Karlsruher Institut für Technologie

IISM

Entwicklung eines mobilen Software-Assistenten zur Unterstützung der vernetzten Pflege

Master-Arbeit
von

Thomas Knapp

An der Fakultät für Wirtschaftswissenschaften
Institut für Informationswirtschaft und -management (IISM)
Information & Market Engineering

Gutachter:	Prof. Dr. rer. pol. Christof Weinhardt
Betreuender Assistent:	Dr. Henner Gimpel
Betreuende Mitarbeiter FZI:	Bruno Rosales Saurer
	Mathias Schmon
	Imanol Bernabeu

31. März 2012

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungsverzeichnis

VERAH	Versorgungsassistentin in der Hausarztpraxis
MFP	Medizinische Fachpflegekraft
DRK	Deutsches Rotes Kreuz
GPS	Global Positioning System
WLAN	Wireless Local Area Network
FZI	Forschungszentrum Informatik
REST	Representational State Transfer
URI	Uniform Resource Identifier
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
XML	Extensible Markup Language
HTML	Hypertext Markup Language
CSS	Cascading Style Sheets
JSON	Java Script Object Notation
App	Application
ID	Identifier

1. Einleitung

Nach Schätzungen des Statistischen Bundesamtes steigt der Anteil der über 65jährigen in der Bundesrepublik Deutschland von 19% im Jahr 2005 auf 30% im Jahr 2050 [Pre06] bei gleichzeitig steigender Lebenserwartung. Dieser demographische Wandel birgt große Herausforderungen für das Gesundheitswesen, da eine größere Anzahl an Patienten mit altersbedingten Krankheiten versorgt werden muss, die häufig multimorbide und immobil sind.
Insbesondere Hausarztpraxen als zentrale Anlaufstellen für Beschwerden im Alter stehen vor neuen Aufgaben. Wo früher nur Heilkunde angewandt wurde, werden heute beispielsweise auch koordinative Aufgaben zwischen Fachärzten, Pflegediensten und Apotheken übernommen. Mit einer gestiegenen Anzahl an pflegebedürftigen Personen wird sich dieser Trend weiter verstärken. Hinzu kommt, dass häufigere Immobilität der älteren Patienten von den Hausärzten mehr zeitaufwendige Hausbesuche abverlangen wird. Dem gegenüber steht eine sinkende Anzahl an Hausärzten, da auch deren Altersstruktur vom demographischen Wandel geprägt ist. Nach Schätzungen wird die Anzahl der Hausärzte bis 2020 bundesweit um etwa 7000 zurückgehen, was einem Anteil von 13.3% entspricht ([Kop10, S. 65]). Überdurchschnittlich hoch wird der Rückgang an Hausärzten in den ländlichen Regionen der neuen Bundesländer ausfallen ([Kop10, S. 52 ff.]). Der Rückgang der Hausärzte wird somit bundesweit zu signifikanten Versorgungsengpässen führen, die kurzfristig nicht durch junge Ärzte ausgeglichen werden können.

Um auch in Zukunft eine ausreichende Versorgung von Patienten gewährleisten zu können, wurde vom Gemeinsamen Bundesausschuss eine Richtlinie herausgegeben, die Rahmenbedingungen formuliert, in denen ärztliche Tätigkeiten auf Berufsangehörige der Alten- und Krankenpflege übertragen werden können [ric12]. Diese Maßnahme soll Hausärzte bei Tätigkeiten entlasten, für die nicht unbedingt ärztliche Expertise notwendig ist. Insbesondere die Anzahl der vom Arzt durchzuführenden Hausbesuche könnte sich hierdurch verringern. Ärztliche Tätigkeiten können dabei sowohl an Arzthelfer in einer Praxis, Mitarbeiter eines ambulanten Pflegedienstes, Pfleger in einem Alten- und Pflegeheim sowie Gesundheits-

und Krankenpflegern in einem Krankenhaus delegiert werden. Vor der Herausgabe der Richtlinie im März 2012 wurde bereits in verschiedenen Modellprojekten der Nutzen von Delegation evaluiert. Mit dem Modellprojekt AGnES [vdBMH+09] beispielsweise ließ sich belegen, dass durch Delegation positive Effekte bzgl. Effizienz und Qualität der ärztlichen Leistungen erzielt werden können. Um diese Effizienz- und Qualität zu erreichen, benötigen die Berufsangehörigen der Alten- und Krankenpflege allerdings gewisse Zusatzqualifikationen, die im Rahmen verschiedener Weiterbildungsprogramme erworben werden können. Beispiele hierfür sind VERAH ([fhFiDHIe08] und [fhFiDHIeV08]) oder MFP [QUELLE]. In einem ca. 200stündigen Fortbildungsprogramm werden unter anderem Kenntnisse aus den Bereichen Case Management, Gesundheits-, Praxis- und Wundmanagement sowie IT vermittelt.
Das Gesamtziel des Gesetzgebers ist, eine integrierte Versorgung von Pflegebedürftigen zu erreichen, bei der die Parteien des Gesundheitssystems (Ärzte, Pflegekräfte, Krankenkassen) intensiv zusammenarbeiten sollen. Eine enge Zusammenarbeit bedeutet einen hohen Koordinationsaufwand und damit verbunden ein hohes Maß an Informationsaustausch. Um das volle Potential der Delegation ärztlicher Leistungen ausschöpfen zu können, müssen entsprechende EDV-Systeme unterstützend eingesetzt werden. Zwar werden auch heute schon vielfältige Verwaltungssysteme zur Dokumentation von entweder ärztlichen Tätigkeiten oder Pflegetätigkeiten eingesetzt, jedoch ist das Angebot an Software zur Unterstützung einer Zusammenarbeit zwischen Ärzten und Pflegedienstleistern gering. Gerade wenn es darum geht, vor Ort beim Klienten Vorgänge zu dokumentieren, wird häufig zunächst auf Papier dokumentiert und die Information erst zu einem späteren Zeitpunkt in ein EDV-System übertragen. Diese Medienbrüche Kosten Zeit und in der Folge Geld und die Fehlerquellen bei der manuellen Übertragung von Informationen sind vielfältig.
Diese Master-Arbeit setzt an der Schnittstelle zwischen Ärzten und Berufsangehörigen der Alten- und Pflegeberufe an.
[Beschreibung der einzelnen Kapitel.]

2. Ziele der Arbeit

Ziel dieser Master-Arbeit ist es, ein elektronisches System zur Unterstützung der Zusammenarbeit von Ärzten und Berufsangehörigen der Alten- und Pflegeberufe, die eine Zusatzqualifikation zur Ausübung ärztlicher Tätigkeiten haben, zu entwerfen und zu implementieren. Einerseits soll eine Delegation von Aufgaben möglich sein, andererseits sollen die ausgeführten Tätigkeiten elektronisch dokumentiert werden können. Da die ärztlichen Tätigkeiten vornehmlich vor Ort beim Klienten durchgeführt werden, lassen sich im Kern zwei notwendige Komponenten identifizieren.

Abbildung 2.1 zeigt eine stilistische Darstellung der Komponenten der Arbeit. Die Basis bildet ein zentrales System, in welchem die Daten zu Klienten, auszuführenden und ausgeführten Tätigkeiten verwaltet werden. Daneben gibt es eine Software auf einem mobilen Endgerät, die es ermöglicht, Informationen über auszuführende Tätigkeiten abzurufen und nach der Ausführung die Ergebnisse zu dokumentieren. Verbunden werden beide Systeme über eine Middleware, die Anfragen an das Verwaltungssystem über verschiedene Schnittstellen ermöglichen soll. Da die Kommunikation zwischen der mobilen Anwendung und dem Verwaltungssystem in dieser Arbeit aber lediglich über einen Schnittstellentyp läuft, dient die Middleware vor allem als Platzhalter, um zu einem späteren Zeitpunkt weitere Schnittstellen implementieren zu können. Die Implementierung weiterer Schnittstellen ist nicht Teil dieser Arbeit.
Idee der direkten elektronischen Dokumentation beim Klienten ist die Vermeidung von Medienbrüchen. Anstatt zunächst auf Papier zu dokumentieren und diese Informationen zu einem späteren Zeitpunkt zu übertragen, sollen die Informationen über eine internetgestützte Schnittstelle direkt an das Verwaltungssystem übertragen werden. Ziel ist eine Einsparung der Zeit, die dafür benötigt wird, die schon dokumentierten Informationen noch einmal manuell zu übertragen. Außerdem wird eine Reduzierung möglicher Fehlerquellen angestrebt, um die Qualität der Dokumentation zu erhöhen.
Die Entwicklung der mobilen Anwendung ist Kern der Arbeit. Vorgabe ist, Tablet-PCs zur Dokumentation einzusetzen, da sie den momentan neuesten Stand der Technik darstellen. Welche Plattform (z.B. Android oder iOS) und welche Hardware genau eingesetzt wird,

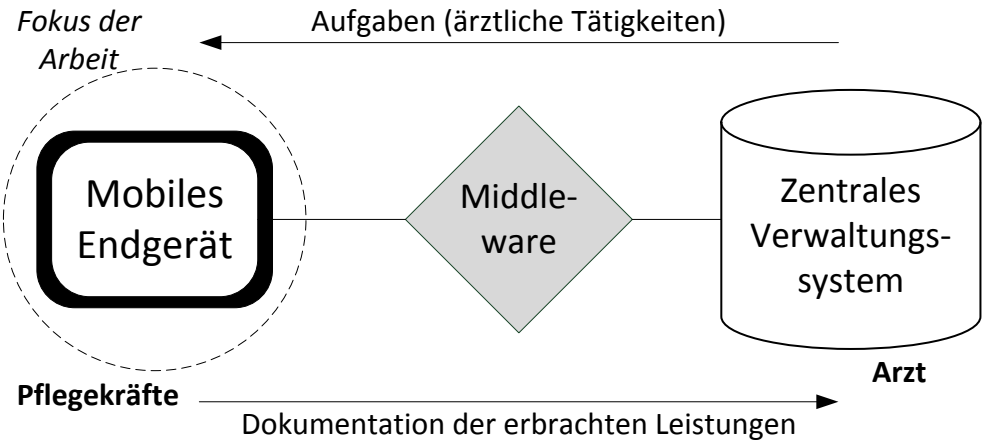

Abbildung 2.1.: Zielkomponenten der Arbeit

ist offen und wird sich aus den zu ermittelnden Anforderungen ableiten.
Der Entwicklungsprozess gliedert sich in eine Anforderungsanalyse, die Implementierung und eine Evaluation der entstandenen Software. Es findet vorerst eine auf Literatur basierende Anforderungsanalyse sowie eine Priorisierung nach technischen und zeitlichen Kriterien statt. Die Implementierung konzentriert sich zunächst auf die Umsetzung der wichtigsten Anforderungen, um einen ausreichend reifen Stand der Software zu erreichen. Die Evaluation gliedert sich drei (ja?) Teile. Nach etwa der halben Bearbeitungszeit wird ein erster Prototyp der Software (auf Basis der in der Literatur gefundenen Anforderungen entwickelt) im Rahmen einer Schulung zur Medizinischen Fachpflegekraft (MFP) den Teilnehmern vorgestellt. Hier sollen Akzeptanz, Benutzbarkeit sowie erwarteter Nutzen der Software evaluiert werden. Die hier gesammelte Resonanz wird in die weitere Entwicklung einfließen.
Neben den fachlichen Anforderungen soll die technische Umsetzung der Software evaluiert werden. Hierzu wird die verwendete Architektur nach Kriterien der ISO-25000 (Software Engineering) [Int05] geprüft.
Im letzten Teil der Evaluation wird überprüft, ob ein typischer Prozess [oder eine Leistung] aus der Richtlinie zur Delegation ärztlicher Leistungen mit der entstandenen Software delegierbar und dokumentierbar ist.

Das zentrale Verwaltungssystem wird in dieser Arbeit nicht von Grund auf entwickelt, sondern basiert auf einer im Projekt VitaBit ([Vit09] und [Kli10]) entwickelten Software-Lösung zur Dokumentation von Pflegeleistungen. Bereitgestellt wird es von der nubedian GmbH [nG13], die außerdem Unterstützung bei der Anpassung des Systems an die Anforderungen der Dokumentation ärztlicher Leistungen anbietet, da das System momentan lediglich zur Dokumentation von Pflegeleistungen gedacht ist. Im Laufe der Entwicklung der mobilen Anwendung sollen notwendige Anpassungen des zentralen Verwaltungssystem identifiziert und dokumentiert werden.

Neben der nubedian GmbH wird der Entwicklungsprozess dieser Arbeit von Mitarbeitern der DRK Residenz Bad Friedrichshall [Res13] unterstützt. Hier werden Pflegekräfte mit einem an VERAH angelehnten Fortbildungsprogramm zu Medizinischen Fachpflegekräften (MFP) ausgebildet. Um die Terminologie in dieser Arbeit zu vereinfachen, wird künftig anstatt von "'Angehörigen der Alten- und Krankenpflege"' oder "'Medizinischen Fachpflegekraft"' stellvertretend für alle Fortbildungsmöglichkeiten zur Erfüllung der Richtlinie nach § 63 Abs. 3c SGB V lediglich die Bezeichnung "'Pflegekraft"' verwendet.

3. Analyse der Anforderungen und daraus abgeleitete Entwurfsentscheidungen

In diesem Kapitel werden die aus Literatur und vorangegangenen Projekten hervorgegangenen Anforderungen an eine mobile Anwendung zur Dokumentation von ärztlich übertragenen Tätigkeiten vorgestellt. In Abschnitt 3.1 werden funktionelle Anforderungen beschrieben und technische Anforderungen daraus abgeleitet. Abschnitt 3.2 stellt Entwurfsentscheidungen vor, die zur Realisierung der Anwendung getroffen wurden.

3.1. Analyse der Anforderungen

Zu Beginn der Bearbeitungszeit der Master-Arbeit konnte kein Termin mit den eigentlichen Endnutzern der mobilen Anwendung vereinbart werden, um eine direkte Anforderungserhebung durchzuführen. Aufgrund dessen musste zunächst eine Anforderungsanalyse auf Basis von Literatur, aktuellen und vorangegangenen Arbeiten ergänzt um Gespräche mit mit sachkundigen Mitarbeitern von Kranken- und Pflegeeinrichtungen durchgeführt werden. Da die Grundlage für die Möglichkeit, ärztliche Tätigkeiten an speziell geschulte Pflegekräfte zu übertragen, die Richtlinie des Gemeinsamen Bundesausschusses basierend auf § 63 Abs. 3c SGB V ist, kann die Dokumentation der hier aufgelisteten übertragbaren Leistungen als Basisanforderung gesehen werden. Ein Problem ist aber, dass aus der Auflistung der Tätigkeiten nicht die zu dokumentierenden Daten hervorgehen. Diese konnten in der Literatur jedoch nicht im Detail ermittelt werden, da einige Leistungen medizinisches Fachwissen über deren Erbringung erfordern. Deswegen wurde zunächst angenommen, dass in der Regel eine Bestätigung, ob eine Leistung erbracht wurde, ausreicht, und sonnst nur die Dokumentation von Blutdruck und Puls als wertbasierte Leistungen notwendig ist, sowie das Fotografieren von Wunden. Weitere zu dokumentierende Werte konnten erst bei einem Treffen mit den Endnutzern ermittelt werden. Die Ergebnisse werden in Kapitel 5.1 vorgestellt.

Das Ausbildungsprogramm zu einer Medizinische Fachpflegekraft (MFP) nimmt keine Einschränkung des Arbeitsgebietes vor. Eine Pflegekraft kann somit sowohl stationär in einer Pflegeeinrichtung arbeiten, als auch im ambulanten Dienst tätig sein, bei der die jeweiligen Klienten besucht werden. Die mobile Anwendung ist folglich so zu gestalten, dass beide Einsatzbereiche abgedeckt werden können. Es ist zu erwarten, dass die zu erbringenden Leistungen im stationären und im ambulanten Bereich die gleichen sein werden, jedoch können sich Unterschiede bei den technischen Anforderungen oder der Terminologie ergeben. Wird im Folgenden keine explizite Unterscheidung zwischen ambulanten und stationären Anforderungen genannt, so gilt eine Anforderung für beide Bereiche.

Jede ärztlich übertragene Tätigkeit kann direkt einer Person zugeordnet werden. Um eine Interaktion mit der Person zu ermöglichen, müssen bestimmte Daten über diese verfügbar sein. Hierzu gehören Stammdaten wie Name, Adresse, Anschrift und Alter, medizinische Daten über aktuell verabreichte Medikamente, Vorerkrankungen oder Allergien sowie Angaben zu Angehörigen, dem zuständigen Hausarzt und einer eventuellen Pflegebedürftigkeit. Aus der Notwendigkeit der Verfügbarkeit persönlicher Daten ergibt sich die Anforderung, diese Daten ausreichend zu schützen und nur dem jeweiligen Pflegepersonal zur Verfügung zu stellen. Folglich muss eine Authentifizierungsfunktion in Form einer Anmeldung mit Benutzername und Passwort (Log-In) implementiert werden. Ein Log-In ermöglicht es außerdem, der angemeldeten Pflegekraft nur die Informationen anzuzeigen, die sie für die Ausführung ihrer Tätigkeit benötigt (Autorisierungsfunktion). So kann ausgeschlossen werden, dass Informationen über Klienten verfügbar sind, die gerade nicht behandelt werden.
Es wird angenommen, dass Pflegekräfte in Schichten arbeiten, da eine Versorgung von Klienten Tag und Nacht gewährleistet sein muss. Weiter wird angenommen, dass jede Pflegekraft während einer Schicht für eine bestimmte Anzahl von Klienten zuständig ist, für die eine Menge vordefinierter Leistungen erbracht werden muss. Zu Beginn jeder Schicht bekommt eine Pflegekraft einen Plan ausgehändigt, in dem die abzuarbeitenden Stationen und die entsprechenden Leistungen aufgelistet sind. In diesem Plan muss jede Leistung dokumentiert werden (vgl. [Har10, S. 18 ff.]). Eine Leistung wird entweder durch das Setzen eines Hakens dokumentiert (Bestätigung) oder durch die Eintragung von Messwerten, beispielsweise im Falle der Erfassung von Blutdruck und Puls. Ziel der mobilen Anwendung ist es, dass sowohl die Übermittlung des Tagesplanes pro Pflegekraft als auch die Dokumentation der Leistungen elektronisch erfolgt. In einer Übersicht über alle Klienten/Stationen sollte direkt erkennbar sein, ob bereits alle Leistungen einer Station erbracht wurden, ohne die Einzelleistungen kontrollieren zu müssen.
Konnte eine Leistung nicht oder nicht vollständig erbracht werden, muss es die Möglichkeit geben, dies zu dokumentieren. Neben einer Markierung der Leistung muss ein Feld für eine Begründung (Freitext oder mit vordefinierten Bausteinen) der Nichterbringung der Leistung verfügbar sein. In der Übersicht über die Stationen einer Schicht sollte neben den Zuständen "'Erledigt"' und "'Offen"' außerdem der Zustand "'Unvollständig abgeschlossen"' ersichtlich sein, um zu zeigen, dass zwar alle Leistungen bearbeitet wurden, jedoch nicht alle erbracht werden konnten.

Bei der manuellen Dokumentation kann es immer wieder zu Fehlern kommen. Beispielsweise wenn Werte falsch eingetragen oder Leistungen bestätigt werden, die eigentlich noch nicht vollständig erfüllt sind. Da diese Fehler auch mit einer elektronischen Dokumentation nicht ausgeschlossen werden können, müssen einmal dokumentierte Leistungen stornierbar und korrigierbar sein, ohne dass im zentralen System eingegriffen werden muss. Um einer Verfälschung von Dokumentationsergebnissen durch nachträgliche Veränderungen beispielsweise von Messwerten entgegen zu wirken, sollte protokolliert werden, wer zu welchem Zeitpunkt eine Änderung vornimmt. Diese Protokollfunktion muss zur vollständigen Überwachung auch im zentralen System verfügbar sein und mit der mobilen Anwendung synchron gehalten werden.
Vor allem im ambulanten Bereich kann es immer wieder vorkommen, dass Klienten in Gegenden besucht werden müssen, in denen kein mobiles Internet verfügbar ist. Da der Datenaustausch zwischen mobiler Anwendung und zentralem Verwaltungssystem über eine Internetverbindung geschieht, wäre somit die Arbeit mit dem mobilen System nicht möglich. Darauf zu setzen, dass beim Klienten vor Ort eine lokale Internetverbindung genutzt werden kann, ist auch keine praktikable Lösung, da die Wahrscheinlichkeit für Probleme mit der Einrichtung und Verfügbarkeit hoch ist. Vor Ort beim Klienten anzukommen und festzustellen, dass der Plan der zu erbringenden Leistungen wegen Verbindungsproblemen nicht verfügbar ist, führt zu nicht hinnehmbaren Einschränkungen, da so eventuell wichtige medizinische Tätigkeiten nicht durchgeführt werden. Notwendig ist deshalb, den Plan für eine oder mehrere Schichten sowie alle relevanten Informationen zu Personen (Klienten, Angehörigen, Ärzten) vor dem Beginn einer Schicht lokal auf dem mobilen Endgerät zu speichern und somit auch ohne Internet verfügbar zu haben. Lediglich Aktualisierungen müssten so während einer Schicht nachgeladen werden. Wird das initiale Laden von Informationen über ein lokales Netzwerk durchgeführt, spart dies außerdem Datenvolumen ein, welches bei Verträgen mit Internet über das Mobilfunknetz meist sehr begrenzt ist (Quelle notwendig?).

Wichtig bei der Dokumentation von Wunden ist neben einer textuellen Beschreibung eine bildliche Erfassung. Dies könnte mit einer mitgeführten Digitalkamera erfolgen, jedoch besteht dann das Problem, dass die aufgenommenen Bilder nach vollendeter Schicht manuell an das zentrale System übertragen werden müssen und den richtigen Patienten zugeordnet. Dies kostet verhältnismäßig viel Zeit und kann zu Fehlern bei der Zuordnung führen. Wünschenswert ist deswegen eine Funktion in der mobilen Anwendung, die es ermöglicht, direkt über eine interne Kamera Wundfotos aufzunehmen und diese an das zentrale System zu übertragen.
Eine ähnliche Anforderung wurde bei der Erfassung der Vitalwerte wie Puls und Blutdruck identifiziert. Neben der manuellen Messung und Eintragung der ermittelten Werte gäbe es die Möglichkeit, ein automatisches Messgerät via Bluetooth an das mobile Endgerät anzuschließen und damit die Messung und Eintragung vollständig zu automatisieren. Dass dies grundsätzlich möglich ist, wurde im Projekt *Stroke Manager* des Forschungszentrum Informatik (FZI) gezeigt [GMR12]. Die automatische Erfassung würde Fehler bei der manuellen Eintragung von Messwerten würden somit ausgeschlossen. Da die Möglichkeit

besteht, dass das automatische Messgerät ausfällt, muss eine manuelle Eintragung jedoch trotzdem möglich sein.
Das Projekt VitaBit befasste sich, ähnlich wie diese Arbeit, mit der Dokumentation von Pflegeleistungen ambulanter Dienste. Auch hier musste einer Pflegekraft ein Tages-/Tourenplan angezeigt werden. Da die Leistungen ambulanter Dienste ausschließlich vor Ort beim Klienten erbracht werden, die häufig allein leben und nur noch eingeschränkt mobil sind, hat die Pflegekraft in der Regel einen Schlüssel zu den Wohnräumen des Klienten. Die Schlüssel aller Klienten werden meist an einem Bund zusammengefasst und mit Nummern zur Zuordnung versehen. Damit die Pflegekraft nicht erst lang nach dem passenden Schlüssel suchen muss, war eine Anforderung, dass die Schlüsselnummer direkt in der Übersicht der Stationen auftaucht. Da die in dieser Arbeit entwickelte Arbeit ebenfalls für den ambulanten Pflegedienst gedacht ist, wird diese Anforderung auch hier übernommen. Im Falle der stationären Betreuung von Klienten soll anstatt einer Schlüsselnummer die Zimmernummer in der Einrichtung angezeigt werden.
Die Zielgruppe, für die die mobile Anwendung entwickelt wird, sind Angehörige der Alten- und Pflegeberufe. Da es sich bei dieser Gruppe in der Regel um wenig IT-affine Personen handelt, muss die Anwendung übersichtlich strukturiert werden und intuitiv bedienbar sein, damit die Hürden der Benutzung durch die Bedienbarkeit möglichst gering sind. Ziel ist es, dass das grundsätzliche Wissen über die Bedienung von Tablet-PCs sowie das Fachwissen über die Dokumentation der Leistungen ausreicht, um die mobile Anwendung zu verwenden. Der Aufwand für Schulungen soll so minimal gehalten werden. Konzeptionell soll eine Orientierung an den graphischen Benutzerschnittstellen von iOS und Android stattfinden, den Betriebssystemen für mobile Geräte von Apple und Google.
Eine Funktionalität, die in erster Linie für ambulante Pflegedienste interessant ist, ist die Möglichkeit die Adresse eines Klienten auf einer Karte anzeigen zu lassen. Alternativ könnte Global Positioning System (GPS) könnte auch die aktuelle Position relativ zur Adresse des Klienten angezeigt werden, um eine Routenplanung durchzuführen und somit Fahrzeiten besser abschätzen zu können. Hierzu ist allerdings eine aktive Internetverbindung notwendig. Eine vollständige Navigation ist nicht geplant.
Um eine Reduzierung der elektronischen Geräte zu erreichen, die eine Pflegekraft mitführen muss, ist eine Funktion zum Führen von Telefongesprächen hilfreich. Einerseits könnten normale Telefongespräche beispielsweise mit der Zentrale, Angehörigen oder Ärzten geführt werden, andererseits wäre es aber auch möglich, einen Notruf-Knopf in die Anwendung zu integrieren, um in Notsituationen direkt Hilfe anfordern zu können. Beschränkt wird die Funktionalität zwar durch die Notwendigkeit der Verfügbarkeit eines Mobilfunknetzes, jedoch kann diese Einschränkung ebenso beim Mitführen eines separaten Mobiltelefons.

Von der bisher beschriebenen Menge an Funktionen ist lediglich die automatische Erfassung von Blutdruck und Puls mittels eines per Bluetooth angeschlossenen Messgerätes durch manuelle Erfassung substituierbar. Alle anderen Anforderungen müssen auf Basis der in der Literatur gewonnen Erkenntnisse umgesetzt werden, um einen sinnvollen Einsatz der mobilen Anwendung überhaupt erst zu ermöglichen.

Aus den beschriebenen funktionalen Anforderungen an die mobile Anwendung ergeben sich eine Reihe von technischen Anforderungen an das Endgerät. Grundanforderung ist eine Internetverbindung sowohl über ein lokales Wireless Local Area Network (WLAN), als auch über ein Mobilfunknetz, um Tourendaten und Aktualisierungen empfangen und senden zu können. Zur Dokumentation von Wunden per Foto muss eine Kamera mit einer ausreichend hohen Auflösung vorhanden sein, damit das medizinische Personal bei der Beurteilung der Wunden nicht durch schlechte Bildqualität eingeschränkt wird. Zur Unterstützung der intuitiven Bedienung sollte das mobile Gerät einen berührungsempfindlichen Bildschirm haben, was aber bei allen Enderäten der Fall sein sollte, da dies eines der Hauptmerkmale von Tablet-PCs ist. Zur Bestimmung der Position einer Pflegekraft, um die Route zum nächsten Klienten zu berechnen, muss eine GPS-Verbindung möglich sein. Sollen Messgeräte für Blutdruck und Puls angeschlossen werden, muss eine Bluetooth-Verbindung verfügbar sein.

Da Alten- und Krankenpflege dem sozialen Bereich zuzuordnen ist und ein Ziel die Einsparung von Kosten ist, sollten die Endgeräte möglichst geringe Kosten verursachen. Obwohl grundsätzlich Tablet-PCs sowohl mit dem Android-Betriebssystem als auch mit iOS die technischen Anforderungen erfüllen, können Android Geräte im Schnitt zu einem deutlich niedrigeren Preis erworben werden [QUELLE]. Aus diesem Grund fällt die Entscheidung bei der Wahl der Plattform auf Android.

3.2. Entwurfsentscheidungen des Gesamtsystems auf Basis der erhobenen Anforderungen

Zur Dokumentation von Pflegetätigkeiten auf Tablet-PC-ähnlichen Endgeräten arbeitete Christina Hardt [Har10] in Ihrer Diplomarbeit mit der im Projekt VitaBIT ([Vit09] und [Kli10]) entwickelten Software. Diese bietet bereits Datenstrukturen für Tourenpläne und die Definition von zu erledigenden Tätigkeiten pro Station. Da die Anforderungen dieser Arbeit auch Touren- bzw. Schichtpläne benötigen, um diese auf einem Tablet-PC abrufen und anzeigen zu können, wird VitaBIT die Basis für das in Kapitel 2 beschriebene zentrale System darstellen. Da es im Kontext dieser Arbeit aber nicht mehr um die Dokumentation von Pflegetätigkeiten geht, sondern um die Übermittlung und Dokumentation ärztlicher Tätigkeiten durch Pflegepersonal, müssen Änderungen an VitaBIT vorgenommen werden. Um den Änderungen und dem Kontext der vernetzten Pflege auch namentlich gerecht zu werden, wurde das zentrale System in CareNet (Care = Pflege, Net = Netz) umbenannt. Auf die Änderungen an VitaBIT wird in dieser Arbeit nicht eingegangen, da diese von Mitarbeitern der nubedian GmbH durchgeführt werden.

Abbildung 3.1 zeigt die Komponenten des Gesamtsystems erweitert um die verwendeten Technologien, auf denen diese basieren. Die Kommunikation zwischen den Komponenten erfolgt via Internet mittels REST. REST steht für *Representational State Transfer* und

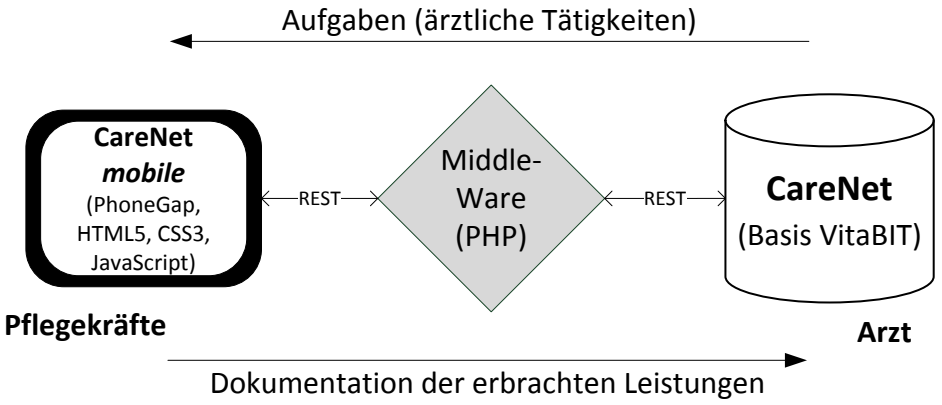

Abbildung 3.1.: Verwendete Technologien in der Gesamtarchitektur

stellt einen Architekturstil zur zustandslosen Interaktion dar. Im Gegensatz zu Sitzungen (Sessions), die von Servern während der Interaktion mit einem Benutzer aufwendig verwaltet werden müssen, werden bei dem REST-Prinzip vorberechnete Ressourcen, beispielsweise eine Webseite, über eine URI (Uniform Resource Identifier) referenziert und geladen. Alle Informationen über den Kontext einer Anfrage sind in der Anfrage selbst enthalten und es werden keine weiteren lokal beim Nutzer oder auf dem Server gespeicherten Informationen benötigt. Eine Kommunikation via REST bedeutet eine lose Verkoppelung von vordefinierten Ressourcen, durch die ein Nutzer anhand gegebener URIs navigieren kann. Es werden lediglich die grundlegenden HTTP-Methoden PUT, POST, GET und DELETE verwendet, um Anfragen zu formulieren. Übertragen werden Ressourcen bzw. Datenobjekte in der Regel in Form einer XML- oder JSON-Datei [zMNS05, S. 19].

Die in dieser Arbeit zu entwickelnde mobile Anwendung stellt im Wesentlichen eine mobile Erweiterung von CareNet dar, aufgrund dessen der Name CareNet*mobile* gewählt wurde. Die Verwendung von Tablet-PCs als Hardware ist vorgegeben, da sie den neuesten Stand der Technik repräsentieren. Es wurde eine Neuentwicklung einer Lösung zur mobilen Dokumentation angestoßen, da in den vorangegangenen Projekten Probleme mit der verwendeten Hardware identifiziert werden konnten. Im Projekt VitaBIT wurden zur Dokumentation Smartphones der ersten Generation eingesetzt und in [Har10] wurde ein Panasonic CF-H1 eingesetzt, ein Vorläufer heutiger Tablet-PCs. Dieses Gerät hatte zwar bereits einen berührungsempfindlichen Bildschirm und eine Kamera, jedoch war die Leistungsfähigkeit nicht ausreichend, was zu Performanzproblemen führte. Außerdem konnte die Intuitivität der Bedienung nicht so umgesetzt werden, wie es beispielsweise unter Android oder iOS heute möglich ist. Das in VitaBIT verwendete Smartphone ist zwar sehr handlich und damit leicht mitzuführen, jedoch schränkt der kleine Bildschirm die Möglichkeiten der Darstellung ein, was gegen die Anforderung der Übersichtlichkeit spricht.

Jedes Betriebssystem von Tablet-PCs basiert auf einer eigenen Programmiersprache. Android basiert beispielsweise auf Java, iOS auf Objective-C und Windows Phone auf C#. Anwendungen für die Betriebssysteme können nur in den jeweiligen Sprachen entwickelt werden. Eine so entwickelte Anwendung wird als *nativ* bezeichnet. Dieser Umstand impliziert, dass bei der Entwicklung neuer Anwendungen vorher die Entscheidung getroffen werden muss, auf welcher Plattform diese bereitgestellt werden soll. Sollen mehrere Plattformen bedient werden, müssen mehrere Anwendungen mit der gleichen Funktionalität, aber anderer Quellcodebasis erstellt werden. Dies führt zu hohem zeitlichen Aufwand und ist kostenintensiv.
Eine Alternative ist die Erstellung von *WebApps*. Prinzipiell handelt es sich hierbei um Webseiten, jedoch sind diese für Touch-Oberflächen optimiert und sie sollen optisch an native Anwendungen erinnern. Da WebApps aber in einem Browser auf dem Tablet-PCs aufgerufen werden müssen, können sie nicht das Gefühl einer vollwertigen App (Application = Anwendung) bieten. Ungleich größer sind die Einschränkungen von WebApps bei der bereitstellbaren Funktionalität. Mit reinen Webtechnologien wie HTML5 oder JavaScript ist es nämlich nicht möglich, hardspezifische Funktionen wie die Kamera, den GPS-Sensor oder eine Bluetooth-Verbindung anzusprechen. Dies ist nur über die jeweilige native Programmiersprache möglich. Da in den Anforderungen aber eine Nutzung der Kamera sowie des GPS-Sensors gefordert wird, ist eine Implementierung von CareNet*mobile* als WebApp nicht möglich.

Ein Ansatz zur Lösung dieses Problems bietet das Projekt PhoneGap [Ado13]. Die Idee ist, in dem Gerüst einer nativen Anwendung, einem nativen Container, einen Browser zu platzieren, der Implementierungen in Webtechnologien wie HTML5, CSS3 und JavaScript interpretieren kann. Abbildung 3.2 zeigt den Zusammenhang der verschiedenen Technologien. Um den Zugriff auf endgerätgspezifische Funktionen zu ermöglichen, bietet PhoneGap zur Kommunikation mit allen wichtigen Hardware-Funktionen JavaScript-Methoden an, die auf die jeweiligen Befehle in der nativen Programmiersprache abgebildet werden. Anstatt beispielsweise die Kamera unter Android direkt über Java anzusprechen, erfolgt der Aufruf in JavaScript, welcher von einer Java-Klasse verarbeitet und an die Hardware weitergereicht wird. PhoneGap verfolgt somit einen hybriden Ansatz aus nativer Entwicklung und der Nutzung von WebApps.
Die reine Nutzung von Webtechnologien und die Bereitstellung verschiedener nativer Container durch PhoneGap für die jeweils gewünschte Plattform ermöglicht eine Portabilität der Anwendung. Einmal entwickelt kann diese beispielsweise sowohl auf Android, als auch auf PhoneGap oder Windows Phone bereitgestellt werden. Diese Möglichkeit bringt Vorteile in Bezug auf Kosten und Zeit gegenüber der Entwicklung individueller nativer Anwendungen.
Als Anforderung an CareNet*mobile* wurde zwar zunächst nur die Bereitstellung auf Android-Endgeräten gefordert, jedoch ist nicht auszuschließen, dass in Zukunft auch die Nutzbarkeit auf iOS- oder Windows-Geräten gefordert wird. Da es mit der Verwendung von PhoneGap möglich ist, diese Option mit geringem Aufwand offen zu halten, wurde entschieden, CareNet*mobile* in als hybride Anwendung zu entwickeln. [Same Origin Policy reinnehmen?]

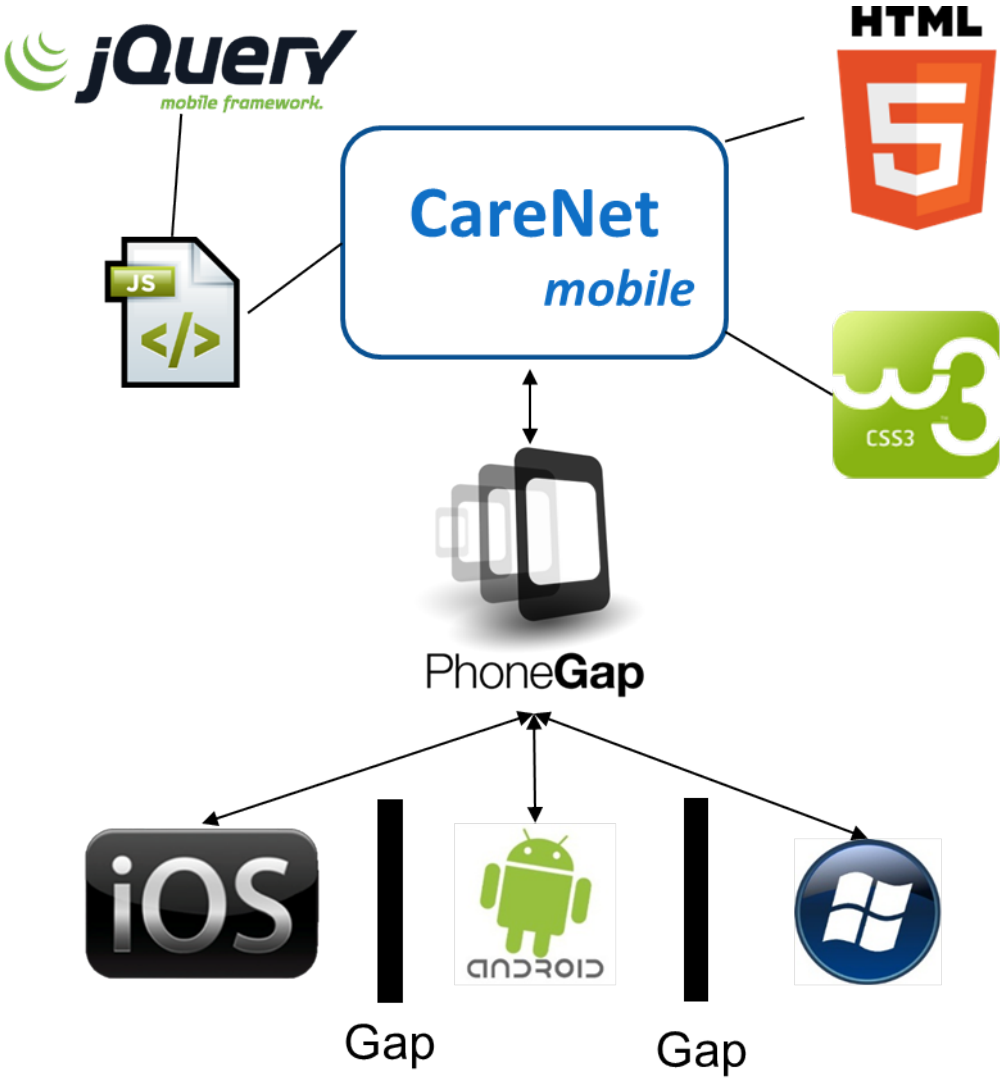

Abbildung 3.2.: Hybride Anwendungsentwicklung mit PhoneGap

Zusammen mit PhoneGap bietet sich die Verwendung bereits existierender JavaScript-Bibliotheken an, die zum Entwurf touch-optimierter Benutzeroberflächen entwickelt wurden. Zwei verbreite Bibliotheken sind jQuery mobile .[jQu13] und Sencha Touch [Sen13]. Im Kontext dieser Arbeit wurde jQuery mobile aufgrund der Einfachheit in der Entwicklung und der Mächtigkeit der gebotenen Funktionalität gewählt.

In [Har10] hat sich gezeigt, dass sich einmal gestellte Anforderungen rasch ändern können und dass eine Anwendung so entworfen sein sollte, dass Anpassungen mit vertretbarem Aufwand durchgeführt werden können. Zwar ist momentan die Menge an zu dokumentierenden Tätigkeiten durch eine abgeschlossene Liste in Richtlinie § 63 Abs. 3c SGB V beschränkt, jedoch sind Änderungen in Zukunft möglich. Neben zusätzlichen Tätigkeiten könnten auch andere funktionale Erweiterungen hinzukommen. Um diesen Anforderungen gerecht zu werden, wird die Anwendung auf Basis einer Plug-In-Architektur entwickelt, angelehnt an die bewährte Architektur der Entwicklungsumgebung Eclipse [The13a]. Bei einer Plug-In-Architektur können Funktionalitäten nach dem Steckerprinzip ("'plug-In"' = einstecken) hinzugefügt und wieder entfernt werden. Im Idealfall sollten Änderungen an den Plug-Ins keine Auswirkungen auf das Gesamtsystem haben. Plug-Ins untereinander können jedoch Abhängigkeiten aufweisen. Wie die Plug-In-Architektur von CareNet*mobile* genau aussehen wird, wird in Kapitel 4 beschrieben.

Wie bereits in Kapitel 2 erwähnt, hat die Middleware in erster Linie eine Platzhalterfunktion, um den Zugang zu CareNet später um weitere Schnittstellen erweitern können. Da im Moment nur eine REST-Schnittstelle angeboten wird, die auch von CareNet*mobile* verwendet wird, wäre die Middleware zur Kommunikation nicht notwendig. Sie nimmt lediglich REST-Aufrufe an und leitet diese unbearbeitet weiter. Es wurde entschieden, die Middleware in PHP [The13b] zu implementieren, da es sich hierbei um eine mächtige und weit verbreite serverseitige Programmiersprache handelt. PHP wird von einem Großteil der Server-Hosting-Anbieter unterstützt und stellt somit keine Limitierung bei der Auswahl des Ortes dar, an dem die Middleware installiert wird.

4. Basisanwendung und Plug-Ins zur Realisierung der identifizierten Anforderungen

[Vorgeplänkel Kapitel]

4.1. Aufgaben und Struktur der Basisanwendung

Um einem Benutzer von CareNet*mobile* die Bedienung der Anwendung zu erleichtern, sollte ein einheitliches Gesamterscheinungsbild eingehalten werden, da ständige Strukturwechsel in der grafischen Oberfläche zu Unübersichtlichkeit führen können [Quelle]. Hierzu wurde die in Abbildung 4.1 gezeigte feste Struktur der grafischen Oberfläche entworfen. Oben links befindet sich das Logo der Anwendung, darunter ist ein Bereich für die Darstellung von Informationen wie das aktuell verwendete Plug-In oder der gerade behandelte Klient reserviert. Am oberen Rand, rechts des Logos, wurde eine Leiste zum Platzieren von Buttons zur Navigation eingefügt. Hier sollen Schaltflächen zur Navigation zwischen Basisanwendung und Plug-Ins platziert werden (z.B. ein "Home"-Button) und auch zur Navigation zwischen verschiedenen Sichten innerhalb eines Plug-Ins. Der größte Bereich ist zur Darstellung und Bearbeitung des eigentlichen Inhalts gedacht.
Die gesamte Anwendung basiert lediglich auf zwei HTML-Dateien und einer Menge von JavaScript-Dateien. Eine HTML-Datei wird für einen Login-Dialog verwendet, die zweite um den kompletten Inhalt darzustellen. Alle Strukturelemente und Funktionalität wird über JavaScript hinzugefügt. Zur Manipulation der HTML-Elemente wird jQuery und jQuery mobile verwendet.

Die Strukturierung und Gestaltung der grafischen Oberfläche erfolgt mittels HTML5 und CSS3, die Funktionalität wird mit JavaScript entwickelt. Gegenüber der Entwicklung einer nativen Anwendung für Android ergeben sich bei der Entwicklung mit JavaScript gewisse Schwierigkeiten. Während Java eine objektorientierte Sprache ist, bei der Typsicherheit und Kapselung von Funktionalität in Klassen zu den Eigenschaften gehören, ist

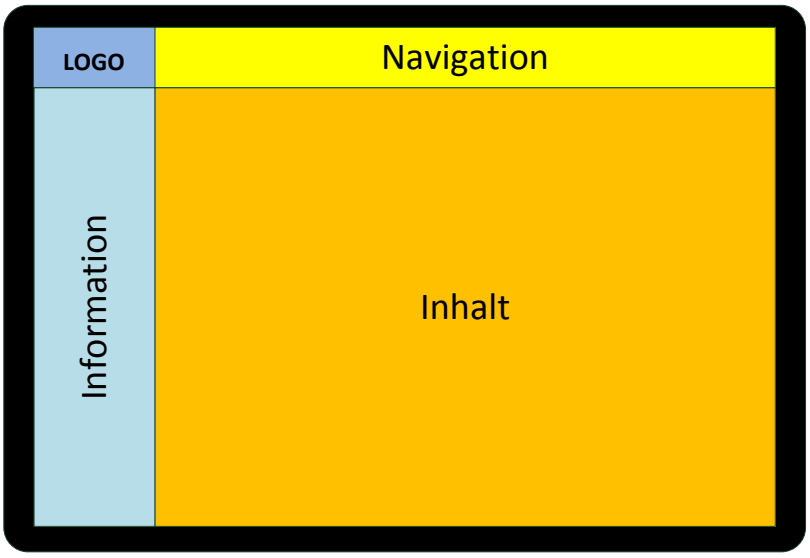

Abbildung 4.1.: Struktur der grafischen Oberfläche von CareNet*mobile*

JavaScript eine Scriptsprache, die keinerlei Typsicherheit und nur bedingte Kapselungsmöglichkeiten bietet. Ein wichtiger Unterschied ist, dass Java vor der Ausführung zunächst kompiliert, d.h. in eine maschinenverständliche Zwischensprache übersetzt werden muss, während JavaScript direkt vom Browser interpretiert werden kann. Dies impliziert, dass JavaScript-Code, selbst wenn er bereits auf einer Webseite ausgeführt wird, dem Nutzer im Klartext zur Verfügung steht, während der Java-Code nur bedingt und durch aufwendige Verfahren wieder dekompiliert und somit im Klartext verfügbar gemacht werden kann. Dieser Eigenschaft muss sich ein Entwickler besonders im Bezug auf im Quellcode vermerkte sicherheitsrelevante Informationen bewusst sein.
Während der Entwicklung kann vor allem die fehlende Möglichkeit der Kapselung problematisch werden. Abbildung 4.2 zeigt Beispielimplementierungen in Java und JavaScript. Links wurde eine Klasse erstellt, die Methoden und Variablen kapselt. Wird eine Variable außerhalb einer Methode definiert (Abbildung 4.2: *ersteVariableJava*), ist diese in der gesamten Klasse verfügbar. Ist die Variable als *public* gekennzeichnet, kann auf sie über *Klassenname.variablenName* zugegriffen werden. Wird sie als *private* deklariert, ist sie nur innerhalb der Klasse verfügbar. Das gleiche gilt für Methoden. Wird eine Variable innerhalb einer Methode definiert (Abbildung 4.2: *zweiteVariable*), kann auch nur innerhalb dieser Methode auf die Variable zugegriffen werden.

In JavaScript sind grundsätzlich alle Methoden global verfügbar. Die Verfügbarkeit beschränkt sich hier nicht auf eine Klasse oder Datei, sondern auf eine Methode kann von allen im Kontext einer HTML-Datei geladenen Dateien zugegriffen werden. Eine Klassendefinition kann nur in sofern erfolgen, dass einem Methodennamen der Name einer anderen vorher definierten Methode vorangestellt wird (Abbildung 4.2: *beispiel.methodeJS()*). Hiermit wird es möglich, einen Namensraum zu definieren, um Methoden einem bestimmten Kontext (hier: *beispiel*) zuzuordnen. Wird allerdings an anderer Stelle im geladenen

Java

```
Datei: beispiel.java

class beispiel{

    public string ersteVariableJava = “”;
            //Zugriff: beispiel.ersteVariableJava

    public void methodeJava(){
            //Zugriff: beispiel.methodeJava();

            string zweiteVariableJava = “”;
            //nur in Methode verfügbar
    }
}
```

JavaScript

```
Datei: beispiel.js

//Konstruktor und „Klasse“
function beispiel(){
            ...
}

var ersteVariableJS = “”;       //global

beispiel.methodenJS = function(){
    //Zugriff: beispiel.methodeJS

    zweiteVariableJS = “”;      //global
    var dritteVariableJS = “”;
            //nur in Methode verfügbar
}
```

Abbildung 4.2.: Vergleich zweier Beispielimplementierungen von Java und JavaScript

JavaScript-Code die selbe Kontext-Methodennamen-Kombination verwendet, kann der Interpreter, der Browser, keine eindeutige Zuordnung des Namens zu ausführbarem Quellcode mehr herstellen und es wird nichts ausgeführt.
Ebenso verhält es sich mit Variablen. Wird eine Variable mit dem Zusatz *var* außerhalb einer Methode definiert, ist sie global verfügbar. Das gleiche gilt, wenn eine Variable ganz ohne Zusatz (Abbildung 4.2: *zweiteVariableJS*) definiert wird. Nur wenn eine Variable innerhalb einer Methode mit dem Zusatz *var* erstellt wird, kann sie nur innerhalb der Methode verwendet werden. Dieser Umstand kann zu massiven Problemen führen. Vergisst ein Entwickler beispielsweise nur ein einziges Mal den Zusatz *var* bei der Definition einer Laufvariable in einer Schleife (häufig mit dem Namen i), wird diese nun global geltende Variable alle anderen Schleifen, die ebenfalls die Laufvariable i verwenden, beeinflussen. Da dies kein syntaktischer, sondern ein semantischer Fehler ist, kann dieser nicht von automatisierten Fehlerfindungswerkzeugen entdeckt werden und es beginnt die sprichwörtliche Suche nach der Nadel im Heuhaufen. Derartige Umstände können sehr viel Zeit kosten.

Einzelne Plug-Ins sollen die in Abbildung 4.1 gezeigte Struktur nutzen, um Inhalte darzustellen. Damit dies möglich ist, müssen die einzelnen Bereiche, welche als HTML-DIV-Elemente erstellt wurden, eindeutig durch IDs gekennzeichnet werden. Da es in JavaScript nicht, wie in Java, möglich ist, Konstanten zu definieren, wurden in einer separaten JavaScript-Datei *CONSTANTS.js* alle IDs in Form von Variablen hinterlegt, welche den DIV-Elementen zugewiesen wurden. Der Nachteil von Variablen ist, dass diese während der Laufzeit aus den Plug-Ins heraus verändert werden können. Aus diesem Grund muss die Konvention eingehalten werden, zentral definierte Variablen nicht zu verändern, da dies Auswirkungen auf andere Plug-Ins haben könnte. Die Eigenschaft einer Plug-In-

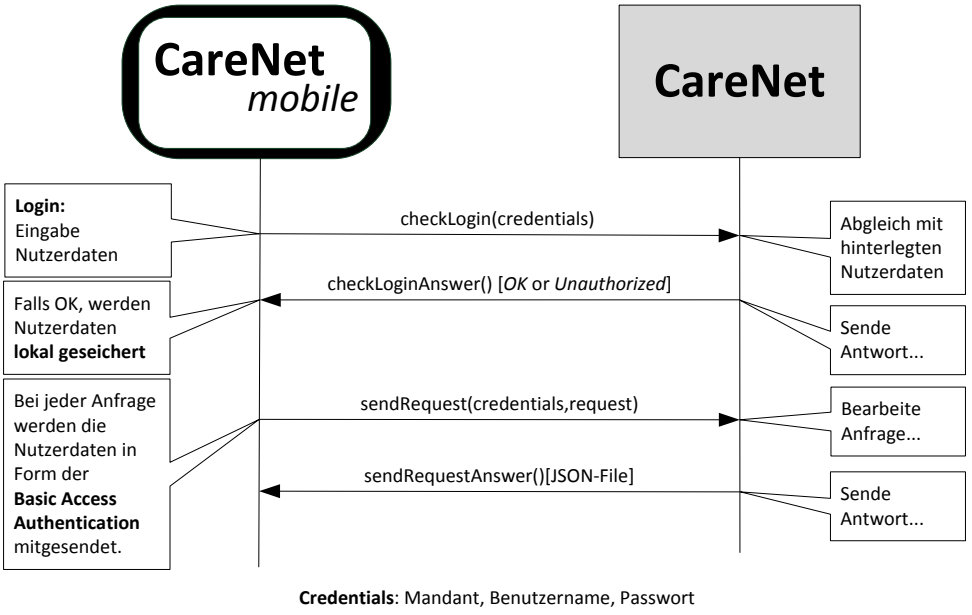

Abbildung 4.3.: Überprüfung der Benutzerdaten in CareNet*mobile*

Architektur, dass Plug-Ins keinen Einfluss auf die Basisanwendung haben, kann somit nicht vollständig in JavaScript umgesetzt werden. Menschliche Fehler durch Nicht-Einhaltung der Konvention bleiben möglich.
Der Vorteil von zentral definierten Variablen ist, dass an einer zentralen, übersichtlichen Stelle nachgeschaut werden kann, wie eine ID lautet, und dass die wichtigen Strukturelemente für Informationen, Navigation und Inhalt zentral referenziert werden können. Die Datei *CONSTANTS.js* wird außerdem dazu genutzt, um alle sonstigen in der Implementierung der Basisanwendung verwendeten Zeichenketten als Variablen zu hinterlegen. Hierzu zählt beispielsweise die Server-Adresse der Middleware oder alle Arten von Meldungen an einen Nutzer. Dieses Verfahren erhöht die Wartbarkeit der Anwendung, da Änderungen an mehrfach verwendeten Zeichenketten nicht *mehrfach* umgesetzt werden müssen, sondern lediglich *einmal* in der zentralen Datei. Die Wahrscheinlichkeit für Fehler durch nicht vollständig umgesetzte Anpassungen kann so verringert werden. Dieses Vorgehen ist angelehnt an die zentrale Verwaltung von *String Ressources* bei der Entwicklung nativer Android-Anwendungen [Goo13].

Eine Grundfunktion, welche die Basisanwendung zur Verfügung stellt, ist ein Log-In zur Sicherung der persönlichen Daten und einer Identifizierung des Nutzers. Da CareNet*mobile* eine mobile Erweiterung von CareNet darstellt, werden keine gesonderten Konten für die Nutzung der mobilen Anwendung angelegt, sondern es werden die schon existierenden Konten in CareNet verwendet. Folglich müssen die bei der Anmeldung eines Nutzer eingegebenen Daten an CareNet gesendet werden, wo sie überprüft werden und das Ergebnis an die mobile Anwendung zurückgesandt wird. Benötigt werden beim Login der Mandant,

der Benutzername und das Passwort.
In Kapitel 3.2 wurde die Kommunikation von CareNet*mobile* mit CareNet via REST erklärt. Aus dieser Kommunikation resultiert eine Besonderheit bei der Authentifizierung eines Nutzers mit einem zentralen System. Abbildung 4.3 zeigt die notwendigen Schritte zur Authentifizierung eines Nutzers und dem anschließenden Senden einer Anfrage. Da keine Sitzung aufgebaut werden kann, in der ein Nutzer dauerhaft angemeldet wird, müssen die kompletten Nutzerdaten bei jeder Anfrage mitgesendet werden.
Meldet sich ein Nutzer in CareNet*mobile* an, so werden zunächst nur die eingegebenen Daten an CareNet gesendet. Als Antwort kommt entweder der HTML-Code 200 (*OK*) oder 401 (*Unauthorized*). Sind die Daten korrekt (Code = 200), werden sie im lokalen Speicher der Anwendung abgelegt, um sie für spätere Anfragen an CareNet nicht noch einmal eingeben zu müssen. Hierzu kann der von PhoneGap angebotene lokale Key-Value-Speicher genutzt werden. In diesem Speicher können Zeichenketten, referenziert über einen Schlüssel, permanent abgelegt werden und auch nach dem Schließen der hybriden mobilen Anwendung wieder abgerufen werden. Die Nutzerdaten werden hier allerdings unverschlüsselt abgelegt. Aus diesem Grund werden diese aus dem lokalen Speicher gelöscht, sobald sich der Nutzer wieder von CareNet*mobile* abmeldet. Weitere Anfragen an CareNet sind so nicht mehr möglich.
Der Login von CareNet*mobile* ist als eigenständige HTML-Datei in Form eines Dialogs realisiert. Wird die Anwendung gestartet, wird zwar zunächst die zentrale Datei *index.html* geladen, jedoch wird vor der Anzeige der Startseite überprüft, ob Nutzerdaten im lokalen Speicher vorhanden sind oder nicht. Sind keine Daten vorhanden, wird zunächst der Login-Dialog aufgerufen und nur wenn die hier eingegebenen Nutzerdaten korrekt sind, erfolgt eine Anzeige der Startseite. Eine reine Überprüfung der Verfügbarkeit von Benutzerdaten im lokalen Speicher reicht als Kriterium zur Anzeige von Inhalten der Startseite aus, da aufgrund der vorherigen Überprüfung ausschließlich validierte Benutzerdaten im Speicher abgelegt werden.
Neben einer Behandlung des Zustandsübergangs beim Starten der Anwendung müssen weitere Zustände im Lebenszyklus einer Android-Anwendung berücksichtigt werden. Wurde eine Anwendung einmal gestartet, kann sie nicht direkt vollständig durch einen Befehl oder die Betätigung des Home-Buttons geschlossen werden, sondern sie wird lediglich *pausiert*. Wird die Anwendung aus dem Task-Manager heraus wieder geöffnet, wird diese nicht neu gestartet, sondern an dem Punkt fortgesetzt, an der sie geschlossen wurde. Dies kann zu einem Sicherheitsproblem werden, wenn ein Nutzer die Anwendung schließt und sich nicht darüber bewusst ist, dass diese eigentlich weiterhin verfügbar ist. Bekommt nun ein Dritter das Tablet in die Hände, kann dieser im Namen der angemeldeten Person Veränderungen vornehmen oder auf sensible Daten zugreifen. Um dies zu verhindern, werden die Nutzerdaten im lokalen Speicher gelöscht, sobald die Anwendung geschlossen wird. Wird diese wieder geöffnet, muss der Nutzer sich neu authentifizieren. Dieses Vorgehen führt zu Einbußen bei der Benutzbarkeit der Anwendung, da immer wieder neu Nutzername und Passwort eingegeben werden muss, auch wenn nur aus Versehen der Home-Button betätigt wurde, jedoch ist dies für den Schutz der sensiblen Daten unabdinglich. Zur Reduzierung der Wahrscheinlichkeit, dass CareNet*mobile* unabsichtlich geschlossen wird, wurde

der Zurück-Button deaktiviert, da dieser bei nur zwei verwendeten HTML-Seiten zu einem Verlassen der Anwendung führt.

In Abbildung 4.3 wird die Rolle der Middleware vernachlässigt, da diese nicht für das grundsätzliche Kommunikationsprinzip relevant ist. Tatsächlich übernimmt die Middleware aber einen Arbeitsschritt bei der Kommunikation. Die Nutzerdaten werden als Zeichenkette *Mandant/Benutzername:Passwort* an die Middleware übertragen. CareNet benötigt die Daten zur Authentifizierung jedoch in Form einer nach der Base Access Authentication verschlüsselten Zeichenkette. Base Access Authentication ist [Erlärung Base64 nach Network security: private communication in a public world, second edition]. Da PHP standardisierte Methoden zur Erstellung dieser Verschlüsselung anbietet, wurde entschieden, die Verschlüsselung in die Middleware zu verlagern.
Auch wenn es sich bei der Base Access Authentication namentlich um eine Verschlüsselung handelt, ist diese jedoch kein Schutz gegen einen Zugriff auf die Daten Dritter, da das Verfahren öffentlich bekannt ist und leicht entschlüsselt werden kann. Im Prinzip werden die Benutzerdaten also im Klartext von CareNet*mobile* zu CareNet übertragen. Hört ein Dritter die Kommunikation ab, kann er die Nutzerdaten auslesen und zweckentfremden. Aus diesem Grund muss die Verbindung an sich verschlüsselt werden. Hierzu wird eine HTTPS-Verbindung sowohl von CareNet*mobile* zur Middleware als auch von der Middleware zu CareNet verwendet.
HTTPS steht für *Hypertext Transfer Protocol Secure* und stellt eine Kombination aus einem symmetrischen und einem asymmetrischen Verschlüsselungsverfahren dar, um Internetverbindungen abhörsicher zu machen. Bei einem symmetrischen Verfahren benutzen Sender und Empfänger beide den gleichen Schlüssel, um Nachrichten zu ver- und entschlüsseln. Bei einem asymmetrischen Verfahren wird eine Nachricht mit einem öffentlichen Schlüssel verschlüsselt und mit einem privaten Schlüssel, den nur der Empfänger kennt, wieder entschlüsselt. Die symmetrische Verschlüsselung ist zwar deutlich schneller, jedoch auch unsicherer, da der Schlüssel zunächst über eine unverschlüsselte Verbindung übertragen werden muss und so eventuell abgehört werden könnte. Beim HTTPS-Verfahren wird zum Aufbau einer sicheren Verbindung der öffentliche Schlüssel an den Empfänger übertragen. Dieser nutzt den Schlüssel, um eine Nachricht mit einem symmetrischen Schlüssel an den Server zu schicken. Diesen symmetrischen Schlüssel können in der folgenden Kommunikation beide Seiten verwenden. Eine langsame asymmetrische Kommunikation findet also nur beim Aufbau der Verbindung statt [Jan07, S. 152].

4.2. Einbinden von Plug-Ins in die Basisanwendung

Wie in Kapitel 3.2 beschrieben, wird CareNet*mobile* auf Basis einer Plug-In-Architektur entwickelt. Die eigentliche Funktionalität, um die Anwendung produktiv einsetzen zu können, soll ausschließlich über Plug-Ins hinzugefügt werden. Diese sind prinzipiell gekapselte Programmstücke, die bestimmte Funktionen bereitstellen. Plug-Ins sollen der Basisanwendung leicht hinzugefügt und wieder entfernt werden können.
Plug-Ins werden in CareNet*mobile* in Form eines Ordners zum Zeitpunkt der Entwicklung

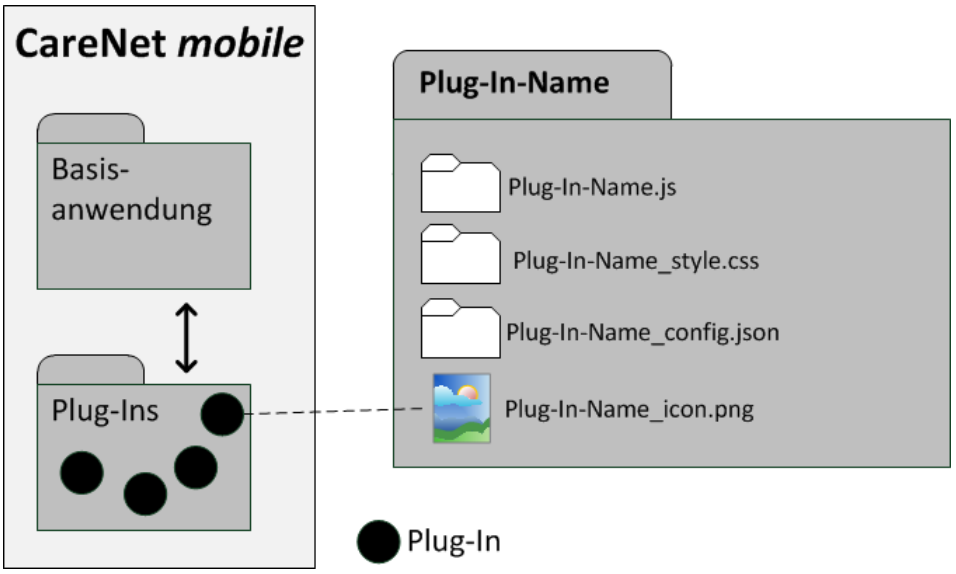

Abbildung 4.4.: Struktur eines Plug-Ins in CareNet*mobile*

eingebunden, bevor die Anwendung zur Installation auf dem Tablet-PC generiert wird. Ein Einbinden von Plug-Ins zur Laufzeit ist somit nicht möglich. Um eine unbegrenzte Anzahl Plug-Ins über ein generisches Verfahren einbinden zu können, muss eine einheitliche Struktur gegeben sein. Abbildung 4.4 zeigt die Struktur eines Plug-Ins. Es besteht aus einer Menge von Dateien, die in einem Ordner zusammengefasst werden. Der Ordner muss den Namen des Plug-Ins tragen und die darin enthaltenen Dateien müssen der Namenskonventionen folgen, dass sie jeweils den Namen des Plug-Ins enthalten und ein beschreibendes Stichwort für den Inhalt der Datei. Zwar könnte auch schon die Dateiendung als Indikator für den Inhalt dienen, jedoch wird es durch den beschreibenden Zusatz noch eindeutiger.

Jedes Plug-In muss die in Abbildung 4.1 gezeigte Struktur der Anwendung nutzen und die einzelnen Div-Elemente über die in Variablen abgelegten IDs ansteuern. Um nicht bei jeder Veränderung eine neue HTML-Datei laden zu müssen, wird der Inhalt der Div-Elemente dynamisch angepasst. Hierzu ist JavaScript notwendig, folglich muss ein Plug-In eine JavaScript-Datei enthalten (*Plug-In-Name.js*), die sowohl die Methoden zum Aufbau der grafischen Oberfläche als auch die Anwendungslogik enthält. Wichtig für die grafische Darstellung von HTML-Elementen sind Cascading Style Sheets (CSS). Sie werden dazu verwendet, wiederverwendbare Formatierungen zu hinterlegen, die den HTML-Elementen zugewiesen werden können. Diese müssen, wie JavaScript auch, in einer separaten Datei zusammengefasst werden (*Plug-In-Name_style.css*).

Die Basisanwendung soll keinerlei Abhängigkeiten zu den Plug-Ins haben, jedoch verwenden Plug-Ins von der Basisanwendung bereitgestellte Methoden. Außerdem können Abhängigkeiten der Plug-Ins untereinander bestehen. Sind notwendige Methoden nicht verfügbar oder wurden diese verändert (beispielsweise die Methodensignatur), kann dies dazu führen, dass ein Plug-In nicht mehr funktioniert. Um die fehlenden Abhängigkeiten nicht erst zu entdecken, während der Nutzer eine nicht verfügbare Funktion versucht zu verwenden, müssen Abhängigkeiten in einer Konfigurationsdatei beschrieben werden.

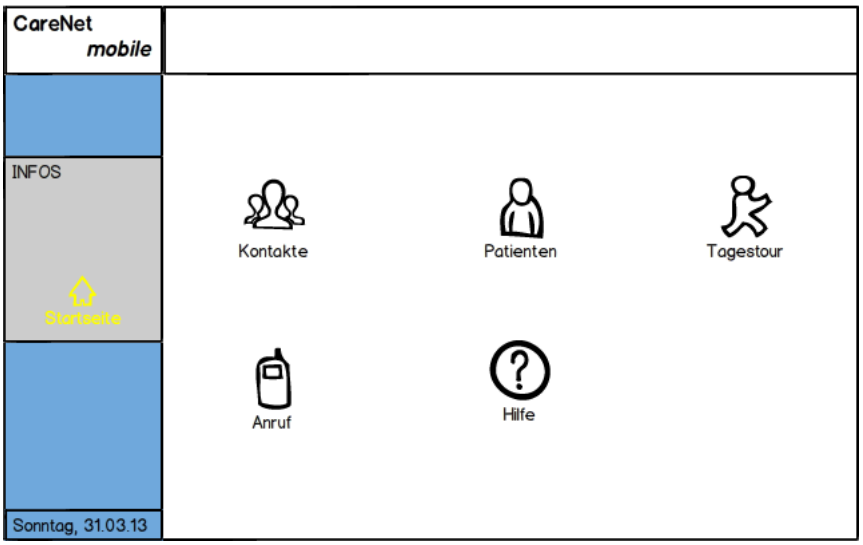

Abbildung 4.5.: Übersicht der verwendbaren der Plug-Ins

Die Datei *Plug-In-Name_config.json* enthält eine Liste aller Plug-Ins sowie deren Versionsnummer, die ein Plug-In benötigt, um grundsätzlich funktionieren zu können. Außerdem enthält die Datei die Versionsnummer des Plug-Ins selbst, um eventuelle Änderungen anzuzeigen. Werden nun Plug-Ins beim Öffnen der Anwendung geladen, kann zunächst überprüft werden, ob alle notwendigen Abhängigkeiten aufgelöst werden können. Ist dies nicht der Fall, wird ein Plug-In nicht geladen und der Nutzer bekommt einen Hinweis angezeigt. Neben den Abhängigkeiten zu anderen Plug-Ins können außerdem Abhängigkeiten zu JavaScript-Bibliotheken wie jQuery mobile oder PhoneGap bestehen. Um hier Redundanzen zu vermeiden, die entstehen, wenn jedes Plug-In diese Bibliotheken selbst einbindet, können auch diese in der Konfigurationsdatei angegeben werden. So wird der benötigte Speicherplatz der Anwendung möglichst gering gehalten.

Die Basisanwendung bietet außer dem Login ausschließlich Funktionalitäten an, die der Nutzer nicht direkt verwenden kann, sondern die nur programmatisch von Plug-Ins genutzt werden können. Da die gesamte produktiv nutzbare Funktionalität folglich über die Plug-Ins hinzugefügt wird, dient die Startseite von CareNet*mobile* als eine Art Sprungbrett, um die hinterlegten Anwendungen zu starten. [Nei12, S. 3 ff.] beschreibt das sogenannte *Springboard*-Muster (Sprungbrett-Muster) als häufig verwendetes und etabliertes Entwurfsmuster für die primäre Navigation in mobilen Anwendungen. Meist wird ein Gittermuster verwendet, in welchem zu den jeweiligen Anwendungen passende Abbildungen und ein textueller Bezeichner angeordnet werden. Bei dem Entwurf der grafischen Oberfläche von CareNet*mobile* wurde dieses Muster gewählt, da es eine hohe Übersichtlichkeit auf der Startseite bietet. Abbildung 4.5 zeigt die Umsetzung in CareNet*mobile* für fünf eingebundene Plug-Ins. Die Abstände zwischen den Symbolen wurden bewusst groß gewählt, um eine Fehlauswahl durch versehentliches Tippen auf falsche Symbole zu vermeiden. Alle Symbole werden in gleicher Größe dargestellt, um dem Nutzer zu vermitteln, dass es sich um gleichwertig gewichtete Anwendungen handelt. Es werden jeweils drei Symbole pro Reihe angezeigt. Bei mehr als drei Plug-Ins werden zusätzliche Reihen nach unten hinzuge-

Datei *beispiel.js*

```
//Konstruktor
function beispiel(){
        //initialisiere Datenbank
        …
}

beispiel.init = function(){
        /*CONTENT_DIV_ID = Variable, die Id des Div-
          *         Elements enthält für Inhalt enthält
          *Methode „append“ fügt gegebenes HTML dem
          *Eltern-Element hinzu
          */
        $('#'+CONTENT_DIV_ID).append('<h1>Hallo</h1>');
}

//Weitere Methoden mit beispiel.methodenname…
```

Abbildung 4.6.: Grundstruktur einer Plug-In-JavaScript-Datei

fügt. Müssen mehr Plug-Ins angezeigt werden, als Platz auf dem Bildschirm zu Verfügung steht, kann der Bereich für den Inhalt nach unten gescrollt werden. Jedes Plug-In muss ein eigenes Icon mit dem Namen *Plug-In-Name_icon.png* bereitstellen.
Zur Zeit der Abgabe dieser Arbeit war es leider noch nicht möglich, JavaScript- und CSS-Dateien zur Laufzeit einzubinden. Es muss jede Datei im Plug-In-Ordner manuell in der zentralen HTML-Datei eingebunden werden. Dies verursacht zusätzlichen Aufwand, jedoch konnte keine andere Lösung gefunden werden. Außerdem muss der Name jedes Plug-Ins in einer Konfigurationsdatei der Basisanwendung hinterlegt werden, denn es werden nur die Plug-Ins eingebunden, die hier aufgelistet wurden. Ein Nachteil der manuellen Einbindung *einer* JavaScript-Datei ist, dass nicht einfach eine Menge von Dateien hinterlegt werden kann, die automatisch verwendet werden. Wäre dies möglich, könnte die Anwendungslogik eines Plug-Ins auf mehrere JavaScript-Dateien aufgeteilt werden. Wird nur eine Datei eingebunden, muss jeglicher JavaScript-Code in dieser Datei zusammengefasst werden, was die Übersichtlichkeit während der Entwicklung hindert, da diese Datei mehrere tausend Zeilen Code enthalten kann.

Die Benutzung von CareNet*mobile* soll auch ohne eine Internetverbindung möglich sein, wenn vorher alle relevanten Daten auf den Tablet-PC geladen wurden. Hierzu wird eine lokale Datenbank von der Basisanwendung bereitgestellt. Die Einzelnen Plug-Ins können über vordefinierte Methoden eigene Tabellen anlegen, um Plug-In spezifische Informationen abzuspeichern. Damit es beim ersten Öffnen eines Plug-Ins zu keiner größeren Verzögerung wegen des Anlegens benötigter Tabellen kommt, ist eine Funktion wünschenswert, mit der Plug-Ins Vorgänge anstoßen können, die schon vor dem ersten Öffnen ausgeführt

werden. Um dies zu ermöglichen, muss die JavaScript-Datei die in Abbildung 4.6 gezeigten Methoden implementieren.
Benötigt wird zunächst ein Konstruktor, der nach dem Plug-In benannt wurde. Dieser wird aufgerufen, sobalb ein Plug-In zur Anzeige auf der Übersicht geladen und festgestellt wurde, dass all Abhängigkeiten erfüllt sind. Hier können beispielsweise Tabellen einer Datenbank initialisiert werden. Neben einem Konstruktor wird eine Methode zur Initialisierung der Anwendung benötigt. Diese wird aufgerufen, sobald ein Nutzer ein Plug-In öffnet. In dieser Methode muss die Anzeige der App in den Bereichen Navigation, Information und Inhalt angepasst werden. Beispielsweise könnte ein Button in die Navigationsleiste gesetzt werden oder, wie in Abbildung 4.6 gezeigt, dem Inhalts-Div der Schriftzug "'Hallo"' in dem HTML-Größenelement "'<h1>"' hinzugefügt werden. Würde die Plug-In-Struktur in Java implementiert, könnte die Methode *init()* in ein Interface gefasst werden, welches jedes Plug-in implementieren muss. Auch ein Konstruktor wäre gegeben. Dies wäre eine bessere Variante, da so sichergestellt werden kann, dass ein Programmierer die Methoden implementiert (alle Methoden eines verwendeten Interfaces müssen in Java implementiert werden). In JavaScript kann die Anforderung nur über Konventionen eingefordert werden, was zu menschlichen Fehlern durch Vergessen der Methoden führen kann.
Ähnlich wie bei der Namensgebung der Dateien müssen auch innerhalb der Dateien Konventionen eingehalten werden. Um zu verhindern, dass Methoden mit gleichen Namen oder CSS-Klassen mit gleichen Namen in verschiedenen Plug-Ins implementiert werden, die sich dann gegenseitig stören, muss jede Methode mit dem Namen des Plug-Ins beginnen (*Plug-In-Name.methode*), ebenso wie jede CSS-Klasse, jedoch hier ohne Punkt (*Plug-In-NameKlassenname*). Auch jede global verwendete Variable muss den Plug-In-Namen vorangestellt bekommen, um Konflikte zu vermeiden. Eine Nicht-Einhaltung der Konventionen kann zu einer völligen Funktionsstörung der Anwendung führen.

4.3. Das Plug-In *Kontakte*

Das Plug-In *Kontakte* wurde entworfen, um gezielt auf in CareNet hinterlegte Informationen über Personen und Organisationen zugreifen zu können, ohne dass diese einen Bezug zu einem aktuell im Tagesplan auszuführenden Auftrag haben. Beispielsweise soll die Telefonnummer eines Arztes abgerufen werden können, um Rückfragen zu einem Termin stellen zu können. Ein anderer Anwendungsfall wäre die Suche nach Angehörigen eines Klienten. Vorwiegend soll mit dem Plug-In auf Stammdaten wie Name, Anschrift, Alter und Kontaktmöglichkeiten zugegriffen werden können, jedoch ist im Falle von Klienten auch eine Erweiterung um Angaben zu Krankheitsverläufen, Vitaldatenstatistiken oder aktuelle Medikation denkbar. Im Rahmen dieser Arbeit wird die Anzeige jedoch auf Stammdaten reduziert.
In CareNet wird unterschieden, ob es sich bei einem Kontakt um eine Organisation oder eine Person handelt. Im Falle von Personen wird jeweils eine bestimmte Adressart wie Klient, Angehöriger oder Ehepartner (nicht abschließend) unterschieden. Bei Organisationen können einige Einrichtungen des Gesundsheitssystems angegeben werden, z.B. ein Krankenhaus, ein ambulanter Pflegedienst oder auch ein Hausarzt. Angezeigt werden die

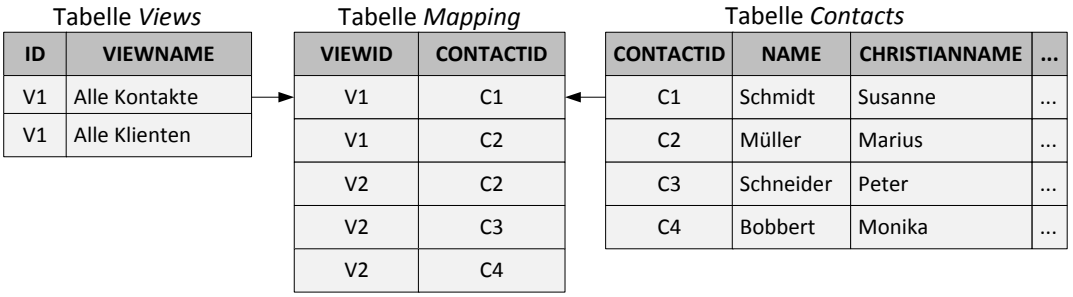

Abbildung 4.7.: Tabellen in der lokalen Datenbank des Plug-Ins *Kontakte*

Kontakte über sogenannte *Sichten* (Views). Eine Sicht stellt einen Filter dar, der jeweils nur die Kontakte einer bestimmten Gruppe anzeigt. Diese sind beispielsweise *Alle Kontakte*, *Alle Angehörigen*, *Alle Klienten* oder auch *Alle Organisationen*. Diese Filterung nach Sichten sollte auch in CareNet*mobile* möglich sein. Bei dem Verhältnis von Kontakten zu Sichten handelt es sich um eine n:n-Beziehung. Das bedeutet, dass ein Kontakt in mehreren Sichten enthalten sein kann und mehrere Kontakte einer Sicht zugeordnet werden. Beispielsweise wird ein Klient sowohl unter *Alle Kontakte* als auch unter *Alle Klienten* angezeigt werden. Außerdem wird es in der Regel mehr als einen Klienten geben.

Da auch die Kontaktinformationen offline verfügbar sein sollen, müssen entsprechende Tabellen zur lokalen Speicherung vorhanden sein. Als Speicher wird eine von HTML5 angebotene Sqlite-Datenbank verwendet. Diese speichert Daten persistent, d.h. auch wenn die Anwendung auf dem Tablet-PC geschlossen wird oder der Tablet-PC heruntergefahren wird, werden die Daten nicht gelöscht. Beim Start von CareNet*mobile* wird eine Datenbank angelegt bzw. überprüft, ob bereits eine Datenbank mit entsprechendem Namen vorhanden ist. Da die Datenbank innerhalb des von PhoneGap angebotenen Browsers/Web Views angelegt wird, kann von außen bzw. anderen auf dem Tablet-PC installierten Anwendungen nicht darauf zugegriffen werden. So entsteht kein Sicherheitsproblem durch die Speicherung sensibler Daten auf dem Tablet-PC. Alle Daten werden in Tabellen gespeichert, die in unbegrenzter Zahl erstellt werden können. Abbildung 4.7 zeigt die notwendige Tabellenstruktur, um Sichten und Kontakte lokal abspeichern zu können. Die linke Tabelle enthält die IDs und Namen aller Sichten, die aus CareNet geladen wurden. In der rechten Tabelle sind alle Kontakte mit IDs und einer Reihe von Attributen enthalten. Die mittlere Tabelle verbindet beide Tabellen, indem sie durch Gegenüberstellung der IDs der Sichten und der Kontakte eine Abbildung schafft, welcher Kontakt in welcher Sicht enthalten ist. Beispielsweise ist in Abbildung 4.7 der Kontakt mit der ID C2 in den Sichten mit den IDs V1 und V2 enthalten, während die anderen Kontakte jeweils nur in einer Sicht enthalten sind. Wählt ein Nutzer eine Sicht aus, wird in der Tabelle *Mapping* nach allen Kontakt-IDs gesucht, die der ID der Sicht zugeordnet sind. Mit Hilfe der gesammelten IDs werden die eigentlichen Kontaktinformationen aus der Tabelle *Contacts* abgefragt, welche nun dargestellt werden können.

Die Darstellung soll sowohl die Sichten als auch die darin enthaltenen Kontakte auf einen

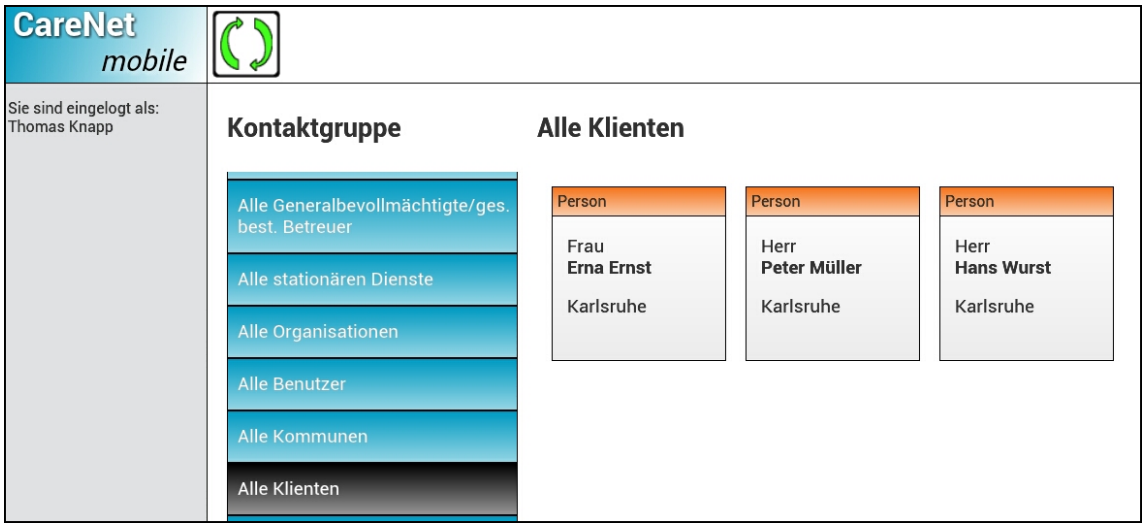

Abbildung 4.8.: Ausschnitt der Darstellung von Kontakten innerhalb einer Sicht

Blick enthalten. Für die Auflistung der verfügbaren Sichten wurde eine einzeilige Tabelle ohne Kopfzeile gewählt. Die zugehörigen Kontakte einer Sicht werden als Kacheln bzw. als Galerie ([Nei12, S. 17]) dargestellt. Abbildung 4.8 zeigt einen Ausschnitt aus der Darstellung der Kontakte in CareNet*mobile* . In dem orangefarbenen Balken ist vermerkt, ob es sich um eine Person oder eine Organisation handelt, im grauen Kasten darunter werden Vorname und Name sowie Wohnort aufgeführt. Es könnten beispielsweise auch die Telefonnummer, die E-Mail-Adresse oder ein Foto einer Person angezeigt werden, jedoch wurden aus Gründen er Übersichtlichkeit diese Informationen gewählt.
Jedes Element in der Tabelle der Sichten weist nach oben und unten einen gewissen Abstand vom Text zum Rand auf, um genug Fläche für eine Auswahl mit dem Finger zu bieten und somit die Wahrscheinlichkeit neben die gewollte Schaltfläche zu tippen zu verringern. Das gleiche gilt für die Kacheln zur Auflistung der Kontakte. Neben der Informationsfunktion soll auch hier eine komfortable Bedienung ermöglicht werden.
Wählt ein Nutzer einen Kontakt aus, werden alle relevanten Informationen in einem separaten Fenster dargestellt. Momentan findet lediglich eine Unterscheidung bei der Darstellung von Personen und Organisationen statt, jedoch ist eine Differenzierung nach Adressarten geplant.
Zum Zeitpunkt der Fertigstellung dieser Arbeit hatte das Plug-In nur eine Funktion zum Anzeigen von Kontakten. Eine Editierfunktion bestehender Kontakte oder die Neuanlage von Kontakten ist nicht möglich.

Die in Abbildung 4.7 gezeigten Tabellen werden im Konstruktor des Plug-Ins erzeugt bzw. wird überprüft, ob sie vorhanden sind. Wird das Plug-In geöffnet und es sind keine Kontaktdaten vorhanden, wird angenommen, dass das Plug-In zum ersten Mal geöffnet wird und es wird automatisch ein Versuch unternommen, Kontaktinformationen von CareNet zu laden. Sind bereits Daten vorhanden, werde diese lediglich angezeigt. Meint der Nutzer, die angezeigten Daten wären nicht aktuell, kann er über einen Button in der Navigationsleiste einen Aktualisierungsvorgang anstoßen. Ist dieser abgeschlossen, werden die aktualisierten

Daten angezeigt.
Da Sichten, Kontakte und das Mapping in verschiedenen Tabellen abgespeichert werden, muss sichergestellt werden, dass alle Daten vollständig geladen wurden, bevor dem Nutzer die Möglichkeit gegeben wird, Sichten auszuwählen. Hierzu wurde ein [ENTWURFSMUSTER zu Callback vorstellen, Umsetzung erklären]

Für alle Plug-Ins feste Beschreibungsstruktur:

- **Begründung für Plug-In (aus welchen Anforderungen geht Plug-In hervor)**
  - Gezielt Infos über Klienten, Ärzte, Angehörige herausfinden [CHECK]
  - Vorwiegend Anzeige von Stammdaten [CHECK]
  - Unterscheidung der Anzeige nach Kontaktart (Klient, Angehöriger, Arzt, Organisation) [CHECK]
  - Aktualisieren von Kontakten [CHECK]
- **Aufbau/Navigationsstruktur**
  - in CareNet: verschiedene Sichten auf Kontakte (soll auch im Mobilteil verfügbar sein) [CHECK]
  - Views und Kontakte über Zeiger miteinander verbunden [CHECK]
  - n:n-Beziehung [CHECK]
  - drei Tabellen: eine mit Views, eine mit Kontaktinfos, eine für Mapping [CHECK]
  - Erstellen der Tabellen bei Initialisierung [CHECK]
  - Füllen der Tabellen beim Öffnen des Plug-Ins, falls keine Daten vorhanden sind [CHECK]
  - Ansonsten Anzeige der vorhandenen Daten, manuelle Aktualisierung notwendig [CHECK]
  - Wie wird Laden der Infos vollzogen, sodass es zu keinen Brüchen kommt (View versucht Kontakte anzuzeigen, die noch nicht geladen wurden)

4.4. Das Plug-In "'Touren"'

- **Begründung für Plug-In (aus welchen Anforderungen geht Plug-In hervor)**
  - Ziel: Anzeige und Dokumentation von Touren
  - Tour für einen bestimmten Tag
  - Auflistung aller Stationen/Klienten eines Tages
  - Geben übersichtlicher Kunzinfos

  - Zeigen einer Schlüsselnummer/Zimmernummer
  - Liste von zu erbringenden Services
  - Abhaken von Tätigkeiten
  - Dokumentation von Blutdruck und Puls [IMPL]
  - Wunddokumentation mit Bildern (interne Kamera) [IMPL]
  - Aktualisieren von Tourdaten [BUG]
  - Senden der dokumentierten Ergebnisse an CareNet [IMPL]
  - Patientenakte + Statistiken [IMPL]

- **Aufbau/Navigationsstruktur**
  - Startseite: Aufgaben des Tages
  - Services nur mit Stichwort und Status zum Aufklappen
  - nur von Übersicht Tagestour kann auf Startseite der App navigiert werden
  - sonst immer noch zurückkehren zum letzten Schritt
  - Atomare Transaktionen: Rückabwicklung, wenn während des Datenabrufs ein Fehler auftritt

- **Zentrale Frage, die beantwortet werden muss: Welche Funktionalität geht aus welcher Anforderung hervor?**
  -

5. Evaluation

5.1. Evaluation von Nutzbarkeit, Funktionsumfang und Akzeptanz anhand einer Zielgruppenschulung

Evaluation der antizipierten Anforderungen aufgrund von Literaturrecherche und logischem Denken. Wichtiger Teil: Ergebnisse der Schulung der Alten- und Pflegekräfte vom 19.01.12 . Wie fanden die Teilnehmer die aufgrund der antizipierten Anforderungen entworfene App? Welche Änderungen wurden vorgeschlagen? Welche Erweiterungen sind notwendig? ⟨ Nutzbarkeit, Funktionsumfang, Akzeptanz ⟩

5.1.1. Vorwissen der Teilnehmer und vorbereitende Maßnahmen

- Eindruck, Vorwissen (inkl. CAS-Bogen)
- CareCM-Schulung (nur erwähnen, kein Mehrwert)
- Grundlagenvermittlung Tablet-PCs

5.1.2. Ermittlung des Status Quo - Ablauf und Schlussfolgerungen

[Orientierung an Fazit]

5.1.3. Anforderungen an CareNet*mobile* aus Sicht der Zielnutzergruppe

5.2. Evaluation der technischen Umsetzung der mobilen Anwendung

Evaluation der Anforderungen dahingehend, welche umgesetzt werden konnten und in welcher Qualität (inkl. der, die noch in der Schulung dazukamen). Vergleich mit Software-Entwicklungs-Standards (ISO xxx), Architekturprinzipien. Evaluation in zwei Teilen:

5.3. Fallstudie Pneunomie

Überprüfung der Anwendbarkeit der mobilen Anwendung auf einen Ablauf in der Praxis.

6. Fazit und zukünftige Entwicklungs- und Einsatzmöglichkeiten

Ausblick auf mögliche Erweiterungen:

- Erweiterbarkeit des Ansatzes ausgrund von Plug-In-Struktur herausstellen
- Weitere Funktionen, um Komfort zu steigern (Prio C Anforderungen)
  - Schließen eines Falles beim Entfernen vom Einsatzort (damit keine überlangen Verweildauern gespeichert werden)
  - Login zu verschiedenen Instanzen von CareNet (Auswahl beim Login)
  - Unterschiedliche Verfügbarkeit von Plug-ins in Abhängigkeit des Rechten desjenigen, der sich anmeldet
- Erweiterung um Sprachpakete durch zentrale Definition von Nachrichten

1. Kurzer Vergleich der nativen Entwicklung unter Android mit der hybriden Entwicklung (Wie hat sich die Entwicklung gestaltet?)
   a) Typsicherheit
   b) Namensräume (z.B. Gefahr von Überschneidung von Variablennamen und Methodennamen)
   c) Unterstützung durch native Java-Bibliotheken vs. JavaScript-Bibliotheken
   d) Einfachheit der Implementierung mit HTML/JavaScript
   e) Vorteil der Potierbarkeit (Test wäre hier praktisch)

7. Erklärung

Ich versichere hiermit wahrheitsgemäß, die Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt, die wörtlich oder inhaltlich übernommenen Stellen als solche kenntlich gemacht und die Satzung des Karlsruher Instituts für Technologie (KIT) zur Sicherung guter wissenschaftlicher Praxis in der jeweils gültigen Fassung beachtet zu haben.

Karlsruhe, den 31. 03. 2012 _____

Thomas Knapp

Anhang

A. Kernanforderungen an eine mobile Anwendung zur Dokumentation ärztlich übertragener Tätigkeiten

- Wie wurden die Anforderungen erhoben?
  - Richtlinie
  - Gespräche mit Raymond
  - Gespräche Corinna
  - Gesunder Menschenverstand
  - Wissen über Zielgruppe
  - Artikel Ärzteblatt
  - Diplomarbeit Christina Hardt
  - Projekt VitaBit (Schlüsselnummer, Tourenplan)
  - ...
- Kernanforderungen (Prio A)
  - Funktionalität
    - Informationen über Klienten müssen verfügbar sein (Patientenakte) [CHECK]
    - Persönliche Daten → Schutz durch Login-Funktion [CHECK]
    - Dokumentation von Tätigkeiten aus Richtlinie muss möglich sein [CHECK]
    - ambulante und stationäre Pflege abdecken! [CHECK]
    - Dokumentierte Tätigkeiten müssen an zentrales System übertragbar sein [CHECK]
    - Ablaufplan jeweils für eine MFP pro Tag/Schicht [CHECK]
    - Übersicht, ob jeweilige Station schon vollständig abgearbeitet [CHECK]
    - Es muss eine Tätigkeit als *nicht durchgeführt* markiert werden können + Begründung,falls etwas dazwischenkommt [CHECK]
    - Einmal dokumentierte Werte sollten korrigierbar sein [CHECK]

        * Nachvollziehbarkeit von Änderungen [CHECK]
        * Daten "'offline"' auf dem Endgerät verfügbar, d.h. keine zwangsläufige Internetverbindung [CHECK]
        * Dokumentation von Wunden (Fotos) [CHECK]
        * Erfassung von Vitalwerten (Blutdruck, Puls), evtl. mittels automatischer Erfassung [CHECK]
        * Schlüsselnummer für Hausbesuche bzw. Zimmernummer bei stationären Behandlungen (aus VitaBIT) [CHECK]
        * Klar und übersichtlich strukturierte Anwendung, intuitive Bedienung (am besten Touch, Endnutzerwissen) [CHECK]
        * Für ambulanten Dienst eventuell Navigation mittels GPS [CHECK]
        * Telefongespräche? [CHECK]
        * ...
    - Abgeleitete technische Anforderungen
        * sozialer Kontext → günstige Hardware [CHECK]
        * **Hardware:** mobiles Endgerät muss internetfähig sein (am besten über Mobilfunknetze für ambulanten Dienst), eine Kamera haben (Wunddokumentation), am besten Touch-Bedienung, darf trotzdem nicht so teuer sein, evtl. GPS, falls automatische Erfassung evtl. Bluetooth → Android-Tablet [CHECK]
        * ...
    - ...
- Wünschenswerte Eigenschaften, um Nutzungskomfort zu steigern (Prio B) [NEIN]
- Anforderungen, die über die reine Dokumentation oder dessen Unterstützung hinausgehen (Prio C, werden in der Arbeit nicht behandelt) [NEIN]

B. Liste grundsätzliche Entwurfsentscheidungen CareNet*mobile*

- Zentrales System: CareNet, basierend auf VitaBit [CHECK]
- Kommunikation mit REST; Was ist REST? Prinzipien? [CHECK]
- **CareNet mobile**
    - Name [CHECK]
    - Warum überhaupt Tablet? (Vgl. mit Diplomarbeit der anderen Studentin auf Nokia-Handy) [CHECK]
    - Nativ - WebApp - Hybrid [CHECK]

  - PhoneGap, jQuery mobile [CHECK]
  - Falls doch jemand Apple will, nach Mglk. suchen, wie das evtl. auch bedienst werden kann [CHECK]
- Architektur (Plug-In-Struktur) [CHECK]
- Änderungen an zu dokumentierenden Tätigkeiten sind zu erwarten, also möglichst flexible Architektur [CHECK]
  - Warum Plug-In? Vorteile? [CHECK]
- **Middleware**
  - Schnittstellenfunktion nochmal erwähnen [CHECK]
  - Technologie (PHP) [CHECK]
  - Wo wird Middleware positioniert? Welcher Server? (konzeptionell) [NEIN]

C. Basisanwendung

- Grundsätzlich: Plug-In-Architektur angelehnt an Eclipse-Bsp [CHECK]
- Bestandteile: Grundanwendung (Basis), Plug-Ins [CHECK]
- Aufgaben Grundanwendung: Sicherheit (Login, Kommunikation), Verwaltung von Plug-Ins, Bereitstellen einer GUI-Struktur, zentrale Verwaltung lokaler Datenbanken (Methoden) [CHECK]
- Programmiereigenschaften von JavaScrip ggü. Java [CHECK]
- Zentrale Verwaltung von IDs und sonstigen Strings [CHECK]
- Plug-Ins sollten keinen Einfluss auf Grundanwendung haben (Kapselung, aber komplette Trennung nicht möglich) [CHECK]
- Struktur GUI
- Grundstruktur: jQuery mobile (Seitenaufbau, alles mit Divs, CSS, wie Webseite) [MMHHH. . . , NÜTZLICH?]
- Login
  - als Dialog (vorgeschaltete Seite) [CHECK]
  - Mandant, Nutzername, PW [CHECK]
  - Wg. REST-Prinzip: Nur einmalige Überprüfung, ob Daten korrekt sind, kein Aufbau einer dauerhaften Verbindung [CHECK]
  - Speicherung der Daten im lokalen Speicher im Klartext (HTML5/PhoneGap Key-Value-Speicher) [CHECK]
  - Senden der Daten bei jeder Anfrage (Beispielanfrage) [CHECK]

  - Authentifizierungstring: mandant/nutzername:passwort [CHECK]
  - Middleware verschlüsselt String mit Base64 und sendet den an CareNet (kann einfach entschlüsselt werden) [TEIL-CHECK]
  - Daten werden so unverschlüsselt gesendet, d.h. Verbindung an sich muss gesichert werden (https) [CHECK]
  - Bei Logout werden Daten einfach aus lokalem Speicher gelöscht [CHECK]
  - Sicherheitsproblem bei eingeschleuster Spyware (JS) über Browser? Nein, wg. Unabhängigkeit von PhoneGap-Browser/WebView und installierten anderen Browsern [MMMHHH...]

D. Plug-In-Struktur

- Plug-Struktur:
  - Plug-In = Menge von Dateien [CHECK]
  - Alle zugehörigen Dateien gekapselt in einem Ordner [CHECK]
  - Namenskonventionen wg. JS global bzgl. Dateien [CHECK]
  - Namenskonventionen bzgl. Methoden, Variablen, CSS-Klassen [CHECK]
  - Konfiguration eingebundener Plug-Ins (JSON) [CHECK]
  - Icons (Darstellung in Dreierreihen + Grafik) [CHECK]
  - CSS-Datei [CHECK]
  - Beschreibung der Abhängigkeiten zu Bibliotheken wie jQuery, jQuery mobile, PhoneGap (Redundanzen vermeiden) [CHECK]
  - Grafik zu Plug-In-Strukur [CHECK]
  - JS-Datei (Problem der dynamischen Einbindung) [CHECK]
  - Konstruktor, Methode zur Initialisierung [CHECK]
- Probleme (Einbinden von JS und CSS Dateien) [CHECK]

Literaturverzeichnis

[Ado13] (2013, Januar) Phonegap homepage. Adobe Systems Inc. [Online]. Available: http://www.phonegap.com/

[fhFiDHIe08] I. für hausärztliche Fortbildung im Deutschen Hausärzteverband (IhF) e.V., "Fortbildungscurriculum "Versorgungsassistentin in der Hausarztpraxis-VERAH"," Januar 2008.

[fhFiDHIeV08] I. für hausärztliche Fortbildung im Deutschen Hausärzteverband (IhF) e. V., "Versorgungsassistentin in der Hausarztpraxis - VERAH® [Informationen für Praxisinhaber]," 2008.

[GMR12] R. Görlitz, L. Müller, and A. Rashid, "An individual patient-oriented stroke manager architecture," in *Proceedings of the eHealth2012*, Schreier, Hayn, Hörbst, and Ammenwerth, Eds., Wien, Österreich, Mai 2012.

[Goo13] (2013, Januar) Verwaltung von String Ressourcen bei der nativen Android-Anwendungsentwicklung. Google Inc. [Online]. Available: http://developer.android.com/guide/topics/resources/string-resource.html

[Har10] C. Hardt, "Entwicklung eines prototypischen Softwaresystems zur Unterstützung der Kommunikation und Dokumentation in der stationären Pflege," Diplomarbeit, Hochschule Karlsruhe Technik und Wirtschaft; Universität Karlsruhe (TH), April 2010.

[Int05] *ISO/IEC 25000 Software engineering - Software product Quality Requirements and Evaluation SQuaRE - Guide to SQuaRE*, International Standards Organization Std. 1, Rev. 1, 8 2005.

[Jan07] K. Janowicz, *Sicherheit im Internet : [Gefahren einschätzen, Risiken minimieren: fundiertes Hintergrundwissen rund ums Internet; effektiver Schutz für den PC: behandelt Windows XP und Vista; mit zahlreichen Sicherheitstools auf CD-ROM]*, 3rd ed., ser. oreillys basics. Beijing: OReilly, 2007, früher mit der Nummer 9783897214255. [Online]. Available: http://swbplus.bsz-bw.de/bsz26744186xcov.htm;http://www.gbv.de/dms/ilmenau/toc/527923656.PDF

[jQu13] (2013) jquery mobile homepage. The jQuery Foundation. [Online]. Available: http://jquerymobile.com/

[Kli10] D. Klingbeil, “Mehr Zeit für die Pflege der Kunden,” *Häusliche Pflege*, vol. 11, pp. 32–34, November 2010.

[Kop10] D. T. Kopetsch, “Dem deutschen Gesundheitswesen gehen die Ärzte aus! Studie zur Altersstruktur-und Arztzahlentwicklung,” Bundesärztekammer und Kassenärztliche Bundesvereinigung, Studie 5, August 2010.

[Nei12] T. Neil, *Mobile Design Pattern Gallery : [UI Patterns for iOS, Android and more]*, 1st ed. Beijing: OReilly, 2012. [Online]. Available: http://deposit.d-nb.de/cgi-bin/dokserv?id=3962922&prov=M&dok_var=1&dok_ext=htm;http://swbplus.bsz-bw.de/bsz363523030cov.htm

[nG13] nubedian GmbH. (2013, Januar). [Online]. Available: http:/www.nubedian.de/

[Pre06] S. B. Pressestelle, “Bevölkerung Deutschlands bis 2050 - 11. koordinierte Bevölkerungsvorausberechnung,” November 2006.

[Res13] (2013, 01) Residenz Bad Friedrichshall. [Online]. Available: http://www.pflegedienste-hn.drk.de/

[ric12] “Richtlinie des Gemeinsamen Bundesauschusses über die Festlegung ärztlicher Tätigkeiten zur Übertragung auf Berufsangehörige der Alten- und Krankenpflege zur selbständigen Ausübung von Heilkunde im Rahmen von Modellvorhaben nach § 63 Abs. 3c SGB V.” p. 1128, März 2012.

[Sen13] (2013, Januar) Sencha touch homepage. Sencha Inc. [Online]. Available: http://www.sencha.com/products/touch

[The13a] (2013, Januar) Architektur Eclipse. The Eclipse Foundation. [Online]. Available: http://help.eclipse.org/juno/index.jsp?topic=%2Forg.eclipse.platform.doc.isv%2Fguide%2Farch.htm

[The13b] (2013, Januar) PHP Homepage. The PHP Group. [Online]. Available: http://www.php.net

[vdBMH+09] N. van den Berg, C. Meinke, R. Heymann, T. Fiß, E. Suckert, C. Pöller, A. Dreier, H. Rogalski, T. Karopka, R. Oppermann, and W. Hoffmann, “AGnES: Hausarztunterstützung durch qualifizierte Praxismitarbeiter; Evaluation der Modellprojekte: Qualität und Akzeptanz,” *Deutsches Ärzteblatt*, vol. Heft 1-2, no. 106, pp. 3–9, Januar 2009.

[Vit09] F. VitaBIT, “Pflegeservice von morgen,” *Deutsches Ärzteblatt*, vol. 106, no. 50, pp. 3,4, 2009.

[zMNS05] M. zur Muehlen, J. V. Nickerson, and K. D. Swenson, “Developing web services choreography standards - the case REST vs. SOAP,” *Decision Support Systems*, vol. 40, no. 1, pp. 9 – 29, 2005, <ce:title>Web services and process management</ce:title>. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0167923604000612