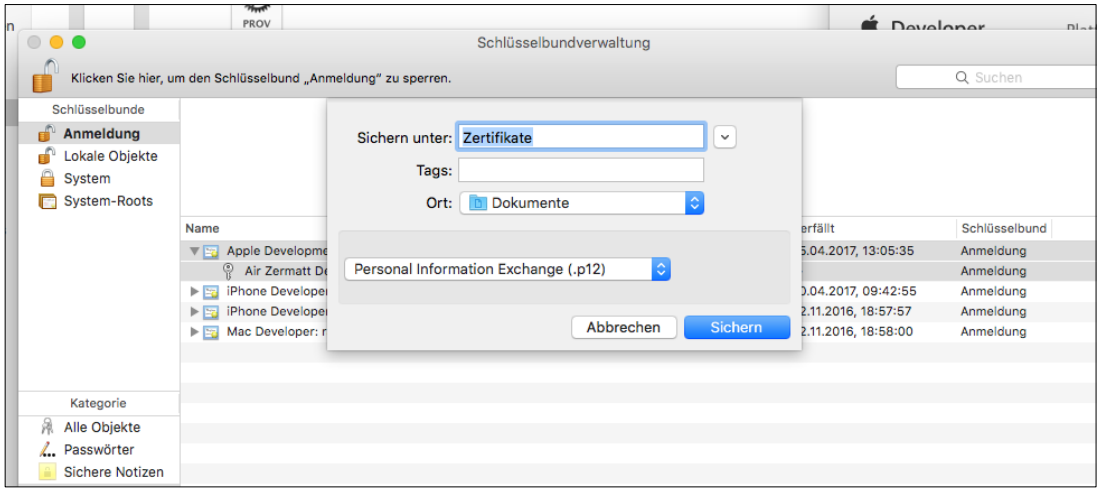


Abbildung 83 Zertifikat als .p12-Datei exportieren

Wichtig ist, dass man dieser Datei kein Passwort gibt, obwohl der Dialog dazu erscheint! Man kann also ohne Passwort-Eingabe fortfahren. Zum Exportieren muss aber das lokale Passwort des Benutzers angeben.

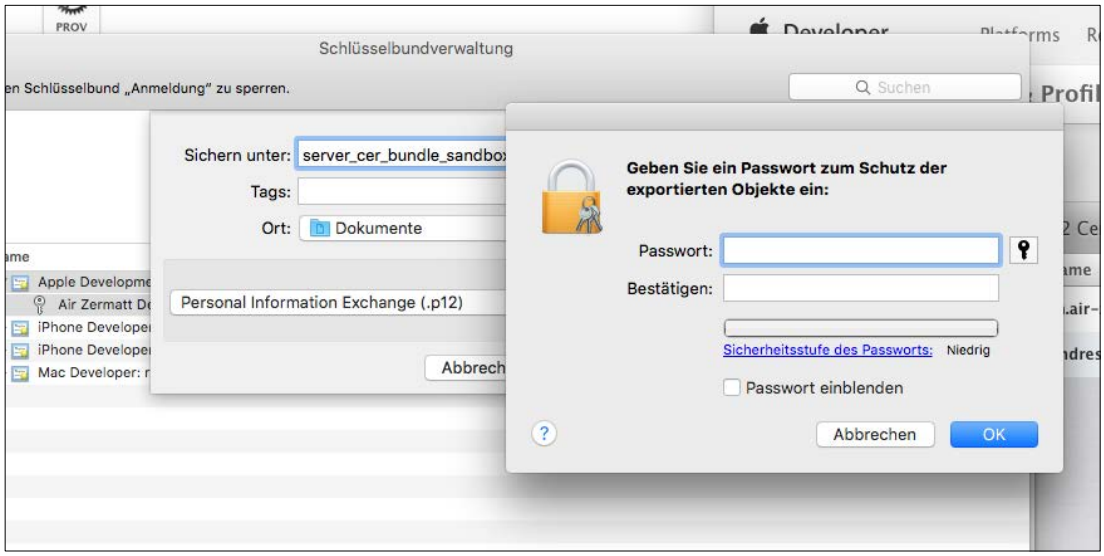


Abbildung 84 Passwort-Dialogfenster

Diese Datei speichere ich wiederum in den Dokumenten. Nun muss man diese .p12-Datei noch konvertieren, damit wir sie auf unseren Server hochladen können. Dazu öffne ich den Terminal am Speicherort und tippe folgenden Befehl ein:

```
1  openssl pkcs12 -in apns_dev_cert.p12 -out
   apns_dev_cert.pem -nodes -clcerts
```

Dies erstellt die Datei *apns_dev_cert.pem*, welche man serverseitig benötigt um den Payload an den APNS zu senden. Diese Datei lade ich später auf den Server hoch.

Als nächstes erstellt man das Provisioning Profile. Dies funktioniert analog zum Kapitel Provisioning Profile erstellen. Man wählt den Typ iOS App Development aus. Dann die zuvor erstellte App ID mit der Bundle ID ch.air-zermatt.app. Daraufhin wählt man das persönliche Zertifikat aus. In meinem Fall Andres Raphael (iOS Development). Zum Schluss können die Geräte ausgewählt werden, auf die man die App installieren und testen möchte. Schlussendlich kann das Profile benennt und generiert werden. In der Übersicht erscheint das Profile mit seinen Informationen:

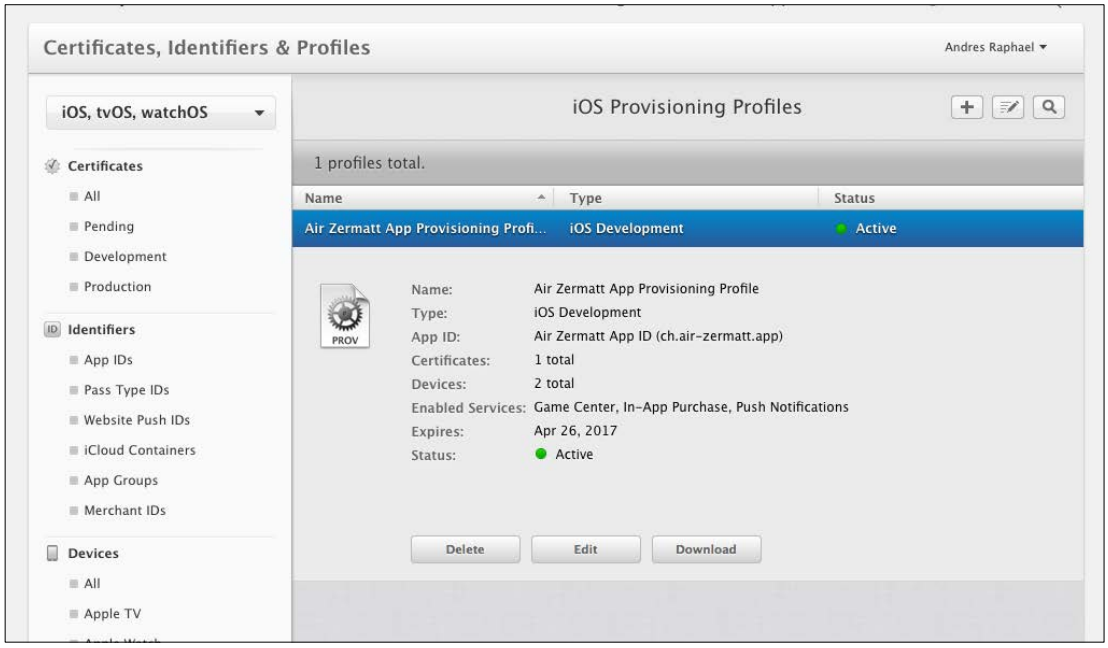


Abbildung 85 Air Zermatt App Provisioning Profile

Dieses Profile fügen wir nun wieder zum Xcode hinzu danach sind wir schon fast fertig! Nun müssen wir nämlich noch überprüfen, dass die App die richtige Bundle ID hat.

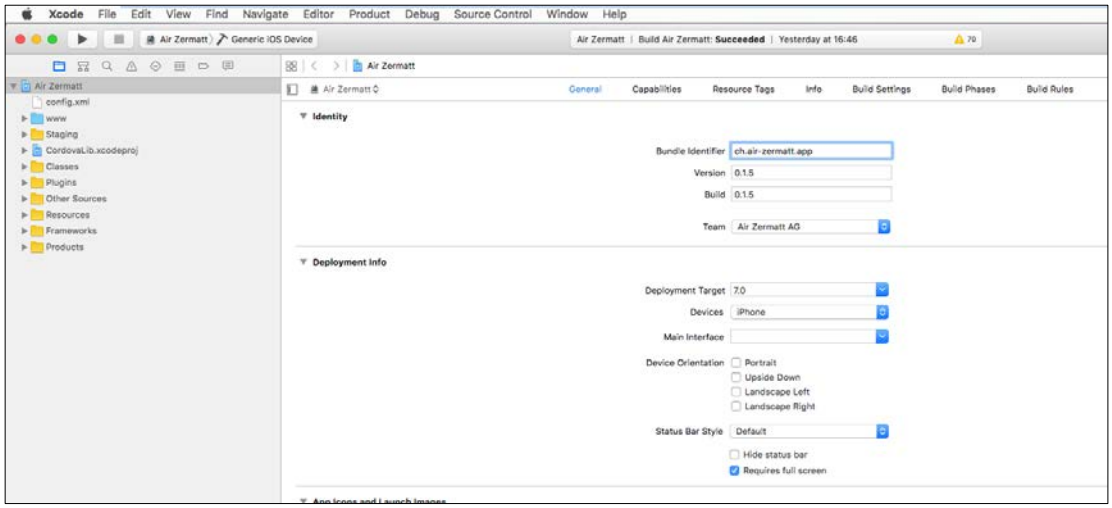


Abbildung 86 Überprüfung der Bundle ID der App

Zum Schluss muss darauf geachtet werden, dass die App mit dem richtigen persönlichen Zertifikat signiert wird. Dazu wählt man das Target des Projekts aus, klickt auf den Menüpunkt Build Settings und findet unter dem Bereich Code Signing die Einstellung Code Signing Identity. Hier ist wichtig, dass das persönliche Zertifikat ausgewählt ist.

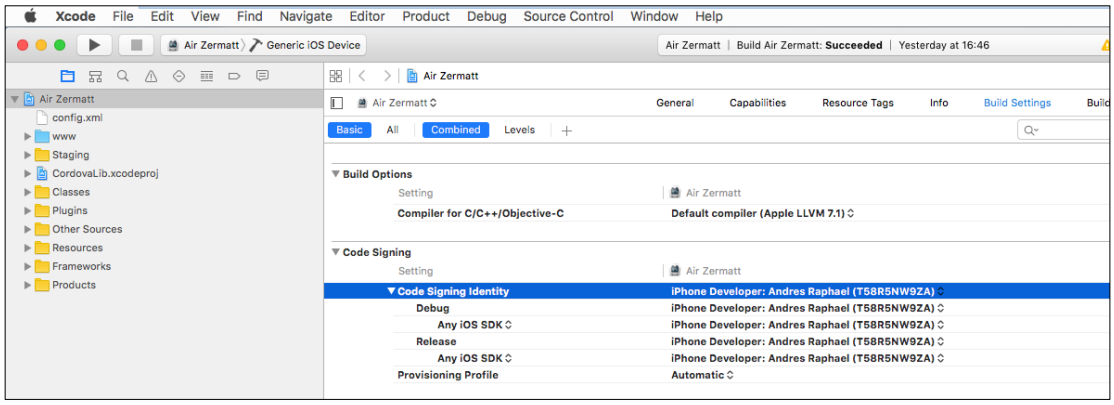


Abbildung 87 Überprüfung der Code Signing Identity

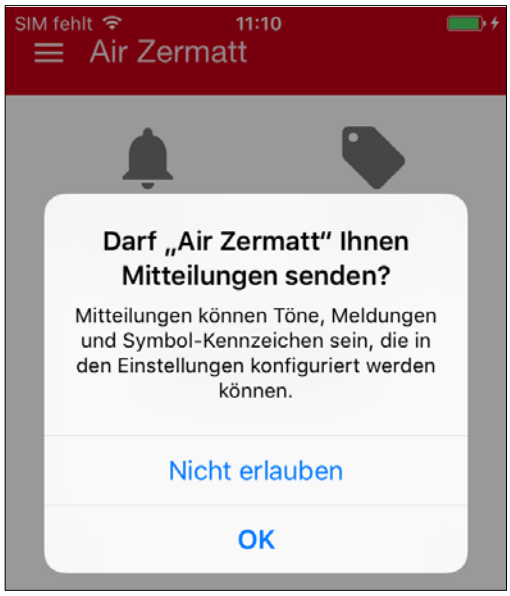


Abbildung 88 Dialogfenster Push erlauben

Nun kann ein Gerät angeschlossen werden und die App darauf getestet werden. Erscheint beim Builden der App keine Fehlermeldung, hat man die vorherigen Schritte richtig gemacht. Nun sollte beim erstmaligen Aufstarten der App die Meldung kommen, ob der Benutzer Benachrichtigungen von dieser App erhalten will. Wird dies vom Benutzer bestätigt, wird im Hintergrund der APNS um ein Device Token angefragt. Dieser Token wird dann an Cordova übergeben, das den Token wiederum in unsere Datenbank speichert. Dieser Token muss bei jeder Push-Benachrichtigung angegeben werden.

17.2.3 Push Funktion für iOS testen

Zum Testen, ob die Zertifikate richtig konfiguriert sind habe ich auf dem Mac das Programm **Pusher** installiert. Damit kann ein Push-Zertifikat auf dem lokalen Computer ausgewählt werden. Hier eignet sich die Datei *apns_dev_cert.p12*. Dann muss ein Device Token angegeben werden. Hier nehme ich den Token, der in der Datenbank gespeichert wurde. Der Payload kann nach Belieben geändert werden.

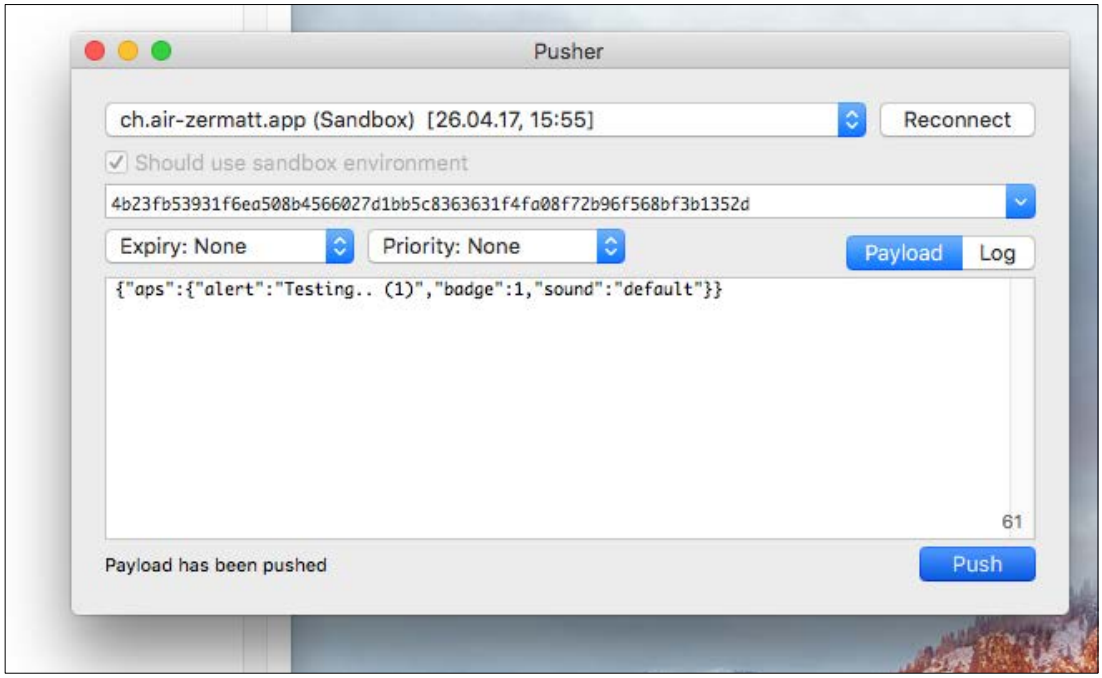


Abbildung 89 Programm Pusher zum Testen der Push-Funktion

17.2.4 Payload an APNS-Server senden

Nun schauen wir uns die benötigten Funktionen auf der Serverseite an, damit wir den Payload an den Apple Push Notification Service senden können. Wie bei Android brauchen wir für iOS die RegistrationID des Gerätes, welches wir mit dem Payload versenden müssen. Zusätzlich müssen wir aber auch noch ein Zertifikat mitsenden, welches wir weiter oben erstellt haben. Die Datei *apns_dev_cert.pem* wird auf dem Server gespeichert. Weiter müssen wir den sogenannten *passphrase* senden, der das Passwort des Zertifikats darstellt. Auf der Seite **http://codular.com/sending-ios-push-notifications-with-php** habe ich eine gute Variante gefunden, wie man den Payload via PHP an den Apple-Server senden kann. Ich habe das erwähnte Beispiel etwas angepasst, um die Payloads an alle Geräte zu senden, die in unserer Datenbank hinterlegt sind:

```
1  // Put your private key's passphrase here:
2  $passphrase = '';
3
4  $ctx = stream_context_create();
5  stream_context_set_option($ctx, 'ssl', 'local_cert',
   '../cert/apns_dev_cert.pem');
6  stream_context_set_option($ctx, 'ssl', 'passphrase',
   $passphrase);
7
8  // Open a connection to the APNS server
9  $fp = stream_socket_client(
```

```
10 'ssl://gateway.sandbox.push.apple.com:2195', $err,
11 $errstr, 60, STREAM_CLIENT_CONNECT|STREAM_CLIENT_PERSIS-
   TENT, $ctx);
12
13 // Encode the payload as JSON
14 $payload = json_encode($body);
15
16 $query = mysqli_query($con, "SELECT registration_id FROM
   tbl_kunde WHERE platform = 'iOS'");
17 while($row = mysqli_fetch_array($query)) {
18 $devices[] = $row["registration_id"];
19 }
20
21 foreach($devices as $device) {
22 // Build the binary notification
23 $msg = chr(0) . pack('n', 32) . pack('H*', $device) .
   pack('n', strlen($payload)) . $payload;
24
25 // Send it to the server
26 $result = fwrite($fp, $msg, strlen($msg));
27
28 if (!$result)
29   echo 'Message not delivered' . PHP_EOL;
30 else
31   echo 'Message successfully delivered to ' . $device .
   PHP_EOL . "</br>";
32
33 }
34
35 // Close the connection to the server
36 fclose($fp);
```

Dieser Code-Snippet bereitet einen PHP Stream vor, fügt die beiden Optionen für das Zertifikat und die Passphrase ein. Dann wird die Verbindung mit dem Apple-Server vorbereitet. In der Variable $payload werden die ganzen Informationen in ein JSON Objekt codiert. Danach werden alle RegistrationIDs geladen, die einem Apple-Gerät angehören und die die Push-Benachrichtigung erhalten möchten. In einer *foreach*-Schlaufe wird dann die eigentliche Nachricht kreiert und mit einer *fwrite()*-Funktion gesendet. Nachdem der Payload an alle Geräte gesendet wurde, wird die Verbindung zum Apple-Server wieder geschlossen.

Wichtig! Beim Umstieg auf das Distribution-Zertifikat hat sich herausgestellt, dass die Push Benachrichtigungen auf den Geräten nicht ankommen, weil in der Datenbank noch Token vom Developer-Zertifikat gespeichert waren. Nachzulesen in diesem **Forumeintrag**.

17.3 Anpassungen am Design für iOS

17.3.1 Das verkaufte Framework Chocolate Chip UI

Als ich nun das Design der iOS-App mit dem Framework Chocolate Chip UI anpassen wollte, habe ich bemerkt, dass die Website des Frameworks gar nicht mehr existiert. Meine Recherche ergab, dass dieses OpenSource-Framework von einer Unternehmung mit dem Namen Sourcebits aufgekauft wurde. Nun steht es also nicht mehr jedem kostenlos zur Verfügung. Das ist schade, da ich damit das Design für iOS und Windows anpassen wollte. Nun muss ich auf ein anderes Framework ausweichen.

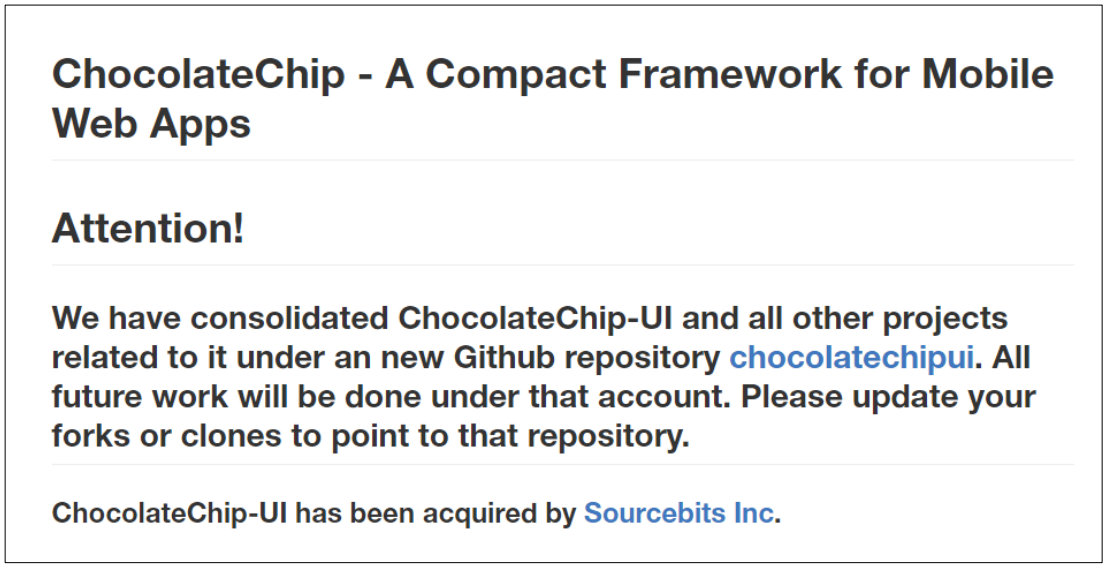


Abbildung 90 Hinweis, dass ChocolateChip erworben wurde

17.3.2 Plan B: Framework7

Als Plan B habe ich das Framework **Framework7** ausgewählt. Damit kann ich aber unglücklicherweise nur die Plattform iOS gestalten und muss für Windows wieder ein anderes Framework finden. Leider habe ich bis jetzt noch kein Framework gefunden, das alle 3 Plattformen zufriedenstellend unterstützt. Mit Framework7 (auch F7 genannt) habe ich ein Framework gefunden, das aus meiner Sicht vom Design her am nächsten an das native Design von iOS kommt. Es ist einfach aufgebaut und auch die **Dokumentation** ist sehr ausführlich und gut erklärt.

Leider hat sich beim Entwickeln für iOS herausgestellt, das sich die beiden Frameworks Framework7 und AngularJS nicht sehr mögen. F7 verfügt über eine eigene Routing-Engine, welche dringend empfohlen wird, wenn mit AngularJS gearbeitet wird. Ich habe also in einem ersten Versuch den RouteProvider von AngularJS entfernt und mit der AJAX-Lösung von F7 gearbeitet. Das Problem hierbei war, dass der Controller einer Unterseite nicht zugänglich war, wenn man zu dieser navigierte, da Angular nichts von diesem Wechsel im DOM mitbekommen hat.

Noch einer langwierigen Recherche habe ich einen **Blog-Beitrag** gefunden, in dem eine Lösung für dieses Problem formuliert war.

```
1  angularApp.controller('YourController', ['$scope', '$com-
   pile', function ($scope, $compile) {
2  $$(document).on('pageInit', function (e) {
3  $compile(e.srcElement)($scope);
4  $scope.$apply();
5  });
6  }]);
```

Dieser Code-Snippets erkennt, wenn eine neue (Unter-)Seite initialisiert wird, durchsucht den neuen DOM nach Controller und fügt diese zur Angular-Logik hinzu. Dies ist eine gute Lösung für Projekte, welche sich auf das Web begrenzen. Da wir aber bei unserer Mobile-App auf native Elemente zurückgreifen müssen, müssen wir den *deviceready*-Event abwarten, um auf alle Funktionen zugreifen zu können. Dies wird bei der obigen Lösung leider nicht gemacht. Also musste ich weitersuchen.

Schlussendlich habe ich eine **akzeptable Lösung** gefunden! Bei Framework7 hat man die Möglichkeit alle Seiten als *Inline Pages* zu strukturieren. Dazu muss in der Konfigurationsdatei von F7 der *domCache* aktiviert werden.

```
1  var mainView = myApp.addView('.view-main', {
2  domCache: true //enable inline pages
3  });
```

Alle Unterseiten, welche wir bei Android also in einem separaten File gespeichert hatten, mussten nun also in ein und dasselbe File eingefügt werden. Dies ist natürliche sehr unübersichtliche, aber es ist die einzige Möglichkeit, dass alle Controller der Unterseiten von Beginn an zugänglich sind. So kann auch auf den *deviceready*-Event gewartet werden und wir sind sicher, dass alle Cordova-Funktionen funktionieren. Die grundlegende Struktur der index.html sieht demnach so aus:

```
1  <html>
2  <head>...</head>
3  <body>
4  ...
5  <!-- Views -->
6  <div class="views">
7  <!-- View -->
8  <div class="view view-main">
9  <!-- Navbar -->
10 <div class="navbar">
11 <!-- Startseite navbar -->
```

```
12 <div class="navbar-inner" data-page="index">
13 <div class="center">Home</div>
14 </div>
15 <!-- Kontaktinformationen navbar -->
16 <div class="navbar-inner cached" data-page="about">
17 <div class="center">About</div>
18 </div>
19 <!-- Angebote navbar -->
20 <div class="navbar-inner cached" data-page="services">
21 <div class="center">Services</div>
22 </div>
23 </div>
24 <!-- Pages -->
25 <div class="pages navbar-through">
26 <!-- Startseite -->
27 <div class="page" data-page="index">
28 <div class="page-content">
29 <p>Home page</p>
30 </div>
31 </div>
32 <!-- Kontaktinformationen -->
33 <div class="page cached" data-page="kontaktinformatio-
   nen">
34 <div class="page-content">
35 <p></p>
36 </div>
37 </div>
38 <!-- Angebote -->
39 <div class="page cached" data-page="angebote">
40 <div class="page-content">
41 <p></p>
42 </div>
43 </div>
44 </div>
45 </div>
46 </div>
47 ...
48 </body>
49 </html>
```

Die Navigationen der Unterseiten sind vor den eigentlichen Seiten zusammengefasst. Wichtig ist, das bei jeder Navbar das Attribut *data-page* mit der betreffenden Seiten angegeben ist. Dieses Element braucht zusätzliche die CSS-Klasse *cached*. Genauso alle Unterseiten. Im Menu

oder Panel, wie es in F7 heisst, muss nur der Link angepasst werden. Anstelle von *angebote.html* wird im *href*-Attribut die gewünschte Seite mit einer Raute angegeben *#angebote*.

17.3.3 Vorschau im Browser ansehen

Ich habe mir also das Framework heruntergeladen und die relevanten CSS- und JS-Dateien in mein Projekt kopiert. Damit man beim Gestalten der Seiten eine Vorschau anschauen kann, installiere ich auf dem Mac das Programm MAMP. MAMP für OS X funktioniert gleich wie XAMPP für Windows. Im Hintergrund wird ein Apache-Server gestartet und in einem Ordner htdocs können die gewünschten Dateien hinterlegt werden.

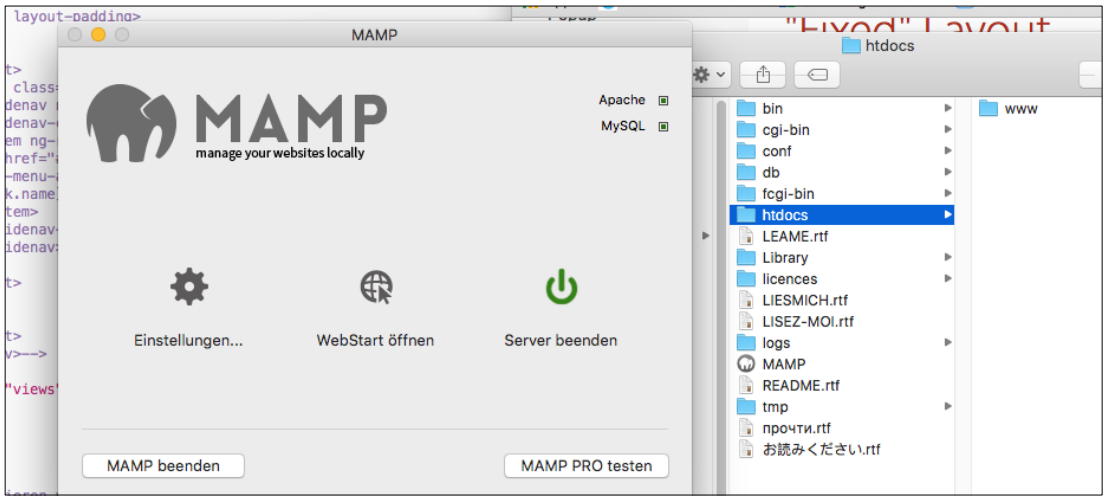


Abbildung 91 MAMP-Installation auf dem Mac mit htdocs-Ordner

Zum Starten muss man das Passwort des Benutzers eingeben. Um den Umgang mit Mamp und cordova möglichst einfach zu halten, habe ich das gesamte Cordova-Projekt in das htdocs-Verzeichnis verschoben. Damit man mit dem Terminal schnell zu diesem Verzeichnis wechseln kann, habe ich einen Alias des genannten Pfads erstellt.

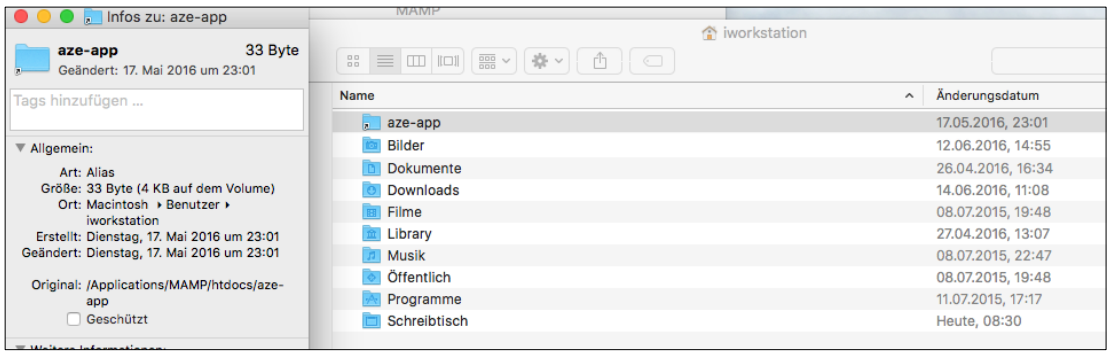


Abbildung 92 Alias zum Cordova-Projekt in den Dokumenten

Danach kann das Cordova-Projekt mit folgendem Befehl aufgerufen werden:

```
cd aze-app
```

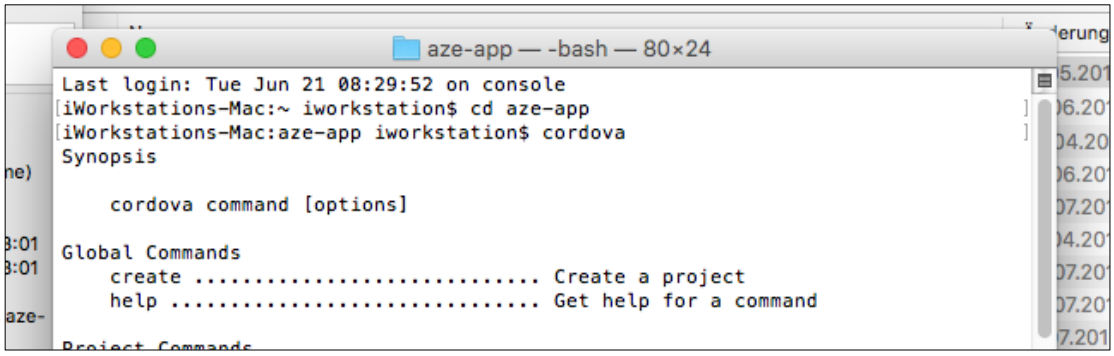


Abbildung 93 Zugang zum Cordova-Projekt im Terminal

17.3.4 Problem bei Input-Feldern

Nach der Übernahme des Projekts auf iOS bin ich auf das Problem gestossen, dass die Eingabe in die Input-Felder nicht möglich war. Es wurde immer nur der erste Buchstabe geschrieben, die restlichen tauchten nicht auf. Um dieses Problem zu lösen muss man mehrere Zeilen CSS-Code einfügen:

```
input {
-webkit-user-select: auto !important;
-khtml-user-select: auto !important;
-moz-user-select: auto !important;
-ms-user-select: auto !important;
user-select: auto !important;
}
```

17.4 Anpassungen für Landscape-Modus und Tablet

Da wir die App auch für Tablet-Besitzer optimieren möchten, müssen dort noch einige Anpassungen durchgeführt werden.

17.4.1 Responsive Masonry-Ansicht für Newsticker-Beiträge und Bilder

Einige Seiten, wie Newsticker, Angebote oder Partner enthalten sogenannte Cards, in denen die spezifischen Informationen geladen werden. Am Smartphone im Hochformat, werden diese Cards gut dargestellt. Wird aber der Bildschirm breiter, wie zum Beispiel im Querformat oder auf dem Tablet, sind die Bilder zu gross und die Informationen sind nicht mehr übersichtlich strukturiert. Um diesem Szenario entgegenzuwirken, habe ich folgende Zeilen in die style.css hinzugefügt, um die Beiträge und Bilder auf mehrere Spalten aufzuteilen:

```
/*****************************
```

```
2  /* Anpassungen für Tablet */
3
4  .wrapper_masonry {
5  -moz-column-count: 3;
6  -webkit-column-count: 3;
7  column-count: 3;
8  column-gap: 1em;
9  display: block !important;
10 }
11
12 .wrapper_masonry > * {
13 display: inline-block;
14 margin: 0 0 1em;
15 width: 100%;
16 }
17
18 @media screen and (max-width: 1024px) {
19
20 .wrapper_masonry {
21 -moz-column-count: 2;
22 -webkit-column-count: 2;
23 column-count: 2;
24 }
25
26 }
27
28 @media screen and (max-width: 590px) {
29
30 .wrapper_masonry {
31 -moz-column-count: 1;
32 -webkit-column-count: 1;
33 column-count: 1;
34 }
35
36 }
```

Die CSS-Klasse *wrapper_masonry* wird einem Container-Element zugewiesen. Dessen Kind-Elemente (meistens durch ng-Repeat generiert) werden dann in Spalten strukturiert. Die Anzahl Spalten sind darunter definiert. Je breiter der Bildschirm ist, umso mehr Spalten gibt es. Die Breakpoints und Anzahl Spalten sind auf diese Weise einfach zu verändern.

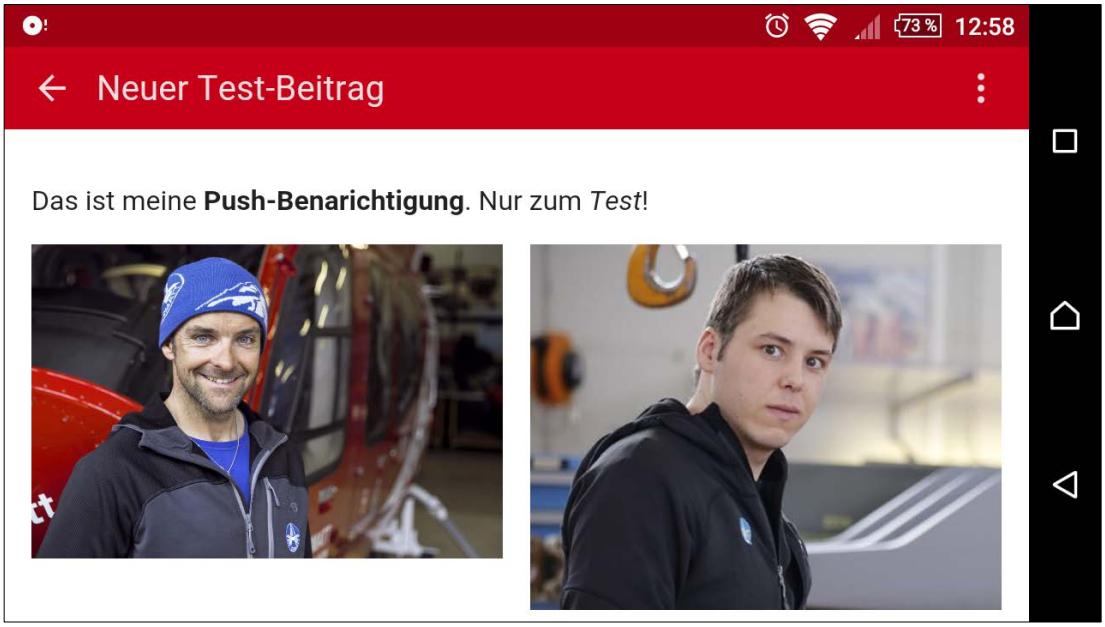


Abbildung 94 Newsticker-Beitrag im Landscape-Modus

18SEITE WEBCAM

Ein sehr wichtiges Element, was bis jetzt noch keine Aufmerksamkeit bekommen hat, sind die Webcam-Bilder. Bereits in der alten App waren diese Bilder abrufbar. Daher übernehmen wir dies auch auf die neue App.

18.1 Änderungen an der Datenbank

Für dieses Vorhaben brauchen wir in der Datenbank des Benutzers eine neue Tabelle, in der die benötigten Informationen gespeichert werden können. Dies ist leider mit etwas Aufwand verbunden. Ich musste diesen Vorgang lange testen, bis beim Update der App alle Anpassungen an der Datenbank erfolgreich ausgeführt wurden. Wichtige Informationen dazu findet man im Kapitel **Verhalten beim Update der App**.

In der Funktion *check_reg()* werden also die Änderungen für das Update auf die nächste Version vorbereitet:

```
1  // Änderungen bei Version 0.3.0
2  var db_version = $scope.compare_db_version(local_version,
   "0.3.0");
3  if(db_version < 0) {
4  }
```

Als erstes wird die neue Tabelle tbl_webcam erstellt:

```
1  db.transaction(function(tx) {
2  tx.executeSql("CREATE TABLE IF NOT EXISTS tbl_webcam
   (PK_webcam INTEGER NOT NULL UNIQUE, title TEXT NOT NULL,
   bild TEXT NOT NULL, interval INTEGER NOT NULL, FK_basis
   INTEGER NOT NULL, PRIMARY KEY(PK_webcam))", [], func-
   tion() {},
3  function(error) {
4  console.log("Fehler! " . error);
5  });
6  }, function(error) {
7  console.log("Fehler! " . error);
8  }, function() {
9  console.log("Tabelle tbl_webcam erstellt");
10 });
```

Anschliessend stelle ich eine Funktion bereit, die jeden Datensatz des folgenden Objekts in die erstellte Tabelle einfügt. Dies musste ich auf diese Weise realisieren, weil alle Befehle für Datenbankmodifikationen asynchron ausgeführt werden. Die Funktion sieht so aus:

```
1  $scope.saveWebcam = function(data) {
2
3  db.transaction(function(tx) {
4  tx.executeSql("INSERT INTO tbl_webcam(title, bild, inter-
   val, FK_basis) VALUES(?, ?, ?, ?)", [data.title,
   data.weblink, data.interval, data.FK_basis], function()
   {},
5  function(error) {
6  console.log("Eintrag nicht gespeichert: " +
   JSON.stringify(error));
7  });
8  }, function(error) {
9  console.log("Eintrag nicht gespeichert: " + error);
10 }, function() {
11 console.log("Eintrag " + data.title + " erfolgreich ge-
   speichert");
12 });
13 }
```

Diese Funktion nimmt einen Datensatz entgegen und speichert ihn in die Tabelle. Danach folgen die eigentlichen Daten. Diese habe ich in ein Object gespeichert:

```
1  var values = {
2  0: {PK_webcam: 1, title: 'Hangar', weblink:
   'http://www.air-zermatt.ch/webcam/Hangar.jpg', interval:
   300, FK_basis: 1},
3  1: {PK_webcam: 2, title: 'Heliport', weblink:
   'http://www.air-zermatt.ch/webcam/Heliport.jpg', inter-
   val: 10, FK_basis: 1},
4  2: {PK_webcam: 3, title: 'Platz 4', weblink:
   'http://www.air-zermatt.ch/webcam/Platz_4.jpg', interval:
   300, FK_basis: 1},
5  3: {PK_webcam: 4, title: 'Zermatt', weblink:
   'http://www.air-zermatt.ch/webcam/Zermatt.jpg', interval:
   300, FK_basis: 1},
6  4: {PK_webcam: 5, title: 'Richtung Goms', weblink:
   'http://www.air-zermatt.ch/webcam/Richtung_Goms.jpg', in-
   terval: 300, FK_basis: 2},
7  5: {PK_webcam: 6, title: 'Heliport 2', weblink:
   'http://www.air-zermatt.ch/webcam/Heliport2.jpg', inter-
   val: 300, FK_basis: 2},
8  6: {PK_webcam: 7, title: 'Heliport 1', weblink:
   'http://www.air-zermatt.ch/webcam/Heliport1.jpg', inter-
   val: 300, FK_basis: 2},
```

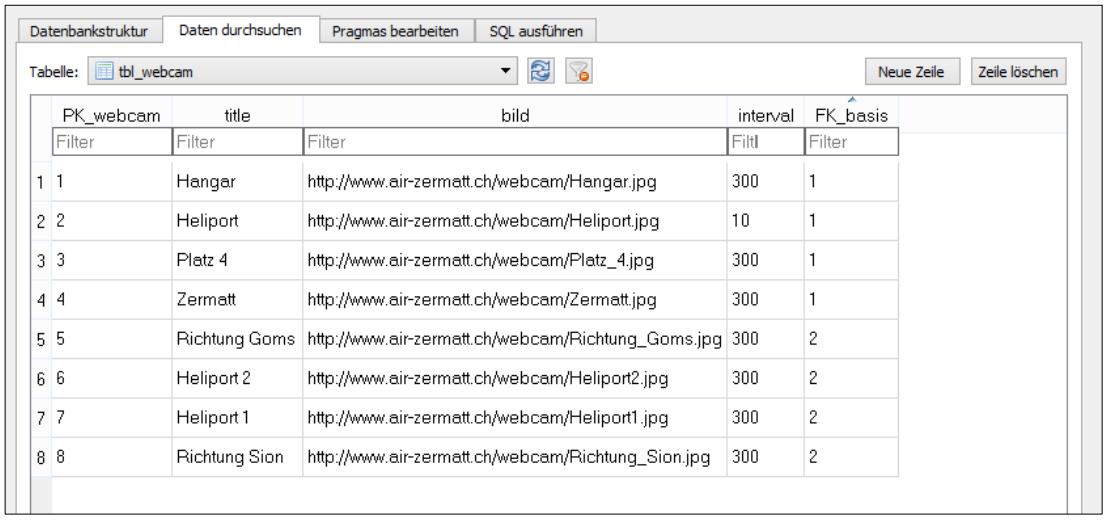
```
9  7: {PK_webcam: 8, title: 'Richtung Sion', weblink:
   'http://www.air-zermatt.ch/webcam/Richtung_Sion.jpg', in-
   terval: 300, FK_basis: 2}
10 }
```

Sind diese Werte angegeben, muss eine Schlaufe erstellt werden, die jedes Element des Objekts ausliest und der obigen Funktion übermittelt.

```
1  var length = Object.keys(values).length
2  console.log(length + " Einträge gefunden: " +
   JSON.stringify(values));
3
4  for(var i in values) {
5
6  if(values.hasOwnProperty(i)) {
7  $scope.saveWebcam(values[i]);
8  }
9  }
```

All diese Schritte sind notwendig, um eine einfache Tabelle einer bestehenden Datenbank hinzuzufügen. Wie im Kapitel **Verhalten beim Update der App** bereits erwähnt muss dieser Code Snippet im Controller bleiben, bis jedes Gerät, auf dem die App installiert ist, dieses Update durchgeführt hat!

Diese Änderungen werden selbstverständlich auch in das Datenbank-File *database.db* im *www*-Ordner eingefügt. Damit ist die Tabelle bei späteren Installationen bereits vorhanden.



	PK_webcam	title	bild	interval	FK_basis
	Filter	Filter	Filter	Filtl	Filter
1	1	Hangar	http://www.air-zermatt.ch/webcam/Hangar.jpg	300	1
2	2	Heliport	http://www.air-zermatt.ch/webcam/Heliport.jpg	10	1
3	3	Platz 4	http://www.air-zermatt.ch/webcam/Platz_4.jpg	300	1
4	4	Zermatt	http://www.air-zermatt.ch/webcam/Zermatt.jpg	300	1
5	5	Richtung Goms	http://www.air-zermatt.ch/webcam/Richtung_Goms.jpg	300	2
6	6	Heliport 2	http://www.air-zermatt.ch/webcam/Heliport2.jpg	300	2
7	7	Heliport 1	http://www.air-zermatt.ch/webcam/Heliport1.jpg	300	2
8	8	Richtung Sion	http://www.air-zermatt.ch/webcam/Richtung_Sion.jpg	300	2

Abbildung 95 Tabelle tbl_webcam im File database.db

18.2 Unterseite Webcam bei Android

Ich erstelle eine neue Unterseite im Template-Ordner. Zudem erstelle ich einen neuen Controller für die Unterseite. Im Script *app.js* füge ich diese Unterseite zum *$routeProvider* hinzu.

Die Bilder der beiden Basen unterteile ich in Tabs. Diese Bilder werden direkt vom Internet heruntergeladen und nicht zwischengespeichert. Daher füge ich eine Nachricht hinzu, wenn der Benutzer offline ist:

```
1  <div class="error_message" ng-hide="online">
2  Sie müssen mit dem Internet verbunden sein, um aktuelle
   Webcam-Bilder zu empfangen.
3  </div>
```

Danach folgen die beiden eigentlichen Tabs:

```
1  <div id="container_webcam_tabs" ng-show="online">
2  <md-tabs class="md-primary" md-stretch-tabs="auto" md-dy-
   namic-height md-swipe-content>
3  <md-tab id="zermatt" class="md-accent" label="Zermatt">
4  <md-content class="md-padding">
5
6  <div class="wrapper_masonry">
7  <md-card ng-repeat="picture in webcam | filter: {basis:
   'Zermatt'}" style="min-height: 100px">
8  <img ng-src="{{picture.bild}}" class="md-card-image"
   alt="Webcam" />
9  <md-card-title>
10 <md-card-title-text>
11 <span class="md-headline">Basis {{::picture.basis}}
   {{::picture.title}}</span>
12 </md-card-title-text>
13 </md-card-title>
14 </md-card>
15
16 </div>
17 </md-content>
18 </md-tab>
19 <md-tab id="raron" class="md-accent" label="Raron">
20 <md-content class="md-padding">
21
22 <div class="wrapper_masonry">
23 <md-card ng-repeat="picture in webcam | filter: {basis:
   'Raron'}" style="min-height: 100px">
```

```
24 <img ng-src="{{picture.bild}}" class="md-card-image"
   alt="Webcam" />
25 <md-card-title>
26 <md-card-title-text>
27 <span class="md-headline">Basis {{::picture.basis}}
   {{::picture.title}}</span>
28 </md-card-title-text>
29 </md-card-title>
30 </md-card>
31
32 </div>
33 </md-content>
34 </md-tab>
35 </md-tabs>
36
37 </div>
```

Die Bilder selbst werden durch ein *ng-repeat*-Befehl in die Cards geladen.

Im Controller WbcamCtrl werden als erstes zwei Arrays definiert:

```
1  var intervals = [];
2  $scope.webcam = [];
```

In die Variable *intervals* werden die einzelnen *$interval*-Befehle geladen. In den Scope webcam werden die relevanten Informationen gespeichert, die auf der Template-Seite benötigt werden.

```
1  // Laden der Informationen über Webcam
2  var query = "SELECT title, bild, interval, basis FROM
   tbl_webcam, tbl_basis WHERE FK_basis = PK_basis ORDER BY
   PK_webcam";
3  $cordovaSQLite.execute(db, query, []).then(function(res)
   {
4  count = res.rows.length;
5
6  if(count > 0) {
7
8  // Speichern der geladenen Daten im Scope
9  for(i=0;i < count;i++) {
10
11 $scope.webcam.push({
12 title: res.rows.item(i).title,
13 bild: res.rows.item(i).bild,
```

```
14 interval: res.rows.item(i).interval,
15 basis: res.rows.item(i).basis,
16 });
17 };
```

Nun werden die Daten aus der Datenbank gelesen und in einer for-Schlaufe in den Scope *webcam* geladen.

Zusätzlich wird in der gleichen Schlaufe eine Funktion erstellt, welche die Bilder in einem gewissen Abstand neu laden:

```
1  if($scope.online) {
2  (function(i) {
3
4  var delay = res.rows.item(i).interval * 1000;
5  var path = res.rows.item(i).bild;
6
7  intervals.push($interval(function() {
8  var random = Date.now() / 1000 | 0;
9  $scope.webcam[i].bild = path + "?v=" + random;
10 }, delay));
11
12 })(i);
13
14 }
```

Um die Performance nicht unnötig zu reduzieren, werden diese Intervalle nur erstellt, wenn das Gerät mit dem Internet verbunden ist. Die Intervalls-Dauer und der Pfad zum Bild werden ebenfalls aus der Tabelle gelesen. Dann wird zur Variable *intervals* jeweils die Funktion *$interval* hinzugefügt. In dieser Funktion kann eine zweite Funktion erstellt werden, welche dann in einem bestimmten Zyklus ausgeführt wird. Als zweiter Parameter übergibt man die Dauer, bis zum nächsten Ausführen der Funktion.

In der Funktion selbst wird ein neuer, aktueller Timestamp erstellt. Dieser wird dann an den Pfad des Bildes angehängt. Dies hat zur Folge, dass das Bild neu vom Server geladen wird.

18.3 Unterseite Webcam bei iOS

Bei iOS sieht das ganze ähnlich aus mit der Ausnahme, dass die Unterseite nicht in einer Template-Seite ist, sondern in der index.html.

19ÜBERNAHME DER BESTEHENDEN APP VON IPEAK

Bekanntlich wurde die vorherige iOS-App von iPeak vertrieben. Bei der neuen App wird der Vertrieb durch uns gemacht. Damit wir denselben App-Namen (Air Zermatt) verwenden konnten, haben wir die App vom Account von iPeak auf unseren Account transferiert. Damit mussten wir auch die Bundle-ID (ch.airzermatt.airzermatt) übernehmen. Dies bedeutete wiederum eine Änderungen an einigen Zertifikaten und Provisioning Profiles.

19.1 App zu iTunes Connect hochladen

Was bei Android die .apk-Datei ist, ist bei bei Apple die .ipa-Datei. Diese Datei kann via Xcode direkt zu iTunes Connect hochgeladen werden. Dazu muss mal mit dem Cordova-CLI eine neue Release-App gebuildet werden:

```
1  cordova build ios -release
```

Danach muss ein iPhone an den Mac angeschlossen werden. In Xcode wird das angeschlossene Gerät erkannt und wird als *Active Scheme* ausgewählt. Dann kann im Menü den Punkt Product >> Archive ausgewählt werden. Es wird ein Archive erstellt. Nach der Kompilierung erschein ein neues Fenster in dem die Archive ersichtlich sind. Von da aus kann die Datei validiert oder direkt zum App Store hochgeladen werden.

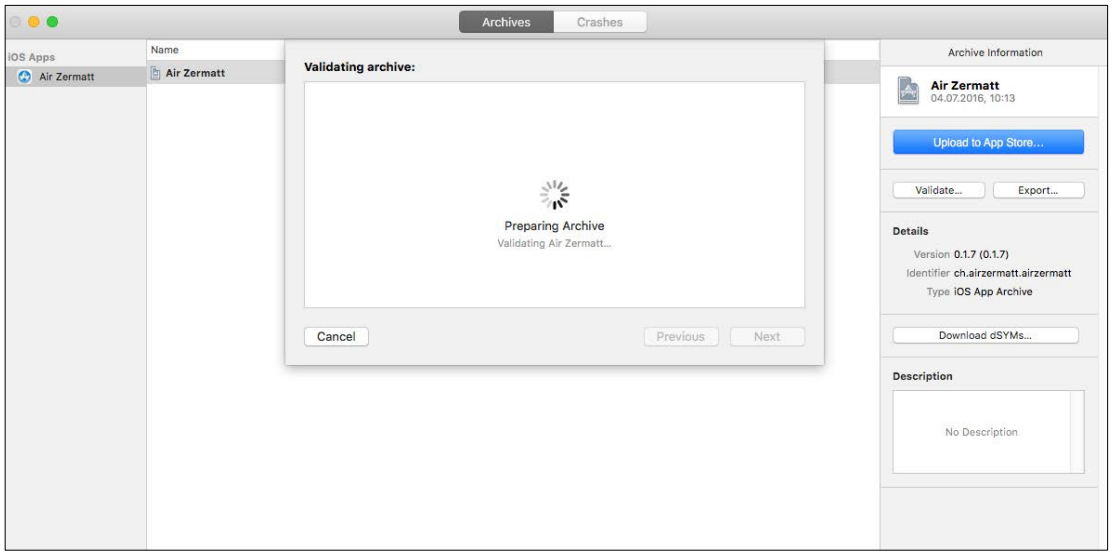


Abbildung 96 Xcode Fenster mit Übersicht der Archives

Während der Validierung muss das ganze Archiv signiert werden. Dies passiert mit einem Schlüssel, der in der Schlüsselbundverwaltung gespeichert ist. Man muss lediglich das Passwort des Benutzers eingeben, um die Signierung zu erlauben. Danach wird die Verbindung zu iTunes Connect erstellt und die Versionen- und Buildnummer werden verglichen. Allfällige

Fehler werden angezeigt. Danach ist das Archive hochgeladen und man erhält eine E-Mail als Bestätigung.

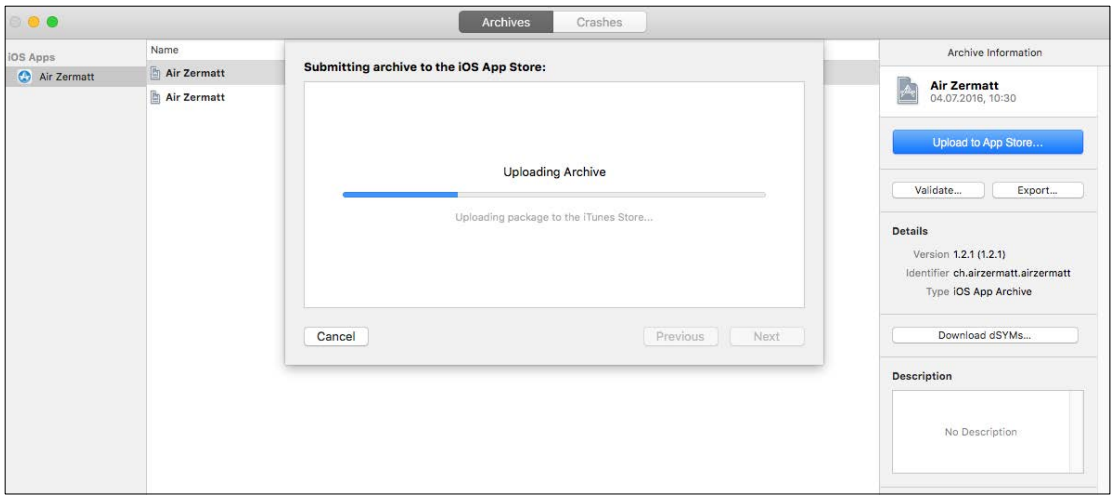


Abbildung 97 Upload der .ipa-Datei zum iTunes Store

Nach dem Upload ist die neue Version auch auf iTunes Connect ersichtlich.

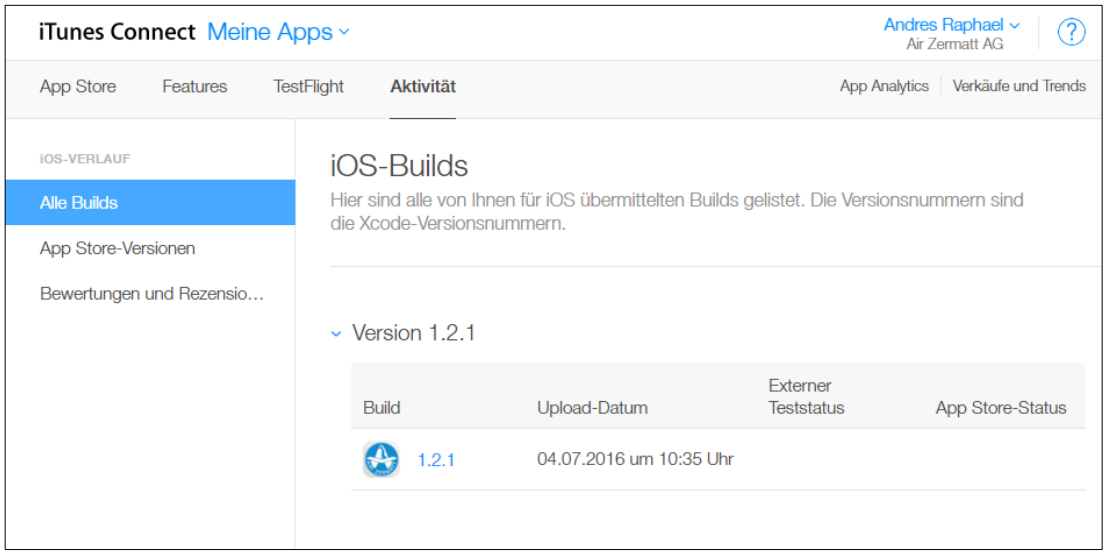


Abbildung 98 iTunes Connect Übersicht über alle iOS-Builds

19.2 App testen mit TestFlight

Ist ein neues Build vorhanden, so kann dieses zum TestFlight hinzugefügt werden. Hier müssen zuerst einige Testinformationen hinterlegt werden. Dann können, ähnlich wie bei Android Tester via E-Mailadresse hinzugefügt werden. Interne Tester sind Benutzer, welche auch auf iTunes Connect registriert sind. In unserem Fall ist sonst kein Benutzer registriert. Es können bis zu 2000 externe Tester angegeben werden. Sobald ein Build die Beta-App-Prüfung bestanden hat, ist sie zum Testen verfügbar.

Abbildung 99 Externe Tester für freigegebenes Build

Bei Externe Tester auf der Unterseite TestFlight erhält man die Übersicht über die zu testende Builds und die Tester. Fügt man ein Build zu Testen hinzu, werden einige Informationen angefordert. Unter anderem wird man gefragt, ob die App Kryptografie verwenden soll. Dies habe ich bisher immer mit *Nein* übersprungen.

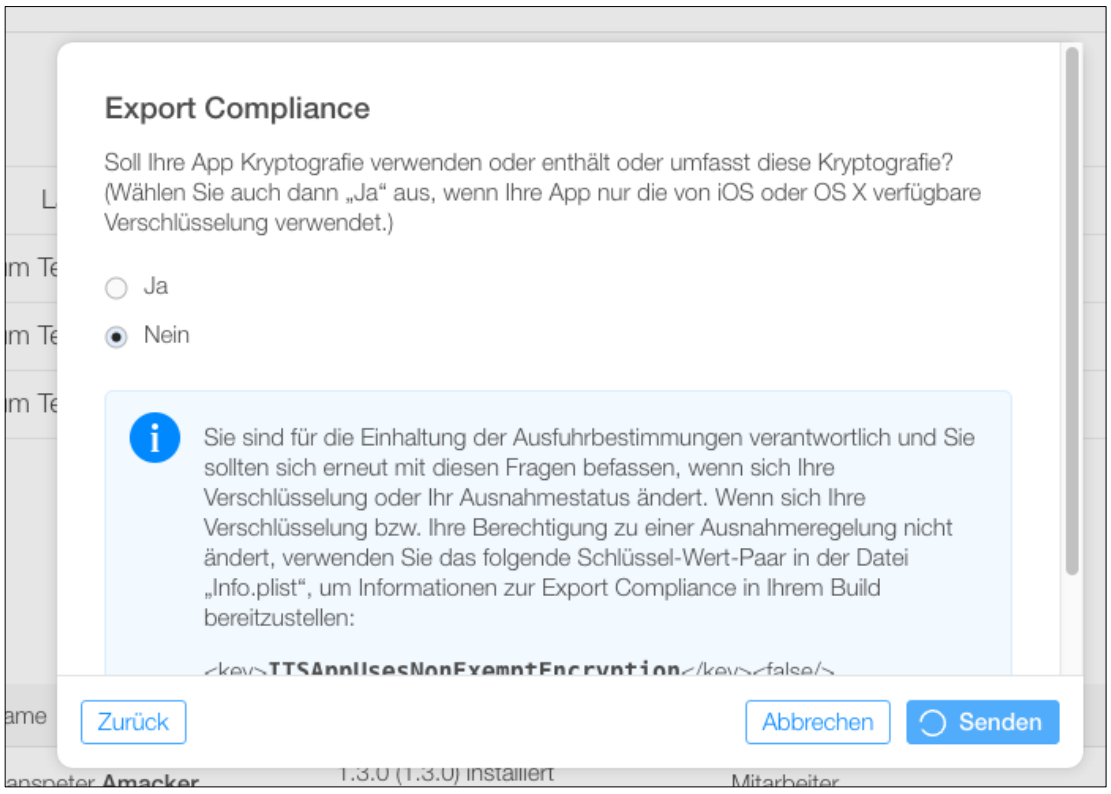


Abbildung 100 Frage über Kryptografie beim Hinzufügen eines Build zum Testen

Wird ein Build zum Testen freigegeben, erhalten alle eingetragenen Tester eine E-Mail mit einer ersten Beschreibung wie sie beim Installieren vorgehen sollen.

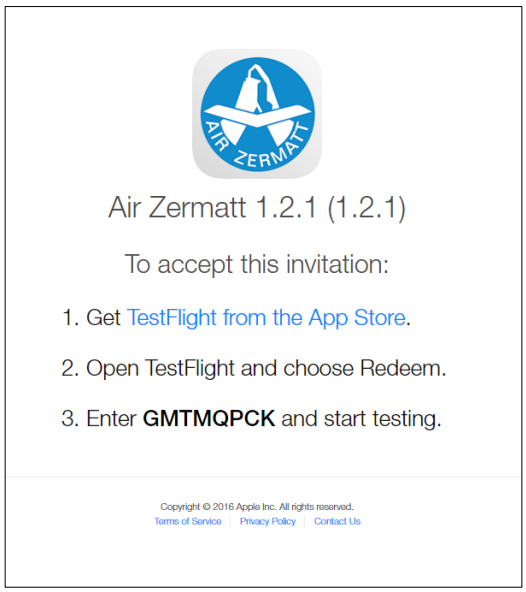


Abbildung 101 Instruktion für TestFlight

Über einen Link gelangen sie zu einer Seite, auf der ein Beschrieb ist, wie die Tester die App installieren können. Auf dem zu testenden Gerät muss zuerst die App **TestFlight** von Apple heruntergeladen werden. Darauf sind die App dann sichtbar, zu dem man eingeladen wurde. Weiter erhält man den Code, welcher man eingeben muss, damit die eigentliche App heruntergeladen werden kann.

Weiter ist auf dieser App für den Tester ersichtlich was er tun soll. Man kann Informationen und Anweisungen hinterlegen, welche der Test-Benutzer testen soll.

20 GÄNGIGE FEHLERQUELLEN

20.1 Fehler bei SQL-Abfrage: bind or column index out of range

Ein Fehler, der mir immer wieder unterlaufen ist, ist auf die unterschiedliche Syntax zwischen MySQL und SQLite zurückzuführen. Währendem bei MySQL die Werte in einer Abfrage in einfachen Anführungszeichen stehen müssen, muss bei SQLite der Platzhalter für den Wert alleine stehen. Beispiel einer MySQL-Abfrage:

```
1  // Abfrage MySQL:
2  UPDATE tbl_benutzer SET vorname = 'Raphael', nachname =
   'Andres';
```

Hingegen bei SQLite:

```
1  // Abfrage SQLite
2  UPDATE tbl_benutzer SET vorname = ?, nachname= ?;
```

IV. ZUSAMMENFASSUNG / FAZIT

Obwohl man bei Cordova mit den Webtechnologien arbeiten kann, sind es trotzdem zwei verschiedene Paar Schuhe. Ich habe vor diesem Projekt bereits zwei grössere Webapplikation realisiert und habe dabei sehr viel Erfahrungen sammeln können. Genau deshalb war ich auch einverstanden, dieses Projekt in Angriff zu nehmen. Während dem Projekt habe ich dann gemerkt, dass mir diese Erfahrungen sehr geholfen haben. Trotzdem gibt es gewisse Unterschiede zwischen dem reinen Realisieren eines Webauftritts auf einem Server, den Clients mithilfe eines Browsers erreichen und dem Realisieren einer mobilen App. Die Webapplikation in einer mobilen App existiert vielleicht hundert Mal, vielleicht tausend Mal, jeweils mit eigener Datenbank, mit anderen Ressourcen, sei es von der Performance, sei es von der Umgebung oder von der Grösse des Bildschirmes. Damit die eigenständige App nach aussen, vordergründig mit dem eigenen Webserver, kommunizieren kann, damit Daten bereitgestellt werden können, von Seiten Webserver oder von Seiten Applikation, oder damit die Applikation auf die Zielplattform massgeschneidert werden kann, braucht es andere Mittel, als dies bei “normalen“ Webapplikationen der Fall ist.

Ich habe gemerkt, dass die einzelnen Komponenten für sich simple und gut verständlich sind. In einem grösseren Kontext wird es aber schon schwieriger, da weitere Faktoren hinzukommen, die eine spezifische Funktion schon nicht mehr so klar ersichtlich macht.

Von den anfänglich gestellten Zielen habe ich nicht alle erreicht. So habe ich die Mobile App nur für die Plattform Android erstellt. Für die beiden anderen Plattformen muss nun aber nur noch kleine Änderungen vorgenommen werden. Die Kunden können sich nun auf dieser App anmelden und die Daten werden auf unserem Server gespeichert. Dieses erste Hauptziel wurde also erreicht.

Das zweite Ziel, das sich die App an Corporate Design der Firma halten soll unter der Berücksichtigung des Look and Feels der Plattform Android habe ich ebenfalls erreicht. Mithilfe von Angular Material konnte ich die App im typischen Material Design aufbauen.

Die Benachrichtigung via Push-Notification war ein weiteres Ziel des Projekts. Dieses habe ich ebenfalls erreicht. Die Mitarbeiter können nun in einem passwortgeschützten Back-End eine Mitteilung verfassen, die an registrierte Smartphones gesendet werden. Trotzdem lässt sich vor allem das Back-End noch etwas verfeinern und verbessern.

Von den vier gesteckten Meilensteinen habe ich immerhin drei erfolgreich erreicht.

V. QUELLENVERZEICHNIS

1 RECHERCHE BEI PROBLEMEN

1.1 Anzeigeproblem Schalter auf der Seite Einstellungen

siehe Arbeitsjournal Montag, 15.02.2016

Diese Seiten wurden aufgerufen:

http://stackoverflow.com/questions/24507539/add-sqlite-query-result-to-angularjs-scope

https://github.com/angular/material/issues/4036

http://stackoverflow.com/questions/27454484/angular-material-design-md-switch-doesnt-work-within-a-tab

1.2 Weitergabe eines Wertes zu einer asynchroner Funktion

siehe Arbeitsjournal Donnerstag, 18.02.2016

Diese Seiten wurden aufgerufen:

http://stackoverflow.com/questions/17244614/passing-variable-to-promise-in-a-loop

http://stackoverflow.com/questions/16305003/callback-function-access-to-closure-variables

http://www.webdeveasy.com/javascript-promises-and-angularjs-q-service/

http://stackoverflow.com/questions/20166033/how-to-access-result-from-the-previous-promise-in-angularjs-promise-chain

https://www.roelvanlisdonk.nl/?p=3952

http://stackoverflow.com/questions/27187495/why-i-cant-pass-parameter-into-then-function-angularjs

VI. ABBILDUNGSVERZEICHNIS

VII. GLOSSAR

AngularJS

ist ein JavaScript-Framework das vor allem in sogenannten SPA vorkommt. Es wurde von Google entwickelt und ist vergleichbar mit jQuery, aber mit dem grossen Unterschied, dass AngularJS eine bidirektionale Datenbindung besitzt.

App-ID

bezeichnet einen Namen der als indirekter Name für die App verwendet wird. Meist sie dieser Name aus, wie eine umgekehrte Domain-Adresse. Beginnend mit einer Top-Level-Domain, dann die eigentliche Domain und abschliessend ein eindeutiger Name der App.

builden

ein verdeutschtes Verb aus dem Englischen, das den Prozess vom Generieren der Applikation beschreibt.

Command Line Interface

kurz CLI, ist die Konsole bzw. das Terminal zur Steuerung einer Software.

Datei Manager

Ein Programm, mit dem man die Ordnerstruktur eines Gerätes ansehen und verwalten kann. Hat man Administratoren-Rechte, kann damit auch auf Verzeichnisse des Systems zugreifen.

Dependency Injection

Anders als **PHP Injection** ist Dependency Injection ein gewolltes und nützliches Tool zur Verarbeitung von Daten. Oft wird verkürzt von DI gesprochen. Dabei können Komponenten verwaltet werden, die für die Verarbeitung der Daten unabdingbar sind.

D-U-N-S Nummer

ist eine neunstellige Nummer, die jedes Unternehmen eines Landes eindeutig identifiziert. D-U-N-S steht für Data Universal Numbering System und wird von **Dun & Bradstreet** herausgegeben.

EventListener

Ein Eventlistener ist eine Funktion bei JavaScript, der auf gewisse HTML-Events "hört" und eine andere Funktion auslöst, wenn dieser gewisse Event zutrifft.

Framework

Ein Framework ist eine erweiterbare Hilfestellung zum Erstellen einer Applikation. Der Programmierer kann über Schnittstellen auf eine Reihe von Klassen zugreifen, mit denen er seine gewünschten Resultate erreichen kann.

git

git ist eine Software, die das Verteilen von Dateien vereinfacht. Mit einen git-client können über CMD oder Terminal Dateien von anderen Servern kopiert werden. Interessant ist auch die vorhandene Dateiversionierung. Über die Online-Plattform **GitHub** werden solche Dateisammlungen veröffentlicht.

Google Play Store

Die App-Plattform von Google, worauf Anwendungen und andere Medien für Android-Geräte angeboten werden.

ionic

Ein Gratis-**Framework** zum Erstellen hybriden Mobile Apps mit Web-Technologien.

Java Development Kit

siehe **Software Development Kit**

Look and Feel

bezeichnet das Aussehen oder Handhabung einer grafischen Benutzeroberfläche (GUI). Dies können gewisse Vorgaben von Farben, Layout und anderen gestalterischen Elemente sein.

Material Design

ist eine Designsprache, die von Google ins Leben gerufen wurde. Viele Apps für Android sind bereits nach diesem Design aufgebaut.

Model View ViewModel

Das Model View ViewModel-Muster ist eine Variante des Model-View-Controller-Musters, kurz MVC. Es soll die Darstellung von der Logik des User Interfaces trennen und damit die Testbarkeit verbessern, sowie das Entwickeln vereinfachen. Die Webdesigner können Ihren Fokus auf die User Experience legen und die Developer auf die Logik, die dahintersteht.

Payload

ist nicht zu verwechseln mit dem Schadteil eines Virus, der Daten löscht, manipuliert oder ausspäht. Via Payloads (auf Deutsch Nutzdaten) können Informationen und Datenpakete, die keine Protokollinformationen enthalten, zwischen zwei Parteien transportiert werden.

PHP Injection

beschreibt eine Sicherheitslücke in einem Web-Formular oder bei sichtbaren Get-Parameter. Diese Sicherheitslücke besteht, wenn Eingabefelder nur schlecht oder gar nicht validiert werden. So können durch ein Eingabefeld SQL-Befehle injizieren werden, die dann auch verarbeitet werden. Um eine solche Injektion zu verhindern muss unbedingt an jeder Stelle, an der Daten von einem Besucher entgegen genommen werden, einen Validierung der Daten durchgeführt werden, bevor die Daten weiterverarbeitet werden.

Programmierschnittstelle

englisch Application Programming Interface (API) genannt. Diese Schnittstelle erleichtert das Entwickeln einer Applikation. Gewisse wiederkehrende Code-Blöcke können aufgerufen werden.

Push-Notification

Der Besitzer eines Smartphone erhält eine Nachricht. Der Informationsfluss wird also vom Sender gesteuert.

Registration-ID

Die Registration-ID ist eine ID, die ein Smartphone erhält, wenn es sich an einem Push-Notification-Provider wie z.B. GCM anmeldet. Diese ID muss beim serverseitigen Versenden eines Payloads mit gesendet werden, damit der Provider weiss, an welches Gerät die Nachricht gesendet werden soll.

root

bezeichnet im Zusammenhang mit mobilen Betriebssystemen die Möglichkeit, das

Gerät als Administrator zu verwalten. Man hat also als root-Benutzer Zugriff auf alle Verzeichnisse und hat höhere Berechtigungen.

Software Development Kit

ist eine Sammlung von fertigen Werkzeugen und Anwendungen. Z.B. bietet das Android SDK eine Vielzahl von Emulatoren zum Testen von Mobilen Applikation.

Single Page Application

kurz SPA, bezeichnet eine einseitige Webanwendung, deren Inhalt dynamisch meist via AJAX nachgeladen wird.

SQLite

ist eine vereinfachte Programmbibliothek, die die meisten SQL-Befehle unterstützt und weitere einzigartige Befehle mitbringt. Es wurde entwickelt für eingebettete Datenbanksystemen, weshalb es nur eine einzigen Benutzter gibt, der alle Abfragen durchführen kann. SQLite kann ohne weiter Server-Software verwendet werden.

XHR

ist die Abkürzung von xmlhttp-Request und bezeichnet die Möglichkeit einen Inhalt einer Seite asynchron zu laden. Das hat den Vorteil, dass beim Laden von neuen Inhalten, nicht die gesamte Seite, mit sämtlichen Ressourcen erneut heruntergeladen und gerendert werden muss, sondern nur der sich ändernde Inhalt.