

—Bachelorarbeit—

# Entwurf und Implementierung einer Anwendung für ambulante Betreuung bei Adipositas

## Design and Implementation of an Ambulant Care Application of Obesity

Dawid Janas

11. September 2013

Fachhochschule Kaiserslautern  
Fachbereich Informatik und Mikrosystemtechnik  
Studiengang Angewandte Informatik

Betreuer: Prof. Dr.-Ing. Uwe Tronnier (FH Kaiserslautern)  
Zweitkorrektor: Prof. Dr. Michael Bender (FH Kaiserslautern)



## Kurzfassung

Im Rahmen des Praxissemesters ist ein verteiltes Softwaresystem zum Überwachen der täglichen Kalorienzufuhr von Patienten entworfen und implementiert worden. Dieses System wird mit Hilfe einer Android-Anwendung und einem Webserver mit Datenbank und Anwendersoftware realisiert. Ziel der Software ist es, dem Benutzer eine Übersicht seiner täglichen Kalorienzufuhr zu verschaffen.

Das Projekt wurde in seinem Umfang innerhalb der Bachelorarbeit um weitere Messsensoren und Funktionalität erweitert. Außerdem ist die Webanwendung mit Ajax umgesetzt worden und die graphische Darstellung der Messdaten komfortabler gestaltet.

## Abstract

The task of the internship was to develop and implement a distributed softwaresystem for monitoring the daily caloric ingestion of patients. This system is implemented using an Android App and a webserver with database and applicationsoftware. The aim of the software is to give the user an abstract of his daily caloric ingestion.

Within the Bachelor's thesis the project was expanded in its scope with more measurement sensors and functionality. Besides, the webapplication is realised with Ajax and the graphical representation of measurements are designed more comfortable.



# Ehrenwörtliche Erklärung

Hiermit erkläre ich, **Dawid Janas**, geboren am **22.08.1978**, **Luban / Polen**, ehrenwörtlich,

- dass ich meine Bachelorarbeit/~~Masterarbeit~~ mit dem Titel

**Entwurf und Implementierung einer Anwendung für ambulante Betreuung bei Adipositas**

selbstständig und ohne fremde Hilfe angefertigt habe und keine anderen als in der Abhandlung angegebenen Hilfen benutzt habe;

- dass ich die Übernahme wörtlicher Zitate aus der Literatur sowie die Verwendung der Gedanken anderer Autoren an den entsprechenden Stellen innerhalb der Arbeit gekennzeichnet habe.

Ich bin mir bewusst, dass eine falsche Erklärung rechtliche Folgen haben kann.

[Ausstellungsort], [Ausstellungsdatum]

[ausgeschriebene Unterschrift]



# **Sperrvermerk**

Die vorliegende Bachelorarbeit/~~Masterarbeit~~ [Entwurf und Implementierung einer Anwendung für ambulante Betreuung bei Adipositas] enthält zum Teil Informationen, die nicht für die Öffentlichkeit bestimmt sind. Der Inhalt darf daher nur mit der ausdrücklichen schriftlichen Genehmigung des Verfassers und [Fachhochschule Kaiserslautern-University of Applied Sciences, Campus Zweibrücken, Amerikastr. 1, D-66482 Zweibrücken] an Dritte weitergegeben werden.

Die Arbeit ist nur den Korrektoren sowie erforderlichenfalls den Mitgliedern des Prüfungsausschusses zugänglich zu machen.

[Ausstellungsort], [Ausstellungsdatum]

[Unterschriften]



# Inhaltsverzeichnis

<b>1 Einleitung</b>	<b>5</b>
1.1 Motivation . . . . .	5
1.2 Aufgabenstellung im Detail . . . . .	5
1.3 Aufbau der Arbeit . . . . .	6
<b>2 Grundlagen</b>	<b>9</b>
2.1 Bluetooth . . . . .	9
2.1.1 RFCOMM . . . . .	9
2.1.2 UUID . . . . .	9
2.2 Verwendete Hardware . . . . .	10
2.2.1 Android Smartphone . . . . .	10
2.2.2 KERN PCB 6000-1 Laborwaage . . . . .	11
2.2.3 Bodytel WeightTel . . . . .	13
2.2.4 Bodytel PressureTel . . . . .	14
2.3 EAN-Barcode . . . . .	15
2.4 REST . . . . .	16
2.5 Ajax-Architektur . . . . .	18
2.5.1 HTML und CSS . . . . .	20
2.5.2 DOM . . . . .	20
2.5.3 JavaScript . . . . .	23
2.5.4 XML und JSON . . . . .	25
<b>3 Sicherheit</b>	<b>27</b>
3.1 Verschlüsselung . . . . .	27
3.1.1 MD5 . . . . .	27
3.1.2 SHA . . . . .	28
3.1.3 Zwei-Wege-Verschlüsselung . . . . .	28
3.2 TLS/SSL . . . . .	29
3.3 Sicherheitskonzept . . . . .	30
3.3.1 Web-API . . . . .	30
3.3.2 Android Sicherheit und Verschlüsselung . . . . .	31

3.3.3	Client-Server Kommunikation . . . . .	32
3.3.4	Anwendungsserver und Datenbankserver . . . . .	32
<b>4</b>	<b>Android-App: ApoCo</b>	<b>33</b>
4.1	Modelierung . . . . .	33
4.2	Apoco Projektstruktur . . . . .	34
4.3	Architektur der ApoCo-Datenbank(SQLite) . . . . .	46
4.3.1	Motivation . . . . .	46
4.3.2	Architekturumsetzung . . . . .	46
4.3.3	ApoCo Datenbankdiagramm . . . . .	51
4.4	Bluetoothkommunikation, Messung und Persistierung . . . . .	52
4.4.1	Bluetoothkommunikation BodyTel . . . . .	54
4.4.2	Dekodieren einer Messung . . . . .	55
4.4.3	Anzeigen und Speichern der Messwerte . . . . .	56
4.5	Datensynchronisation mit REST-Schnittstelle . . . . .	58
4.5.1	Datensynchronisation . . . . .	58
4.5.2	REST-Schnittstelle . . . . .	60
4.6	BarcodeScanner . . . . .	62
4.7	ApoCo Use Cases . . . . .	64
4.7.1	Use Case- Diagramm . . . . .	64
4.7.2	Aktoren . . . . .	64
4.7.3	Use Case Kurzbeschreibungen . . . . .	64
4.8	ApoCo-GUI Gestaltung . . . . .	66
4.8.1	Bedienelemente . . . . .	67
4.8.2	Activity-Wechsel mit Animation . . . . .	70
4.8.3	Views und Struktur der gegenseitigen Aufrufe . . . . .	71
<b>5</b>	<b>Webanwendung: Controlling-Software</b>	<b>75</b>
5.1	Modellierung . . . . .	75
5.2	Anwendungslogik . . . . .	75
5.3	Klasse Calendar . . . . .	79
5.4	Kalendervisualisierung . . . . .	82
5.5	Use Cases der Webanwendung . . . . .	83
5.5.1	Use Case-Diagramm . . . . .	83
5.5.2	Aktoren . . . . .	84
5.5.3	Use Case Kurzbeschreibungen . . . . .	84
5.6	Fundament der Anwendung . . . . .	85
5.7	Webanwendung-GUI Gestaltung . . . . .	86

<b>6 Zusammenfassung / Ausblick</b>	<b>91</b>
<b>Literaturverzeichnis</b>	<b>95</b>
<b>Abbildungsverzeichnis</b>	<b>99</b>
<b>Listings</b>	<b>101</b>
<b>A Anhang</b>	<b>103</b>
A.1 Glossar . . . . .	103
A.2 CD-Inhalt . . . . .	103



# 1 Einleitung

## 1.1 Motivation

Adipositas gilt in Deutschland als eine chronische Gesundheitsstörung. Diese geht aus einer falschen Ernährung und Stoffwechselproblemen hervor. Dabei leiden die Patienten an starkem Übergewicht. Mangelnde Bewegung durch sitzende Tätigkeiten am Arbeitsplatz und eine passive Freizeit begünstigen zusätzlich die Entwicklung von Übergewicht. Bleibt Adipositas unbehandelt, so kann es zu schwerwiegenden Folgeerkrankungen führen wie etwa Bluthochdruck, Verkalkung der Herzkranzgefäße, Zuckerkrankheit, Krebs, aber auch psychische Leiden nach sich ziehen. Hilfe findet man in Kliniken, Beratungsstellen, Selbsthilfe- und Wohngruppen. In der heutigen Zeit besitzt fast jeder Bürger ein Smartphone. Android ist aktuell mit 79,3% Marktanteil[And13] das meist verbreitete Smartphone-Betriebssystem der Welt. Es bietet aufgrund seiner vielseitigen Konnektivität durch Technologien wie WLAN, 3G/4G und Bluetooth eine exzellente Grundlage für die Entwicklung von Softwaresystemen zur Unterstützung von Geschäftsprozessen. Einige Hersteller haben sich bereits darauf spezialisiert sogenannte *E-Health* Systeme anzubieten. Dabei soll mit Hilfe von elektronischen Geräten die Qualität zur medizinischen Versorgung im Gesundheitswesen gesteigert werden.

Das Ziel der Bachelorarbeit ist, die Entwicklung und Realisierung eines mehrschichtigen Softwaresystems zum Erfassen von Messdaten und Unterstützung von Betreuer und Patient.

## 1.2 Aufgabenstellung im Detail

### Praxisprojekt

In dem Praxisprojekt soll eine Android-Anwendung zum Monitoring der täglichen Kalorienzufuhr von Patienten entwickelt und implementiert werden. Dabei wird die Anwendung eine Handykamera als Barcodescanner nutzen, um Lebensmitteleigenschaften aus dem Internet von einer Datenbank abzufragen. Über die Differenzwägung und die Lebensmitteleigenschaf-

ten wird die entsprechende Kalorienzahl der entnommenen Menge in einem Tagesprotokoll festgehalten. Die Wiegedaten sollen über Bluetooth übertragen werden. Die aufgezeichneten Tagesprotokolle werden zwecks späterer Auswertemöglichkeit in eine Webserver Datenbank gesendet. Für die Auswertung soll eine Webanwendung entwickelt und implementiert werden, welche die Tagesprotokolle übersichtlich veranschaulicht.

## Bachelorarbeit

Während der Bachelorarbeit wird die in dem Praxisprojekt erstellte Arbeit erweitert und verbessert. Die Software soll Adipositaspatienten helfen durch eine Übersichtsfunktion den Diätplan einzuhalten. Dem behandelnden Arzt soll die Software ermöglichen das Ernährungsverhalten seiner Patienten zu überwachen und bei einer ambulanten Adipositas-Betreuung zu unterstützen.

Zusätzlich zur Ermittlung der täglichen Kalorienzufuhr wird nun auch das Körpergewicht und der Blutdruck inklusive Puls erfasst und die Tagesprotokolle dadurch erweitert. Für das Erfassen der Daten sollen eine Bluetooth-fähige Körperwaage und ein Blutdruckmessgerät genutzt werden. Die webseitige Anwendung wird mit der Ajax-Architektur umgesetzt. Die aufgezeichneten Daten sollen in einem Graphen, der über ein Kalender gesteuert wird, dargestellt werden. Der Arzt soll die Möglichkeit haben, die Auflösung der dargestellten Daten zwischen Tag, Woche und Monat umzuschalten und über ein Kalender-Widget navigieren zu können. Die Navigation ermöglicht es zum gewünschten Datum zu springen und das Wochenende kann an eine beliebige Stelle in der Darstellung gerückt werden. Alle Daten werden als Liniendiagramme oder Liste dynamisch erstellt und visualisiert. Somit wird auch eine detaillierte Einsicht in die Zusammensetzung der Mahlzeiten ermöglicht. Als weiterer wichtiger Aspekt soll ein Konzept von Sicherheitsmechanismen für die Endpunkt zu Endpunkt Verschlüsselung der Daten ausgearbeitet werden. Das betrifft die Datenübertragung vom Handy zum Server und vom Server zur Monitoring Software.

## 1.3 Aufbau der Arbeit

- Kapitel 1:

Das Kapitel 1 umfasst die Einleitung und gibt einen Überblick über die geleistete Arbeit und Motivation während der Zeit im Praxisprojekt und der Bachelorarbeit.

- Kapitel 2:

In diesem Kapitel werden die notwendigen Grundlagen für die verwendete Hardware, Technologien und Standards erläutert, die wichtig für die Durchführung des Projekts

waren. Es wird unter anderem die Umsetzung von Ajax in einer Webanwendung anhand von Beispielen erklärt, wie eine REST-Schnittstelle funktioniert oder ein EAN-Code aufgebaut ist.

- Kapitel 3:

Dieses Kapitel stellt Verschlüsselungsmechanismen vor, mit denen eine sichere Datenübertragung ermöglicht werden soll. Es werden Hashfunktionen sowie Zwei-Wege-Verschlüsselungen vorgestellt. Außerdem wird ein Konzept für eine sichere Web-API und eine verschlüsselte Kommunikation einer Android-Anwendung ausgearbeitet.

- Kapitel 4:

In diesem Kapitel werden Designentscheidungen bei der Modellierung der Android-Anwendung erläutert. Es werden die verwendeten Architekturen und die Modellierung der persistierenden Schicht verdeutlicht. Darüber hinaus werden Hintergrundprozesse der Anwendung anhand von Beispiel-Codes und Diagrammen erklärt. Am Ende wird auf die graphische Oberfläche näher eingegangen.

- Kapitel 5:

Das Kapitel 5 beschäftigt sich mit der Webanwendung. Wie schon bei der Android-Anwendung werden hier Designentscheidungen während der Softwareentwicklung besprochen. Es wird auf die Umsetzung der Darstellung von Informationen in einem Graph und damit zusammenhängende Steuerprozesse durch ein Kalender-Widget eingegangen.

- Kapitel 6:

Dieses Kapitel beinhaltet eine Zusammenfassung der erarbeiteten Ergebnisse und einen Ausblick auf mögliche Erweiterungen.



## 2 Grundlagen

Dieses Kapitel enthält eine Einführung in die für das Projekt eingesetzten Technologien und verwendete Hardware.

### 2.1 Bluetooth

Bluetooth ist eine funkbasierte Technologie für den Datenaustausch zwischen Geräten auf kurzer Distanz. Dieser Standard wurde von der *Bluetooth Special Interest Group* (SIG)[Blu13a, Blu13b] in den 90er Jahren entwickelt und ist von dem *Institute of Electrical and Electronics Engineers* (IEEE) als Standard IEEE 802.15[Blu02] anerkannt. Einsatzmöglichkeiten für Bluetooth sind zum Beispiel die Verbindung zwischen einem Headset und Mobiltelefon für handfreie Kommunikation oder Verbindung von Messgeräten, wie zum Beispiel einer Körperwaage und einem Smartphone für Protokollierung und Auswertung der Messdaten.

#### 2.1.1 RFCOMM

RFCOMM steht für Radio Frequency Communication und ist ein Transportprotokoll. Es bietet vergleichbare Ausfallsicherheit, wie das TCP Protokoll. Das Ziel des RFCOMM-Protokolls ist die Emulierung einer seriellen Schnittstelle, vergleichbar der RS-232-Schnittstelle. Dies soll den Herstellern die Anbindung ihrer Hardware, die nach diesem Standard arbeitet, erleichtern. [AS07, S. 10]

#### 2.1.2 UUID

Verbindet sich ein Client mit einem Server, dann sucht dieser nach einem Service, den seine Anwendungsschicht verwenden möchte. Um den entsprechenden Service zu erhalten, wird hier eine sogenannte *Universally Unique Identifier* (UUID) verwendet. Das ist ein

128-Bit-Schlüssel, bestehend aus hexadezimal kodierten Stellen, welcher dem Client und Server bekannt sein muss. Für die im Projekt entwickelte Software ist die UUID *00001101-0000-1000-8000-00805F9B34FB* von großer Bedeutung. Diese repräsentiert das im Projekt verwendete *Serial Port Profile* (SPP). Dies ist notwendig, da die im Projekt eingesetzten Geräte Bluetooth als serielle Schnittstelle mit diesem Profil nutzen. [AS07, S. 17]

## SPP

Das Serial Port Profile ist eine Spezifikation, welche eine serielle Datenübertragung über Bluetooth ermöglicht.

## 2.2 Verwendete Hardware

Für die Entwicklung der Software im Projekt wurden mehrere Geräte verwendet. Als Basis dient ein Android-Smartphone zum Protokollieren der Messdaten. Die Lebensmittel werden mittels einer Laborwaage gewogen und für das Erfassen von Körpergewicht und Blutdruck werden entsprechende Messsensoren verwendet. Alle Messgeräte sind Bluetooth-fähig.

### 2.2.1 Android Smartphone

Die Software wird für ein Android-Smartphone entwickelt. Android ist ein weit verbreitetes Betriebssystem auf Smartphones und Tablets. Ein Android-Smartphone bringt sehr viele Möglichkeiten für Softwareentwickler mit sich. Durch die Möglichkeiten der Gestaltung von nahezu intuitiv bedienbaren grafischen Oberflächen, profitieren die Benutzer. Aber auch die Fähigkeit der Anbindung von externen Geräten über Schnittstellen wie Bluetooth oder die Kommunikationsfähigkeit mit dem Internet über WLAN oder das Mobile Netzwerk, bieten den Softwareentwicklern die Möglichkeit eine verteilte und komplexe Software mit enormen Wachstumsmöglichkeiten zu entwickeln.

### 2.2.2 KERN PCB 6000-1 Laborwaage

Die PCB 6000-1 ist eine hochpräzise Waage der Firma KERN. Die Abbildung 2.1 veranschaulicht das Gerät. Durch ihre Genauigkeit ist sie besonders gut geeignet, um Lebensmittel in sehr kleinen aber auch größeren Mengen zu wiegen. Sie verfügt über eine RS-232-Schnittstelle, über die man das gewogene Gewicht an einen Drucker oder Computer übertragen kann [KER13, S. 18 - 19].

#### Technische Daten:

- Wägebereich [Max]: 6 kg
- Reproduzierbarkeit: 0.1 g
- Ablesbarkeit: 0.1 g
- Linearität: 0.3 g
- Kleinstes Teilgewicht [Zählen]: 0.2 g/Stückseite
- Typ: PCB 6000-1
- Abm. Wiegeplatte: 150 x 170 mm
- Spannungsversorgung: 230 V/AC oder optionaler Akkupack
- Abm.: [L x B x H] 245 x 163 x 79 mm
- Besonderheit: RS-232-Schnittstelle

#### RS-232

Die RS-232-Schnittstelle dient der seriellen Datenübertragung. Es gibt sie in verschiedenen Ausführungen. In der Abbildung 2.2 wird die RS-232-Schnittstelle auf der Rückseite der PCB 6000 Waage als D-Sub-Buchse mit Adapter auf D-Sub-Stecker gezeigt. In Verbindung mit dem Projekt wird diese Schnittstelle mit einem Bluetooth-Dongle genutzt, um die Wiegedaten ohne Kabel an ein Smartphone zu übertragen.



Abbildung 2.1: KERN PCB 6000 Laborwaage



Abbildung 2.2: KERN PCB 6000 Laborwaage, Rückseite mit RS-232-Schnittstelle

### **Bluetooth-Dongle**

Als Bluetooth-Dongle kommt das Gerät Pico Plug der Firma Sphinx Elektronik zum Einsatz. Der Bluetooth-Dongle wird durch die Abbildung 2.3 veranschaulicht. Er ermöglicht eine kabellose Kommunikation zwischen zwei Geräten und sendet dabei im 2,4 GHz ISM-Band. Des Weiteren unterstützt er das RFCOMM-Protokoll und SPP-Profil, welches in der Software, was später gezeigt wird, angewendet wird.

**Technische Daten PICO Plug (BT-Plug)[Pic13]:**

- Ausführung: Bluetooth-Adapter mit Microcontroller und RS-232-Schnittstelle (seriell/parallel)
- Protokolle: L2CAP, RFCOMM, SDP
- Profile: GAP, SPP, Dial-Up-Networking Profile, LAN Access Profile
- Schnittstelle I: (Seriell) RS-232, Stecker D-Sub 9pol, Übertragungsgeschwindigkeit bis zu 115 kBaud
- Schnittstelle II: (Parallel) Centronics 36pol (Stecker), Übertragungsgeschwindigkeit 400 KBit/s
- Reichweite: 10 m



Abbildung 2.3: Pico Plug Bluetooth-Dongle

### 2.2.3 Bodytel WeightTel

Um das Körpergewicht zu erfassen, wird die Körperwaage WeightTel [Wei13] der Firma Bodytel eingesetzt. Hierbei handelt es sich um eine normale elektronische Körperwaage ohne Zulassung als Medizinprodukt nach dem Medizinproduktgesetz. Die Abbildung 2.4 veranschaulicht die Waage. Dieses Gerät ist mit einer Bluetooth-Technologie ausgestattet und erlaubt die Messdaten an weitere Geräte, wie Smartphones oder stationäre Home-Gateways zu übertragen. Die gesendeten Daten werden in einem SMS-ähnlichen Format verpackt. Auf den Aufbau und die Implementierung der SMS wird zu einem späteren Zeitpunkt näher eingegangen.



Abbildung 2.4: Bodytel WeightTel Körperwaage

#### Technische Daten WeightTel:[Wei13]

- Messbereich: Bis zu 220 kg
- Abstufung: in 100 g Schritten
- Trittfäche: 405 x 325 mm

#### 2.2.4 Bodytel PressureTel

Für Blutdruck-Monitoring wird der PressureTel Sensor der Firma Bodytel verwendet.[Pre13] Dieses Produkt erfüllt die Medizinproduktrichtlinie 93/42/EEC [med13] und ist FDA<sup>1</sup> zugelassen. Die Richtlinie dient als Instrument zum Nachweis von Sicherheit und medizinisch-technischer Leistungsfähigkeit von Medizinprodukten. Das PressureTel ist in der Lage 99 Messwerte zu speichern und diese zu einem beliebigen Zeitpunkt über Bluetooth zu Dokumentationszwecken zu übertragen. Hierbei wird, ähnlich wie schon bei der Körperwaage, ein SMS-ähnliches Format für die Datenformatierung genutzt. Das Gerät PressureTel ist in

<sup>1</sup>Food and Drug Administration: ist eine behördliche Lebensmittelüberwachungs- und Arzneimittelzulassungsbehörde in den USA[FDA13].

der Abbildung 2.5 abgebildet.



Abbildung 2.5: Bodytel WeightTel Körperwaage

#### **Technische Daten:[Pre13]**

- Messbereich Blutdruck:

Systolisch: 70 bis 260 mmhg

Diastolisch: 30 bis 180 mmHg

Puls: 40 bis 240 Schläge pro Minute

- Druckgenauigkeit: +/- 3 mmHg
- Messdauer: 30 Sekunden
- Interface: Bluetooth
- Messverfahren: Oszillometrisch

## **2.3 EAN-Barcode**

Die *European Article Number/EAN13b, EAN13a/* (EAN) ist eine international unverwechselbare Produktbezeichnung. Sie ist international eindeutig und wird individuell je einem Produkt zugewiesen. Dies wird für jedes Unternehmen von der *GS1 Germany GmbH* übernommen. Man findet die EAN als Barcode im Handel auf allen Artikeln, wie zum Beispiel

Lebensmitteln. In der Regel besteht die EAN aus 13 Ziffern. Für kleine Artikel gibt es eine Ausnahme, um Platz zu sparen. Dort ist die EAN nur 8 Ziffern lang. Sie werden auf einer Verpackung als schwarze und weiße Striche dargestellt. Dabei steht Schwarz für eine 1 und Weiß für eine 0 und kodiert die EAN-Nummer binär. Der Aufbau der EAN ist wie folgt:

- Zahlen 1 bis 3 stehen für den Ländercode. Deutschland hat den Präfix 400 bis 440.
- Zahlen 4 bis 9 können von der GS1 Germany einem Unternehmen nach Antrag zugeordnet werden.
- Zahlen 8 bis 12 werden in Abhängigkeit von dem Unternehmenspräfix von dem Unternehmen selbst für seine Artikel vergeben. Dabei muss die Gesamtlänge der EAN immer 13 Ziffern ergeben.
- Zahl 13 ist eine Prüfziffer und dient zum Verifizieren der EAN-Nummer.

Für die Produktidentifikation wird die EAN als Zahl eingegeben oder bequem mit einem Barcodescanner erfasst. Nach Erfassen des Barcodes kann in einer Datenbank nach den Produkteigenschaften automatisch gesucht werden. Die Abbildung 2.6 veranschaulicht ein Beispiel für einen EAN-Barcode.



Abbildung 2.6: Beispiel für ein EAN Produkt Code

## 2.4 REST

*Representational State Transfer*[RES13] (REST) ist eine Architektur in der Softwareentwicklung von Webanwendungen. Dafür wird meist das zustandslose HTTP- / HTTPS-Protokoll als Transportprotokoll genutzt. Jede Anfrage an den Server ist in sich geschlossen und der Server merkt sich keinen Zustand. Dabei werden an eine bestimmte URL alle notwendigen Parameter übergeben und die Server Software generiert eine entsprechende Antwort. Hierfür wird zum Beispiel eine weitere Anfrage an einen Datenbankserver ausgeführt. Die Antwort aus der Datenbank kann dann in einem beliebigen Format verpackt

werden. Meistens nutzt man dafür eine XML- oder JSON-Struktur. Sie können aber auch als Text oder HTML-Dokument ausgegeben werden. Für die Umsetzung von REST kommt man mit den folgenden vier Methoden des HTTP-Protokolls aus: GET, POST, PUT und DELETE.

- GET: Erlaubt das Lesen einer Ressource unter Angabe von einer URI und Parametern.
- POST: Ermöglicht ein Update von Ressourcen. Die Funktionalität ähnelt der GET-Methode, allerdings können mit der POST-Methode theoretisch unbegrenzt viele Daten an einen Server übertragen werden.
- PUT: Dient zum Hochladen oder Erzeugen einer Ressource auf einen Server.
- DELETE: Ermöglicht das Löschen einer bestimmten Ressource vom Server.

Bekommt der Client eine Antwort vom Server, so kann er mit den erhaltenen Daten seine Arbeit verrichten und kümmert sich selbst um seinen momentanen Zustand. Eine Sitzungsverwaltung durch den Server ist somit nicht erforderlich.

In der Abbildung 2.7 wird eine mögliche REST-Anfrage demonstriert.

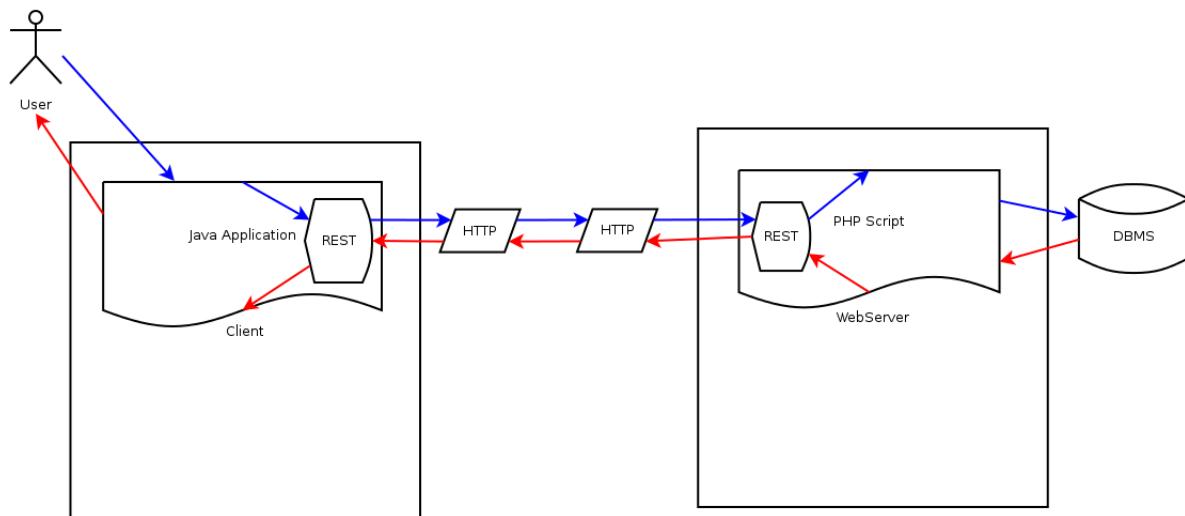


Abbildung 2.7: Beispiel einer möglichen REST-Anfrage

Der User startet eine Anfrage über seine Anwendung. Diese sammelt alle notwendigen Parameter und verpackt sie zum Beispiel in einen JSON-String. Der JSON-String wird als Parameter an eine bestimmte URL über HTTP gesendet. Zum Nachverfolgen ist die Senderrichtung in der Abbildung 2.7 blau gekennzeichnet. Auf dem Server läuft ein PHP-Skript,

welches sich um die Bearbeitung der Anfrage kümmert. Das Skript extrahiert aus dem HTTP-POST oder GET den JSON-String und aus diesem die erforderlichen Parameter und verarbeitet sie. Dazu kann zum Beispiel eine neue Anfrage an einen Datenbankserver gesendet werden. Die Antwort aus der Datenbank wird entweder nochmal im Skript verarbeitet oder als JSON-String zurück an den Client über HTTP gesendet. Die Rücksenderichtung ist hierbei rot gekennzeichnet. Die Software auf der Client-Seite liest die Antwort des Servers aus dem JSON-String und reagiert entsprechend darauf.

## 2.5 Ajax-Architektur

Ajax steht für *Asynchronous JavaScript and XML*[Chr07, Kapitel 18] und stammt ursprünglich von Microsoft. Es handelt sich dabei um eine Anwendungsart von bereits lange existierenden Webtechnologien wie: HTML und CSS, DOM, JavaScript, XML und JSON und XMLHttpRequest-Objekt. Ajax ermöglicht eine asynchrone Datenübertragung zwischen Client und Server. Dabei werden Daten vom Server angefordert und in die HTML-Seite geladen, ohne, dass die Seite selbst neu geladen wird. Der Aufbau einer solchen Kommunikation beginnt immer mit dem Erzeugen eines XMLHttpRequest-Objekts. Das Listing 2.1 veranschaulicht diesen Vorgang mit der Unterstützung für die gängigsten Browser.

```

1 function createXHRObject() {
2     var resObject = null;
3     try {
4         // IE 6 +
5         resObject = new ActiveXObject("Microsoft.XMLHTTP");
6     } catch (Error) {
7         try {
8             // IE 5
9             resObject = new ActiveXObject("MSXML2.XMLHTTP");
10        } catch (Error) {
11            try {
12                // MOZILA, Opera, Safari ...
13                resObject = new XMLHttpRequest();
14            } catch (Error) {
15                alert("Ajax is not supported on this browser");
16            }
17        }
18    }
19    return resObject;
20}

```

Listing 2.1: createXHRObject() Funktion in JavaScript

Als nächstes muss eine Verbindung zur Zielseite auf dem Server hergestellt werden. Dafür stellt das XHR-Objekt die Methode `open()` zur Verfügung. Sie bekommt eine URL und zwei Parameter übergeben. Die URL ist das Zieldokument, das aufgerufen wird. Der erste Parameter entscheidet, ob die Daten über POST oder GET übertragen werden und der zweite Parameter entscheidet, ob die Methode synchron oder asynchron ausgeführt wird. Synchron bedeutet, dass das ausführende Skript angehalten wird bis eine Antwort vom Server zurückkommt. Im Gegensatz dazu arbeitet bei der asynchronen Übertragung das Skript weiter, da die Anfrage parallel im Hintergrund ausgeführt wird. Um auf das Ergebnis einer asynchronen Anfrage reagieren zu können, wird eine Callback-Funktion verwendet. Diese wird als Referenz an die Variable `onreadystatechange` des XHR-Objekts übergeben. Das Absenden der Anfrage wird über die Methode `send()` ausgeführt. Wird die Anfrage als GET an den Server gesendet, so bindet man die Übergabeparameter an die URL als String. Dabei bekommt die Methode `send()` beim Aufruf den Wert `null` als Parameter übergeben. Wenn die Anfrage an den Server hingegen als POST gesendet wird, übergibt man die Werte als Parameter an die `send()` Methode. Die Callback-Funktion wird immer aufgerufen, wenn sich der Status des XHR-Objekts ändert. Deshalb muss die Callback-Funktion beim Aufruf immer zuerst den Status des XHR-Objekts überprüfen. Folgende Werte sind an dieser Stelle möglich:

- 0: nicht initialisiert
- 1: lädt
- 2: fertig geladen
- 3: wartet
- 4: fertig

Wird der Wert 4 ausgegeben, so kann der `responseText`, also die Rückgabe des XHR-Objekts, ausgewertet werden. Das Listing 2.2 veranschaulicht die Implementierung für diesen Vorgang.

```

1 function sendRequest() {
2   if (post) {
3     xhr.open("post", "zielurl.php", true);
4     xhr.onreadystatechange = callback;
5     xhr.setRequestHeader("Content-Type", "application/x-www-form-urlencoded");
6     xhr.send("a=1&b=2");
7   } else if (get) {
8     xhr.open("get", "zielurl.php?a=1&b=2", true);
9     xhr.onreadystatechange = callback;
10    xhr.send(null);
11  }

```

```

12 }
13
14 function callback() {
15   if (xhr.readyState==4) {
16     //do smth...
17   }
18 }
```

Listing 2.2: Verbindungsauflaufbau und Callback-Funktion bei Ajax

Das Anwenden von Ajax bietet viele interessante Möglichkeiten bei der Entwicklung von Websoftware. Da die Daten hier immer im Klartext gesendet werden, kann es sich dabei um eine normale Textdatei, HTML-Fragmente, XML oder JSON-Strings handeln. Somit ist es nicht nur möglich die Inhalte der sonst statischen Webseite dynamisch zu laden, sondern es besteht auch die Möglichkeit die Webseite an sich im eingeschränkten Umfang durch HTML-Fragmente neu zu gestalten. Außerdem wird die Webseite durch Ajax interaktiv und kann auf die Benutzereingaben reagieren. Ein bekanntes Beispiel ist das Google Suggest. Diese Anwendung ist die Autovervollständigung in der Eingabemaske von der Suchmaschine Google.

### 2.5.1 HTML und CSS

Die *Hypertext Markup Language*[HTM13] (HTML) ist eine Auszeichnungssprache und dient zur Strukturierung von Inhalten eines HTML-Dokuments. HTML-Dokumente dienen zur Darstellung von Text- und Bildinformationen in Webbrowsersn. Sie sind die Grundlage des *World Wide Web*. Das *Cascading Style Sheets*[CSS13] (CSS) ist eine deklarative Sprache und dient der Formatierung von zum Beispiel HTML-Dokumenten. Es ermöglicht nicht nur eine einfache Formatierung eines HTML-Dokuments, sondern kann auch beispielsweise bei der grafischen Oberflächengestaltung einer QT-Anwendung (C++) genutzt werden. In CSS3 sind einfache Animationen möglich und im Rahmen eines *Responsive Webdesign*[RWD13] kann man damit flexible Webseiten erstellen, welche sich an das Ausgabegerät, wie zum Beispiel einen Desktop-Monitor oder ein Smartphone-Display, automatisch anpassen.

### 2.5.2 DOM

Das *Document Object Model*[Chr11, Kapitel 3] (DOM) ist ein Konzept einer Schnittstelle für den Zugriff auf strukturierte Dokumente, wie HTML oder XML. Die Bestandteile eines HTML-Dokuments bilden dabei eine Art Baumstruktur. Beim Laden der Webseite wird

diese Struktur vom Webbrowser aufgebaut. Danach kann mittels JavaScript dynamisch auf die einzelnen Elemente zugegriffen werden. Die Ideen für dieses Konzept stammen ursprünglich von Netscape und wurden mittlerweile in alle anderen Browser übernommen. Das folgende Listing 2.3 zeigt den HTML-Code einer Demo-Webseite. In der darauf folgenden Abbildung 2.8 wird ein Ausschnitt aus dem Ergebnisdokument im Webbrowser präsentiert. Der dazugehörige DOM-Baum wird anschließend in Abbildung 2.9 veranschaulicht.

```

1 <html xmlns="http://www.w3.org/1999/xhtml" lang="de">
2   <head>
3     <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
4     <title>demo</title>
5   </head>
6
7   <body>
8     <div id="knoten1">
9       <h1 id="u1"> Meine Überschrift </h1>
10    </div>
11    <div id="knoten2"></div>
12      <p>ein Paragraph</p>
13      <p>noch ein Paragraph</p>
14    <div id="knoten3"></div>
15  </body>
16 </html>
```

Listing 2.3: Beispiel für ein einfaches HTML-Dokument

## Meine Überschrift

ein Paragraph

noch ein Paragraph

Abbildung 2.8: Das Ergebnis im Browser aus dem Listing 2.3

Mit dem DOM-Konzept besteht die Möglichkeit auf Elemente im Dokument direkt zuzugreifen. Darüber hinaus hat man sogar die Option wichtige Elemente des Browsers, wie die Adressleiste, über JavaScript zu manipulieren und kann sogar neue Elemente im DOM-Baum erzeugen. Einige wichtige Objekte aus dem DOM-Konzept sind:

- `document`: dieses Objekt ist das zentrale Objekt moderner Programmierung. Damit lassen sich neue Elemente im DOM-Baum erzeugen oder vorhandene manipulieren. Es erlaubt zum Beispiel dynamisch eine Tabelle zu erzeugen oder Bilder in die Webseite zu laden.

```

▼<html xmlns="http://www.w3.org/1999/xhtml" lang="de">
  ▼<head>
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8">
    <title>demo</title>
  </head>
  ▼<body>
    ▼<div id="knoten1">
      <h1 id="ul"> Meine Überschrift</h1>
    </div>
    <div id="knoten2"></div>
    <p>ein Paragraph</p>
    <p>noch ein Paragraph</p>
    <div id="knoten3"></div>
  </body>
</html>

```

Abbildung 2.9: Das Ergebnis des Listing 2.3 als DOM-Struktur im Webbrowser Chrome

- event: Steht für das Ereignis-Objekt. Es ist in Browsern der verschiedenen Hersteller unterschiedlich implementiert. Aus diesem Grund muss man mit einer Art Browser-Weiche arbeiten, wenn man damit Events auswerten möchte. Das können zum Beispiel Maus-Events oder Tastatureingaben sein.
- form: Darauf hat man Zugriff auf Formulare in einer Webseite.
- history: Das gibt den Zugriff auf die Historie des Browsers. Im Grunde hat man den Zugriff auf den Zurück-Button des Browsers.
- image: Das gibt den Zugriff auf alle Bilder einer Webseite.
- location: Mit *location* hat man den Zugriff auf die Addresszeile des Besucher-Browsers.

Das Listing 2.4 demonstriert in *JavaScript* eine mögliche Manipulation des DOM-Baums, der zuvor vorgestellten Demo Webseite.

```

1
2 var k3 = document.getElementById("knoten3");
3 var tab = document.createElement("table");
4
5 k3.appendChild(tab);
6
7 var row1 = document.createElement("tr");
8 var row2 = document.createElement("tr");
9
10 for (var i = 0; i<10; i++) {
11
12   var dat = document.createElement("td");
13   dat.textContent = i;
14
15   if (i%2==0)
16     row1.appendChild(dat);
17

```

```

18     else
19         row2.appendChild(dat);
20 }
21
22 tab.appendChild(row1);
23 tab.appendChild(row2);

```

Listing 2.4: Beispiel für dynamischen Zugriff auf den DOM-Baum

In der Abbildung 2.10 folgt nun das aus dem Listing 2.4 resultierende Ergebnisdokument. Im Browser erscheinen die Ziffern 0 bis 9, je nachdem, ob sie gerade oder ungerade sind, in der oberen oder unteren Reihe.

Im Listing 2.4 wurde in JavaScript auf das Ergebnisdokument über die Methode *document.getElementById()* auf den *knoten3* zugegriffen. So erhält man eine Referenz auf dieses Objekt im DOM. Mit der Methode *document.createElement()* erzeugt man weitere Objekte, welche im HTML-Code als *tags* erscheinen. Auf diese Weise wurde eine Tabelle mit zwei Reihen erzeugt. Mit der *for-Schleife* wurde die Tabelle befüllt. In der Abbildung 2.11 sieht man nun wie sich der DOM-Baum verändert hat. Der Zugriff über JavaScript muss hier nicht beim Laden der Webseite geschehen. Während der Benutzer die Webseite bereits betrachtet, kann eine solche Tabelle durch Interaktion zwischen Benutzer und Webseite dynamisch nachgeladen werden.

## Meine Überschrift

ein Paragraph

noch ein Paragraph

0 2 4 6 8  
1 3 5 7 9

Abbildung 2.10: Das Ergebnis im Browser aus dem Listing 2.4

### 2.5.3 JavaScript

JavaScript[Chr11] ist eine Skriptsprache. Das bedeutet, der Quellcode wird nicht kompiliert, wie das bei *C/C++* der Fall ist, sondern von einem Interpreter zur Laufzeit ausgeführt. Ein entsprechender JavaScript-Interpreter läuft im Webbrowser. Kommt der Browser beim Laden der Webseite an eine Skript-Stelle, wie im Listing 2.5, führt er das Skript entsprechend aus.

```

▼ <body>
  ► <div id="knoten1">..</div>
  <div id="knoten2"></div>
  <p>ein Paragraph</p>
  <p>noch ein Paragraph</p>
  ▼ <div id="knoten3">
    ▼ <table>
      ▼ <tr>
        <td>0</td>
        <td>2</td>
        <td>4</td>
        <td>6</td>
        <td>8</td>
      </tr>
      ▼ <tr>
        <td>1</td>
        <td>3</td>
        <td>5</td>
        <td>7</td>
        <td>9</td>
      </tr>
    </table>
  </div>
  <script type="text/javascript" src="demo.js"></script>
</body>

```

Abbildung 2.11: Der DOM-Baum nach Zugriff über JavaScript

```

1 <html xmlns="http://www.w3.org/1999/xhtml" lang="de">
2   <head>
3     <!-- hier ist das JavaScript eingebunden --&gt;
4     &lt;script type="text/javascript" src="lib/js/myscript.js"&gt;&lt;/script&gt;
5   &lt;/head&gt;
6   &lt;body&gt;
7     ...
8   &lt;/body&gt;
9 &lt;/html&gt;
</pre>

```

Listing 2.5: Eine JavaScript Datei eingebunden in ein HTML-Dokument

Wie bereits im Abschnitt der Erläuterung von DOM gezeigt wurde, wird JavaScript eingesetzt, um damit über das DOM-Konzept dynamisch auf die einzelnen Bestandteile einer Webseite zugreifen zu können und diese zu manipulieren. Das ist auch der Haupteinsatz von JavaScript. Des Weiteren wird JavaScript genutzt, um Eingaben in Webformularen zu überprüfen. Dies geschieht direkt im Browser, noch bevor das Dokument zum Server abgeschickt wird. Auch für Animationen auf einer Webseite oder für *Rich Internet Applications* gewinnt JavaScript immer mehr an Bedeutung. Rich Internet Applications stellen im Webbrower Funktionalität zur Verfügung, wie man sie von einer Desktopanwendung kennt. Zum Beispiel ein interaktiver Kalender oder eine Software für Textverarbeitung. Letztendlich ist JavaScript ausschlaggebend für die Ajax-Technologie. JavaScript ermöglicht das Senden und Empfangen von Daten zwischen Client und Server und das Einbauen

dieser in die Webseite. Dabei muss die Webseite nicht neu geladen werden. Die Sprache ist objektbasiert und dynamisch typisiert. Das bedeutet, dass hier die Datentypen nicht streng verwaltet werden, wie in Java oder C++ das der Fall ist, sondern sie ergeben sich zur Laufzeit, je nach Situation.

#### 2.5.4 XML und JSON

Bei *Extensible Markup Language*[XML13a] (XML) handelt es sich um eine Auszeichnungssprache für eine strukturelle Darstellung von Daten im Klartext. XML dient dem Datenaustausch zwischen Computersystemen. Dabei soll es aber Plattform- und Implementierungsunabhängig sein. Neben dem Datenaustausch lassen sich mit XML auch Strukturen, wie der grafische Aufbau von Android *Activities* umsetzen. Das Listing 2.6 soll mit einem Beispiel für eine XML-Struktur den Aufbau verdeutlichen. Ein wohlgeformtes[XML13b] XML-Dokument beginnt mit einem Prolog. Dieser beinhaltet die XML-Deklaration, welche die Version, sowie den Zeichensatz angibt. Als nächstes folgt das Wurzelement, welches alle anderen Elemente umschließt. In diesem Beispiel heißt das Wurzelement *<messung>*. Jedes Element, wie auch die Wurzel, werden mit einem *tag* angekündigt und abgeschlossen. Innerhalb der Wurzel können dann beliebig viele Elemente verschachtelt werden.

```

1  <?xml version="1.0" encoding="UTF-8"?>
2  <messung>
3      <messart>Blutdruck</messart>
4      <werte>
5          <datum>2013-03-07 06:25</datum>
6          <systolic>125</systolic>
7          <diastolic>75</diastolic>
8          <pulse>58</pulse>
9      </werte>
10 </messung>

```

Listing 2.6: Beispiel für eine Datenstruktur im XML

Heute wird in Ajax kaum noch XML benutzt und wurde durch *JavaScript Object Notation*[JSO13] (JSON) ersetzt. Durch seine Struktur aus Schlüssel-Wert-Paaren, ist JSON ohne zusätzliche Mittel einfach zu lesen und ist, wie der Name schon sagt, eine Objekt-Notation aus JavaScript. In nahezu jeder Sprache gibt es einen JSON-Parser. JSON kennt in seiner Notation Objekte und Arrays. Objekte werden immer innerhalb von geschweiften Klammern und Arrays in eckigen Klammern notiert. Objekte und Array-Elemente besitzen Eigenschaften, welche immer eindeutig sein müssen. Eine Eigenschaft besteht aus einem Schlüssel als Zeichenkette und einem Wert, wobei beide durch einen Doppelpunkt getrennt

werden. Im Listing 2.7 wird die gleiche Struktur aus dem Listing 2.6 als JSON veranschaulicht. Es fällt auf, dass der Umfang der Notation als JSON-String, im Vergleich zu XML, geringer wird.

```
1 {
2     "messart": "Blutdruck",
3     "werte": {
4         "datum": "2013-03-07 06:25",
5         "systolic": "125",
6         "diastolic": "75",
7         "pulse": "58"
8     }
9 }
```

Listing 2.7: Beispiel aus Listing 2.6 umgesetzt in JSON

# 3 Sicherheit

Im folgenden Kapitel wird ein Konzept von Sicherheitsmechanismen vorgestellt, welches für eine sichere Datenübertragung zwischen der Anwendersoftware und Web- bzw. Datenbankserver sorgen soll. Eine Webanwendung ist eine Software, die beim Benutzer im Webbrowsert abläuft. Sie kommuniziert dabei über das Internet mit einem Server, von dem aus sie gestartet wird. Einige Anwendungen auf einem lokalen Rechner oder Smartphone erfordern für ihre Funktionalität ebenfalls eine Verbindung zum Internet und tauschen in vielen Fällen sensible Informationen aus. Diese Daten müssen vor unbefugtem Zugriff und Einsicht durch Dritte geschützt werden. Zum Schutz der Datenübertragung werden verschiedene Schutzmechanismen eingesetzt. [SG12, S. 499] [Jö10]

## 3.1 Verschlüsselung

Unter *Verschlüsselung* versteht man eine Klartextnachricht mittels Kryptographie in eine unleserliche Form zu bringen. Für diesen Vorgang gibt es Ein-Weg- und Zwei-Wege-Verschlüsselungen. Die Ein-Weg-Verschlüsselung chiffriert eine Nachricht nur in eine Richtung. Das bedeutet, diese kann nicht mehr entschlüsselt werden. Diese Eigenschaft wird genutzt, um zum Beispiel Passwörter nicht als Klartext, sondern als eine Prüfsumme über das Netzwerk zu übertragen. Auf diese Weise ist es für einen Angreifer beim Auffangen der Übertragung im Internet oder direkten Zugriff auf die Datenbank nicht möglich das Passwort in Klartext zu lesen. Diese Art der Verschlüsselung wird auch als Hashing-Verfahren bezeichnet. Bei dem Verfahren der Zwei-Wege-Verschlüsselung wird eine Nachricht verschlüsselt und beim Empfänger wieder entschlüsselt.

### 3.1.1 MD5

Ein aktuelles Hashing-Verfahren ist MD5. Das steht für *Message Digest Version 5* [SG12, S. 507]. Beim Verschlüsseln erzeugt MD5 aus einer Zeichenkette von beliebiger Länge einen

128-Bit-Hashwert und stellt diesen hexadezimal dar. In der Abbildung 3.1 wird ein möglicher Hashwert aus dem String "Dawid Janas" dargestellt.

```
:/home/      # echo -n Dawid Janas |md5sum| cut -f1 -d' '
66c522d56e4f5eff1472ec7ba61cebb0
```

Abbildung 3.1: MD5 Hash aus dem String "Dawid Janas"

### 3.1.2 SHA

Ein weiteres Ein-Weg-Verschlüsselungsverfahren ist SHA. SHA ist eine Alternative zu MD5 und steht für *Secure Hash Algorithm*. In der Regel wird aus der SHA-Familie das SHA1 Verfahren genutzt. Neben diesen existieren noch weitere Algorithmen wie SHA256 oder SHA512 und sollen in der Zukunft SHA1 als Standard ersetzen. SHA1 erzeugt einen 40-Bit langen Hashwert und ist im Gegensatz zu MD5 robuster gegen Brute-Force-Attacken. Aktuell sind MD5 und SHA1 zum Schutz ausreichend. Der Zeitaufwand, um deren Verschlüsselung zu brechen, ist heute noch zu groß, so dass sie mit ruhigem Gewissen genutzt werden können.

### 3.1.3 Zwei-Wege-Verschlüsselung

Bei der Entwicklung von Anwendungen, bei denen es notwendig ist kryptographische Mechanismen zu benutzen, gibt es zahlreiche Optionen. Die Entwicklungswerkzeuge für Android bieten von Haus aus Pakete wie *javax.crypto* oder *java.security*, welche Schnittstellen und Klassen mit Algorithmen für Verschlüsselung oder Zertifikat- und Signaturverifizierung bereitstellen, an.

Die Sprache PHP stellt für die Verschlüsselung das Modul *mcrypt* bereit[SG12, S. 510].

Dieses Modul stellt unter anderem folgende Verschlüsselungsalgorithmen zur Verfügung:

- Blowfish: ist ein Blockverschlüsselungsalgorithmus mit einer Schlüssellänge zwischen 32-Bit und 448-Bit.
- (Triple-)DES: symmetrischer Verschlüsselungsalgorithmus mit einer Schlüssellänge von 168-Bit.
- Rijndael-Algorithmen: auch genannt Advanced Encryption Standard (AES). Die Variante AES-192 und AES-256 sind in den USA für die Verschlüsselung von staatlichen Dokumenten zugelassen[Sic13a].

## 3.2 TLS/SSL

Zum Versenden von Daten über das Internet bietet das HTTP-Protokoll die zwei Methoden GET und POST zur Auswahl. Beide senden Daten im Klartext. Für neugierige Augen scheint die Methode POST sicherer zu sein als GET. Das liegt daran, dass GET die Formulardaten einfach an die URL in der Adressleiste anhängt. Bei POST sind die Daten nicht sichtbar, da sie im HTTP-Header versteckt werden. Mittels *Sniffing* lassen sich HTTP-Header leicht abfangen und Werkzeuge, wie Wireshark, machen den gesamten Datenverkehr für den Angreifer sichtbar. Um ungewolltes Mitlesen zu verhindern, ist es notwendig den Datenverkehr über eine sichere Leitung und verschlüsselt zu schicken. Für diesen Zweck setzt man für die Kommunikation SSL/TLS ein. SSL steht für *Secure Socket Layer* [SG12, S. 513][JK08, S. 766] und TLS für *Transport Layer Security*[Sic13c]. Dabei ist TLS eine modifizierte Version von SSL Version 3. Beide Protokolle werden heute von allen Browsern und Servern unterstützt. Sie sorgen für Vertraulichkeit, Datenintegrität und Endpunktauthentifizierung, während einer Verbindung zwischen Client und Server. Tägt man heute einen Einkauf im Internet und bezahlt mit Kreditkarte, so wird mit Sicherheit SSL eingesetzt. Man erkennt dies daran, dass die URL der Webseite nicht mit *http*, sondern mit *https* beginnt. SSL kann von jeder Anwendung genutzt werden, die über TCP kommuniziert. Über entsprechende Bibliotheken hat man Zugriff auf die SSL-Socket-Schnittstelle. Die Programmierung ist analog zur TCP-Anwendungsprogrammierschnittstelle (TCP-API). TLS/SSL nutzt ein hybrides Verschlüsselungsverfahren aus Ein- und Zwei-Wege-Verschlüsselungen. Beim Aufbau einer Verbindung stellt der Client eine TCP-Verbindung zum Server auf. Als nächstes überprüft der Client, ob der Server wirklich der ist, als wen er sich ausgibt. Dies geschieht mittels eines Zertifikats, welches den Eigentümer bestätigt. Ein solches Zertifikat bekommt man gegen Geld bei entsprechenden Zertifizierungsstellen. Dabei sollte nicht vergessen werden, dass selbsterstellte, nicht beglaubigte Zertifikate von der Android-Plattform verweigert werden[AM11, S. 356]. Beim Verbindungsauflauf sendet der Server sein Zertifikat an den Client. Gleichzeitig bekommt der Client auch einen öffentlichen Schlüssel vom Server, mit dem er den Rest der Kommunikation verschlüsselt. Der Client generiert nun ein so genanntes *Master Secret* und sendet es an den Server. Mit dem Master Secret berechnen nun Server und Client verschiedene Schlüssel für Chiffren und die Integritätsprüfung. Während der Kommunikation werden die Datenpakete mit Sequenznummern versehen, um sie vor Manipulation von Außen zu schützen.

### 3.3 Sicherheitskonzept

Ein wichtiger Punkt in der Bachelorarbeit ist das Sicherheitskonzept. Der Patient protokolliert Informationen über sein Essverhalten und seinen Gesundheitszustand. Sein Arzt unterliegt der ärztlichen Schweigepflicht. Demselben Grundsatz unterliegt die Software. So mit müssen die Tagesprotokolle des Patienten beim Übertragen und Persistieren vor Einsicht durch Dritte geschützt werden. Die gesamte Anwendung, sprich Android-Anwendung, Web-Anwendung, Anwendungsserver und Datenbankserver bilden eine 3-Tier-Architektur. Hier findet ein Datenfluss über unsichere Medien, wie das Internet und ein Intranet, statt. Um für eine sichere Behandlung der Daten zu sorgen, müssen aus diesem Grund an mehreren Stellen der Software unterschiedliche Sicherheitsmechanismen angewandt werden.

#### 3.3.1 Web-API

Der Zugriff auf die Funktionalität der Webanwendung erfolgt über eine Zugriffs-API. Der Client ruft nicht direkt eine PHP-URL mit einem Skript, welches die gewünschte Funktion erfüllt, auf, sondern übergibt einen Schlüssel als Parameter und sendet die Anfrage über GET oder POST an die Zugriffs-API. Dieser Schlüssel ist an die gewünschte Funktion auf der Serverseite gebunden. Zum Beispiel will der Client Informationen über alle Patientennamen erhalten. Für diesen Fall ist die Anfrage im Listing 3.1 demonstriert.

```

1 // pseudo code
2 parameter = "get_all_patientennamen" + "myUserName" + "myPassword" + "api_key";
3 patientenNamen = HTTP.post("web_api.php", "parameter");

```

Listing 3.1: Beispiel für einen Zugriff über eine Zugriffs-API

Die Zugriffs-API überprüft die Parameter. Der Parameter *api\_key* muss ein gültiger Schlüssel für die Web-API sein. Dieser Schlüssel bestätigt, dass der Anwender die API benutzen darf. Im Erfolgsfall erzeugt die Web-API ein Objekt, welches sich um die gewünschte Funktionalität kümmert und eine entsprechende Antwort für den Client generiert. Ein Beispiel liefert das Listing 3.2.

```

1 // pseudo code
2 WebApi {
3     api_key = geheim;
4     receiveMessage(parameter) {
5         checkApiAccess(parameter.api_key);
6         if ok
7             checkUser(parameter.username, parameter.password);

```

```

8     if ok
9         new RequestHandler().handleRequest(parameter);
10    return "Zugriff verweigert";
11 }
12
13
14 RequestHandler {
15     handleRequest(parameter) {
16         switch(parameter.fkt)
17         case get_all_patientennamen:
18             // erzeuge Objekt, welches sich um die Anfrage kümmert
19         default:
20             return "unzulässige Funktion";
21     }

```

Listing 3.2: Beispiel für Behandlung nach einem Aufruf der Web-API

Dieses Vorgehen kapselt die Funktionalität der Web-API und verhindert, dass jemand ohne gültigen Zugang die API benutzen kann. Im weiteren Schritt wird der API-Schlüssel mittels SHA oder MD5 Algorithmus kodiert und die gesamte Anfrage chiffriert übermittelt. Der Server muss die Parameter entschlüsseln, um die Anfrage und Benutzer zu identifizieren. Der Hash-Code des API-Schlüssels muss dem Hashwert entsprechen, welchen der Server bei der Authentifizierung berechnet. Dies verhindert, dass beim Absenden einer Anfrage ein Angreifer den API-Schlüssel mitlesen oder die API-Funktionalität erfahren kann. Für das Chiffrieren der Anfrage bietet sich eine starke Verschlüsselung, wie AES, an.

### 3.3.2 Android Sicherheit und Verschlüsselung

Seit Java 1.4 steht die *Java Cryptography Extension (JCE)*[AM11, S. 352] zur Verfügung. Diese ist auch im Android-SDK nutzbar. Mit dieser Erweiterung ist es möglich Objekte oder Texte zu verschlüsseln. Sie basiert zu großen Teilen auf Implementierungen von *Bouncy Castle*[Sic13b]. Bouncy Castle ist eine Sammlung von APIs für Kryptographie. Sie stehen für Java und C# zur Verfügung. Im Kapitel 3.2 wurde erwähnt, dass selbsterstellte Zertifikate für eine SSL-Verbindung unter Android nicht akzeptiert werden. Dies unterstützt Android nicht, da der vorhandene *Keystore* eine zentrale Verwaltung für alle Anwendungen auf dem Gerät für Zertifikate darstellt. Um dieses Problem zu umgehen, entwickelt man einen exklusiven Keystore für seine Anwendung und hat somit auch keine Sicherheitslücke, welche mit dem zentralen Keystore des Android-Systems entstehen würde.

Android verwendet *SQLite* als Datenbank. SQLite bietet keine Möglichkeit die Daten beim Speichern automatisch zu verschlüsseln[AM11, S. 352]. Die Datenbank im Android ist eine Datei und beim Verlust eines Geräts kann man leicht an die Daten rankommen. Die Daten müssen auf dem Android-Gerät verschlüsselt in der Datenbank gespeichert werden. Dafür

implementiert man selbst einen einfachen Mechanismus, der für das Chiffrieren beispielsweise AES nutzt. Die Daten werden nur zum Anzeigen oder für die Vorbereitung einer Übertragung an den Server wieder entschlüsselt[AM11, S. 368].

### 3.3.3 Client-Server Kommunikation

Die Kommunikation zwischen Client und Server wird stets über eine sichere Verbindung mittels SSL/TLS aufgebaut. Das gilt für den Client des Arztes im Intranet, wie auch den Android-Client des Patienten im Internet. SSL verschlüsselt die Patientendaten nochmal (doppelte Sicherheit) und sorgt für eine sichere Authentifizierung der Kommunikationspartner. Dies gilt für alle möglichen Clients, sprich Android-Anwendung zu Server und Webanwendung des Arztes zu Server.

### 3.3.4 Anwendungsserver und Datenbankserver

Die Anwendungs-Tier und Persistenz-Tier können auf verschiedene Server aufgeteilt werden. Dabei wird Kommunikation der einzelnen Server ausschließlich über SSL/TLS aufgebaut. Die sichere Verbindung soll mögliche Angriffspunkte im Intranet minimieren. Die Abbildung 3.2 veranschaulicht die gesamte verteilte Anwendung als Diagramm.

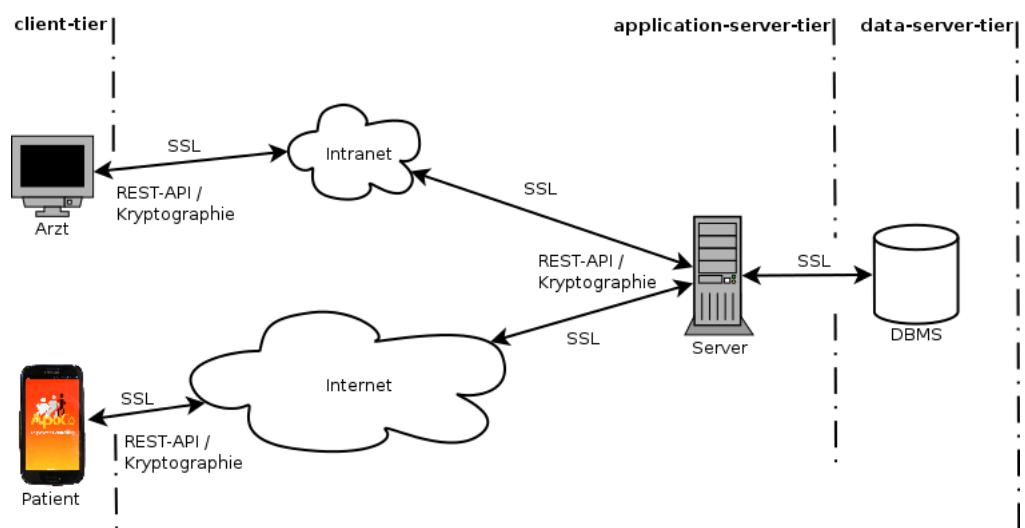


Abbildung 3.2: Sicherheitskonzept als Diagramm

# 4 Android-App: ApoCo

Dieses Kapitel beschäftigt sich mit der Android-Anwendung. Hier werden im Folgenden Designentscheidungen in der Architektur, GUI und Implementierung beschrieben. Außerdem werden wichtige Hintergrundprozesse wie Bluetooth und die REST-Schnittstelle erläutert.

## 4.1 Modellierung

Die Software ist stark modularisiert und domänenorientiert aufgebaut. Wichtige Aspekte, die zu dieser Entscheidung geführt haben, sind unter anderem, dass die Software später einfach und schnell erweiterbar, änderbar und partiell wiederverwendbar sein soll. Die Modularisierung erfolgt nach Funktionen und Teilaufgaben innerhalb einer Funktion. ApoCo wird mittels verschiedener Architekturen realisiert. Zunächst ist die grobe Struktur der Android-Anwendung als *Model-View-Controller (MVC)* modelliert. Die einzelnen Schichten Model, View und Controller der Android-Anwendung sind wie in der Abbildung 4.1 aufgeteilt.

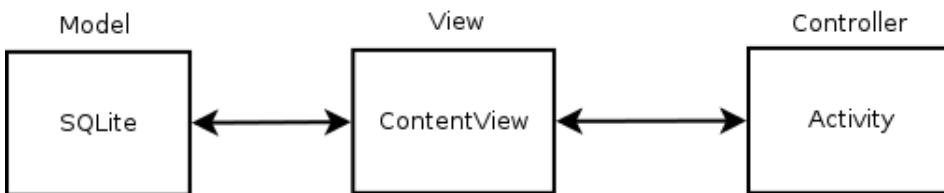


Abbildung 4.1: MVC-Architektur der ApoCo-Anwendung

- Model: Die Model-Schicht ist die interne Datenbank der ApoCo-Anwendung. Als Datenbank nutzt Android SQLite. Dabei ist die Datenbank als eine Datei umgesetzt. Diese ist auf dem Dateisystem in Ordnern der ApoCo-Anwendung hinterlegt. Die Anfragen und Statements werden hier genauso verwendet wie das zum Beispiel der Fall bei MySQL oder Oracle Datenbanken ist.
- View: Die View-Schicht wird durch die *Views* einer Android-Activity repräsentiert. Eine solche View wird in den meisten Fällen in einer XML-Datei beschrieben. Diese

Datei enthält die Struktur einer View, welche dem Benutzer anschließend auf dem Bildschirm präsentiert wird.

- Controller: Die Controller-Schicht wird durch die Anwendungslogik in den Activities umgesetzt. Hier wird die Interaktion des Patienten mit ApoCo abgefangen und die Ausgaben für den Bildschirm gesteuert.

Wird das gesamte Projekt betrachtet, handelt es sich dabei um eine Client-Server-Architektur. In der Abbildung 4.2 wird die Aufteilung der einzelnen Software-Komponenten der Client-Server-Architektur dargestellt.

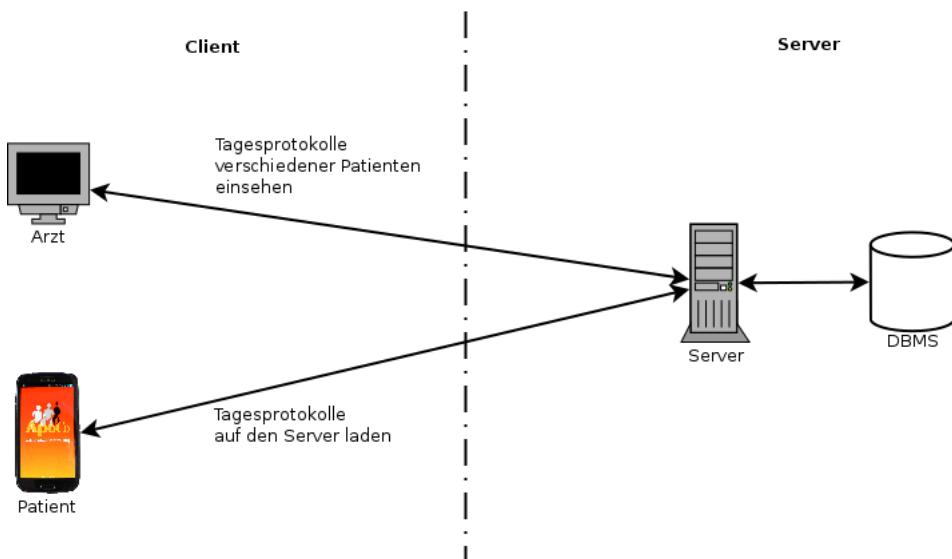


Abbildung 4.2: Client-Server-Architektur des gesamten Projekts

ApoCo ist der Teil der gesamten Software, welcher sich um das Aufzeichnen von Tagesprotokollen kümmert. Um die Daten auswerten zu können, benutzt der Arzt eine Webanwendung. Dabei werden die Tagesprotokolle von der Datenbank geladen und im Webbrower des Arztes angezeigt.

## 4.2 Apoco Projektstruktur

Die Abbildung 4.3 veranschaulicht die Paketstruktur der ApoCo-Anwendung. Sie spiegelt die Modularisierung der ApoCo-Anwendung wieder. Im Ordner `src` befindet sich der gesamte Quellcode der Anwendung. Dieser liegt aufgeteilt in den jeweiligen Modul-Paketen im Hauptpaket der Anwendung `com.janas.apoco`.

## Paket activity

Das Paket *activity* beinhaltet alle für die ApoCo-Anwendung notwendigen Activities. In diesem Paket befinden sich folgende Klassen:

- ActivityBloodpressure:

Hier steht die gesamte Anwendungslogik für das Protokollieren von Blutdruck. Die Activity veranschaulicht in einer scrollbaren Liste alle vorhandenen Messungen. Darüber hinaus werden die Daten beim Verlassen der Activity über das Internet mit einem Webserver synchronisiert.

- ActivityBodyweight:

Diese Activity behandelt das Protokollieren von Körpergewicht des Patienten und beinhaltet die dafür notwendige Anwendungslogik. Auch hier werden in einer Listenansicht alle getätigten Messungen angezeigt. Beim Beenden der Activity werden die Protokolle mit einem Webserver synchronisiert.

- ActivityDeviceList:

Diese Activity ist notwendig für das Koppeln über Bluetooth von Sensoren mit dem Smartphone. Es ist eine von zwei Möglichkeiten des Koppelungsvorgangs in der ApoCo-Anwendung. Hier verhält sich das Smartphone wie ein Client. Die Activity hat zwei Listen. In der einen Liste werden bereits bekannte Geräte aufgelistet. Nach einem erfolgreichen Suchvorgang werden die gefundenen Geräte in der zweiten Liste angezeigt. Man wählt aus einer der Listen das gewünschte Gerät zum Koppeln aus und sie verbinden sich untereinander.

- ActivityDevices:

Diese Activity bietet die Möglichkeit den Koppelungsvorgang zu starten. Man wählt die Art des Messgeräts aus, zum Beispiel Körperwaage, Blutdruckmesser oder Lebensmittelwaage. Anschließend wählt man die Koppelungsart aus. Zur Auswahl stehen die Möglichkeiten *als Server* und *als Client* zur Verfügung. Die Methode *pairing als Client* wird mittels der Klasse ActivityDeviceList ausgeführt. Die Methode *pairing als Server* erledigt die ActivityDevices selbst. Dafür startet sie einen Thread mit einem BluetoothServerSocket. Dieser hört auf Anfragen von externen Geräten zum Koppeln.

- ActivityFoodKcal:

Hier wird der Benutzer drauf aufmerksam gemacht, wieviele Kilokalorien er am aktuellen Tag bereits zu sich genommen hat. Zusätzlich wird die erlaubte Restmenge

berechnet und dem Benutzer angezeigt. Wird die erlaubte Tagesdosis überschritten, erscheint eine Warnung auf dem Display. In einer Liste werden vorhergehende Mahlzeiten aufgelistet. Beim Klicken auf eine Mahlzeit wird die Activity *ActivityMealenergyDetails* als *Dialog* eingeblendet und listet die einzelnen Positionen der Mahlzeit mit Informationen über Energie und Gewicht auf. Von hier aus gelangt der Benutzer weiter zur Protokollierung einer neuen Mahlzeit und zur Geräte-Koppelung.

- **ActivityMealenergy:**

Beim Start ist die Activity immer leer. Der Benutzer stößt von hier aus den Protokollierungsvorgang an. Über den Barcode-Button startet der Benutzer eine Activity für das Scannen von EAN-Codes. Anschließend wird eine interne und externe Datenbank nach der gescannten EAN-Nummer durchgesucht. Wird ein Eintrag gefunden, so wird der Benutzer zur Activity *ActivityMealContent* weitergeleitet, sonst wird eine Fehlmeldung ausgegeben, dass das Lebensmittel nicht identifiziert werden konnte.

- **ActivityMealenergyContent:**

In diese Activity gelangt man nur, wenn das Lebensmittel in der Datenbank identifiziert wurde. Hier werden Detailinformationen über das Lebensmittel angezeigt und über den Button *Waage verbinden?* kann eine Verbindung zur Lebensmittelwaage geöffnet werden.

Nach erfolgreicher Wägung errechnet die Activity die Energiemenge des gewogenen Lebensmittels, die Gesamtenergie aller Mahlzeiten und die noch zur Verfügung stehende Energiemenge für den aktuellen Tag.

- **ActivityMealenergyDetails:**

Diese Activity erscheint lediglich im Stil eines *Dialog*. Das bedeutet, dass die alte Activity im Hintergrund sichtbar bleibt und der modale Dialog tritt wie ein Pop-up in den Vordergrund. Sie besteht nur aus einer scrollbaren Liste und zeigt Details einer Mahlzeit auf.

- **ActivityLogin:**

Diese Activity dient dem Anmelden in der ApoCo-Anwendung. Alternativ kann ein neuer Benutzer zur Activity *ActivityRegister* wechseln, um sich zu registrieren.

- **ActivityRegister:**

Die Activity tritt als modaler Dialog auf und dient der Erfassung eines neuen Benutzers. Hier trägt der Benutzer seine Daten wie Vor- und Nachname, Email und Passwort ein. Die Activity führt eine Plausibilitätskontrolle durch. Anschließend sendet sie die Daten an einen Webserver, welcher überprüft, ob der Benutzername bereits vergeben ist. Sollte der Name zur Verfügung stehen, wird der Benutzer in die Datenbank

aufgenommen, andernfalls wird er abgelehnt. Die Activity reagiert auf die Serverantwort und akzeptiert entsprechend die Benutzerdaten oder fordert ihn erneut auf seine Eingabe zu überarbeiten.

- **ActivityServerOptions:**

Diese Activity erscheint als modaler Dialog und nimmt lediglich die Adresse und Port des Webservers entgegen.

- **ActivitySplashscreen:**

Das Splashscreen wird beim Start von ApoCo für wenige Sekunden angezeigt. Diese Activity präsentiert das Logo und den Schriftzug (*Adipositas Controlling*). Die Activity beendet sich nach einem Timerablauf automatisch und startet die Start-Activity von ApoCo.

- **ActivityStart:**

Das ist die Hauptactivity in der ApoCo-Anwendung. Hier wählt man die Art der Protokollierung aus. Das kann zum Beispiel eine Körpergewichtsmessung, Blutdruckmessung oder das Protokollieren der eigenen Nahrungsaufnahme sein. Darüber hinaus hat der Benutzer von hier aus den Zugriff auf die Funktion zum Koppeln von Messsensoren.

## Paket interfaces

Das Paket *activity* beinhaltet außerdem das Unterpaket *interfaces*. Hier befinden sich Java-Schnittstellen, welche das Zusammenspiel zwischen Activities und Objekten für Anwendungslogik unterstützen. Diese Schnittstellen geben, außer einigen Konstanten, auch Objektmethoden vor und werden für Polymorphie benutzt. Folgende Interfaces liegen im Paket vor:

- **AccessableCreatorIF:**

Dieses Interface dient zum Erzeugen von unterschiedlichen Threads zur Kommunikation zwischen Software und externen Sensoren. Für jeden Sensor steht ein angepasster Thread zur Kommunikation bereit. Über dieses Interface wird immer das richtige Objekt durch die jeweilige Activity erzeugt und weitergereicht.

- **ActivityExtrasCodesIF:**

In Android ist es möglich Objekte oder einzelne Werte zwischen den Activities auszutauschen. Dafür sind in diesem Interface Konstanten zum Zugriff auf die Daten bereitgestellt, die überall in der ApoCo-Anwendung zwischen den Activities ausgetauscht werden.

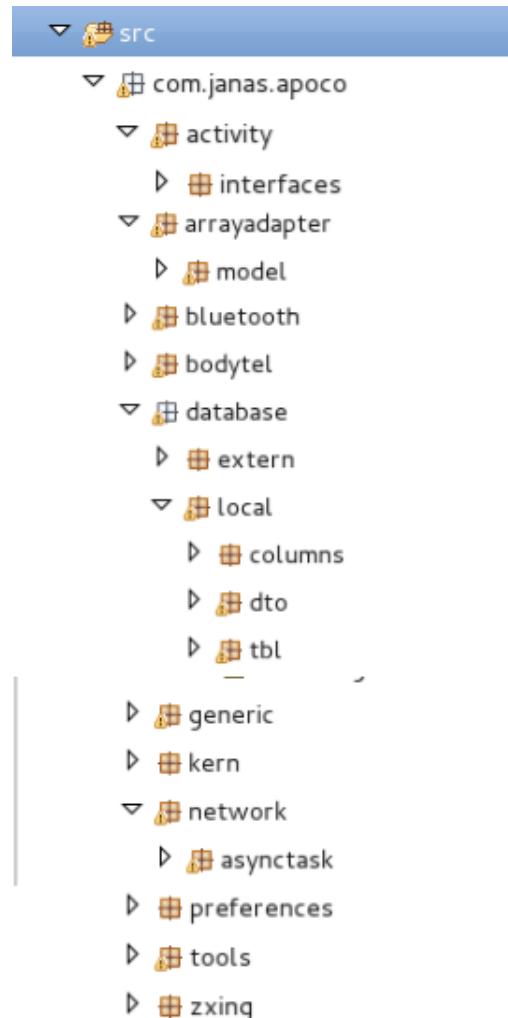


Abbildung 4.3: Projekt-Paketstruktur der ApoCo-Anwendung

- ActivityRequestCodesIF:

Soll aus einer Activity eine weitere gestartet werden und erwartet man von dieser ein Ergebnis, so wird bei diesem Vorgang ein *RequestCode* mitgegeben. Nach Ablauf einer Aufgabe kehrt eine Antwort zur ursprünglichen Activity mit dem Ergebnis und *RequestCode* zurück. Mit diesem RequestCode kann die Activity unterscheiden welche Antwort sie gerade bekommen hat.

- CloseableIF:

Eine Activity, die dieses Interface implementiert, darf zum Beispiel von einem Thread oder *Handler* geschlossen werden.

- WriteToPerformableIF:

Dieses Interface muss von einer Activity implementiert werden, wenn sie eine Nachricht

über Bluetooth versenden muss. Die Nachricht wird an ein Thread übergeben und dieser schreibt sie dann in den Streaminput eines *BluetoothSocket*.

## Paket arrayadapter

Im Paket *arrayadapter* befinden sich Implementierungen von performanten ArrayAdaptern. Werden Daten in einer ListView dargestellt, so kann man sie bequem aus einem Container, mittels einem ArrayAdapter in die ListView laden. Die meisten Listen in ApoCo benutzen keine Standarddarstellung von Daten in einer ListView, sondern eine individuell zugeschnittene Version. Für diesen Zweck benötigt man einen zugeschnittenen ArrayAdapter, dessen Funktionalität implementiert werden muss. Die eigenen ArrayAdapter sind in der Lage Daten zwischenspeichern, wenn durch die ListView hin und her gescrollt wird. Diese Daten müssen nicht mehr nachgeladen werden und das sorgt für bessere Performance.

Im Paket *arrayadapter* befindet sich ein Unterpaket. Hier werden *Model-Klassen* hinterlegt. Das sind Daten-Container, welche Informationen für genau eine Messung speichern. Zusätzlich bietet jedes Model eine statische Methode zum Konvertieren von Datenobjekten in Model-Objekte an. Dabei werden die Datenobjekte aus der Datenbank gelesen. Jede ListView findet hier einen eigenen ArrayAdapter und das dazugehörige Model.

## Paket bluetooth

In diesem Paket befinden sich Klassen und Interfaces, die sich um die Kommunikation über Bluetooth kümmern.

- AcceptThread:

Diese Klasse ist ein Thread, der einen BluetoothServerSocket startet und auf ankommende Anfragen zur Kommunikation wartet. Das ist der erste Schritt von Kommunikationsaufbau.

- AccessableIF:

Über dieses Interface kommunizieren Activities mit Threads, welche eine BluetoothSocket-Verbindung halten.

- BluetoothManager:

Der BluetoothManager soll die gesamte Bluetooth-Funktionalität in einer Klasse vereinigen. Mit dem BluetoothManager-Objekt wird die Kommunikation aufgebaut, kontrolliert und beendet.

- ConnectingThread: Dieser Thread ist der zweite Schritt zur Kommunikation über Bluetooth. Er öffnet einen Socket zur Gegenstelle und sendet Daten zwischen der Gegenstelle und der Activity.
- HandlerMessageIF:  
Dieses Interface beinhaltet alle Konstanten für einen Nachrichtenaustausch über einen Handler. Die Konstanten ermöglichen der Activity zu erkennen, wie sie mit der Nachricht umgehen soll.
- PairingThread:  
Das ist ein Thread, der keine Kommunikation aufbaut. Er wird nur ganz kurz für das Koppeln von Smartphone und einem Sensor genutzt.
- StandardUUIDsIF:  
Dieses Interface ist vorgesehen, um bekannte standardisierte UUIDs als Konstante bereitzustellen.
- StartableCancellableIF:  
Dieses Interface schreibt einem Thread vor, dass er von Außen gestartet und beendet werden kann.

## Paket bodytel

Dieses Paket beinhaltet alle Klassen und Interfaces, die notwendig sind, um eine Kommunikation mit Geräten der Firma Bodytel herzustellen. Im Augenblick werden die Sensoren WeightTel und PressureTel unterstützt.

- BloodpressureResult:  
Nach einer Blutdruckmessung wird aus dieser Klasse ein Objekt erzeugt und mit den Messwerten initialisiert. Das Objekt wird anschließend in ein Model- und DTO-Objekt konvertiert. Das Model-Objekt wird zum Anzeigen in einer ListView genutzt und das DTO-Objekt zum Speichern der Messdaten in der Datenbank.
- BodyTelUUIDsIF:  
Dieses Interface stellt Konstanten bereit, welche von der Firma BodyTel als UUIDs für die Bluetoothverbindung genutzt werden.
- BodyweightResult:  
Ein Objekt dieser Klasse wird genauso verwendet wie bereits bei der Blutdruckmessung beschrieben wurde, nur dass der Messwert mit der Körpergewichtsmessung initialisiert wird.

- PressureTelConnectedThread:  
Dieser Thread baut ein BluetoothSocket zwischen Smartphone und dem PressureTel-Blutdruckmesser auf.
- PressureTelCreator:  
Eine Activity nutzt diese Klasse, um dem BluetoothManager mitzuteilen, dass beim Bluetooth-Verbindungsauftbau ein Thread zur Kommunikation mit dem PressureTel gewünscht wird.
- PressureTelMeasurementDecoder:  
Diese Klasse dekodiert eine Nachricht von dem Blutdrucksensor. Dafür wird die Nachricht in einem *PressureTelSMS*-Objekt verpackt und alle Messungen, die in der Nachricht enthalten sind, als separate *BloodpressureResult*-Objekte in einer *List<BloodpressureResult>* zurückgegeben.
- PressureTelMessageProtocol:  
Geräte der Firma BodyTel kommunizieren über eine Art Konversationsprotokoll. Um die Messwerte auszulesen, muss dieses Protokoll erfüllt werden. In diesem Interface werden die notwendigen Konstanten für die Kommunikation mit dem Sensor *PressureTel* bereitgestellt.
- PressureTelMessageReader:  
Diese Klasse reagiert nach dem Protokoll auf die Anfragen bei der Kommunikation mit dem PressureTel-Blutdrucksensor. Sie nutzt die anderen PressureTel-Klassen, um eine Nachricht zu analysieren, auf sie zu antworten, Messwerte aus dieser zu dekodieren und sie in einer Liste an die Activity weiterzugeben.
- PresurreTelSMS:  
Diese Klasse ist eine Wraper-Klasse für eine SMS-Nachricht des PressureTel-Sensors. Ein Objekt von dieser Klasse wird mit einer SMS initialisiert und konvertiert diese in ein Format, dass in ApoCo verwendet werden kann.
- WeightTelConnectedThread:  
Mit diesem Thread wird ein BluetoothSocket zwischen Körperwaage und Smartphone geöffnet.
- WeightTelCreator:  
Mit Hilfe dieser Klasse erzeugt der BluetoothManager einen Thread, der an die Kommunikation mit der WeightTel-Körperwaage angepasst ist.
- WeightTelMeasurementDecoder:  
Diese Klasse wird genutzt, um eine Nachricht der Körperwaage zu dekodieren. Der

Vorgang entspricht dem Dekodierungsvorgang des bereits beschriebenen Dekodierungsvorgangs beim PressureTel-Sensor.

- WeightTelMessageProtocol:

Für die Kommunikation mit dem WeightTel-Sensor stehen hier die notwendigen Konstanten bereit.

- WeightTelMessageReader:

Wie beim PressureTel-Sensor, verwaltet diese Klasse den Dekodierungsvorgang einer Messung für den WeightTel-Sensor.

- WeightTelSMS:

Diese Klasse ist das zentrale Koppelungselement zwischen einer SMS-Nachricht des WeightTel-Sensors und einer Datenstruktur, die für ApoCo verwendbar ist.

## Paket Database

Die Klassen und Schnittstellen in diesem Paket ermöglichen einen Zugriff auf die interne und externe Datenbank. Im Unterpaket *extern* befinden sich zwei Interfaces. Auf dem Webserver befindet sich in einem bestimmten Verzeichnis die REST-Schnittstelle zum Zugriff auf den MySQL-Datenserver. Die Schnittstelle *ExternServerDIR* hat eine Konstante, in der das Verzeichnis abgespeichert ist. Für jede Funktionalität der REST-Schnittstelle gibt es eine entsprechende PHP-Datei. Im Interface *PHP\_URL\_IF* befinden sich Konstanten mit den Namen der PHP-Dateien. Zum Beispiel entspricht die Konstante *REGISTER\_USER* der URL *register\_user.php*. Ruft ApoCo diese URL mit den notwendigen Parametern auf, so ist es möglich einen neuen User im System anzulegen. Im Unterpaket *local* sind Interfaces und Klasse für die interne Datenbank auf dem Smartphone. Das Paket beinhaltet weitere Unterpakete: *column*, *dto* und *tbl*. Dies gehört zu einem objektorientierten Konzept zum Zugriff auf die Datenbank, welches im Kapitel 4.3 erläutert wird. Des Weiteren enthält das Paket *local* die Klasse DBManagerLocal und das Interface DBManagerPreferencesIF. Der DBManagerLocal dient jeder Activity als Zugriffsmanager auf die interne Datenbank. Das Interface DBManagerPreferencesIF dient zum Erstellen und der Konfiguration der Datenbank für die ApoCo-Anwendung.

## Paket generic

In diesem Paket befinden sich drei Klassen für die Durchführung einer Wägemessung mit der Lebensmittelwaage.

- GenericCreator:

Diese Klasse erzeugt einen Thread zur Kommunikation zwischen einer Activity und der KERN PCB-Laborwaage.

- KcalConnectedThread:

Dieser Thread baut einen BluetoothSocket mit der Waage auf und kommuniziert mit ihr.

- KcalResult:

Diese Klasse repräsentiert einen Messwert. Er setzt sich aus dem Gewicht und Eigenschaften des gewogenen Lebensmittels zusammen.

## Paket kern

In diesem Paket ist nur die Klasse *KERN\_PCB\_MessageBuilder* enthalten. Die Kern-Laborwaage sendet eine Messung als Datenblöcke, die in kurzen Zeitabständen nacheinander empfangen werden. Diese Datenblöcke werden in der Klasse gesammelt, analysiert und anschließend ein Messwert extrahiert. Nachdem dieser Vorgang abgeschlossen ist, kann eine Activity über die Methode *readMessage()* den Messwert auslesen.

## Paket network

Hier befinden sich alle Klassen und Schnittstellen, die für eine Kommunikation über WLAN oder das Mobile-Netz notwendig sind.

- JSON\_TAG\_IF:

Diese Schnittstelle stellt alle *tag*-Namen zur Verfügung, die bei einem Datenaustausch mit dem Webserver im JSON-Format genutzt werden.

- NetworkHandler:

Diese Klasse ist ein Handler, der in jeder Activity genutzt wird, welche mit dem Webserver kommunizieren muss. Vor einem Datenaustausch wird überprüft, ob eine WLAN- oder Mobile-Netz-Verbindung besteht und ob der Webserver antwortet. Je nach Ergebnis reagiert der Handler und benachrichtigt den Benutzer, wenn der Server nicht erreichbar ist. Eine Activity kann von diesem Handler geschlossen werden. Dazu muss sie die Schnittstelle *CloseableIF* implementieren und mit einem Flag bestätigen. Durch die Schnittstelle kann sie beendet werden, aber erst mit dem Flag wird dies auch abhängig von der Situation angefordert.

Im Unterpaket *asynctask* befinden sich Klassen, welche von der Klasse `AsyncTask<T,T,T>` abgeleitet werden. Es handelt sich dabei um Threads, die einen Teil der Arbeit im UI-Thread durchführen und den Hauptteil als Nebenläufigkeit. Mit diesen Threads wird je eine bestimmte Netzwerkfunktionalität der ApoCo-Anwendung erledigt.

## Paket preferences

Dieses Paket beinhaltet den *PreferencesManager* und eine Schnittstelle `APOCO_PREFERENCES`. Der *PreferencesManager* behandelt ApoCo-spezifische Parameter. Normalerweise können Werte in einer Datei oder Datenbank persistent gespeichert werden. Shared Preferences ist eine weitere Möglichkeit in Android, anwendungsbezogene Werte als eine Art Schlüssel-Wert-Paar zu speichern. Die Schnittstelle `APOCO_PREFERENCES` beinhaltet alle Schlüssel für den Zugriff auf die Shared Preferences.

## Paket tools

Das Paket *tools* beinhaltet Werkzeuge, die projektübergreifend nützlich sein können.

- **BloodpressureDiagnose:**  
Diese Klasse bietet Funktionen, die eine einfache Interpretation von Blutdruckwerten übernehmen. Man übergibt Blutdruckwerte an die Methode `performDiagnose()` und bekommt eine Aussage über die Messung.
- **BodyweightDiagnose:**  
Diese Klasse errechnet die Differenz zwischen Dem Zielkörpermengewicht und dem aktuellen Körpermengewicht.
- **ConnectivityTest:**  
Mit dieser Klasse ist es möglich durch den Aufruf der Methode `isAnyNetworkReachable()` festzustellen, ob das Smartphone über WLAN oder Mobile-Netz mit dem Internet verbunden ist.
- **DateTemplateIF:**  
Diese Schnittstelle enthält verschiedene Datumsmuster als Konstanten für eine Konvertierung eines Datums in ein gewünschtes Format.
- **FloatPrecision:**

Diese Klasse überprüft einen Fließkommawert, ob er in der Nähe der Zahl 0 liegt. Für den Test ist ein Epsilon für die Präzision notwendig.

- HexConverter:

Diese Klasse bietet mehrere Methoden an, um hexadezimal codierte Bytearrays in lesbare Strings umzuwandeln und umgekehrt.

- JSONParser:

Diese Klasse erleichtert das Senden und Empfangen von JSON-Strings zwischen einer Anwendung und einem Webserver. Dabei ist es möglich die HTTP-Methoden POST und GET zu benutzen. Die Serverantwort wird als JSON-Objekt zurückgegeben.

- PasswordCheck:

Diese Klasse überprüft, ob ein Passwort der gewünschten Vorgabe entspricht. Es wird die Mindestlänge und die wiederholte Eingabe des Passwortes geprüft.

- ResourcesTools:

Diese Klasse ist ein Wraper, um aus Android-Ressourcen eine String-Ressource zu lesen.

- TimeTools:

Diese Klasse konvertiert einen Timestamp vom Typ Long in einen String mit dem Format *dd-MM-yyyy HH:mm* und umgekehrt.

- Toasting:

Diese Klasse erleichtert Informationen über einen Toast anzuzeigen. Der hier ausgebene Toast entspricht nicht den Standardvorgaben, sondern ist individualisiert.

- URLBuilder:

Diese Klasse bietet die Methode *getURL()* an. Mit dieser Methode wird aus Parametern und einem formatierbaren String eine vollständige URL-Adresse zusammengefügt.

## Paket zxing

Dieses Paket enthält zwei Klassen. *IntentIntegrator* und *IntentResult*. Diese zwei Klassen sind notwendig, um eine externe Barcodescanner-Anwendung zu nutzen. Die Anwendung wird mit der Unterstützung dieser Klassen in die eigene Anwendung integriert und man erhält Zugriff auf den Barcode als String.

## 4.3 Architektur der ApoCo-Datenbank(SQLite)

Im Buch von Arno Becker und Marcus Pant[AM11, S.244], wird ein Architekturvorschlag für eine Datenbankzugriffsschicht für Android-Anwendungen gemacht. ApoCo folgt diesem Vorschlag, der hier erläutert wird.

### 4.3.1 Motivation

Beim Umgang mit einer Datenbank ist es oft erwünscht eine von der Anwendungsschicht getrennte Zugriffsschicht auf die Datenbank zu haben. Dafür kann man sich an dem Konzept der Programmierung mit Java-EE orientieren. Android ist für ein solches Konzept nicht ausgelegt und so ergeben sich folgende Nachteile:

- Die Anwendung wird ungewollt groß und die Performance sinkt.
- Da keine Frameworks, wie zum Beispiel Spring oder Hibernate existieren, sind Änderungen an der Datenbank sehr aufwendig.
- Es gibt keine Möglichkeit die Datenbankschicht von der Präsentationsschicht ohne spürbaren Leistungsabfall zur Laufzeit zu trennen [AM11, S.244].

Außerdem wurde festgestellt, dass eine reine Schichtentrennung unmöglich ist, wenn man mit einem Cursor-Objekt arbeiten möchte. Der Cursor ist wie ein Zeiger und ist darauf ausgelegt mit großen Datenmengen in einer Datenbank umgehen zu können.

### 4.3.2 Architekturnsetzung

Die Zugriffsschicht auf die Android-Datenbank besteht aus den folgenden Klassen:

- DBManagerLocal
- TabellennameColumns
- TabellennameDTO
- TabellennameTbl

Dabei ist zu beachten, dass für jede Tabelle eine *Columns*, *DTO* und *Tbl*-Klasse existiert. Die Bezeichnung dieser Klassen setzt sich aus dem Tabellennamen als Präfix und den so

eben genannten Klassenendungen zusammen.

Die Klasse *DBManagerLocal* ist eine Verwaltungsklasse, um die Datenbank in der Activity greifbar zu machen. Diese Verwaltungsklasse erweitert die Klasse SQLiteOpenHelper aus dem Android-SDK. Der SQLiteOpenHelper weiß wie eine Datenbank erzeugt und geändert wird. Dafür sind zwei Methoden vorgegeben: *onCreate()* und *onUpgrade()*. Mit diesen zwei Methoden wird das Datenbankschema aufgebaut und bei Änderungen gelöscht und neu aufgebaut. Um einen Backup-Mechanismus muss man sich selbst kümmern. Des Weiteren sind im *DBManagerLocal* Methoden implementiert, um mit der Datenbank Daten auszutauschen. Im Listing 4.1 wird eine reduzierte Version des *DBManagerLocal* implementiert und es wird eine Tabelle *user* angelegt. Die Klasse *DBManagerLocal* implementiert die Schnittstelle *DBManagerPreferencesIF*. In dieser Schnittstelle wird der Name der Datei für die Datenbank und die Versionsnummer hinterlegt. Die Versionsnummer ist hier sehr wichtig. Wird das Datenbankschema geändert, so muss die Versionsnummer inkrementiert werden. Nur so erkennt die Klasse SQLiteOpenHelper, dass die Datenbank neu aufgebaut werden muss.

```

1 public class DBManagerLocal extends SQLiteOpenHelper implements
2   DBManagerPreferencesIF {
3
4   public DBManagerLocal(Context context) {
5     super(context, DATENBANK_NAME, null, DATENBANK_VERSION);
6   }
7   public void onCreate(SQLiteDatabase db) {
8   }
9   public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
10 }

```

Listing 4.1: Beispiel für reduzierte Implementierung der *DBManagerLocal* Klasse

Die *TabellenameColumns*-Klassen sind als Schnittstellen und werden pro Tabelle angelegt. Sie besitzen Konstanten für jede Spaltenbezeichnung der jeweiligen Tabelle. Im Listing 4.2 wird ein Beispiel *Columns-Klasse* für die Tabelle *user* veranschaulicht.

```

1 public interface UserColumns {
2   static final String _ID      = "_id";
3   static final String VORNAME  = "vorname";
4   static final String NACHNAME = "nachname";
5   static final String EMAIL    = "email";
6   static final String PASSWORD = "password";
7 }

```

Listing 4.2: Die Schnittstelle *UserColumns*

Beim Zugriff auf die Tabelle *user* wird im SQL-Statement über die Konstanten der *Columns-Klasse* auf die einzelnen Tabellenspalten zugegriffen. In der Klasse *TabellennameTbl* werden Schmainformationen und SQL-Statements als Konstanten vom Typ String bereitgestellt. Die Klasse *DBManagerLocal* nutzt die Schmainformationen, um eine Tabelle zu erzeugen. Die SQL-Statements sind als Prepared Statements zu verstehen. Das bedeutet, dass sie nur ein SQL-Statement-Muster ohne Parameter repräsentieren. Für Parameter wird das Fragezeichen (?) als Platzhalter verwendet, das vor dem Zugriff auf die Datenbank durch einen echten Parameter ersetzt wird. Das Listing 4.3 veranschaulicht mit der *UserTbl* eine Tabellenschema-Klasse.

```

1 public final class UserTbl implements UserColumns {
2
3     //Tabellenname
4     public static final String TABLE_NAME = "user";
5
6     //Create-Statement zum Anlegen der Tabelle
7     public static final String SQL_CREATE =
8         "CREATE TABLE " + TABLE_NAME + "(" +
9             "_ID + " INTEGER PRIMARY KEY AUTOINCREMENT," +
10            VORNAME + " TEXT NOT NULL," +
11            NACHNAME + " TEXT NOT NULL," +
12            EMAIL + " TEXT NOT NULL," +
13            PASSWORD + " TEXT NOT NULL" + ")";
14
15    //Drop-Statement zum L\\"oschen der Tabelle
16    public static final String SQL_DROP =
17        "DROP TABLE IF EXISTS " + TABLE_NAME;
18
19    //Beispiel-Statement, welches alle Benutzer mit dem gleichen Nachnamen
20    //zurueckgibt.
21    //Der Nachname wird als Parameter uebergeben.
22    public static final String STMT_GET_USER =
23        "SELECT * FROM " + TABLE_NAME +
24        "WHERE " + NACHNAME + "=?";
```

Listing 4.3: Die Klasse *UserTbl*

Listing 4.4 zeigt ein Beispiel, um die Tabelle *user* zu erzeugen oder sie bei Änderungen des Datenbankschemas neu zu erzeugen. Dafür ruft die *DBManagerLocal*-Klasse die Methode *onCreate()* bzw. *onUpgrade()* automatisch auf.

```

1 public void onCreate(SQLiteDatabase db) {
2     //erzeuge user-Tabelle
3     db.execSQL(UserTbl.SQL_CREATE);
4 }
5
6 public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
```

```

7 // lösche und erzeuge die user-Tabelle
8 db.execSQL(UserTbl.SQL_DROP);
9 onCreate(db);
10 }

```

Listing 4.4: *DBManagerLocal* erzeugt die Tabelle *user*

Beim Aufrufen des DROP- und CREATE-Statements muss wegen der Tabellenabhängigkeiten auf die richtige Reihenfolge geachtet werden. Erstellt werden immer zuerst Tabellen ohne Fremdschlüssel und dann der Rest. Die Tabellen werden in genau umgekehrter Reihenfolge wieder gelöscht. Die *TabellenameDTO*-Klassen sind einfache Container-Klassen und ermöglichen einen objektorientierten Umgang mit den Ergebnissen aus Datenbankanfragen oder das Übergeben eines *DTO*-Objekts an die Datenbank zum Speichern. Im Listing 4.5 wird eine stark reduzierte *DTO*-Klasse für die Tabelle *user* veranschaulicht. Die *DTO*-Klassen haben Felder, die den Spalten der Tabelle entsprechen, welcher sie angehören. Ein *DTO*-Objekt kann auf verschiedene Arten entstehen. Aus Parametern, die sich aus Messungen und Benutzereingaben ergeben, aus einer Datenbankanfrage, die einen Cursor liefert oder einem JSON-Objekt aus einer Anfrage an den Webserver. Es kann auch notwendig sein ein Objekt mit einem Standardkonstruktor zu erzeugen und die Felder später zu initialisieren. Für diese Fälle werden vier Konstruktoren benötigt. Nicht jede *DTO*-Klasse benötigt immer alle vier Konstruktoren.

```

1 public class UserDTO {
2
3     public long _id;
4     public String vorname;
5     public String nachname;
6     public String email;
7     public String password;
8
9
10    public UserDTO(){};
11    public UserDTO(long, String, String, String, String){...}
12    public UserDTO(Cursor){...}
13    public UserDTO(JSONObject){...}
14    //getter & setter
15
16    public JSONObject toJSONObject() {... return jsonObj;}
17 }

```

Listing 4.5: *UserDTO*-Klasse in leicht reduzierter Form

Eine wichtige Methode jeder *DTO*-Klasse ist die *toJSONObject()*-Methode. Für den Fall, dass Messungen oder Benutzereingaben an den Webserver geschickt werden müssen, geschieht dies über eine REST-Schnittstelle mit einem JSON-String. Um es so einfach wie möglich zu halten die Objekte als Parameter an die REST-Schnittstelle zu senden, hat

jedes Objekt die Fähigkeit seine gespeicherten Eigenschaften als ein JSON-Objekt zurückzugeben. Die Implementierung dieser Methode veranschaulicht das Listing 4.6.

```

1 public class UserDTO {
2     public JSONObject toJSONObject() {
3         JSONObject obj = new JSONObject();
4         try {
5             obj.put(UserTbl._ID, this._id);
6             obj.put(UserTbl.VORNAME, this.vorname);
7             obj.put(UserTbl.NACHNAME, this.nachname);
8             obj.put(UserTbl.EMAIL, this.email);
9             obj.put(UserTbl.PASSWORD, this.password);
10        } catch (JSONException e) {
11            Log.d(CLASS_NAME, "toJSONObject failed: " + e.getMessage());
12        }
13        return obj;
14    }
15}

```

Listing 4.6: *UserDTO*-Klasse in leicht reduzierter Form

Das Listing 4.6 demonstriert auch gleichzeitig wie die *Tbl*-, *Columns*- und *DTO*-Klassen zusammenarbeiten. Um die richtige Spaltenbezeichnung zu nutzen, greift die *DTO*-Klasse über die *Tbl*-Klasse auf die *Columns*-Schnittstelle und dort auf die Konstanten der Spaltennamen zu. Damit eine Activity mit dem *DBManagerLocal* auf die Datenbank Zugriff bekommt, müssen Methoden für den Zugriff in der Klasse *DBManagerLocal* implementiert werden. Im Listing 4.3 wird bereits das SQL-Statement *STMT\_GET\_USER* in der *UserTbl*-Klasse gezeigt. Dieses Statement nutzt der *DBManagerLocal* in einer eigenen Methode, um eine Anfrage daraus zu bauen. Eine Beispielanfrage wird mit dem Listing 4.7 demonstriert.

```

1 public class DBManagerLocal ... {
2     ...
3     //Methode fuer eine Datenbankanfrage
4     public Cursor getUserByNachname(UserDTO user) throws SQLException{
5         //Hier wird der Nachname aus dem DTO-Objekt als Parameter in einem String-
6         //Array gespeichert.
7         String[] param = {user.nachname};
8         //getReadableDatabase liefert eine Referenz auf ein Objekt, welches die
9         //Datenbankverbindung zum Lesen oeffnet.
10        //das Statement aus der Klasse UserTbl und die Parameter werden an die Methode
11        //rawQuery uebergeben.
12        //rawQuery liefert einen Cursor auf die Antwort aus der Datenbank.
13        Cursor cursor = getReadableDatabase().rawQuery(UserTbl.STMT_GET_USER, param);
14        return cursor;
15    }
16}

```

Listing 4.7: Methode *getUserByNachname()* der *DBManagerLocal*-Klasse

Das Ergebnis der Anfrage kann in einer Activity anschließend ausgewertet werden. Eine Demonstration liefert das Listing 4.8. Hier wird zuerst mit dem if-Statement geprüft, ob ein Ergebnis in der Datenbank gefunden wurde. Die while-Schleife bewegt beim ersten Durchlauf den Cursor auf die erste Zeile der Antwort. Im Rumpf der while-Schleife werden die Ergebnisse bearbeitet, und das so lange bis die Methode `moveToNext()` ein false liefert.

```

1 public class MyActivity... {
2     ...
3     DBManagerLocal dbManager = new DBManagerLocal(MyActivity.this);
4
5     public void todo() {
6         Cursor result = dbManager.getUserByNachname(user);
7         if (result.getCount() > 1) {
8             while(result.moveToNext()) {
9                 UserDTO user = new UserDTO(result); ...
10            }
11        }
12    }
13 }
```

Listing 4.8: Methode `getUserByNachname()` der `DBManagerLocal`-Klasse

### 4.3.3 ApoCo Datenbankdiagramm

In der Abbildung 4.4 wird das ApoCo Datenbankschema als Klassendiagramm dargestellt. Die Abkürzungen *FK1*, *FK2* stehen dabei für Fremdschlüssel. Die Primärschlüssel werden als Schlüsselsymbol dargestellt. Folgende Tabellen sind im Diagramm zu sehen:

- user:

Diese Tabelle enthält Daten der Patienten. Die ApoCo-Anwendung ist so konzipiert, dass sie mehrere Patienten auf einem Smartphone verwalten kann. Jeder Patient hat seine eigenen Anmeldedaten und kann die Daten der anderen Patienten nicht einsehen.

- bloodpressure:

In dieser Tabelle werden Messprotokolle für den Blutdruck persistent gespeichert. Neben dem Datum und den Spalten für die Messwerte ist hier noch eine weitere Spalte mit der Bezeichnung *sync* zu sehen. Diese Spalte ist für die Synchronisation der Daten mit dem Webserver notwendig. Es handelt sich dabei um einen Flag der von 0 auf 1 gesetzt wird, wenn die Messwerte an den Webserver übertragen wurden.

- bodyweight:

Die Tabelle *bodyweight* speichert Messdaten zum Körpergewicht. Auch hier ist ebenfalls eine Flag-Spalte für den Synchronisationsvorgang.

- **mealenergy:**

Die Tabelle *mealenergy* repräsentiert pro Eintrag eine ganze Mahlzeit. Hier wird nur festgehalten wann die Mahlzeit eingenommen wurde und ob die Mahlzeit mit dem Webserver synchronisiert ist.

- **mealenergy\_content:**

Diese Tabelle enthält einzelne Positionen einer Mahlzeit. Sie speichert aber nur das Gewicht und die Energiemenge der jeweiligen Position, wie auch die dazugehörige EAN-Nummer. Die Tabellen *mealenergy* und *mealenergy\_content* bilden somit eine Einheit bei der Berechnung von Energieaufnahme durch die Nahrung.

- **food:**

Die Tabelle *food* dient als Ergänzung der Positionen einer Mahlzeit. Jedes Lebensmittel, das zum ersten Mal mit Hilfe des Barcodescanners erkannt wurde, wird hier für schnellen Zugriff gespeichert.

- **tageseinheiten:**

Der Arzt gibt seinen Patienten vor, wieviel Kilokalorien sie am Tag zu sich nehmen dürfen. Die aktuellen Werte werden vom Webserver geladen und in dieser Tabelle abgelegt.

## 4.4 Bluetoothkommunikation, Messung und Persistierung

Um Messdaten zu erhalten, kommuniziert ApoCo mit den Messsensoren über Bluetooth. Die eingesetzten Sensoren der Firma BodyTel verwenden für die Übermittlung der Messdaten ein Protokoll, das dem *PDU-Modus* einer SMS entspricht. PDU steht für *Protocol Data Unit* und ist eine von drei Spezifikationen, welche vom *Europäischen Institut für Telekommunikationsnormen* (ETSI)[ETS12] zum Standard für SMS-Nachrichten erklärt wurde. In diesem Protokoll werden AT-Kommandos für den Verbindungsauflaufbau eingesetzt und anschließend die Messwerte im PDU-Modus übermittelt. AT-Kommandos sind ein Befehlssatz, der zum Parametrieren und Konfigurieren von Modems genutzt wird[AT-13]. Mit einem AT-Kommando teilt der Messsensor der Software mit, wenn er im nächsten Schritt eine Nachricht übermitteln möchte. Da jeder Messsensor nur für eine bestimmte Art von Messung geeignet ist, unterscheiden sich die PDUs der einzelnen Geräte. Nach dem ein Sensor über Bluetooth mit dem Smartphone gekoppelt ist und verwendet werden soll, muss ApoCo als Bluetooth-Server auf ankommende Nachrichten hören.

Im Gegensatz zu der Bluetoothkommunikation der BodyTel-Geräte wird bei der Verbindung mit der Laborwaage für das Abwiegen von Lebensmitteln kein Kommunikationsprotokoll verwendet. Die Waage kommuniziert über eine RS-232-Schnittstelle direkt ohne die Nach-

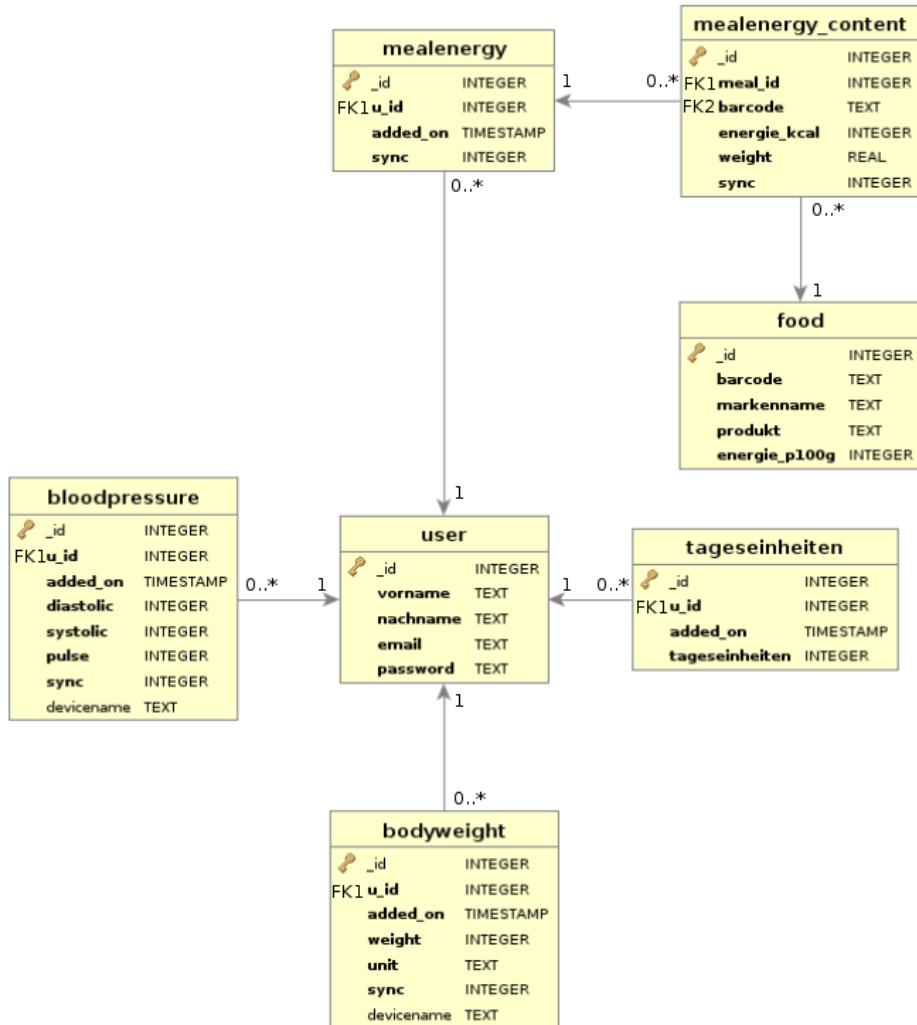


Abbildung 4.4: ApoCo Datenbankdiagramm

richt zu kodieren. An dieser Schnittstelle sitzt ein Bluetooth-Dongle und ersetzt ein serielles Kabel. ApoCo baut eine BluetoothSocket-Verbindung zu der Bluetooth-Dongle Adresse auf. Die Waage sendet die Messwerte im 8-Bit ASCII-Code. Was beim Wiegen an Daten gesendet wird, kann an der Waage parametriert werden. Für die Kommunikation mit ApoCo wird das gemessene Nettogewicht übertragen. Direkt nachdem eine Socketverbindung zwischen ApoCo und der Waage aufgebaut ist, können Daten von der Waage im ASCII-Code empfangen werden.

#### 4.4.1 Bluetoothkommunikation BodyTel

In der Abbildung 4.5 wird am Beispiel der Körpergewichtsmessung demonstriert, wie ein Verbindungsauflaufbau und Datenaustausch zwischen ApoCo und der Körperwaage zustande kommt. Nach dem Start initialisiert die Activity *ActivityBodyweight* den *BluetoothManager* und erzeugt ein Objekt vom Typ *WeightTelCreator*. Anschließend wird die Methode *listenForInquiryConnections()* des *BluetoothManager* gerufen. Dabei übergibt die Activity der Methode ein Objekt vom Typ Handler und *WeightTelCreator*. Der Handler ist zuständig für Intraprozesskommunikation zwischen der Activity und einem Thread. Kommt eine Nachricht über den BluetoothSocket rein, so kann diese nicht direkt an die Activity gegeben werden. Das kann der Thread nur über den Handler tun. Der *WeightTelCreator* erzeugt den Thread und gibt eine Referenz auf ihn zurück. Die Referenz ist eine Schnittstelle vom Typ *AccessibleIF*. Über diese Schnittstelle kann die Activity Nachrichten an den Thread senden. Dafür gibt es die folgenden Methoden:

- `writeTo():`  
Über diese Methode können Nachrichten an den Thread gesendet werden. Der Thread leitet die Nachrichten über den BluetoothSocket an den Messsensor weiter.
- `preformStart():`  
Mit dieser Methode wird der Thread von der Activity gestartet.
- `cancel();`  
Mit dieser Methode fordert die Activity den Thread auf sich zu beenden.

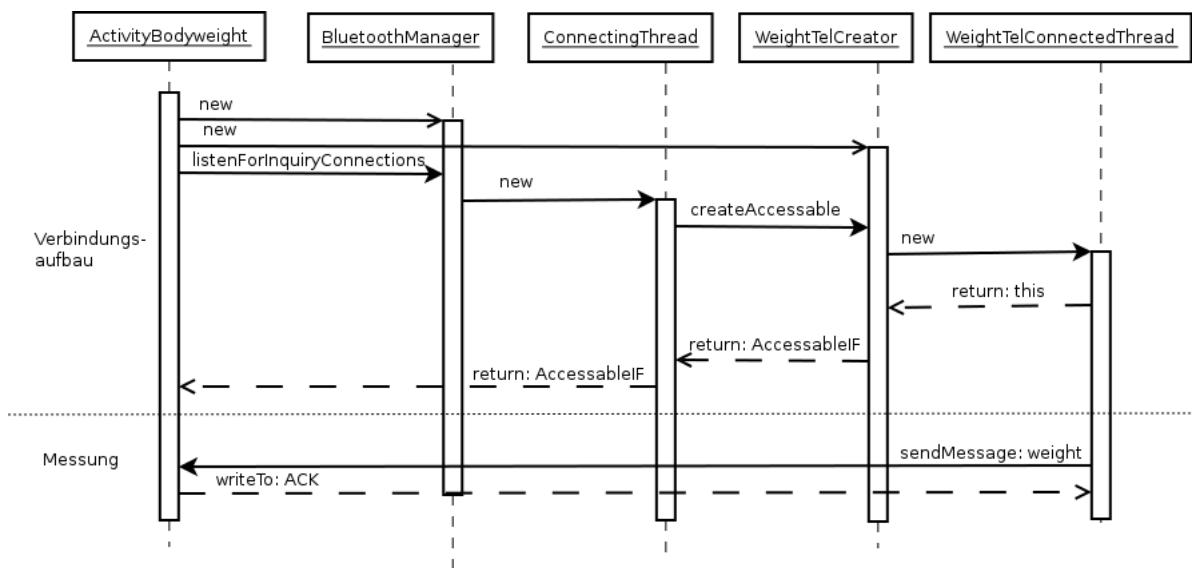


Abbildung 4.5: Sequenzdiagramm für Bluetoothverbindung und Datenaustausch

#### 4.4.2 Dekodieren einer Messung

Das Dekodieren einer Messung wird am Beispiel der Körpergewichtsmessung erklärt. Um eine Messung zu bekommen, muss zuerst das Protokoll der BodyTel-Geräte erfüllt werden. Der Ablauf dabei sieht folgendermaßen aus:

- Sensor sendet: ATE0  
Das ist ein Echo und prüft, ob jemand antwortet.
- ApoCo sendet: \n \r OK \n \r  
Antwort auf das Echo.
- Sensor sendet: AT+CMGS=142  
Das Kommando vor dem Gleichheitszeichen bedeutet, es wird eine Nachricht gesendet und die Zahl danach ist die Länge der SMS.
- Sensor sendet: Nachricht als Hexadezimalzahlen.
- ApoCo sendet: \r \n +CMGS  
Bestätigung der Nachricht.

Ende der Kommunikation.

Ab jetzt muss die Nachricht dekodiert werden. Dafür veranschaulicht die Abbildung 4.6 in einem Klassendiagramm, welche Klassen für diesen Vorgang notwendig sind.

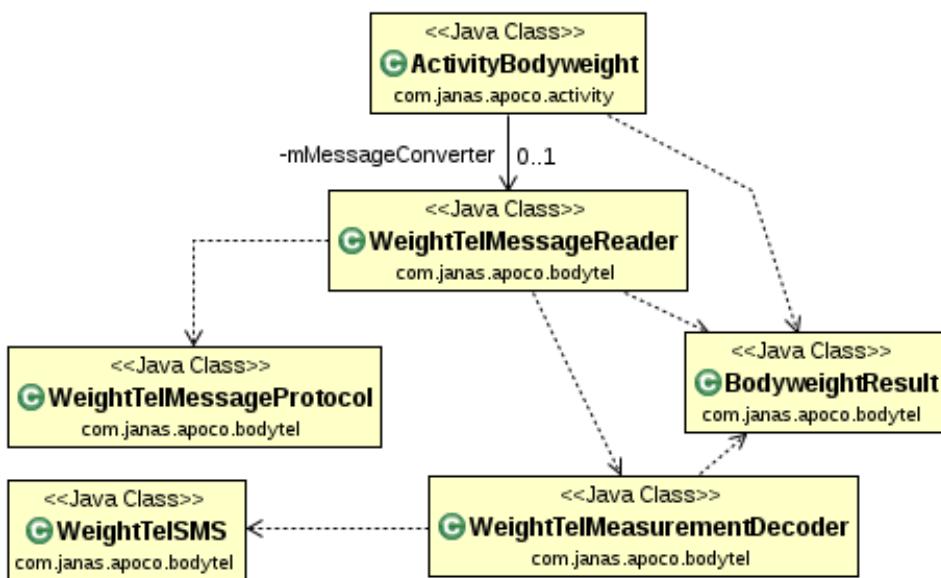


Abbildung 4.6: Klassendiagramm für Dekodieren von Nachrichten

Zum Einlesen der ankommenden Nachricht benutzt die Klasse *ActivityBodyweight* ein Objekt der Klasse *WeightTelMessageReader*. Die Klasse *WeightTelMessageReader* analysiert die Nachricht und setzt die einzelnen ankommenden Blöcke in einem Stück zusammen. Während der Analyse ordnet die Klasse *WeightTelMessageReader* die richtigen Antwort-Strings zu, die an den Sensor zurückgegeben werden. Für diese Analyse stehen in der Klasse *WeightTelMessageProtocol* die entsprechenden Konstanten bereit. Wurde die Nachricht zum Ende gelesen, wird die SMS als String an die Klasse *WeightTelMeasurementDecoder* weitergereicht. Hier wird die Methode *decodeMeasurement()* aufgerufen. Die Klasse *WeightTelMeasurementDecoder* erzeugt ein Objekt vom Typ *WeightTelSMS* und initialisiert es mit dem SMS-String. Im Konstruktor der Klasse *WeightTelSMS* wird der SMS-String an entsprechenden Stellen getrennt und in lesbare Werte umgewandelt. Nach diesem Vorgang erzeugt die Klasse *WeightTelMeasurementDecoder* ein Objekt von Typ *BodyweightResult* und initialisiert es mit den Messergebnissen aus dem *WeightTelSMS*-Objekt. Das *Result*-Objekt wird anschließend an die Activity durchgereicht. Dieser Vorgang wird durch das Sequenzdiagramm in der Abbildung 4.7 verdeutlicht.

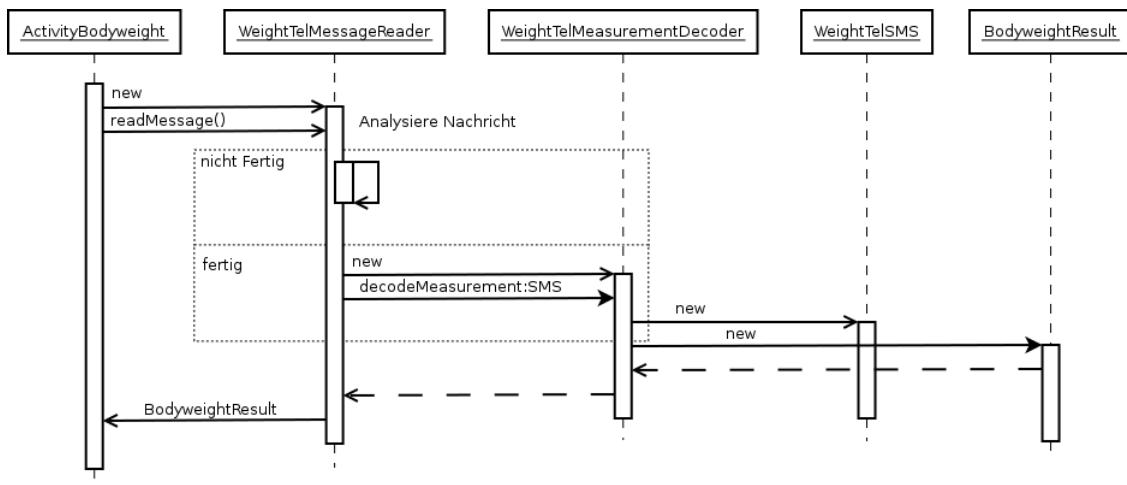


Abbildung 4.7: Sequenzdiagramm für Dekodieren von Nachrichten

#### 4.4.3 Anzeigen und Speichern der Messwerte

Nach einer Messung erscheint der neue Messwert ganz oben in einer ListView der Activity und wird anschließend in der Datenbank gespeichert. Das funktioniert folgendermaßen: Wenn das *Result*-Objekt nach dem Dekodieren der SMS an die Activity durchgegeben wurde, erzeugt der Handler der Activity ein *DTO*-Objekt. Im Fall der Körpergewichtsmessung ist das ein Objekt der Klasse *BodyweightDTO*. Dieses Objekt wird mit dem *Result*-Objekt initialisiert. Nun wird aus dem *DTO*-Objekt mit Hilfe der static Methode

`convertDTO_to_MODEL()` der Klasse `BodyweightModel`, ein Objekt der Klasse `BodyweightModel` erzeugt und an einen ArrayAdapter zum Anzeigen in der ListView übergeben. Das `DTO`-Objekt wird durch die Klasse `DBManagerLocal` in der Datenbank gespeichert. Dieser Vorgang soll durch die Diagramme in den Abbildungen 4.8 und 4.9 nochmals verdeutlicht werden.

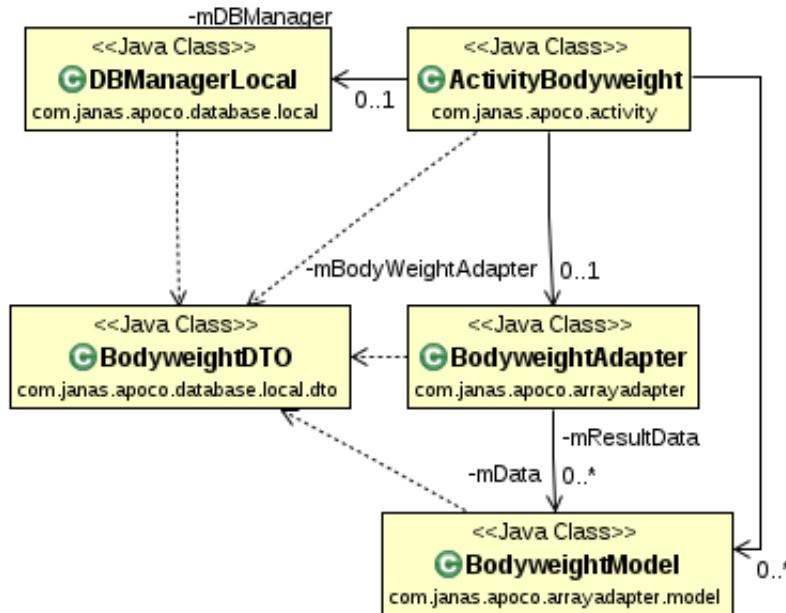


Abbildung 4.8: Klassendiagramm, Visualisierung und Speicherung der Daten

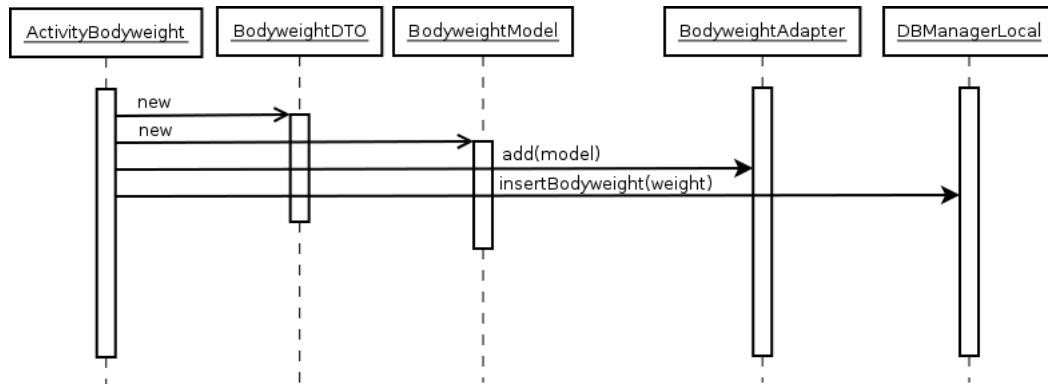


Abbildung 4.9: Sequenzdiagramm, Visualisierung und Speicherung der Daten

## 4.5 Datensynchronisation mit REST-Schnittstelle

### 4.5.1 Datensynchronisation

Nachdem die Messwerte aufgezeichnet sind, werden sie dem Arzt zugänglich gemacht. So macht sich der Arzt ein Bild vom Zustand des Patienten und stellt eine Diagnose. Für diesen Zweck werden die Messprotokolle an einen Webserver gesendet. Das Senden oder Synchronisieren startet immer wenn eine Activity beendet werden soll. Als Beispiel wird die Activity für die Blutdruckmessung näher betrachtet. Damit das GUI nicht blockiert wird, muss das Synchronisieren immer nebenläufig geschehen. Für diesen Zweck wird eine Klasse vom Typ AsyncTask genutzt. Ein AsyncTask ist in drei Arbeitsbereiche aufgeteilt. Im ersten Arbeitsbereich wird die Aufgabe noch im UI-Thread ausgeführt. Dafür ruft er seine Methode *onPreExecute()* auf. Hier kann der AsyncTask noch direkt auf die Activity zugreifen und die Aufgabe vorbereiten. Im nächsten Schritt startet der AsyncTask einen eigenen Thread und führt die Aufgabe nun in seine *doInBackground()*-Methode aus. Das ist der nebenläufige Teil der Ausführung. Ist dieser beendet, wird die Methode *onPostExecute()* aufgerufen. Er befindet sich wieder im UI-Thread. Bevor ein AsyncTask beendet wird, werden hier noch die Ergebnisse an die Activity übergeben. Das Listing 4.9 zeigt eine reduzierte Form der *SynchronizeBloodpressure*-Klasse, welche die Klasse AsyncTask erweitert. Die Messwerte werden mit einem AsyncTask gesendet. Zuerst wird die Netzwerk- und Serververfügbarkeit geprüft. Schlägt der Test fehl, so wird die Ausführung beendet. Als nächstes werden JSON-Objekte erzeugt, in denen die Messdaten abgelegt werden. Im nächsten Schritt werden Tagesprotokolle, die noch nicht synchronisiert wurden, aus der Datenbank gelesen und in JSON-Objekte gepackt. Danach wird die URL zum Webserver zusammengebaut. Dieser Vorgang ist wichtig, da zwar immer dieselbe IP genutzt wird, aber für jede Aufgabe wird ein anderes PHP-Skript vom Server aufgerufen. Die PHP-Skripte bilden eine REST-Schnittstelle für die ApoCo-Anwendung. Nachdem die URL bereitgestellt ist, wird eine Verbindung zum Webserver aufgebaut und ein *HTTPRequest* ausgeführt. Dabei werden die JSON-Strings mitgesendet. Im letzten Schritt wird die Antwort vom Server ausgewertet und der Synchronisationsvorgang in der lokalen Datenbank bestätigt. Dabei wird das *Sync*-Flag jeder synchronisierten Messung in der lokalen Datenbank auf den Wert 1 gesetzt und eine Rückmeldung an die Activity ausgegeben. Der Synchronisationsvorgang wird in Abbildung 4.10 veranschaulicht.

```

1 public class SynchronizeBloodpressure extends AsyncTask<UserDTO, Void, Boolean> {
2
3     protected void onPreExecute() {
4         ... /*Ablauf im UI-Thread*/
5     }
6     /*ab hier startet ein neuer Thread*/

```

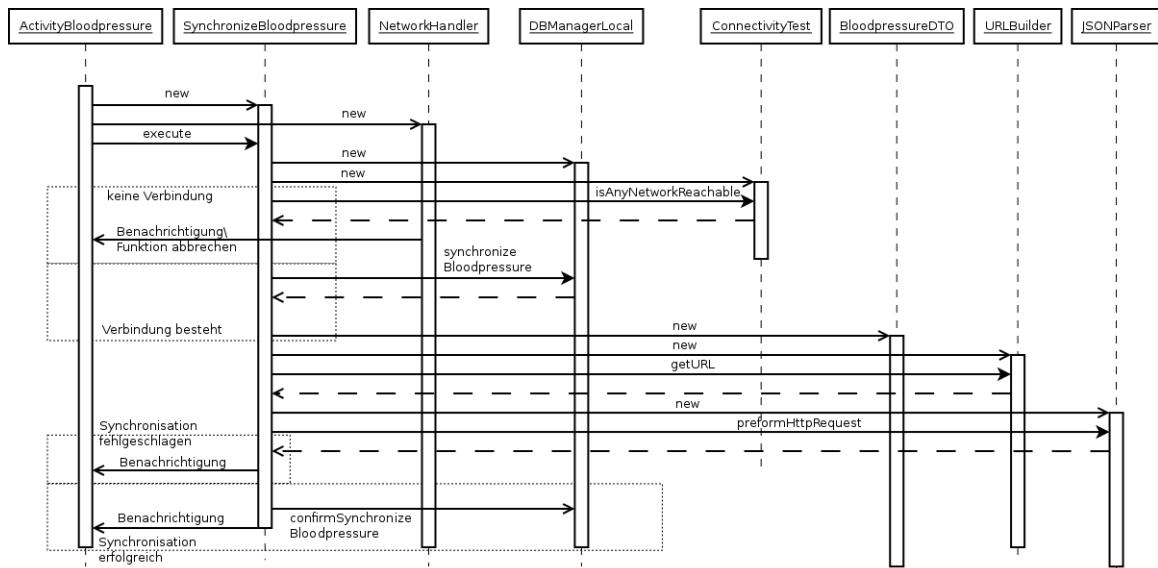


Abbildung 4.10: Sequenzdiagramm für das Senden der Tagesprotokolle an den Webserver

```

7  protected Boolean doInBackground(UserDTO... pUser) {
8      // Netzwerkverfügbarkeit pruefen
9      isNetworkConnected = new ConnectivityTest(mContext).isAnyNetworkReachable(
10          mHandlerNet);
11     if (!isNetworkConnected) return false;
12     mUser = pUser[0];
13     // JSON - Strukturen zum Senden werden angelegt
14     JSONObject user = mUser.toJSONObject();
15     JSONArray payload = new JSONArray();
16     try {
17         // Daten zum Synchronisieren aus der Datenbank laden
18         Cursor cursor = mDBManager.synchronizeBloodpressure(mUser);
19         while (cursor.moveToNext()) {
20             // Daten in JSON - Strings verpacken
21             BloodpressureDTO bpdto = new BloodpressureDTO(cursor);
22             payload.put(cursor.getPosition(), bpdto.toJSONObject());
23         }
24     } catch (JSONException e) {}
25     List<NameValuePair> params = new ArrayList<NameValuePair>();
26     // JSON - String als Parameter fuer die Uebergabe an eine URL vorbereiten
27     params.add(new BasicNameValuePair(JSON_TAG_IF.USER, user.toString()));
28     params.add(new BasicNameValuePair(JSON_TAG_IF.PAYLOAD, payload.toString()));
29     // aus Konfigurationsdaten die URL zusammenbauen
30     String url = new URLBuilder().getURL(mContext, PHP_URL_IF.INSERT_BLOODPRESSURE
31     );
32     // HTTP - Anfrage an den Webserver senden
33     jsonObject = new JSONParser().preformHttpRequest(url, JSONParser.
34         RequestMethodE.POST, params);
35     try {
36         // Antwort vom Webserver lesen
37         int success = jsonObject.getInt(JSON_TAG_IF.SUCCESS);
38         if (1 == success) {... return true;}
39     }

```

```

36         else { return false; }
37     }
38     //Ende der Ausfuehrung, jetzt wieder im UI-Thread
39     protected void onPostExecute(Boolean result) {
40         if (result) {
41             }
42             //Nachricht kann direkt an die Activity uebergeben werden, hier geschieht
43             es ueber ein Handler
44             mHandlerAct.obtainMessage(ActivityBloodpressure.
45             SYNCHRONIZING_DATA_COMPLETE).sendToTarget();
46         }
47         progressDialog.dismiss();
48     }

```

Listing 4.9: SynchronizeBloodpressure sendet Blutdruckmesswerte an den Webserver

#### 4.5.2 REST-Schnittstelle

Die Rest-Schnittstelle auf dem Webserver ist aus mehreren PHP-Skripten aufgebaut. Es gibt pro Funktionalität ein Skript. Die Persistenzschicht ist auf dem Webserver identisch modelliert, wie das in der Android-Anwendung der Fall ist. Dafür gibt es für jede Tabelle in der Datenbank eine entsprechende *COLUMNS*-, *DTO*- und *TBL*-Klasse. Diese Klassen haben die gleiche Funktionalität wie in der Android-Anwendung. Sie beschreiben die Struktur und Tabellen der Datenbank als Objekte. Für die Verwaltung der Persistenzschicht ist die Klasse *DB\_Manager* verantwortlich. Dieser Manager baut bei Bedarf eine Verbindung zur Zieldatenbank auf und leitet die Anfragen an sie weiter. Hier werden zwei unterschiedliche Datenbanken verwendet. Eine gehört zum Projekt *ClearFood*. Dort werden Informationen über die Lebensmittel abgefragt. Die andere Datenbank gehört zu der Android-Anwendung ApoCo. Hier werden die Messprotokolle aller Patienten gespeichert. Wie eine Verbindung zur Datenbank gemacht wird, veranschaulicht das Listing 4.10. Hier wird zum Aufbau der Kommunikation die Methode *connect()* der Klasse *DB\_Manager* gerufen. Da die meisten Anfragen an die ApoCo-Datenbank gerichtet sind, wird hier standardmäßig die Verbindung zu dieser Datenbank aufgebaut.

```

1 class DB_Manager {
2     private $db;
3     function connect() {
4         $this->db = new mysqli(SERVER, USER, PASSWORD, DATABASE);
5         return $this->db;
6     }
7 }
8 //Verbindungsaufbau
9 $db = new DB_Manager();
10 $db->connect();

```

---

Listing 4.10: Verbindungsauflauf zur Datenbank

Die Anfragen sendet ApoCo an den Webserver über HTTP. Dabei wird für die gewünschte Anfrage ein PHP-Skript aufgerufen. Eine solche Anfrage wird hier am Beispiel der Registrierung eines neuen Patienten im System veranschaulicht. Für das Registrieren wird das Skript *register\_user.php* benutzt. Der erste Schritt wird im Listing 4.11 demonstriert. Hier wird ein Array mit der Bezeichnung *response* und ein Objekt der Klasse *DB\_Manager* erstellt. Das Array wird für die Antwort vom Server an die Android-Anwendung genutzt.

```
1 $response = array();
2 $db = new DB_Manager();
```

Listing 4.11: Benutzer registrieren, Schritt 1

Die Kommunikation zwischen ApoCo und Webserver wird hier durchgehend über die HTTP-Methode POST gemacht. Im nächsten Schritt werden mit der Funktion *checkPOSTValues()* die Parameter in der Variablen POST auf Gültigkeit überprüft. Die Methode *checkPOSTValues()* wird im Listing 4.12 demonstriert. Sind die empfangenen Parameter in Ordnung, wird die Abfrage ausgeführt. Hat der Test fehlgeschlagen, wird eine entsprechende Nachricht als JSON-String zurück an den Aufrufer gesendet.

```
1 if (checkPOSTValues()) {
2     //ok
3 } else {
4     $response["success"] = 0;
5     $response["message"] = "required field(s) is missing";
6     echo json_encode($response);
7 }
8
9 function checkPOSTValues() {
10     return isset($_POST[UserColumns::VORNAME]) &&
11     isset($_POST[UserColumns::NACHNAME]) &&
12     isset($_POST[UserColumns::EMAIL]) &&
13     isset($_POST[UserColumns::PASSWORD]);
14 }
```

Listing 4.12: Benutzer registrieren, Schritt 2

Sind die Parameter akzeptiert, so geht es mit dem nächsten Schritt weiter. Mit der Klasse *UserDto* wird aus den Parametern ein User-Objekt erzeugt. Anschließend wird eine Verbindung zur Datenbank geöffnet. Es wird geprüft, ob die Email des neuen Benutzers bereits in der Datenbank vorhanden ist. Ist das der Fall, wird seine Registrierung abgelehnt. Handelt

es sich dabei um keine bekannte Email, wird der neue Benutzer in die Datenbank aufgenommen und eine entsprechende positive Rückmeldung an den Aufrufer gesendet. Diesen Vorgang veranschaulicht das Listing 4.13.

```

1 $user = new UserDto($_POST);
2 $db->connect();
3 if($db->emailInUse($user->getEmail()) == 0) {
4     $result = $db->registerNewUser($user);
5     if($result) {
6         $response["success"] = 1;
7         $response["message"] = "user successfully registered";
8         echo json_encode($response);
9     } else {
10        $response["success"] = 0;
11        $response["message"] = "user registering failed";
12        echo json_encode($response);
13    }
14 } else {
15     $response["success"] = 0;
16     $response["message"] = "email <" . $user->getEmail() . "> already in use";
17     echo json_encode($response);
18 }
```

Listing 4.13: Benutzer registrieren, Schritt 3

## 4.6 Barcodescanner

Für die Berechnung der zugeführten Kilokalorien müssen die Eigenschaften eines Lebensmittels erkannt werden. Die Basis dafür bildet der EAN-Code, der auf allen Nahrungsmitteln vorhanden ist. Nachdem der EAN-Code gescannt wurde, ist der Software die EAN-Nummer bekannt. Mit dieser Nummer wird in einer Datenbank nach einer Übereinstimmung gesucht. Für die Projektumsetzung greift die ApoCo-Anwendung auf eine bereits existierende Datenbank aus dem Projekt *ClearFood*. ClearFood bietet alle notwendigen Informationen über verschiedene Lebensmittel an. Aus den Angaben über Energiemenge pro 100g und dem gewogenen Gewicht wird die Energiemenge, die der Patient einnehmen möchte, berechnet. Zum Scannen des EAN-Codes wird die ZXing[EAN13c]-Bibliothek genutzt. ZXing wird *zebra crossing* ausgesprochen. Es handelt sich dabei um eine Bibliothek für 1D und 2D Barcode-Bildverarbeitung. Diese Bibliothek ist in Java implementiert und bietet eine Client-Anwendung für Android an. Der ZXing-Client ist in ApoCo integriert. Nach dem Start kann mit der Kamera auf der Rückseite des Smartphones ein Barcode erfasst werden. Der ZXing-Client analysiert das Bild und gibt, nachdem ein Barcode erkannt wurde, diesen als EAN-Nummer an die ApoCo-Anwendung zurück. Für diesen Zweck dienen die zwei Klassen *IntentIntegrator* und *IntentResult* im Paket *zxing* der ApoCo-Anwendung. Zudem muss das

Manifest der Android-Anwendung modifiziert werden. Im Listing 4.14 wird die Integration der *CaptureActivity* aus der ZXing Bibliothek demonstriert.

```
1 // ApoCo_Manifest
2 ...
3 <activity
4     android:name="com.google.zxing.client.android.CaptureActivity"
5     android:label="@string/app_name"
6     android:screenOrientation="landscape">
7 </activity>
```

Listing 4.14: ApoCo-Manifest, Integration von ZXing

Über die Klasse *ActivityMealenergy* wird der Barcodescanner verwendet. Zum Starten dient hier die Klasse *IntentIntegrator*. Die Implementierung demonstriert das Listing 4.15. Hier wird ein *Integrator*-Objekt im *OnClickListener* eines Buttons erzeugt. Anschließend wird der Scan mit der Methode *initiateScan()* gestartet.

```
1 barcodeScannerBTN.setOnClickListener(new OnClickListener() {
2     IntentIntegrator ii = new IntentIntegrator(ActivityMealenergy.this);
3     ii.initiateScan();
4 });
```

Listing 4.15: Barcodesuche starten

Der *IntentIntegrator* bekommt im Konstruktor eine Referenz auf ein Objekt der Klasse *ActivityMealenergy* übergeben. Nach der Initialisierung des Vorgangs bereitet er ein Intent zum eigentlichen Scan vor. Anschließend wird das Intent als Parameter an die Methode *startActivityForResult()* übergeben, welche über die Referenz zur *ActivityMealenergy* aufgerufen wird. Ist der EAN-Code eingelesen, wird er an die Activity *ActivityMealenergy* zurückgegeben. Das muss mit der Methode *onActivityResult()* abgefangen werden. Im Listing 4.16 wird das Abfangen demonstriert.

```
1 protected void onActivityResult(int request, int result, Intent data) {
2     super.onActivityResult(request, result, data);
3     switch(request) {
4         case IntentIntegrator.REQUEST_CODE:
5             if (RESULT_OK == result) {
6                 // Ergebnis handhaben
7                 IntentResult r = IntentIntegrator.parseActivityResult(request, result,
8                     data);
9                 String barcode = r.getContents();
10                Context activity = ActivityMealenergy.this;
11                // GetFood ist ein AsyncTask zum Zugriff auf die Datenbank
12                new GetFood(activity, new NetworkHandler(activity, true), mHandlerAct)
13                    .execute(barcode);
```

```

12         }
13     break;
14 }
15 }
```

Listing 4.16: Ergebnis der Suche abfangen

## 4.7 ApoCo Use Cases

### 4.7.1 Use Case- Diagramm

Die Abbildung 4.11 veranschaulicht ein Use Case-Diagramm der ApoCo-Anwendung. Hier werden Interaktionsmöglichkeiten des Benutzers mit der Android-Anwendung und die Kommunikation mit dem Webserver abgebildet.

### 4.7.2 Akteure

- Patient: Benutzer der ApoCo-Anwendung
- Webserver: Webserver mit Service zur Verwaltung der Tagesprotokolle

### 4.7.3 Use Case Kurzbeschreibungen

- UC01: User Login  
Der Benutzer meldet sich bei der ApoCo-Anwendung an.
- UC02: Register new User  
Ein neuer Benutzer registriert sich im System. Die Benutzerdaten werden zum Server gesendet und von diesem erlaubt oder verwiegt.
- UC03: Bodyweight  
Der Benutzer führt eine Körpergewichtsmessung durch. Zielgewicht wird vom Server geladen. Die Messung wird an den Server gesendet.
- UC04: Bloodpressure  
Der Benutzer führt eine Blutdruckmessung durch. Die Messung wird an den Server gesendet.
- UC05: Kcal  
Der Benutzer führt eine Protokollierung seiner Mahlzeit durch. Informationen zum

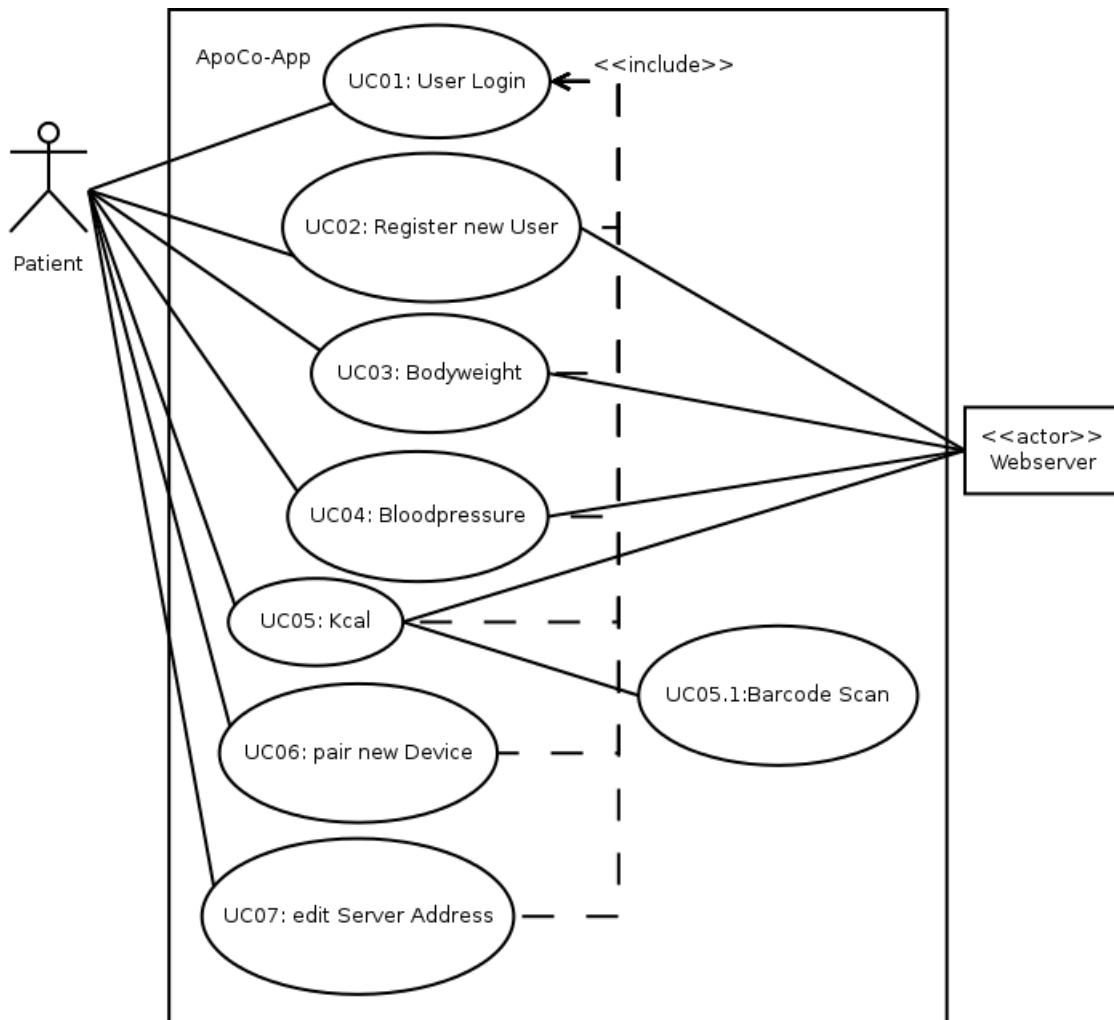


Abbildung 4.11: ApoCo Use Case-Diagramm

Lebensmittel werden vom Webserver geladen. Mahlzeitprotokoll wird an den Server gesendet.

- UC05.1: Barcode Scan

Der Benutzer scannt während einer Mahlzeitprotokollierung ein Lebensmittel mittels Barcodescanner ein.

- UC06: Pair new Device

Der Benutzer führt eine Geräte-Koppelung durch.

- UC07: edit Server Address

Der Benutzer konfiguriert die Webadresse des Servers.

## 4.8 ApoCo-GUI Gestaltung

Die grafischen Elemente, wie Hintergrundbilder, farbige Buttons, Splashscreen, Farbgebung und das Logo, sind mit einer Software für Bildbearbeitung und Illustration, speziell für die Android-Anwendung designet. Soweit es die Funktionalität der einzelnen Activity zulässt, sind alle Activities identisch strukturiert. Sie besitzen in diesem Fall immer die Elemente *Header*, *Title*, *Body* und *Footer*. Das Einhalten der Struktur soll durch ein einheitliches Layout eine angenehme und einfache Bedienung der Software ermöglichen. Die Abbildung 4.12 veranschaulicht und benennt die Hauptstrukturelemente einer Activity anhand der Start-Activity der ApoCo-Anwendung.



Abbildung 4.12: Start-Activity als Beispiel für Strukturierung

Die Strukturelemente haben jeweils eine eigene Funktion.

- Header:

Im *Header* ist immer das ApoCo-Logo und wenn notwendig, ein *Back-Button* angebracht.

- Title:

Der *Title* Bereich informiert den Benutzer auf welcher Activity er sich im Augenblick befindet.

- **Body:**

Im *Body* sind Interaktionselemente oder weitere notwendige Informationen der aktuellen Activity angebracht. Das können Buttons zum Anstoßen von Messvorgängen oder ListViews sein. Eine ListView informiert den Benutzer über alle bereits verzeichneten Messungen in der entsprechenden Activity.

- **Footer:**

Am Ende der Activity ist der *Footer*. Er beinhaltet Buttons, mit denen man in die Bereiche zum Geräte-Koppeln, Sprung zur Start-Activity und zur Server-Konfiguration gelangt.

#### 4.8.1 Bedienelemente

##### Zurück-Funktion

Der Back-Button ist eines von mehreren Möglichkeiten, mit der ein Benutzer zu der vorherigen Activity zurückkehren kann. Einige davon haben lediglich die Funktion *zurück und Daten verwerfen* und andere wiederum *zurück und Daten speichern*. Die Abbildung 4.13 veranschaulicht einen *Header* mit *Back-Button*.



Abbildung 4.13: Header einer Activity mit *Back-Button*

Die Abbildung 4.14 zeigt die Activity für Blutdruckmessung. Mit Buchstaben sind hier alle Möglichkeiten der Software gekennzeichnet, die zurück in die vorherige Activity führen.

- a) *Back-Button*, Daten werden nicht gespeichert, zurück zur vorherigen Activity.
- b) *OK-Button*, Daten werden gespeichert, anschließend zurück zur vorherigen Activity.
- c) Sprung zur *Start-Activity*, Daten werden nicht gespeichert.

Neben einer Softwaremöglichkeit ist auch der Hardware-Button des Smartphones implementiert. In der Abbildung 4.15 ist der Hardware-Button mit einem roten Kreis gekennzeichnet. Hier findet kein Speichern der Daten statt. Befindet man sich in der *Start-Activity*, beendet ein Drücken auf die Hardware-Taste die ApoCo-Anwendung.

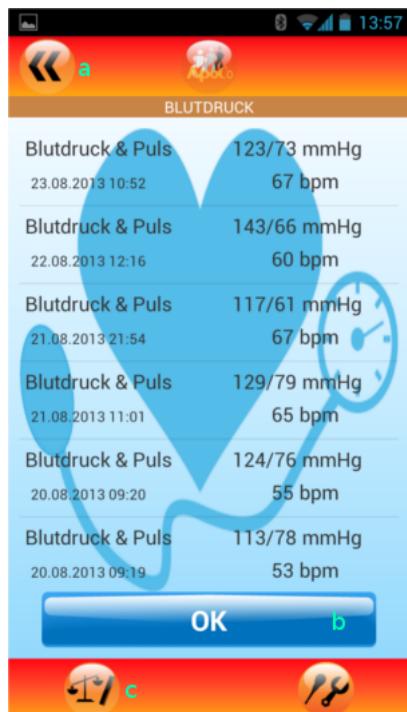


Abbildung 4.14: Zurück-Möglichkeiten einer Activity



Abbildung 4.15: Zurück mit einem dem Hardware-Button des Smartphones

## Geräteverwaltung

In der Abbildung 4.16 ist der Button zur Geräteverwaltung abgebildet. Mit diesem Button gelangt man in den Bereich der Android-Anwendung, in welchen Messgeräte gekoppelt und Servereinstellungen vorgenommen werden können.

Die Abbildung 4.17 veranschaulicht anschließend die Activity der Geräteverwaltung. Hier wird über Radio-Buttons eine Messung ausgewählt und über eine der Koppelungsfunktionen die externe Hardware verbunden. Ob hier das Koppeln als Server oder Client gewählt wird, ist von dem externen Messsensor abhängig.

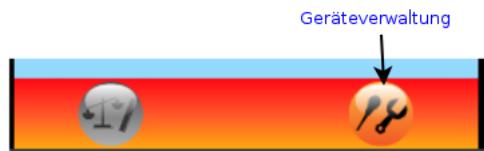


Abbildung 4.16: Geräteverwaltung

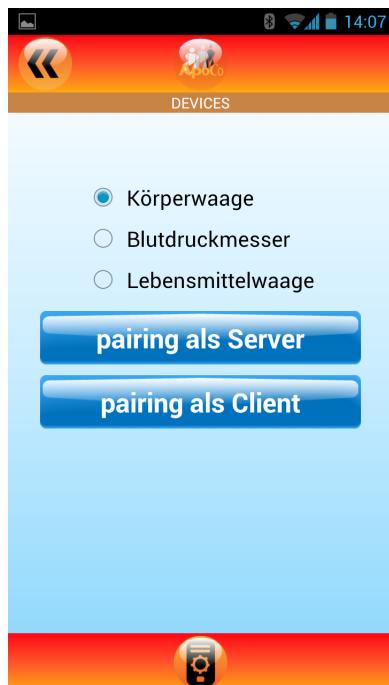


Abbildung 4.17: Koppelungsactivity

Im *Footer*-Bereich der Abbildung 4.17 ist auch der Button für die Serverkonfiguration zu finden. Die Serverkonfiguration wird mittels einer Activity realisiert, die sich allerdings im *Dialog-Stil* präsentiert. Diese Aktivity wird in der Abbildung 4.18 veranschaulicht und das Listing 4.17 erläutert, wie man eine Activity über das *AndroidManifest* als Dialog erscheinen lässt.

```

1 <activity
2   android:name="com.janas.apoco.activity.ActivityServerOptions"
3   //die zweite Zeile 1\"asst die Activity als Dialog erscheinen
4   android:theme="@android:style/Theme.Dialog" >
5 </activity>
```

Listing 4.17: Activity als Dialogfenster erscheinen lassen

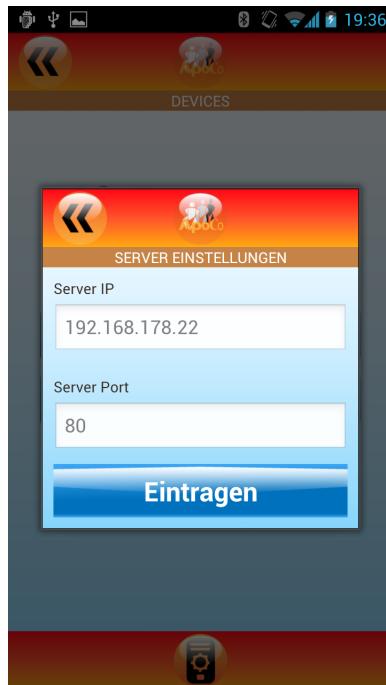


Abbildung 4.18: Serverkonfiguration

#### 4.8.2 Activity-Wechsel mit Animation

Der Wechsel von einer Activity zur anderen wird mit Animationsskripten gemacht. Dabei wird die erste Activity zum Bildrand rausgezogen und die zweite Activity ins Bild hineingezogen. Handelt es sich um den Wechsel zu einer Activity, die sich auf eine Parametrierung der Software bezieht, so verläuft die Animation von unten nach oben. Wechselt man aber zur einer Messung, so verläuft die Animation von rechts nach links. Eine vollständige Animation besteht immer aus den zwei Teilen *Eingangs*- und *Ausgangs*-Animation. Die Animationen sind im Odrner */res/anim/* in XML-Dateien abgelegt. Das Listing 4.18 veranschaulicht die Definition einer Animation und wie man sie für den Wechsel der Activities nutzt.

```

1 //right_side_in.xml
2 <?xml version="1.0" encoding="utf-8"?>
3 <set xmlns:android="http://schemas.android.com/apk/res/android"
4     android:interpolator="@android:anim/accelerate_decelerate_interpolator" >
5     <translate
6         android:fromXDelta="100%" 
7         android:toXDelta="0"
8         android:duration="500"/>
9 </set>
10 //Anwendung der Animation in einer Activity
11 finish();
12 overridePendingTransition(R.anim.left_side_in, R.anim.left_side_out);

```

Listing 4.18: Animation zum Verlassen einer Activity

Im Listing 4.18 ist ein Beispiel einer Animation für die reinkommende Activity zu sehen. Durch den Aufruf der Methode `overridePendingTransition()` wird eine Animation gestartet. Diese Methode hat jedes Objekt vom Typ *Activity*. Sie muss zum Aufrufen überschrieben werden. Der Aufruf muss anschließend immer nach einer Methode erfolgen, welche für das Schließen einer Activity zuständig ist. Das sind die Methoden `finish()` und alle aus der Familie `startActivity...()`. Die Abbildung 4.19 zeigt zwei Screenshots mit jeweils einer Animation. Bei der ersten ist der Wechsel in eine Messung zu sehen und in der zweiten zum Koppelungsvorgang.

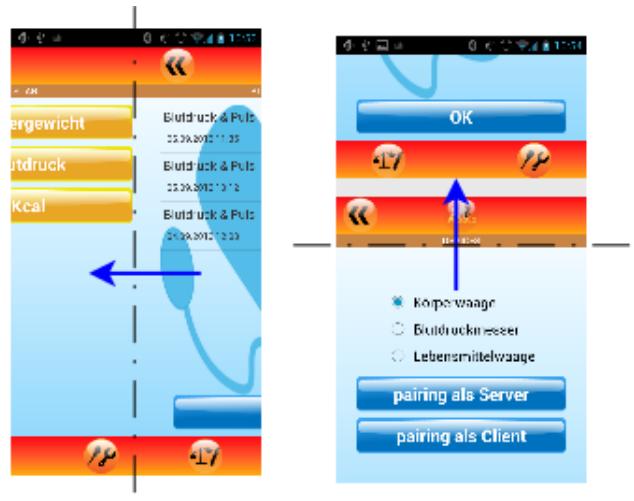


Abbildung 4.19: Übergangsanimation

### 4.8.3 Views und Struktur der gegenseitigen Aufrufe

In der Abbildung 4.20 ist die gesamte Viewschicht der Android-Anwendung veranschaulicht. Die großen blauen Makierungen identifizieren die Views. Wird ein Button gedrückt, weist eine kleine blaue Makierung darauf hin, welche View als nächstes geöffnet wird. Eine Ausnahme ist die View *Splashscreen* (*s1*) und der Barcodescanner (*b2*). Der Übergang von *Splashscreen* zur Anmelde-View (*a1*) geschieht über einen Timer. Ist der Timer abgelaufen, so startet der Übergang automatisch. Im Fall der Barcodescanner-View hängt der Übergang davon ab, ob nach dem Scan der Barcode in der Datenbank für Lebensmittel gefunden wurde oder nicht. War die Suche erfolgreich, dann geht es mit der View (*k3*) weiter. Ist die Suche fehlgeschlagen, findet der Übergang zur View (*k2*) statt.

Um den Zusammenhang der Views besser darzustellen, zeigt die Abbildung 4.21 ein Zustandsdiagramm. Dieses Zustandsdiagramm stellt die Beziehungen der Views untereinander dar. Damit die Komplexität des Diagramms überschaubar bleibt, wurde auf die Modellierung der Buttons für *akzeptierende* und *ablehnende* Aktionen verzichtet. Im Diagramm sind nur die möglichen Übergänge zwischen den Views dargestellt. Die Bezeichner der Zustände im Diagramm entsprechen den Makierungen in der Abbildung 4.20, welche die Views identifizieren. Zusätzlich sind die Übergangspfade beschriftet und nummeriert. Die Beschriftungen sagen aus, welche Aktion auf diesem Pfad vollzogen wird. Die Nummerierungen bilden eine Hierarchie und sollen anzeigen auf welche Weise ein Pfad verfolgt werden darf. Es ist nur erlaubt einem wegführenden Pfeil zu folgen, wenn der Pfad zuvor die gleiche Nummerierung hatte, oder seine Nummerierung dem Folgepfeil als Präfix vorangestellt ist. Gelangt man über einen Pfad zu einem verzweigenden Zustand, dann wird die Nummerierung des letzten Pfeils zum Präfix der nächsten Pfade.

Betrachtet man den Zustand  $(r1)$  in Abbildung 4.21., so ist es möglich von  $(r1)$  in den Zustand  $(s3)$  hin und wieder zurück zu wechseln. Da keiner der restlichen Pfeile von  $(s3)$  aus mit der 1 im Präfix nummeriert ist, ist es beispielsweise nicht erlaubt von  $(s3)$  nach  $(k4)$  oder den übrigen Zuständen zu wechseln.

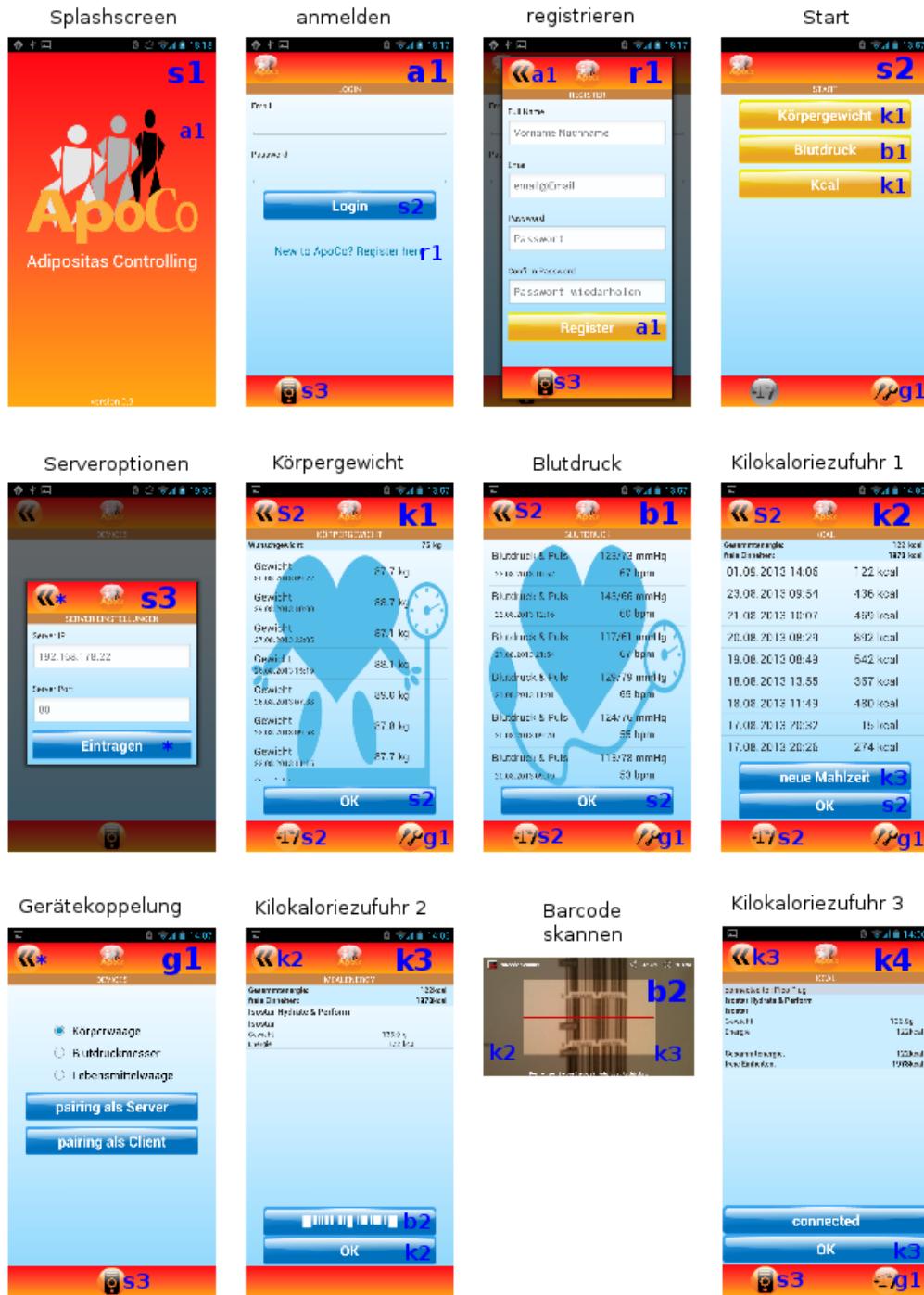


Abbildung 4.20: Darstellung aller Activities

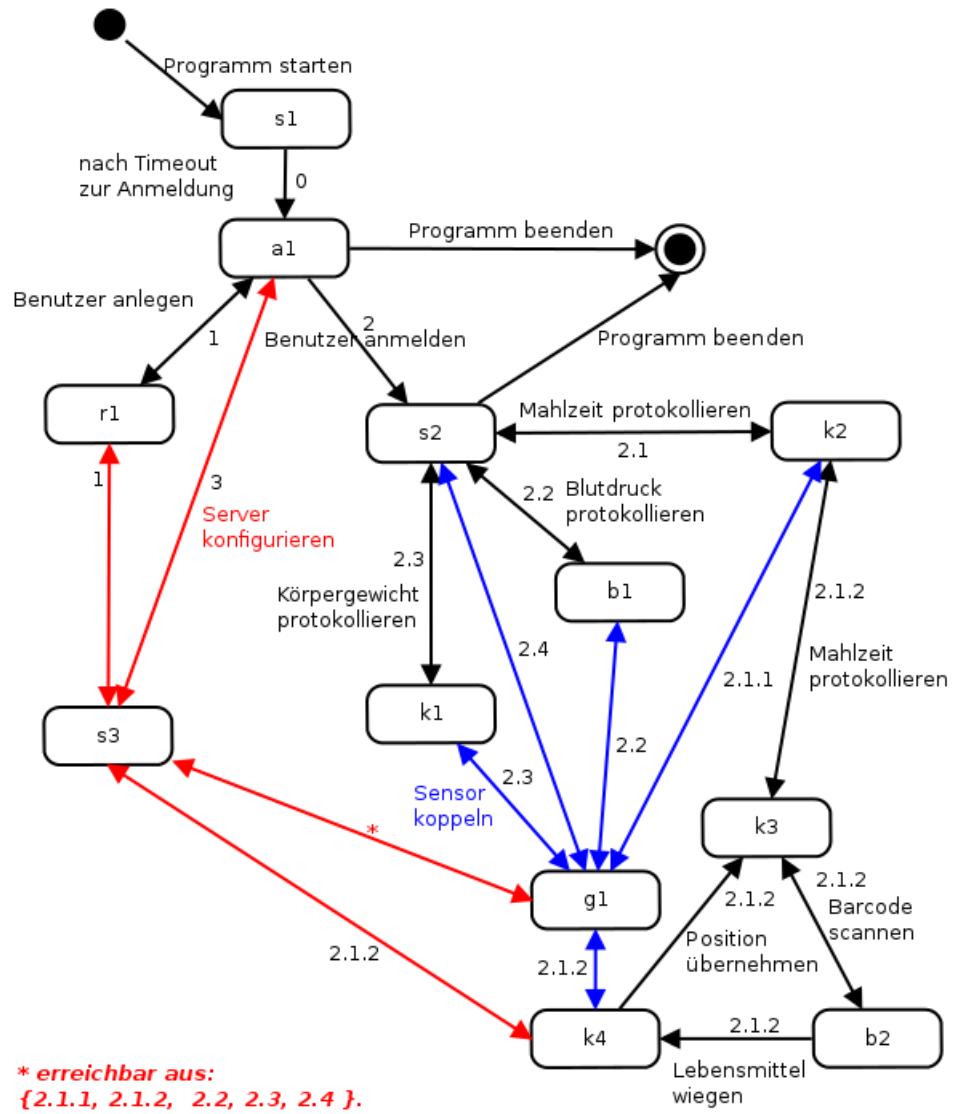


Abbildung 4.21: Zustandsdiagramm für Beziehungen der gegenseitigen Aufrufe

# 5 Webanwendung: Controlling-Software

In diesem Kapitel wird auf die Webanwendung eingegangen. Sie wird durch die Modellierung, Architektur, Implementierung und den GUI-Design beschrieben.

## 5.1 Modellierung

Die Webanwendung für den Arzt ist, wie die Android-Anwendung, stark modularisiert. Sie ist ebenfalls als *MVC*- und *Client-Server*-Architektur realisiert. Für die Umsetzung werden folgende Mittel eingesetzt:

- HTML:

Mit HTML wird die Struktur der Webanwendung aufgebaut.

- CSS:

Mit CSS wird die Seite gestaltet. CSS definiert hier die Positionierung und Farbgebung der einzelnen Elemente der Anwendung.

- PHP:

PHP dient hauptsächlich dem Zugriff auf die Datenbank über eine REST-Schnittstelle.

- JavaScript:

JavaScript setzt die Anwendungslogik auf dem Client um. Mit der Skriptsprache wird die Webanwendung als Ajax-Anwendung umgesetzt.

## 5.2 Anwendungslogik

Die Anwendungslogik ist auf mehrere Skripte aufgeteilt. Jedes ist verantwortlich für eine bestimmte Teilaufgabe.

- ajaxmanager.js:

Der *AjaxManager* stellt Zugriffsmethoden für die REST-Schnittstelle zur Verfügung.

Beim Aufruf einer Methode werden Parameter und eine Referenz auf das Aufruferobjekt mitgegeben. Der *AjaxManager* wählt das richtige PHP-Skript aus und delegiert den Aufruf weiter an die *AJAX*-Klasse. Anschließend wird die Anfrage über Ajax im Hintergrund des Browsers ausgeführt. Das Listing 5.1 veranschaulicht diesen Vorgang durch einen Ausschnitt der Implementierung der *AjaxManager*-Klasse.

```

1 function AjaxManager() {
2     this.getUserList = function(callbackObject, keyValueList) {
3         var url = "/apoco/apoco_web_ui/db_url/user_list.php";
4         go(callbackObject, keyValueList, url);
5     };
6
7     function go(callbackObject, keyValueList, url) {
8         var ajax = new AJAX();
9         var params = keyValueList.toParams();
10        ajax.sndReqGET(url, params, callbackObject);
11    }
12}

```

Listing 5.1: Beispiel für *AjaxManager*-Implementierung

- *xhrobj.js*:

Hier befindet sich die *AJAX*-Klasse. Sie erzeugt ein XHR-Objekt, mit dem die Daten im Hintergrund des Webbrowsers an einen Webserver gesendet werden. Das Senden kann mittels der beiden Methoden *sndReqGET()* und *sndReqPOST()* geschehen. Die Aufrufe werden in der Webanwendung ausschließlich über *GET* getätigten. Diese Vorgehensweise wird durch das Listing 5.2 veranschaulicht.

```

1 this.sndReqGET = function(url, params, cbo) {
2     callbackObject = cbo;
3     if (null == xhrObj) {
4         alert("An Error occurred when trying to initialize XMLHttpRequest!");
5     } else {
6         xhrObj.open("get", url + params, true);
7         xhrObj.onreadystatechange = handleResponse;
8         xhrObj.send(null);
9     }
10}

```

Listing 5.2: Beispiel für eine Methode zum Senden über Ajax

Eine Antwort vom Server wird mit der Funktion *handleResponse()* im Listing 5.3 behandelt. Die Implementierung verdeutlicht das Zusammenspiel der Anwendung. Wenn die Anfrage fertig ist und eine Antwort empfangen wird, ruft die *handleResponse()*-Funktion die *update()*-Methode des *callbackObject* auf und übergibt ihm die Serverantwort als JSON-String. Hinter dem *callbackObject* steckt hier die Klasse *Drawer*, da

diese sich um die graphische Darstellung der Serverrückmeldung kümmert.

```

1 var handleResponse = function() {
2     if (xhrObj.readyState==4 && xhrObj.status==200) {
3         var json = JSON.parse(xhrObj.responseText);
4         if (json.success == 1) {
5             if (null != callbackObject) {
6                 callbackObject.update(json);
7             }
8         }
9     }
10 };

```

Listing 5.3: handleResponse Funktion zum Auffangen von Serverantworten

- `btn_init.js`:

Dieses Skript bindet an alle *onclick*-Methoden von Buttons die entsprechenden Funktionen. Hier wird auch reguliert welche Elemente wann sichtbar sein sollen und wie sie zusammen interagieren.

- `calendar.js`:

In diesem Skript steckt die gesamte Anwendungslogik für den Kalender. Außerdem enthält es einige Konstanten für einen Modifizierer mit der Bezeichnung *calendar-ResolutionModif*. Hier wird ein Zeitintervall hinterlegt. Die Software kombiniert das Zeitintervall mit einem Zeitstempel und berechnet so den Zeitrahmen, für den die Daten aus der Datenbank gelesen werden sollen. Zusätzlich orientiert sich der Algorithmus zum Zeichnen des Graphen an diesem Modifizierer. Er berechnet damit die Abstände der angezeigten Daten auf der Zeitachse. Zur Auswahl stehen die Werte Tag, Woche und Monat zur Verfügung.

- `drawer.js`:

Das Objekt *Drawer* ist verantwortlich dafür, dass nach jeder Benutzerinteraktion, Informationen als Graph oder Liste dargestellt werden. Zusätzlich gibt es hier zwei Modifizierer für die Auswahl was gezeichnet werden soll. Der Modifizierer *viewValues-Modif* gibt an, was im Graphen gezeichnet wird. Dabei werden die entsprechenden Werte über Ajax aus der Datenbank geladen. Zur Auswahl gibt es die Möglichkeiten Blutdruck und Puls, Kilokalorien und Körpergewicht, alle diese vier Messungen zusammen oder die Liste der Mahlzeiten. Mit dem Modifizierer *DRAW\_MODIF* wird festgelegt welche Webseitelemente gezeigt werden sollen. Dafür gibt es die Auswahlmöglichkeiten Patientenübersicht, Graph, Mahlzeitenliste und Patienten-Preview.

- `graph_drawer.js`:

Dieses Skript beinhaltet Hilfsfunktionen zum Zeichnen. Nachdem Daten aus der Da-

tenbank gelesen wurden, bekommt das Objekt *GraphDrawer* diese übergeben und stellt sie in einem Graph oder einer Liste dar. Der Graph wird mit JavaScript über das *tag canvas* gezeichnet. Dieses wird nur in ganz modernen Webbrowsern unterstützt.

Internet Explorer ab 9.0

Mozilla Firefox ab 3.6

Google Chrome ab 14.0

- **key\_value\_list.js:**

Die Klasse *KeyValuePair* ist eine Liste aus Wertepaaren. Mit der Methode *addKeyValue()* werden neue Wertepaare eingetragen und mit der Methode *toParams()* wird ein String aus den Wertepaaren ausgegeben. Dieser String dient als Parameterübergabe an die *HTTP-GET*-Methode zum Aufruf einer Anfrage an die REST-Schnittstelle.

- **key\_value\_pair.js:**

Hier ist die Klasse *KeyValuePair* implementiert. Objekte dieser Klasse werden in einer *KeyValuePairList* gespeichert. Das Listing 5.4 gibt ein Beispiel für die Implementierung und Verwendung der Klassen *KeyValuePairList* und *KeyValuePair*.

```

1 //KeyValuePair
2 function KeyValuePair(key, value) {
3     this.key = key;
4     this.value = value;
5 }
6 //KeyValuePairList
7 function KeyValuePairList() {
8     var keyValuePairs = new Array();
9     var elements = 0;
10    this.addKeyValue = function(kv) {
11        if (KeyValuePair.prototype.isPrototypeOf(kv)) {
12            keyValuePairs[elements] = kv;
13            elements++;
14        }
15    };
16    this.toParams = function() {
17        var params;
18        for (var index in keyValuePairs) {
19            if (index > 0) {
20                params += "&";
21            } else if (index == 0) {
22                params = "?";
23            }
24            params += keyValuePairs[index].key + "=" + keyValuePairs[index].
25                value;
26        }
27    }
28 }
29 
```

```

26     return params;
27 }
28 }
29
30 var paramlist = new KeyValueList();
31 paramlist.addKeyValue( new KeyValuePair("datum", "2013-09-06"));
32 paramlist.addKeyValue( new KeyValuePair("name", "Dawid"));
33 var ergebnis = paramlist.toParams();
34 //ausgabe: ?datum=2013-09-06&name=Dawid

```

Listing 5.4: handleResponse Funktion zum Abfangen von Serverantworten

Die Ausgabe aus dem Listing 5.4 kann so als Parameter zum Absenden mit der HTML-GET-Methode an die URL gehängt werden.

- **page\_chooser.js:**

Die Webanwendung ist im Grunde genommen aus zwei Webseiten konzipiert. Beim Start befindet sich der Benutzer auf der Webseite *Patientenliste*. Wird ein Patient ausgewählt und klickt der Benutzer auf Details, so befindet er sich auf der Webseite *Detailansicht*. Mit Ajax verschmelzen beide Webseiten zu einer Fundamentseite und es werden nur die Inhalte im Webdokument per Ajax ausgetauscht. Die Klasse *PageChooser* ist hier eine Art Zustandsautomat und bietet zum Umschalten zwischen den beiden Ansichten den Modifizierer *PAGE\_MODIF* an.

- **apoco\_web\_ui\_init.js:**

Dieses Skript wird beim Start der Websoftware als letztes geladen. Es wartet ab bis die vollständige Webseite geladen ist und initialisiert die Anwendung. Dabei wird das Gerüst für den Kalender aufgebaut. Es bietet zudem Funktionen, um nach einer Interaktion mit dem Benutzer den Kalender neu zu zeichnen.

## 5.3 Klasse Calendar

Die Klasse *Calendar* ist ein zentrales Element der Anwendungslogik der Webanwendung. Hier befinden sich Methoden zur Berechnung und Auswahl eines Datums oder Zeitraums, von denen so gut wie die gesamte Softwarefunktionalität abhängig ist. Dazu gehören zum Beispiel das Zeichnen des Graphen oder das Anfragen von Patienteninformationen für einen ausgewählten Zeitabschnitt der Therapie. Die Abbildung 5.1 zeigt das *Calendar-Widget* der Websoftware. Dieses Widget wird in der *Calendar*-Klasse berechnet. Nach der Berechnung werden die Daten zum Anzeigen in der Präsentationsschicht weitergegeben.

In der Klasse Calendar sind folgende Klassen-Member enthalten:



Abbildung 5.1: Calendar-Widget

- **calendarResolutionModif:**

Dieses Element wurde bereits erläutert. Es ist ein Modifikator für die Auflösung der Zeitachse.

- **currentDate:**

Hier wird das aktuelle Datum hinterlegt.

- **viewedMonthStart:**

In dieser Variablen ist ein Datum-Objekt für den angezeigten Monatsanfang hinterlegt. Der Monatsanfang soll immer griffbereit sein. Er ist notwendig für die Berechnung der richtigen Zelle im Kalender, in der ein Tag eingetragen wird. In der Abbildung 5.1 ist zu sehen, dass der erste Tag des angezeigten Monats September, ein Sonntag ist. Er taucht nicht in der ersten, sondern in der letzten Zelle im Kalender auf. Davor und am Ende des Monats werden die Zellen grau ausgefüllt und nummeriert. Für diese Kalkulationen ist die Variable *viewedMonthStart* besonders wichtig.

- **markedDate:**

Je nach Zeitachsenauflösung wird ein Tag, eine Woche oder der Monat selbst als markiert gewählt. Die Variable *markedDate* speichert den markierten Tag. Der Hintergrund eines markierten Tages wird ebenfalls, wie in der markierten Woche zu sehen ist, blau gefüllt.

- **markedWeek:**

Die Variable *markedWeek* ist ein Array und speichert insgesamt sieben Zellen. Diese Zellen repräsentieren eine markierte Woche. Im Widget werden diese Zellen anschlie-

ßend blau gezeichnet.

- **drawMatrix:**

Das zweidimensionale Array *drawMatrix* speichert pro Zelle eine Datenstruktur. Die Elemente der Datenstruktur beinhalten Informationen wie die Zelle in der Präsentationsschicht gezeichnet werden soll. Hier werden folgende Informationen für je eine Zelle des Kalenders hinterlegt:

*DAY\_NUMBER*: Zahl für den Tag im Kalender.

*ONMONTH*: Sagt aus, ob die Zelle sich im aktuellen Monat (schwarz beschriftet) oder außerhalb des Monats (grau beschriftet) befindet.

*DATUM*: Hier wird ein vollständiges Datumsobjekt für diese Zelle hinterlegt.

*HIGHLIGHT*: Diese Variable ist für eine zukünftige Funktionalität des Kalenders reserviert.

*CURRENT\_DAY*: Das Flag sagt aus, ob das Datum der Zelle dem aktuellen Tag entspricht.

*MARKED*: Dieses Flag sagt aus, ob die Zelle markiert ist.

Abgesehen von der internen Logik, bietet die Klasse *Calendar*, Methoden für die Webanwendung an und lässt sich so durch den Benutzer kontrollieren.

- **recalculate:**

Die Methode *recalculate()* wird automatisch gerufen, wenn eine Änderung durch den Benutzer am Kalender erfolgt. Wird zum Beispiel zum nächsten Monat gewechselt, so wird die *drawMatrix* neu berechnet und kann neu gezeichnet werden.

- **nextMonth, prevMonth:**

Der Wechsel zum nächsten oder zum vorherigen Monat wird über die Methoden *nextMonth()* und *prevMonth()* gemacht.

- **resolutionModif:**

Hat der Benutzer die Auflösung von Woche auf Tag oder Monat umgestellt, so wird das intern über die Methode *resolutionModif()* ausgeführt.

- **jumpToCurrentDay:**

Die Methode *jumpToCurrentDay()* erlaubt dem System nach Interaktion des Benutzers zum aktuellen Tag, Monat oder zur aktuellen Woche zu springen.

- **prevDay, nextDay:**

Der Benutzer hat auch die Möglichkeit in der Graphen- und Mahlzeitenansicht mit der Intervalllänge von einem Tag vor oder zurück zu gehen. Das wird intern über die

Methoden *nextDay()* und *prevDay()* getan.

## 5.4 Kalendervisualisierung

Zum Darstellen des Kalenders auf der Webseite wird das DOM-Konzept benutzt. Im Skript *apoco\_web\_ui\_inits.js* befindet sich die Funktion *redrawCalendarWidget()*. Mit jeder Interaktion des Benutzers mit dem Kalender wird der Kalender mit dieser Funktion neu gezeichnet. Es wird nun erläutert wie dieses Prinzip funktioniert.

Mit der Methode *buildCalendarTable()* wird beim Start der Anwendung eine HTML-Tabelle in das Webdokument gezeichnet. Das geschieht dynamisch beim Aufruf der Webseite. Dabei wird nur die Struktur des Kalenders vorgegeben. Zuerst wird der Kalenderkopf mit den Bedienungselementen, der Datumanzeige und die Zellen mit der Beschriftung der Wochentage erstellt. Darauf folgen die einzelnen Zellen für die Tage im Monat, diese werden aber leer belassen. Jede Zelle der Tabelle, auf die später zugegriffen werden soll, bekommt eine *id* zugewiesen. Im Listing 5.5 wird demonstriert, wie die Zellen erzeugt werden.

```

1 for(var row = 0; row < 6; row++) {
2
3 //hier wird eine Zeile erzeugt
4   var calRow = document.createElement("tr");
5   var calRow_class = document.createAttribute("class");
6   var calRow_id = document.createAttribute("id");
7   calRow_class.nodeValue = "calendar-row";
8   calRow_id.nodeValue = "calRow_" + row;
9   calRow.setAttributeNode(calRow_class);
10  calRow.setAttributeNode(calRow_id);
11 //hier werden Zellen fuer die aktuelle Zeile erzeugt
12  for (var col = 0; col < 7; col++) {
13    var calDay = document.createElement("td");
14    var calDay_class = document.createAttribute("class");
15    calDay.setAttributeNode(calDay_class);
16    calRow.appendChild(calDay);
17  }
18 //eine fertige Zeile wird an die Tabelle angehaengt
19  tbody.appendChild(calRow);
20 }
```

Listing 5.5: Erzeugen von Zellen im Kalender per DOM

Die Klasse *Calendar* kalkuliert alle notwendigen Zelleninformationen. In einem CSS-Dokument befinden sich Klassen, die das Erscheinungsbild des Kalenders in Form und Farbe beeinflussen. Welche CSS-Klassen einer Zelle zugeordnet werden, wird aus der *drawMatrix* im *Calendar*-Objekt gelesen. Diesen Arbeitsschritt übernimmt die Funktion *redrawCalendarWidget()* im *apoco\_web\_ui\_inits.js* Skript. Das Listing 5.6 demonstriert den Ablauf dieser Funktion in Kurzform.

```

1 for(var row = 0; row < 6; row++) {
2     var calRow = document.getElementById("calRow_" + row);
3     //loesche Eintraege in dieser Zeile
4     calRow.innerHTML = "";
5     //erzeuge die Zeile mit neuen Eintraegen
6     for (var col = 0; col < 7; col++) {
7         var calDay = document.createElement("td");
8         var calDay_class = document.createAttribute("class");
9         //stelle fest, ob die Zelle zum aktuellen Monat gehoert.
10        if (calendar.drawMatrix[row][col][CMZ.ONMONTH]) {
11            calDay_class.nodeValue = "calendar-day-onmonth";
12        } else {
13            calDay_class.nodeValue = "calendar-day-offmonth";
14        }
15        //Feststellung, ob die Zelle markiert wird
16        if (calendar.drawMatrix[row][col][CMZ.MARKED]) {
17            calDay_class.nodeValue = calDay_class.nodeValue + " " + "calendar-day-
18                marked";
19        }
20        //repraesentiert die Zelle den aktuellen Tag?
21        if (calendar.drawMatrix[row][col][CMZ.CURRENT_DAY]) {
22            calDay_class.nodeValue = calDay_class.nodeValue + " " + "calendar-day-today
23                ";
24        }
25        calDay.setAttributeNode(calDay_class);
26        //Tag als Zahl in die Zelle eintragen
27        calDay.innerHTML = "<span>" + calendar.drawMatrix[row][col][CMZ.DAY_NUMBER] +
28            "</span>";
29        calDay.datum = new Date(calendar.drawMatrix[row][col][CMZ.DATUM]);
30        calDay.calendar = calendar;
31        ...
32        calRow.appendChild(calDay);
33    ...
34 }
35 }
```

Listing 5.6: Ablauf der redrawCalendarWidget-Funktion

## 5.5 Use Cases der Webanwendung

### 5.5.1 Use Case-Diagramm

Die Abbildung 5.2 veranschaulicht ein Use Case-Diagramm der Webanwendung. Hier werden Interaktionsmöglichkeiten des Arztes mit der Software und die Kommunikation der Software mit dem Webserver dokumentiert.

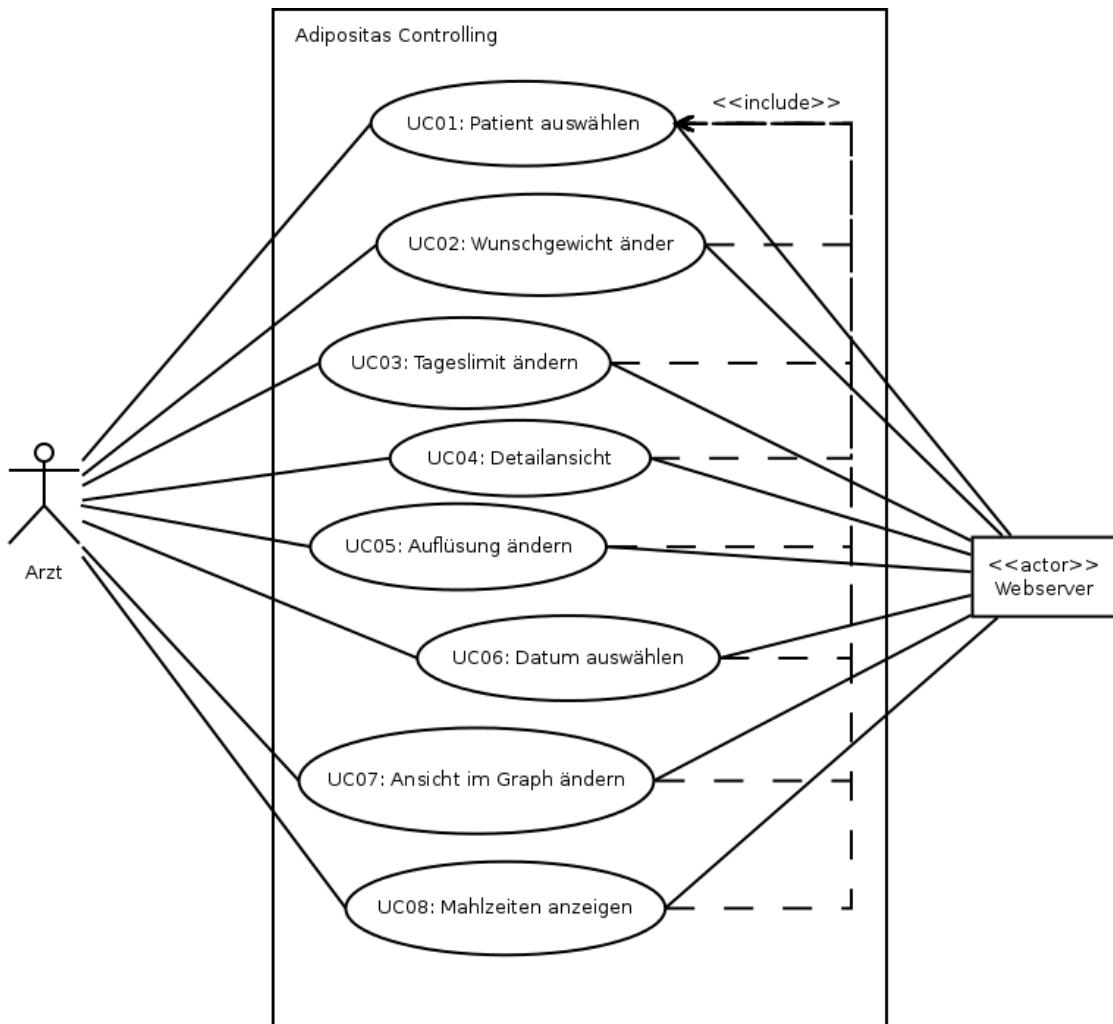


Abbildung 5.2: Use Case-Diagramm der Controlling-Software

### 5.5.2 Aktoren

- Arzt: Benutzer der Webanwendung
- Webserver: Webserver mit Service zum Auslesen der Tagesprotokolle aus der Datenbank.

### 5.5.3 Use Case Kurzbeschreibungen

- UC01: Patient auswählen  
Der Arzt wählt einen Patienten aus einer Liste aus.
- UC02: Wunschgewicht ändern

Der Arzt trägt ein neues Wunschgewicht für den Patienten ein.

- UC03: Tageslimit ändern

Der Arzt trägt eine Begrenzung an erlaubten Kilokalorien pro Tag für den Patienten ein.

- UC04: Detailansicht

Der Arzt wechselt zur Detailansicht eines Patienten.

- UC05: Auflösung ändern

Der Arzt ändert die zeitliche Auflösung der Graphen.

- UC07: Ansicht im Graphen ändern

Der Arzt ändert die Ansicht von Informationen, welche im Graphen angezeigt werden.

- UC08: Mahlzeiten anzeigen

Der Arzt wechselt zur Listenansicht von Mahlzeiten des Patienten.

## 5.6 Fundament der Anwendung

Das Fundament für die grafische Oberfläche der Software befindet sich im Webdokument *index.html*. Es gibt die Struktur der Webanwendung vor. Zusätzlich werden hier auch alle notwendigen Skripte und Stylesheets im *header* und *body* der *index.html*-Datei eingebunden. Neben der Struktur befinden sich bereits einige Buttons für die Navigation im Dokument. Die Hauptinhalte der Websoftware werden erst zur Laufzeit dynamisch mit JavaScript über das DOM-Konzept erzeugt und in das Dokument nachgeladen. Die Abbildung 5.3 veranschaulicht mit einem Wireframe das Layout der Anwendung.

Im *header*-Bereich befindet sich der Titel der Anwendung (*Adipositas-Controlling*) und ein Logo. Der Bereich *head-navigation* beinhaltet Buttons für die Datumsnavigation und einen Button für die Rückkehr zur Startseite. Im *main*-Bereich befinden sich zwei Container. Der erste wird als *draw-holder* und der zweite als *info-content* bezeichnet. In beiden Containern werden Informationen zu einem Patienten dargestellt. Dabei werden diese Informationen im Container *draw-holder* als Graph oder Liste präsentiert. Im Container *info-content* werden die Informationen in Textform dargestellt. Unten im Layout der Websoftware befindet sich der Container *footer*. Hier sind Buttons vorhanden, die zum Filtern der Informationen im Graphen dienen. Nach dem Start der Software sind zwar bereits alle Buttons im Dokument enthalten, werden aber mittels CSS ausgeblendet. Erst wenn der Benutzer einen Patienten aus einer Liste auswählt, werden die Inhalte in die Webseite dynamisch eingebaut und die nötigen Schaltflächen ein und ausgeblendet.

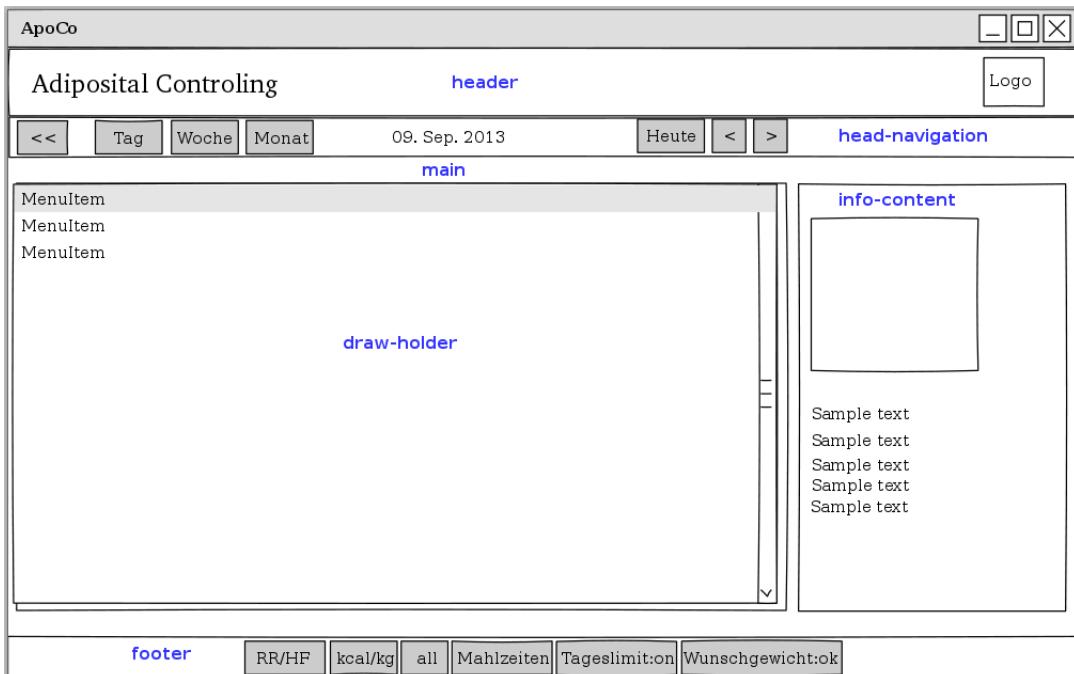


Abbildung 5.3: Wireframe für die Struktur der Webanwendung

## 5.7 Webanwendung-GUI Gestaltung

Die Abbildung 5.4 veranschaulicht die Startseite der Webanwendung. Hier werden alle Patienten in einer Liste dargestellt. Diese Liste ist in die folgenden fünf Spalten aufgeteilt.

- **Patient:**  
Hier steht der Name des Patienten.
- **Kalorienzunahme:**  
In dieser Spalte wird die Menge der Kilokalorien angezeigt, welche der Patient am aktuellen Tag bereits eingenommen hat. Ist der Hintergrund *grün*, so ist die eingenommene Menge unterhalb des vorgegebenen Limits. Beim Überschreiten der erlaubten Menge wird der Hintergrund *rot* angezeigt.
- **RR/HF:**  
Hier wird der Blutdruck als Durchschnittswert der letzten vier Tage angezeigt. Je nach Klassifikation wird der Hintergrund *grün*, *gelb* oder *rot* gefärbt. Die Klassifikation der Blutdruckbereiche richtet sich nach der Weltgesundheitsorganisation (WHO)[WHO13] und wurde so in die Software übernommen:

Optimaler Blutdruck: systolisch < 120, diastolisch < 80.

Normaler Blutdruck: systolisch < 130, diastolisch < 85.

Hochnormaler Blutdruck: systolisch < 140, diastolisch < 90.

Hypertonie Grad 1: systolisch < 160, diastolisch < 100.

Hypertonie Grad 2: systolisch < 180, diastolisch < 110.

Hypertonie Grad 3: systolisch > 179, diastolisch > 109.

Isolierte systolische Hypertonie: systolisch > 140, diastolisch < 90.

- Gewicht:

An dieser Stelle wird das zuletzt gemessene Körpergewicht des Patienten angezeigt.

Der Hintergrund wird je nach Abweichung vom Wunschgewicht in unterschiedlichen Färbungen dargestellt:

Grün: Abweichung < 10%.

Gelb: Abweichung > 10%.

weiches Rot: Abweichung > 20%.

dunkles Rot: Abweichung > 30%.

grelles Rot: Abweichung > 50%.

- Auswahl:

In dieser Spalte befindet sich ein Button, mit dem ein Patient ausgewählt wird. Nach dem Drücken wird im Bereich *p1* eine Patientenvorschau eingeblendet.

The screenshot shows a web-based application for managing patient data. The main interface displays a table of patients with columns for Name, Calorie intake, Blood Pressure (RR/HF), Weight, and Selection. A tooltip 'p1' is shown over the selection button for the first patient, 'dawid janas'. A detailed view window is open for this patient, showing personal information like Vorname (dawid), Nachname (janas), Email (@), and various weight-related metrics. The 'Detailansicht p1' button is highlighted at the bottom of the detail window.

Patient	Kalorienzunahme	RR / HF	Gewicht	Auswahl
dawid janas	381	130 / 72	87.6	> p1
Tom Jerry	...	...	65	>
patrik patient	...	...	...	>
bob bobwich	...	...	...	>
thomas tannenberger	...	...	...	>
julia hummel	...	...	...	>
manuela ela	...	...	...	>
nick mitnick	...	...	...	>
John Doe	...	...	...	>
Jane Doe	...	...	...	>
Eugen Fischer	...	...	...	>
Helene Fischer	...	...	...	>
Peter kreuzberg	...	...	...	>
Andrea elch	...	...	...	>

Vorname: dawid  
 Nachname: janas  
 Email: @  
 Körpergewicht: 87.6 kg  
 Wunschgewicht: 72 kg  
 Tageslimit: 500 kcal  
 RR / HF 4-Tage Schnitt  
 RR: 130 / 72 mmHg  
 HF: 57 bpm  
 Detailansicht p1

Abbildung 5.4: Startseite der Webanwendung

In der Vorschau  $p1$  sind detaillierte Informationen zum Patienten und drei weitere Buttons enthalten. Der Button mit der Bezeichnung *ändern* und mit der Verlinkung auf  $z1$ , öffnet einen modalen Dialog zum Ändern des gewünschten Körpergewichts für den Patienten. Der zweite Button mit der Bezeichnung *ändern* und der Verlinkung auf  $t1$ , öffnet hingegen einen modalen Dialog zum Ändern der maximal erlaubten Kilokalorienmenge für den Patienten. Beide Dialoge sind immer aus dem Bereich  $p1$  erreichbar, wenn dieser eingeblendet ist. In der Abbildung 5.5 werden beide Dialoge veranschaulicht.

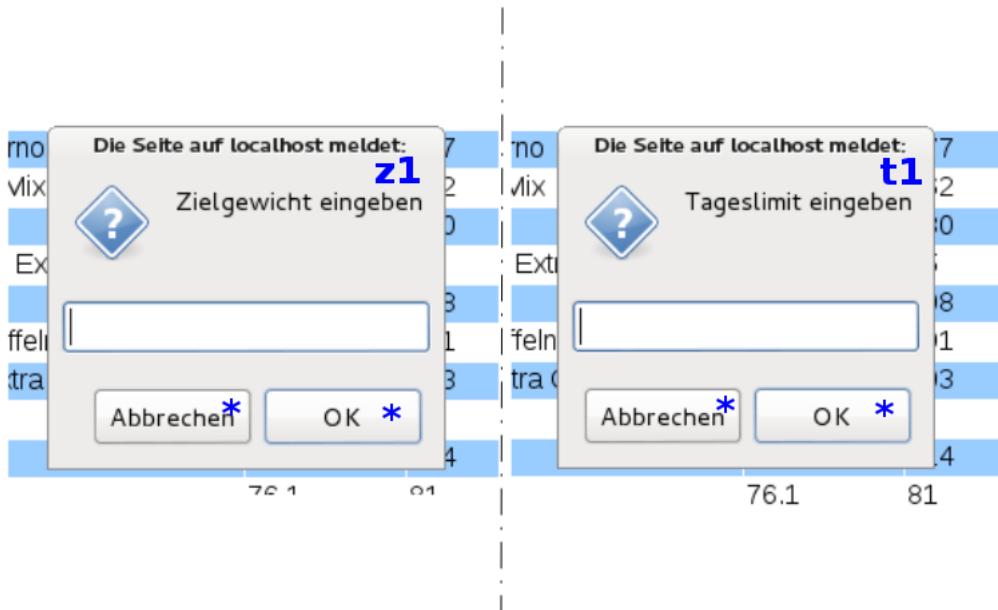


Abbildung 5.5: modale Dialoge zum Ändern vom Zielgewicht und Tageslimit für einen Patienten

In der Abbildung 5.6 und 5.7 wird die Webanwendungsview  $d1$  dargestellt. Hier gelangt der Anwender beim Drücken des Buttons *Detailansicht*. In dieser View wird im Bereich  $p1$  zusätzlich ein Kalender-Widget eingeblendet. Im Container *draw-holder* werden Messwerte eines Patienten als Graph  $g1$  oder Liste  $m1$  von protokollierten Mahlzeiten abgebildet. Das Kalender-Widget dient hier als Steuerelement für die abgebildeten Werte. Der Kalender und die Buttons in der *head-navigation* mit der Verlinkung auf  $ca/g1/m1$ , nehmen Einfluss auf den Zeitraum, für welchen die Daten angezeigt werden sollen.

Die Abbildung 5.8 präsentiert ein Zustandsdiagramm der Websoftware. Es verdeutlicht die Beziehungen von Webelementen in der Anwendung untereinander. Die Modellierung entspricht dem Zustandsdiagramm aus Kapitel 4 in der Abbildung 4.21. Die Zustände  $s1$ ,  $z1$ ,  $t1$  und  $d1$  sind mehr globale Zustände der Anwendung. Sie repräsentieren jeweils eine Art

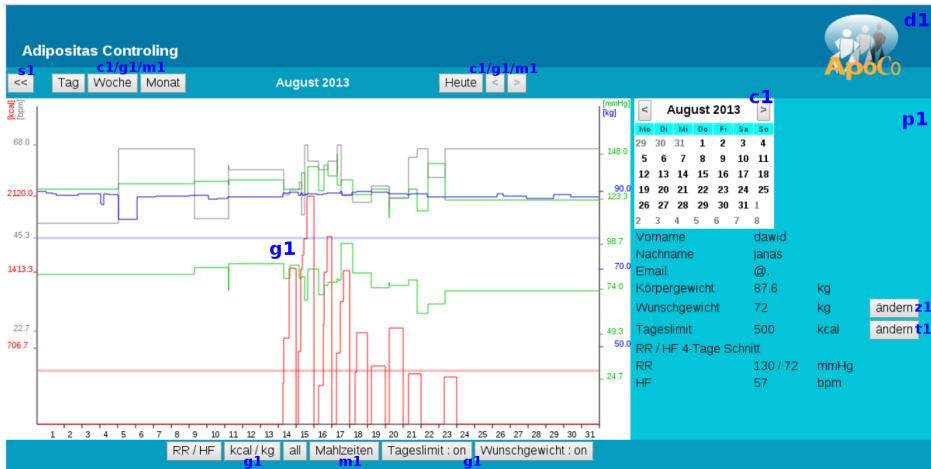


Abbildung 5.6: Tagesprotokolle in der Graphansicht

Datum	Markenname	Produkt	Gewicht	Energie
Mittwoch 14 August				
12:27 Uhr	Stork	Mini Dickmanns	76.6	326
12:27 Uhr	Kühne	Mixed Pickles	157.9	52
12:58 Uhr	Gut & Günstig	Haltbare Schlagsahne	144.3	424
20:26 Uhr	Griesson	Soft Cake Orange	144.1	585
20:26 Uhr	Kellogg's	FROSTIES	157.7	58
Donnerstag 15				
		<b>m1</b>		
08:37 Uhr	Knorr	Lasagne al forno	219.8	677
12:06 Uhr	Mondamin	Waffeln Teig-Mix	206	762
14:16 Uhr	HARIBO	Goldbären	37.9	130
16:25 Uhr	Zents	Diat Konfitüre Extra Pflaume	37.9	45
16:25 Uhr	PEMA	REIS-BROT	51.6	108
20:25 Uhr	Pfann	Country Kartoffeln	257.5	291
20:25 Uhr	Uncle Ben's	Süß-Sauer Extra Gemüse	114.1	103
Freitag 16 August				
11:30 Uhr	Müller's Mühle	Perlsago	62.5	214
11:30 Uhr	Herta	Saftschinken	76.1	81

Abbildung 5.7: Mahlzeitenprotokoll

View. Ist die Anwendung im Zustand *d1*, so beinhaltet sie ein Webelement mit zwei eigenen Zuständen *g1* und *m1*. Diese repräsentieren den Inhalt des Containers *draw-holder*. Im Zustand *g1* wird ein Graph gezeichnet und im Zustand *m1* eine Liste.

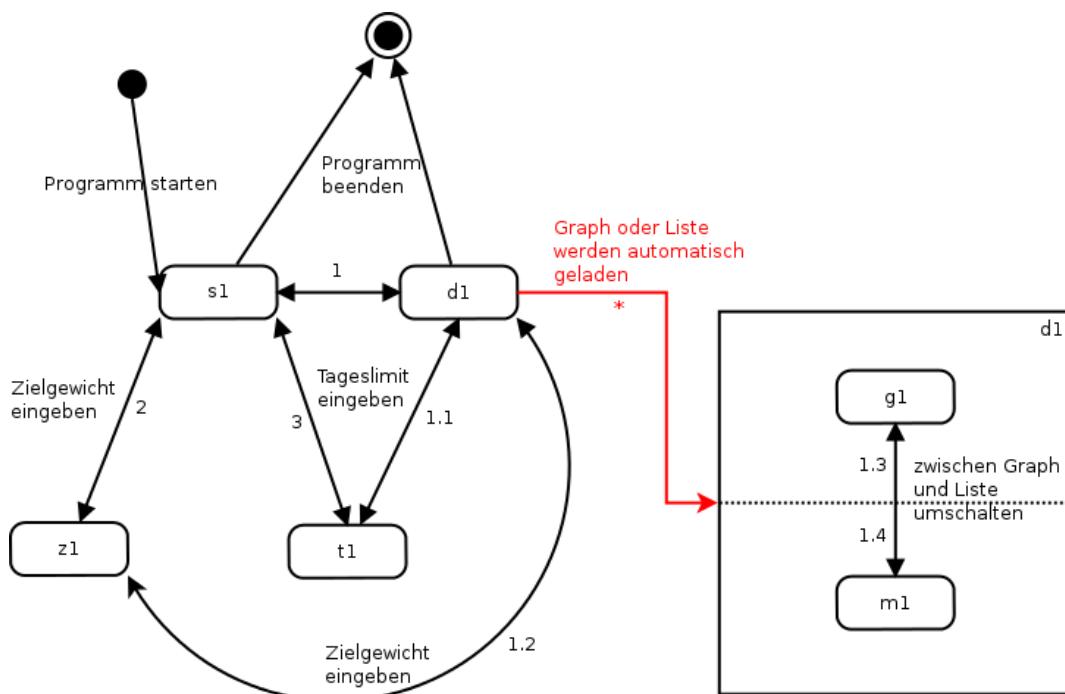


Abbildung 5.8: Zustandsdiagramm der Websoftware

## 6 Zusammenfassung / Ausblick

Während der Praxisarbeit und der Bachelorthesis wurde ein vielschichtiges Softwaresystem entwickelt, welches gleich mehrere bekannte und wichtige Architekturen, wie MVC und Client-Server und Technologien, wie zum Beispiel Ajax vereint. Für die Erweiterung des Smartphones um Messsensoren, wie die Körperwaage oder das Blutdruckmessgerät wurde die Bluetooth-Schnittstelle verwendet. Die Datenübertragung von Smartphone an einem Webserver geschieht unter dem Einsatz der REST-Architektur und die Messwerte werden auf einen MySQL-Datenbankserver persistent gespeichert. Die Benutzerschnittstelle der Webanwendung ist mit Ajax realisiert, was für mehr Interaktivität und eine bessere User Experience sorgt, da die Software aktiv auf den Benutzer reagiert. Dadurch bekommt der Benutzer das Gefühl die Anwendung würde wie eine Desktop-Anwendung direkt auf seinem Rechner ausgeführt werden. Dabei muss die Software nicht auf einem Rechner installiert werden, sondern hier reicht schon ein Ajax-fähiger Webbrower. Somit besteht auch nicht die Gefahr, dass die Software durch Viren geschädigt werden kann. Die Android-Anwendung benutzt die Kamera auf der Rückseite des Smartphones als Barcodescanner. Damit wird es möglich auf eine bequeme Art und Weise Lebensmittel zu identifizieren, ohne, dass man mühsam die EAN-Nummer oder sogar die Lebensmitteleigenschaften manuell eingeben muss. Die Daten werden über das Internet von einem Datenbankserver abgefragt und verwendet. Da die Messsensoren in die Software über Bluetooth-Technologie eingebunden sind, spart sich der Benutzer das Ablesen und Eingeben der Werte mit der Hand. Die Daten werden sicher über Bluetooth übertragen und in der Anwendung verwaltet. Die Software ist an Menschen mit einer gesundheitlichen Einschränkung gerichtet und durch die erwähnten Fähigkeiten hat sie eine unterstützende Wirkung für den Benutzer. Sein Aufwand wird durch den Einsatz der erwähnten Technologien minimiert, was ihn somit nicht zusätzlich belastet. Dadurch, dass die Tagesprotokolle über das Internet an den behandelnden Arzt geschickt werden, verringert sich der Stressfaktor für den Patienten, da er nicht zu jeder Untersuchung einreisen muss. Auf der anderen Seite wird auch der Aufwand für den Arzt minimiert, der in besonders schweren Fällen auch öfters einen Hausbesuch machen muss. Durch die Übersichtsfunktion und farbliche Untermalung der Messwerte bei Gefahren, wird der Arzt auf die Patienten, die seine Betreuung im besonderen Maße erfordern, aufmerksam gemacht. Die Software steigert somit auch die Effizienz der Betreuung, da der Arzt sich mehr auf hochgradig relevante Fälle konzentrieren kann. Die verschiedenen Messsensoren

führen zu einer aussagekräftigeren Diagnose. Sollte der Patient sich nicht an die Vorgaben des Arztes halten und es versäumen seine Mahlzeiten gründlich zu protokollieren, so würden andere Messwerte über sein Körpergewicht und seinen Blutdruck darauf hinweisen. Es ist dennoch von großer Bedeutung, dass die Zusammenarbeit zwischen Arzt und Patient stimmig ist. Der Patient muss diszipliniert mit der Protokollierung umgehen und ehrlich sein, damit die Messwerte eine korrekte Aussage nicht verwischen. Mit der ambulanten Betreuung durch den Arzt und der Protokollierung hat der Patient eine bessere Übersicht und kann sich selbst besser kontrollieren, da er darauf aufmerksam gemacht wird, wenn er zum Beispiel zu viel Nahrung zu sich nehmen sollte. Die Bestätigung, dass sein Gesundheitszustand sich nach und nach verbessert, wirkt auf den Patienten mit Sicherheit auch positiv und es fördert seine Genesung auf psychischer Ebene, da er Resultate protokolliert vor sich hat. Die Arbeit hat gezeigt, dass ein handelsübliches Smartphone, ausgestattet mit den üblichen Standardschnittstellen, in ein sehr umfangreiches und seriöses Projekt, welches über die Smartphone-Grenzen hinaus reicht, integriert werden kann. Nach Aussagen des Marktforschungsunternehmens *International Data Corporation* (IDC)[And13], ist Android mit 79,3% das führende Smartphonebetriebssystem weltweit. Die Entscheidung, eine Software für Android zu implementieren, war deshalb richtig. Auf diese Weise wird eine breite Masse der Smartphonebenutzer erreicht. Während der Bachelorarbeit entstanden einige Entwicklungsprobleme zwischen der Bodytel WeightTel Körperwaage und der Android-Anwendung. Im Projekt wurde als Entwicklungsgerät ein *Samsung Galaxy S3* genutzt. Das Smartphone wurde schon zu Beginn mit der damals neusten Android-Version 4.1 aktualisiert. Es stellte sich heraus, dass die Körperwaage nicht mit dieser Android-Version zusammenarbeiten kann. Nachforschungen haben ergeben, dass bis zur Android-Version 4.04, die Firma Google für die Bluetoothfunktionalität die freie Software *BlueZ* [Blu13c] verwendet hat. *BlueZ* ist unter Linux die Standardimplementierung der Protokolle für den Bluetooth Stack. Die höheren Android-Versionen verwenden die Implementierung von *Broadcom* [Bro13]. Der Broadcom Stack soll für Android speziell optimiert worden sein. Leider gibt es Geräte, wie die WeightTel Körperwaage, die damit nicht kompatibel sind. Zur Lösung des Problems im Projekt wurde ein Downgrade auf Android 4.04 vollzogen. Dies wurde auch vom Hersteller Bodytel empfohlen. Da die Messgeräte auch in Zukunft verwendet werden sollten, muss eine andere Lösung entworfen werden. Hier ergeben sich mehrere Ansätze. Die einfachste Möglichkeit wäre, dem Benutzer eine manuelle Messdateneingabe anzubieten. Dieser würde die Messwerte ablesen und sie in der Anwendung eintragen. Eine weitere Möglichkeit ist eine Art Homeserver zu entwickeln. Dieser Homeserver kann zum Beispiel auf einem Minicomputer, wie dem *Raspberry Pi* [ras13] basieren. Der *Raspberry Pi* würde mit Linux arbeiten, wobei er als Basis bereits den BlueZ Stack nutzt. Der Homeserver wäre eine zentrale Stelle, um die Messgeräte zu erfassen und würde anschließend die Messwerte an das Smartphone über Bluetooth senden. Ein dritter Vorschlag wäre, einen eigenen Bluetooth Stack unter Verwen-

dung der BlueZ-Bibliotheken zu entwickeln. Der Stack würde von der Android-Anwendung benutzt werden. Dabei könnte er nach Außen mit allen Messgeräten kommunizieren und die Kommunikation an die Android-Anwendung weitergeben. Intern würde die Anwendung weiterhin mit dem Bluetooth-Modul des Gerätes arbeiten. Beim Ausblick in die Zukunft könnte die Software mit weiterer Funktionalität und Messsensoren ausgebaut werden. Zum Beispiel durch das Erfassen von Blutzuckerwerten oder als Langzeit-EKG. Hierbei sollte eine Bedarfsanalyse durchgeführt und ein Konzept für eine umfangreiche Plattform zur Förderung der Gesundheit ausgearbeitet werden.



# Literaturverzeichnis

- [AM11] Arno, Becker and Marcus, Pant, *Android 2, Grundlagen und Programmierung*, Addison-Wesley, 2011.
- [And13] *Apple Cedes Market Share in Smartphone Operating System Market as Android Surges and Windows Phone Gains, According to IDC*, Website, 2013, <http://www.idc.com/getdoc.jsp?containerId=prUS24257413>; Besucht am 10. September 2013.
- [AS07] Huang Albert S., *Bluetooth Essentials for Programmers*, Cambridge University Press, 2007.
- [AT-13] *AT-Befehlssatz*, Website, 2013, <http://de.wikipedia.org/wiki/AT-Befehlssatz>; Besucht am 04. September 2013.
- [Blu02] *Bluetooth wird IEEE-Standard*, Website, 2002, [http://www.tecchannel.de/netzwerk/networkworld/technologyupdate/402832/bluetooth\\_wird\\_ieee\\_standard/](http://www.tecchannel.de/netzwerk/networkworld/technologyupdate/402832/bluetooth_wird_ieee_standard/); Besucht am 21. August 2013.
- [Blu13a] *Bluetooth*, Website, 2013, <http://www.bluetooth.com/Pages/basics.aspx>; Besucht am 21. August 2013.
- [Blu13b] *Bluetooth Special Interest Group (SIG)*, Website, 2013, <https://www.bluetooth.org/en-us>; Besucht am 21. August 2013.
- [Blu13c] *BlueZ never be supported from Android4.2*, Website, 2013, [http://idea2well.com/index.php?option=com\\_content&view=article&id=62:bluez-idea2well&catid=37:idea2well-android&Itemid=44](http://idea2well.com/index.php?option=com_content&view=article&id=62:bluez-idea2well&catid=37:idea2well-android&Itemid=44); Besucht am 10. September 2013.
- [Bro13] *Official Linux Bluetooth protocol stack*, Website, 2013, <http://www.bluez.org/>; Besucht am 10. September 2013.
- [Chr07] Wenz Christian, *JavaScript und AJAX, Das umfassende Handbuch (Auflage 7)*, Gallileo Press, 2007.
- [Chr11] ———, *JavaScript, Das umfassende Training*, DVD, 2011.
- [CSS13] *Cascading Style Sheets*, Website, 2013, [http://de.wikipedia.org/wiki/Cascading\\_Style\\_Sheets](http://de.wikipedia.org/wiki/Cascading_Style_Sheets); Besucht am 23. August 2013.
- [EAN13a] *European Article Number*, Website, 2013, [http://de.wikipedia.org/wiki/European\\_Article\\_Number#EAN-Strichcode\\_zur\\_Codierung\\_der\\_GTIN\\_.28ehem.\\_EAN-13.29](http://de.wikipedia.org/wiki/European_Article_Number#EAN-Strichcode_zur_Codierung_der_GTIN_.28ehem._EAN-13.29); Besucht am 22. August 2013.

- [EAN13b] *Produkte  $\tilde{A}^1_4$ berall finden*, Website, 2013, <http://ean-suche.org/ean-code-suche/>; Besucht am 22. August 2013.
- [EAN13c] *ZXing (Zebra Crossing)*, Website, 2013, <https://code.google.com/p/zxing/>; Besucht am 05. September 2013.
- [ETSI12] *ETSI*, Website, 2012, <http://www.etsi.org/>; Besucht am 03. September 2013.
- [FDA13] *Food and Drug Administration*, Website, 2013, [http://de.wikipedia.org/wiki/Food\\_and\\_Drug\\_Administration](http://de.wikipedia.org/wiki/Food_and_Drug_Administration); Besucht am 10. September 2013.
- [HTM13] *Hypertext Markup Language*, Website, 2013, [http://de.wikipedia.org/wiki/Hypertext\\_Markup\\_Language](http://de.wikipedia.org/wiki/Hypertext_Markup_Language); Besucht am 23. August 2013.
- [JK08] James F., Kurose and Keith W., Ross, *Computernetzwerke Der Top-Down-Ansatz (Auflage 4)*, Addison-Wesley, 2008.
- [Jö10] Schwenk Jörg, *Sicherheit und Kryptographie im Internet (Auflage 3)*, Vieweg + Teubner Verlag, 2010.
- [JSO13] *JavaScript Object Notation*, Website, 2013, [http://de.wikipedia.org/wiki/JavaScript\\_Object\\_Notation](http://de.wikipedia.org/wiki/JavaScript_Object_Notation); Besucht am 26. August 2013.
- [KER13] *KERN PCB 6000*, Website, 2013, <http://www.kern-sohn.com/data/zusatzseiten/downloads/z-cb-de-kp-precision-balances.pdf>; Besucht am 22. August 2013.
- [med13] *Medizinprodukt Richtlinie*, Website, 2013, [http://de.wikipedia.org/wiki/Richtlinie\\_93/42/EWG\\_%C3%BCber\\_Medizinprodukte](http://de.wikipedia.org/wiki/Richtlinie_93/42/EWG_%C3%BCber_Medizinprodukte); Besucht am 22. August 2013.
- [Pic13] *Pico Plug*, Website, 2013, [http://www.bluetoothupgrades.de/pico\\_plug.htm](http://www.bluetoothupgrades.de/pico_plug.htm); Besucht am 22. August 2013.
- [Pre13] *PressureTel*, Website, 2013, <http://www.bodytel.com/pressuretel.html>; Besucht am 22. August 2013.
- [ras13] *Raspberry Pi*, Website, 2013, <http://www.raspberrypi.org/>; Besucht am 10. September 2013.
- [RES13] *Representational State Transfer*, Website, 2013, [http://de.wikipedia.org/wiki/Representational\\_State\\_Transfer](http://de.wikipedia.org/wiki/Representational_State_Transfer); Besucht am 23. August 2013.
- [RWD13] *Responsive Webdesign*, Website, 2013, [http://de.wikipedia.org/wiki/Responsive\\_Webdesign](http://de.wikipedia.org/wiki/Responsive_Webdesign); Besucht am 23. August 2013.
- [SG12] Stefan, Reimers and Gunnar, Thies, *PHP 5.4 & MySQL 5.5 (Auflage 4)*, Gallileo Press, 2012.
- [Sic13a] *Advanced Encryption Standard*, Website, 2013, [http://de.wikipedia.org/wiki/Advanced\\_Encryption\\_Standard](http://de.wikipedia.org/wiki/Advanced_Encryption_Standard); Besucht am 29. August 2013.
- [Sic13b] *The Legion of the Bouncy Castle*, Website, 2013, <http://www.bouncycastle.org/>; Besucht am 31. August 2013.
- [Sic13c] *Transport Layer Security*, Website, 2013, [http://de.wikipedia.org/wiki/Transport\\_Layer\\_Security](http://de.wikipedia.org/wiki/Transport_Layer_Security)

- Transport\_Layer\_Security; Besucht am 29. August 2013.
- [Wei13] WeightTel, Website, 2013, <http://www.bodytel.com/weighttel.html>; Besucht am 22. August 2013.
- [WHO13] World Health Organization, Website, 2013, <http://www.who.int/en/>; Besucht am 11. September 2013.
- [XML13a] Extensible Markup Language, Website, 2013, [http://de.wikipedia.org/wiki/Extensible\\_Markup\\_Language](http://de.wikipedia.org/wiki/Extensible_Markup_Language); Besucht am 25. August 2013.
- [XML13b] Websiteentwicklung: XML: Aufbau eines XML-Dokumentes, Website, 2013, [http://de.wikibooks.org/wiki/Websiteentwicklung:\\_XML:\\_Aufbau\\_eines\\_XML-Dokumentes](http://de.wikibooks.org/wiki/Websiteentwicklung:_XML:_Aufbau_eines_XML-Dokumentes); Besucht am 26. August 2013.



# Abbildungsverzeichnis

2.1	KERN PCB 6000 Laborwaage . . . . .	12
2.2	KERN PCB 6000 Laborwaage, Rückseite mit RS-232-Schnittstelle . . . . .	12
2.3	Pico Plug Bluetooth-Dongle . . . . .	13
2.4	Bodytel WeightTel Körperwaage . . . . .	14
2.5	Bodytel WeightTel Körperwaage . . . . .	15
2.6	Beispiel für ein EAN Produkt Code . . . . .	16
2.7	Beispiel einer möglichen REST-Anfrage . . . . .	17
2.8	Das Ergebnis im Browser aus dem Listing 2.3 . . . . .	21
2.9	Das Ergebnis des Listing 2.3 als DOM-Struktur im Webbrowser Chrome . .	22
2.10	Das Ergebnis im Browser aus dem Listing 2.4 . . . . .	23
2.11	Der DOM-Baum nach Zugriff über JavaScript . . . . .	24
3.1	MD5 Hash aus dem String "Dawid Janas" . . . . .	28
3.2	Sicherheitskonzept als Diagramm . . . . .	32
4.1	MVC-Architektur der ApoCo-Anwendung . . . . .	33
4.2	Client-Server-Architektur des gesamten Projekts . . . . .	34
4.3	Projekt-Paketstruktur der ApoCo-Anwendung . . . . .	38
4.4	ApoCo Datenbankdiagramm . . . . .	53
4.5	Sequenzdiagramm für Bluetoothverbindung und Datenaustausch . . . . .	54
4.6	Klassendiagramm für Dekodieren von Nachrichten . . . . .	55
4.7	Sequenzdiagramm für Dekodieren von Nachrichten . . . . .	56
4.8	Klassendiagramm, Visualisierung und Speicherung der Daten . . . . .	57
4.9	Sequenzdiagramm, Visualisierung und Speicherung der Daten . . . . .	57
4.10	Sequenzdiagramm für das Senden der Tagesprotokolle an den Webserver . .	59
4.11	ApoCo Use Case-Diagramm . . . . .	65
4.12	Start-Activity als Beispiel für Strukturierung . . . . .	66
4.13	Header einer Activity mit <i>Back-Button</i> . . . . .	67
4.14	Zurück-Möglichkeiten einer Activity . . . . .	68
4.15	Zurück mit einem dem Hardware-Button des Smartphones . . . . .	68
4.16	Geräteverwaltung . . . . .	69

4.17 Koppelungsactivity . . . . .	69
4.18 Serverkonfiguration . . . . .	70
4.19 Übergangsanimation . . . . .	71
4.20 Darstellung aller Activities . . . . .	73
4.21 Zustandsdiagramm für Beziehungen der gegenseitigen Aufrufe . . . . .	74
5.1 Calendar-Widget . . . . .	80
5.2 Use Case-Diagramm der Controlling-Software . . . . .	84
5.3 Wireframe für die Struktur der Webanwendung . . . . .	86
5.4 Startseite der Webanwendung . . . . .	87
5.5 modale Dialoge zum Ändern vom Zielgewicht und Tageslimit für einen Patienten . . . . .	88
5.6 Tagesprotokolle in der Graphansicht . . . . .	89
5.7 Mahlzeitenprotokoll . . . . .	89
5.8 Zustandsdiagramm der Websoftware . . . . .	90

# Listings

2.1	createXHRObject() Funktion in JavaScript . . . . .	18
2.2	Verbindungsauflaufbau und Callback-Funktion bei Ajax . . . . .	19
2.3	Beispiel für ein einfaches HTML-Dokument . . . . .	21
2.4	Beispiel für dynamischen Zugriff auf den DOM-Baum . . . . .	22
2.5	Eine JavaScript Datei eingebunden in ein HTML-Dokument . . . . .	24
2.6	Beispiel für eine Datenstruktur im XML . . . . .	25
2.7	Beispiel aus Listing 2.6 umgesetzt in JSON . . . . .	26
3.1	Beispiel für einen Zugriff über eine Zugriffs-API . . . . .	30
3.2	Beispiel für Behandlung nach einem Aufruf der Web-API . . . . .	30
4.1	Beispiel für reduzierte Implementierung der <i>DBManagerLocal</i> Klasse . . . . .	47
4.2	Die Schnittstelle <i>UserColumns</i> . . . . .	47
4.3	Die Klasse <i>UserTbl</i> . . . . .	48
4.4	<i>DBManagerLocal</i> erzeugt die Tabelle <i>user</i> . . . . .	48
4.5	<i>UserDTO</i> -Klasse in leicht reduzierter Form . . . . .	49
4.6	<i>UserDTO</i> -Klasse in leicht reduzierter Form . . . . .	50
4.7	Methode <i>getUserByNachname()</i> der <i>DBManagerLocal</i> -Klasse . . . . .	50
4.8	Methode <i>getUserByNachname()</i> der <i>DBManagerLocal</i> -Klasse . . . . .	51
4.9	SynchronizeBloodpressure sendet Blutdruckmesswerte an den Webserver . . . . .	58
4.10	Verbindungsauflaufbau zur Datenbank . . . . .	60
4.11	Benutzer registrieren, Schritt 1 . . . . .	61
4.12	Benutzer registrieren, Schritt 2 . . . . .	61
4.13	Benutzer registrieren, Schritt 3 . . . . .	62
4.14	ApoCo-Manifest, Integration von ZXing . . . . .	63
4.15	Barcodesuche starten . . . . .	63
4.16	Ergebnis der Suche abfangen . . . . .	63
4.17	Activity als Dialogfenster erscheinen lassen . . . . .	69
4.18	Animation zum Verlassen einer Activity . . . . .	70
5.1	Beispiel für <i>AjaxManager</i> -Implementierung . . . . .	76
5.2	Beispiel für eine Methode zum Senden über Ajax . . . . .	76

5.3	handleResponse Funktion zum Abfangen von Serverantworten . . . . .	77
5.4	handleResponse Funktion zum Abfangen von Serverantworten . . . . .	78
5.5	Erzeugen von Zellen im Kalender per DOM . . . . .	82
5.6	Ablauf der redrawCalendarWidget-Funktion . . . . .	83

# A Anhang

## A.1 Glossar

**IEEE** Institute of Electrical and Electronics Engineers ist ein Berufsverband von Ingenieuren für die Normierung von Techniken, Hardware und Software.

**SIG** Bluetooth Special Interest Group ist ein Zusammenschluss von mehreren Firmen, die an Bluetooth zusammenarbeiten und weiterentwickeln.

**WLAN** drahtloses lokales Netzwerk, bei dem die Rechnerkommunikation über Funk abgewickelt wird.

**UI** das User Interface ist die Schnittstelle zwischen Computer und Mensch.

**User Experience** das Anwendererlebnis beschreibt alle Aspekte der Erfahrungen eines Anwenders bei der Interaktion mit einem Produkt.

**ISM-Band** Industrial, Scientific and Medical Band ist ein Frequenzbereich für Geräte, die in der Industrie, Wissenschaft, Medizin und in häuslichen oder ähnlichen Bereichen genutzt werden können.

**Gateway** dient zur Übermittlung von Nachrichten zwischen mehreren Netzen.

**FDA** Food and Drug Administration kontrolliert unter anderem Medizinprodukte in den USA und hat die Aufgabe die öffentliche Gesundheit zu schützen.

## A.2 CD-Inhalt

- BA\_858168\_DawidJanas.pdf
- Bachelorarbeit Latex-Dateien
- Programmcode der Android-Anwendung im Verzeichnis: android

- Programmcode der Webanwendung im Verzeichnis: ajax
- SQL-Datenbankschema im Verzeichnis: **sql\_db**
- APK-Datei der Android-Anwendung im Verzeichnis: apk
- Sicherung der online Quellenangaben im Verzeichnis: pdf