HOCHSCHULE
MITTWEIDA
UNIVERSITY OF
APPLIED SCIENCES

BACHELORARBEIT

Herr
Christoph Singer

Implementierung von Push-Benachrichtigungen für die Android-Applikation HSMWmobil

Mittweida, 2013

Fakultät Elektro- und Informationstechnik

BACHELORARBEIT

Implementierung von Push-Benachrichtigungen für die Android-Applikation HSMWmobil

Autor:
Herr

Christoph Singer

Studiengang:
Multimediatechnik

Seminargruppe:
MK10w1-B

Erstprüfer:
Prof. Dr.-Ing. Frank Zimmer

Zweitprüfer:
M.Sc. Rico Thomanek

Einreichung:
Mittweida, 23.09.2013

Verteidigung/Bewertung:
Mittweida, 2013

Faculty Elektro- und Informationstechnik

BACHELOR THESIS

Implementation of Push-Notifications for the Android-Application HSMWmobil

author:
Mr.

Christoph Singer

course of studies:
Multimediatechnik

seminar group:
MK10w1-B

first examiner:
Prof. Dr.-Ing. Frank Zimmer

second examiner:
M.Sc. Rico Thomanek

submission:
Mittweida, 23.09.2013

defence/ evaluation:
Mittweida, 2013

Bibliografische Beschreibung:

Singer, Christoph:
Implementierung von Push-Benachrichtigungen für die Android-Applikation HSMWmobil. – 2013. – 10, 46, 2 S.
Mittweida, Hochschule Mittweida, Fakultät Elektro- und Informationstechnik, Bachelorarbeit, 2013

Referat:

Die vorliegende Arbeit befasst sich mit der Implementierung von Push-Benachrichtigungen für die Android-Hochschul-App HSMWmobil. Als Push-Benachrichtigungssystem wird der Dienst „Google Cloud Messaging“ genutzt, welcher von der Google Inc. entwickelt wurde. Der Dienst wird in die vorhandene Hochschul-App HSMWmobil für das Betriebssystem Android integriert, damit diese unterschiedliche Push-Benachrichtigungen anzeigen kann. Daraufhin wird ein Webserver eingerichtet, welcher benötigt wird, um Push-Benachrichtigungen an mobile Endgeräte über den „Google Cloud Messaging“-Server zu versenden. Auf dem Webserver befindet sich eine Benutzeroberfläche, von der aus verschiedene Push-Benachrichtigungen an alle registrierten mobilen Endgeräte, welche in einer XML-Datei auf dem Webserver abgespeichert sind, versendet werden können. Durch die Push-Benachrichtigungen sollen die Benutzer der Hochschul-App HSMWmobil schnellstmöglich über aktuelle Ereignisse der Hochschule Mittweida informiert werden.

Inhalt

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungsverzeichnis

API	Application Programming Interface
AVD	Android Virtual Device
cURL	Client for URLs
DOM	Document Object Model
HSDPA	High Speed Downlink Packet Access
HTML	HyperText Markup Language
ID	Identifikator
IP	Internetprotokoll
JSON	JavaScript Object Notation
SDK	Software Development Kit
SSL	Secure Sockets Layer
UI	User Interface (Benutzeroberfläche)
URL	Uniform Resource Locator
UTF	Unicode Transformation Format
W3C	World Wide Web Consortium
WLAN	Wireless Local Area Network
XML	Extensible Markup Language

1 Einleitung

Viele Applikationen für das mobile Betriebssystem Android bieten die Möglichkeit aktuelle Informationen anzeigen zu lassen. Dabei kann es sich beispielsweise um Sportergebnisse, Wetterinformationen oder andere relevante Nachrichten handeln. Viele Applikationen aktualisieren diese Informationen erst beim Aufruf der Applikation durch den Benutzer. Einige Applikationen bieten jedoch auch die Möglichkeit, Informationen im Hintergrund zu synchronisieren, ohne das der Benutzer die Applikation starten muss. So können dem Benutzer immer die aktuellsten Informationen auf seinem mobilen Endgerät zur Verfügung gestellt werden. Das Problem dabei besteht jedoch darin, dass der Benutzer die neuen Informationen erst bemerkt, sobald dieser die Applikation startet. Dies ist für einige Benutzer ausreichend, da sie nicht sofort über die neusten Ereignisse informiert werden wollen, sondern erst dann, wenn sie die Applikation starten. Jedoch gibt es auch Benutzer, welche sofort über die aktuellsten Informationen benachrichtigt werden wollen, sobald diese zur Verfügung stehen. Um dieses Problem zu lösen, wird der Dienst „Google Cloud Messaging“ genutzt. Dieser Dienst teilt dem Benutzer, über eine Push-Benachrichtigung mit, dass neue Informationen zur Verfügung stehen.

Die Hochschule Mittweida besitzt seit dem 17. Dezember 2012 eine eigene Hochschul-App, mit dem Namen HSMWmobil. Diese ist für das mobile Betriebssystem Android und iOS in den jeweiligen App Stores verfügbar. Die Hochschul-App soll dem Benutzer wichtige Informationen über das Studium und die Hochschule bieten. Sie wurde dem Inhalt nach den Bedürfnissen von Studenten und Mitarbeitern angepasst und soll eine Hilfe im Hochschulalltag sein. Somit ist es dem Benutzer möglich, beispielsweise den Speiseplan der Mensa für die aktuelle und die darauffolgende Woche oder den eigenen, aktuellen Stundenplan anzeigen zu lassen. Über den Campusplan besteht die Möglichkeit, sich zu Hochschulgebäuden navigieren zu lassen und über ein Kontaktverzeichnis Informationen wie Sprechzeiten, Telefonnummern und Emailadressen von Hochschulmitarbeitern einzusehen. Neuigkeiten über die Hochschule werden dem Benutzer über HSMW News und HSMW Blog berichtet. Des Weiteren kann die aktuelle Ausgabe der Hochschulzeitung „Die Novum“ gelesen und heruntergeladen oder der Internet-Livestream des Hochschulradios „99drei Radio Mittweida“ angehört werden.

Die Hochschul-App verfügt über viele Informationskanäle, wie beispielsweise den Bereich der Hochschulzeitung „Die Novum“ oder den Bereich HSMW Blog und HSMW News, zu denen häufig neue Informationen hinzugefügt werden. Damit der Benutzer schnellstmöglich über aktuelle Informationen rund um die Hochschule versorgt wird, bietet sich die Integration von Push-Benachrichtigungen über den Dienst „Google Cloud Messaging“ in der Hochschul-App an. Die Push-Benachrichtigung stellt eine sehr nützliche neue Funktion für die Hochschul-App dar, und bietet dem Benutzer einen vorher nicht dagewesenen Mehrwert an. Durch die Bachelorarbeit kann der Autor seine vorhandenen Kenntnisse in der Android- und PHP-Programmierung sowohl verbessern, als auch erweitern.

2 Zielsetzung

Das Ziel der Bachelorarbeit ist die Erweiterung der schon vorhandenen Hochschul-App HSMWmobil für Android um die Möglichkeit der Push-Benachrichtigung über den Dienst „Google Cloud Messaging“.

Durch Push-Benachrichtigungen soll der Benutzer der Hochschul-App darauf aufmerksam gemacht werden, dass neue Informationen aus den Informationskanälen der Hochschul-App zur Verfügung stehen, ohne die eigentliche Information des Informationskanales vorweg zu benennen.

Die Bachelorarbeit soll die Funktionsweise des Dienstes „Google Cloud Messaging“ im Allgemeinen erläutern, sowie die konkrete Anwendung durch die Implementierung innerhalb der Hochschul-App. Dabei wird konkret auf die Registrierung für den Dienst „Google Cloud Messaging“ innerhalb der Hochschul-App sowie den Broadcast Receiver eingegangen, welcher für den Erhalt von „Google Cloud Messaging“-Benachrichtigungen über einen Broadcast Intent verantwortlich ist.

Es soll auch der Server anhand der darauf befindlichen PHP-Skripte sowie dessen Anbindung an die Hochschul-App erörtert werden. Die PHP-Skripte sind für den Erhalt und die Weitergabe der Registration ID vom mobilen Endgerät (Client) zum „Google Cloud Messaging“-Server, sowie der Erstellung und Weitergabe von Benachrichtigungen an den „Google Cloud Messaging“-Server verantwortlich.

Die Möglichkeit Push-Benachrichtigungen innerhalb der Hochschul-App zu empfangen und die Erstellung von Push-Benachrichtigungen über eine Benutzeroberfläche auf dem Server werden für die Informationskanäle medien-mittweida.de, HSMW Blog und HSMW News sowie die Hochschulzeitung „Die Novum“ erstellt.

3 Randbedingungen

Um Push-Benachrichtigungen über den Dienst “Google Cloud Messaging” in einer Applikation nutzen zu können, muss mindestens die Android Version Froyo (2.2) bzw. der API-Level 8 auf dem mobilen Endgerät vorhanden sein. Zusätzlich muss auf dem mobilen Endgerät die Google Play Store Applikation installiert und ein Google Account eingerichtet sein. Ab der Android Version Ice Cream Sandwich (4.04) bzw. dem API-Level 15 ist kein Google Account mehr notwendig, um den Dienst “Google Cloud Messaging” nutzen zu können. (Google Inc., 2013 a)

Die Hochschul-App HSMWmobil kann erst ab Android Version Gingerbread (2.3.3) bzw. dem API-Level 10 im Google Play Store angeboten und auf mobilen Endgeräten installiert werden. Deswegen muss bereits die Google Play Store Applikation installiert und ein Google Account eingerichtet sein, damit das mobile Endgerät auf den Google Play Store zugreifen kann. Damit sind alle Voraussetzungen für den Betrieb des Dienstes „Google Cloud Messaging“ auf den mobilen Endgeräten geschaffen.

4 Theoretische Grundlagen

In den beiden Informationsaustauschverfahren Polling und Push werden Daten zwischen dem Server und dem Client ausgetauscht. Beim Polling-Verfahren muss der Empfänger immer aktiv nach einer neuen Information beim Absender fragen, die womöglich für ihn zur Verfügung stehen könnte. Damit liegt die Verantwortung über den Austausch von Informationen beim Empfänger. Der Absender legt die Informationen an einer geeigneten und auffindbaren Stelle ab, ohne einen konkreten Empfänger anzugeben. Der Empfänger kann die Informationen daraufhin nach individuellen Kriterien anfordern.

Beim Push-Verfahren dagegen, wird der Empfänger immer automatisch über neue Informationen vom Absender informiert. Damit liegt hier die Verantwortung über den Austausch von Informationen beim Absender, welcher bewusst konkrete Empfänger auswählt. (Schäfer, 2013)

Der Dienst „Google Cloud Messaging“ ist ein von Google Inc. entwickelter Dienst, welcher am 27. Juni 2012 auf der alljährlichen Entwickler-Konferenz Google I/O, vorgestellt wurde und den vorhergehenden Dienst „Google Cloud to Device Messaging“ ablöst. Der Dienst sendet sogenannte Push-Benachrichtigungen direkt vom „Google Cloud Messaging“-Server zu einem mobilen Endgerät (Client). Push-Benachrichtigungen werden nahezu in Echtzeit übertragen und bieten gegenüber dem Polling-Verfahren, bei dem das mobile Endgerät (Client) einen Server in periodischen Zeitabständen nach neuen Benachrichtigungen abfragt, mehrere Vorteile. Der Energieverbrauch und die serverseitige Ressourcenauslastung sind geringer, da das mobile Endgerät (Client) keine periodischen Abfragen des Servers auf Updates durchführen muss. Es besteht eine permanente passive Internetverbindung, diese wird jedoch erst aktiv genutzt, sobald der „Google Cloud Messaging“-Server eine Benachrichtigung an das mobile Endgerät (Client) gesendet hat. Bei der passiven Internetverbindung wird nur sehr wenig Energie verbraucht. Im Gegensatz dazu, benötigt das Polling-Verfahren, durch die periodischen Abfragen des Servers, im Allgemeinen, einen höheren Energieverbrauch. Die Internetverbindung wird durch die periodische Abfrage häufiger aktiv genutzt, selbst wenn keine neuen Benachrichtigungen für das mobile Endgerät (Client) zur Verfügung stehen.

Durch das Push-Verfahren besteht nicht die Gefahr, eine Benachrichtigung zu spät zu erhalten, welche durch ein zu lange eingestelltes periodisches, zeitliches Intervall der Serverabfrage beim Polling-Verfahren auftreten kann. (Rémond, 2013)

5 Entwurf

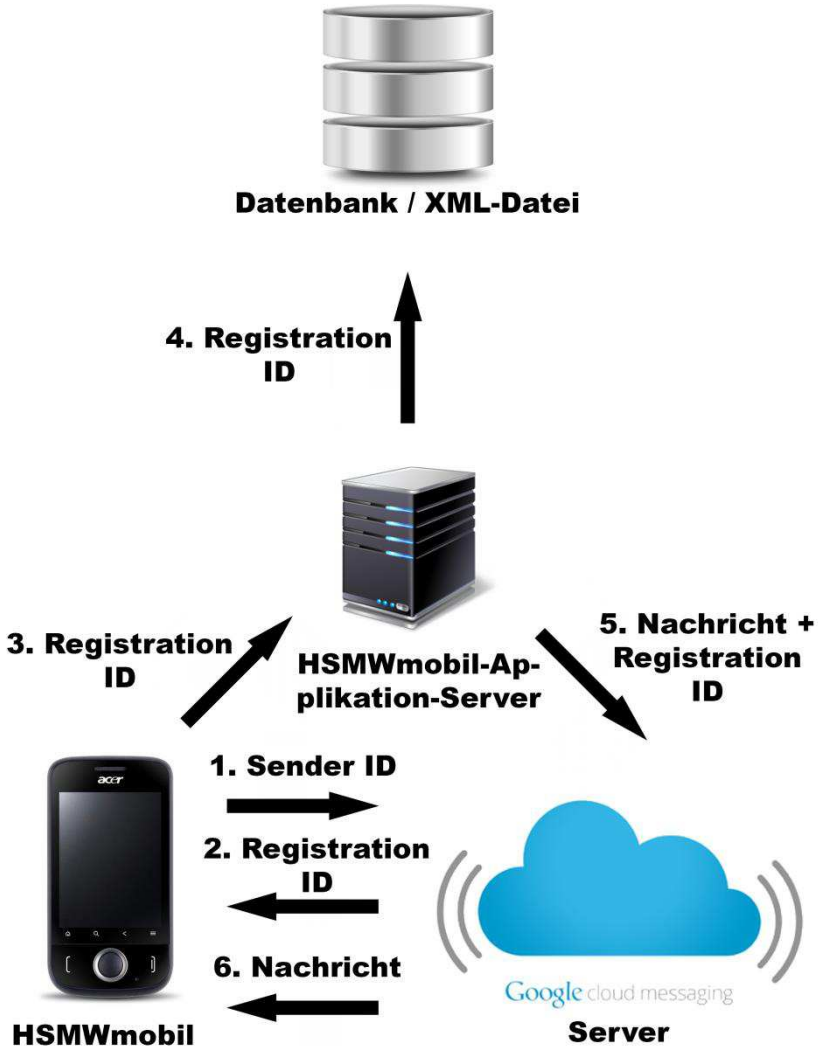

Abbildung 1: Darstellung der Funktionsweise des Dienstes „Google Cloud Messaging“.

Um den Dienst „Google Cloud Messaging“ in der Hochschul-App HSMWmobil nutzen zu können, muss sich jedes mobile Endgerät (Client) mit dem Betriebssystem Android am „Google Cloud Messaging“-Server mit seiner Sender ID registrieren. (1.) Der „Google Cloud Messaging“-Server bestätigt die erfolgreiche Registrierung des mobilen Endgerätes (Client) mit der Registration ID, welche an das mobile Endgerät (Client) zurückgeschickt wird. (2.) Die Registration ID wird über das mobile Endgerät (Client), mit Hilfe eines HTTP-Posts, an den HSMWmobil-Applikation-Server geschickt. (3.) Dort wird die Registration ID in einer Datenbank oder XML-Datei für die spätere Nutzung gespeichert. (4.) Sollte eine Push-Benachrichtigung vom HSMWmobil-Applikation-Server aus an das mobile Endgerät (Client) abgeschickt werden, so wird diese, zusammen mit der Registration ID, welche in der Datenbank oder XML-Datei gespeichert ist, mit Hilfe eines HTTP-Posts, an den „Google Cloud Messaging“-Server gesendet. (5.) Der „Google Cloud

Messaging“-Server sendet die Benachrichtigung an die entsprechenden mobilen Endgeräte (Client), mit der vorher definierten Registration ID. (6.) (Tamada, 2012)

Das in Abbildung 2 dargestellte Use-Case-Diagramm zeigt die Benutzung der Push-Benachrichtigungen innerhalb der Hochschul-App HSMWmobil. Der Push-Versender wählt eine der Informationskanäle aus, für die er eine Push-Benachrichtigung an die Benutzer der Hochschul-App versenden möchte. Daraufhin betätigt der Versender den, auf der Benutzeroberfläche des HSMWmobil-Applikation-Servers befindlichen Button. Unmittelbar danach wird die Push-Benachrichtigung auf allen mobilen Endgeräten, auf der die Hochschul-App installiert ist, dem Benutzer angezeigt.

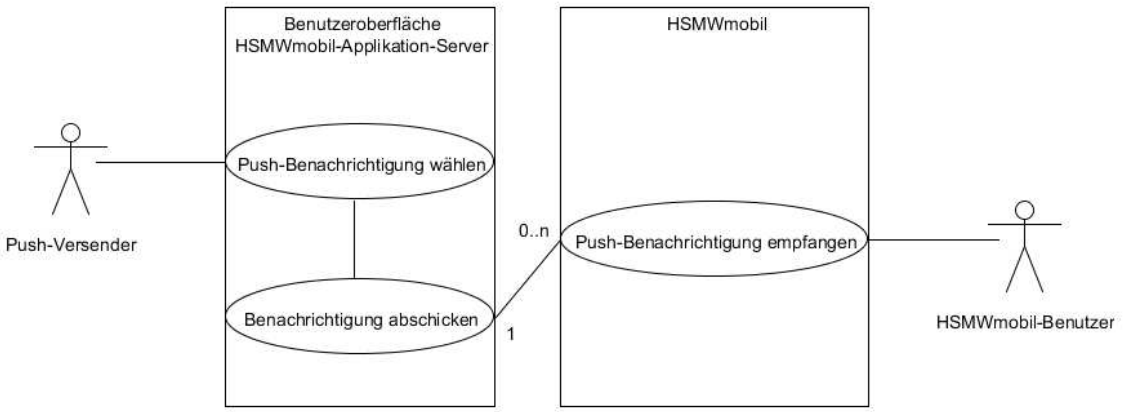

Abbildung 2: Use-Case-Diagramm zur Benutzung der Push-Benachrichtigungen innerhalb der Hochschul-App HSMWmobil.

6 Implementierung

6.1 „Google Cloud Messaging" Integration in Applikation

6.1.1 Registrierung der Applikation bei „Google Cloud Messaging"

Damit eine Registrierung der Hochschul-App HSMWmobil bei „Google Cloud Messaging" gelingt, muss die Funktion, wie in Abbildung 3 zu sehen, über die Internetseite Google API Code erst für das Projekt HSMWmobil aktiviert werden. Auf der Internetseite Google API Code ist es möglich Projekte zu erstellen, mit denen man Zugriff auf verschiedene APIs für Services bekommt, welche die Firma Google Inc. zur Verfügung stellt.

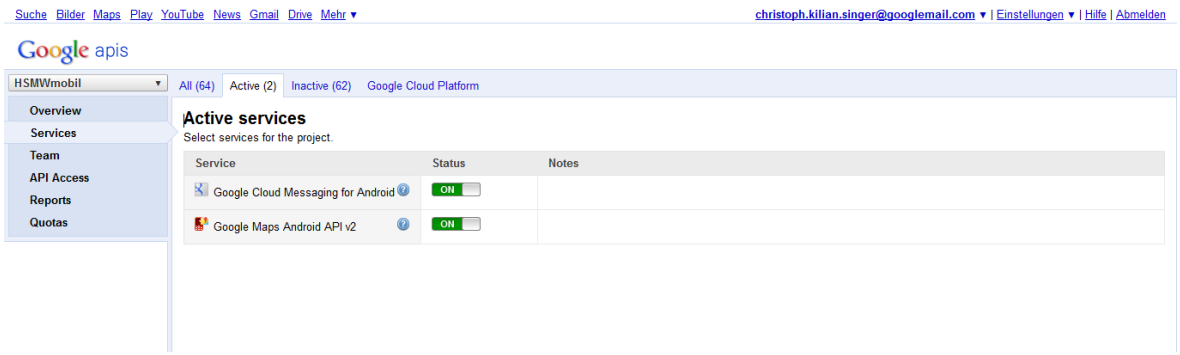

Abbildung 3: Google API Code mit aktivierter „Google Cloud Messaging"-API für die Hochschul-App HSMWmobil.

Das Projekt HSMWmobil verfügt schon über eine vorhandene Project Number. Daher muss diese nicht neu erstellt werden, sondern kann aus dem bereits vorhandenen Projekt übernommen werden. Sie wird als Sender ID für die Anmeldung des Dienstes „Google Cloud Messaging" in der Hochschul-App genutzt. Da das Projekt jedoch nicht über einen Server-API-Key verfügt, muss dieser neu erstellt werden. Er dient im PHP-Skript des Servers zur Authentisierung am „Google Cloud Messaging"-Server, damit Benachrichtigungen an den „Google Cloud Messaging"-Server versendet werden können.

Der Dienst „Google Cloud Messaging" benutzt eine API des Google Play Service SDK. Um diese nutzen zu können, muss die Google Play Service Client Library in die Applikation integriert werden. Da die Hochschul-App schon den Google Play Service Google Maps Android API v2 benutzt, um den Campusplan anzeigen zu lassen, ist die Google Play Service Client Library schon in die Hochschul-App integriert. (Google Inc., 2013 b)

6.1.1.1 Permissions

Die *Permission*-Erweiterung im Manifest zur Registrierung der Applikation am Dienst „Google Cloud Messaging“ sieht wie in Abbildung 4 dargestellt aus.

```
<uses-permission android:name="com.google.android.c2dm.permission.RECEIVE" />
```

Abbildung 4: Permission zur Registrierung des Dienstes „Google Cloud Messaging“ im Manifest.

Des Weiteren werden die in Abbildung 5 dargestellten *Permissions* für die Registrierung der Applikation am Dienst „Google Cloud Messaging“ benötigt. Diese sind jedoch schon in der Manifest-Datei der Hochschul-App vorhanden und müssen nicht noch einmal hinzugefügt werden.

```
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.GET_ACCOUNTS" />
```

Abbildung 5: Permissions im Manifest, um die Internetverbindung sowie den Google Account nutzen zu können.

6.1.1.2 Internetverbindung

Als erstes sollte überprüft werden, ob eine aktive Internetverbindung für das mobile Endgerät besteht, damit eine Kommunikation zwischen den Servern stattfinden kann.

Die Hochschul-App HSMWmobil benötigt jedoch zwingend eine aktive Internetverbindung um vom Benutzer gestartet zu werden. Daher ist eine solche Überprüfung schon in der Hochschul-App vorhanden und muss nicht noch einmal integriert werden.

6.1.1.3 Registrierung

Die Datei HsmwApplication.java befindet sich innerhalb des Paketes de.hsmw.app. Die schon vorhandene Klasse *HsmwApplication*, welche sich in der Datei HsmwApplication.java befindet, wird erweitert, um den Dienst „Google Cloud Messaging“ nutzen zu können. Damit ein mobiles Endgerät, auf dem die Hochschul-App HSMWmobil installiert ist, eine eindeutige Registration ID erhält, muss die Methode *register()* in der Klasse *HsmwApplication* aufgerufen werden. So können Benachrichtigungen des

„Google Cloud Messaging“-Servers empfangen und somit die Applikation für den Dienst registriert werden. Die Methode *register()* sendet - wie in Abbildung 6 dargestellt - die Sender ID mit. Sie ist die Project Number des Projektes HSMWmobil aus der Internetseite Google API Code. Die Methode *register()* gibt daraufhin die Registration ID zurück, welche als Antwort vom „Google Cloud Messaging“-Server kommt. Diese wird der Methode *setRegistrationId()* übergeben.

```
regid = gcm.register(SENDER_ID);
setRegistrationId(context, regid);
```

Abbildung 6: Aufruf der Methode register() und der Methode setRegistrationId().

Innerhalb der Methode *setRegistrationId()* wird daraufhin die Registration ID, wie in Abbildung 7 zu sehen, über die Methode *putString()* in den *SharedPreferences* gespeichert. Zusätzlich zur Registration ID wird noch der Version Code der Applikation über die Methode *putInt()* sowie die Expiration Time über die Methode *putLong()* in den *SharedPreferences* abgespeichert.

```
final SharedPreferences prefs = getGCMPreferences(context);
int appVersion = getAppVersion(context);
SharedPreferences.Editor editor = prefs.edit();
editor.putString(PROPERTY_REG_ID, regId);
editor.putInt(PROPERTY_APP_VERSION, appVersion);
long expirationTime = System.currentTimeMillis()+REGISTRATION_EXPIRY_TIME_MS;
editor.putLong(PROPERTY_ON_SERVER_EXPIRATION_TIME, expirationTime);
editor.commit();
```

Abbildung 7: Speicherung der Registration ID, des Version Codes und der Expiration Time in die SharedPreferences.

Innerhalb der Methode *getAppVersion()* wird der Version Code, wie in Abbildung 8 dargestellt, über die Methode *getPackageInfo()* aus der *PackageInfo* herausgesucht.

```
PackageInfo packageInfo = context.getPackageManager()
.getPackageInfo(context.getPackageName(), 0);
return packageInfo.versionCode;
```

Abbildung 8: Heraussuchen des Version Codes aus der PackageInfo.

Die Expiration Time setzt sich, wie in Abbildung 7 zu sehen, aus der aktuellen Zeit und einer vorgegebenen Zeitspanne zusammen. Über die *SharedPreferences* können Variablen und deren Inhalte gespeichert werden, damit diese in mehreren Activitys aufgerufen und genutzt werden können.

In der Methode *getRegistrationId()* soll die Registration ID zurückgegeben werden. Dabei wird, wie in Abbildung 9 dargestellt, mit der Methode *Length()* überprüft, ob die Registration ID die Länge 0 besitzt und somit nicht in den *SharedPreferences* vorhanden ist. Ist dies der Fall, so wird eine leere Menge zurückgegeben.

```
final SharedPreferences prefs = getGCMPreferences(context);
String registrationId = prefs.getString(PROPERTY_REG_ID, "");
if (registrationId.length() == 0) {
	return "";
}
```

Abbildung 9: Überprüfung, ob Registration ID schon in den SharedPreferences vorhanden ist.

Um ein Fehlverhalten der Hochschul-App zu vermeiden, ist eine Überprüfung notwendig, ob die vorhandene Hochschul-App aktualisiert wurde. Dabei wird, wie in Abbildung 10 zu sehen, der Version Code einer schon vorher installierten Version der Hochschul-App, welcher in den *SharedPreferences* abgelegt worden sein könnte, mit dem aktuellen Version Code verglichen. Sollten beide Version Codes unterschiedlich sein, so wird auch hier eine leere Menge zurückgegeben.

```
int registeredVersion = prefs.getInt(PROPERTY_APP_VERSION,
Integer.MIN_VALUE);
int currentVersion = getAppVersion(context);
if (registeredVersion != currentVersion || isRegistrationExpired()) {
	return "";
}
```

Abbildung 10: Überprüfung, ob die Hochschul-App aktualisiert wurde.

Es wird außerdem über die Methode *isRegistrationExpired()* überprüft, wie in Abbildung 11 zu sehen, ob die vorher in den *SharedPreferences* festgelegte Expiration Time, welche den Zeitpunkt festlegt, in dem die Registrierung der Applikation beim Dienst „Google Cloud Messaging“ ablaufen soll, überschritten wurde. Dabei wird die aktuelle Systemzeit mit der Expiration Time verglichen und ein boole´scher Wert zurückgegeben. Ist die Systemzeit größer als die Expiration Time, so gibt die Methode *isRegistrationExpired()* ein *true* zurück, welches wiederum in der Methode *getRegistrationId()* eine leere Menge zurückgibt. Sollte die Registration ID weder leer, noch abgelaufen oder zu einer veralteten Applikation gehören, dann gibt die Methode *getRegistrationId()* die Registration ID aus den *SharedPreferences* zurück.

```
final SharedPreferences prefs = getGCMPreferences(context);
long expirationTime = prefs.getLong(PROPERTY_ON_SERVER_EXPIRATION_TIME, -1);
return System.currentTimeMillis() > expirationTime;
```

Abbildung 11: Überprüft, ob die Expiration Time überschritten wurde.

In der *onCreate()*-Methode wird, wie in Abbildung 12 dargestellt, der Rückgabewert der Methode *getRegistrationId()* auf Inhalt überprüft. Sollte dieser Inhalt leer bzw. den Zahlenwert 0 zurückliefern, so wird die Methode *registerBackground()* aufgerufen. Sie führt den AsyncTask *RegistrationTask* aus.

```
regid = getRegistrationId(context);
if (regid.length() == 0) {
        registerBackground();
}
```

Abbildung 12: Überprüfung des Rückgabewertes der Methode getRegistrationId().

Ein AsyncTask ist eine Helferklasse, die einen Thread neben dem UI Thread ausführt, ohne dabei auf ein komplettes Programmiergerüst eines normalen Threads mit einem vorhandenen Handler zurückzugreifen. Ein AsyncTask sollte jedoch nur für kurze Tätigkeiten genutzt werden und kann den Umfang eines normalen Threads nicht ersetzen. Da innerhalb eines AsyncTask immer die Methode *doInBackground()* die Hauptaufgabe übernimmt, wird darin die Methode *register()* ausgeführt und wie in Abbildung 13 zu sehen, der Thread *UploadPostThread* erstellt und gestartet.

```
uploadThread = new UploadPostThread(myHandler, regid);
uploadThread.start();
```

Abbildung 13: Erstellen und starten des Threads UploadPostThread.

Die durch die Methode *register()* vom „Google Cloud Messaging“-Server zurückkommende Registration ID wird dem Thread bei der Erstellung mitübergeben. (Google Inc, 2013 c)

6.1.2 Broadcast Receiver

Der Broadcast Receiver wird dazu verwendet mit dem „Google Cloud Messaging“-Server zu kommunizieren und nimmt ankommende Benachrichtigungen entgegen. Dazu werden Intents verwendet, die vom „Google Cloud Messaging“-Server verschickt werden. Ein Intent ist eine asynchrone Nachricht, die es Komponenten von Applikationen erlauben die Funktionalität von anderen Android Komponenten anzufordern und mit ihnen zu interagieren. (Vogel, Vogella, 2013 a) Dazu werden meist Daten mitgeliefert welche zum Interagieren mit den Android Komponenten benötigt werden. Die vom „Google Cloud Messaging“-Server verschickten Intents bestehen zum einen aus dem Benachrichtigungs-Typ, dem so genannten MessageType und zum anderen aus den zusätzlichen Daten, welche dem Intent angehangen wurden. In den Daten befindet sich dann der eigentliche Text der Benachrichtigung, welche an den Benutzer des mobilen Endgerätes gerichtet ist. Diese wurde von einem Server aus an den „Google Cloud Messaging“-Server verschickt.

Innerhalb des Broadcast Receivers wird nach dem Erhalt des Intents daraufhin der Benachrichtigungstext in eine Push-Benachrichtigung integriert. Sie wird innerhalb des Notification Drawers des mobilen Endgerätes angezeigt.

6.1.2.1 Permissions

Um den Broadcast Receiver zu realisieren, muss jedoch zuerst - wie in Abbildung 14 dargestellt - eine *Permission*-Erweiterung im Manifest vorgenommen werden. So kann ein mobiles Endgerät mit der *Permission android.permission.WAKE_LOCK* bei eingehender Push-Benachrichtigung aus dem Ruhemodus aktiviert werden. Außerdem können durch die *Permission permission.C2D_MESSAGE* mit dem *protectionLevel signature* keine anderen Android Applikationen auf die Benachrichtigungen der Hochschul-App zugreifen.

```
<permission
        android:name="de.hsmw.app.permission.C2D_MESSAGE"
        android:protectionLevel="signature" />
<uses-permission android:name="de.hsmw.app.permission.C2D_MESSAGE" />
<uses-permission android:name="android.permission.WAKE_LOCK" />
```

Abbildung 14: Permissions im Manifest, um ein mobiles Endgerät aus dem Ruhemodus zu aktivieren und um zu verhindern, dass keine anderen Applikationen auf die Benachrichtigungen der Hochschul-App zugreifen können.

Außerdem muss der Broadcast Receiver, wie in Abbildung 15 zu sehen, im Manifest registriert werden, damit dieser in der Applikation genutzt werden kann. Die *Permission c2dm.intent.RECEIVE* muss gesetzt werden, damit nur der „Google Cloud Messaging"-Server Benachrichtigungen senden kann. Über den Intent-Filter wird daraufhin die eigentliche Benachrichtigung erhalten. (Google Inc, 2013 c)

```
<receiver
        android:name=".GcmBroadcastReceiver"
        android:permission="com.google.android.c2dm.permission.SEND" >
        <intent-filter>
                <action android:name="com.google.android.c2dm.intent.RECEIVE" />
                <category android:name="de.hsmw.app" />
        </intent-filter>
</receiver>
```

Abbildung 15: Registrierung des Broadcast Receivers im Manifest.

6.1.2.2 Intents

Die Klasse *GcmBroadcastReceiver* befindet sich in der Datei GcmBroadcastReceiver.java innerhalb des Paketes de.hsmw.app.gcm. In der *onReceive()*-Methode, welche sich in der Klasse *GcmBroadcastReceiver* befindet, wird, wie in Abbildung 16 dargestellt, der vom „Google Cloud Messaging"-Server verschickte Intent mit der Methode *getMessageType()* auf den Benachrichtigungs-Typ, der MessageType hin überprüft. Sollte der MessageType gleich MESSAGE_TYPE_SEND_ERROR bzw. MESSAGE_TYPE_DELETED sein, so wird die Methode *sendNotification()* aufgerufen und die Fehlermeldung „Send error: " bzw. „Deleted message on server: " sowie die zusätzlichen Daten des Intents als String mitgegeben. Die Methode *getExtras()* versucht dabei die Daten des Intents auszulesen.

```
GoogleCloudMessaging gcm = GoogleCloudMessaging.getInstance(context);
ctx = context;
String messageType = gcm.getMessageType(intent);
if (GoogleCloudMessaging.MESSAGE_TYPE_SEND_ERROR.equals(messageType)) {
	sendNotification("Send error: " + intent.getExtras().toString(), "");
} else if (GoogleCloudMessaging.MESSAGE_TYPE_DELETED.equals(messageType)) {
	sendNotification("Deleted messages on server: " +
	intent.getExtras().toString(), "");
}
```

Abbildung 16: Überprüfung des Intents auf den Benachrichtigungs-Typ.

Bei jeder anderen MessageType wird auch die Methode *sendNotification()* aufgerufen. Es werden jedoch, wie in Abbildung 17 dargestellt, nur diese zusätzlichen Daten des Intents mit der Methode *getExtras().getString()* ausgegeben, welche eine vorgegebene Variable besitzt, die in der Methode *getString()* angegeben werden muss. Dies sind zum einen die eigentliche Benachrichtigung und zum anderen die Notification ID.

Die Notification ID gibt in der Methode *sendNotification()* an, welche Activity bei der Betätigung der Benachrichtigung innerhalb des Notification Drawers angezeigt werden soll.

```
} else {
	sendNotification(intent.getExtras().getString("message").toString(),
	intent.getExtras().getString("notificationID").toString());
}
```

Abbildung 17: Aufruf der Methode sendNotification() mit der Weitergabe zweier String Variablen, die aus den zusätzlichen Daten des Intents entnommen werden.

6.1.2.3 Erstellung der Benachrichtigung

In der Methode *sendNotification()* wird der Benachrichtigungstext in eine Benachrichtigung integriert und dann auf dem mobilen Endgerät innerhalb des Notification Drawers angezeigt.

Dafür werden über die *NotificationManager*-Klasse Benachrichtigungen erstellt.

Um ein Benachrichtigungs-Objekt zu erstellen, dem spezielle Eigenschaften mitgegeben werden können, wird daraufhin, wie in Abbildung 18 zu sehen, die *NotificationCompat.Builder*-Klasse benutzt.

```
NotificationCompat.Builder mBuilder = new NotificationCompat.Builder(ctx)
```

Abbildung 18: Ein Benachrichtigungs-Objekt wird durch die Klasse NotificationCompat.Builder erstellt.

Um dem Benachrichtigungs-Objekt spezielle Eigenschaften mitzugeben, werden diese wie in Abbildung 19 dargestellt gesetzt:

```
.setSmallIcon(R.drawable.ic_hsmw_notification)
.setContentTitle("HSMWmobil")
.setStyle(new NotificationCompat.BigTextStyle()
.bigText(msg))
.setContentText(msg)
.setAutoCancel(true)
.setDefaults(Notification.DEFAULT_SOUND | Notification.DEFAULT_VIBRATE)
.setLights(0xff0000ff,100,3000)
.setPriority(Notification.PRIORITY_DEFAULT)
.setOnlyAlertOnce(true);
```

Abbildung 19: Methoden für die Eigenschaften des Benachrichtigungs-Objekts.

Die Methode *setSmallIcon()* legt das Icon fest, welches in der Benachrichtigung am oberen linken Rand sowie im Notification Area, wie in Abbildung 20 zu sehen, angezeigt werden soll. Bei dem Icon handelt es sich um das Hochschullogo. Dadurch kann der Benutzer anhand des Icons sehr schnell feststellen, dass es sich bei der erhaltenen Benachrichtigung um eine Mitteilung der Hochschul-App handelt, ohne die Benachrichtigung gelesen zu haben.

Abbildung 20: Anzeige des Hochschullogos als Icon innerhalb des Notification Areas.

Über die Methode *setContentTitle()* wird der Titel der Benachrichtigung festgelegt, welcher am oberen Rand der Benachrichtigung angezeigt wird. Bei dem Titel handelt es sich bei jeder Benachrichtigung um den Namen der Hochschul-App HSMWmobil, damit der Benutzer weiß, zu welcher Applikation die Benachrichtigung gehört.

Durch die Methode *setStyle()* wird das Aussehen der Benachrichtigung festgelegt. In diesem Fall ist es ein Big View Style, der es erlaubt in einer Benachrichtigung den Text über mehrere Zeilen anzeigen zu lassen, wie in Abbildung 21 dargestellt ist.

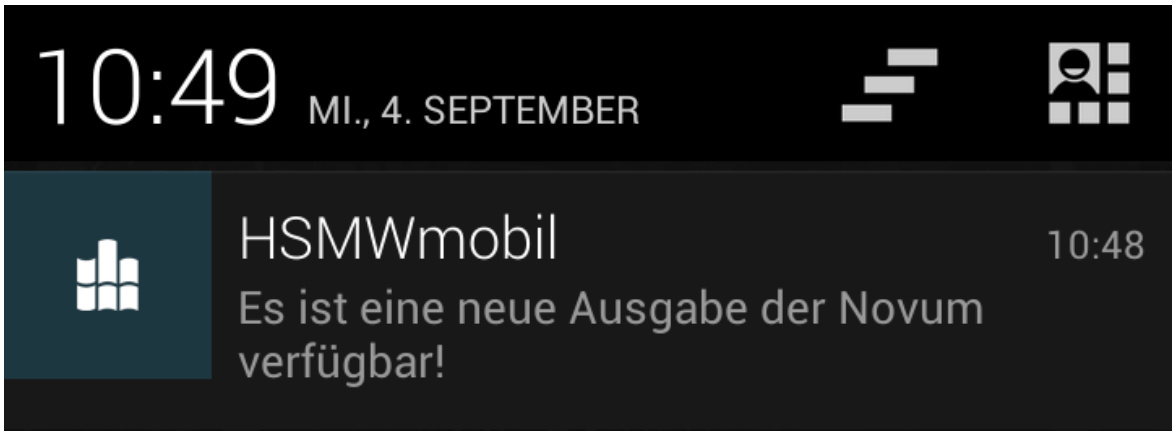

Abbildung 21: Mehrzeilige Benachrichtigung im Notification Drawer.

Sollten mehrere Benachrichtigungen eingegangen sein, so wird die Benachrichtigung verkleinert dargestellt. Dabei wird, wie in Abbildung 22 zu sehen, der über mehrere Zeilen gehende Text nach der ersten Zeile abgebrochen. Für den Benutzer besteht jedoch die Möglichkeit die Benachrichtigung wieder zu vergrößern und den Text in voller Länge anzeigen zu lassen. Dafür muss der Benutzer mit einer Wischgeste nach unten die Benachrichtigung vergrößern. Voraussetzung für eine Benachrichtigung im Big View Style ist jedoch mindestens ein mobiles Endgerät mit der Android Version Jelly Bean (4.1) bzw. dem API-Level 20. Sollte dies nicht vorhanden sein, so wird der Text in einer normalen einzeiligen Benachrichtigung angezeigt.

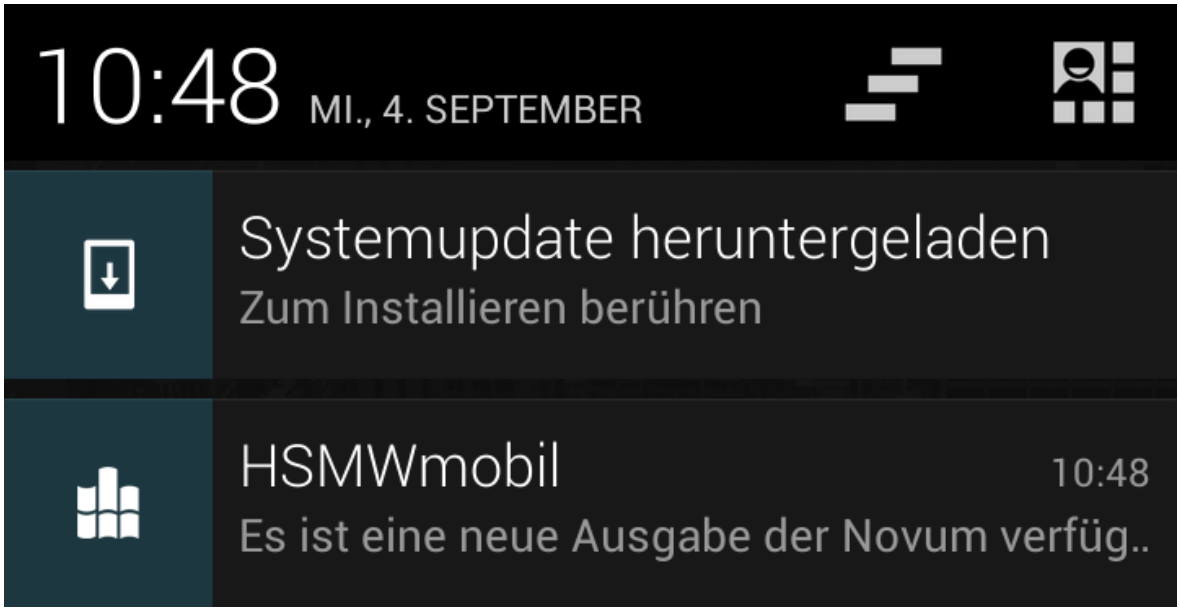

Abbildung 22: Einzeilige Benachrichtigung im Notification Drawer.

Die Methode *setContentText()* legt den Text für die Benachrichtigung fest, welcher die eigentliche Information für den Benutzer darstellt. Der angezeigte Text ist in der String Variable msg gespeichert. Die Variable msg wird über die Methode *onReceive()* aus den zusätzlichen Daten des Intents mit dem Benachrichtigungstext gefüllt. Durch den Methodenaufruf *setAutoCancel()* wird die Benachrichtigung gelöscht, sobald der Benutzer das Benachrichtigungsfeld betätigt.

Über die Methode *setDefaults()* können die voreingestellten Optionen für Benachrichtigungston, –vibration und -licht festgelegt werden. Da nur der voreingestellte Benachrichtigungston und die –vibration genutzt werden soll, sind nur diese in der Methode *setDefaults()* vorhanden. Für das Benachrichtigungslicht in der LED-Anzeige soll nicht die Standardfarbe weiß verwendet werden, sondern die für die Hochschule Mittweida wiedererkennbare blaue Farbe aus dem Corporate Design.

Somit kann der Benutzer, sofern dessen mobiles Endgerät über eine LED-Anzeige verfügt, anhand der LED-Farbe feststellen, dass es sich bei der erhaltenen Benachrichtigung um eine Mitteilung der Hochschul-App handelt. Über die Methode *setLights()* wird im ersten Wert festgelegt, in welcher Farbe die LED-Anzeige leuchten soll. Der zweite und dritte Wert gibt die Zeit in Millisekunden an, in der die LED-Anzeige leuchten bzw. nicht leuchten soll.

Die Methode *setPriority()* legt fest, wie wichtig die Benachrichtigung sein soll. Dafür wird die Standardeinstellung übernommen.

Durch die Methode *setOnlyAlertOnce()* wird die Benachrichtigung nicht noch einmal signalisiert, sollte diese schon aufgerufen sein. (Google Inc., 2013 d)

6.1.2.4 PendingIntent

Durch die Klasse *PendingIntent* wird, wie in Abbildung 23 dargestellt, ein neuer Intent mit dem Namen *contentIntent* erzeugt. Dieser führt, sobald der Benutzer das Benachrichtigungsfeld betätigt, über den *notificationIntent* eine Activity aus.

```
PendingIntent contentIntent = PendingIntent.getActivity(ctx, 0,
notificationIntent, PendingIntent.FLAG_UPDATE_CURRENT);
```

Abbildung 23: Die Klasse PendingIntent erzeugt einen neuen Intent, welcher den Intent notificationIntent startet.

Ein *PendingIntent* ist ein Token, welches einer anderen Applikation gegeben wird, damit diese *Permissions* anderer Applikationen nutzen kann, um einen vordefinierten Befehl ausführen zu können. (Vogel, Vogella, 2013 b)

Durch den mitgegebenen Parameter Flags innerhalb der Methode *PendingIntent.getActivity()* wird beschrieben, was getan werden soll, falls ein vorher erstellter *PendingIntent* noch existiert. Folgende Konstanten können verwendet werden:

- FLAG_CANCEL_CURRENT – Der vorher erstellte *PendingIntent* wird beendet, bevor der neue erstellt wird.
- FLAG_NO_CREATE – Der vorher erstellte *PendingIntent* bleibt bestehen, es wird kein neuer erstellt.
- FLAG_ONE_SHOT – Es kann nur ein *PendingIntent* erstellt werden.
- FLAG_UPDATE_CURRENT – Der vorher erstellte *PendingIntent* bleibt bestehen, nur die zusätzlichen Daten des alten *PendingIntent* werden durch die des neuen aktualisiert.

Da in der Hochschul-App kein neuer Intent erstellt werden muss, werden nur die zusätzlichen Daten durch die Konstante FLAG_UPDATE_CURRENT aktualisiert. (Google Inc., 2013 e)

Der gestartete Intent *notificationIntent* gibt darüber Auskunft, welche Activity ausgeführt wird, sobald der Benutzer das Benachrichtigungsfeld betätigt. Dies geschieht, wie in Abbildung 24 zu sehen, durch die Abfrage der Notification ID, welche aus den zusätzlichen Daten des Intent an die Methode *sendNotification()* geschickt wird. Diese wird in die Integer-Variable int_notificationID gespeichert. Durch eine Überprüfung der Variable int_notificationID wird festgestellt, um welche Notification ID es sich handelt. Jede Notification ID steht für eine andere Activity, die mit Hilfe eines neu erstellten Intents gestartet wird. Sollte die Notification ID den Integer-Wert „1“ beinhalten, so wird die Activity NovumPdfActivity aufgerufen und die Hochschulzeitung „Die Novum“ angezeigt. Sollte die Notification ID den Integer-Wert „2“, „3“ oder „4“ beinhalten, so wird die Activity NewsActivity aufgerufen. Durch die Methode *putExtra()* wird dem neu erstellten Intent ein Zuweisungs-Typ angehängt. Über den Zuweisungs-Typ wird festgelegt, welches Fragment innerhalb der NewsActivity angezeigt werden soll. Beinhaltet die Notification ID den Integer-Wert „2“, so wird dem Intent der Zuweisungs-Typ „news“ mitgegeben und das Fragment der HSMW News dargestellt. Beinhaltet die Notification ID allerdings den Integer-Wert „3“, so wird dem Intent der Zuweisungs-Typ „blog“ mitgegeben und das Fragment der HSMW Blog dargestellt. Beinhaltet die Notification ID jedoch den Integer-Wert „4“, so wird dem Intent der Zuweisungs-Typ „mm“ mitgegeben und das Fragment der medien-mittweida.de dargestellt.

```
int int_notificationID = Integer.valueOf(notificationID.toString());
Intent notificationIntent = new Intent();
if (int_notificationID == 1){notificationIntent = new Intent(ctx,
de.hsmw.app.novum.NovumPdfActivity.class);};
if (int_notificationID == 2){notificationIntent = new Intent(ctx,
de.hsmw.app.news.NewsActivity.class); notificationIntent.putExtra("ACTION",
"news");};
if (int_notificationID == 3){notificationIntent = new Intent(ctx,
de.hsmw.app.news.NewsActivity.class); notificationIntent.putExtra("ACTION",
"blog");};
if (int_notificationID == 4){notificationIntent = new Intent(ctx,
de.hsmw.app.news.NewsActivity.class); notificationIntent.putExtra("ACTION",
"mm");};
```

Abbildung 24: Abfrage der Notification ID und Erstellung eines neuen Intents, in dem - in Abhängigkeit der Notification ID – eine Activity gestartet wird.

Dem Intent *notificationIntent* wird, wie in Abbildung 25 dargestellt, über die Methode *setFlags()* mitgeteilt, was geschehen soll, falls die aufzurufende Activity bereits gestartet wurde.

```
notificationIntent.setFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP
|Intent.FLAG_ACTIVITY_SINGLE_TOP);
```

Abbildung 25: Aufruf der Methode setFlags() für den Intent notificationIntent.

Über die Konstante FLAG_ACTIVITY_CLEAR_TOP wird, sollte die aufzurufende Activity bereits gestartet sein, keine neue Instanz der Activity erstellt. Vielmehr werden alle Activitys, welche sich oberhalb der aufzurufenden Activity im Back Stack befinden, geschlossen. Der alte Intent für die bereits gestartete Activity wird dabei zum neuen Intent.

Activitys werden im Back Stack – eine Art Liste – in der Reihenfolge angeordnet, in welcher sie zeitlich geöffnet wurden. An oberster Stelle des Back Stack befindet sich die zuletzt geöffnete Activity. Jedes Mal, wenn eine neue Activity geöffnet wird, wird diese an oberster Stelle des Back Stack abgelegt. Sollte eine Activity geschlossen werden - mit dem Drücken des Zurück-Buttons auf dem mobilen Endgerät - so wird diese aus dem Back Stack genommen. Durch das Drücken des Home-Buttons auf dem mobilen Endgerät sowie beim Aufruf einer neuen Activity innerhalb einer anderen Activity, wird die Activity nicht geschlossen, sondern nur pausiert. Dabei wird sie im Back Stack abgelegt.

Activitys stehen an oberster Stelle des Back Stack, wenn sie bereits gestartet sind und im Vordergrund laufen. Ist dies bereits der Fall, bewirkt die Konstante FLAG_ACTIVITY_SINGLE_TOP, dass keine neue Activity aufgerufen wird.

6.2 Server

6.2.1 Anbindung der Applikation an den Webserver

Nachdem die Applikation bei "Google Cloud Messaging" registriert wurde, und so die Kommunikation zwischen der Applikation und dem "Google Cloud Messaging"-Server hergestellt wurde, muss die Kommunikation zwischen der Applikation und dem Webserver hergestellt werden.

Innerhalb des AsyncTask *RegistrationTask* wird über die Methode *register()*, wie in Kapitel 6.1.1.3 Registrierung beschrieben, die Registration ID vom „Google Cloud Messaging“-Server abgerufen. Daraufhin wird diese dem Thread *UploadPostThread* übergeben, welcher anschließend gestartet wird. Der Thread *UploadPostThread* verschickt daraufhin die Registration ID mit einem HTTP-Post an den Webserver. Dafür wird, wie in Abbildung 26 dargestellt, innerhalb des Konstruktors die Registration ID entgegengenommen und in einer String-Variablen gespeichert.

```
public UploadPostThread(Handler handler, String regid) {
	this.myHandler = handler;
	this.regid = regid;
}
```

Abbildung 26: Konstruktor, in dem der Handler und die Registration ID in Variablen gespeichert werden.

Über die *run()*-Methode wird, wie in Abbildung 27 zu sehen, als erstes die Methode *sendMsg()* aufgerufen und die Status-Benachrichtigung RegId_UPLOAD_STARTED mitübergeben. Die Status-Benachrichtigungen sollen Aufschluss darüber geben, ob der Upload der Registration ID an den Webserver erfolgreich durchgeführt wurde oder nicht.

```
@Override
public void run() {
	sendMsg(RegId_UPLOAD_STARTED);
```

Abbildung 27: Aufruf der sendMsg()-Methode mit Übergabe der Status-Benachrichtigung innerhalb der run()-Methode

Daraufhin wird, wie in Abbildung 28 dargestellt, ein HTTP-Client für die Verbindung zum Webserver sowie der eigentliche HTTP-Post erstellt. Der HTTP-Post beinhaltet innerhalb der Variable url_gcm die URL zum PHP-Skript ids.php. Dieses befindet sich auf dem Webserver. Mit dem PHP-Skript werden die Registration IDs in eine XML-Datei gespeichert.

```
HttpClient client = new DefaultHttpClient();
HttpPost post = new HttpPost(url_gcm);
```

Abbildung 28: Erstellung des HTTP-Clients und des HTTP-Posts innerhalb des Threads UploadPostThread.

Mit Hilfe der Registration ID wird, wie in Abbildung 29 zu sehen, ein Entity erzeugt, welches an den HTTP-Post geknüpft wird. Dabei wird zuerst ein neues ArrayList mit dem Namen nameValuePairs erstellt. Dem ArrayList wird die Registration ID mit der Methode *add()* hinzugefügt. Daraufhin wird dieses als Entity an den HTTP-Post gehängt.

```
List<NameValuePair> nameValuePairs = new ArrayList<NameValuePair>();
nameValuePairs.add(new BasicNameValuePair("regid", regid));
post.setEntity(new UrlEncodedFormEntity(nameValuePairs));
```

Abbildung 29: Erzeugung eines Entity, mit Hilfe eines ArrayList, welches die Registration ID beinhaltet und Anknüpfung an den HTTP-Post.

Ein HTTP-Response wird, wie in Abbildung 30 dargestellt, ausgelöst. Er verschickt den HTTP-Post an den Webserver.

```
HttpResponse response = client.execute(post);
```

Abbildung 30: HTTP-Response Auslösung, welcher den HTTP-Post an den Webserver verschickt.

Nachdem der HTTP-Post erfolgreich an den Webserver verschickt wurde, sendet dieser einen HTTP-StatusCode zurück an den Thread *UploadPostThread*.

Sollte dieser HTTP-StatusCode, wie in Abbildung 31 dargestellt, dem Integer-Wert „204“ entsprechen, so wird die Methode *sendMsg()* aufgerufen und die Status-Benachrichtigung RegId_UPLOAD_COMPLETE mitübergeben. Der HTTP-StatusCode 204 bedeutet, dass der Server die Aufgabe erfolgreich ausgeführt hat, jedoch ohne Daten an das mobile Endgerät (Client) zurückzusenden. Sollte der HTTP-StatusCode nicht dem Integer-Wert „204“ entsprechen oder aber ein anderer Fehler auftreten, so wird auch hier die Methode *sendMsg()* aufgerufen, jedoch mit der Status-Benachrichtigung RegId_UPLOAD_ERROR als Übergabewert.

```
try {
	...
	int responseCode = response.getStatusLine().getStatusCode();
	if (responseCode == 204) {
		sendMsg(RegId_UPLOAD_COMPLETE);
	} else {
		sendMsg(RegId_UPLOAD_ERROR);
	}
} catch (Exception e) {
	sendMsg(RegId_UPLOAD_ERROR);
}
```

Abbildung 31: Überprüfung des vom Webserver verschickten HTTP-StatusCodes und dem daraus folgenden Aufruf der Methode sendMsg() mit Übergabe der passenden Status-Benachrichtigung

Innerhalb der Methode *sendMsg()* wird daraufhin, wie in Abbildung 32 dargestellt, die Status-Benachrichtigung in die Message-Variable msg gespeichert und diese daraufhin an den Handler *UploadHandler* innerhalb der Datei HsmwApplication.java geschickt.

```
Message msg = Message.obtain(myHandler, message, 0, 0, 0);
myHandler.sendMessage(msg);
```

Abbildung 32: Versenden der Status-Benachrichtigung an den Handler UploadHandler innerhalb der Methode sendMsg().

Der Handler *UploadHandler* innerhalb der Datei HsmwApplication.java empfängt die vom Thread *UploadPostThread* gesendeten Status-Benachrichtigungen über den Versand

der Registration ID an den Webserver. Daraufhin werden die Status-Benachrichtigungen in der Methode *handleMessage()* überprüft. Sollte der Handler, wie in Abbildung 33 zu sehen, die Status-Benachrichtigung RegId_UPLOAD_STARTED empfangen, so wird die Nachricht, dass das Versenden der Registration ID gestartet ist, im LogCat über die Methode *Log.i()* ausgegeben. Diese Benachrichtigung ist für den Benutzer der Hochschul-App nicht sonderlich relevant und wird daher nur für den Entwickler im LogCat ausgegeben. Sie ist deshalb für den normalen Benutzer nicht sichtbar.

Das LogCat ist ein Android-Logging-System, in welchem alle Systemaufrufe und Veränderungen des Betriebssystems Android und dessen Applikationen aufgezeichnet werden. Dies ist für Entwickler hilfreich, da durch die Möglichkeit des Loggings schnell Fehler in der Entwicklung einer Applikation gefunden werden können.

```
@Override
public void handleMessage(Message msg) {
	HsmwApplication activity = ref.get();
	switch(msg.what) {
		case UploadPostThread.RegId_UPLOAD_STARTED:
			Log.i(TAG, R.string.gcm_start + "");
			break;
		...
	}
}
```

Abbildung 33: Überprüfung der Status-Benachrichtigungen im Handler UploadHandler und Ausgabe der entsprechenden Benachrichtigung im LogCat.

Sollte der Handler, wie in Abbildung 34 dargestellt, die Status-Benachrichtigung RegId_UPLOAD_COMPLETE erhalten, so wird die Nachricht, dass die Anmeldung der Push-Benachrichtigung erfolgreich war als Toast für den Benutzer sichtbar gemacht. Ein Toast ist ein kleines Pop-up-Fenster, welches im unteren Bereich des Bildschirmes Text anzeigt. Dieses wird dazu verwendet dem Benutzer kurze Informationen mitzuteilen.

```
case UploadPostThread.RegId_UPLOAD_COMPLETE:
	Toast.makeText(activity, R.string.gcm_successful,
	Toast.LENGTH_LONG).show();
	break;
```

Abbildung 34: Bei Erhalt der Status-Benachrichtigung RegId_UPLOAD_COMPLETE, wird die entsprechende Benachrichtigung als Toast ausgegeben.

Wie in Abbildung 35 zu sehen, wird beim Erhalt der Status-Benachrichtigung RegId_UPLOAD_ERROR als Toast die Nachricht ausgegeben, dass die Anmeldung der Push-Benachrichtigung fehlgeschlagen ist.

```
case UploadPostThread.RegId_UPLOAD_ERROR:
      Toast.makeText(activity, R.string.gcm_error,
      Toast.LENGTH_LONG).show();
      break;
```

Abbildung 35: Bei Erhalt der Status-Benachrichtigung RegId_UPLOAD_ERROR, wird die entsprechende Benachrichtigung als Toast ausgegeben.

6.2.2 Webserver

Die Registration IDs der mobilen Endgeräte sollen in einer Datenbank oder XML-Datei gespeichert werden. Diese sollen dann mitsamt der Benachrichtigung an den „Google Cloud Messaging“-Server gesendet werden. Dafür wird ein Webserver zwischen den mobilen Endgeräten und dem „Google Cloud Messaging“-Server benötigt.

Zu Beginn der Bachelorarbeit besitzt der Autor noch keinen Zugriff und keine Berechtigung auf dem HSMWmobil-Applikation-Server. Daher wird zu Testzwecken aus einem lokalen PC ein Webserver erstellt. Dazu ist die Installation von XAMPP erforderlich. XAMPP ist eine Zusammenstellung von Softwarekomponenten um beispielsweise einen Apache-Webserver zu erstellen. Dieser soll als Entwicklungsumgebung dienen, auf dem die PHP-Skripte ausgeführt werden. Jedoch kommt es bei der Benutzung von XAMPP im Hochschulnetz zu Problemen. Das mobile Endgerät, welches zu Testzwecken verwendet wird, kann nicht immer auf den Webserver zugreifen. Womöglich wird dies durch die Benutzung von unterschiedlichen Subnetzen ausgelöst, die der Webserver – der lokale PC auf dem XAMPP läuft - und das mobile Endgerät nutzen. Dadurch kann das mobile Endgerät die vorgegebene URL des Webservers mit festgelegter IP-Adresse nicht aufrufen.

Es war nicht möglich, die PHP-Skripte innerhalb des für Studenten der Hochschule Mittweida zur Verfügung stehenden Webspeichers auszuführen. Der hochschuleigene Server hat die für den Betrieb der PHP-Skripte notwendige cURL-Programmbibliothek nicht installiert. Da der Autor keine Berechtigung zum Installieren der benötigten cURL-Programmbibliothek zu Testzwecken besitzt und nur einen begrenzten Zugriff auf den hochschuleigenen Server verfügt, ist die Benutzung des Webspeichers des Autors als Webserver nicht möglich.

Zum Ende der Bachelorarbeit werden der Zugriff und eine Berechtigung des HSMWmobil-Applikation-Servers für den Autor eingerichtet.

Auf diesem Server ist die notwendige cURL-Programmbibliothek bereits installiert. Darauf ist es dann möglich die PHP-Skripte zu Testzwecken als auch später für den Live-Betrieb auszuführen.

Die Implementierung des eigenen Servers erfolgt mit PHP und HTML. Der Server nimmt die Registration ID des mobilen Endgerätes entgegen und speichert diese in einer Datenbank oder XML-Datei ab. Die Benutzung einer Datenbank ist für viele Datensätze die bessere und schnellere Lösung. Der Geschwindigkeitsvorteil einer Datenbank gegenüber einer XML-Datei kann bei einer geringen Anzahl an Datensätzen wie den Registration IDs kaum ausgenutzt werden.

Das Ablegen der Datensätze in einer XML-Datei stellt des Weiteren die einfachere, mit weniger Aufwand verbundene Lösung dar. Daher wird die Benutzung einer XML-Datei der einer Datenbank vorgezogen.

6.2.2.1 DOM-Dokument Objekt

Die PHP-Skripte, welche sich auf dem Webserver befinden, sind in mehreren PHP-Dateien aufgeteilt. Die PHP-Datei ids.php nimmt die Registration IDs, welche per HTTP-Post vom Thread *UploadPostThread* geschickt werden, entgegen und speichert diese in einer XML-Datei ab. Dabei wird, wie in Abbildung 36 dargestellt, in der Klasse *Ids* die Methode *addId()* aufgerufen. Darin wird überprüft, ob der gesendete HTTP-Post leer ist. Sollte dies der Fall sein, so wird eine Fehlermeldung ausgelöst, dass der HTTP-Post leer ist.

```
class Ids {
	public function addId() {
		try {
			if (empty($_POST['regid'])) throw new
			Exception('$_POST[\'regid\'] is null');
		...
		}
	}
}
```

Abbildung 36: Überprüfung ob der gesendete HTTP-Post leer ist, und Ausgabe einer Fehlermeldung sollte dies der Fall sein.

Es wird, wie in Abbildung 37 zu sehen, ein neues Document Object Model (DOM)-Dokument Objekt erstellt, welches als Root-Element der XML-Datei dient. Ein Document Object Model (DOM) ist eine plattform- und programmiersprachenunabhängige Schnittstelle. Sie ermöglicht es, Programmen und Skripten Zugriff auf ein Dokument, zum Beispiel ein XML-Dokument, zu erlangen. Dort kann der Inhalt, die Struktur sowie der Style eines Dokumentes dynamisch verändert werden. (W3C, 2005)

```
$dom = new DOMDocument('1.0', 'utf-8');
```

Abbildung 37: Erstellung eines neuen Document Object Model (DOM)-Dokument Objektes innerhalb der Methode addId().

Dem neu erstellten Document Object Model (DOM)-Dokument Objekt werden die in Abbildung 38 dargestellten Eigenschaften gegeben. Durch die Eigenschaft *formatOutput* werden die Elemente, welche in der XML-Datei gespeichert werden, formatiert abgelegt. Dabei wird jedes Element zum jeweiligen nächst oberen Element eingerückt. Dadurch erhält man die gewünschte XML-Struktur, welche sich in Anlage Teil 1 befindet.

Durch die Eigenschaft *preserveWhiteSpace* wird der gesamte Leerraum, welcher durch Zeilenumbrüche, Leerzeichen und Tabulatoren erstellt wurde, ignoriert.

```
$dom -> formatOutput = true;
$dom -> preserveWhiteSpace = false;
```

Abbildung 38: Eigenschaften zum Formatieren der XML-Datei, die dem DOM-Dokument gegeben werden.

Damit die zu speichernde XML-Datei schnell gefunden wird, wird diese in dem Unterordner files unter dem Dateinamen ids.xml abgespeichert. Wie in Abbildung 39 zu sehen, wird der Dateipfad in der Variable $file gespeichert.

```
$file = 'files/ids.xml';
```

Abbildung 39: Abspeicherung des Dateipfades in der Variable $file.

Daraufhin wird überprüft, ob schon eine XML-Datei unter dem vorbestimmten Dateipfad angelegt wurde. Ist dies der Fall, so wird die schon vorhandene XML-Datei, wie in Abbildung 40 dargestellt, eingelesen, damit diese erweitert werden kann.

```
$dom -> load($file);
```

Abbildung 40: Einlesen der schon angelegten XML-Datei, damit diese erweitert werden kann.

Es wird, wie in Abbildung 41 zu sehen, das Root- sowie das erste Element der schon vorhandenen XML-Datei ermittelt.

```
$root = $dom -> firstChild;
$first = $root -> firstChild;
```

Abbildung 41: Ermittlung des Root- sowie des ersten Elements der schon vorhandenen XML-Datei.

Nachdem die Elemente ermittelt wurden, wird, wie in Abbildung 42 dargestellt, ein Kind-Element mit der Bezeichnung *device* über die Methode *createElement()* erstellt und in die Variable $node abgespeichert. Dieses wird als letztes Element über die Methode *insertBefore()* in die schon vorhandene XML-Datei angefügt.

```
$root -> insertBefore($node = $dom -> createElement("device"), $last);
```

Abbildung 42: Erstellung eines Kind-Elements mit der Bezeichnung device und Anfügen dieses Elements als letztes Element der schon vorhandenen XML-Datei.

Sollte jedoch noch keine XML-Datei angelegt worden sein, so wird, wie in Abbildung 43 zu sehen, das Root-Element mit der Bezeichnung *root* über die Methode *createElement()* erstellt. Daraufhin wird es an das Document Object Model (DOM)-Dokument Objekt über die Methode *appendChild()* angehängt.

```
$root = $dom -> createElement('root');
$dom -> appendChild($root);
```

Abbildung 43: Erstellung des Root-Elements mit der Bezeichnung root und Anhängen dieses Elements an das DOM-Dokument Objekt.

Auch hier wird, wie schon in Abbildung 42 gezeigt, ein Kind-Element mit der Bezeichnung *device* erstellt.

Die Registration ID soll, wie in Abbildung 44 dargestellt, eine Eigenschaft des soeben erstellten Elements *device* sein, welche unter der Variable $node abgespeichert wurde. Dazu wird die Registration ID aus dem HTTP-Post in einem neue erstellten Element mit der Bezeichnung *regid* über die Methode *createElement()* abgespeichert. Dieses Element wird daraufhin über die Methode *appendChild()* als Kind-Element des Elements device erstellt.

```
$node -> appendChild($dom -> createElement('regid', $_POST['regid']));
```

Abbildung 44: Die Registration ID aus dem HTTP-Post wird in einem neuen Kind-Element als Eigenschaft des Elements device gespeichert.

Sollte noch keine XML-Datei erstellt worden sein, wird, wie in Abbildung 45 zu sehen, mit der Methode *fopen()* und dem speziellen mode-Parameter 'w' eine XML-Datei erstellt. Ansonsten wird diese nur geöffnet. Der mode-Parameter spezifiziert den Zugriffstyp.

Der mode-Parameter 'w' öffnet eine Datei nur zum Schreiben. Sollte diese Datei jedoch nicht vorhanden sein, so wird versucht diese zu erzeugen. (The PHP Group, 2013 a)

```
$file = fopen($file, 'w');
```

Abbildung 45: Öffnet bzw. erzeugt eine neue XML-Datei unter dem in der Variable $file angegebenen Dateipfad.

In die geöffnete oder erzeugte XML-Datei wird, wie in Abbildung 46 dargestellt, das neu erstellte Document Object Model (DOM)-Dokument Objekt mit der Methode *fwrite()* geschrieben. Anschließend wird die XML-Datei mit der Methode *fclose()* geschlossen.

```
fwrite($file, $dom -> saveXML());
fclose($file);
```

Abbildung 46: Schreiben des DOM-Dokument Objektes in die XML-Datei und das Schließen dieser Datei hinterher.

6.2.2.2 HTTP-StatusCode

Es wird ein Header an das mobile Endgerät (Client) gesendet, welcher einen HTTP-StatusCode beinhaltet. Ein HTTP-StatusCode wird von einem Server als Antwort versandt, sobald dieser eine HTTP-Anfrage von einem Client erhält. Dadurch bekommt der Client eine Information, ob seine HTTP-Anfrage erfolgreich ausgeführt wurde. Ist dies nicht der Fall, so kann der HTTP-StatusCode darüber Auskunft geben, wo ein Fehler aufgetreten ist.

Wie in Abbildung 47 zu sehen, beinhaltet der Header den HTTP-StatusCode „204 No Content“. Dies bedeutet, dass der Server die Aufgabe erfolgreich ausgeführt hat. Es werden jedoch keine weiteren Daten an das mobile Endgerät (Client) zurückgesandt. Sollte ein Fehler auftreten, so wird der HTTP-StatusCode „400 Bad request“ versandt. Dies bedeutet, dass die vom Client gesendete HTTP-Anfrage fehlerhaft ist und daher nicht ausgeführt wurde.

```
        header("HTTP/1.0 204 No Content");
} catch(Exception $e) {
        header("HTTP/1.0 400 Bad request");
}
```

Abbildung 47: Versenden eines Headers an den Client mit einem HTTP-StatusCode.

Der HTTP-StatusCode wird innerhalb des Clients im Thread *UploadPostThread* entgegengenommen. Dort wird daraufhin die entsprechende Status-Benachrichtigung, wie in Kapitel 6.2.1 Anbindung der Applikation an den Webserver verdeutlicht, aufgerufen.

6.2.2.3 Benutzeroberfläche

Die PHP-Datei index.php bietet dem Versender der Push-Benachrichtigungen eine einfache Benutzeroberfläche. Sie kann über jeden üblichen Browser angezeigt und benutzt werden. In der Benutzeroberfläche befindet sich für jede Art von Push-Benachrichtigung ein Button, um die Push-Benachrichtigung loszusenden, sowie eine Tabelle, in der alle Registration IDs angezeigt werden.

PHP-Befehle können nur beim Aufruf einer Datei vom Server ausgeführt werden. Daher ist es nicht möglich die PHP-Befehle, welche zum Senden der Push-Benachrichtigungen benötigt werden, in dieselbe PHP-Datei zu schreiben. Ebenso kann diese nicht durch die Betätigung eines Buttons aufgerufen werden. Daher werden bei der Betätigung eines der Buttons die jeweilige PHP-Datei geöffnet und die Befehle zum Senden der Push-Benachrichtigung ausgeführt.

Um die Tabelle mit den vorhandenen Registration IDs zu füllen, muss, wie in Abbildung 48 dargestellt, als erstes die XML-Datei ids.xml, mit der Methode *simplexml_load_file()* aus dem Unterordner files geladen und auf die Variable $idfile gespeichert werden. In dieser sind die Registration IDs abgespeichert.

```
$idfile = simplexml_load_file('files/ids.xml');
```

Abbildung 48: XML-Datei ids.xml wird aus dem Unterordner files geladen.

Daraufhin werden alle Registration IDs, wie in Abbildung 49 zu sehen, mit Hilfe einer *for*-Schleife aus der geladenen XML-Datei mit der Bezeichnung $idfile ausgelesen. Sie werden in ein neu erstelltes Array mit dem Namen $ids gespeichert.

```
for ($i = 0; $i <= $idfile->device->count()-1; $i++) {
      $ids[$i] = '' . $idfile->device[$i]->regid . '';
}
```

Abbildung 49: Registration IDs aus der XML-Datei werden in einem Array gespeichert.

Wie in Abbildung 50 dargestellt, wird durch das Element <table> eine Tabelle erstellt, in der alle Registration IDs aus dem Array angezeigt werden. Das Sprach-Konstrukt echo lässt die Tabelle auf der Benutzeroberfläche anzeigen. Durch das Element <tr> bzw. <td> wird eine Tabellenzeile (**t**able **r**ow) bzw. die Tabellendaten (**t**able **d**ata) oder auch Tabellenspalte erstellt. Mit Hilfe einer *for*-Schleife wird das Array durchlaufen und alle vorhandenen Registration IDs in der Tabelle angefügt.

```
echo '<table border="1">';
       echo '<tr>';
       echo "<td><b>Device #</b></td>";
       echo "<td><b>Registration ID</b></td>";
       echo '</tr>';
for ($i = 0; $i <= $idfile->device->count()-1; $i++) {
       $k = $i+1;
       echo '<tr>';
       echo "<td>$k</td>";
       echo "<td>$ids[$i]</td>";
       echo '</tr>';
}
echo "\n</table>";
```

Abbildung 50: Tabelle in der alle Registration IDs aus dem Array angezeigt werden.

Somit ist es möglich, die PHP-Datei index.php mit der vorhandenen Benutzeroberfläche zum Versenden der Push-Benachrichtigungen über jeden üblichen Browser, wie in Abbildung 51 zu sehen, anzeigen zu lassen.

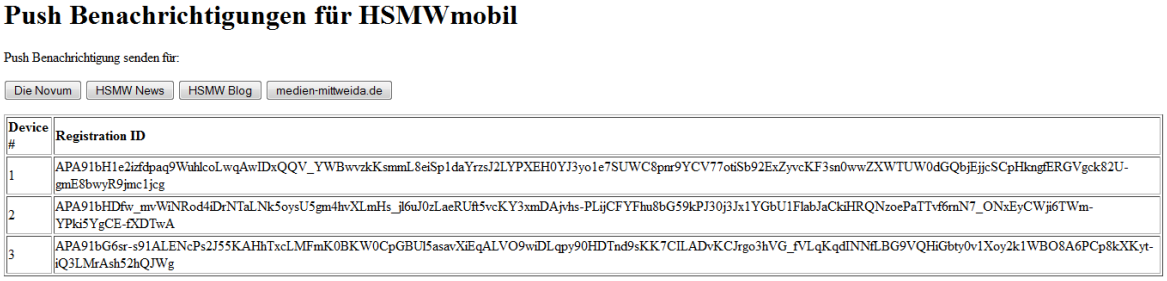

Abbildung 51: Ansicht der Benutzeroberfläche zum Versenden der Push-Benachrichtigungen im Browser.

6.2.2.4 Versenden von Push-Benachrichtigungen

Nachdem der Benutzer die Buttons betätigt hat, führen die PHP-Dateien die eigentlichen Befehle zum Senden der Push-Benachrichtigung aus. Sie benutzen einen HTTP-Post, um die Registration IDs sowie die Push-Benachrichtigung an den „Google Cloud Messaging“-Server zu senden.

Dazu wird, wie schon in der PHP-Datei index.php und in Abbildung 48 dargestellt, die XML-Datei ids.xml aus dem Unterordner files geladen.

Daraufhin werden die Registration IDs, wie schon in Abbildung 49 zu sehen war, mit Hilfe einer *for*-Schleife aus der geladenen XML-Datei ausgelesen. Sie werden in ein neu erstelltes Array mit dem Namen $ids gespeichert.

Unter der Variable $message wird der Benachrichtigungstext gespeichert. Dieser lautet zum Beispiel: „Es ist eine neue Ausgabe der Novum verfügbar!“. Damit der Benachrichtigungstext auch jegliches Zeichen speichern kann, wird dieser, wie in Abbildung 52 dargestellt, in die Zeichenkodierung UTF-8 umgewandelt. Somit können auch Umlaute und Sonderzeichen in der Push-Benachrichtigung benutzt werden.

```
$message = utf8_encode($message);
```

Abbildung 52: Umwandlung des Benachrichtigungstextes in die Zeichenkodierung UTF-8.

Die Post-Variablen werden, wie in Abbildung 53 zu sehen, festgelegt. Diese bestehen zum einen aus der Variable $url, in dem die URL des „Google Cloud Messaging“-Servers angegeben ist, an welche der HTTP-Post verschickt wird. Zum anderen bestehen die Variablen aus zwei Arrays.

Dem Array $fields, welches wiederum aus zwei Arrays zusammengesetzt ist. Das ist das Array $registrationIDs, in der alle Registration IDs vorhanden sind, sowie ein Array in dem der Benachrichtigungstext über die Variable $message sowie eine Notification ID vorhanden sind. Die Notification ID wird dazu benötigt, um festzulegen, welche Activity bei der Betätigung des Benachrichtigungsfelds innerhalb des Notification Drawers angezeigt werden soll.

Das zweite Array der Post-Variablen mit dem Namen $headers besteht aus dem Server-API-Key sowie dem Content-Type, welcher dem „Google Cloud Messaging“-Server mitteilt, dass der HTTP-Post im JSON-Format vorliegt, wie dieser es verlangt.

Der Server-API-Key, welcher aus dem Projekt HSMWmobil, von der Internetseite Google API Code stammt, dient zur Authentisierung am „Google Cloud Messaging“-Server.

```
$url = 'https://android.googleapis.com/gcm/send';

$fields = array(
                    'registration_ids'  => $registrationIDs,
                    'data'              => array( "message" => $message,
                    "notificationID" => 1),
                );

$headers = array(
                    'Authorization: key=' . $apiKey,
                    'Content-Type: application/json'
                 );
```

Abbildung 53: Festlegung der Variablen für den HTTP-Post (Nava, 2013).

Der HTTP-Post wird mit Hilfe der cURL-Programmbibliothek versendet. Dazu wird, wie in Abbildung 54 dargestellt, eine cURL-Session initialisiert.

```
$ch = curl_init();
```

Abbildung 54: Initialisierung einer cURL-Session (Nava, 2013).

Nun werden, wie in Abbildung 55 zu sehen, über die Methode *curl_setopt()* Optionen für den cURL-Transfer gesetzt.

```
curl_setopt( $ch, CURLOPT_URL, $url );

curl_setopt( $ch, CURLOPT_POST, true );
curl_setopt( $ch, CURLOPT_HTTPHEADER, $headers);
curl_setopt( $ch, CURLOPT_RETURNTRANSFER, true );
curl_setopt($ch, CURLOPT_SSL_VERIFYPEER, false);

curl_setopt( $ch, CURLOPT_POSTFIELDS, json_encode( $fields ) );
```

Abbildung 55: Setzen der Optionen für den cURL-Transfer (Nava, 2013).

In der Option CURLOPT_URL wird die URL des „Google Cloud Messaging"-Servers angegeben, an welche der HTTP-Post verschickt wird. Die Option CURLOPT_POST hat den boole´schen Wert *true*. Dadurch wird ein HTTP-Post-Request abgesetzt. Durch die Option CURLOPT_HTTPHEADER wird der HTTP-Header weitergegeben, welcher aus dem Array $headers besteht, in dem der Server-API-Key sowie der Content-Type gespeichert ist. Die Option CURLOPT_RETURNTRANSFER hat den boole´schen Wert *true*, damit die Abfrage, nachdem der HTTP-Post ausgeführt wurde, als String zurückgeliefert wird, anstatt sie direkt auszugeben. Über die Option CURLOPT_SSL_VERIFYPEER wird die Überprüfung des SSL-Zertifikats mit dem boole´schen Wert *false* unterdrückt. Im Live-Betrieb muss die Überprüfung erfolgen und ein SSL-Zertifikat vorhanden sein. Die Option CURLOPT_POSTFIELDS stellt die Daten zur Verfügung, die in einem HTTP-Post übertragen werden sollen. Über die Funktion *json_encode()* wird das Array $fields in JSON-Format umgewandelt. Dies wird benötigt, da für den „Google Cloud Messaging"-Server die Daten des HTTP-Post-Requests im JSON-Format vorliegen müssen. (The PHP Group, 2013 b)

Damit der HTTP-Post verschickt wird, muss die cURL-Session, wie in Abbildung 56 dargestellt, mit der Methode *curl_exec()* ausgeführt werden.

```
$result = curl_exec($ch);
```

Abbildung 56: Der HTTP-Post wird durch die Ausführung der cURL-Session verschickt (Nava, 2013).

Daraufhin wird die cURL-Session, wie in Abbildung 57 zu sehen, mit der Methode *curl_close()* wieder beendet.

```
curl_close($ch);
```

Abbildung 57: Beendet die cURL-Session (Nava, 2013).

Zur Überprüfung wird dem Benutzer, wie in Abbildung 58 dargestellt, der Text der versandten Push-Benachrichtigung im Browser ausgegeben und ein Button angezeigt. Bei Betätigung des Buttons gelangt der Benutzer auf die vorhergehende Benutzeroberfläche, um weitere Push-Benachrichtigungen zu versenden.

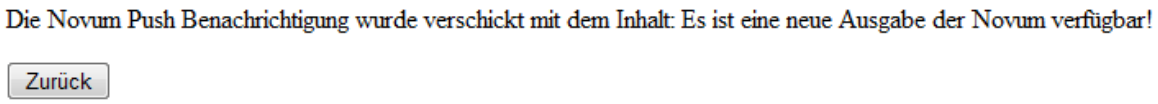

Abbildung 58: Ausgabe des versandten Push-Benachrichtigungstextes im Browser.

Jede Push-Benachrichtigung hat einen Payload von maximal 4096 Bytes und ist daher nur für kurze Text-Benachrichtigungen gedacht. Die Push-Benachrichtigung wird voreingestellte vier Wochen auf dem „Google Cloud Messaging“-Server gespeichert, sollte die Push-Benachrichtigung das mobile Endgerät (Client) vom „Google Cloud Messaging“-Server nicht erreichen können. Sobald jedoch innerhalb des HTTP-Posts in dem Array fields ein Time to Live-Wert spezifiziert wurde, gilt dieser. Er legt fest, wie lange versucht werden soll, die Benachrichtigung an das mobile Endgerät (Client) mit der passenden Registration ID zu senden. Sollte das mobile Endgerät (Client) in dieser Zeit vom „Google Cloud Messaging“-Server nicht verfügbar sein, so wird die Benachrichtigung gelöscht. (Google Inc., 2013 f)

7 Tests

Durch den Test der entwickelten Software soll festgestellt werden, ob diese auf unterschiedlichen Testumgebungen, wie beispielsweise unterschiedliche Android Versionen, sowie bei verschiedenartigen Internetverbindungen fehlerfrei funktioniert. Die Funktionsfähigkeit wird auf möglichst alle vorstellbaren Zustände hin überprüft. Sollten Fehler innerhalb der Software entdeckt werden, so werden diese beseitigt und die Software daraufhin erneut überprüft.

7.1 Test der Applikation

Der Funktionstest der Applikation HSMWmobil, mit der Möglichkeit Push-Benachrichtigungen zu empfangen, wird auf verschiedenen mobilen Endgeräten und Android Virtual Devices (AVD) mit unterschiedlicher Android Version ausgeführt, um möglichst viele Situationen in denen die Applikation ausgeführt werden kann abzudecken.

Mobiles Endgerät	Android Version	API-Level	Ergebnis
Sony Ericsson Xperia mini	Gingerbread (2.3.4)	10	Erster Funktionstest nicht bestanden. Erst durch *Permission*-Erweiterung im Manifest konnte der Funktionstest bestanden werden.
Samsung Galaxy S Plus	Gingerbread (2.3.6)	10	Funktionstest bestanden
Sony Ericsson Xperia U	Ice Cream Sandwich (4.0.4)	15	Funktionstest bestanden
Samsung Galaxy Nexus	Jelly Bean (4.2.2)	17	Funktionstest bestanden

Tabelle 1: Mobile Endgeräte, auf denen der Funktionstest ausgeführt wurde.

Der Funktionstest der Applikation HSMWmobil auf dem Testgerät Sony Ericsson Xperia mini mit der Android Version Gingerbread (2.3.4) bzw. dem API-Level 10 verlief beim ersten Test nicht zufriedenstellend. Beim Erhalt einer Push-Benachrichtigung stürzte die Applikation mit einer Fehlermeldung, welche in Abbildung 59 zu sehen ist, ab.

```
FATAL EXCEPTION: main
java.lang.RuntimeException: Unable to start receiver
de.hsmw.app.gcm.GcmBroadcastReceiver: java.lang.SecurityException: Requires
VIBRATE permission
```

Abbildung 59: Fehlermeldung bei Erhalt der Push-Benachrichtigung auf dem Testgerät Sony Ericsson Xperia mini.

Erst durch eine *Permission*-Erweiterung im Manifest, wie in Abbildung 60 dargestellt, konnte der Fehler behoben werden und das Testgerät Sony Ericsson Xperia mini Push-Benachrichtigungen erhalten.

```
<uses-permission android:name="android.permission.VIBRATE" />
```

Abbildung 60: Permission-Erweiterung im Manifest, um den Fehler zu beheben.

Weitere Funktionstests wurden auf den Testgeräten Samsung Galaxy Nexus mit der Android Version Jelly Bean (4.2.2) bzw. dem API-Level 17, dem Sony Ericsson Xperia U mit der Android Version Ice Cream Sandwich (4.0.4) bzw. dem API-Level 15 sowie dem Samsung Galaxy S Plus unter Android Version Gingerbread (2.3.6) bzw. dem API-Level 10 ohne Probleme durchgeführt. Für die Funktionstests wurde die Internetverbindung der mobilen Endgeräte über WLAN sowie über das mobile Internet mit dem Datenübertragungsverfahren HSDPA hergestellt. Alle Funktionstests konnten nach der Beseitigung des Fehlers auf dem Testgerät Sony Ericsson Xperia mini erfolgreich durchgeführt werden.

7.2 Test des Webservers

Die PHP-Dateien, welche sich auf dem Webserver befinden, werden auf Fehler und auf ihre Funktionsfähigkeit überprüft.

Die Überprüfung aller PHP-Dateien auf Fehler, die sich auf den Doctype XHTML 1.0 Strict beziehen, werden mit dem Markup Validation Service des World Wide Web Consortiums (W3C) durchgeführt. Dieser wird auf deren Internetseite zur Verfügung gestellt. Über den Markup Validation Service werden Fehler auf Internetseiten gefunden und hervorgehoben. Fehler sind solche, welche nicht mit dem eingestellten Document Type Definition übereinstimmen. Durch den Markup Validation Service konnten kleine Fehler in den PHP-Dateien entdeckt werden, die nicht mit dem Doctype XHTML 1.0 Strict übereinstimmten. Die Fehler wurden beseitigt, sodass alle PHP-Dateien dem Doctype XHTML 1.0 Strict entsprechen.

Außerdem fand eine Überprüfung der fehlerfreien Darstellung der Benutzeroberfläche zum Versenden der Push-Benachrichtigungen auf unterschiedlichen Browsern statt. Dabei wurden alle Funktionen erfolgreich auf folgenden Browsern getestet:

- Mozilla Firefox in der Version 23.0.1 von Mozilla Foundation
- Google Chrome in der Version 29.0.1547.62 m von Google Inc.
- Internet Explorer in der Version 10.0.9200.16660 von Microsoft Corporation

Um die PHP-Dateien auf ihre Funktionsfähigkeit zu überprüfen, wird der Google Chrome Browser mit der Erweiterung Postman – REST Client benutzt. Die Erweiterung Postman – REST Client ist ein HTTP-Client, mit dem Webdienste getestet werden können. Dadurch ist es möglich, einen HTTP-Post mit vordefiniertem Inhalt an eine vorgegebene URL zu senden. Um die Funktionsfähigkeit der PHP-Dateien zu testen, wird ein HTTP-Post mit einer selbst erstellten Registration ID an die URL der PHP-Datei ids.php gesendet. Nun wird überprüft, ob die versandte Registration ID mit Hilfe der PHP-Datei ids.php in die XML-Datei ids.xml gespeichert wurde. Dadurch wird eine erfolgreiche Anmeldung eines mobilen Endgerätes beim Dienst „Google Cloud Messaging“ vorgetäuscht. Bei einer erfolgreichen Anmeldung sendet das mobile Endgerät seine Registration ID mit Hilfe eines HTTP-Posts an die PHP-Datei ids.php. Diese Möglichkeit der Versendung von HTTP-Posts stellt eine große Erleichterung für den Entwickler da. Dieser hätte ansonsten bei jedem Testlauf, bei dem ein HTTP-Post versendet werden sollte, die Applikation auf einem mobilen Endgerät installieren müssen.

Ein weiterer Funktionstest der PHP-Dateien sieht vor, ein HTTP-Post mit der Registration ID sowie der Push-Benachrichtigung an den „Google Cloud Messaging“-Server zu senden. Dies geschieht durch die Betätigung eines der Buttons auf der Benutzeroberfläche. Dabei sollte der HTTP-Post erfolgreich abgeschickt werden und alle Daten im HTTP-Post wie zum Beispiel der Server-API-Key im Header und den Registration IDs aus den Daten korrekt sein.

Ist dies der Fall, so versendet der „Google Cloud Messaging“-Server die Push-Benachrichtigung an die entsprechenden mobilen Endgeräte. Hierdurch wird überprüft, das der HTTP-Post, welcher an den „Google Cloud Messaging“-Server versendet wird, korrekt ist.

Nachdem eine Push-Benachrichtigung vom Webserver an die mobilen Endgeräte geschickt wurde kam es in seltenen Fällen dazu, dass die Push-Benachrichtigung mit mehreren Minuten zeitlicher Verzögerung auf den mobilen Endgeräten ankam, obwohl diese eine dauerhafte Netzwerkverbindung zum Internet besaßen. Der Grund für die Verzögerung lag vermutlich am „Google Cloud Messaging“-Server, der zu dem Zeitpunkt womöglich ausgelastet war und das Versenden der Push-Benachrichtigungen erst zu einem späteren Zeitpunkt vornehmen konnte, bei dem die Auslastung geringer war.

Beim erneuten Überprüfen der PHP-Dateien konnten keine weiteren Fehler entdeckt werden. So kann abschließend gesagt werden, dass alle Tests für den Webserver erfolgreich waren.

8 Fazit und Erweiterung

Das Ziel der Bachelorarbeit ist es, die schon vorhandene Hochschul-App HSMWmobil für Android um die Möglichkeit der Push-Benachrichtigung über den Dienst „Google Cloud Messaging“ zu erweitern. Dieses Ziel ist mit dem Abschluss der Bachelorarbeit erreicht worden.

Nun besteht die Möglichkeit, den Benutzer der Hochschul-App HSMWmobil über Push-Benachrichtigungen zu informieren. Dazu steht eine einfache Benutzeroberfläche zum Versenden von Push-Benachrichtigungen auf dem Webserver bereit. Die schon vorhandenen vordefinierten Push-Benachrichtigungen informieren den Benutzer der Hochschul-App HSMWmobil über neu erschienene Artikel und Ausgaben verschiedener Informationskanäle. Zu den Informationskanälen zählen zurzeit die Hochschulzeitung „Die Novum“, HSMW Blog und HSMW News sowie medien-mittweida.de. Weitere Informationskanäle, die auch die Möglichkeit der Push-Benachrichtigung nutzen wollen, können schnell und einfach ergänzt werden.

Weitere Anwendungsfälle, für den die Push-Benachrichtigung innerhalb der Hochschul-App HSMWmobil genutzt werden könnte, wären in einer späteren Erweiterung der Applikation möglich. Beispielsweise könnte dem Studenten bei einer Stundenplanänderung seines eigenen Stundenplanes eine Push-Benachrichtigung mit der Änderung zugesandt werden. Dazu müsste allerdings eine Möglichkeit existieren, dem Webserver eine solche Stundenplanänderung bei Bekanntgabe sofort mitzuteilen. Außerdem wäre es denkbar, dem Studenten eine Push-Benachrichtigung zukommen zu lassen, wenn eine neue Note in seiner Notenanzeige zur Verfügung steht.

Ein weiterer Anwendungsfall wäre der Erhalt einer Push-Benachrichtigung bei einem Notfallalarm. Dieser Notfallalarm würde bei kritischen Situationen innerhalb des Hochschulcampus ausgelöst werden, wie beispielsweise einem Brand in einem der Hochschulgebäude. Somit wäre es möglich, auf schnelle Weise viele Leute auf dem Hochschulcampus zu informieren, da viele der Studenten und Mitarbeiter ein Android-Smartphone mit der Hochschul-App HSMWmobil besitzen.

Durch die Bachelorarbeit hat der Autor einen weiteren Einblick in die Android- und PHP-Programmierung erhalten. Was eine Verbesserung und Erweiterung seiner Programmierkenntnisse zur Folge hat.

Glossar

Activity	Ansicht innerhalb einer App, welche den ganzen Bildschirm ausfüllt
Android Virtual Device	Ist ein Emulator, welcher ein virtuelles mobiles Endgerät erstellt, bei dem spezielle Hardware- und Software- Eigenschaften eingestellt werden können.
Application Programming Interface	Programmierschnittstelle
AsyncTask	Helferklasse, die einen Thread neben dem UI Thread ausführt, ohne ein komplettes Programmiergerüst eines normalen Threads
Client for URLs	Ein Kommandozeilen-Programm zum Austausch von Dateien mit Hilfe eines URL-Syntaxes (Stenberg, 2013)
Document Type Definition	Ist eine Dokumententypdeklaration, welche die Syntax einer bestimmten Metasprache (Standard Generalized Markup Language) deklariert (W3C, 2012 a)
Document Object Model	Ist eine Plattform- und Programmiersprachenunabhängige Schnittstelle, welche es Programmen und Skripten erlaubt, Zugriff auf ein Dokument, zum Beispiel ein XML- oder HTML-Dokument, zu erlangen. Dort kann der Inhalt, die Struktur sowie der Style eines Dokumentes dynamisch verändert werden. (W3C, 2005)
Fragment	Ist eine unabhängige Komponente innerhalb von Activitys um mehrere Benutzeroberflächen in einer Activity zu nutzen. Diese können mehrmals in verschiedenen Activitys genutzt werden und besitzen einen eigenen Lebenszyklus. (Vogel, Vogella, 2013 c)
High Speed Downlink Packet Access	Weiterentwicklung des Mobilfunkstandards UMTS zur Steigerung der Datenrate und des Datendurchsatzes im Mobilfunk (Eichin & Söder, 2011)

Intent	Sind asynchrone Nachrichten, die es Komponenten von Applikationen erlauben, die Funktionalität von anderen Android Komponenten anzufordern und mit ihnen zu interagieren. (Vogel, Vogella, 2013 a)
JavaScript Object Notation	Kompaktes Datenformat zum Datenaustausch, welches für Menschen und Maschinen einfach zu lesen und schreiben ist (JSON, 2013)
LogCat	Android-Logging-System
Notification Area	Leiste am oberen Bildschirmrand, welche systemrelevante Informationen, wie zum Beispiel Netzwerkverbindungen und Benachrichtigungen anzeigt
Notification Drawer	Durch eine Wischgeste aufklappbare Leiste im Bereich der Notification Area, welche eine detaillierte Anzeige von Systemrelevanten Informationen und Benachrichtigungen anzeigt
Secure Sockets Layer	Veraltetes Protokoll für die Authentisierung und Verschlüsselung von Verbindungen im Internet zur sicheren Datenübertragung. Weiterführung als Transport Layer Security (TLS)-Protokoll
Software Development Kit	Werkzeuge zur Erstellung von Software
Thread	Neben dem Hauptprozess im Hintergrund laufender, zeitlich unabhängiger Prozess
User Interface (Benutzeroberfläche) Thread	Hauptprozess
UTF-8	Ist die Byte-orientierte und am weitesten verbreitete Form der Unicode-Zeichen, in der jegliches Zeichen vorhanden ist (Unicode Inc., 2013)

World Wide Web Consortium	Konsortium, welches sich zur Aufgabe gemacht hat, technische Web-Standardisierungen festzulegen (W3C, 2012 b)

Literatur

Eichin, D.-I. K., & Söder, P. D.-I. (26. Oktober 2011). *Lerntutorial für Nachrichtentechnik.* Abgerufen am 1. September 2013 von http://www.lntwww.de/downloads/Beispiele%20von%20Nachrichtensystemen/Theorie/Kapitel4/Bei_Kap4.4.pdf

Google Inc. (3. Juni 2013 c). *Android Developers*. Abgerufen am 3. Juni 2013 von http://developer.android.com/google/gcm/client.html

Google Inc. (23. Juni 2013 a). *Android Developers*. Abgerufen am 23. Juni 2013 von http://developer.android.com/google/gcm/gcm.html

Google Inc. (21. Juni 2013 b). *Android Developers*. Abgerufen am 21. Juni 2013 von http://developer.android.com/google/gcm/gs.html

Google Inc. (23. Juli 2013 d). *Android Developers*. Abgerufen am 23. Juli 2013 von http://developer.android.com/reference/android/support/v4/app/NotificationCompat.Builder.html

Google Inc. (24. Juli 2013 e). *Android Developers*. Abgerufen am 24. Juli 2013 von http://developer.android.com/reference/android/app/PendingIntent.html

Google Inc. (14. Juli 2013 f). *Android Developers*. Abgerufen am 14. Juli 2013 von http://developer.android.com/google/gcm/adv.html

JSON. (16. August 2013). *JSON*. Abgerufen am 25. August 2013 von http://www.json.org/

Nava, E. (28. Juni 2013). *Stackoverflow*. Abgerufen am 18. Juli 2013 von http://stackoverflow.com/questions/11242743/gcm-with-php-google-cloud-messaging

Rémond, M. (17. Mai 2013). *Process One*. Abgerufen am 12. Juli 2013 von http://blog.process-one.net/google-cloud-messaging-update-boosted-by-xmpp/

Schäfer, S. (25. August 2013). *Schaeferblick Weblog*. Abgerufen am 19. September 2013 von http://schaeferblick.wordpress.com/2009/07/06/push-vs-pull-in-der-informationsverteilung-%E2%80%93-eine-definitionsfrage/

Stenberg, D. (06. Mai 2013). *cURL*. Abgerufen am 10. August 2013 von http://curl.haxx.se/docs/faq.html#What_is_cURL

Tamada, R. (14. Oktober 2012). *Android Hive*. Abgerufen am 25. Mai 2013 von http://www.androidhive.info/2012/10/android-push-notifications-using-google-cloud-messaging-gcm-php-and-mysql/

The PHP Group. (21. August 2013 a). *PHP*. Abgerufen am 21. August 2013 von http://php.net/manual/de/function.fopen.php

The PHP Group. (25. August 2013 b). *PHP*. Abgerufen am 25. August 2013 von http://php.net/manual/de/function.curl-setopt.php

Unicode Inc. (28. Februar 2013). *Unicode*. Abgerufen am 24. August 2013 von http://www.unicode.org/faq/utf_bom.html

Vogel, L. (16. August 2013 a). *Vogella*. Abgerufen am 26. August 2013 von http://www.vogella.com/articles/AndroidIntent/article.html

Vogel, L. (12. Juni 2013 b). *Vogella*. Abgerufen am 04. August 2013 von http://www.vogella.com/articles/AndroidNotifications/article.html#pendingintent

Vogel, L. (20. August 2013 c). *Vogella*. Abgerufen am 17. September 2013 von http://www.vogella.com/articles/AndroidFragments/article.html

W3C. (19. Januar 2005). *W3C*. Abgerufen am 11. August 2013 von http://www.w3.org/DOM/

W3C. (2012 a). *W3C*. Abgerufen am 29. August 2013 von http://validator.w3.org/docs/sgml.html

W3C. (2012 b). *W3C*. Abgerufen am 29. August 2013 von http://www.w3.org/Consortium/

Anlagen

Anlagen, Teil 1

Die vorliegende XML-Struktur entspricht Beispielhaft der XML-Datei ids.xml, welche sich auf dem Webserver befindet. Darin werden die Registration IDs der mobilen Endgeräte gespeichert.

```
<?xml version="1.0" encoding="utf-8"?>
<root>
      <device>
            <regid>APA91bHDfw_mvWiNRod4iDrNTaLNk5oysU5gm4hv</regid>
      </device>
      <device>
            <regid>APA91bH1e2izfdpaq9WuhlcoLwqAwIDxQQV_YWBw</regid>
      </device>
      <device>
            <regid>APA91bF34mEEE3l0ZNqyEwZbPg4cPBM7gquQjaIw</regid>
      </device>
      <device>
            <regid>APA91bG6sr-s91ALENcPs2J55KAHhTxcLMFmK0BK</regid>
      </device>
</root>
```

Anlagen, Teil 2

Inhalte der beigelegten CD:

- Projektdateien der HSMWmobil-App
- ActionBarSherlock-Bibliothek (für das HSMWmobil-Projekt benötigt)
- Google Play Services (für das HSMWmobil-Projekt benötigt)
- Dateien des Webservers

Eigenständigkeitserklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig und nur unter Verwendung der angegebenen Literatur und Hilfsmittel angefertigt habe.

Stellen, die wörtlich oder sinngemäß aus Quellen entnommen wurden, sind als solche kenntlich gemacht.

Diese Arbeit wurde in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegt.

Mittweida, den 23.09.2013

Christoph Singer