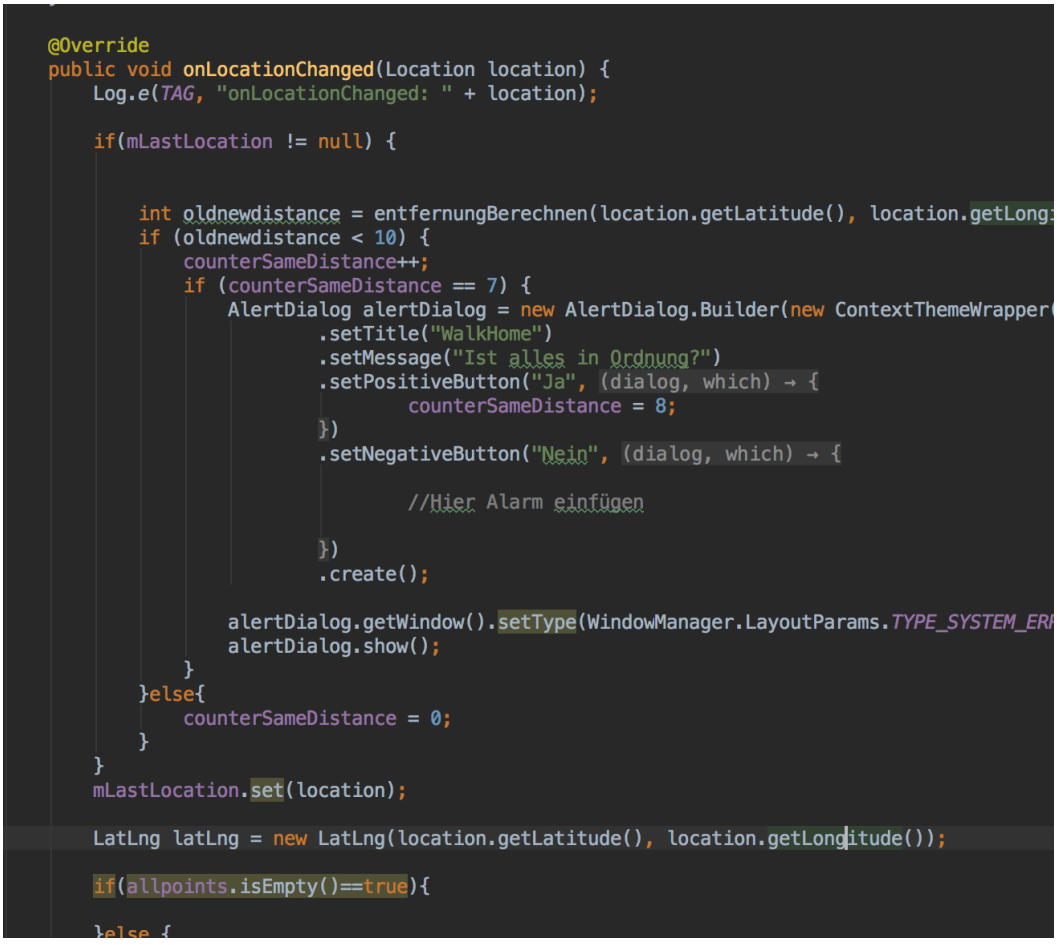
Technology
Arts Sciences
TH Köln

Implementations Dokumentation

Name: WalkHome
Team: 2
Gruppe: 23
Autoren: Yannik Huber, Tobias Szuminski
Betreuer: Gerhard Hartmann, Kristian Fischer, Ngoc-Anh Dang, Daniela Reschke
Modul: EIS Projekt Wintersemester 2016/17



```
@Override
public void onLocationChanged(Location location) {
    Log.e(TAG, "onLocationChanged: " + location);

    if(mLastLocation != null) {


        int oldnewdistance = entfernungBerechnen(location.getLatitude(), location.getLong
        if (oldnewdistance < 10) {
            counterSameDistance++;
            if (counterSameDistance == 7) {
                AlertDialog alertDialog = new AlertDialog.Builder(new ContextThemeWrapper
                        .setTitle("WalkHome")
                        .setMessage("Ist alles in Ordnung?")
                        .setPositiveButton("Ja", (dialog, which) → {
                                counterSameDistance = 8;
                        })
                        .setNegativeButton("Nein", (dialog, which) → {

                                //Hier Alarm einfügen

                        })
                        .create();

                alertDialog.getWindow().setType(WindowManager.LayoutParams.TYPE_SYSTEM_ERR
                alertDialog.show();
            }
        }else{
            counterSameDistance = 0;
        }
    }
    mLastLocation.set(location);

    LatLng latLng = new LatLng(location.getLatitude(), location.getLongitude());

    if(allpoints.isEmpty()==true){

    }else {
```

1 Implementations Dokumentation

1.1.1 Client - Server

Es wurde das Client Server Paradigma gewählt, da die Clients viele Informationen zur Verfügung stellen sollen, die dann von anderen Clients eingesehen werden müssen. Es ist keine Dienstgeber Architektur, da wir keinen allgemeinen Dienst anbieten der für andere Applikationen zugänglich sein soll, sondern einen Server der speziell für unsere App entwickelt wurde. Die App läuft auf Android, der Server mittels Node.js und Redis. Der Server funktioniert nach dem REST-Prinzip. So werden die Http-Verben für die Zwecke eingesetzt, für die sie eigentlich gedacht sind. Es wird get, put, post und delete verwendet. Es wurden eindeutige Ressourcen für die zugehörigen Funktionen vergeben. Und des weiteren werden Links zu bestimmten Ressourcen mitgesendet, um Hypermedia zu gewährleisten. In den meisten Fällen, „pullen“ die User die Daten vom Server um diese in der App zu aktualisieren. Um den Usern wichtige Datenänderungen auf dem Server mitzuteilen, wird an manchen Stellen Firebase Cloud Messaging verwendet. Falls sich der Status eines User ändert, da er zum Beispiel einen Alarm ausgelöst hat um seine Notfallkontakte zu kontaktieren, wird diese Information vom Server an die Kontakte „gepusht“.

1.1.1.2 Client (Android Applikation)

Die Android App, dient zur Verwaltung der Notfallkontakte, zur Navigation zu einem bestimmten Punkt, zur Überprüfung von anderen Begleitpersonen, zur Registrierung und zum Aufzeichnen Analysieren von bestimmten Sensoren des Smartphones. Des weiteren stellt sie eine Verbindung zum Server her, auf welchem die Daten gespeichert und teilweise analysiert werden.

In dieser Dokumentation wird nur auf die Funktionen der App eingegangen. Die Designentscheidungen, sind der MCI-Dokumentation zu entnehmen.

1.1.1.2.1 Registrierung

Die User registrieren sich mittels ihrer AndroidID, welche vom Betriebssystem vorgegeben ist. Des weiteren mittels eines Usernamen, den sich die Nutzer ausdenken können. Für den Usernamen gibt es keine Formellen Vorgaben, jeder Name ist allerdings einzigartig und kann über die Route

„/user/:USERNAME“ auf dem Server aufgerufen werden. Hier werden die Informationen über den Nutzer gespeichert. Die andere wichtige Route des Registrierungsprozesses ist „/userAID/:AID“ AID steht für AndroidID. Hier wird die AndroidID zusammen mit dem Nutzernamen gespeichert. Wenn die App dann gestartet wird, wird über die AndroidID der Nutzername abgefragt. Falls die AndroidID noch nicht auf dem Server gespeichert ist, kann der User sich registrieren. Die Daten werden über die Bibliothek HttpOk an der Server gesendet. Das jeweilige „post,get,delete oder put“ wir dann in einem asynchronen Prozess in der App ausgeführt, damit die App nicht angehalten werden muss. Die zugehörige Java-Datei ist „Registrieren.java“. Damit die App keine falschen Daten an den Server sendet, wird bei der Eingabe überprüft, ob der User alle Daten korrekt eingegeben hat. Falls dies nicht der Fall ist, wird eine Fehlermeldung ausgegeben. Im Server wird dann Sicherheitshalber auch nochmal überprüft ob alle Daten korrekt sind, dazu mehr in der Server Dokumentation. Die Antwort des Servers wird analysiert um zu überprüfen, ob der Username schon vergeben ist. Falls dies der Fall ist wird das dem User angezeigt.

1.1.1.2.2 Verwaltung der Notfallkontakte

Die Notfallkontakte werden über die Java-Datei „Contacts.java“ verwaltet. Es ist eine Suchfunktion vorhanden, die Asynchron „get“ auf „/user“ an den Server sendet. Der „ResponseBody“ wird dann von der App analysiert und verändert, sodass nur der in der Suche angefragte Name oder ein ähnlicher angezeigt wird. Angepasst auf die Anzahl der Objekte die damit übereinstimmen, werden dynamisch Button erzeugt und dem Layout hinzugefügt. Diese beinhalten den Namen des jeweiligen Users. Hier gibt es zwei Buttons, der eine um den Kontakt hinzuzufügen, den andern um die Kontaktdaten(Vorname, Nachname, PLZ) des Users einzusehen. Hier wird ein GET auf die Ressource des Users gemacht, welche im vorherigen GET auf alle Kontakte im Body mit gesendet wurde. Wenn der „Kontakt hinzufügen Button“ gedrückt wird, wird dieser zu den Notfallkontakten hinzugefügt. Darauf hin werden die vorhandenen Kontakte neu geladen, sodass der neu hinzugefügte Kontakt dort sofort angezeigt wird. Das Verfahren zum Anzeigen der vorhandenen Kontakte ist der Suche sehr ähnlich. Hier werden die Kontakte über ein „get“ auf „/user/:USERNAME/contact“ gepullt. Danach werden auch hier dynamisch Buttons erzeugt und mit den entsprechenden Kontaktnamen gefüllt. Wenn der hinzugefügte Kontakt einen allerdings noch nicht hinzugefügt hat, ist der Button nicht anklickbar und wird um den Text „(noch nicht bestätigt)“ erweitert. Die Abfrage darauf läuft ebenfalls über die Ressource „/user/:USERNAME/contact“ in dem Falle allerdings mit dem USERNAMEN des jeweiligen Kontaktes. Es wird überprüft ob der User einen in den Kontakten hat, falls dies der Fall ist, wird der Button aktiviert und kann betätigt werden. Wenn man nun auf den Button drückt, gelangt man zu einem neuen Layout und zu einer neuen „Activity“. Diese ist „ContactStatus.java“. Hier wird einem angezeigt, wo die letzte geortete Position des Kontaktes war und wie der aktuelle Status des Kontaktes ist. Außerdem kann man den Kontakt anrufen oder den Notruf betätigen, zum Beispiel wenn der Kontakt einen Alarm ausgelöst hat. Die Informationen zu Standort und Status werden über ein get auf die Ressource „/user/:USERNAME/alarm" gepullt. Falls sich der Status auf dem Server ändert, werden die Daten allerdings über Firebase Cloud Messaging an die Notfallkontakte gepusht. Die Karte auf welcher der Standort des Kontaktes gezeigt wird, wird über den Google Maps Service auf das Smartphone geladen.

1.1.1.2.2 Navigation zu einem bestimmten Punkt

Die Navigation findet über den Einstiegspunkt der Navigation statt. Die „Activity“ heißt „MapsActivity.java“. Hier wird auch eine Karte mittels Google Maps geladen. Der User kann ein gewünschtes Ziel eingeben und auf „Route suchen“ klicken. Die Route wird dann auf der Karte in einer Übersicht angezeigt. Danach kann der User auf „navigieren“ drücken und die Karte springt auf den akutellen Standpunkt. Die Navigation kann jederzeit abgebrochen werden. Bei „Route suche“ wird die Java Klasse „DirectionFinder“ aufgerufen. Hier wird eine Anfrage an die Google Maps API geschickt. Enthalten ist die URL mit dem angefragt Ziel und der Art der Fortbewegung. Diese ist auf „ZuFuß“ gesetzt. Da die User nicht mit dem Auto fahren sondern zu Fuß gehen.

Das JSON, welches als Antwort des Servers kommt, wird von der App „geparsed“ um die Route und die Dauer und Länge der Strecke herauszufiltern. „MapsActivity.java“ wartet mittels eines Listener Interface „DirectionFinderListener.java“ auf die Antwort von „DirectionFinder.java“.
Bei einer Antwort, wird in „MapsActivity.java“ die Methode „OnDirectionFinderSuccess“ aufgerufen. Hier wir die Route um weitere Punkte erweitert, um die Abweichung von der Route genauer bestimmten zu können, dies ist in der WBA-Dokumentation genauer beschrieben. Danach wird die Route der Karte hinzugefügt und gerendert. Die Routenpunkte werde des weiteren im Service „LocationService.java“ gespeichert, damit diese auch beim schließen der App vorhanden bleiben.

1.1.1.2.3 Aufzeichnen und Analysieren von Smartphone-Daten.

In dem Service „LocationService.java“ werden die GPS-Daten mittels des LocationManagers aufgezeichnet und analysiert. So wird überprüft ob der User von der vorher festgelegten Route abgekommen ist oder für eine bestimmte Zeit an einem Punkt bleibt. Die Berechnung findet in der Methode „onLocationchanged“ statt, die beim Aktualisieren der Standortdaten aufgerufen wird.
Die genaue Funktionsweise wird in der WBA-Dokmentation weiter beschrieben. Die User können einen Kopfhörer-modus in den Einstellungen auswählen. Wenn dieser aktiv ist, fungiert der einsteckte Kopfhörer wie eine art Reißleine. Wenn die Kopfhörer rausgezogen werden, wird der Nutzer gefragt ob alles in Ordnung ist, falls er nicht antwortet wird ein Alarm an die Notfallkontakte gesendet. Dies wird über den AudioManager geprüft.
Die GPS-Daten werden außerdem alle 30 Sekunden mit einem put auf „/user/:USERNAME/alarm“ an den Server gespusht. Diese Ressource steht für den aktuellen Status des Users (Alarm ausgelöst, Alarm zurückgenommen, unterwegs, angekommen), sowie die GPS Daten. Die App der Notfallkontakte aktualisiert den Status und den Standort des Users alle 30 Sekunden über ein get auf die Ressource, wenn die Aktivität ContactStatus aktiv ist. Falls sich der Status ändert wird dies durch ein push über Firebase Cloud Messaging an die Notfallkontakten, sowie die Personen im Umkreis(falls die Funktion aktiviert ist) gesendet. Um die Personen im Umkreis zu ermitteln, wird alle 5 Minuten der GPS-Standort des Users, über ein post auf die Ressource „/place“, an den Server gesendet. Dies passiert auch im Service „LocationService.java“. Auf die Ermittlung der Personen im Umkreis wird in der Dokumentation des Servers weiter eingegangen.

1.1.1.3 Server (Node.js + Redis)

Der Server funktioniert nach dem REST Prinzip. Die Meisten Routen des Servers dienen zur Datenspeicherung, der Nutzerdaten. Hierbei werden über die HTTP-Verben „post“, „put“, „delete“ und „get“ Ressourcen angelegt, aktualsisiert, gelöscht und abgefragt. Eine besondere Funktion des Servers ist die Einteilung der User in Ortsgruppen. Wenn das Smartphone eines Users ein post mit den GPS Daten auf die Ressource „/place“ an den Server sendet, wird dieser GPS-Standort analysiert. Er wird an die Geocoding API gesendet um herzauszufinden, in welcher Postleitzahl sich der User befindet. Falls diese Postleitzahl noch nicht auf dem Server angelegt wurde, wird diese unter „/place/:PLZ“ angelget. Danach wird der User dieser Ortsgruppe hinzugefügt. Wenn die App des Users erneut die Standortdaten an den Server schickt und sich die Postleitzahl ändert, wird der Nutzer aus der alten Ortsgruppe gelöscht und in die Ortsgruppe der neuen PLZ eingetragen.
An einigen Stellen sendet der Server in der Response eine URL, über die die App auf die bestimmte Ressource zugreifen kann. Dies entspricht dem 3 Level des Maturity Models von Richardson.

1.1.2 Abweichungen von MS2

1.1.2.1 Abwägungen

1.1.2.1.1 Login in die App

Die App verwendet keinen Standart Login mit Name und Passwort. Der Login wird über die Android ID die bei jedem Gerät einzigartig ist gesteuert. So werden neue Clients erkannt und zum Registrieren aufgefordert. Clients die bereits registriert sind werden automatisch eingeloggt.

1.1.2.2 Ergänzungen

1.1.2.2.1 Reißleine Alarmierenden wenn der Kopfhörer gezogen wird

Im Service „LocationService.java“ wird der Kopfhörer Anschluss überwacht. Diese Funktion muss zu erst in den Einstellungen aktiviert werden, dazu muss ein Kopfhörer angeschlossen sein. Die Aufgabe der Funktion ist es im Notfall zu erkennen, wenn der Anwender den Kopfhörer aus seinem Handy zieht, um einen Alarm abzusenden. Diese Funktion hat sich bei der Erarbeitung des menschzentrierten Gestaltungsprozesses als sinnvolle Ergänzung der Anwendung ergeben.

1.1.2.2.2 Speichern von Einstellungen

Um dem Anwender die Möglichkeit zu geben die Funktionsweise der App zu verändern gibt es einige Einstellungen in der APP. Diese ermöglichen z.B. die „Leute in der Umgebung zu benachrichtigen“ oder die Standortüberprüfung zu deaktivieren.

1.2 Installationsdokumentation

Die Installationsdokumentation dient zum Aufsetzten der gesamten Anwendung. Wichtig zu beachten ist das die Anwendung nur dann voll funktionsfähig ist wenn ein laufender Server zur Verfügung steht. Dieser läuft dauerhaft auf einem V-Server. Der Server kann auch lokal installiert werden. Hier ist zu beachten die Variable url/hostURL in den Dateien zu ändern.

1.2.1 Server

Der Server ist die gemeinsame Schnittstelle in dem Verteilten System welche mit allen Clients von jedem Ort aus angefragt werden kann. Um dies bei der Entwicklung zu gewährleisten und auch beim Testen mit mehreren Leuten alle den selben Stand haben zu haben, läuft der Server im Internet auf einem V-Server.

Der V-Server ist ein virtueller Server welcher im Internet zur Verfügung steht und von überall erreichbar ist. Dieser ist erreichbar unter www.walkhome.de / 5.199.129.74:81 .

Wenn der Server zu Testzwecken Lokal installiert wird, muss in der app.js die Variable
var url= "http://5.199.129.74:81";
auf
var url= "localhost:81";
geändert werden.

Wie der Server Lokal installiert wird wird im Punkt 1.1.1.1 beschreiben.

1.2.1.1 Nodeserver

Der Node Server stellt die Anwendung auf dem Port 81 zur Verfügung. Wenn der Server Lokal installiert wird kann über diesen Port der Server angefragt werden. Die Installation wird in den folgenden zwei Punkten beschreiben. Der Node Server muss auf dem selben Gerät installiert werden, wie die Datenbank und die APP.

1.2.1.1.1 Windows
- Download von https://nodejs.org/en/download/ für die vorliegende Windowsversion.
- Die Installation durchführen
- In der Konsole, in den Ordner Navigieren, in dem der Server abgelegt ist.
- npm install ausführen
- Der Server wird mit > node app.js gestartet.

1.2.1.1.2 MacOS
- Download von https://nodejs.org/en/download/
- Die Installation durchführen
- In der Konsole, in den Ordner Navigieren, in dem der Server abgelegt ist.
- npm install ausführen
- Der Server wird mit > node app.js gestartet.

1.2.1.2 Datenbank
Die Redis Datenbank läuft auf einem Standart Port der zur Anwendung nicht angepasst werden muss. Die Server muss auf dem selben Computer laufen wie der Node Server und die APP.

1.2.1.2.1 Windows
- Download von https://github.com/MSOpenTech/redis/releases
- Aktuelle Version herunterladen
- Archiv entdecken
- Redis-server.exe starten

1.2.1.2.2 MacOS
- Download von https://redis.io/download
- Archive entdecken
- In das Verzeichnis navigieren
- % make
- Im Terminal redis-server um den Server zu starten

1.2.2 App

1.2.2.1 Mindestvorrausetzungen

- Internetverbindung
- GPS Verbindung
- Android
- API Level 15 - Android Version 4.0.3

1.2.2.2 Installation der APP

Die ist noch nicht im Play Store veröffentlicht und muss somit manuell installiert werden. Dazu muss die WalkHome.apk auf dem Smartphone oder Emulator installiert werden. Im Anschluss kann die APP gestartet werden.

Wenn die APP zu Testzecken Lokal installiert wird muss die APP editiert werden.
In den Klassen Contact.java & LocationServer.java & MapsActivity & ContactStatus.java & Registrieren.java & UserSettings.java
muss die Variable:
Final String hostUrl = "http://5.199.129.74:81";
auf
Final String hostUrl = "localhost:81";
Geändert werden.

Zu beachten ist, dass beim ausführen auf dem Emulator ein Problem auftreten kann. Unser Emulator konnte nicht auf manche permissions von Android zugreifen wodurch, keine System Alert Meldung aufgerufen werden kann. Diese ist jedoch wichtig, da die Alert Meldung in jedem Zustand des Systems aufgerufen werden muss.

2 Fazit

2.1 Diskussion des Zielerreichungsgrad

Der Zielerreichungsgrad von Walkhome wird gemessen an den Anforderungen die vor der Entwicklung Festgelegt wurden. Wir haben alle funktionalen Anforderungen die wir durch die Anforderungsanalyse bestimmt haben vollständig umgesetzt. Wir haben uns während der Entwicklung der APP weitere Ziele gesetzt und die Anwendungslogik erweitert.

Um genau festzustellen ob die verteilte Anwendung das Ziel erreicht hat, wurde sie unter realen Bedingungen getestet. Die Test haben immer wieder gezeigt das die APP auch so funktioniert wie sie soll. Vor allem die wichtigen Funktionen, „Abweichen von der Route“ und „liegen/stehen bleiben“ wurden ausgiebig getestet.

2.2 Persönliches Fazit

EIS ist ein sehr umfangreiches Projekt, welches nicht zu unterschätzen ist.

Wir haben uns die Zeit gut eingeteilt. Erst haben wir uns stark an den Meilensteinen orientiert. Doch nach einiger Zeit haben wir bemerkt, dass wir noch etwas mehr machen müssen und haben uns die Aufgaben eingeteilt.

2.2.1 Ideenfindung

Wir haben bei der Ideenfindung etwas Zeit verloren. Zum einen hatten einige Ideen zu wenig Anwendungslogik oder hatten keine verteilte Anwendungslogik. Letztlich haben wir durch Brainstorming und Marktanalyse einen Problemraum gefunden. In diesem gibt es schon einige Lösungen, aber keine mit unserem Alleinstellungsmerkmal, der Routenabweichung. Damit sind einige funktionale Anforderungen entstanden die sich mit dem Alleinstellungsmerkmal zu einer guten Idee entwickelt haben.

2.2.2 Entwicklung

Bei der Entwicklung gab es einige Probleme. Zum einen beim Testen der App auf Android Emulatoren und zum anderen mit der Client Server Kommunikation.
Zum Ausführen und zum Testen der App haben wir zu erst den Emulator von Android Studio verwendet. Dieser war sehr langsam und hat einige Fehler bei verwendeten Klassen gemacht. Um dieses Problem zu umgehen sind wir umgestiegen auf den Genymotion Emulator. Dieser lief um schneller und macht keine Probleme mehr.
Zu dem mussten wir den Umgang lernen mit den unterschiedlichen Versionen von Android und Google Diensten bzw. den Google Play Services.

Wir hatten auch ein Hardware Android Device zum Testen. Um auf diesen Gerät und auf allen anderen Emulationen immer den selben Stand zu haben haben wir uns dafür entschieden den Server auf einem V-Server laufen zu lassen.

2.2.2.1 Gemeinsam Programmieren

Wir haben uns den Code aufgeteilt. Wir haben abwechselt gearbeitet und uns abgesprochen wann wer an der App arbeitet um Überschreibungen im Code zu verhindern. Da Yannik mit beim MCI Teil schon sehr viel am Layout und der Grundstruktur der APP gearbeitet hat hat Tobias in kleinen Android Testanwendungen entwickelt um z.B. die Verbindung zum Firebase Service zu testen.

Um einige Code Fragmente zusammen zufügen haben wir uns getroffen. Allgemein haben wir die Erfahrung gemacht das zusammen arbeiten sehr gut Funktioniert. Da man sich gegenseitig motiviert und dem jeweils anderen über die Schulter schauen kann. Zusammen kann man schneller Fehler finden oder sich bei kleinen Denkfehlern helfen.

3 Prozessassements

3.1 **Einhaltung des Projektplanes**

Wir haben den Projektplan verwendetet um uns an den Meilensteinen zu orientieren. Zu den Meilensteinen mussten bestimmte Artefakte abgegeben werden. Damit diese eingehalten werden haben wir geplant was gemacht werden muss und uns eigene Ziele gesetzt, damit zur Abgabe der Meilensteine alle Artefakte fertig sind.

Den genauen Ablauf der Entwicklung, Dokumentation und des MCI Teiles haben wir nicht im Projektplan geplant da wir nicht an festgelegten Tagen gearbeitet haben, sondern immer so wie wir gerade Zeit hatten. Um die Zeiten die wir für die einzelnen Artefakte benötigt haben zu verbuchen, haben wir nachträglich den Projektplan angepasst, um den Arbeitsverlauf zwischen den Meilensteinen und im gesamten Projekt aufzulisten.

Die Prozesse die wir im Projektplan vorher festgelegt haben (Meilensteine),konnten wir gut einhalten so das wir abschließend sagen können das wir die Zeit gut eingeteilt haben.

4 Quellen

https://developer.android.com/develop/index.html (Zuletzt aufgerufen am 09.01.2017)

http://stacktips.com/tutorials/android/get-device-id-example-in-android (Zuletzt aufgerufen am 09.01.2017)

https://nodejs.org/en/download/ (Zuletzt aufgerufen am 09.01.2017)

https://nodejs.org/en/download/ (Zuletzt aufgerufen am 09.01.2017)