HOCHSCHULE BREMEN
UNIVERSITY OF APPLIED SCIENCES

Hochschule Bremen, Fachbereich Informatik

Bachelor-Thesis

Zur Erlangung des akademischen Grades Bachelor of Science (B.Sc.)

Im Studiengang Technische Informatik, Fachrichtung: Angewandte Informatik

Thema:

Kunden-Datenbank mit Android™-Client

Autor: Jan Stalhut
Matrikelnr.: 242085

Abgabe am: 12.12.2011
Prüfer: Prof. Dr. rer. nat. Ulrich Breymann
Korreferent: Prof. Dr.-Ing. Uwe Meyer

Zusammenfassung

Im Rahmen der vorliegenden Bachelor-Thesis wurde eine normalisierte Datenbank für die Daten eines in Form einer Excel-Tabelle existierenden Kunden-Stamms (ca. 400 Datensätze) entworfen und auf einem PostgreSQL-Server eingerichtet werden. Bei der Datenbank-Struktur waren dabei einige Besonderheiten zu beachten. So zum Beispiel, dass es zu einem Firmen-Kunden mehrere Ansprechpartner und Kunden mit mehreren Adressen (bedingt zum Beispiel durch abweichende Rechnungsadressen oder mehrere Filialen) gibt.

Zur komfortablen Pflege der Daten von einem PC wurde eine Weboberfläche programmiert, die sich im auch auf einem mobilen Endgerät mit kleinem Display bedienen lässt.

In einem weiteren Schritt wurde eine Anwendung (App) für das Smartphone-Betriebssystem Android™ entwickelt, mit dem der Zugriff auf die Daten aus der Kunden-Datenbank komfortabel ermöglicht wird. Es wird dort im Wesentlichen ein Eingabefeld zur Suche bereitgestellt, bei dem nach Eingabe weniger Zeichen entsprechende Vorschläge mit Übereinstimmungen gemacht werden. Bei hinterlegten Telefonnummern ist ein direktes Anrufen aus der App heraus möglich. Für Anschriften bot es sich an, diese auf Wunsch direkt an die auf Android™-Geräten obligatorische „Maps“ oder „Navigation“ Apps weiterzuleiten, oder auch andere installierte und kompatible Apps zur Auswahl anzubieten. E-Mail-Adressen können entsprechend an eine installierte (als Standard definierte) E-Mail-App zum Verfassen einer neuen E-Mail übergeben werden und eine zu einem Kunden hinterlegte Webseite lässt sich direkt im mobilen Standard-Webbrowser öffnen.

Zur Kommunikation zwischen dem Datenbank-Server und der Client-App sollte das kompakte Datenformat JSON zum Einsatz kommen und das SSL-verschlüsselte Protokoll HTTPS genutzt werden. Serverseitig wurde dazu als Schnittstelle zur Datenbank ein entsprechendes PHP-Script entwickelt, das sowohl von der Weboberfläche als auch dem Android™-Client genutzt werden kann. Ein besonderes Augenmerk bei der Kommunikation wurde auf die Datensicherheit und Authentifizierung gelegt, weil die Kundendaten aus Datenschutzgründen selbstverständlich nicht in fremde Hände geraten dürfen. Gleichzeitig sollte es, zumindest vom Android™-Client aus, auch ermöglicht werden, ohne jedes Mal ein kryptisches Passwort eingeben zu müssen, schnell auf Kundendaten zugreifen zu können. Dies wurde durch Benutzer-Sessions realisiert.

Inhaltsverzeichnis

1 Einleitung

Im Rahmen der vorliegenden Bachelor-Thesis soll eine Datenbank nach den Regeln der Normalisierung eingerichtet werden, auf die zum einen über eine Weboberfläche mit aktuell gängigen Webbrowsern lesend und schreibend zugegriffen werden kann. Zum anderen soll das auch für den mobilen Einsatz mit Hilfe einer entsprechenden Anwendung auf Smartphones mit Android™-System lesend möglich sein.

In diesem Rahmen werden verschiedene aktuelle Techniken bzw. Softwarekomponenten behandelt und zum Einsatz gebracht: HTML5 einschließlich JavaScript und aktuellem CSS3 für die Weboberfläche, das kompakte Datenformat JSON zur Kommunikation zwischen Client und Server, ein PostgreSQL 8.4 Datenbank-Server zur serverseitigen Datenhaltung und das Android™ SDK in Release 15 für die Entwicklung von Apps für Android-Systeme bis einschließlich Version 4.0 „Ice Cream Sandwich“.

1.1 Motivation

Mobile Plattformen sind derzeit in aller Munde, die Preise für Smartphones und insbesondere solchen mit dem Android™-Betriebssystem von Google sinken rapide. [1] [2] Inzwischen läuft auf mehr als der Hälfte der verkauften Smartphones das Google Betriebssystem. [3] Nur im Business-Bereich konnten sich die Geräte bisher noch nicht durchsetzen, was Google jetzt unter anderem mit dem neuen Android™ 4.0 „Ice Cream Sandwich“ (kurz ICS) ändern möchte, das unter anderem auch mit neuen Funktionen in dieser Richtung aufwartet. [4] So handelt es sich ja schließlich bei dem – im Rahmen dieser Bachelor-Thesis zu entwickelnden – Android-Client auch um eine Anwendung für den Geschäftsbereich.

Speziell richtet sich die geplante Anwendung an Außendienstmitarbeiter, die auch unterwegs einen einfachen Zugang zu den Daten der Kunden haben wollen, ohne gleich bspw. das Notebook hochfahren zu müssen oder in extra mitzunehmenden Zetteln suchen müssen, um die Telefonnummer oder Adresse des nächsten Kunden herauszufinden.

1.2 Hintergrund

Die Idee für diese Anwendung entstand bei meiner eigenen nebenberuflichen selbstständigen Arbeit im Bereich IT-Dienstleistungen, bei der unterwegs häufig das Problem auftrat, dass z. B. für den nächsten Kunden die Telefonnummer herausgesucht werden musste, um diesen um eine kurzfristig entstandene Verspätung zu informieren, oder nochmal schnell die Adresse ermittelt werden musste.

Ergänzend kommt hinzu, dass die bisherige Kundendatenbank von einer Excel-Tabelle (bzw. genauer gesagt einem LibreOffice Tabellendokument) repräsentiert wird. Diese ist inzwischen durch ihren Umfang von über 400 Kunden-Datensätzen mit teilweise mehreren Telefonnummern, Kontaktpersonen, Adressen und E-Mail-Adressen recht unübersichtlich geworden, sodass es längst überfällig war, eine bessere Verwaltungssoftware für diese Datenmengen einzurichten. Auf der Suche nach derartiger Software ist es jedoch schwer fündig zu werden, wenn man die genannten besonderen Ansprüche (mehrere Kontaktpersonen, Adressen etc. zu einem Kunden-Datensatz) voraussetzt und die Lösung auch noch möglichst günstig sein soll. So entstand das Bedürfnis nach einer Eigenentwicklung.

1.3 Vorgehen

Bei dem Vorgehen im Rahmen dieser Bachelor-Thesis wurde sich an dem allgemein üblichen Vorgehensmodellen für die Softwareentwicklung orientiert, wobei sich das hier eingesetzte am sogenannten Wasserfall-Modell orientiert:

- Zunächst wurden die konkreten technischen Anforderungen festgelegt, die im folgenden Kapitel 2 (Seite 3) wiederzufinden sind.
- Nach Festlegung der notwendigen Rahmenbedingungen in Bezug auf die Entwicklungsplattform bzw. -umgebung (Kapitel 3, Seite 8) begann im nächsten Schritt der Entwurf eines Software-Designs, siehe Kapitel 4 (Seite 13).
- Mit dem Festlegen der Datenbankstruktur und der JSON-Schnittstelle konnte dann bereits, wie in Kapitel 5 (Seite 20) beschrieben, die Server-Anwendung in Form eines PHP-Scripts entwickelt werden, das als Schnittstelle zwischen der PostgreSQL-Datenbank und den JSON-Anfragen/-Antworten fungiert und sich auch um die Authentifizierung der Benutzer kümmert.
- Im nächsten Schritt wurde zunächst die Weboberfläche bzw. der Webclient auf Basis von HTML, CSS und JavaScript entwickelt, der bereits mit dem Server kommuniziert und es so ermöglicht, Daten in die Datenbank einzupflegen, anzupassen und abzufragen. Als praktischer Nebeneffekt entstand dabei ohne nennenswert größeren Aufwand eine App für den Webbrowser Chrome, die sich in selbigen installieren lässt, sodass die eigentliche Weboberfläche dann offline in diesem Browser zur Verfügung steht. (Kapitel 6, Seite 28)
- Schließlich wurde die eigentliche App für Android™-basierte Systeme entwickelt, die ebenfalls über die definierte JSON-Schnittstelle mit dem Server kommuniziert, um Daten der Datenbank abzufragen (Kapitel 7, Seite 38). Diese Anwendung lässt sich neben Smartphones dann auch ohne weitere Probleme auf einem Android™-Tablet installieren.
- Abschließend fand eine Validierung des Ergebnisses gegen die ursprünglich festgelegten Anforderungen statt, ehe die Software in Betrieb genommen werden kann. (Kapitel 8, Seite 55)

1.4 Abgrenzung

Da es sich in dieser Bachelor-Thesis schwerpunktmäßig um die *Entwicklung* der Web- und Android™-Anwendung drehen soll, wird das Thema des Testens nicht im Einzelnen behandelt werden. Die Android™-Plattform bzw. dessen SDK von Google bieten zahlreiche Mechanismen um die eigene App komfortabel und automatisiert bspw. auf verschiedensten Emulatoren und auch auf per USB angeschlossenen Android™-Geräten entsprechend programmierten Tests (automatisiert) zu unterziehen. [5] Auch in Bezug auf die Weboberfläche gibt es verschiedenste interessante Methoden des automatisierten Testens (z. B. in Form von Add-Ons für verschiedene gängige Webbrowser). Diese Themen hätten jedoch den zeitlichen Rahmen dieser Bachelor-Thesis gesprengt, sie könnten vielmehr problemlos das alleinige Thema einer separaten Bachelor-Thesis werden. Es wird auf die Prozesse des Testens in dieser Thesis zumindest am Rande eingegangen, da es selbstverständlich Bestandteil eines jeden Softwareentwicklungsprozesses sein sollte.

2 Anforderungen

Im Folgenden sollen die konkreten technischen Anforderungen an das gesamte geplante Software-System ausgeführt werden, sodass am Ende das System auf diese Anforderungen getestet werden kann.

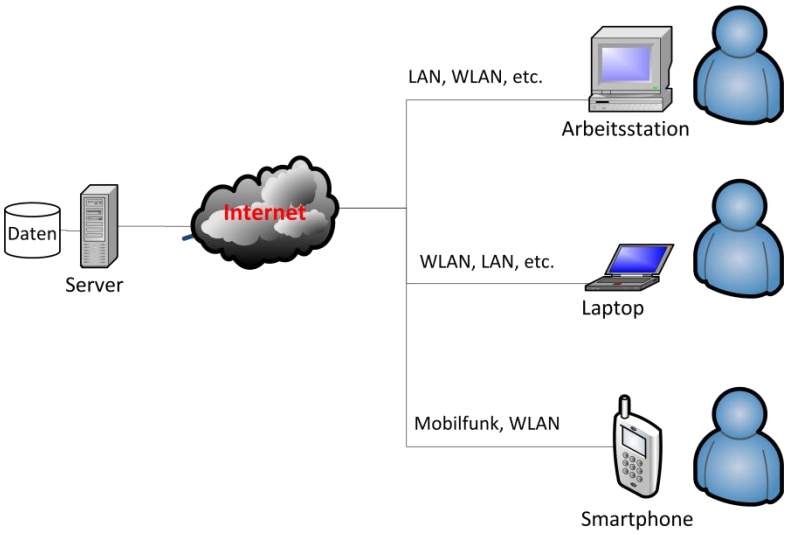

Abbildung 1: Gesamtkonzept der Anwendung

In Abbildung 1 ist die gewünschte Gesamtstruktur zu sehen. Es wird ersichtlich, dass es eine zentrale Serveranwendung mit Datenbank geben soll, auf die verschiedenartige Clients über das Internet zugreifen können. Bei diesen verschiedenartigen Clients ist zum einen der normale Desktop-Arbeitsplatz denkbar, als auch ein Laptop (Notebook), das sich per WLAN oder auch unterwegs per Mobilfunk mit dem Internet verbindet. Und schließlich sind da noch die Smartphones, die typischerweise per Mobilfunk eine Internet-Verbindung herstellen, und die dann ebenfalls Zugriff auf die Serveranwendung haben sollen.

2.1 Datenbank / Serveranwendung

Generell soll die Serverseite mehrere Instanzen der Software ermöglichen, die jeweils eine eigene Datenbank nutzen und ggf. auch unterschiedliche Versionen der zu entwickelnden Software einsetzen können.

Die Datenbank-Struktur soll es ermöglichen, dass es sowohl Kundendatensätze mit mehreren Ansprechpartnern (Personen), als auch Personen mit mehreren zugehörigen Kundendatensätzen geben kann. Einem Kunden können dabei (optional) ein oder auch mehrere (bis zu 10) Adressen sowie Webseiten zugeordnet werden. Den Personen können Telefonnummern, Faxnummern und Email-Adressen zugeordnet werden, auch hier wieder je Kunde null bis beliebig viele, wobei jeweils auch 10 als Obergrenze angesehen werden kann. Zu jeder Information (Kunde, Person, Adresse, Webseite, E-Mail-Adresse, Faxnummer, Telefonnummer) soll es optional möglich sein, einen Kommentar zu speichern, um weitere Informationen zu hinterlegen, die in kein anderes Feld passen.

2.1.1 Kunden

Jedem Kunden wird ein Typ zugeordnet, der festlegt, ob es sich um einen Privat- oder Geschäftskunden handelt. Wenn es sich um einen Geschäftskunden handelt, muss auch der Name des Unternehmens gespeichert werden können. Außerdem wird hinterlegt, auf welche Weise die Rechnung zugestellt werden soll (E-Mail, Post, Telefax) und welche Zahlungsweise mit dem Kunden vereinbart wurde (Überweisung, Barzahlung, PayPal).

Um eine später geplante Verknüpfung mit einer Rechnungsverwaltung zu ermöglichen, sollen die dafür relevanten Daten (Anschrift und Name) unveränderbar und nicht endgültig löschbar sein.

2.1.2 Fax- und Telefonnummern

Die Fax- und Telefonnummern werden in Ländervorwahl, Ortsvorwahl, Durchwahl und Hauptrufnummer unterteilt und ohne führende Nullen gespeichert, damit sie sich später beliebig formatiert anzeigen lassen und z. B. die Ländervorwahl für Deutschland nicht mit angezeigt werden muss und für die automatische Anwahl eine reine Zahlenfolge erzeugt werden kann.

2.1.3 Webseiten

Zu jeder Webseite wird neben der URL, bei der es sich sowohl um einen HTTP- als auch um einen HTTPS-Link handeln kann, noch eine Notiz hinterlegt.

2.1.4 Adressen

Bei der Adresse werden die wesentlichen Informationen: Straßennamen, Hausnummer, Postleitzahl, Ort und Land angegeben. Zusätzlich soll sich zur besseren Orientierung der Stadtteil speichern lassen und die (mit dem Kunden vereinbarte) Anfahrtspauschale hinterlegt werden.

2.1.5 Person

Jede Person (Kontaktperson) eines Kunden hat einen Nachnamen, eine Anrede (Herr oder Frau) und die bevorzugte Höflichkeitsform (duzen oder siezen). Optional können ein Vorname und beliebig viele Kontaktdaten in Form von E-Mail-Adressen, Telefonnummern und Faxnummern angegeben werden (von diesen reichen, wie erwähnt, jeweils max. 10 Einträge). Auch soll hinterlegt werden können, ob bei einem Anruf die eigene Rufnummer übermittelt oder unterdrückt werden soll.

2.2 Datenaustausch

Der Datenaustausch zwischen Serveranwendung und Clients soll SSL-verschlüsselt über das HTTP-Protokoll erfolgen (HTTPS). Dabei kann aus Kostengründen zunächst ein selbstsigniertes Zertifikat zum Einsatz kommen. Beim Datenaustausch sollte auf das kompakte JSON-Format gesetzt werden, für dessen Verarbeitung (Analysierung) die einzusetzenden Programmiersprachen (PHP, Java und JavaScript) fertige Bibliotheken zur Verfügung stellen, die leicht genutzt werden können. Des Weiteren ist dieses Format vom Datenvolumen kompakter als bspw. XML-Daten, was wiederum besonders für die mobile Anwendung - bei vergleichsweise teuren volumenbasierten Datentarifen – auch Rolle spielt.

Die zwischen Client und Server auszutauschenden Daten sollten möglichst gering gehalten werden, um zum einen die Serverlast niedrig zu halten und auch die notwendige Bandbreite auf Client- und Server-Seite auf das Notwendigste zu reduzieren. Dies ist insbesondere bei der Implementierung der Instant-Suche zu berücksichtigen.

2.3 Authentifizierung

Zur Authentifizierung sollen serverseitig Benutzer mit Benutzernamen und Passwort hinterlegt werden können. Zur Zuweisung verschiedener Benutzerrechte (zum Bearbeiten und Hinzufügen oder nur Einsicht von Kundendaten), sollen sich Benutzer in Gruppen zusammenfassen lassen. Eine Benutzergruppe wird zur Wiedererkennbarkeit mit einem eindeutigen Namen versehen. Zu einem Benutzer sollen der Zeitpunkt des letzten erfolgreichen Logins, der Zeitpunkt des letzten fehlgeschlagenen Logins sowie die Anzahl der fehlgeschlagenen Logins festgehalten werden, um so einen Login nach drei Fehlschlägen für 30 Minuten zu unterbinden. Zur Speicherung dieser Daten ist ebenfalls die PostgreSQL-Datenbank einzusetzen.

2.3.1 Sitzungsverwaltung

Auf dem Client, insbesondere bei der Android™-App, soll es ermöglicht werden, dass der Benutzer angemeldet bleiben kann, bis er sich explizit abmeldet (oder eine Inaktivität von 30 Tagen überschritten wird). Da hierbei aus Sicherheitsgründen nicht das Kennwort des Benutzers auf Client-Seite gespeichert werden sollte, ist ein Session-System vorzuziehen, bei dem jedes Endgerät einen automatisch generierten individuellen (u. a. auf der aktuellen Uhrzeit basierenden) Zugangsschlüssel nutzt um sich anzumelden. So sollte es dann auch von einem anderen Client aus möglich sein, die eigenen Sitzungen einzusehen und ggf. einzelne zu löschen, bspw. wenn ein Smartphone, auf dem noch eine aktive Session besteht, verloren geht und ein Zugriff auf die Kundendaten von jemandem, der das Gerät in die Hände bekommt, verhindert werden soll. Bei der Annahme einer gestohlenen Session sollte selbige umgehend deaktiviert werden, sodass sich der Benutzer beim nächsten Zugriff wieder mit seinem persönlichen Passwort anmelden muss.

2.3.2 Passwortspeicherung

Bei der Speicherung der Passwörter ist darauf zu achten, dass sich diese nicht im Klartext auslesen lassen. Sowohl die Zugangsschlüssel für die Benutzer-Sitzungen als auch die persönlichen Kennwörter sollten folglich mit einem Hashverfahren verschlüsselt gespeichert werden, damit sich diese nicht entschlüsseln lassen, sondern lediglich durch erneutes Verschlüsseln der Hash verglichen werden kann.

2.4 Webclient

Der Webclient sollte auf allen aktuellen Browser-Versionen lauffähig sein:

- Microsoft Internet Explorer 9 vom 15.03.2011 [6]
- Opera 11 vom 16.12.2010 [7]
- Mozilla Firefox 7 vom 27.09.2011 [8]
- Google Chrome 14 vom 16.09.2011 [9]
- Apple Safari 5.0 vom 07.06.2010 [10]

So ist auch zugleich gewährleistet, dass sie sich von verschiedenen gängigen Betriebssystemen, wie folgenden Mindestversionen, nutzen lassen:

- Microsoft Windows XP SP3 vom 07.05.2008 [11]
- Ubuntu Linux 10.04 vom 29.04.2011 [12]
- Apple Mac OS X 10.5 vom 26.10.2007 [13]

Um die Anwendung auch auf Smartphones nutzen zu können, auf denen nicht Android läuft, liegt es nahe, auch die Weboberfläche dahingehend zu optimieren, dass diese für mobile Webbrowser tauglich ist. Dazu sind im Wesentlichen folgende Browser zu nennen:

- Mozilla Firefox Mobile
- Dolphin Browser
- Opera Mobile
- Apple Safari des iOS 5
- Google Browser des Android™-Systems

Beim Aufruf aus nicht unterstützten Webbrowsern sollte nach Möglichkeit eine entsprechende Information anstelle der Weboberfläche auftauchen, die kompatible Webbrowser-Alternativen auflistet.

2.4.1 Formulare zur Kundendaten-Verwaltung

Über den Webclient soll es sowohl möglich sein, bereits eingetragene Kunden zu suchen, anzuzeigen und neue hinzuzufügen, als auch vorhandene Datensätze zu bearbeiten. Bei Letzterem ist die oben genannte Einschränkung bzgl. der geplanten Anbindung an eine Rechnungsverwaltung zu berücksichtigen und folglich die Bearbeitung der rechnungsrelevanten Daten zu verhindern. Lediglich ein Ergänzen noch fehlender Angaben kann möglich sein, wie z. B. das Hinzufügen eines noch fehlenden Vornamens einer Person oder eine fehlende Hausnummer einer Kundenadresse.

Des Weiteren sollten die Formulare weitestgehend dynamisch sein, sodass es ohne großen Aufwand möglich ist mehrere Kontaktpersonen, Telefonnummern, Adressen etc. einem Kunden hinzuzufügen oder zu entfernen. Bei der Eingabe der Anfahrtspauschale wäre eine Umrechnung zwischen brutto und netto Beträgen unter Berücksichtigung des aktuellen Umsatzsteuersatz von 19% sinnvoll.

2.4.2 Weitere Funktionen

Bei der Anmeldung an der Weboberfläche soll der Benutzer die Wahl haben, ob er auch über das Beenden des Browsers hinaus angemeldet bleiben soll oder nur solange, bis der Browser-Tab bzw. das -Fenster geschlossen wird. Bei der Suche nach Kunden sollen während der Eingabe, bereits nach wenigen Buchstaben, passende Treffer aus der Datenbank angezeigt werden, in denen an beliebiger Stelle im Personen- oder Firmennamen die gesuchte Zeichenkette vorkommt. Schließlich soll in der Weboberfläche auch eine Funktion zum Ändern des eigenen Passworts zur Verfügung stehen und eine Übersicht der aktuell aktiven Sessions einsehbar sein, in der sich auch einzelne aktive Sitzungen anderer Clients beenden lassen.

2.5 Android™-App

Beim ersten Starten der App soll zunächst ein Login-Formular erscheinen, mit dem sich der Benutzer mit seinem Kennwort anmelden kann und auch über einen App- oder System-Neustart angemeldet bleibt, solange er sich nicht explizit abmeldet, die Session von einem anderen Client aus beendet oder die Frist von 30 Tagen Inaktivität erreicht wird. Die Startansicht stellt dann ein Suchfeld dar, in das - wie auch auf der Weboberfläche - Teile eines Kunden- oder Personennamens eingegeben werden können, woraufhin direkt nach dieser Zeichenkette in der Datenbank gesucht wird und entsprechende Ergebnisse direkt unter dem Suchfeld aufgelistet werden.

Durch Antippen eines Suchtreffers soll zu einer Listendarstellung gewechselt werden, in der alle Informationen zu dem Kundendatensatz aufgeführt werden. Die ggf. hinterlegten Adressen sollen sich hier direkt durch Antippen in einer der Apps öffnen lassen, die auf dem Gerät installiert sind, und solche Daten (Adressen) entsprechend weiter verarbeiten können. Sollte keine kompatible Anwendung installiert sein, wird der Benutzer entsprechend über diesen Umstand informiert. Auch hinterlegte Webseiten sollen sich aus dieser Listendarstellung heraus direkt im Standard-Browser des Systems öffnen lassen. Die mit dem Kunden verknüpften Personen werden zunächst nur namentlich aufgelistet und können ebenfalls angetippt werden um zugehörige Details in einer neuen Liste anzuzeigen. Hier können dann wiederum die unter anderem angezeigten Telefonnummern und E-Mail-Adressen eingetippt werden, um einen Anruf zu initiieren oder eine neue E-Mail mit der Standard E-Mail-App des Systems zu verfassen. Auch hier gilt wieder, dass ein entsprechender Hinweis erschei-

nen soll, wenn keine passende Anwendung auf dem System vorhanden ist. Zusätzlich werden wiederum die zugeordneten Kundendatensätze aufgeführt, mit denen die Person verknüpft ist, sodass man diese auch direkt durch Antippen aufrufen kann.

2.5.1 Kompatibilität

Die App sollte ab Android™-Version 1.6 „Donut“, die im September 2009 veröffentlicht wurde [14], lauffähig sein, aber auch mit dem aktuellen Android™ 4.0 „Ice Cream Sandwich“ vom Oktober 2011 [15] funktionieren. Da ein Test auf jeder denkbaren Hardware-Plattform unmöglich ist, sollte zum Testen zumindest der von Google Inc. bereitgestellte Emulator genutzt werden, um die Funktionalität auf den genannten Plattform-Versionen zu gewährleisten. Des Weiteren sollte die Anwendung auch auf verschiedene Geräte-Typen (Smartphone und Tablet), die im Wesentlichen unterschiedlich große Displays nutzen, lauffähig sein.

2.5.2 Weitere Funktionen

Über ein Menü oder Ähnliches soll es die Möglichkeit geben, das eigene Kennwort zu ändern und die aktuellen Benutzer-Sitzungen von anderen Geräten anzuzeigen. Außerdem soll es über eine Einstellungsseite möglich sein, die Adresse des Servers, auf dem die Serveranwendung läuft, anzupassen.

Beim direkten Wählen von Rufnummern soll berücksichtigt werden, ob für die jeweilige Person eingestellt ist, dass die eigene Mobilfunknummer übermittelt werden soll oder nicht (CLIR). Die Codes, die dazu der eigentlichen Rufnummer vorangestellt werden müssen, sollen sich ebenfalls in den Einstelllungen festlegen lassen, allerdings auch mit den in Deutschland üblichen Codes standardmäßig voreingestellt sein.

3 Entwicklungsplattform

Im Folgenden sollen die verschiedenen Komponenten vorgestellt, werden, die für die darauffolgende Entwicklung notwendig waren, bzw. zu denen ein gewisses Grundwissen notwendig ist.

Abbildung 2: Android™ Roboter [30]

3.1 Google Android™

Da das von Google entwickelte und auf Linux basierende Smartphone-Betriebssystem in dieser Bachelor-Thesis eine wesentliche Rolle spielen wird, sollen an dieser Stelle ein paar Hintergrundinformationen zu dem System mit dem grünen Roboter-Maskottchen (Abbildung 2) zusammengefasst werden.

3.1.1 Geschichte

Die erste Version von Android™ erschien im Februar 2008. Das System entwickelte sich sehr schnell, sodass nach Version 1.5 „Cupcake“ und 1.6 „Donut“ bereits im Oktober 2009 Version 2.0 „Eclair“ folgte. Im Mai bzw. Dezember 2010 kamen dann die Versionen 2.2 „Froyo“ und 2.3 „Gingerbread“, die aktuell auf den gängigen Android™-Smartphones anzutreffen sind. Es folgte im Februar 2011 die für Tablets ausgelegte Version 3.0 „Honeycomp“. Die nun vor Kurzem (Oktober 2011) fertiggestellte Version 4.0 „Ice Cream Sandwich“ soll die Tablet- und Smartphone-Versionen (Android™ 2 und 3) wieder vereinen. [16]

3.1.2 Verbreitung

Laut einiger aktueller Statistiken wird derzeit auf den meisten Smartphones das Google-Betriebssystem Android™ eingesetzt, gefolgt vom iOS auf iPhones aus dem Hause Apple und Blackberry-Geräten. Windows Mobile und insbesondere das neuere System Windows Phone 7 aus dem Hause Microsoft spielen da mit Marktanteilen von unter 10% eine eher untergeordnete Rolle, ähnlich wie Symbian und PalmOS. [3] [17] [18]

3.1.3 System-Architektur

Das Android™-System basiert auf einem Linux-Kernel, der auch erforderliche Gerätetreiber und einige Optimierungen bzgl. Energieverbrauch und Speichermanagement mitbringt. Jede Anwendung startet innerhalb einer Dalvik Virtual Machine (DVM), die wiederum jeweils in einem eigenen Prozess läuft. Die dadurch erreichbare Sicherheit (kein gemeinsamer Speicher unter verschiedenen Anwendungen) und Systemstabilität (ein hängender Prozess reißt nicht das ganze System oder andere Anwendungen mit) wird mit höherem Ressourcenbedarf bezahlt. Standardbibliotheken, wie z. B. für die Grafik, Datenbank und Multimedia sind in C/C++ implementiert. Alle weiteren Systemklassen, wie u. a. Oberflächenelemente, Telefonie-Manager, Benachrichtigungs-Manager und Fenster-Manager sind komplett in Java implementiert. Diese Schicht wird auch als Anwendungsrahmen bezeichnet. Auf der darüber liegenden Anwendungsschicht der Android™-Architektur laufen dann sowohl die Android-eigenen Anwendungen, als auch die eigens programmierten bzw. von Drittanbietern bereitgestellten Anwendungen. [19]

3.1.3.1 Die Dalvik Virtual Machine

Die primär von Google entwickelte virtuelle Maschine stellt den Kern der Android™-Laufzeitumgebung dar, basiert auf der „klassischen“ Java Virtual Machine (JVM) und wurde auf die Anforderungen mobiler Endgeräte zugeschnitten: Also etwa eine CPU mit 500 MHz und 64 MB Spei-

cher, von denen je Anwendung 20 zur Verfügung stehen mit einem Linux-System ohne swap space und Geräte-Betrieb mit Batterien. [20]

Anwendungen werden zur Entwicklungszeit wie üblich in Java-Bytecode übersetzt aber anschließend durch das Android™-SDK in den notwendigen Dalvik-Bytecode umgewandelt, der dann später auf dem Gerät von der DVM ausgeführt werden kann. [19]

Die DVM ist im Gegensatz zur JVM, die als Stack-Maschine realisiert ist, als Register-Maschine umgesetzt. Das heißt, die Bytecodes und Operanden werden aus (virtuellen) Registern geholt, was natürlich auch bedeutet, dass Operanden je nach Opcode in bestimmten Registern abgelegt und aus diesen gelesen werden müssen, und nicht wie bei einer JVM immer vom Stack geholt werden. Dadurch, dass diese virtuellen Register auf reale Register des Mikroprozessors abgebildet werden können, bekommt man einen deutlichen Geschwindigkeitsgewinn. So werden also die Möglichkeiten moderner Prozessoren besser ausgenutzt als von einer „normalen“ JVM. Dies hat allerdings auch den Nachteil, dass einem die Portabilität einer Stack-Maschine verloren geht. Das ist in diesem Fall nicht so tragisch ist, da das Android™-System ohnehin auf ARM-Architektur ausgerichtet ist. Inzwischen gibt es Prozessorhersteller, die die DVM auf ihre Prozessorarchitektur adaptiert haben. [19] [20]

3.1.4 App-Programmierung

Im Folgenden sollen ein paar Begriffe und Besonderheiten der Entwicklung von Anwendungen für die Android™-Plattform vorgestellt werden.

3.1.4.1 Komponenten

Eine Android App besteht aus einer oder mehreren der folgenden Komponenten:

- **Activities** spiegeln Oberflächen einer App wider, die Texte, Eingabefelder, Buttons etc. beinhalten können. Ebenso kümmern sie sich um die Behandlung von Benutzeraktionen, lesen Inhalte aus Eingabefeldern und reagieren auf die Auswahl eines Menüpunktes. Mehrere dieser Activities können zu einer umfangreicheren Anwendung verknüpft werden. Jede Activity hat ihren eigenen Lebenszyklus. Im folgenden Abschnitt (3.1.4.2) wird näher auf diese App-Komponente eingegangen.
- **Services** kümmern sich um Hintergrundprozesse, die typischerweise für längere Zeit unsichtbar im Hintergrund laufen sollen. Jeder Service läuft in einem eigenen Prozess. Diese Art der Komponente wird im Rahmen der vorliegenden Arbeit nicht zum Einsatz kommen und von daher auch nicht näher behandelt werden.
- **Content Provider** verwalten Daten und stellen diese über eine definierte Schnittstelle bestimmten Anwendungen oder auch allen Anwendungen zur Verfügung. Auch diese Komponenten-Art wird nicht Bestandteil der vorliegenden Ausarbeitung sein.
- **Broadcast Receiver** empfangen Nachrichten, die von anderen Anwendungen oder dem Android™-System versendet wurden, die bspw. über eine niedrige Akku-Leistung informieren. Derartige Komponenten sollen hier ebenfalls nicht näher beschrieben werden.

Jede in Form einer Java-Klasse existierende Komponente muss in der sogenannten Manifest-Datei vom Entwickler der Anwendung eingetragen und dabei wird auch dessen Funktion festgelegt. Bei der Manifest-Datei handelt es sich um eine Datei im XML-Format, die außerdem u. a. auch den Titel oder eine kurze Beschreibung der Anwendung enthält. Diese Informationen werden dann z. B. vom Android™-System verwendet, um die Anwendung in der Liste der installierten Apps aufzulisten. [19]

3.1.4.2 Activities

Eine Activity stellt eine sichtbare Benutzeroberfläche der App dar. Eine Anwendung besteht typischerweise aus mehreren Activities. Jede dieser Activities muss dabei – wie jede andere Komponente auch – in die Manifest-Datei eingetragen werden. Dabei kann dann z. B. zu einer Activity festgelegt werden, dass diese ein Icon in der Liste der Anwendungen des Android™-Systems erhält, über das dann diese Activity gestartet wird. Eine Activity kann dann andere Activities starten, z. B. wenn der Benutzer auf einen Button tippt oder einen Menüpunkt auswählt. [21]

3.1.4.3 Intents

Als Intent werden abstrakte Beschreibungen für Operationen bezeichnet, die ausgeführt werden sollen. Dies kann z. B. dazu genutzt werden, um eine (andere) Activity zu starten, einen Broadcast zu verschicken oder einen Service zu aktivieren. Dabei können beliebige Parameter mitgegeben werden, die z. B. von der aufgerufenen Activity ausgewertet werden können. [22]

3.1.4.4 Berechtigungen

Da jede Anwendung in einer Art Sandbox läuft, also mit Ihrer eigenen VM in einem eigenen Prozess, eigenem System-User, eigenem Bereich im Hauptspeicher und eigenem Bereich im Dateisystem, kann Sie zunächst auch auf keine Komponenten außerhalb dieser Sandbox zugreifen. Hier kommen die vom Entwickler individuell festlegbaren Berechtigungen ins Spiel, die es der Anwendung dann ermöglicht gezielt auf bestimmte Ressourcen zuzugreifen. Derartige Ressourcen können z. B. der Zugriff auf das Internet sein, eine Berechtigung zum SMS-Versand oder zur Anruf-Initialisierung. Diese Berechtigungen werden in der bereits erwähnten Manifest-Datei festgelegt. Bei der späteren Installation der Anwendung auf dem Endgerät wird der Benutzer über die gewünschten Berechtigungen informiert und kann sich aufgrund dessen entscheiden, ob er die Anwendung tatsächlich installieren möchte. [19]

3.1.4.5 R-Klasse

Mit Hilfe der R-Klasse, die vom Android™-SDK automatisch generiert wird, kann aus Java komfortabel auf einzelne Ressourcen, wie z. B. Grafiken und Layouts verwiesen werden. Auch der Zugriff auf einzelne Elemente eines Layouts, die mit einer ID versehen wurden, und die einzelnen Strings einer String-Ressourcen-Datei erfolgt aus dem Java-Programm heraus über diese R-Klasse, die im Wesentlichen öffentliche statische Variablen enthält.

3.2 PostgreSQL Datenbank

Als Backend für die Anwendung sollte ja bekanntermaßen eine PostgreSQL-Datenbank zum Einsatz kommen. Diese wurde dazu in Version 8.4 auf einem virtualisierten Server mit einem 64-Bit Linux-System (Ubuntu Server 10.04.3 LTS) aus den System-Repositories installiert.

Bei PostgreSQL handelt es sich um eine seit 1989 existierende Open-Source-Software, die ein objektrelationales Datenbankmanagementsystem darstellt. Dabei wird überwiegend der SQL-Standard umgesetzt und um einige weitere Funktionen ergänzt. Die hier eingesetzte und noch immer sehr verbreitete Version 8.4 erschien im Juli 2009 und wurde seitdem mit einigen Sicherheitsupdates aktualisiert. Inzwischen wird diese nach und nach von der neueren Version 9 bzw. 9.1 abgelöst. [23]

Zusätzlich kam zum Testen und zur Administration der Datenbank das grafische Verwaltungs-Programm pgAdmin III zum Einsatz, bei dem es sich ebenfalls um eine Open-Source-Software handelt. Hierüber ist ein komfortabler Zugriff auf die gesamte PostgreSQL-Funktionalität möglich. [23]

3.3 Apache Webserver

Für den Zugriff vom Client auf die Datenbank sollte ein PHP-Script zum Einsatz kommen, das via HTTPS mit dem Client Daten austauschen und auf der anderen Seite direkt mit der PostgreSQL-Datenbank via SQL Kontakt aufnehmen kann. Hierzu wurde auf dem bereits erwähnten virtualisierten Server auch der Apache Webserver in Version 2.0 und PHP (PHP: Hypertext Preprocessor) in Version 5.3 aus den Ubuntu-Repositories installiert. PHP klinkt sich dabei via API (Application Programming Interface) beim Apache-Server ein, sodass die PHP-Skripte bei ihrem Aufruf serverseitig ausgeführt werden und dessen Ausgaben per HTTPS an den Client übermittelt werden können.

Um die verschiedenen Instanzen zu realisieren, wurden unterhalb des vom Apache veröffentlichten Stammverzeichnisses entsprechende Unterverzeichnisse angelegt.

Um die SSL-verschlüsselte HTTP-Verbindung zu ermöglichen, wurde ein Zertifikat erstellt und von dem kostenlosen Anbieter StartSSL[1] signiert. Dies hat gegenüber einem selbst oder einem von CAcert signierten Zertifikat den Vorteil, dass es zumindest von ein paar aktuellen Browsern ohne Benutzer-Rückfrage akzeptiert wird, die das entsprechende Root-Zertifikat als vertrauenswürdig einstufen und mitbringen.

Da sich der Server hinter einem NAT-Router befindet, hinter dem auch bereits ein anderer Webserver läuft, wurde ein abweichender TCP-Port für HTTPS verwendet (446), der sich jedoch später auch problemlos am Client einstellen lässt. Zum komfortableren Aufruf der Webseite wurde eine Weiterleitung für den eingesetzten Domain-Namen zu dem abweichenden Port auf dem Server eingerichtet, der an dem Standard TCP-Ports von HTTP (80 und 443) lauscht.

3.4 Webbrowser

Da auf eine starke Abwärtskompatibilität bei den Webbrowsern gemäß der genannten Anforderungen (siehe 2.4, Seite 5) verzichtet werden konnte, wurde bei der Entwicklung auf aktuelle Techniken aus HTML5, CSS3 und JavaScript gesetzt, um die Weboberfläche möglichst dynamisch erscheinen zu lassen. Während der Entwicklung wurde dabei primär der Webbrowser Mozilla Firefox verwendet, da dieser mit den kostenlosen Add-ons Web Developer[2] und Firebug[3] viele komfortable Möglichkeiten zum Debuggen und Testen bietet, sowie der in Android™ standardmäßig vorhandene Webbrowser als beispielhafte Plattform mit kleinem Display. Zum abschließenden Testen wurden dann ergänzend die geforderten Browser in den jeweils genannten Versionen unter Windows XP, Windows 7, Ubuntu Linux 10.04 und Mac OS X 10.6 sowie die verschiedenen mobilen Browser unter Android 2.3 und 3.0 sowie iOS 4.3 herangezogen.

3.5 Eclipse IDE

Bei der IDE fiel die Wahl ohne nennenswerte Alternativen auf Eclipse Indigo: Das Android™ SDK von Google wird exklusiv für diese Entwicklungsumgebung zur Verfügung gestellt. Ein von Drittanbietern angebotenes Plugin für NetBeans kann hingegen lange nicht an den Komfort und Funktionsumfang des Eclipse-Plugins heranreichen. Da sich auch die Eclipse IDE auch gut für die anderen eingesetzten Programmier- und Auszeichnungssprachen (Java, PHP, JavaScript, CSS, HTML) eignet, wurde dann diese Software konsequent für die Entwicklung aller Software-Komponenten genutzt. Zur Verbesse-

[1] Weitere Informationen: https://www.startssl.com/
[2] Weitere Informationen und Download: http://chrispederick.com/work/web-developer/
[3] Weitere Informationen und Download: http://getfirebug.com/

rung der Unterstützung von JavaScript und HTML wurde jedoch zusätzlich das kostenlose Aptana Studio 3[4] installiert, das sich als Plugin in Eclipse integriert.

3.6 Android™ SDK

Das für die Entwicklung und Publizierung von Android™-Anwendungen notwendige SDK wird von Google kostenlos unter http://developer.android.com/sdk/ zum Download zur Verfügung gestellt. Im Rahmen dieser Bachelor-Thesis wurde mit dem derzeit aktuellen Release 15 bzw. anfangs noch mit dem Release 14 gearbeitet.

3.7 Subversion (SVN)

Um während der Entwicklung eine komfortable Versionsverwaltung zur Verfügung zu haben, wurde auf dem ohnehin notwendigen Server auch ein Apache Subversion[5] Repository eingerichtet.

Mit Hilfe des SVN ist es problemlos möglich, von einzelnen Dateien ältere Versionen wiederherzustellen, wenn bspw. bei der aktuellen Version Änderungen gemacht wurden, die nicht mehr rekonstruierbar sind, und durch die etwas nicht mehr so funktioniert, wie vorher. Außerdem ermöglicht es ein komfortables Arbeiten von verschiedenen PCs aus: mal vom Notebook und ein andermal am Arbeitsplatz zu Hause.

Dabei wurde serverseitig dafür gesorgt, dass bei jedem Commit, auch auf dem Webserver unter der Instanz „test“ der aktuelle Stand aus dem SVN-Repository online verfügbar gemacht wurde. So war es komfortabel möglich, immer den aktuellen Stand direkt online im Browser zu testen, ohne nochmal manuell per FTP oder Ähnlichem die aktualisierten Dateien auf den Webserver hochladen zu müssen.

[4] Weitere Informationen und Download: http://aptana.com/products/studio3
[5] Weiterführende Informationen: http://subversion.apache.org/

4 Software-Design

Durch die vereinheitlichte Kommunikationsschnittstelle via HTTPS im JSON-Format konnten die einzelnen Softwarekomponenten (Serveranwendung, Webclient und Android™-App) autonom entwickelt werden. An dieser Stelle sollen daher die Software-Designs der einzelnen Komponenten vorgestellt werden.

4.1 Serveranwendung

Die Serveranwendung wird im Wesentlichen durch das PHP-Script repräsentiert, das eine Schnittstelle zwischen SQL-Datenbank und dem JSON-Datenformat darstellt.

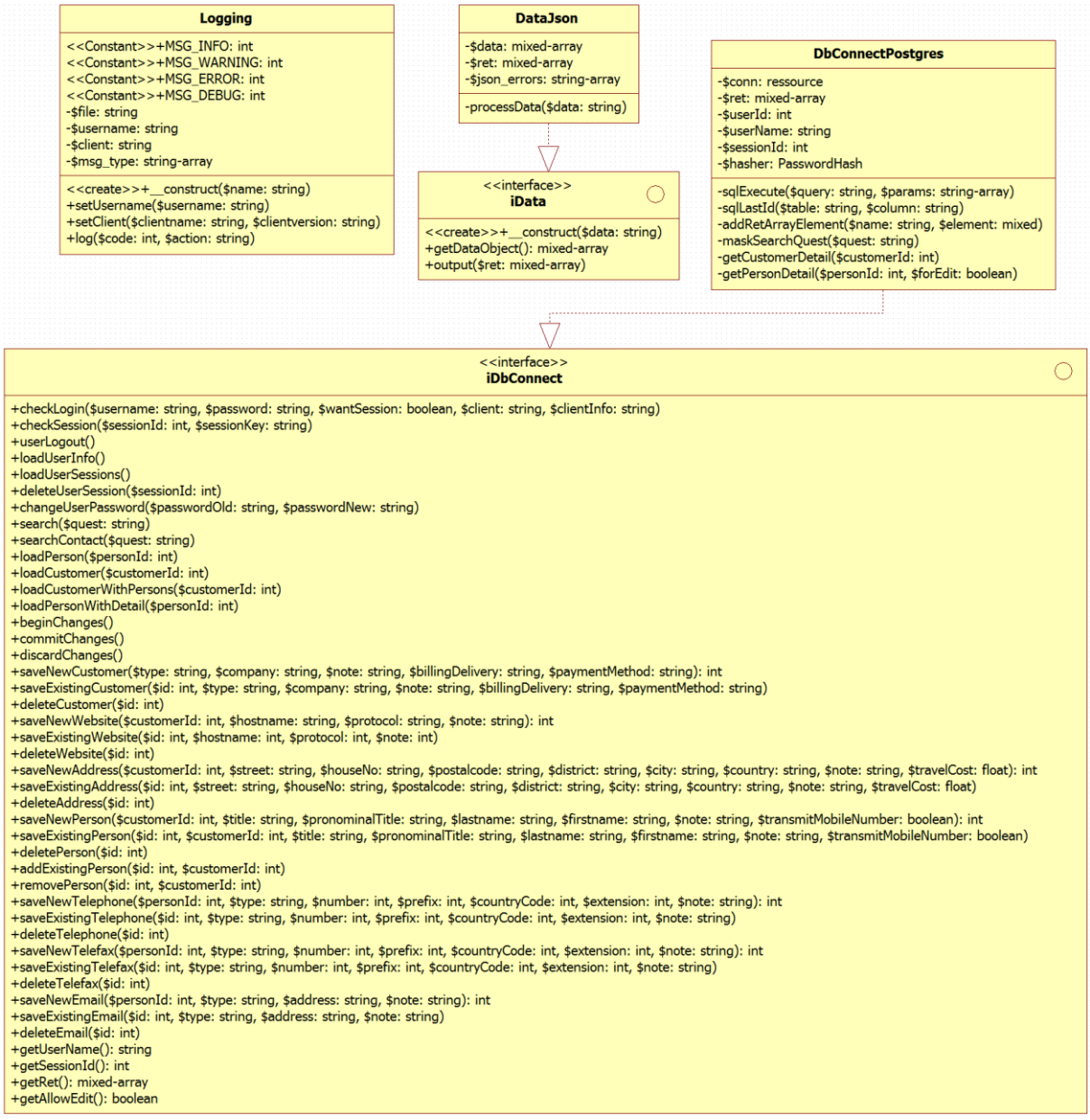

Abbildung 3: Klassendiagramm der PHP-Serveranwendung (leicht gekürzt)

Da es sich bei PHP ursprünglich um eine prozedurale Skriptsprache handelte, bei der erst mit Version 4 objektorientierte Konstrukte eingeführt wurden und dann mit PHP5 erstmals bspw. Sichtbarkeitsbeschränkungen wie *protected* und *private* umgesetzt wurden, sind viele Teile (Bibliotheken) der

Sprache nach wie vor nicht rein objektorientiert nutzbar. Folglich gibt es auch neben den Klassen, die in der obigen Abbildung 3 ersichtlich sind, noch eine Datei mit dem Hauptprogramm und globalen Funktionen. Das ist dann auch die Datei, die vom Client via HTTPS aufgerufen wird, woraufhin dann entsprechende Objekte der obigen drei Klassen (Logging, DataJson und DbConnectPostgres) erzeugt werden. Eine zusätzliche Datei beinhaltet von der Logik abgekoppelte Einstellungen, wie z. B. die Daten, die zur Verbindung mit der Datenbank nötig sind. Außerdem kommt noch eine Klasse zum Einsatz die lediglich kontante String enthält. Dbaei handelt es sich um die Schlüssel, die zum Aufbau der JSON-Objekte benötigt werden. Diese Liste von Konstanten lässt sich dann recht einfach mit der Java-Klasse der Android-App abgleichen, die ebenfalls diese Strings benötigt.

4.2 Webclient

Der Webclient gliedert sich in drei Schichten, die in diesem Kapitel jeweils vorgestellt werden sollen: Zum einen gibt es den HTML-Teil, der die grobe Dokumenten-Struktur abbildet. Darin werden der CSS-Teil – für ein einheitliches Layout – und der JavaScript-Teil verknüpft, in dem die Logik steckt und der auch die Interaktion ermöglicht. Im Folgenden soll jeweils auf die Designs bzw. Strukturen innerhalb der einzelnen Schichten eingegangen werden:

4.2.1 HTML

Der Webclient setzt sich aus mehreren Dateien zusammen, zum einen einer HTML-Datei, die die grobe Struktur für die Darstellung im Webbrowser repräsentiert. Die enthält im Wesentlichen leere Objekte mit IDs, die später durch JavaScript individuell mit Inhalt gefüllt werden. Des Weiteren sind statische Elemente enthalten, die sich nie ändern (Überschrift, Menü-Zeile etc.). Bei Bedarf werden dann von der JavaScript-Logik einzelne – für die aktuelle Ansicht nicht zutreffende – Elemente ausgeblendet. Der HTML-Teil wird vollständig in einer Datei untergebracht, die vom Server abgerufen wird, wenn der Webclient gestartet wird. Darin befinden sich dann für den Webbrowser Informationen zu den weiteren notwendigen CSS- und JavaScript-Dateien, die vom Server geladen werden müssen.

Die HTML-Struktur gliedert sich wie folgt:

- head (meta-Informationen, weitere zu ladende Dateien)
- body
  - img ID „companylogo“ (Logo des Unternehmens)
  - h1 (Hauptüberschrift)
  - div ID „menu“ (Menü-Zeile mit einzelnen Buttons)
  - div ID „nojs“ (Information über deaktiviertes JavaScript)
  - div ID „nohtml5“ (Information über nicht kompatiblen Webbrowser)
  - form ID „login“ (Login-Formular)
  - div ID „search“ (Such-Formular)
  - div ID „searchresults“ (Such-Ergebnisse)
  - div ID „customerdetail“ (Detail-Darstellung eines Kunden)
  - div ID „persondetail“ (Dateil-Darstellung einer Person)
  - div ID „editcustomer“ (Formular zum Bearbeiten von Kundendaten)
  - div ID „loading“ (Ladebalken)
  - div ID „popupbg“ (Hintergrund des Popup zur Ausblendung anderer Inhalte)
  - div ID „popupview“ (Über der Seite liegendes Popup)
  - div ID „debugview“ (Debug-Zeile am unteren Rand)

4.2.2 CSS

Für die Feinjustierung des Layouts wird eine separate CSS-Datei verwendet. Hierin werden Elemente anhand ihres Tag-Namens, ihrer ID oder ihres Klassennamens mit einem entsprechenden Layout versehen. Hier sind also u. a. Informationen wie Schriftgröße/-farbe, Umrandungsfarbe/-dicke und Abstände zwischen Objekten festgelegt. Auch abweichende Layout-Angaben für unterschiedlich große Displays sind in dieser CSS-Datei hinterlegt.

4.2.3 JavaScript

Bei JavaScript gibt es keine Klassen-Definition, wie man sie bspw. von Java oder C++ kennt, so gibt es auch kein derartiges Schlüsselwort wie *class*. Vielmehr wird bei JavaScript alles als *function* bezeichnet. Dabei handelt es sich jedoch im Gegensatz zu anderen Programmiersprachen auch um Objekte, die folglich wiederum Eigenschaften und Objekte beinhalten können. Eine Funktion kann damit auch gleichzeitig immer als Konstruktor einer gleichnamigen Klasse betrachtet werden. Schließlich sei noch erwähnt, dass es auch keine Sichtbarkeitsbeschränkungen gibt, sodass praktisch alles als *public* anzusehen ist. Aufgrund dieser Tatsachen ist es meistens wenig sinnvoll ein abstraktes Klassenmodell aufzustellen, deshalb wurde sich auch bei der Entwicklung des Webclients dagegen entschieden. Stattdessen wurde auf eine Strukturierung der globalen Funktionen in mehrere Dateien gesetzt. In diesem Sinne kann von einer Programmierung im funktionalen Stil gesprochen werden. Es wurden die einzelnen Funktionalitäten zur besseren Übersicht in folgende Dateien aufgeteilt:

- webclient.js (Hauptfunktionalität, die u. a. die initialisierende Funktion enthält)
- viewswitching.js (Sorgt für den Wechsel zwischen verschiedene Ansichten/Formularen)
- htmlcreation.js (Funktionen zur Erzeugung von HTML-Strukturen bzw. -Elementen)
- tools.js (Allgemeine Funktionen, die an verschiedenen Stellen genutzt werden)
- editform.js (Funktionen, für das dynamische Formular zur Datensatz-Bearbeitung)
- communicate.js (Kümmert sich um die (asynchrone) Kommunikation mit dem Server)
- eventhandling.js (Reagiert auf allgemeine Ereignisse, wie das Fokussieren eines Feldes)
- config.js (Enthält Konfigurations-Parameter zur Abkopplung von der Logik)

In allen JavaScript-Dateien kam die Notation für sogenanntes striktes JavaScript zum Einsatz, was durch die folgende einleitende Zeile festgelegt wird:

```
"use strict";
```

Browser, die den strikten Modus nicht unterstützen (etwa der Internet Explorer bis einschließlich Version 9), ignorieren diesen einfachen String, während andere dann eine Fehlermeldung produzieren, sobald ein Konstrukt verwendet wird, das im strikten Modus nicht (mehr) vorgesehen ist, wie z. B. die Variablen-Definition ohne das Schlüsselwort `var`.

4.3 Android™-App

Ähnlich wie bei HTML erfolgt auch die Gestaltung des Layouts einer Android™-App typischerweise innerhalb von XML-Dateien. Des Weiteren sind

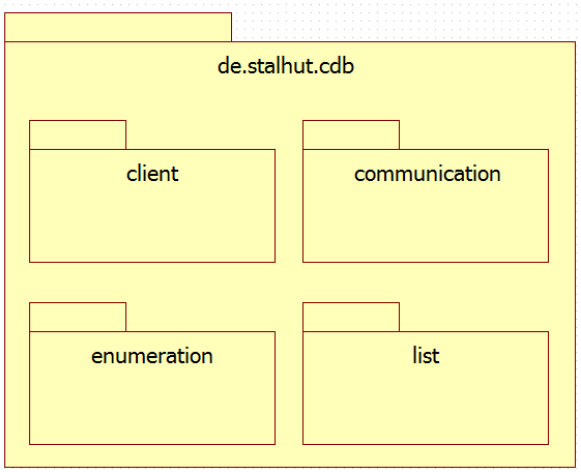

Abbildung 4: Paketdiagramm der Android™-App

auch alle sprachabhängigen Strings in eine XML-Datei auszulagern. Ansonsten können übliche Java-Klassen-Strukturen eingesetzt werden, wobei lediglich für jede Activity zwingend eine Java-Klasse vorhanden sein muss. Die Activities müssen sich im Haupt-Package „de.stalhut.cdb.client“ (das als solches in der Manifest-Datei angegeben werden muss) befinden, während die anderen Klassen zur besseren Strukturierung in seperate Packages ausgelagert werden. So wurden für die Kommunikations-Klassen ein Paket namens *cummunication* angelegt, für die verwendeten Enumerationen ein gleichnamiges Package und für die speziellen Listen-Klassen, die später für die ListViews des UI notwendig sind, ein Package namens *list* hinzugefügt. Das Hautpacke nimmt hier zusätzliche die Rolle der eindeutigen Identifizierung ein, da das Android™-System daran beim aktualisieren erkennt, ob es sich um die gleiche App handelt. Im Folgenden sollen auf die Activity-Struktur im Hauptpaket sowie die Inhalte der einzelnen Unterpakete genauer eingegangen werden.

4.3.1 Activities

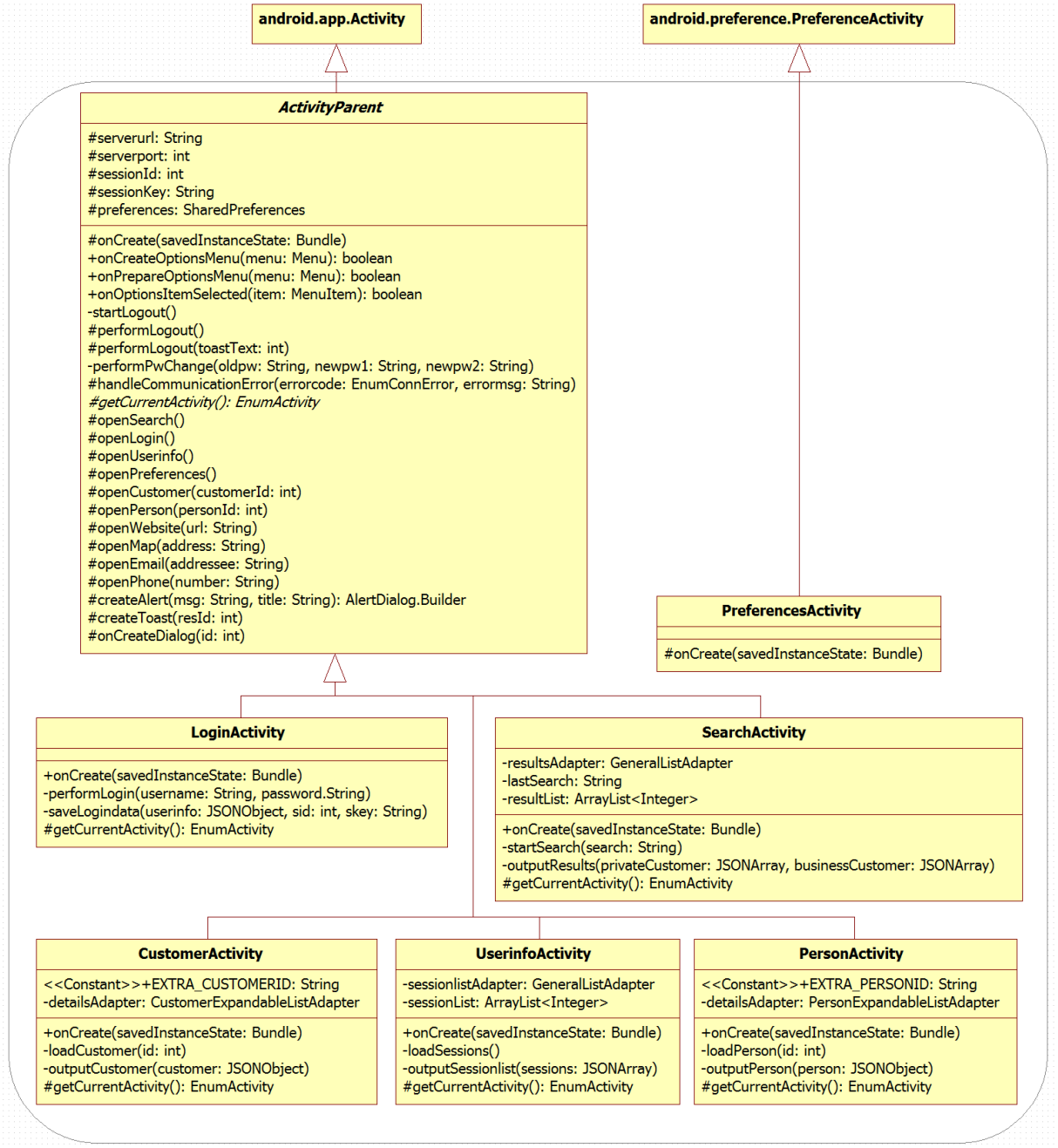

Abbildung 5: Klassenstruktur im Haupt-Package mit den Klassen der einzelnen Activities (leicht gekürzt)

Alle gemeinsamen Funktionen wurden, wie in Abbildung 5 zu sehen, in eine übergeordnete und als abstrakt definierte übergeordnete Klasse ausgelagert. Ansonsten gibt es seitens des Android™-Systems Klassen, von denen geerbt werden muss, damit die Klasse eine Activity repräsentieren kann. Bei der Activity für die App-Einstellungen wurde die vom System angebotene spezielle *PreferenceActivity* für diese Funktion verwendet, da dann fast nichts mehr programmiert werden muss, sondern nur die gewünschten Parameter in einer XML-Datei festgelegt werden müssen.

4.3.2 Listen

Android™ bietet standardmäßig Elemente, die zur Darstellung von scrollfähigen Listen verwendet werden können, dabei hat man jedoch keinen Einfluss auf das Layout und ist an die Verwendung von einzeiligen Listen-Einträgen gebunden. Zur Realisierung individueller Listen sind sogenannte Adapter zu implementieren, wobei von den jeweiligen Standard-Klassen für Listen-Adapter geerbt wird.

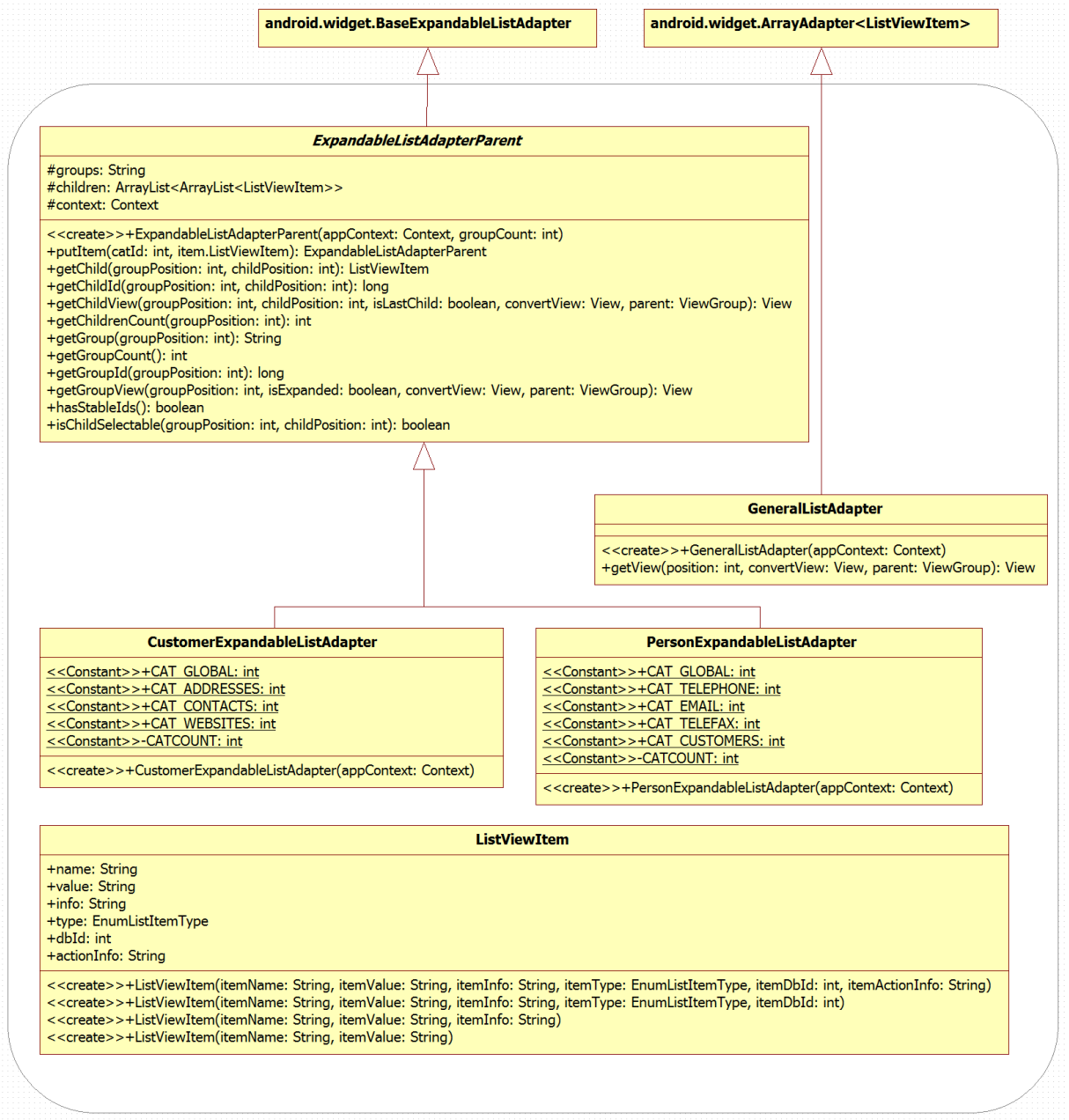

Abbildung 6: Klassendiagramm der Klassen im Package list

Zusätzlich wurde eine Datenklasse programmiert, die die Informationen eines Listen-Elements aufnehmen kann und von der dann mehrere Objekte als ArrayList an den Adapter übergeben werden können. Die genaue Struktur kann dem obigen Diagramm (Abbildung 6) entnommen werden.

Individuelle Listen finden auf den Detailseiten von Kunden und Personen Verwendung, und werden für die Suchergebnisse und Session-Auflistung eingesetzt. Für die Detailseite kommt dabei die spezielle Form der *ExpandableList* zum Einsatz, bei der mehrere Elemente zu einer ausklappbaren Gruppe zusammengefasst werden.

4.3.3 Kommunikation

Die Kommunikation mit dem Server muss in einen separaten Thread ausgelagert werden, damit nicht der UI-Thread während dem Nachladen von Daten zu „hängen“ scheint. Dazu bietet das Android™-System eine Klasse Namens AsyncTask. Das Programm verwendet nun, wie in dem folgenden Diagramm (Abbildung 7) zu sehen ist, eine abstrakte Kommunikations-Klasse, die von dieser AsyncTask-Klasse abstammt. Für die konkreten Kommunikations-Aufgaben wie Login, Logout, Suche usw. wird dann in der jeweiligen Activitiy eine private innere Klasse implementiert, die von dieser Kommunikations-Klasse erbt. Im Klassendiagramm sind diese inneren Klassen anhand der beiden Beispiele Login und Suche dargestellt:

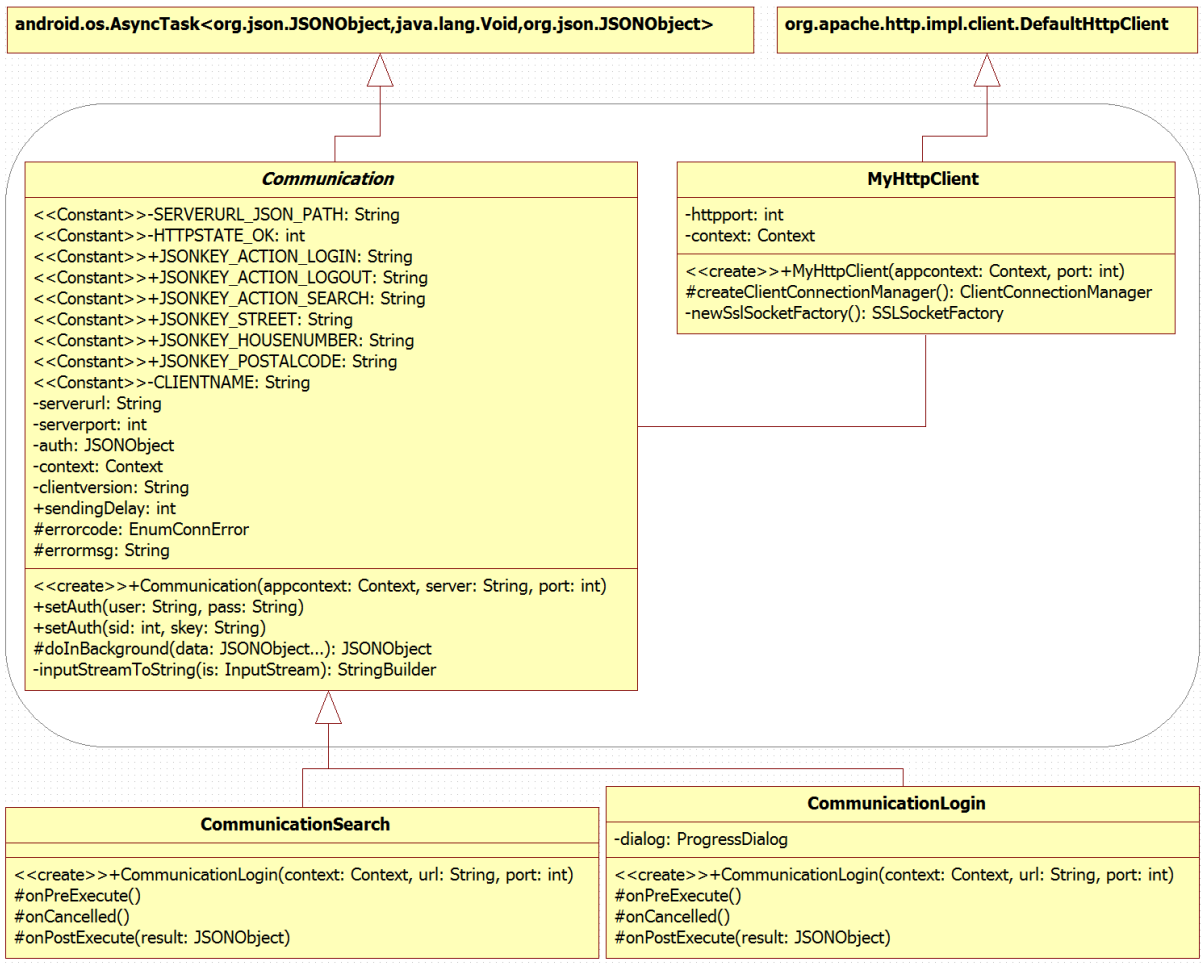

Abbildung 7: Klassendiagramm zum Package *communication* mit zwei beispielhaften inneren Klassen aus einzelnen Activities. (Die JSONKEY_* Konstanten wurden zur besseren Übersicht nicht alle dargestellt.)

Zusätzlich ist eine Anpassung des Standard HTTP-Clients notwendig, da sonst die Verbindung wegen vermeintlich ungültigen SSL-Zertifikaten scheitern würde, weil es von einem dem Android™-System

unbekannten Stammzertifikat signiert wurde. So kann jedoch auf das extra in der App eingebundene Zertifikat verwiesen und dieses explizit als gültig erklärt werden.

4.3.4 Enumerations

An verschiedener Stelle im Programm ist der Einsatz von Enumerations sinnvoll, diese werden in diesem Package zusammengefasst definiert. Die konkreten Emum-Definitionen können dem zugehörigen Klassendiagramm (Abbildung 8) entnommen werden.

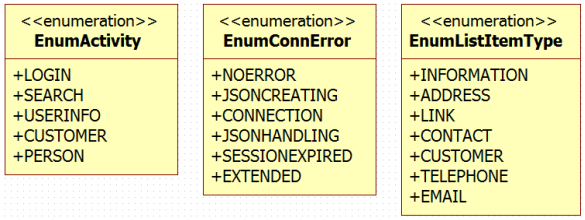

Abbildung 8: Klassendiagramm des Enumeration-Package

5 Server-Anwendung

Die bereits in den Anforderungen beschriebene Serveranwendung (Abschnitt 2.1, Seite 3) besteht im Wesentlichen aus einer relationalen Datenbank. Auf diese PostgreSQL-Datenbank greift ein PHP-Skript zu, das angeforderte Daten per JSON an den Client sendet und dabei auch die Benutzerauthentifizierung übernimmt. Für den Zugriff auf die SQL-Datenbank beinhaltet PHP bereits eine Erweiterung, die auch keiner weiteren speziellen Konfiguration bedarf. [24]

5.1 Datenbank

Für die Datenbank wurde - gemäß den Anforderungen - das in Abbildung 9 dargestellte normalisierte Datenbank-Modell aufgestellt (der SQL-Code zur Datenbank-Erzeugung kann Anhang F, Seite 72 entnommen werden):

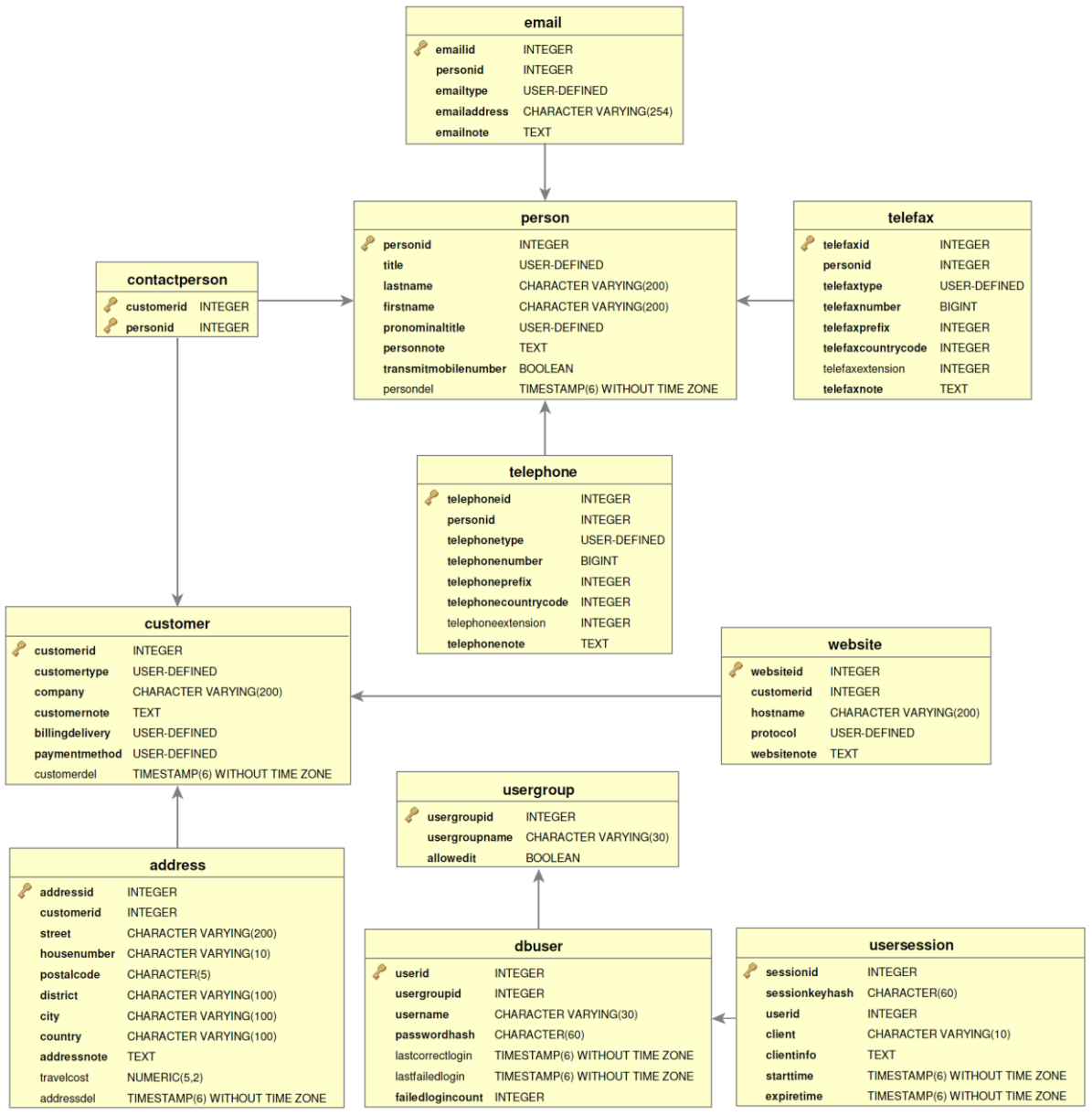

Abbildung 9: Entitiy-Relationship-Diagramm der normalisierten Datenbank

Beim Design der Datenbank wurde nach den üblichen Regeln der Normalisierung vorgegangen, die Redundanzen vermeiden und damit die Gefahr der Inkonsistenz verringern: [25]

- **Nullte Normalform:** Datenelemente der realen Welt werden als Tabelle aufgelistet, wobei berechnete Felder weggelassen werden. Diese Tabelle kann auch als unnormalisiert bezeichnet werden.
- **Erste Normalform:** Alle Attribute müssen atomar sein, das heißt es dürfen zum einen nicht mehrere verschiedenartige Werte in einem Attribut zusammengefasst sein, was hier u. a. bedeutet, dass es nicht ein Feld für die Adresse gibt, sondern diese in PLZ, Ort, Land, Straße und Hausnummer aufgeteilt wird. Zum anderen müssen Wiederholungsgruppen eliminiert werden, sodass bspw. nicht mehrere Telefonnummern einer Person in einem Feld aufgelistet werden, sondern diese auf mehrere Zeilen (Datensätze) verteilt und diese dann in einer separate Tabelle aufgelistet werden.
- **Zweite Normalform:** Zusätzlich zu den Bedingungen der ersten Normalform darf keines der Nicht-Schlüsselattribute von einem der Schlüsselkandidaten voll funktional abhängen. Hier ist ansonsten ggf. ein Aufteilen der Tabelle in einzelne Tabellen notwendig.
- **Dritte Normalform:** Neben den Bedingungen der zweiten Normalform ist es notwendig, dass jedes Nichtschlüsselattribut ausschließlich von einem Primärschlüssel abhängig ist und nicht zusätzlich von einem anderen Nichtschlüsselattribut. Auch hier ist ggf. ein Aufteilen der Tabellen nötig, um die Normalform zu erreichen.
- Weitere weniger relevante Normalisierungsstufen sind die **Boyce-Code Normalform**, gefolgt von der sogenannten **vierten** und schließlich **fünften Normalform**.

Generell war das Ziel die 3. Normalform zu erreichen. Diese wird nach den genannten Vorgaben von der obigen Datenbank (Abbildung 9) nicht vollständig erfüllt, da in der Adressen-Tabelle der Ort von der Postleitzahl abhängt und das Land wiederum vom Ort, sodass man diese Tabelle streng genommen auf drei Tabellen verteilen müsste. Auf diesen letzten Schritt der Normalisierung wurde jedoch an dieser Stelle bewusst verzichtet, da eine derartige Aufteilung zusätzlichen Aufwand beim Zugriff (lesen, einfügen und aktualisieren) auf die Daten bedeuten würde, insbesondere da diese Informationen von der geplanten Anwendung ausschließlich gemeinsam abgefragt werden (als gesamte Adresse). Auch das Problem der Inkonsistenz ist in diesem Kontext nicht relevant: Es wäre beispielsweise irrelevant, wenn einmal die PLZ 28844 mit dem Ort „Weyhe“ und einmal mit „Weyhe bei Bremen“ eingetragen würde.

5.1.1 Speicherung von Telefon- und Telefaxnummern

Die Telefonnummern (und genauso auch Telefaxnummern) wurden in ihre Bestandteile (Ländervorwahl, Ortsvorwahl, Durchwahl und Hauptrufnummer) aufgesplittet. Dies hat zum einen den Vorteil, dass die Nummern nicht wegen möglicher führender Nullen (Verkehrsausscheidungsziffern) oder Plus-Zeichen der Vorwahlen als Text-Strings abgespeichert werden muss, sondern stattdessen die weniger Platz beanspruchenden Integer verwendet werden können. Zusätzlich ermöglicht die Aufteilung eine einheitliche (bspw. gemäß DIN 5008[6]) Darstellung der Telefonnummern bei der Ausgabe, ohne dass diese bei der Eingabe bereits festgelegt sein müssen und vom Benutzer zu beachten wären.

[6] DIN 5008 „Schreib- und Gestaltungsregeln für die Textverarbeitung“, siehe auch http://www.nia.din.de/cmd?level=tpl-artikel&cmstextid=125498

5.1.2 Speicherung der Webseiten-Adressen

Auch bei der Speicherung der Webseiten-Adressen wird eine Aufteilung in Protokoll (http oder https) und Hostname durchgeführt. So ist gewährleistet, dass auch jede URL eine Angabe des Protokolls beinhaltet und damit wiederum eine einheitliche Darstellung bei der Ausgabe.

5.1.3 Formatierung der Ausgabe

Die Formatierung der aufgesplitteten Telefon-/Telefaxnummern und Webseiten-Adressen für die Ausgabe soll durch die Datenbank übernommen werden. Es wurden datenbankseitig Methoden in der prozeduralen Sprache PL/pgSQL geschrieben, die diese Aufgabe übernehmen:

- tele_format_international(INT, INT, BIGINT, INT) RETURNS VARCHAR(25)
- tele_format_national(INT, BIGINT, INT) RETURNS VARCHAR(25)
- tele_format_international_num(INT, INT, BIGINT, INT) RETURNS VARCHAR(25)
- tele_format_national_num(INT, BIGINT, INT) RETURNS VARCHAR(25)
- tele_format_auto(INT, INT, BIGINT, INT) RETURNS VARCHAR(25)
- tele_format_auto_num(INT, INT, BIGINT, INT) RETURNS VARCHAR(25)

Die beiden ersten geben die Rufnummer gemäß DIN 5008 formatiert aus, sodass eine internationale Rufnummer dann bspw. als „+49 421 52079911“ ausgeben wird oder national als „0421 52079911“. Eine Durchwahl würde jeweils ggf. mit einem Bindestrich angehängt. Bei den nächsten beiden Funktionen wird hingegen eine rein nummerische Ausgabe (ohne Leer- oder Pluszeichen) produziert, wie sie bspw. gut zur Übergabe an die Telefon-App eines Smartphones geeignet ist. Das dritte Funktionspaar generiert automatisch eine nationale oder internationale Darstellung in Abhängigkeit davon, ob es sich um eine Rufnummer mit der Ländervorwahl für Deutschland handelt. So wird je nach Nummer automatisch eine für uns übliche Darstellung erzeugt.

Zur komfortableren Abfrage der formatierten Ausgaben wurden analog zu den Tabellen für Telefon, Telefax und Webseite je eine Sicht (oder auch engl. View) in der Datenbank ergänzt, die jeweils die aufgesplitteten Werte durch die Ausgaben der oben genannten Funktionen ersetzen. Bei der Ausgabe der Webseiten-URL erfolgt die Zusammensetzung von Protokoll und Hostname direkt in der View ohne zusätzlichen Funktionsaufruf. Die fertigen Views sind in Abbildung 10 dargestellt:

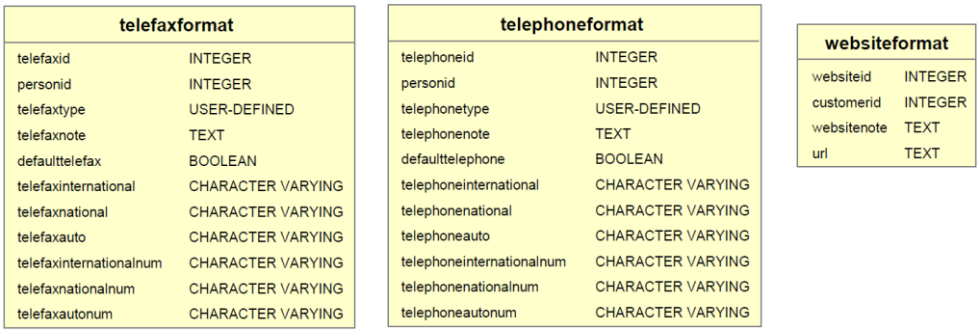

Abbildung 10: Views der Datenbank

Auch die erwähnten Funktionen sowie Definitionen der Views können dem Anhang F, Seite 72 entnommen werden.

5.1.4 ENUM-Datentypen

Für einige Spalten wurden eigene ENUM-Datentypen definiert, die eine feste Auswahl an Werten aufnehmen können, wie z. B. bei der Anrede „Herr“ oder „Frau“. So ist gewährleistet, dass in diesen

Spalten nur einer dieser konkreten Werte eingetragen werden kann. Nachstehend findet sich eine vollständige Auflistung der Datentypen-Definitionen:

- `T_TITLE` („Frau“, „Herr“)
- `T_PRONOMINALTITLE` („siezen“, „duzen“)
- `T_BILLINGDELIVERY` („E-Mail“, „Post“, „Telefax“)
- `T_TELEPHONETYPE` („privat“, „geschäftlich“, „mobil“)
- `T_TELEFAXTYPE` („privat“, „geschäftlich“)
- `T_EMAILTYPE` („privat“, „geschäftlich“)
- `T_PROTOCOL` („http“, „https“)
- `T_PAYMENTMETHOD` („Barzahlung“, „Überweisung“, „PayPal“)
- `T_CUSTOMERTYPE` („privat“, „gewerblich“)

5.2 JSON-Schnittstelle

Auch zur De- und Encodierung von Daten im JSON-Format bringt PHP eine fertige Erweiterung mit. [26] So stellte das Einlesen und Generieren von JSON-Daten serverseitig kein großes Problem dar. Im Folgenden soll der festgelegte Aufbau für die JSON-Objekte vorgestellt werden, die zur Übertragung zwischen Client und Server genutzt wird.

Generell kann mit JSON eine beliebig (durch Arrays und Objekte) verschachtelte Struktur von Datenobjekten übertragen werden, wie man sie von JavaScript kennt. Das beinhaltet neben dem allgemeinen JavaScript-„Objekt“ auch Arrays und Daten von den Typen String, Integer, Float und Boolean.

Zunächst gliedern sich die Client-Anfragen wie folgt in verschiedene Arten, die jeweils mit einem eindeutigen Kürzel versehen wurden:

- Benutzer-Anmeldung `a`
- Benutzer-Abmeldung `ad`
- Kunden durchsuchen `q`
- Personen (Kontaktpersonen) durchsuchen `qc`
- Details zu einer Person laden `p`
- Details zu einem Kunden laden `c`
- Details zu einem Kunden zur Bearbeitung laden `e`
- Details zu einer Person zur Bearbeitung laden `ep`
- Kunden-Datensatz speichern `s`
- Liste der Sitzungen laden `us`
- Einzelne Sitzung auflösen `ds`
- Benutzer-Passwort ändern `pw`
- Log-Nachricht speichern `l`

Die genannten Kürzel werden als Schlüssel im JSON-Objektstamm verwendet. Hieran kann der Server direkt den Typ der Anfrage erkennen und ggf. die weiteren Informationen, die in dem Objekt unterhalb des Schlüssels enthalten sind, passend weiter verarbeiten.

Hinter separaten Schlüsseln des JSON-Objekts stecken Informationen zum Client und dessen Versionsnummer (`cl` und `clV`). Außerdem werden die Authentifizierungsinformationen übermittelt (`auth`).

Dieses Konzept soll einmal anhand des Beispiels zum Durchsuchen der Kunden nach dem String „stalhut“ verdeutlicht werden. Der Client sendet eine Anfrage in der folgenden Form via POST des SSL-verschlüsselten HTTP-Protokolls an den Server.

Die Darstellung der folgenden JSON-Objekte wurde zur besseren Lesbarkeit um Leerzeichen, Zeilenumbrüchen und Einrückungen ergänzt. Tatsächlich handelt es sich um einen einfachen einzeiligen Text, der übertragen wird.

```
{
    "q": "stalhut",
    "cl" "Webclient",
    "clV": "0.6.2",
    "auth": {
        "username": "Jan",
        "password": "12345678"
    }
}
```

Daraufhin wird serverseitig die Datenbank nach der angegebenen Zeichenkette durchsucht und anschließend das folgende JSON-Objekt als Antwort auf seine HTTP-Anfrage an den Client zurückgesendet:

```
{
    "authOk": true,
    "business": [{
        "id": 3,
        "company": "IT-Services Jan Stalhut",
        "contact": "Herr Jan Stalhut"
    }],
    "private": [{
        "id": 1,
        "fullname": "Herr Jan Stalhut"
    }, {
        "id": 5,
        "fullname": "Herr Christian Stalhut"
    }]
}
```

In der Antwort ist zunächst eine Bestätigung zur korrekten Authentifizierung zu finden. Es folgen dann zwei Arrays (`business` und `private`), die jeweils eine Liste der Treffer im Geschäfts- und Privatkundenbereich enthalten. Dabei sind auch die jeweiligen Array-Elemente Objekte mit Schlüsseln für die Datenbank-ID, den Personen-Namen und ggf. den Firmennamen.

Die vollständige Definition der JSON-Schnittstelle für alle oben genannten Anfrage-Arten und den entsprechenden Antworten kann dem Anhang E (Seite 65) entnommen werden.

5.3 PHP-Skript

Die als PHP-Skript realisierte Schnittstelle zwischen JSON-Objekten und der Datenbank ist gemäß dem in 4.1 (Seite 13) erläutertem Software-Design realisiert. Der Quellcode des PHP-Programms ist in Anhang F (Seite 72) zu finden.

5.3.1 Logging

Um beim Auftreten von Fehlern diese zurückverfolgen zu können, wurde serverseitig eine Logging-Funktion implementiert, die jede Serveranfrage mit einem Zeitstempel und Informationen zum anfragenden Client in einer Logdatei speichert. Tagesweise wird dabei eine neue Logdatei angelegt, damit es später einfach möglich ist, alte nicht mehr relevante Logdateien vom Server zu löschen. Ergänzend ist es auch dem Client möglich, mit Hilfe des passenden Schlüsselworts im JSON-Objekt eine beliebige Textzeile an den Server zu senden, die dann auch unverändert in die Logdatei übernommen wird. Aus Sicherheitsgründen wurde diese Funktion des clientseitigen Sendens von Logdaten auf 100 Zeichen lange Zeichenketten beschränkt. Im Produktiv-Einsatz empfiehlt es sich jedoch ohnehin diese Funktion serverseitig zu deaktivieren, da sonst recht leicht unkontrolliert Daten von einem beliebigen Host in die Logdatei des Servers eingetragen werden können.

5.3.2 Error-Handling

Das Error-Handling wurde auf Serverseite dahingehend angepasst, dass Fehlermeldungen, die das PHP-Skript verursacht, in die oben erwähnte Logdatei geschrieben werden. Zusätzlich werden diese auch als JSON-Objekt an den Client weitergegeben, so dass dieser sie dem Benutzer präsentieren kann. Für den Produktiv-Einsatz ist dabei eine Variable in der Konfigurationsdatei (config.php) vorgesehen, mit der sich das Senden von detaillierten Fehlermeldungen an den Client deaktivieren lässt. Dies ist aus Sicherheitsgründen sinnvoll, da derartige Fehlermeldungen u. a. auch Details zur Datenbankstruktur preisgeben, die bei einem Angriff auf die Plattform einem Hacker hilfreiche Informationen liefern könnten.

5.3.3 Authentifizierung

Für jede Anfrage an den Server ist eine Authentifizierung des Benutzers notwendig, andernfalls antwortet der Server nur mit einer entsprechenden Antwort, dass die Authentifizierung fehlgeschlagen ist. Serverseitig werden hierbei prinzipiell zwei Möglichkeiten zur Verfügung gestellt: Das jeweilige Mitsenden der Benutzername-Passwort-Kombination und das Senden einer Session-ID mit zugehörigem Schlüssel. Bei dem Senden von Benutzername und Passwort kann der Client auch eine neue Session-ID einschließlich Schlüssel anfordern, die dann mit der Antwort an den Client gesendet wird. Der Einsatz der Sessions hat den Vorteil, dass das Speichern des Logins clientseitig möglich wird, ohne dass dort ein Benutzername mit einem persönlichen Kennwort im Klartext beim Client hinterlegt werden muss. Stattdessen wird nur die anonyme Session-ID mit einem automatisch generierten Zugangsschlüssel gespeichert. Erkennt der Server eine Session-Anmeldung mit falschem Schlüssel wird die betreffende Session-ID serverseitig umgehend als ungültig erklärt, während bei Benutzern lediglich die Fehllogins gezählt werden und dann nach 3 Fehllogins eine Sperrung des Benutzerkontos erfolgt. Zur Generierung des Session-Zugangsschlüssels wird die aktuelle Uhrzeit in Millisekunden mit einer Pseudozufallszahl kombiniert. Dazu bringt PHP die folgende Funktion mit:

```
string uniqid([string $prefix = "" [, bool $more_entropy = false]])
```

Im konkreten Einsatzfall, wird der erste Parameter nicht benötigt, jedoch der zweite für die Miteinbeziehung von Zufallszahlen auf *true* gesetzt.

5.3.4 Passwort Verschlüsselung

Sowohl die persönlichen Kennwörter von Benutzern als auch die Zugangsschlüssel für Sessions werden serverseitig in der bereits vorgestellten Datenbankstruktur (Abbildung 9, Seite 20) hinterlegt. Dabei wurde auch Serverseitig auf Sicherheit Wert gelegt, sodass diese Daten dort verschlüsselt abgelegt werden. Die klassische Möglichkeit dazu wäre das Bilden eines einfachen Hashes (z. B. md5), der dann gespeichert und bei einer Anfrage mit dem Hash des eingegeben Passworts verglichen wird. Inzwischen gilt dieses Verfahren jedoch nicht mehr als sicher, da es mit relativ geringem Aufwand möglich ist, aus dem md5-Hash das ursprüngliche Passwort zu ermitteln und man dabei bspw. auch gleiche Passwörter sofort am identischen Hash-Wert erkennt. [27]

Um ein sicheres Passwort-Verschlüsselungs-Verfahren zur Verfügung zu haben, wurde auf das existierende PHP-Framework phpass[7] zurückgegriffen. Hierin kommt ein auf bcrypt basierender Blowfish-Algorithmus zum Einsatz, der das Passwort wiederholt mit einem Salt kombiniert und einen Hash bildet. Die Anzahl der Durchgänge wird dabei als Potenz von 2 angegeben und mit in dem resultierenden verschlüsselten Passwort hinterlegt, sodass diese Information später beim Vergleich von Passwort und Hash zur Verfügung steht. Ein solcher Hash zu dem Passwort „12345678“ sieht dann bspw. wie folgt aus:

```
$2a$10$YDWl7KpNiXHYdMMVMITfXumeNHU7KUhW9nI.pdaOHzN36lZahfo6K
```

Er könnte für das gleiche Passwort jedoch auch z. B. folgendermaßen lauten, da durch den zufällig generierten Salt jeder Hash - auch bei gleichem Passwort - anders aussieht:

```
$2a$10$E8nxXKBwVcZSPIexqk/Az.UgMDoA13WLRapSKujoFZ2CR/cvQ/x/K
```

Je nach PHP-Version werden unterschiedliche Hash-Verfahren unterstützt, das Framework erkennt dies automatisch und verwendet die bestmögliche, was bei der hier eingesetzten PHP-Version 5.3 die erwähnte Blowfish-Methode ist. Auch die Information über das eingesetzte Hashverfahren wird in dem fertigen Hash hinterlegt. So kann auch in den obigen Hashes an den Ziffern zwischen den Dollar-Zeichen am Anfang erkannt werden, dass es sich um den Blowfish-Algorithmus handelte (`2a`) und 2^{10} (`10`) Iterationen durchlaufen wurden.

5.3.5 Überprüfung der Client-Version

Um später auch Änderungen an der Schnittstelle zu ermöglichen, ohne dass es mit verschiedenen Versionen bei Client und Server zu unvorhersehbaren Fehlern kommt, wurde serverseitig eine Überprüfung der Versionsnummer implementiert. Mit jeder Client-Anfrage wird auch die Versionsnummer des Clients übermittelt. Diese wird dann mit der serverseitig eingesetzten Version verglichen. Die Versionsnummer gliedert sich in Hauptversionsnummer, Nebenversionsnummer und Revisionsnummer, die jeweils mit einem Punkt getrennt angegeben werden. Die aktuell gültige Versionsnummer lautet 0.5.x. Bei dem x handelt es sich um die erwähnte Revisionsnummer, die bei verschiedenen Clients unterschiedlich sein kann, ohne dass sich Änderungen an der Serverschnittstelle ergeben. Sobald jedoch Änderungen an der JSON-Schnittstelle notwendig werden, ist die Nebenversionsnummer zu erhöhen, was dann bei unterschiedlichen Client- und Server-Angaben zu einer Fehlermeldung führt, die an den Client gesendet wird: Der Benutzer wird aufgefordert seinen Client zu aktualisieren.

[7] Weiter Informationen und Download: http://www.openwall.com/phpass/

5.4 Testen

Auf reine Tests der Serveranwendung wurde an dieser Stelle verzichtet, da die Entwicklung der Serveranwendung praktisch parallel zu dem im folgenden Kapitel vorgestellten Webclient stattfand, wurde bei den Tests des Webclients gleichzeitig auch die Serveranwendung mitgetestet.

6 Webclient

Der Webclient wurde gemäß dem in 4.2 (Seite 14) vorgestellten Design mit den drei Schichten (HTML, CSS und JavaScript) realisiert. Der vollständige Quellcode des Webclients ist in Anhang F (Seite 72) zu finden. Eine Live-Demo des Webclients kann unter der Adresse http://cdb.stalhut.de/example/ aufgerufen werden. Dazu kann als Benutzername „********“ mit dem Passwort „********“ verwendet werden.

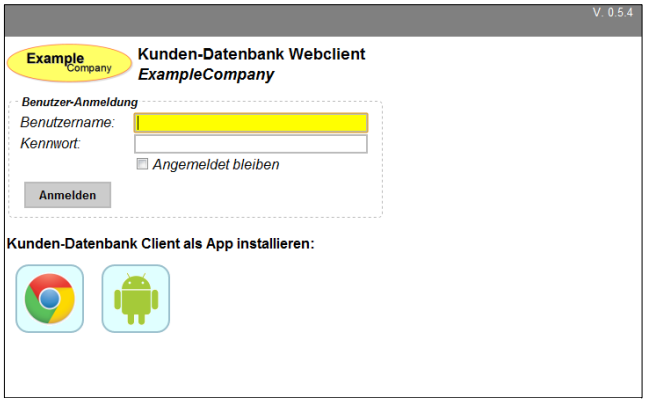

Abbildung 12: Loginformular des Webclients

Abbildung 11: Loginformular des Webclients unter Android

6.1 Layout mit CSS

Bei dem Erstellen des Layouts mit CSS kamen auch aktuelle Techniken von CSS3 zum Einsatz. Dazu gehören die runden Ecken bei den Umrandungen der Formularfelder und Popups und die halbtransparente Fläche, die den Hintergrund bei einem offenen Popup abdunkelt. Des Weiteren bietet sich durch CSS die Möglichkeit, verschiedene Darstellungen für unterschiedlich große Bildschirme (genau genommen unterschiedliche Browser-Fenster) festzulegen. So passt sich die oben in Abbildung 12 auf dem PC geöffnete Login-Seite auf dem Smartphone durch CSS-Anweisungen entsprechend an und sieht dann bspw. im Android™-Browser aus wie in Abbildung 11 (Die abweichende Schrift-Art ist an dieser Stelle durch den Android™-Browser bedingt, der alle Schriften durch den im System festgelegten Standard-Schrifttyp ersetzt.) bzw. auf dem iPhone mit iOS 5 wie in Abbildung 13. Eine solche Beschränkung von CSS-Anweisungen auf eine bestimmte Anzeigebreite erfolgt mit Hilfe eines CSS-Blockes wie dem folgenden:

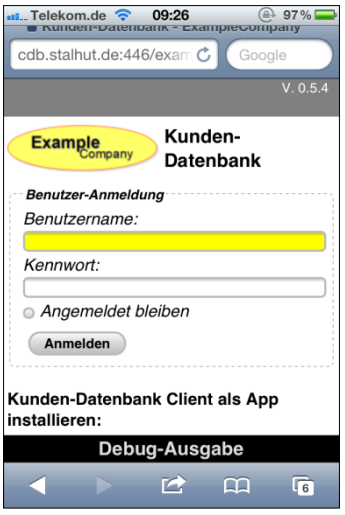

Abbildung 13: Loginformular des Webclients in Safari unter iOS 5

```
@media all and (min-width: 450px){
        /* weitere CSS-Anweisungen folgen hier */
}
```

Im obigen Beispiel würden die CSS-Anweisungen innerhalb des Blockes nur ausgeführt, wenn der Anzeigebereich mindestens 450 Pixel breit ist. Ergänzend wäre an dieser Stelle auch die Abfrage auf die Höhe des Anzeigebereiches sowie Breite und Höhe des Gerätes (Monitors) möglich. Außerdem steht auch noch das Schlüsselwort `orientation` zur Verfügung, mit dem sich – speziell für mobile Geräte – die aktuelle Ausrichtung (Querformat oder Hochformat) mit in die Bedingung einbeziehen lässt. Über das im Beispiel verwendete Schlüsselwort `all` wird zusätzlich festgelegt, dass die Angaben für alle Arten von Ausgaben gelten soll. Als Alternative wäre hier z. B. `only screen` einzusetzen, damit die

Angaben u. a. nicht beim Ausdrucken der Seite Anwendung finden. Das Pendant dazu wäre das Schlüsselwort `print` – speziell für die Druckausgabe.

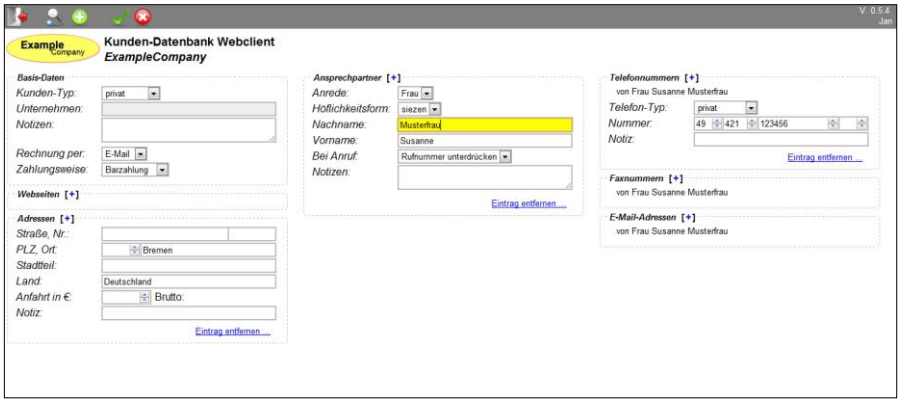

Abbildung 14: Webclient mit dem Formular zur Eingabe eines neuen Kunden

Auch wurde es mit Hilfe solcher CSS-Anweisungen realisiert, damit sich das recht umfangreiche Formular – zum Einfügen eines neuen Kunden bzw. zum Bearbeiten eines vorhandenen Kunden –bei entsprechender Bildschirmbreite auf bis zu vier Spalten verteilt. Auf dem Smartphone wiederum wird es in einer einzelnen langen schmalen Spalte dargestellt. Dieses Formular ist in Abbildung 14 in der Variante mit drei Spalten zu sehen.

Abbildung 15: Webclient-Ausgabe bei nicht unterstütztem Webbrowser

6.2 Browser-Unterstützung

Der Webclient testet beim Aufruf, ob der aktuelle Browser die notwendigen Funktionen unterstützt, indem mit JavaScript versucht wird auf den mit HTML5 eingeführten *Local Storage* zuzugreifen. Ergänzend wird noch der Microsoft Internet Explorer 8 ausgeschlossen, der zwar diese HTML5 Funktionalität bietet aber bspw. nicht die gewünschten per CSS formatierten runden Ecken darstellen kann. Wenn diese Überprüfung einen nicht kompatiblen Browser feststellt, wird statt dem Login dann die in Abbildung 15 sichtbare Hinweismeldung ausgegeben, die den Benutzer über kompatible Webbrowser für verschiedene Betriebssysteme informiert.

6.3 Serverkommunikation

Beim Aufruf des Webclients und später beim (asynchronen) Nachladen von Daten wird dem Benutzer ein Ladebalken eingeblendet, um diesen über die aktuellen ansonsten unsichtbaren Aktivitäten zu informieren. Zur Kommunikation mit dem Server wird dann ein JavaScript-Objekt vom Typ *XMLHttpRequest* erzeugt. Auf dessen ältere Implementierung des Internet Explorers in Form eines ActiveX-Objektes brauchte an dieser Stelle keine Rücksicht mehr zu genommen werden, da ja ohnehin der Internet Explorer in Version 9 vorausgesetzt wird, bei dem dieser „Umweg“ nicht mehr nötig ist.

Die in Form eines normalen JavaScript-Objekts vorliegenden zu übertragenden Daten werden mit Hilfe des JSON-Parsers *json2.js*[8] in JSON-Strings umgewandelt und als solche an den Server geschickt. Mit dem gleichen Parser erfolgt dann anschließend auch die Umwandlung der Antwort – in Form eines JSON-Strings – zurück in ein JavaScript-Objekt. Letztere Aufgabe ließe sich zwar auch mit einem einfachen Aufruf der Standard Funktion `eval(str)` realisieren, die den übergebenen String als

[8] Download unter: https://github.com/douglascrockford/JSON-js

JavaScript-Code interpretiert. Aus Sicherheitsgründen sollte hier jedoch eine Filterung stattfinden, damit nicht auch beliebige JavaScript-Funktionen im Kontext der Webseite auf diese Weise ausgeführt werden. Und genau dies wird von dem genannten JSON-Parser berücksichtigt.

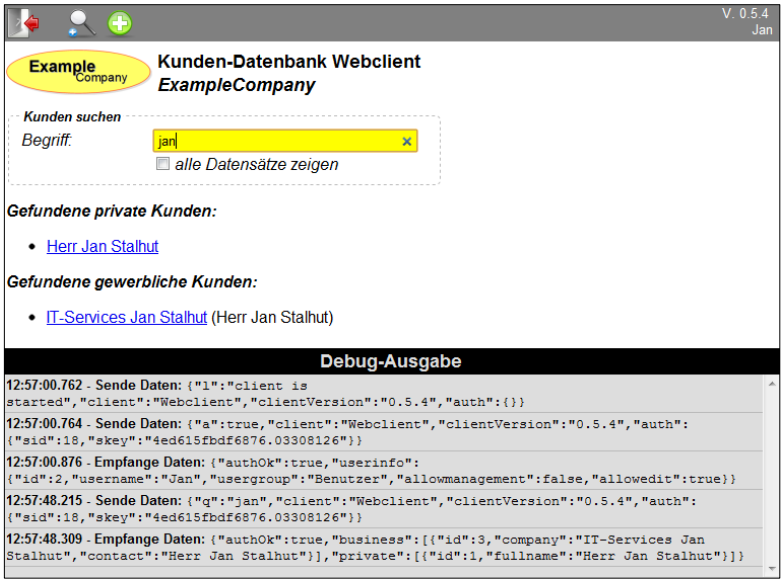

Abbildung 16: Webclient mit eingeblendeter Debug-Ausgabe nach dem Login eines Benutzers und dem Start der Suche

Zur besseren Nachvollziehbarkeit während der Entwicklung und des Testens wurde eine Debug-Ausgabe eingebaut, die sich vom unteren Bildschirmrand her einblenden lässt. Im Produktivbetrieb kann diese später mit Hilfe einer Konfigurations-Variablen im JavaScript-Code deaktiviert werden. Hier werden jedoch alle gesendeten und empfangenen JSON-Strings zur Kontrolle zusammen mit einem Zeitstempel ausgegeben. Wie dies dann bspw. nach dem Login und der Such nach „jan“ aussieht, ist in Abbildung 16 zu sehen.

6.4 An- und Abmeldung

Für die Anmeldung von Benutzern gibt es wie bereits serverseitig zwei Möglichkeiten: Die Nutzung einer Session ID mit passendem Schlüssel oder das Mitsenden von Benutzernamen und Passwort bei jeder Anfrage. Bei dem Webclient hat der Benutzer bei der Anmeldung die Wahl zwischen diesen beiden Varianten, indem er sich entscheiden kann, angemeldet zu bleiben oder nicht. Entscheidet dieser sich dafür nicht angemeldet zu bleiben, wird clientseitig der Benutzername und das Passwort in einer JavaScript-Variablen, folglich lediglich im Arbeitsspeicher des PCs, gespeichert und damit automatisch verworfen, sobald der Benutzer die Weboberfläche verlässt. Entscheidet er sich hingegen für die Option „Angemeldet bleiben“, wird vom Server eine Session-ID mit Schlüssel angefordert. Diese Daten werden dann clientseitig im *Local Storage* des Webbrowsers gespeichert und können dort auch einen Browser- und PC-Neustart überdauern. Beim Aufruf des Webclients wir daher zunächst geprüft, ob derartige Anmeldeinformationen in *Local Storage* hinterlegt sind. Ist das der Fall, werden diese direkt zur Überprüfung an den Server gesendet, ehe anschließend direkt das Formular zur Kundensuche erscheint. Sollten die Daten vom Server nicht (mehr) akzeptiert werden, wird der Benutzer darüber informiert, dass seine Session abgelaufen ist und er sich bitte neu anmelden möchte. Sind hingegen keine Session-Daten hinterlegt, gelangt der Benutzer direkt zum Login-Formular des Webclients.

6.5 Kunden-Suche

Auf der Seite zur Kunden-Suche erfolgt bereits mit der Eingabe des zweiten Buchstabens die Suche nach passenden Kunden-Datensätzen, indem der

Abbildung 17: Webclient-Suche in Aktion

eingegebene String an den Server gesendet wird. Der dann wiederum eine Trefferliste, getrennt nach Privat- und Geschäftskunden, zurückliefert (siehe Abbildung 17). Um unnötige Serverlast und Traffic zu vermeiden, wenn bei jedem Tastenanschlag eine Anfrage an den Server gesendet würde, wird vor dem Absenden der Anfrage noch 0,7 Sekunden gewartet, ehe die Anfrage tatsächlich an den Server geschickt wird. Die Ausgabe der Suchergebnisse erfolgt dann getrennt nach Privat- und Geschäftskunden, wobei sich alle Treffer entsprechend anklicken lassen um zu den Details zum Kundendatensatz zu gelanden. Bei Geschäftskunden erfolgt neben der Anzeige vom Namen des Unternehmens auch direkt der vollständige Name einer zugeordneten Kontaktperson.

6.6 Dateneingabe

Für die Eingabe der Daten steht ein recht umfangreiches Formular zur Verfügung, das sich, wie bereits erwähnt, auch dynamisch an einen größeren Monitor anpassen kann, sodass der bestmögliche Komfort bei der Daten-Eingabe bei einem vierspaltigen Layout auf einem Monitor mit einer Auflösung von mind. 1920x1080 Pixel (FullHD) gegeben ist. Durch das zusätzliche Hervorheben des gerade fokussierten Formular-Elements in Gelb ist für den Benutzer ohne lange Suche erkennbar, wo sich gerade der Eingabe-Cursor befindet.

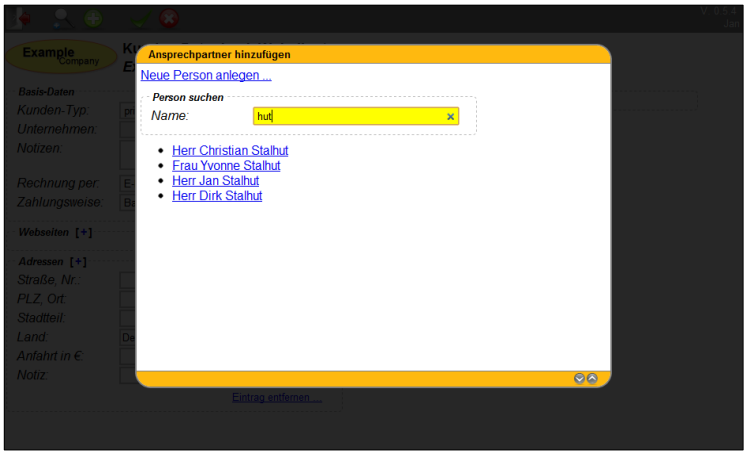

Abbildung 18: Webclient mit dem Popup zum Hinzufügen einer Person zu einem Kundendatensatz

Bedingt durch die Anforderung, dass es mehrere Personen zu einem Kunden geben kann und eine Person mehreren Kundendatensätzen zugeordnet werden können soll, muss beim Hinzufügen einer Person zu einem (neuen) Kundendatensatz der Benutzer entscheiden, ob eine bereits vorhandene Kontaktperson auch mit diesem Kunden verknüpft werden soll oder ein komplett neuer Personen-Datensatz anzulegen ist. Dazu erscheint dann ein Popup, wie es in Abbildung 18 zu sehen ist. Darin kann dann entweder mit einem Klick ein neuer leerer Kundendatensatz verwendet oder mit Hilfe einer Personen-Suche ein bereits vorhandener Kunde herausgesucht und ausgewählt werden.

Weitere mit JavaScript implementierte Funktionen sind die automatische Berechnung des aktuellen Brutto-Wertes vom eingegebene Netto-Betrag für die Anfahrtspauschale zu einer Anschrift und das dynamische Wechseln zwischen den verschiedenen Formularfelder-Sätzen für Kontaktinformationen der einzelnen Kontaktpersonen: Da es für jede Person separate Telefonnummern, Faxnummern und E-Mail-Adressen geben kann und es zu viel wäre, bei mehreren Ansprechpartnern zu jedem diese zahlreichen Formularfelder anzuzeigen, wurde sich dafür entschieden, dass automatisch die Kontaktdaten-Formularfelder zu der Person eingeblendet werden, dessen Formular-Felder den Fokus erhalten. Das bedeutet, wenn der Benutzer in das Namensfeld von Kontaktperson A klickt, um dort etwas einzutragen, werden automatisch die Felder für Telefonnummer usw. dieser Person eingeblendet und die anderer Personen ausgeblendet. Sobald der Fokus dann bspw. zur Anrede-Auswahl der Person B wechselt, werden dessen Kontaktdaten-Formularfelder sichtbar. Um diesen Wechsel für den Benutzer übersichtlich zu halten, wird der Name der Person aus den Feldern Anrede, Nachname und Vorname zusammengesetzt und über den zugehörigen Kontaktdaten-Feldern eingeblendet.

Auch für die Formularfelder wurden die neuen Funktionen von HTML5 eingesetzt, sodass es nicht nur Formularfelder vom klassischen Typ `text` gibt, sondern auch speziellere wie `email`, `number` und `url`. Dies sorgt je nach Unterstützung der Webbrowser u. a. für eine Validierung der Benutzereingaben, sodass bspw. der Browser Chrome in einem als Typ `number` deklarierten Eingabefeld nur (Fließkomma-)Zahlen zulässt und alle anderen Zeichen direkt wieder entfernt. Oft zeigen die Browser dann auch am rechten Rand eines solchen Eingabefeldes kleine Pfeile an, mit denen der Benutzer die Zahl durch Klicken inkrementieren und dekrementieren kann. Dies ist zwar in diesem Fall (z. B. bei Telefonnummern) nicht wirklich hilfreich, lässt sich jedoch auch derzeit nicht gezielt deaktivieren. Zusätzlich sorgen diese speziellen Typen-Angaben dafür, dass mobile Webbrowser bei Android und insbesondere beim iPhone das Tastatur-Layout einer On-Screen-Tastatur entsprechend anpassen und beispielsweise beim Typ `email` das @-Zeichen an prominenter Stelle unterbringen statt in der zweiten Ebene, oder bei `number`-Eingabefeldern eine reine Ziffern-Tastatur einblenden. Für Browser, die diese Typen nicht unterstützen, stellt es auch kein wirkliches Problem dar, da diese dann einfach vom Standard-Typ `text` ausgehen.

Abbildung 19: Webclient-Ausgabe der Kunden-Details

6.7 Datenausgabe

Abbildung 20: Webclient-Ausgabe der Personen-Details

Die Datenausgabe erfolgt im Wesentlichen in zwei Ansichten. Zunächst werden die Details zu einem Kunden aufgeführt (siehe Abbildung 19) und zugeordnete Personen erst einmal nur namentlich aufgelistet. Hier lässt sich bereits die Adresse anklicken, um so direkt Google Maps zu öffnen, worin die Adresse herausgesucht und markiert wird. Auch Webseiten, sofern angegeben, können auf der Detailseite eines Kunden direkt angeklickt werden, um die Webseite anzuzeigen. Für all diese Funktionen wurden JavaScript Funktionen geschrieben, die jeweils beim Klicken mit entsprechenden Parametern aufgerufen werden. Sollte man dann einmal generell bspw. alle Adressen statt mit Google Maps mit Microsofts Bing Maps anzeigen wollen, wäre nur die Funktionalität dieser JavaScript-Funktion entsprechend anzupassen.

Die Kontaktpersonen lassen sich dann wiederum anklicken um auf die andere Detail-Ansicht zu gelangen, auf der dann die Kontaktmöglichkeiten zu dieser Person aufgezeigt werden, wie es auf Abbildung 20 zu sehen ist.

Auch hier lässt sich dann wiederum jede hinterlegte E-Mail-Adresse anklicken, um mit dem als Standard definierten Mail-Programm eine E-Mail an diese Mail-Adresse zu verfassen. Ansonsten folgt hier wiederum eine Liste der zugeordneten Kunden-Datensätze, die sich anklicken lassen um wiederum zu der Detail-Seite des jeweiligen Kunden zu gelangen.

Auf den Detailseiten steht jeweils ein Link bereit, um zu der letzten Suche zurückzukehren, wenn man bspw. bei mehreren Suchergebnissen nicht sicher ist, kann man so recht schnell jeweils einzelne Kunden-Details aufrufen. Auf der Personen-Detailseite wurde zusätzlich ein Link implementiert, der zurück zum zuletzt angezeigten Kunden führt.

6.8 Benutzerverwaltung

Klickt der Benutzer oben rechts im Webclient auf seinen eigenen Benutzernamen, erscheint auch ein Popup (siehe Abbildung 21). In diesem wird dem Benutzer zum einen die Möglichkeit gegeben, sein eigenes Kennwort zu ändern und zum anderen werden die derzeit aktiven Sitzungen an verschiedenen Geräten aufgelistet. So bekommt der Benutzer einen Überblick über die aktiven Sitzungen und hat auch die Möglichkeit einzelne Sitzungen zu beenden. Letzteres könnte z. B. interessant sein, wenn man auf seinem Android™-Smartphone mit der Client-App eingeloggt ist, aber sein Telefon verlegt hat. Damit dann niemand anderes, der das Smartphone in die Hände bekommt, Zugriff auf die Kundendaten erhält, kann hier die entsprechende Session über den PC gelöscht werden. Wird später versucht mit der ID einer gelöschten Session auf den Server zuzugreifen, erhält der Benutzer die Info, dass die Sitzung abgelaufen sei und er sich neu anmelden müsse.

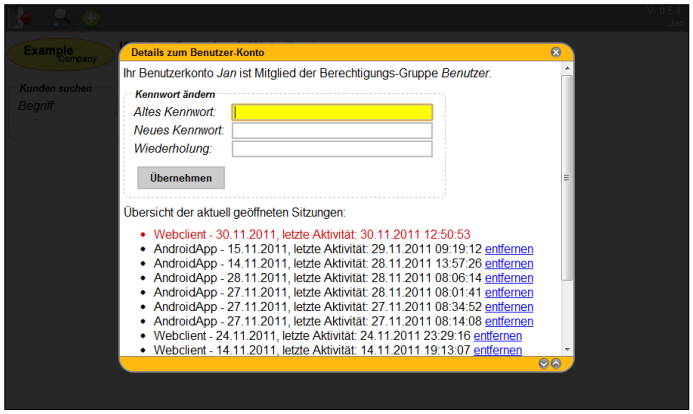

Abbildung 21: Webclient mit dem Popup zur Passwort-Änderung und Session-Verwaltung

6.9 Datenhaltung

Durch die Verwendung der HTML5-Technik, die den Local Storage im Webbrowser ermöglicht, ist der Einsatz von Cookies überflüssig. Letztlich sind ohnehin die einzigen Daten, die clientseitig gespeichert werden, jene, die zur Speicherung der Anmeldedaten (Session-ID und Session-Key) verwendet werden. Alle anderen Daten werden bei jeder Anfrage im Hintergrund vom Server geladen, so ist gewährleistet, dass der Benutzer stets den aktuellen Stand der Daten zu Gesicht bekommt.

6.10 Anpassungen für mobile Browser

Speziell für mobile Browser wurden ein paar Anpassungen vorgenommen. Außer den generellen Layout-Anpassungen an kleineren Displays, wie bereits in 6.1 erläutert, mussten den Popups explizit Buttons zum Rauf- und Runterscrollen hinzugefügt werden. Das war notwendig, da aktuelle Browser zum Scrollen bekanntermaßen keine Scroll-Balken anzeigen, sondern durch eine entsprechende wischende Touchscreen-Geste auf einer Webseite gescrollt wird. Dies wird jedoch (mit Ausnahme des Safari-Browsers in Apples iOS 5) nicht in inneren HTML-Elementen unterstützt, sondern immer die ganze Seite gescrollt. So war es ergänzend auch notwendig, das Scrollen per Touchscreen-Geste per JavaScript zu unterbinden, solange ein Popup sichtbar ist. Für derartige Dinge stehen in JavaScript die Event-Handler *window.ontouchmove* und *window.ontouchend* zur Verfügung.

Zusätzlich wurde sich aus optischen Gründen dafür entschieden, die fixierten Elemente (Toolbar am oberen Rand und ggf. Debug-Ausgabe am unteren Rand) – mit Hilfe der oben genannten Event-Handler – auszublenden, solange ein Scroll-Vorgang per Touch-Geste aktiv ist. Da mobile Browser auch bei diesen per CSS als *fixed* markierten Elemeten mit der Darstellung nicht nachkommen. Obwohl diese beim Scrollen normalerweise stehenbleiben sollen, wandern diese beim Scrollen in mobilen Browsern zunächst mit und „springen“ dann am Ende des Scroll-Vorgangs erst wieder an die korrekte Stelle.

6.11 Chrome-App

Viele der derzeit so zahlreich verfügbaren Chrome-Apps kann man eigentlich eher als etwas besseres Lesezeichen bezeichnen, da sie lediglich eine Weiterleitung zu einer bestimmten Adresse enthalten. Es ist jedoch auch möglich HTML-, CSS-, JavaScript- und Grafik-Dateien zusammen in einem Paket als App für Google Chrome (und kompatible Browser, wie z. B. Iron) bereitzustellen. Eine installierte App wird dem Benutzer, wie in Abbildung 22 zu sehen, typischerweise zur Auswahl angeboten, sobald ein neuer Tab geöffnet wird.

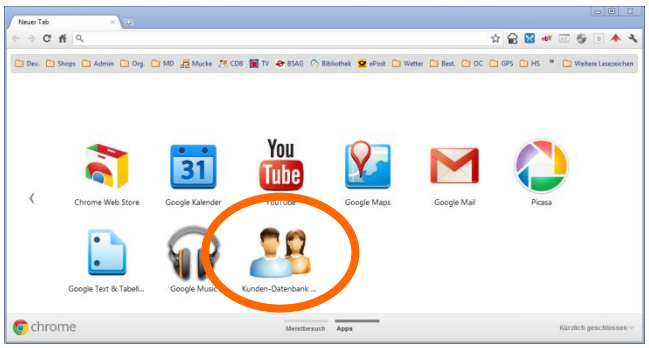

Abbildung 22: Ansicht beim Öffnen eines neuen Tabs in Google Chrome mit installierter App

Durch den gewählten Ansatz, dass nur beim Aufruf des Webclients alle notwendigen HTML, CSS und JavaScript-Dateien vom Server geladen werden und jede weitere Server-Kommunikation per JavaScript realisiert ist, war es mit geringem Aufwand möglich, aus dem Webclient eine lokal laufende Chrome-App zu erstellen. Dazu musste lediglich eine spezielle Manifest-Datei im JSON-Format erstellt werden. Diese Datei enthält u. a. einen Titel (`name`), eine Beschreibung (`description`) und eine Versionsbezeichnung (`version`) der App sowie Verweise auf Grafikdateien verschiedener Größe (`icons`), die dann als Symbol der App verwendet werden. Die fertige *manifest.json* sieht dann wie folgt aus:

```
{
	"name": "Kunden-Datenbank der ExampleCompany",
	"description": "Kunden-Datenbank ChromeClient - Zum komfortablen Zugriff " +
		"auf die Kunden-Datenbank der ExampleCompany",
	"homepage_url": "http://cdb.stalhut.de/",
	"version": "0.6.0",
	"icons": {
		"128": "icon/mainicon_128.png",
		"48": "icon/mainicon_48.png",
		"16": "icon/mainicon_16.png"
	},
	"permissions": [
		"https://cdb.stalhut.de:446/"
	],
	"app": {
		"launch": {
			"local_path": "index.htm"
		}
	},
	"update_url": "https://cdb.stalhut.de:446/example/chromeAppUpdate.xml"
}
```

Abbildung 23: Bestätigung zur Installation der ChromeApp

Eine weitere Angabe, die in der JSON-Datei zu finden ist, sind Berechtigungen, die dem Benutzer bei der Installation mit der Bitte um Genehmigung gezeigt werden (siehe Abbildung 23). In diesem Beispiel ist hier eine Serveradresse angegeben, mit der die App später kommunizieren möchte. Schließlich kann noch, wie auch hier geschehen, optional eine Adresse zu einer speziellen XML-Datei angegeben werden, in der Informationen zur neusten Version hinterlegt sind (`update_url`), damit die App sich ggf. automatisch aktualisieren kann. Diese XML-Datei mit Update-Informationen sieht dann wie folgt aus:

```
<?xml version='1.0' encoding='UTF-8'?>
<gupdate xmlns='http://www.google.com/update2/response' protocol='2.0'>
  <app appid='ompknnhkhmhonmggikamjicoiieklgnc'>
    <updatecheck
        codebase='https://cdb.stalhut.de:446/example/CDB-ChromeClient.crx'
        version='0.6.0' />
  </app>
</gupdate>
```

Hier ist neben der Versions-Angabe und dem Download-Link zum crx-Paket der neusten Version auch eine eindeutige ID der App hinterlegt, die beim ersten Erstellen des Pakets erzeugt wird und dann diese App eindeutig identifiziert.

Zur Erstellung des Installationspakets in Form einer crx-Datei müssen lediglich alle zugehörigen Dateien sowie die manifest.json in einem Verzeichnis liegen. In einer gewöhnlichen Chrome-Installation ist dann in der Liste der Erweiterungen zur Entwickleransicht zu wechseln, wo man dieses Verzeichnis zum „Packen einer Erweiterung“ auswählen kann. Zum Testen der App kann hier auch das Verzeichnis direkt als App eingebunden werden.

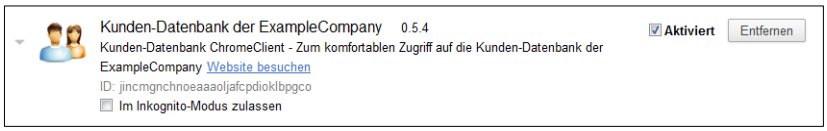

Abbildung 24: Darstellung der Chrome-App in der Liste der installierten Erweiterungen

Ist die Anwendung installiert, taucht sie dann beim Benutzer in der Liste der installierten Erweiterungen auf (siehe Abbildung 24) und kann dort bei Bedarf auch wieder deinstalliert oder nur vorübergehend deaktiviert werden.

Damit die Chrome-App sich serverseitig (in den Logdateien) erkennen und vom normalen Webclient unterscheiden lässt, wurde als einzige Änderung in der *config.js* der Name „WebClient“ in „ChromeApp“ geändert. In der Initialisierungs-Funktion sorgt dies dann auch dafür, dass die Hinweise auf die App-Downloads auf der Login-Seite nicht mehr auftauchen und auch im Titel der Seite statt „WebClient“ dann „ChromeApp“ steht.

6.12 Testen

Wie bereits in 1.4 (Seite 2) erwähnt, soll das Testen nicht Hauptthema dieser Bachelor-Thesis sein. Daher soll hier nur kurz das Vorgehen erläutert werden. Während der Entwicklung wurde überwiegend mit dem Webbrowser Firefox gearbeitet und darin einzelne Funktionen der Weboberfläche getestet. Insbesondere die korrekte Darstellung wurde zusätzlich auch in den anderen gängigen und in den Anforderungen genannten Webbrowsern getestet. Die finale Testfolge, wie sie in jedem (mo-

bilen) Browser manuell abgearbeitet wurde, sieht dabei grob dargestellt wie folgt aus: (Diese setzt die Testdaten in der Datenbank voraus, wie sie auch in Anhang F zu finden sind.)

1. Laden der genannten sql-Dateien in die Datenbank, die eine neue leere Tabellen-Struktur anlegen und die Beispieldaten einfügen.
2. Aufruf des Webclients → nach kurzer Sichtbarkeit der Anzeige, dass die Seite geladen wird ein leeres Login-Formular
3. Anmeldungs-Versuch mit einem nicht existenten Benutzernamen → Fehlermeldung und „neues“ leeres Login-Formular
4. Anmeldung mit korrektem Benutzernamen aber falschem Passwort → Fehlermeldung, Erhöhung des entsprechenden Zählers des Benutzers in der Datenbank und „neues“ leeres Login-Formular
5. Anmeldung mit korrekten Zugangsdaten → Anzeige des Benutzernamens in der oberen rechten Ecke und Sichtbarkeit des leeren Suchformulars
6. Eingabe der Buchstabenfolge „abc“ → Anzeige „keine Treffer“ und Suchbegriff und Cursor bleiben im Suchfeld stehen
7. Logout des Benutzers mit Symbol in der oberen linken Ecke → leeres Login-Formular
8. Anmeldung mit korrektem Zugangsdaten bei aktivierter „Angemeldet bleiben“ Option → Anzeige des Benutzernamens in der oberen rechten Ecke und Sichtbarkeit des leeren Suchformulars
9. Beenden des Webbrowsers, erneuter Start und Aufruf des Webclients → Anzeige des Benutzernamens in der oberen rechten Ecke und Sichtbarkeit des leeren Suchformulars
10. Eingabe der Zeichenfolge „tal“ → zwei Privatkunden gefunden (Christian und Jan Stalhut)
11. Klick auf „Herr Christian Stalhut“ → Anzeige von Details zum Kunden und Auflistung der zwei Ansprechpartner (Herr und Frau Stalhut)
12. Klick auf die dargestellte Andresse → Google Maps öffnet sich in einem neuen Fenster (oder Tab je nach Browser) und zeigt die Adresse auf der Karte.
13. Klick auf den Ansprechpartner „Herr Christian Stalhut“ → Anzeige der verschiedenen Kontaktdaten (Telefonnummer, E-Mail-Adresse) und Auflistung der zugeordneten Kunden-Datensätze (hier nur „(privat)“)
14. Klick auf die E-Mail-Adresse → Standard Mail-Programm des Systems öffnet das Fenster zum Verfassen einer neuen E-Mail und bereits eingetragenem Empfänger.
15. Klick auf „zurück zum Kundendatensatz“ → erneute Anzeige von Details zum Kunden
16. Klick auf das Bearbeiten-Symbol der oberen Leiste. → Formular zum Bearbeiten der Daten erscheint, in dem aktuellen Daten des Kunden stehen bereits in den Feldern
17. Eintrag von „test“ im Feld „Notizen“ der „Basis-Daten“ und Klick auf das „Änderungen verwerfen“ Symbol → Anzeige von Details zum Kunden nach wie vor mit dem Hinweis „keine Notiz hinterlegt“
18. Klick auf das Bearbeiten-Symbol der oberen Leiste. → Formular zum Bearbeiten der Daten erscheint erneut mit eingetragenen Daten
19. Änderung einiger Details (Notizen des Kunden und der Person, Anfahrtspauschale, Entfernen eines Ansprechpartners, Hinzufügen einer Webseite, Änderung der Telefonnummer des anderen Ansprechpartners. → die Brutto-Anfahrtspauschale wird aktualisiert auf Basis der eingegebenen Netto-Anfahrtspauschale, Formularfelder für Kontaktdaten des entfernten Ansprechpartners sind nicht mehr sichtbar.

20. Klick auf das „Änderungen speichern“ Symbol → Anzeige der Kunden-Details mit den durchgeführten Änderungen, einschließlich korrekt berechneter Brutto-Anfahrtspauschale
21. Klick auf den einen verbliebenen Ansprechpartner → Anzeige der veränderten Telefonnummer und Notiz.
22. Klick auf das „Kunden hinzufügen“ Symbol → Popup zum Hinzufügen einer Person erscheint über dem ausgegrauten Formular zur Dateneingabe, Cursor steht im Suchfeld
23. Klick auf den Link „neue Person anlegen“ → Popup verschwindet, ein leeres Formular zur Eingabe der Daten für einen neuen Kunden einschließlich eines Ansprechpartners
24. Klick auf das + beim Ansprechpartner → Popup zum Hinzufügen einer Person erscheint über dem ausgegrauten Formular zur Dateneingabe, Cursor steht im Suchfeld
25. Eingabe der Zeichenfolge „dorf“ → Auflistung der Personen „Frau Christiane Wachtendorf“ und „Frau Gisela Dorfschmidt“
26. Klick auf den Namen „Frau Gisela Dorfschmidt“ → Popup verschwindet, Formularfelder mit dieser Kontaktperson und den hinterlegten Kontaktdaten werden im Formular eingeblendet.
27. Klick auf „Eintrag entfernen“ beim ersten Ansprechpartner → Die zur Kontaktperson gehörenden Formularfelder verschwinden
28. Klick auf das „Änderungen speichern“ Symbol → der neu angelegte Kunden-Datensatz wird angezeigt mit der Standard Adressangabe Bremen, Deutschland und Anfahrt von 10,08 € netto bzw. 12,00 € brutto.
29. Klick auf „Datensatz bearbeiten“ → Formular erscheint mit den vorhandenen Daten, Felder für Kunden-Typ, Ort, Land, Anrede, Vor- und Nachname sind ausgegraut.
30. Ergänzung der PLZ „12345“, Klick auf das „Änderungen speichern“ Symbol → die Detail-Seite erscheint erneut, nun mit der ergänzten PLZ.
31. Klick auf „zurück zur Suche“ → Anzeige des Suchformulars mit noch sichtbarer letzten Eingabe „tal“ und Anzeige der entsprechenden Treffer
32. Klick auf das „neue Suche“ Symbol → leeres Formular zur Eingabe eines Suchbegriffs, keine Treffer mehr sichtbar, Cursor steht im Suchfeld.
33. Aktivierung der Option „alle Datensätze zeigen“ → Eintrag des Schlüsselworts „ALLITEMS“ im nun deaktivierten Suchfeld, Auflistung aller 5 Privatkunden und der 2 gewerblichen Kunden.
34. Login am Webclient mit einem anderen Webbrowser aber dem gleichen Benutzer und Klick auf den Benutzernamen oben rechts → Auflistung der aktiven Sessions listet die aktive Session des vorher genutzten Browsers auf.
35. Klick auf „entfernen“ bei der betreffenden Session, schließen des Browsers und zurück zum anderen Browser → Es wird unverändert die Liste aller Kunden-Datensätze angezeigt
36. Klick auf einen der angezeigten Kunden → Meldung, dass die Session abgelaufen ist. Anzeige des Login-Formulars.
37. Login mit einem Benutzer, der keine Bearbeitungsrechte hat → leeres Formular zur Eingabe eines Suchbegriffs, Symbol zum Hinzufügen eines neuen Benutzers ist nicht sichtbar.
38. Suche nach „jan“ → Treffer „Herr Jan Stalhut“ erscheint
39. Klick auf den angezeigten Treffer → Anzeige von Details zum Kunden, Symbol zum Bearbeiten des Datensatz ist nicht sichtbar.

7 Android™-App

Die Android™-App wurde nach den im Abschnitt 4.3 (Seite 15) festgelegten Software-Design umgesetzt. Der vollständige Quellcode der Anwendung ist in Anhang F (Seite 72) nachzulesen. Im Rahmen der Live-Demo des Webclients unter der Adresse http://cdb.stalhut.de/example/ kann auch die Apk-Datei per QR-Code auf das Mobiltelefon heruntergeladen werden. Zur Nutzung kann dann ebenfalls der Benutzername „********“ mit dem Passwort „********“ verwendet werden. Im Folgenden soll auf einige Besonderheiten bei der Umsetzung (auch im Vergleich zu herkömmlichen Java-Anwendungen) und wichtige Stellen im Quellcode eingegangen werden.

7.1 Manifest-Datei

Bei der Entwicklung der App wurde die Haupt API-Version auf 14 festgelegt, was dem neusten Android 4.0 „Ice Cream Sandwich“ entspricht. Nur so kann man bei der Entwicklung auch dessen Neuheiten nutzen. Gleichzeitig wurde auf eine Abwärtskompatibilität geachtet, sodass die App auch auf dem älteren Android 1.6 (API-Version 4) noch voll lauffähig ist. Eine Aufwärtskompatibilität ist seitens Google generell garantiert, sodass die App auf jeder zukünftig erscheinenden Android™-Version ohne weiteres lauffähig sein sollte. Diese Angaben zu unterstützten Android™-Versionen werden in der im XML-Format vorliegenden Manifest-Datei des Projekts festgelegt, so finden sich dort unter anderem folgende Zeilen:

```
<uses-sdk
    android:minSdkVersion="4"
    android:targetSdkVersion="14" />
```

Zusätzlich benötigt die Anwendung Berechtigungen, die später bei der Installation vom Benutzer eingefordert werden. Das ist durch das bereits erläuterte Sandkasten-Prinzip von Android notwendig, bei dem jede App standardmäßig keinen Zugriff auf das restliche Android™-System hat. In der Manifest-Datei werden dazu die folgenden zwei Zeilen eingetragen:

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

Diese fordern die Berechtigungen für die Nutzung einer bestehenden Internet-Verbindung ein und erlauben es der App einen (möglicherweise kostenpflichtigen) Anruf zu initiieren. Die Berechtigung ist notwendig, damit der Anruf mit einem Tipp auf die Nummer in der eigenen App durch die entsprechende Telefonie-App gestartet wird. Ohne diese Berechtigung könnte zwar die Nummer an diese App übergeben werden, sodass der Benutzer dann die Nummer bereits eingegeben in seiner Telefonie-App sehen würde, aber er müsste noch durch einen weiteren Tipp den eigentlichen Anruf starten.

Ansonsten erfolgt in der Manifest-Datei ein Eintrag für jede Activity einschließlich eines Titels, der dann in der Standard Titelzeile bzw. ab Android 3.0 in der Action-Bar dargestellt wird:

```
<activity
    android:configChanges="orientation|keyboardHidden|keyboard"
    android:label="@string/app_title_search"
    android:name="SearchActivity" >
</activity>
```

Zusätzlich erfolgt hier die Angabe des `configChange`-Parameters. Hierzu muss man wissen, dass normalerweise die Activity gestoppt und neu gestartet wird, wenn sich die äußeren Bedingungen ändern, wie z. B. das Drehen des Bildschirms oder das Ändern der System-Sprache. Dies ist notwendig, weil bei derartigen Änderungen unter Umständen andere Ressourcen-Dateien eingelesen werden müssen, die dann für die Darstellung der Activity verwendet werden. Da die hier verwendeten Layouts jedoch keine unterschiedlichen Darstellungen für das Quer- und Hochformat einsetzen, wäre ein Neustart der Activity – der natürlich auch eine gewisse Zeit benötigt, da dabei u. a. Inhalte von Formularfeldern zwischengespeichert werden müssen – beim Drehen des Bildschirms überflüssig. Mit diesem Parameter werden Ereignisse definiert, bei denen sich die Activity selbst um dessen Behandlung kümmert und daher kein pauschaler Activity-Neustart durchgeführt wird.

Für die Activity, die als erstes starten soll, wenn die Anwendung gestartet wird, ist ein entsprechender Intent-Filter in der XML-Datei zu definieren:

```
<intent-filter>
    <action android:name="android.intent.action.MAIN" />
    <category android:name="android.intent.category.LAUNCHER" />
</intent-filter>
```

Mit der angegebenen `action` wird festgelegt, dass es sich um die Haupt-Activity handelt, die keine Übergabeparameter erwartet und auch keine Daten zurückgibt. Über die `category` wird festgelegt, um welche Art von Activity es sich handelt: hier eine Anwendung, die im App-Launcher auftaucht.

7.2 Benutzeroberfläche

Die Oberfläche einer Android™-App wird idealerweise komplett mit entsprechenden XML-Dateien erstellt. Eine Umsetzung direkt im Java-Code, wie in herkömmlichen Java-Anwendungen, ist zwar auch unter Android möglich, jedoch wird von Google das Verfahren über XML-Dateien empfohlen insbesondere auch weil man dabei von dem SDK und dem zugehörigen Eclipse-Plugin sehr gut bei der Erstellung unterstützt wird. Dies reicht bis hin zu einem WYSIWYG-Editor (What You See Is What You Get) für die Benutzer-Oberfläche, der sich in Eclipse integriert. Darin lässt sich auch schon erkennen wie das Layout auf unterschiedlich großen Displays und bei verschiedener Ausrichtung (Hoch- oder Querformat) aussehen wird. Für die Feinheiten kann man dann zu dem XML-Editor wechseln, der auch eine passende Autovervollständigungsfunktion mitbringt.

Eine Sonderrolle nehmen die sogenannten Dialoge ein. Diese stellen keine Bildschirmfüllende Activity dar, sondern vielmehr eine Art Popup das sich über die aktuelle Activity legt. Solche Dialoge können zwar auch per XML mit einem Layout versehen werden, jedoch ist dies hier typischerweise nicht der Fall, da es bereits vorgefertigte Layouts hierfür gibt, mit denen sich bspw. Text und optimal am unteren Rand ein oder mehrere Button darstellen lassen. So wurde der „Über"-Dialog als ein solcher ein Alert-Dialog realisiert. Auch bei den an verschiedenen Stellen in der App auftauchenden Hinweis-Popups, die darüber informieren, dass im Hintergrund Daten geladen werden – wie auch eines in Abbildung 26 zu sehen ist – und die sich nicht schließen lassen son-

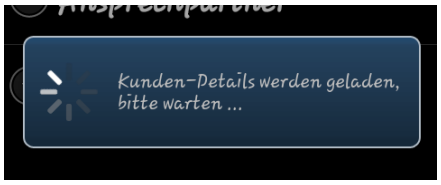

Abbildung 26: Android-App Anzeige bei Ladevorgang

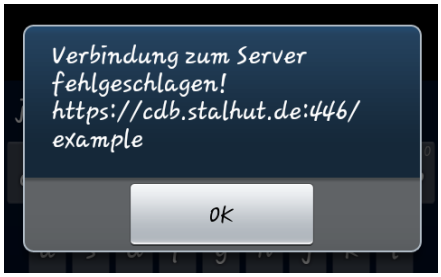

Abbildung 25: Android-App Fehlermeldung bei Verbindungsfehler

dern von allein verschwinden, wenn der Ladevorgang abgeschlossen ist handelt es sich um eine Unterart der Alert-Dialoge, die sogenannten Progress-Dialoge. Ansonsten finden derartige Alert-Dialoge im Fehlerfall ihre Anwendung, bspw. wenn der Server mangels Internet-Verbindung nicht erreichbar ist, wie es in Abbildung 25 zu sehen ist.

7.2.1 Menü

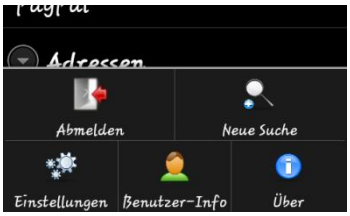

Abbildung 27: Android-App-Menü bei Android 2.3

Unabhängig von der jeweils sichtbaren Activity soll beim Drücken der Menü-Taste ein einheitliches Menü erscheinen. Dieses wurde daher in der übergeordneten Klasse definiert, von der alle weiteren Activity-Klassen erben, wie es bereits in 4.3.1 (Seite 16) dargestellt wurde.

Generell werden die Menü-Einträge wieder empfohlenerweise in einer entsprechenden XML-Datei definiert. Ein solcher Menü-Eintrag sieht darin dann z. B. wie folgt aus:

```
<item
    android:id="@+id/logout"
    android:icon="@drawable/logout"
    android:showAsAction="ifRoom"
    android:title="@string/label_menulogout"/>
```

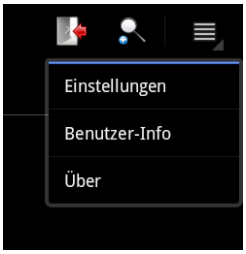

Abbildung 28: Android-App-Menü bei Android 3.0

Neben der obligatorischen ID und einem Titel, der als Menü-Eintrag angezeigt wird, wurde hier auch ein Icon angegeben, das bei Versionen vor Android 3.0 über dem jeweiligen Titel im Menü angezeigt wird, sodass das Menü dann wie in Abbildung 27 zu sehen, am unteren Bildschirm-Rand eingeblendet wird. Ab Android 3.0 können dann bestimmte Menü-Einträge statt in der Menü-Auswahl (die nun keine Icons mehr darstellt) auch direkt als Symbol in der Action-Bar angezeigt werden, wobei dann auch wiederum nur das Icon, nur der Titel oder Icon mit Titel angezeigt werden können. Ob der Menüpunkt in der Action-Bar auftaucht, entscheidet der Parameter `android:showAsAction`, mit dem hier gewählten Wert `ifRoom`. Dieser Eintrag taucht nur in der Action-Bar auf, wenn dort (u. a. in Abhängigkeit von der Größe des Bildschirms) noch genügend Platz ist, andernfalls „verschwindet“ er wieder hinter der Menü-Auswahl. So sieht dann die Menü-Auswahl bei ausgeklapptem Menü auf einem Tablet aus wie in Abbildung 28, als Ausschnitt der oberen rechten Bildschirm-Ecke.

Bei Icons des Menüs, gleiches gilt auch für das Icon der App selber, ist zu beachten, dass diese in drei verschiedenen Auflösungen bereitgestellt werden sollten, um die verschiedensten Display-Auflösungen der Smartphones abzudecken. Dazu gibt es drei passende Standard-Ordner (drawable-hdpi, drawable-ldpi und drawable-mdpi), in denen jeweils gleichnamige Grafikdateien abgelegt werden können, sodass das Android™-System dann je nach DPI des Bildschirms die passende Grafik auswählen kann.

Zum Hinzufügen des Menüs zur Activity wird die entsprechende Methode wie folgt überschrieben, wobei im Wesentlichen die als Datei *global.xml* abgelegte Menü-Definition eingebettet wird, indem auf diese mit Hilfe der bereits in 3.1.4.5 (Seite 10) vorgestellten R-Klasse verwiesen wird:

```
public boolean onCreateOptionsMenu(final Menu menu) {
    MenuInflater inflater = getMenuInflater();
    inflater.inflate(R.menu.global, menu);
    return true;
}
```

Auf einzelnen Activities sollen bestimmte Menü-Einträge nicht sichtbar sein: Beispielsweise sollte der Eintrag „Benutzer-Info“ nicht auftauchen, wenn man sich bereits auf der entsprechenden Activity befindet, die die Benutzer-Infos darstellt, und es macht auch wenig Sinn die Benutzer-Infos, eine neue Suche oder die Logout-Option anzubieten, wenn der Benutzer noch nicht eingeloggt ist. Diese Anpassungen werden beim Aufklappen des Menüs vorgenommen, dazu wird die entsprechende zuständige Methode `boolean onPrepareOptionsMenu(Menu menu)` überschrieben. Hierin können dann ggf. einzelne Menü-Einträge mit `menu.removeItem(id);` entfernt werden, wobei die zu übergebene ID wieder über die R-Klasse referenziert wird.

Abbildung 29: Android-App, Login-Activity unter Android 2.3

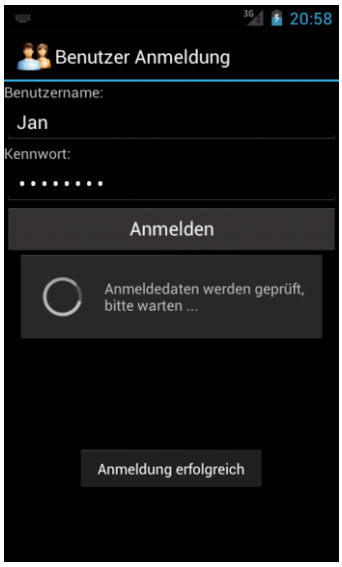

Abbildung 30: Android-App, Verarbeitung des Logins unter Android 4.0

7.2.2 Login-Formular

Die Login-Activity enthält im Wesentlichen die Eingabefelder für Benutzernamen und Kennwort. Wie diese dann unter Android 2.3 dargestellt wird, kann der Abbildung 29 entnommen werden. Mit dem Tippen auf den Anmelden-Button werden diese an den Server gesendet und währenddessen wird dem Benutzer ein Alert mit der Information gezeigt, dass die Anmeldedaten geprüft werden. Bei erfolgreichem Login wird dann ein sogenannter Toast (kleine Info-Meldung, die ebenfalls Popup-artig über der aktuellen Ansicht „schwebt“, jedoch automatisch nach wenigen Sekunden wieder verschwindet) eingeblendet, der den Benutzer über die erfolgreiche Anmeldung informiert, wie es in Abbildung 30 dargestellt ist. Beim Start der Login-Activity (und damit beim Start der App) wird intern zunächst geprüft, ob bereits ein Benutzer angemeldet ist, dessen Session-ID und -Schlüssel gespeichert sind. Ist dies der Fall, erfolgt direkt eine „Umleitung“ zur Activity der Kunden-Suche.

7.2.3 Kunden-Suche

Für die Suchfunktion wurde eine Activity erstellt, die neben dem Suchfeld nur ein sogenanntes *ListView* enthält, das dann automatisch den restlichen verfügbaren Bildschirmplatz einnimmt. Beim Aufruf dieser Activity bekommt automatisch das Suchfeld den Focus, sodass sich auch automatisch die Tastatur einblendet. Sobald der Benutzer anfängt etwas einzutippen, beginnt auch die Suche mit der Arbeit: Das wird dem Benutzer durch einen kleinen Info-Text unter dem Eingabefeld angezeigt. Dieser Zustand ist in Abbildung 31 zu sehen. Der Ladevorgang ist in einen im Hintergrund laufenden AsyncTask ausgelagert, damit der Thread, der sich um die Benutzeroberfläche kümmert (UI-Thread), nicht mit dieser Aktion blockiert wird. Eine solche Blockierung des UI-Threads würde nach 10 Sekunden zu einer Systemmeldung führen, die dem Benutzer mitteilt, dass sich die Anwendung aufgehängt hat und anbietet, dessen Schließen zu erzwingen. Nach wenigen Sekunden (je nach verfügbarer Bandbreite zwischen

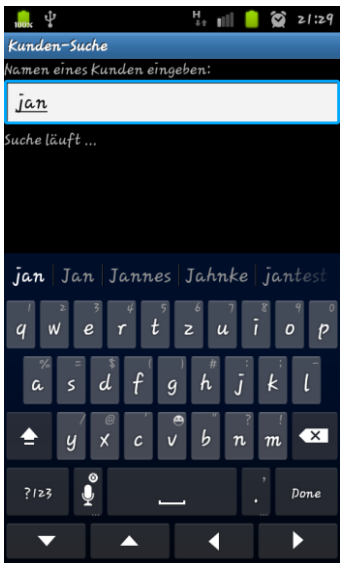

Abbildung 31: Android-App, Suche in Aktion mit Android 2.3

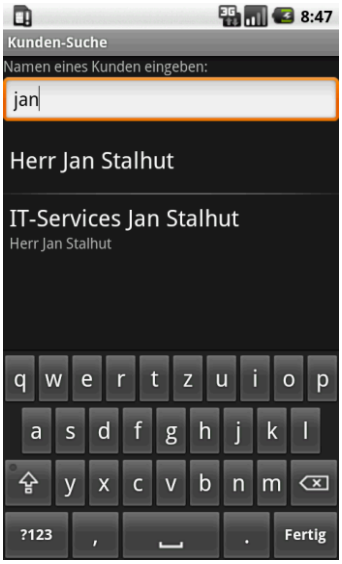

Abbildung 32: Android-App, Suchergebnisse bei Android 1.6

Client und Server) werden dann die Ergebnisse der Suche, wie in Abbildung 32 zu sehen, als Elemente der *ListView* ausgegeben. Die ListView sorgt automatisch dafür, dass bei nicht ausreichendem Bildschirm-Platz rauf und runter gescrollt werden kann.

Um bei der Suche während der Eingabe zu vermeiden, dass für jeden „Tastenanschlag" eine Serveranfrage ausgelöst wird, wird jeweils im Hintergrund noch eine Sekunde gewartet, ob noch ein weiterer Buchstabe eingegeben wird, ehe die Anfrage tatsächlich an den Server gesendet wird. Sollten bereits Ergebnisse angezeigt werden und der Benutzer dann noch weitere Zeichen in das Suchfeld eingeben, erscheint zwischen ListView und Eingabefeld wieder der Hinweis, dass die „Suche läuft" und sobald eine Antwort vom Server vorliegt, wird der Inhalt der ListView entsprechend aktualisiert. Außerdem müssten mindestens zwei Zeichen eingegeben worden sein, ehe die Suche überhaupt beginnt.

Bei Privatkunden wird in der Liste der Ergebnisse direkt der vollständige Name angezeigt. Bei Geschäftskunden wird der angegebene Name des Unternehmens angezeigt und darunter etwas kleiner der vollständige Name einer Kontaktperson. Generell ist in der Liste erst die Ausgabe der bzw. des Privatkunden zu sehen und darunter folgen dann ggf. passende Geschäftskunden. Jeder Eintrag der Liste kann angetippt werden, um dann zu der Detailansicht des gewählten Kunden zu gelangen.

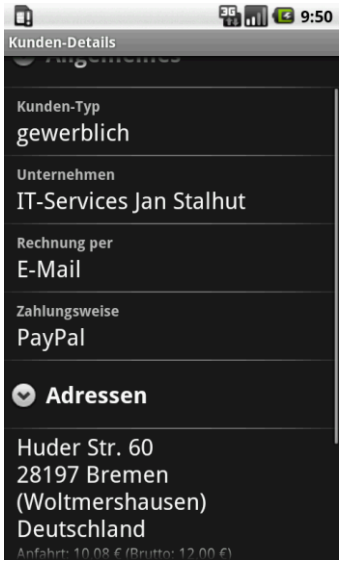

Abbildung 33: Android-App, Kundendetails unter Android 1.6

7.2.4 Detail-Ansichten

In der Detailansicht eines Kundendatensatzes kommt eine sogenannte *ExpandableListView* zum Einsatz. Diese ermöglicht ergänzend zur einfachen ListView die Gruppierung von Listen-Einträgen und das gezielte Aus- und Einklappen einzelner Kategorien. Beim Aufruf sind hier zunächst alle Gruppen zugeklappt und es erscheint ein Progress-Dialog, der darüber informiert, dass die Daten geladen werden, was wieder in einen AsyncTask ausgelagert ist. Anschließend lassen sich dann vom Benutzer durch Antippen jeweils die Gruppen aufklappen, sodass es dann etwa aussehen kann wie in Abbildung 33 abgebildet. In dieser Darstellung könne einzelne Listen-Einträge als antippbar definiert werden. So lassen sich die einzelnen Infos, wie Kunden-Typ, Zahlungsweise etc. nicht antippen, während ein Tipp auf die Adresse einen entsprechenden Intent auslöst, auf den installierte Apps, die mit einer Adressinformation etwas anfangen können, reagieren können. Konkret wurde es bei der Adresse so realisiert, dass immer eine Auswahl der möglichen Anwendungen erscheint, die eine Aktion zu der Adresse durchführen können. Wenn nun z. B. eine App wie „Öffi" installiert ist, bietet diese hier an, eine Verbindung mit öffentlichen Verkehrsmitteln zu dieser Andresse herauszusuchen. Ist die (normalerweise immer vorhandene) Maps-App von Google vor-

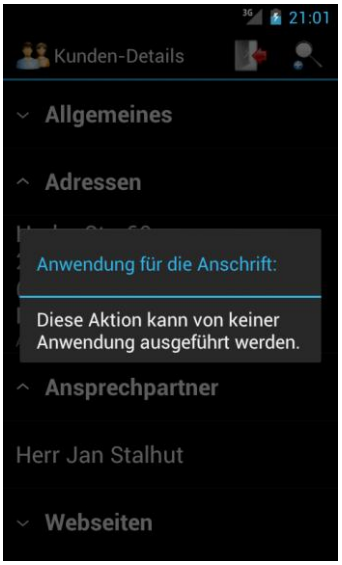

Abbildung 34: Android-App, keine passende App gefunden unter Android 4.0

handen, so bietet diese an, die Adresse auf der Karte herauszusuchen und anzuzeigen. Diese Auswahl-Popup, das vom Android™-System generiert wird, sieht dann bspw. so aus wie in Abbildung 35. Sollte einmal keine passende App vorhanden sein, die etwas mit einer Adress-Angabe anfangen kann, wird eine entsprechende Meldung angezeigt, wie sie in Abbildung 34 zu sehen ist. Um dies unabhängig von einer möglicherweise als Standard festgelegten App zu realisieren, wurde die folgende Funktion implementiert:

```
protected final void openMap(final String address) {
    final Intent intent = new Intent(Intent.ACTION_VIEW);
    final Uri uri = Uri.parse("geo:0,0?q=" + Uri.encode(address));
    intent.setData(uri);
    startActivity(Intent.createChooser(intent,
        getString(R.string.chooseapp_address)));
}
```

Dem Konstruktor des Intents wird direkt der Typ der Aktion übergeben. Im Falle eines Anrufes stände hier stattdessen `Intent.ACTION_CALL`. Der URI hat ein vorgegebenes Format, auf das dann kompatible Apps einen entsprechenden Intent-Filter im System registrieren können, um bei der Auswahl der angebotenen Apps dabeizusein. Hier wird nun nicht direkt der Intent an die startActivity-Methode übergeben, sondern der gewünschte *Chooser*, der für das sichtbare Auswahl-Popup sorgt, auch wenn normalerweise für diese Aktion eine Standard-Anwendung im System festgelegt ist.

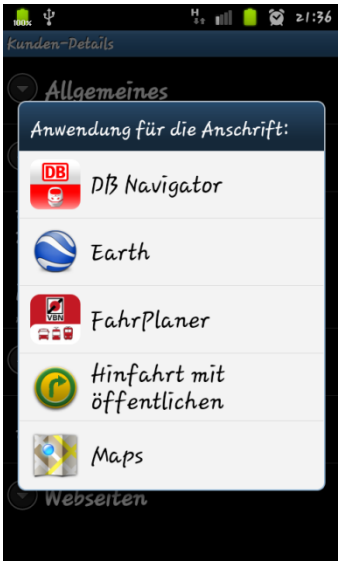

Abbildung 35: Android-App, Auswahl der Aktion (Android 2.3)

Schließlich lassen sich auch die Einträge unter „Webseiten“ entsprechend antippen, um die Adresse im Standard Webbrowser zu öffnen.

Wird einer der verknüpften Ansprechpartner angetippt, wird zur Activity mit den Personen-Details gewechselt. Auch hier wird die gleiche Darstellungsweise mit dem ExpandableListView eingesetzt, die Informationen in die Gruppen „Allgemeines“, „Telefonnummern“, „E-Mail-Adressen“, „Faxnummern“ und „zugehörige Kunden“ unterteilt (siehe auch Abbildung 36). Auch hier sind wieder die Einträge unter „Allgemeines“ sowie „Faxnummern“ nicht mit weiteren Aktionen versehen. Die Telefonnummern lassen sich hingegen antippen, woraufhin direkt mit der Standard Telefonie-App ein Anruf eingeleitet wird. Ebenso lässt sich mit einem Tipp auf eine der E-Mail-Adressen direkt eine neue E-Mail an diese Adresse mit der Standard Mail-Anwendung verfassen.

Abbildung 36: Android-App, Detailansicht für Personen

Bei allen Aktionen, die sich auf die Standard App des Systems beziehen, wird ggf. vom System auch eine Auswahl angeboten, wenn mehrere passende Apps vorhanden sind und noch keine davon als Standard festgelegt wurde. Dieses Auswahlfeld beinhaltet dann auch ein Häkchen, das der Benutzer aktivieren kann, um diese Aktion zukünftig immer mit der gewählten Anwendung aufzurufen. Sollte einmal keine passende App auf dem System vorhanden sein, wird auch hier wieder eine entsprechende Meldung ausgegeben, die den Benutzer über diese Umstände informiert. Im Java-Code sieht dies dann in Form einer

Funktion folgendermaßen aus:

```
protected final void openWebsite(final String url) {
    final Intent intent = new Intent(Intent.ACTION_VIEW);
    final Uri uri = Uri.parse(url);
    intent.setData(uri);
    try {
        startActivity(intent);
    } catch (ActivityNotFoundException e) {
        showDialog(DIALOG_ACTIVITYNOTFOUND_ID);
    }
}
```

Als URI wird hier die unveränderte URL der Webseite übergeben. Sollte es keine passende App geben, die etwas mit den übergebenen Daten anfangen kann – sprich im System einen passenden Intent-Filter registriert hat – wird die Exception geworfen, wodurch in der obigen Funktion ein entsprechender Dialog angezeigt wird.

Bei dem Wählen einer Rufnummer wird zusätzlich die zu einer Person hinterlegte Einstellung berücksichtigt, ob ihr die eigene Rufnummer übermittelt werden soll oder nicht (Leistungsmerkmal CLIR). Um diese Funktion zu realisieren, werden der eigentlichen Rufnummer passende Codes vorangestellt, die vom Netzbetreiber entsprechend ausgewertet werden. So ist dies generell unabhängig von der Einstellung, die diesbezüglich in den Systemeinstellungen vorgenommen wurde. Diese Codes können in den Einstellungen der App angepasst werden, sind jedoch bereits standardmäßig auf die in Deutschland üblichen Codes `*31#` (zur Übermittlung der Rufnummer) und `#31#` (zur Unterdrückung der Rufnummer) voreingestellt. Die Initalisierung des Anrufes geschieht dann ähnlich wie der Aufruf anderer Intents mit der folgenden Funktion:

```
protected final void openPhone(final String number) {
    final Intent intent = new Intent(Intent.ACTION_CALL);
    final Uri uri = Uri.parse("tel:" + Uri.encode(number));
    intent.setData(uri);
    try {
        startActivity(intent);
    } catch (ActivityNotFoundException e) {
       showDialog(DIALOG_ACTIVITYNOTFOUND_ID);
    }
}
```

Zu beachten ist lediglich, dass der Aufruf eines solchen Intents vom Typ Intent.ACTION_CALL die entsprechende Berechtigung benötigt, da ansonsten an dieser Stelle eine Exception geworfen werden würde, diese wurde bereits – wie in 7.1 (Seite 38) erläutert – in der Manifest-Datei eingetragen. Der URI wird hier wieder in der entsprechenden – für Telefonnummern festgelegten – Form angegeben.

Abbildung 38: Android-App, Dialog zur Passwort-Änderung (Android 2.3)

7.2.5 Benutzer-Informationen

Über das Menü lässt sich – wie bereits erwähnt – die Activity mit Benutzer-Informationen aufrufen. Hier werden dann der Benutzername, die zugeordnete Benutzergruppe und eine Auflistung der aktiven Sitzungen auf verschiedenen Geräten sichtbar. Wie dies dann auf dem Smartphone aussieht, kann der Abbildung 37 entnommen werden. Die Auflistung der aktiven Sitzungen ist wieder als ListView implementiert, sodass auch automatisch eine Scroll-Funktion vorhanden ist. Des Weiteren steht ein Button zur Verfügung, über den der Benutzer sein eigenes Kennwort ändern kann. Dieser Dialog ist in Abbildung 38 zu sehen. Hier wurde – im Gegensatz zu anderen Alert-Dialogen – ein eigenes Layout verwendet, das in einer extra XML-Datei definiert wurde. Die sichtbaren Button „OK“ und „Abbruch“ sind dann wieder Bestandteil des Standard Alert-Dialogs und müssen nur entsprechen beschriftet und mit Aktionen versehen werden. Zum Laden der Session wird wieder der übliche AsyncTask verwendet, sodass bei Aufruf der Activity zunächst zu lesen ist, dass die Session-Informationen geladen werden. Auch die Passwort-Änderung wird dann wieder mit einem entsprechenden AsyncTask in den Hintergrund verlagert, sodass im Vordergrund mit einem Progress-Dialog auf die Hintergrund-Aktivität hingewiesen werden kann.

Abbildung 37: Android-App, Benutzer-Informationen unter Android 2.3

7.2.6 Listen

Abbildung 39: Listenelement mit kleiner Überschrift (Android 1.6)

Die eingesetzten ListViews bzw. deren Unterart – die ExpandableListView – haben seitens des Systems ein Standard-Layout und Standard-Funktionalitäten, wie die Möglichkeit Elemente anzutippen und ggf. innerhalb der Liste zu scrollen. Mit den Bordmitteln ist es möglich, in jedem Listenelement einen String ohne speziellere Formatierungsmöglichkeiten auszugeben. Da für die hier gewünschte Darstellung auch noch ein kleinerer Text unterhalb und/oder oberhalb des Haupttextes eines Elements der ListView angezeigt werden soll (siehe Abbildung 39), war eine eigene Spezialisierung des ArrayAdapters (für normale ListViews) und des BaseExpandableListAdapters (für ExpandableListVIews) notwendig. In dieser wird dann im Wesentlichen die Methode getView überschrieben, die für jedes Listen-Element automatisch aufgerufen wird, sobald diese auf dem Display sichtbar wird. Hier wird dann die eigens dafür erstellte Layout-Datei *list_item.xml* verwendet, jeweils mit Inhalt gefüllt und ggf. das TextView oder- und/oder unterhalb des Hauptinhalts ausgeblendet. Als Besonderheit sei hier erwähnt, dass ggf. nicht mehr auf dem Display sichtbare View zum Recyceln als Parameter an die Funktion übergeben werden, die dann aus Performance-Gründen auch ggf. genutzt werden sollten, ehe unnötig neue Views aus der XML-Datei erzeugt werden (hier am Beispiel des Adapters für das einfache ListView):

```
public View getView(final int position, final View convertView,
        final ViewGroup parent) {
    View item = convertView;
    if (item == null) {
        LayoutInflater inflater = LayoutInflater.from(getContext());
        item = inflater.inflate(R.layout.list_item, parent, false);
    }
    /* ... Werte der einzelnen TextViews des Items setzen ... */
    return item;
}
```

Mit dem Inflator wird hier das XML-Layout in das view-Element geladen, falls kein wiederzuverwendendes View übergeben wurde. Anschließend werden die einzelnen TextViews mit Hilfe deren ID aufgesucht und mit dem aktuellen Inhalt gefüllt oder ggf. ausgeblendet.

7.2.7 Theme des Layouts

Für die Layouts lassen sich bestimmte Themes festlegen (für einzelne Activities oder auch die komplette Anwendung). Darauf wurde in diesem Fall verzichtet, was dafür sorgt, dass jeweils das Standard-Theme des Systems verwendet wird, auf dem die Anwendung ausgeführt wird. Dies kann das Standard Theme sein, das Google bei Android™ mitliefert, kann aber auch ein vom Smartphone-Hersteller (farblich) angepasstes Theme sein. So sieht man auf dem in den vorherigen Abschnitten sichtbaren Screenshots, die mit Version 2.3 gekennzeichnet sind, eine blau hinterlgte Titelzeile, eine blaue Umrandung um das aktuelle Eingabefeld und Ähnliches. Diese ist durch die von Samsung auf dem Galaxy S durchgeführten Änderungen am Standard Theme bedingt. Es würden ansonsten auch alternative Themes zur Verfügung stehen, wie z. B. eines, das einen weißen statt schwarzen Hintergrund mitbringt. Letztlich sind natürlich auch manuelle Änderungen an sämtlichen UI-Elementen denkbar.

7.3 Mehrsprachigkeit

Generell ist die Android™-Architektur sehr gut für eine mehrsprachige Unterstützung vorbereitet, da alle (sprachrelevanten) Strings in eine entsprechende XML-Datei ausgelagert werden können. Diese XML-Dateien befinden sich im Standard-Verzeichnis *values*. Das SDK biete nun die Möglichkeit, für weitere Sprachen einfach zusätzliche *values*-Ordner mit jeweils entsprechend angehängten Sprachkürzeln anzulegen, in denen dann gleichnamige XML-Dateien liegen, die die Strings in der jeweiligen Sprache enthalten. Das Android™-System wählt dann automatisch je nach gewählter System-Sprache die passenden Strings. Sollte keine Sprachdatei zur aktuellen Sprache zur Verfügung stehen, wird auf die Daten aus dem Standard-Verzeichnis *values* (ohne angehängtes Kürzel) zurückgegriffen. So ist es empfehlenswert, in diesem Verzeichnis die englischsprachigen Zeichenketten zu hinterlegen und für jede weitere unterstützte Sprache entsprechende Verzeichnisse anzulegen.

So wurden auch für das vorliegende Projekt zunächst alle Texte in englischer Sprache in der Datei *strings.xml* im *values*-Verzeichnis hinterlegt und die deutschen Übersetzungen in einer gleichnamigen Datei im dafür angelegten Verzeichnis *values-de*. Folglich ist diese Anwendung schon vollständig zweisprachig verfügbar und für jede beliebige weitere Sprache, die hinzugefügt werden soll, würde dies nur noch den reinen Übersetzungsaufwand bedeuten.

7.4 Serverkommunikation

Für die Kommunikation mit dem Server werden – wie bereits erwähnt – sogenannte *AsyncTasks* verwendet. Dies ist notwendig, damit der UI-Thread weiterlaufen kann und nicht das Android™-System

nach 10 Sekunden meldet, dass die Anwendung nicht mehr reagiere, nur weil eine Übertragung zwischen Client und Server mal etwas länger dauert.

Zur De- und Encodierung der JSON-Daten bringt auch das Android™-System bereits eine passende Bibliothek mit, die lediglich Java-typisch eingebunden werden muss:

```
import org.json.JSONException;
import org.json.JSONObject;
```

Sämtliche im Rahmen der definierten JSON-Schnittstelle notwendigen Schlüssel wurden als statische öffentliche Variablen in der Klasse Communication definiert, so kann auf diese unkompliziert und schnell zugegriffen werden, wenn Daten in ein JSON-Objekt gepackt werden sollen um dieses dann an den Server zu senden:

```
try {
    JSONObject pw = new JSONObject();
    pw.putOpt(Communication.JSONKEY_PWOLD, oldpw);
    pw.putOpt(Communication.JSONKEY_PWNEW, newpw1);
    JSONObject data = new JSONObject();
    data.put(Communication.JSONKEY_ACTION_PWCHANGE, pw);
    CommunicationChangePw comm =
        new CommunicationChangePw(this, serverurl, serverport);
    comm.setAuth(sessionId, sessionKey);
    comm.execute(data);
} catch (JSONException e) {
    showDialog(DIALOG_JSONCREATINGERROR_ID);
}
```

Hier sieht man am Beispiel der Passwort-Änderung, wie zunächst das alte und das neue Kennwort in ein neues JSON-Objekt gelegt werden. Das wird wiederum unter dem fest definierten Schlüssel in ein übergeordnetes JSON-Objekt gepackt. An diesem Schlüssel wird der Server später erkennen, welche Aktion vom Client gefordert ist. Anschließend wird eine neue Instanz der von `Communication` spezialisierten Klasse `CommunicationChangePw` unter Angabe der aktuell eingestellten Parameter bzgl. Server-Adresse und -Port erzeugt. Mit dem Aufruf der Methode `setAuth` werden dem Objekt die aktuellen Zugangsdaten übergeben. Die Methode `execute` nimmt schließlich das erzeugte JSON-Objekt entgegen, fügt noch die Authentifizierungsinformation ein, sowie Informationen zum aktuellen Client und dessen Versionsnummer und überträgt alles an den Server. Bei der Methode `execute` handelt es sich um die vom AsyncTask geerbte Methode, die automatisch in einem separatem Hintergrund-Prozess ausgeführt wird.

Die spezialisierte Klasse `CommunicationChangePw` ist als innere Klasse der Activity implementiert, in die dann im Wesentlichen nur noch die vom AsyncTask geerbten Methoden `onPreExecute`, `onCancelled` und `onPostExecute` implementiert wurden.

Die Funktion `onPreExecute` wird in dem Moment ausgeführt, wenn der separate Thread im Hintergrund gestartet wird und enthält dabei den folgenden Befehl, der für die Anzeige des erwähnten Progress-Dialogs sorgt. Dieser soll dem Benutzer symbolisieren, dass im Hintergrund Daten mit dem Server ausgetauscht werden. Bei der Suche würde an dieser Stelle der kleine Text unter dem Suchfeld eingeblendet, sodass gleichzeitig weitere Eingaben möglich sind.

```
dialog = ProgressDialog.show(ActivityParent.this, "",
        getString(R.string.loading_pwchange), true, false);
```

Die Funktion `onCanceled` wird in dem Fall aufgerufen, wenn der laufende AsyncTak unterbrochen wird. Dies ist hier z. B. der Fall wenn der Server nicht erreichbar sein sollte. Zum einen wird hier der Progress-Dialog geschlossen und zum anderen wird die Fehlermeldung an die Methode `handleCommunicationError` weitergegeben, die in der abstrakten `ActivityParent`-Klasse definiert ist und die dann eine entsprechende Fehlermeldung für den Benutzer ausgibt.

```
dialog.dismiss();
handleCommunicationError(errorcode, errormsg);
```

Schließlich wird die Funktion onPostExecute ausgeführt, wenn der Hintergrund-Prozess vollständig abgearbeitet wurde. Hier wird bei erfolgreicher Passwort-Änderung ein *Toast* mit dieser Info eingeblendet. Sollte etwas schief gelaufen sein, wird eine entsprechende Fehlermeldung angezeigt und in jedem Fall wird auch der Progress-Dialog geschlossen.

```
try {
    if (result.has(Communication.JSONKEY_PWCHANGEOK)
            && result.getBoolean(Communication.JSONKEY_PWCHANGEOK)) {
        createToast(R.string.pwchangeok);
    } else {
        showDialog(DIALOG_PWCHANGEERROR_ID);
    }
} catch (JSONException e) {
    showDialog(DIALOG_JSONHANDLINGERROR_ID);
}
dialog.dismiss();
```

Durch die Struktur der inneren Klasse können bei Fertigstellung des Hintergrund-Threads problemlos ermittelte Ergebnisse oder Daten in das UI eingebunden werden. Es gibt also keine Synchronisierungs-Schwierigkeiten zwischen UI- und Hintergrund-Thread, wie man sie hätte, wenn man einen klassischen Java-Thread starten würde, der am Ende Daten im UI ausgeben soll.

7.4.1 SSL-Zertifikat

Damit vom Android™-System das Server-Zertifikat akzeptiert wird, das von einem dem Android™-System unbekannten Root-Zertifikat zertifiziert wurde, musste dies in einen eigenen Schlüssel-Speicher importiert und mit der Anwendung mitgeliefert werden. Um dieses Zertifikat dann nutzen zu können, war das Überschreiben der `DefaultHttpClient`-Klasse in einer eigenen spezialisierten `MyHttpClient`-Klasse notwendig. In dieser wird eine eigene SslSocketFactory definiert, die auf den mit dem *raw*-Verzeichnis der App mitgelieferten Zertifikatsspeicher zurückgreift und das Serverzertifikat mit dessen Hilfe validiert. Zusätzlich erfolgt auch hier die Angabe des vom Standard abweichenden HTTPS-Port.

7.5 Einstellungen

Für Einstellungen, die dem Anwender zur Verfügung gestellt werden sollen, wird von Android™ eine vorgefertigte *ReferenceActivity* mitgeliefert, die dann wie in Abbildung 40 dargestellt wird. Dabei wird die *PreferenceActivity*-Klasse spezialisiert, sodass dort in der *onCreate*-Methode die Angabe der zu verwendenden XML-Datei angegeben werden kann:

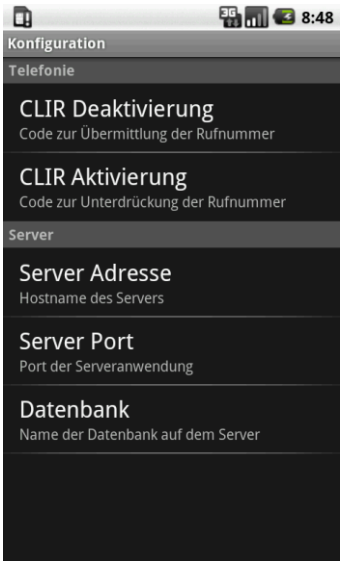

Abbildung 40: Einstellungs-Activity unter Android 1.6

```
protected void onCreate(final Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    addPreferencesFromResource(R.xml.preferences);
    /* ... register the sharedPreferenceChangeListener ... */
}
```

Die dort über die R-Klasse referenzierte XML-Datei *preferences.xml* befindet sich im Ressourcen-Verzeichnis *xml*. Darin sind die konkreten einzelnen einstellbaren Parameter hinterlegt und in Kategorien gruppiert. Eine dieser Einstellungen sieht dann bspw. wie folgt aus:

```
<EditTextPreference
    android:defaultValue="*31#"
    android:key="codeNumberTransmit"
    android:summary="@string/config_dial_cliroff_note"
    android:title="@string/config_dial_cliroff"
    android:inputType="phone" />
```

In diesem Beispiel wird die bereits erwähnte Einstellung für den Rufnummern-Präfix definiert, der dafür sorgt, dass die eigene Rufnummer übermittelt wird. Hier erfolgt neben der Angabe des Standard-Wertes (`android:defaultValue`) und eines eindeutigen Bezeichners (`android:key`), die für den Benutzer sichtbare Bezeichnung (`android:title`) und Beschreibung (`android:summary`) in Form einer Referenz auf die lokalisierte XML-Datei *strings.xml*. Der hier zusätzlich definierte `android:inputType` sorgt dafür, dass bei der Änderung durch den Benutzer das Tastatur-Layout verwendet wird, das nur Telefonnummern-typische Zeichen erlaubt. Die Dialoge, wie einer in Abbildung 41 zu sehen ist, werden dann vom Android™-System automatisch erzeugt.

Um bei Änderung einer der Server betreffenden Einstellungen die aktuelle Benutzer-Session clientseitig zu löschen (diese wäre ja dem neuen/anderen Server nicht bekannt), wurde in der onCreate_Methode zusätzlich on Listener registriert, der ausgelöst wird, sobald eine Einstellung geändert wird. Dieser prüft dann, ob es sich um eine der betreffenden Einstellungen handelt und löscht dann die gespeicherten Session-Daten – falls vorhanden. Beim Aufruf einer Activity wiederum wird geprüft, ob Session-Daten gespeichert sind und ansonsten zur Login-Activity umgeleitet.

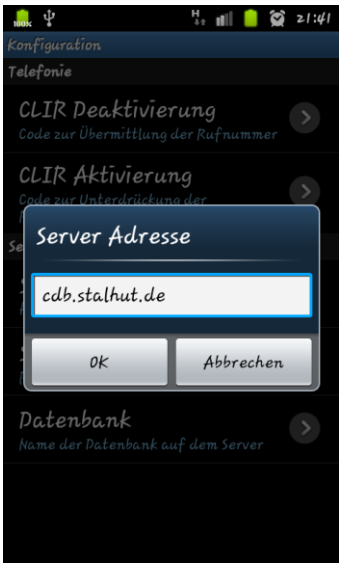

Abbildung 41: Konfiguration des Servers unter Android 2.3

Soll später im Java-Code auf die aktuellen Einstellungen zugegriffen werden, steht dafür der PreferenceManager zur Verfügung:

```
PreferenceManager.setDefaultValues(this,
R.xml.preferences, false);
preferences = PreferenceManager
        .getDefaultSharedPreferences(this);
serverport = Integer.parseInt(
        preferences.getString(PREF_SERVERPORT, ""));
```

In der 2. Zeile werden mit dem PreferenceManager die aktuellen Einstellungen geladen, um dann mit der *getString*-Methode konkrete Werte zu holen. Bei dieser Methode muss als zweiter Parameter ein Standard-Wert angegeben werden, der zurückgegeben wird, wenn die Einstellung noch nicht konfiguriert wurde. Um jedoch nicht an zwei Stellen (hier im Java-Code und in der preferences.xml) Standard Werte für die Einstellungen pflegen zu müssen hat man sich für die Variante entschieden, zunächst mit der *setDefaultValues*-Methode (1. Zeile) die

Standard-Werte zu laden, falls das Programm zum ersten Mal aufgerufen wird bzw. die PreferencesActivity noch nie zuvor aufgerufen wurde. Der dritte boolesche Parameter sorgt dafür, dass die Standard-Einstellungen nicht geladen werden, wenn bereits Anpassungen vorgenommen wurden.

7.6 An- und Abmeldung

Beim Login-Vorgang werden – genauso wie beim Webclient – Benutzername und Passwort an den Server gesendet, der im Erfolgsfall u. a. auch die Informationen zum Benutzer zurückliefert. Dazu gehören Benutzername, Benutzer-ID, Benutzergruppe und Berechtigungen. Die Android™-App speichert diese Informationen dann lokal, ebenso wie die für zukünftige Server-Zugriffe notwendige Session-ID und den zugehörigen Session-Key. Eine Anmeldung ohne Speicherung als Session ist hier nicht vorgesehen. Über das Menü, bzw. ab Android 3.0 über den Button in der Action-Bar, hat der Benutzer jedoch jederzeit die Möglichkeit sich abzumelden, wodurch die lokal gespeicherten Informationen gelöscht werden und wieder das Login-Formular angezeigt wird.

Wurde die aktive Session des Smartphones vom Webclient aus gelöscht, oder ist die Session schlicht aufgrund 30-tägiger Inaktivität abgelaufen, meldet der Server dies dem Client zurück, wenn dieser versucht Daten abzufragen. Daraufhin informiert die App den Benutzer darüber, dass seine Session abgelaufen ist und zeigt ebenfalls wieder die Login-Activity.

7.7 Datenhaltung

Zur clientseitigen Speicherung von Informationen, die im Rahmen des Login-Vorgangs anfallen, werden die vom Android™-System für jede App zur Verfügung gestellten *SharedPreferences* genutzt. Dabei handelt es sich um einfache Schlüssel-Wert-Paare. Während als Key für jeden Wert ein String Verwendung findet, kann der Wert von verschiedenen Typs sein: Boolean, Float, Integer, Long, String, und ein String-Set werden unterstützt. Um Daten als SharedPreference abzulegen, wird ein ShredPreferences-Editor-Objekt benötigt:

```
final SharedPreferences userdata = getSharedPreferences(USERDATA_PREF,
        MODE_PRIVATE);
SharedPreferences.Editor editor = userdata.edit();
editor.putInt(USERDATA_PREF_USERID, userinfo.getInt(Communication.JSONKEY_ID));
editor.putString(USERDATA_PREF_USERNAME,
        userinfo.getString(Communication.JSONKEY_USERNAME));
editor.commit();
```

In diesem Beispiel, das zur Speicherung der Benutzer-ID und des Benutzernamens dient, wird zunächst auf eine bestimmte SharedPreferences-Datei, dessen Name mit der Konstanten `USERDATA_PREF` festgelegt ist, zugegriffen bzw. ggf. diese Datei angelegt. Mit dem zweiten Parameter wird angegeben, dass es sich dabei um Einstellungen handeln soll, auf die nur die App selber Zugriff haben soll. Alternativ könnte man hier auch anderen Apps Lese- oder sogar Schreibzugriff gestatten. Im nächsten Schritt muss zur Bearbeigung ein Editor-Objekt geholt werden, über das dann mit Hilfe von Put-Methoden Daten in der Datei gespeichert werden können. Abschließend muss die commit-Methode aufgerufen werden, da erst dann die Änderungen tatsächlich in die Datei aufgenommen werden. Um dann zu einem anderen Zeitpunkt wieder die gespeicherten Einstellungen zu lesen, wird wie folgt vorgegangen:

```
SharedPreferences settings = getSharedPreferences(USERDATA_PREF, MODE_PRIVATE);
sessionId = settings.getInt(USERDATA_PREF_SID, 0);
sessionKey = settings.getString(USERDATA_PREF_SKEY, "");
```

In diesem Beispiel werden die gespeicherten Session-ID und -Schlüssel aus den SharedPreferences gelesen. Hierbei erfolgt der Zugriff direkt über *get*-Methoden des *SharedPreferences*-Objekts. Diesen wird jeweils ein Standard-Wert mitgegeben, der dann zurückgegeben wird, wenn unter dem angegebenen Schlüssel (noch) keine Einstellung hinterlegt ist. In diesem Fall würde dies dann bedeuten, dass derzeit kein Benutzer angemeldet ist.

7.8 Performance-Optimierungen

Besonders auf einer mobilen Plattform mit ihren geringen Ressourcen (im Vergleich zu einem „normalen“ PC), sollte auf Optimierungen in Richtung geringem Speicherverbrauch und wenig Rechenaufwand für den Prozessor Wert gelegt werden. Auch wenn die Anwendung bereits schnell genug zu laufen scheint, bedeuteten derartige Performance-Optimierungen auch einen geringeren Energieverbrauch. Das spielt besonders bei den mobilen mit einem Akku betriebenen Geräten eine wesentliche Rolle.

Bei der Entwicklung wurde sich daher an den Empfehlungen von Google orientiert, wie sie auch in [28] nachgelesen werden können. Diese basieren auf den Ergebnissen durchgeführter Benchmark-Tests. In der Praxis wurde dabei besonders auf folgende Punkte Wert gelegt:

- Methoden, die nicht auf Attribute des Objekts zugreifen, sollten als *static* deklariert werden, da dies zu einem Geschwindigkeitsgewinn von 15-20% gegenüber virtueller Funktionen führt und auch eine hilfreiche Information innerhalb der Methoden-Signatur ist, um von „außen“ zu erkennen, dass hier keine Änderungen am Objekt vorgenommen werden.
- Der Zugriff auf Objekt-Eigenschaften über Getter und Setter benötigt 3- bis 7-mal so lange, wie der direkte Zugriff auf Variablen. Daher wurde sich auch bei dieser Softwareentwicklung für die Verwendung von *public* Variablen entschieden anstatt jede Eigenschaft über eine get-Methode zu holen bzw. via set-Methode zu ändern.
- Variablen, die später nicht mehr geändert werden müssen, sollten auch als solche – sprich als *final* – gekennzeichnet werden. Dadurch erspart man der VM zur Laufzeit das Zeitaufwendige nachschauen in entsprechenden Referenz-Listen. Auch wenn sich das nur für primitive Datentypen und String-Konstanten einen Geschwindigkeitsgewinn bedeutet, sorgt es bei der Entwicklung für eine bessere Übersicht, wenn auch andere Objekte, überall wo dies möglich ist, als final deklariert werden.
- Zum Durchlaufen eines Arrays sollte die dafür vorgesehene *for-each* Schleife einer selbst programmierten *for*-Schleife vorgezogen werden. Die einzige Ausnahme dabei ist die Iteration von *ArrayLists*, hier ist die manuell geschriebene *for*-Schleife bis zu 3-mal schneller.

7.9 Testen

Alle möglichen Android™-Geräte mit Tests abzudecken kann wohl als unmöglich angesehen werden. Da es dabei verschiedenste Zusammenstellungen sowohl seitens der Software als auch Hardware gibt, kann nicht ausgeschlossen werden, dass die App auf dem ein oder anderen Gerät nicht korrekt dargestellt wird oder möglicherweise sogar nicht richtig funktioniert. Dank der vom Android™-SDK bereitgestellten Emulatoren lassen sich jedoch einige Hardware-Kombinationen beim Einsatz verschiedener Software-Versionen zumindest unter simulierten Idealbedingungen testen. Diese Möglichkeit wurde auch hier genutzt, um zum einen die App von Android™ 1.6 bis hin zum neusten Android™ 4.0, zu dem in Deutschland aktuell gerade mal seit dem 2. Dezember 2011 ein erstes Gerät verfügbar ist [29]. Andererseits ist es bspw. auch möglich, Geräte mit unterschiedlicher Bildschirm-

auflösung zu emulieren. Zum Testen standen auf diese Weise folgende konkrete Gerätekonfigurationen zur Verfügung:

- Smartphone-Emulator mit Android 1.6 mit 240x320 Bildschirm Pixeln
- Smartphone-Emulator mit Android 1.6 mit 480x800 Bildschirm Pixeln
- Smartphone-Emulator mit Android 2.1 mit 480x800 Bildschirm Pixeln
- Smartphone-Emulator mit Android 2.2 mit 480x800 Bildschirm Pixeln
- Smartphone-Emulator mit Android 2.3.3 mit 480x800 Bildschirm Pixeln
- Tablet-Emulator mit Android 3.1 mit 1280x800 Bildschirm Pixeln
- Smartphone-Emulator mit Android 4.0 mit 480x800 Bildschirm Pixeln
- Tablet-Emulator mit Android 4.0 mit 1280x800 Bildschirm Pixeln
- Smartphone: Samsung Galaxy S I9000 mit Android 2.3.5 und 480x800 Bildschirm Pixeln
- Tablet: Dell Streak 5 mit Android 2.2 und 800x480 Bildschirm Pixeln
- Tablet: Samsung GalaxyTab 10.1V mit Android 3.0 und 1280x800 Bildschirm Pixeln

Während der Entwicklungsphase wurde primär mit der „echten“ Hardware getestet, da diese tendenziell schneller reagiert als ein Emulator. Abschließend wurde dann die folgende Testfolge auf den allen oben genannten Testgeräten absolviert, die jeweils eine „frische“ Installation ohne gespeicherte Daten auf dem jeweiligen Gerät voraussetzt. Des Weiteren werden serverseitig – genauso wie für die Tests des Webclients – die Testdaten in der Datenbank „example“ vorausgesetzt, wie sie in Anhang F zu finden sind. Außerdem beziehen sich die Anweisungen auf die deutschsprachige Version. Der Test auf englischsprachig eingestellten Geräten verläuft ansonsten analog.

1. Starten der App → Anzeige der Login-Activity
2. Menü öffnen → angezeigt werden die Menüpunkte „Einstellungen“ und „Über“
3. Eingabe ungültiger Benutzerdaten und Tipp auf den Anmelden-Button → Fehlermeldung bzgl. fehlgeschlagener Anmeldung.
4. Eingabe gültiger Benutzerdaten und Tipp auf den Anmelden-Button → Anzeige der Kunden-Suche mit leerem fokussiertem Eingabefeld
5. Menü öffnen → „Abmelden“, „Neue Suche“, „Einstellungen“, „Benutzer-Info“ und „Über“ werden als Menüpunkte angezeigt
6. Eingabe der Zeichenfolge „abc“ → nach der kurzzeitigen Sichtbarkeit von „Suche läuft …“ erscheint nichts weiter auf der Actiity (keine Treffer).
7. Löschen der vorherigen Eingabe und Eingabe von „hut“ → Die Kunden „Herr Jan Stalhut“, „Herr Christian Stalhut“ und „IT-Services Jan Stalhut“ erscheinen.
8. Tipp auf den Kunden „IT-Services Jan Stalhut“ → die Detailseite des Kunden mit den einzelnen Kategorien erscheint
9. Öffnen der Kategorie „Allgemeines“ → „Kunden-Typ“, „Unternehmen“, „Rechnung per“ und „Zahlungsweise“ werden angezeigt, diese lassen sich nicht antippen.
10. Öffnen der Kategorie „Adressen“ → Hinterlegte Adresse mit Anfahrtspauschale in Netto und Brutto wird angezeigt.
11. Tipp auf die Adresse → Auswahlfeld der möglichen Aktion (im Emulator die Meldung, dass keine passende Anwendung installiert ist)
    a. Tipp auf Maps → Anzeige der Adresse in Google Maps
    b. Betätigung der Zurück-Taste → Die letzte Darstellung der Kunden-Details wird wieder sichtbar

12. Öffnen der Kategorie „Webseiten“ → Anzeige der Webseiten-URL
13. Tippen auf die Webseiten-URL → Seite öffnet sich im Standard-Browser, oder Auswahl der installierten Browser erscheint, wobei nach Tipp auf einen dieser Browser die Webseite erscheint.
14. Betätigung der Zurück-Taste → Die letzte Darstellung der Kunden-Details wird wieder sichtbar
15. Öffnen der Kategorie „Ansprechpartner“ → Listeneintrag „Herr Jan Stalhut“ erscheint
16. Tipp auf den Ansprechpartner-Eintrag → die Detailseite der Person mit den einzelnen Kategorien erscheint
17. Öffnen der Kategorie „Allgemeines“ → „Name“, „Höflichkeitsform“ und „bei Anruf“ werden angezeigt, diese lassen sich nicht antippen.
18. Öffnen der Kategorie „Faxnummern“ → Anzeige der als geschäftlich markierten Faxnummer, diese lässt sich nicht antippen.
19. Öffnen der Kategorie „E-Mail-Adressen“ → Anzeige der drei hinterlegten E-Mail-Adressen
20. Tipp auf eine der E-Mail-Adressen → Eine neue E-Mail öffnet sich in der Standard Mail-App oder Auswahl der installierten Mail-Apps erscheint, wobei nach Tipp auf eine dieser Apps sich die neue E-Mail mit eingetragenem Empfänger öffnet.
21. Betätigung der Zurück-Taste → Die letzte Darstellung der Person-Details wird wieder sichtbar
22. Öffnen der Kategorie „Telefonnummern“ → Anzeige der drei hinterlegten Telefonnummern
23. Tipp auf eine der Telefonnummern → Die Telefonie-App startet und wählt die gewünschte Nummer an. (Auf Tablets ohne Telefonie-Funktion erscheint stattdessen das Angebot, die Telefonnummer ins Adressbuch aufzunehmen)
24. Öffnen der Kategorie „zugehörige Kunden“ → Anzeige der zwei Kundendatensätze „IT-Services Jan Stalhut“ und „(privat)“
25. Tipp auf den Eintrag „(privat)“ → Anzeige der Kunden-Details des Privatkunden „Jan Stalhut“
26. Betätigung der Zurück-Taste → Die letzte Darstellung der Person-Details wird wieder sichtbar
27. Betätigung der Zurück-Taste → Die letzte Darstellung der Kunden-Details von „IT-Services Jan Stalhut“ wird wieder sichtbar
28. Betätigung der Zurück-Taste → Die letzte Darstellung der Such-Activity, einschließlich der letzten Eingabe und der letzten Suchergebnisse, wird wieder sichtbar.
29. Betätigung der Home-Taste → Android™-Homescreen wird angezeigt.
30. Start der App → Die letzte Darstellung der Such-Activity, einschließlich der letzten Eingabe und der letzten Suchergebnisse, wird wieder sichtbar.
31. Drehen des Gerätes um 90° → Der Bildschirm-Inhalt dreht sich passend, der Inhalt der Such-Activity bleibt erhalten.
32. Drehen des Gerätes um -90° → Der Bildschirm-Inhalt dreht sich passend, der Inhalt der Such-Activity bleibt erhalten.
33. Beenden der App über die Anwendungsverwaltung der App und erneuter Start der App → Anzeige der Kunden-Suche mit leerem fokussiertem Eingabefeld
34. Öffnen der Einstellungen über das Menü → Ansicht mit Einstellungen wird sichtbar.
35. Änderung des Datenbanknamens in „example2“, Schließen der App über mehrmaliges Betätigen des Zurück-Buttons und erneuter Start der App → Anzeige der Kunden-Suche mit leerem fokussiertem Eingabefeld
36. Eingabe von „jan“ in das Suchfeld → Nach wenigen Sekunden Fehlermeldung, dass Server nicht erreichbar ist.

37. Änderung des Datenbanknamens in „example“, Schließen der App über mehrmaliges Betätigen des Zurück-Buttons und erneuter Start der App → Anzeige der Kunden-Suche mit leerem fokussiertem Eingabefeld
38. Öffnen der Benutzer über das Menü → Auflistung der aktiven Sitzungen nach wenigen Sekunden, Anzeige des eigenen Benutzernamens und der Benutzergruppe
39. Menü öffnen → „Abmelden“, „Neue Suche“, „Einstellungen“ und „Über“ werden als Menüpunkte angezeigt
40. Wählen der Option „Abmelden“ im Menü → Anzeige des Login-Formulars.

7.10 Veröffentlichung

Zur Veröffentlichung bzw. Installation der App auf beliebigen Android™-geräten muss ein Installationspaket erstellt werden. Dieses trägt die Dateiendung apk und kann direkt aus Eclipse heraus mit Hilfe des Android™-SDKs erstellt werden. Für die App wird dabei ein Zertifikat als „Ausweis“ des Entwicklers erstellt oder ggf. ein bereits generiertes Zertifikat zur Verwendung ausgewählt. Zur Installation bspw. durch Kopieren auf die SD-Karte eines Gerätes ist zu beachten, dass die Installation nur erfolgreich ist, wenn die Option für „unbekannte Anwendungsquellen“ in den System-Einstellungen deaktiviert ist. Der offizielle von Google empfohlene Weg für die Installation ist über den Android™-Market. Eine derartige Veröffentlichung macht jedoch (zum aktuellen Zeitpunkt) keinen Sinn für diese Anwendung, da niemand außer den eingeweihten Benutzern damit etwas anfangen könnte.

8 Validierung gegen die Anforderungen

Abschließend wurden die Ergebnisse der Softwareentwicklung gegen die ursprünglich in Kapitel 2 (Seite 3) festgelegten Anforderungen geprüft. So ließ sich am Ende sagen, dass alle genannten Anforderungen erfüllt und zum Teil sogar überschritten wurden:

So hieß es beispielsweise, dass es möglich sein soll, von Adressen und einzelnen Kontaktdaten-Typen jeweils bis zu 10 einem Kunden bzw. einer Person zuzuordnen. Durch die gewählte Datenbankstruktur ist es letztlich kein Problem, theoretisch beliebig viele Adressen und Kontaktdaten zu speichern. Es ist hier also keine harte Grenze bei 10 Datensätzen festgelegt, was den Vorteil hat, dass es kein Problem ist, wenn im Einzelfall z. B. doch 11 oder 12 Adressen zu einem Kunden hinterlegt werden müssen.

Durch die Sitzungsverwaltung wurde es nicht nur für den Android™-Client ermöglicht, dass man angemeldet bleiben kann. Dank dem LocalStorage von HTML5 war es auch kein großer Aufwand, dies im Webclient zu implementieren, da die JSON-Schnittstelle diese Funktionalität sowieso für den Android™-Client bereitstellen muss.

Bei der Sicherheit wurde durch die Nutzung des PHP-Frameworks phpass zur Passwortverschlüsselung und durch die Richtlinien zur Kontosperrung bei mehreren Fehllogins ein hoher Grad an Sicherheit erreicht. So das die Sicherheit des Systems im Wesentlichen noch von der Komplexität der durch den Benutzer gewählten Passwörter abhängt.

Getestet wurde wie gefordert auf allen gängigen Webbrowsern und auf verschiedenen Betriebssystemen, sodass ein Reibungsloser Betrieb auf allen Plattformen möglich sein sollte. Ergänzend wurde hier sogar stichprobenartig getestet, wie weit sich mit den einzelnen Browser-Versionen zurückgehen lässt, um diese dann als Mindestversion in die Liste der kompatiblen Webbrowser aufzunehmen. Sie wird angezeigt, wenn der Webclient einmal mit einem nicht kompatiblen Webbrowser geöffnet wird.

Beim Android™-Client wurden ebenfalls alle geforderten Plattform-Versionen getestet und ergänzend auch Geräte mit unterschiedlich großen Displays in den Test mit aufgenommen, sodass die App auf allen gängigen Android™-Geräten lauffähig sein sollte. Auf Grund verschiedener Anpassungen des Android™-Systems durch Geräte-Hersteller kann keine Gewähr übernommen werden.

Eine weitere Überschreitung der Anforderungen ist bei der quasi als Nebenprodukt entstandenen Chrome-App zu sehen. Wodurch Nutzer des Browsers Chrome (und kompatiblen Abspaltungen) in Genuss einer etwas performanteren Version des Webclients kommen, indem dieser lokal im Browser verfügbar gemacht wird.

9 Schluss

Zusammenfassend lässt sich sagen, dass in dem Zeitrahmen der Bachelor-Thesis erfolgreich das gewünschte Software-Produkt geplant, entwickelt und in Betrieb genommen wurde. Es entstand ein Gesamtkonzept für Serveranwendung mit verschiedenartigen Clients. Durch den gewählten Ansatz der recht flexiblen Kommunikation über JSON-Objekte sollte einer Erweiterung um zusätzliche Funktionen oder auch die Entwicklung weiterer Clients (z. B. für die iOS-Plattform) kein großes Problem darstellen.

9.1 Ausblick

Im aktuellen Zustand ist die Software bereits gut für den produktiven Einsatz geeignet, es wäre allerdings noch die Portierung der vorhandenen Kundendaten zu realisieren, was bedingt durch eine absolut „unnormalisierte“ Excel-Tabelle noch die ein anderen Probleme bereiten dürfte und sich schließlich wohl auch nicht vollständig automatisieren lassen wird.

Für die Zukunft der Software ist – wie bereits erwähnt – eine Erweiterung um eine Rechnungsverwaltung angedacht, sodass Rechnungen für konkrete Kunden erstellt, verschickt und archiviert werden können. Eine weitere praktische Erweiterung, besonders beim mobilen Android™-Client für den Außendienstmitarbeiter, könnte die Möglichkeit der Zeiterfassung sein. So könnte dann in Folge dessen auch bei der Rechnungserstellung direkt auf den eingetragenen Zeitaufwand zurückgegriffen werden.

Auch eine Benutzerverwaltung innerhalb des Webclients wäre hilfreich, um die Benutzer nicht manuell in der Datenbank eintragen zu müssen und sie dann auch komfortabler einer Benutzer-Gruppe zuordnen zu können.

In der Datenbankstruktur sollte die Möglichkeit ergänzt werden, dass eine der Kontaktpersonen eines Kunden als Standard festgelegt werden kann und auch für Kontaktpersonen sollte es wiederum möglich sein, bei mehreren Telefonnummern, eine als Default festzulegen. (Ähnliches gilt dann auch für Faxnummern, E-Mail-Adressen und Adressen.)

Für den Android™-Client könnte auch die Implementierung einer Caching-Funktionalität in Erwägung gezogen werden, sodass auch noch auf Daten zugegriffen werden kann, wenn man sich mit dem Smartphone einmal an einem Ort befindet, wo mangels Mobilfunkverfügbarkeit keine mobile Internetverbindung bereitsteht.

Auch wenn die Datenpflege wohl immer primär am PC mit einer „richtigen“ Tastatur stattfinden wird, könnte es dennoch hilfreich sein, auch von unterwegs am Smartphone einmal eine kleinere Änderung bzw. Ergänzung im Datenbestand der Kunden vorzunehmen.

Der Chrome-App würde eine Konfigurations-Seite (ähnlich wie der in der Android-App) gut stehen. Unter anderem damit nicht für jede Plattform-Instanz (Datenbank) eine eigenes individuelles Paket erstellt werden muss, sondern sich der Datenbank-Name dann einstellen lässt.

Was für die Android™-Anwendung quasi selbstverständlich ist, weil es durch die entsprechenden Vorbereitungen im System keinen großen Aufwand bedeutet, ist die Mehrsprachigkeit der Anwendung. Dies wäre dementsprechend auch eine sinnvolle Erweiterung für den Webclient, wenn dieser in verschiedenen Sprachen (zumindest noch zusätzlich englisch) verfügbar gemacht werden würde. Bei einer vollständigen Unterstützung für unterschiedliche Sprachen wäre allerdings auch das Daten-

bank-Design zu überdenken, da dort verschiedene ENUM-Datentypen mit festen deutschsprachigen Werten definiert sind.

Schließlich wären natürlich auch weitere Apps für nicht Android™-basierte mobile Geräte, wie iPhone/iPad, Backbarry und Windows Mobile/Phone eine Überlegung wert. Auf diese Weise kann eine komfortablere und besser angepasste Benutzeroberfläche bereitgestellt werden, als es mit einer Weboberfläche im Browser machbar ist.

9.2 Fazit

Bei der Realisierung des vorliegenden Software-Produkts konnten einige Erfahrungen in den Bereichen PHP-, Datenbanken-, Webseiten- und Android™-Entwicklung gesammelt werden. Insbesondere die Vielfalt der eingesetzten Programmiersprachen stellte eine interessante Herausforderung dar. Auch die Aufgabe, eine Weboberfläche zu erstellen, die möglichst auf den verschiedensten Geräten und damit Display-Größen gut bedienbar sein sollte, stelle sich als gar nicht so einfach heraus. Zusätzlich erschwert wurde dies durch teilweise unvollständige Unterstützung einzelner CSS-Parameter oder bspw. der fehlenden Unterstützung des Scrollens für innere Elemente bei den Browsern auf Smartphones.

Sobald die bestehenden Kundendaten in die Anwendung portiert worden sind, steht dem eigenen produktiven Einsatz der Anwendung nichts mehr im Weg und sollte an mancher Stelle im Arbeitsablauf eine deutliche Erleichterung darstellen. So lässt sich abschießend sagen, dass sich der Aufwand für eine solche Anwendung gelohnt hat: Einmal durch die praktischen Erfahrungen, die während der Entwicklung gesammelt werden konnten, aber auch durch den zukünftigen Nutzen der fertigen Anwendung: Zum einen lassen sich die Kundendaten über die Weboberfläche komfortabler pflegen, als in einer Excel-Tabelle und zum Anderen stehen hier auch mehr Möglichkeiten Zur Verfügung. So war es in der Excel-Tabelle immer sehr unübersichtlich wenn mehrere Personen in einer Zeile zu einem Kunden eingetragen wurden von einer Zuordnung jeweiliger Telefonnummern ganz zu schweigen. Auch ist es natürlich unterwegs deutlich praktischer in der neuen App die Kontaktdaten zum nächsten Kundentermin heraus zu suchen, als die Excel-Tabelle auf dem Smartphone zu öffnen, wobei einige Sekunden ins Land gingen und von Übersichtlichkeit beim Betrachten keine Rede gewesen sein kann.

Schließlich ist mit dieser App auch der erste Schritt in Richtung einer komfortableren Rechnungsverwaltung und Zeiterfassung getan. Dies sind ebenfalls Dinge die aktuell noch mit Hilfe von Excel-Tabellen bzw. Word-Dateien gemanagt werden, so dass sich auch hier dann durch eine vernünftige spezialisierte Anwendung viel Zeit einsparen lässt, die man sonst mit lästigen Büro-Arbeiten verbringt.

A. Literaturverzeichnis

[1] sueddeutsche.de GmbH, „Android-Smartphones werden zur Billig-Alternative," 15 Oktober 2011. [Online]. Available: http://www.sueddeutsche.de/digital/google-betriebssystem-android-smartphones-werden-zur-billig-alternative-1.1163344. [Zugriff am 18 November 2011].

[2] P. Zschunke, „Preisrutsch bei Android," *Weser Kurier,* 20. Oktober 2011.

[3] SPIEGEL ONLINE GmbH, „Android läuft auf jedem zweiten Smartphone," 15 November 2011. [Online]. Available: http://www.spiegel.de/wirtschaft/unternehmen/0,1518,797986,00.html. [Zugriff am 21 November 2011].

[4] I. Steinhaus, „Eingeschränkt tauglich: iOS und Android im Unternehmen," 17 November 2011. [Online]. Available: http://www.mobilebusiness.de/nc/home/news-detail/article/eingeschraenkt-tauglich-ios-und-android-im-unternehmen.html. [Zugriff am 18 November 2011].

[5] A. Limburg, „Tut er's oder tut er's nicht?," *Android 360,* pp. 45-51, 2011.

[6] H. Braun, „Internet Explorer 9 erscheint Dienstag früh," 10 März 2011. [Online]. Available: http://heise.de/-1205774. [Zugriff am 21 November 2011].

[7] F. M. Palinkas, „Opera 11.00 for Windows changelog," [Online]. Available: http://www.opera.com/docs/changelogs/windows/1100/. [Zugriff am 21 November 2011].

[8] Mozilla, „Releases," 19 November 2011. [Online]. Available: https://wiki.mozilla.org/Releases. [Zugriff am 21 November 2011].

[9] G. Himmelein, „Chrome 14 führt C-Code im Browser aus," 17 September 2011. [Online]. Available: http://heise.de/-1345167. [Zugriff am 21 November 2011].

[10] J. Bager, „Safari 5 ist da," 8 Juni 2010. [Online]. Available: http://heise.de/-1017350. [Zugriff am 21 November 2011].

[11] A. Vahldiek, „Service Pack 3 für Windows XP nun auch offiziell zum Download," 7 Mai 2008. [Online]. Available: http://heise.de/-205734. [Zugriff am 21 November 2011].

[12] K. Stewart, „LucidReleaseSchedule," 14 Juni 2011. [Online]. Available: https://wiki.ubuntu.com/LucidReleaseSchedule. [Zugriff am 21 November 2011].

[13] A. Beier, „Leopard in freier Wildbahn," 26 Oktober 2007. [Online]. Available: http://heise.de/-189742. [Zugriff am 21 November 2011].

[14] Google Inc., „Android 1.6 Platform," [Online]. Available:

http://developer.android.com/sdk/android-1.6.html. [Zugriff am 27 November 2011].

[15] Google Inc., „Android 4.0 Platform," [Online]. Available: http://developer.android.com/sdk/android-4.0.html. [Zugriff am 21 November 2011].

[16] K. Nicholas, „Die Android History: Vom Anfang bis zur Gegenwart und warum Ice Cream Sandwich so fantastisch ist," 3 November 2011. [Online]. Available: http://www.androidpit.de/de/android/blog/399317/Die-Android-History-Vom-Anfang-bis-zur-Gegenwart-und-warum-Ice-Cream-Sandwich-so-fantastisch-ist. [Zugriff am 21 November 2011].

[17] Nielsen, „Android Leads in U.S. Smartphone Market Share and Data Usage," 21 Mai 2011. [Online]. Available: http://blog.nielsen.com/nielsenwire/consumer/android-leads-u-s-in-smartphone-market-share-and-data-usage/. [Zugriff am 21 November 2011].

[18] SPIEGEL ONLINE GmbH, „Android überholt Apple iOS in Deutschland," 9 Mai 2011. [Online]. Available: http://www.spiegel.de/netzwelt/netzpolitik/0,1518,761434,00.html. [Zugriff am 21 November 2011].

[19] A. Becker und M. Pant, Android 2 - Grundlagen und Programmierung, Heidelberg: dpunkt.verlag, 2010.

[20] Institut für Mobile und verteilte Systeme, „Einblick in die Dalvik Virtual Machine," 2009. [Online]. Available: http://www.fhnw.ch/technik/imvs/publikationen/fokus-report/2009/view. [Zugriff am 21 November 2011].

[21] H. Mosemann und M. Kose, Android: Anwendungen für das Handy-Betriebssystem erfolgreich programmieren, München: Carl Hanser Verlag GmbH & CO. KG, 2009.

[22] Google Inc., „Intent," [Online]. Available: http://developer.android.com/reference/android/content/Intent.html. [Zugriff am 24 November 2011].

[23] A. Scherbaum, PostgreSQL: Datenbankpraxis für Anwender, Administratoren und Entwickler, München: Open Source Press, 2009.

[24] The PHP Group, „PostgreSQL," [Online]. Available: http://www.php.net/manual/de/book.pgsql.php. [Zugriff am 29 November 2011].

[25] A. Kelz, „Der Königsweg: Normalisierung," 16 November 1998. [Online]. Available: http://v.hdm-stuttgart.de/~riekert/lehre/db-kelz/chap4.htm. [Zugriff am 29 November 2011].

[26] The PHP Group, „JavaScript-Objekt-Notation," [Online]. Available: http://de2.php.net/manual/de/book.json.php. [Zugriff am 29 November 2011].

[27] heise Security, „Cracker-Bremse," 3 Juni 2011. [Online]. Available: http://heise.de/-1253931. [Zugriff am 30 November 2011].

[28] Google Inc., „Designing for Performance,“ [Online]. Available: http://developer.android.com/guide/practices/design/performance.html. [Zugriff am 10 Dezember 2011].

[29] COMPUTER BILD Digital GmbH, „Verkausstart Samsung Galaxy Nexus,“ 2 Dezember 2011. [Online]. Available: http://www.computerbild.de/artikel/cb-News-Handy-Samsung-Galaxy-Nexus-Nexus-Prime-Media-Markt-6875062.html. [Zugriff am 7 Dezember 2011].

[30] Google Inc., „Press/Media,“ [Online]. Available: http://www.android.com/media/. [Zugriff am 15 November 2011].

B. Abkürzungsverzeichnis

ARM	Advanced RISC Machines
bspw.	beispielsweise
bzw.	beziehungsweise
CLIR	Calling Line Identification Restriction
CSS	Cascading Style Sheets
DVM	Dalvik Virtual Machine
FTP	File Transfer Protocol
ggf.	gegebenenfalls
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
ICS	Ice Cream Sandwich
ID	Identifikator
IDE	Integrated Development Environment
JSON	JavaScript Object Notation
JVM	Java Virtual Machine
LAN	Local Area Network
NAT	Network Address Translation
PC	Personal Computer
PHP	PHP: Hypertext Preprocessor
PLZ	Postleitzahl
SDK	Software Development Kit
SQL	Structured Query Language
SSL	Secure Sockets Layer
SVN	Subversion
TCP	Transmission Control Protocol
u. a.	unter anderem
UI	User Interface
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
WLAN	Wireless Local Area Network
XML	Extensible Markup Language
z. B.	zum Beispiel

C. Abbildungsverzeichnis

D. Eidesstattliche Erklärung

Ich erkläre hiermit, dass ich die vorliegende Arbeit selbstständig ohne fremde Hilfe verfasst habe und keine anderen als die angegebenen Hilfsmittel von mir verwendet wurden.

Alle wörtlichen oder sinngemäßen Übernahmen aus anderen Werken wurden von mir als solche kenntlich gemacht.

Ort, Datum Unterschrift

E. JSON Schnittstellendefinition (bei Version 0.6.x)

Basis-Daten jeder Anfrage:

```
data = {
    "cl": clientName, // something like "ChromeApp" or "WebClient"
    "clV": clientVersion, // something like "0.6.0"
    "auth": autenticationData
};

/* authentication without session */
autenticationData = {
    "username": currentUsername, //username to login
    "password": currentPassword, //password for user-login
    "save": createSessionBoolean //optional, create session or not
};

/* authentication with exist session */
autenticationData = {
    "sid": sessionId, // an integer id like 123
    "skey": sessionKey // someting like "4edb34f73eaf95.46485567"
};
```

Ergänzende Angaben je nach gewünschter Aktion:

```
data["a"] = true; // only authentication request

data["ad"] = true; // logout request, delete current session

data["q"] = searchString; // something for customer searching like "it-services"

data["qc"] = searchString; // something for person searching like "stalhut"

data["p"] = personId; // loading details for person with this database id

data["c"] = customerId; // loading details for customer with this database id

data["e"] = customerId; // loading details for customer with this database id
                        // incl. persons for edit

data["ep"] = personId; // loading details for person with this database id
                      // for edit

data["s"] = { // saving the (new) customer details
    "company": nameOfCompany, // comapny name like "IT-Services Jan Stalhut"
    "customerId": databaseId, // db id of an axists customer, null when it's new.
    "customerType": customerType, // type of customer: "gewerblich" or "privat"
    "billingDelivery": deliveryType, // type for billing delivery: "E-Mail",
                                     // "Post" or "Telefax"
    "paymentMethod": paymentType, // type of payment method: "Barzahlung",
                                  // "Überweisung", "PayPal"
    "note": customerNote, // some additional notes for the customer dataset
    "website": [{ // array with websites
        "url": websiteUrl, // string with an website URL, like "http://www..."
        "note": websiteNote, // some additional notes for this website
        "id": websiteId // database id of an exists website, null when it's new
    }, ...],
    "address": [{ // array with addresses
```

```
        "street": streetName, // street name, like "Huder Str."
        "streetNo": houseNumber, // house number, like "61c"
        "postalcode": postalcode, // postalcode, like "28197"
        "city": city, // city, like "Bremen"
        "district": district, // dsitrict of the city, like "Woltmershausen"
        "country": country, // country, like "Deutschland"
        "travelCost": travelCost, // travel costs, like "10,08" or as float 10.08
        "note": addressNote, // some additional notes for this address
        "id": addressId // database id of an exists address, null when it's new
    },_...],
    "contact": [{ // array of contact persons
        "lastname": lastname, // lastname of the person
        "firstname": firstname, // firstname of the person
        "id": personId, // database id of an exists person, null when it's new
        "title": title, // title of the person: "Herr" or "Frau"
        "pronominalTitle": pronominalTitle, // pronominal title: "duzen" or
                                            // "siezen"
        "transmitMobileNumber": transmitNumber, // trasnmit (true) or
                                                // hide (false) own number
        "note": contactpersonNote, // some additional notes for this person
        "telephone": [{ // array of telephone numbers
            "numberCountry": countryCode, // country code for telehopne, like 49
            "numberPrefix": numberPrefix, // telephone prefix (city code),
                                          // like 421
            "number": number, // main telephone number
                   "numberExt": numberExtension, // number extension
            "note": telephoneNote, // some additional notes for this number
            "id": telephoneId, // database id of an exists telephonenumber,
                               // null when it's new
            "type": telephoneType // type of telephone: "privat", "geschäftlich"
                                  // or "mobil"
        },_...],
        "telefax": [{ // array of telefax numbers
            "numberCountry": countryCode, // country code for telefax, like 49
            "numberPrefix": numberPrefix, // telefax prefix (city code), like 421
            "number": number, // main telefax number
                   "numberExt": numberExtension, // number extension
            "id": telefaxId, // database id of an exists telephonenumber,
                             // null when it's new
            "type": telefaxType // type of telefax: "privat" or "geschäftlich"
        },_...],
        "email": [{ // array with email addresses
            "address": mailAddress, // full mail address
            "id": emailId, // database id of an exists mailaddress,
                           // null when it's new
            "type": emailType // type of mailaddress: "privat" or "geschäftlich"
        },_...]
    },_...] ,
    "delWebsites": [], // Optional array with IDs of deleted websites
    "delAddresses": [], // Optional array mit IDs of deleted addresses
    "delContacts": [], // Optional array mit IDs of removed cnóntacts
    "delTelephones": [], // Optional array mit IDs of deleted telephones
    "delTelefaxes": [], // Optional array mit IDs of deleted telefaxes
    "delEmails": [] // Optional array mit IDs of deleted emails
}

data["us"] = true; // requests the list of active sessions

data["ds"] = sessionId; // deleting the session with this database id
```

```
data["pw"] = { // changing the user password
    "old": currentPassword, // the current/old user password
    "new": newPassword // a new user password
};

data["l"] = stringForLogging; // serverside logging
```

Basis-Daten jeder Server-Antwort:

```
res = {
      "authOk": boolAuth // Authentifizierung successful (true) or failed
}
```

Zusätzliche Antwort-Daten, bei reiner Authentifizierungsanfrage:

```
res["userinfo"] = {
    "id": userId, // database id of the logged in user as int
    "username": username, // username of the logged in user
    "usergroup": usergroupname, // name of the usergroup
    "allowedit": editAllowed // editing allowed for current user (true or false)
}
```

Zusätzliche Antwort-Daten, bei Logoutanfrage:

```
res["logoutOk"] = success; // logout successful (true)
```

Zusätzliche Antwort-Daten, bei Suche nach Kunden:

```
res["business"] = [{ // array with hits of business customers
    "id": customerId, // database id of the customer as int
    "company": companyName, // comapany name of the customer
    "contact": contactName // full name incl. title of one contact
}, ...]

res["private"] = [{ // array with hits of private customers
    "id": customerId, // database id of the customer as int
    "fullname": contactName // full name incl. title of one contact
}, ...]
```

Zusätzliche Antwort-Daten, bei Suche nach Personen:

```
res["persons"] = [{ // array with hits of persons
    "id": personId, // database id of the person as int
    "fullname": contactName // full name incl. title of the person
}];
```

Zusätzliche Antwort-Daten, beim Laden von Personen-Details:

```
res["person"] = {
    "id": personId, // database id of person as int
    "pronominal": pronominalTitle, // pronominal title: "duzen" or "siezen"
    "note": personNote, // some additional notes of the person
    "transmitmobile": transmitNumber, // transmit (true) or hide (false) own
                                      // mobile number
    "fullname": personName, // full name incl. title of the person
    "telephones": [{ // array with telephone numbers
        "id": telephoneId, // database id of telephone number as int
        "type": telephoneType, // type of telephone: "privat", "geschäftlich"
```

```
                                // or "mobil"
        "note": telephoneNote, // some note for telephone number
        "number": telephoneRead, // for reading formated telephone number
        "numberdial": telephoneCall // for calling formatted telephone number
    }, ...],
    "telefaxes": [{ // array with telefax numbers
        "id": telefaxId, // database id of telefax number as int
        "type": telefaxType, // type of telephone: "privat" or "geschäftlich"
        "note": telefaxNote, // some note for telefax number
        "number": telefaxRead, // for reading formated telephone number
        "numberdial": telefaxCall // for calling formatted telephone number
    }, ...],
    "emails": [{ // array with email addresses
        "id": emailId, // database id of email address as int
        "type": emailType, // type of telephone: "privat" or "geschäftlich"
        "note": emailNote, // some note for email address
        "address": emailAddress // full email address
    }, ...],
    "customers": [{ // array with connected customers
        "id": customerId, // database id of customer as int
        "company": comapnyName // name of company or "(privat)"
    }, ...]
}
```

Zusätzliche Antwort-Daten, beim Laden vom Kunden-Details:

```
res["customer"] = {
    "id": customerId, // database id of the customer as int
    "type": customerType, // type of customer: "gewerblich" or "privat"
    "company": nameOfCompany, // comapny name
    "note": customerNote, // some additional notes for the customer dataset
    "billingdelivery": deliveryType, // type for billing delivery: "E-Mail",
                                     // "Post" or "Telefax"
    "paymentmethod": paymentMethod, // type of payment method: "Barzahlung",
                                    // "Überweisung", "PayPal"
    "websites": [{ // array with websites
        "id": websiteId, // database id of website as int
        "note": websiteNote, // additional note for website
        "url": websiteUrl // website URL
    }, ...],
    "addresses": [{ // array with addresses
        "id": addressId, // database id of address as int
        "street": streetName, // street of the address
        "houseno": houseNumber, // housenumber of the address
        "postalcode": postalcode, // postalcode of the address
        "district": district, // district of the city
        "city": city, // city of the address
        "country": country, //country of the address
        "note": addressNote, // additional note for address
        "travelcost": travelCost // travel costs as an float
    }, ...],
    "persons": [{ // array with connected persons
        "id": personId, // database id of contactperson as int
        "fullname": personName, // full name incl. title of the person
        "note": personNote // additional note for person
    }, ...]
}
```

Zusätzliche Antwort-Daten, beim Laden von Kunden- mit Personen-Daten zur Bearbeitung:

```
res["customer"] = {
    "id": customerId, // database id of the customer as int
    "type": customerType, // type of customer: "gewerblich" or "privat"
    "company": nameOfCompany, // comapny name
    "note": customerNote, // some additional notes for the customer dataset
    "billingdelivery": deliveryType, // type for billing delivery: "E-Mail",
                                     // "Post" or "Telefax"
    "paymentmethod": paymentMethod, // type of payment method: "Barzahlung",
                                    // "Überweisung", "PayPal"
    "websites": [{ // array with websites
        "id": websiteId, // database id of website as int
        "note": websiteNote, // additional note for website
        "url": websiteUrl // website URL
    }, ...],
    "addresses": [{ // array with addresses
        "id": addressId, // database id of address as int
        "street": streetName, // street of the address
        "houseno": houseNumber, // housenumber of the address
        "postalcode": postalcode, // postalcode of the address
        "district": district, // district of the city
        "city": city, // city of the address
        "country": country, //country of the address
        "note": addressNote, // additional note for address
        "travelcost": travelCost // travel costs as float
    }, ...],
    "persons": [{ // array with connected persons
        "id": personId, // database id of person as int
        "pronominal": pronominalTitle, // pronominal title: "duzen" or "siezen"
        "note": personNote, // some additional notes of the person
        "transmitmobile": transmitNumber, // transmit (true) or hide (false) own
                                          // number
        "title": "Herr", // title of the person
        "lastname": "Stalhut", // lastname of the person
        "firstname": "Jan", // firstname of the person
        "telephones": [{ // array with telephone numbers
            "id": telephoneId, // database id of telephone number as int
            "type": telephoneType, // type of telephone: "privat", "geschäftlich"
                                   // or "mobil"
            "note": telephoneNote, // some note for telephone number
            "number": mainNumber, // main telephone number as int
            "prefix": numberPrefix, // telephone prefix as int
            "countrycode": countryPrefix, // country prefix as int
            "extension": numberExtension // extension as int, or null
        }, ...],
        "telefaxes": [{ // array with telefax numbers
            "id": telefaxId, // database id of telefax number as int
            "type": telefaxType, // type of telephone: "privat" or "geschäftlich"
            "note": telefaxNote, // some note for telefax number
            "number": mainNumber, // main telefax number as int
            "prefix": numberPrefix, // telefax prefix as int
            "countrycode": countryPrefix, // country prefix as int
            "extension": numberExtension // extension as int, or null
        }, ...],
        "emails": [{ // array with email addresses
            "id": emailId, // database id of email address as int
            "type": emailType, // type of telephone: "privat" or "geschäftlich"
            "note": emailNote, // some note for email address
            "address": emailAddress // full email address
        }, ...]
    }, ...]
```

```
}
```

Zusätzliche Antwort-Daten, beim Laden von Personen-Daten zur Bearbeitung:

```
res["person"] = {
    "id": personId, // database id of person as int
    "pronominal": pronominalTitle, // pronominal title: "duzen" or "siezen"
    "note": personNote, // some additional notes of the person
    "transmitmobile": transmitNumber, // transmit (true) or hide (false) own
                                      // number
    "title": "Herr", // title of the person
    "lastname": "Stalhut", // lastname of the person
    "firstname": "Jan", // firstname of the person
    "telephones": [{ // array with telephone numbers
        "id": telephoneId, // database id of telephone number as int
        "type": telephoneType, // type of telephone: "privat", "geschäftlich" or
                               // "mobil"
        "note": telephoneNote, // some note for telephone number
        "number": mainNumber, // main telephone number as int
        "prefix": numberPrefix, // telephone prefix as int
        "countrycode": countryPrefix, // country prefix as int
        "extension": numberExtension // extension as int, or null
    }, ...],
    "telefaxes": [{ // array with telefax numbers
        "id": telefaxId, // database id of telefax number as int
        "type": telefaxType, // type of telephone: "privat" or "geschäftlich"
        "note": telefaxNote, // some note for telefax number
        "number": mainNumber, // main telefax number as int
        "prefix": numberPrefix, // telefax prefix as int
        "countrycode": countryPrefix, // country prefix as int
        "extension": numberExtension // extension as int, or null
    }, ...],
    "emails": [{ // array with email addresses
        "id": emailId, // database id of email address as int
        "type": emailType, // type of telephone: "privat" or "geschäftlich"
        "note": emailNote, // some note for email address
        "address": emailAddress // full email address
    }, ...]
}
```

Zusätzliche Antwort-Daten, beim Speichern von Kunden-Details:

```
res["savedCustomerId"] = customerId // id of saved customers
res["deletedCustomerId"] = [] // array of ids of deleted customers
res["savedAddressId"] = [] // array of ids of saved addresses
res["deletedAddressId"] = [] // array of ids of deleted addresses
res["savedWebsiteId"] = [] // array of ids of saved websites
res["deletedWebsiteId"] = [] // array of ids of deleted websites
res["savedPersonId"] = [] // array of ids of saved persons
res["deletedPersonId"] = [] // array of ids of deleted persons
res["savedContactPerson"] = [] // array of ids of conntected contact persons
res["notSavedContactPerson"] = [] // array of ids of not connected persons
res["removedPersonId"] = [] // array of ids of removed persons
res["savedTelephoneId"] = [] // array of ids of saved telephones
res["deletedTelephoneId"] = [] // array of ids of deleted telephones
res["savedTelefaxId"] = [] // array of ids of saved telefaxes
res["deletedTelefaxId"] = [] // array of ids of deleted telefaxes
res["savedEmailId"] = [] // array of ids of saved email addresses
res["deletedEmailId"] = [] // array of ids of deleted email addresses
```

Zusätzliche Antwort-Daten, beim Laden der Liste aktiver Sessions:

```
res["sessions"] = [{ // array of current user sessions
    "id": sessionId, // database id of session as int
    "client": clientName, // name of client that uses the session
    "clientinfo": clientInfo, // additional slient info
    "starttime": startingTime, // date and time session start
    "lastactivity": lastActivityTime // date and time of last activity
}, ...]
```

Zusätzliche Antwort-Daten, beim Löschen einer aktiven Session:

```
res["deletedUsersessionId"] = sessionId // id of deleted session
```

Zusätzliche Antwort-Daten, bei Änderung des Benutzer-Passworts:

```
res["pwChangeOk"] = success // password changing successful
```

F. Quellcode

Alle erwähnten und im Rahmen dieser Bachelor-Thesis entstandenen Quellcodes finden sich auf der beiliegenden CD-ROM. Auf der CD findet sich eine Gliederung wie folgt:

- **Datenbank:** SQL-Dateien, zur Generierung der Tabellenstruktur, PL/pgSQL-Funktionen und Sichten der Datenbank.
- **Serveranwendung:** Das in PHP entwickelte serverseitig ablaufende Skript, das die Schnittstelle zwischen der Datenbank und JSON darstellt.
- **Webclient:** Die HTML, CSS, JavaScript und Grafik-Dateien, aus denen die Weboberfläche besteht.
- **ChromeApp:** Dateien, die für die Chrome-App nötig sind. Im Wesentlichen sind dies die gleichen wir für den Webclient.
- **AndroidApp:** Alle zur Android™-Anwendung gehörenden Quelldateien und generiertem JavaDoc, einschließlich der XML-Dateien für Layout, Zeichenketten und Menüs entsprechend der Verzeichnisstruktur, wie sie durch das Android™-SDK vorgegeben wird.