Medizinische Informatik
Universität Heidelberg
Hochschule Heilbronn

Hochschule Heilbronn
TECHNIK WIRTSCHAFT INFORMATIK

Computer-gestütztes, internet-basiertes Dokumentations- und Überwachungssystem für medizinisches Gehtraining bei Patienten mit peripherer arterieller Verschlusskrankheit

---

Diplomarbeit

Sabrina Lang, 164790

Referent: **Prof. Dr. Martin Haag**

Korreferent: **Prof. Dr. Daniel Pfeifer**

27. November 2010

Danksagung

An dieser Stelle möchte ich mich bei allen Personen bedanken, die mich bei der Erstellung dieser Arbeit unterstützt haben.

Ein besonderer Dank gilt meinem Referenten, Herrn Prof. Dr. Martin Haag, der stets für mich ansprechbar war und mit sehr viel Engagement und unermüdlichem Einsatz meine Diplomarbeit betreut hat.

Ebenfalls danken möchte ich meinem Korreferenten, Herrn Prof. Dr. Daniel Pfeifer für die Unterstützung.

Des Weiteren möchte ich mich bei Herrn Prof. Dr. med. Thomas Dengler für die reibungslose Zusammenarbeit und die Bereitstellung des Themas bedanken.

Großer Dank gebührt meinen Eltern, die, davon abgesehen, dass sie mir das Studium ermöglichten, immer großes Interesse an meiner Arbeit zeigten und mich so gut wie möglich unterstützt haben.

Zu guter Letzt danke ich Simon Pezold und Carolin Roth für das Korrekturlesen meiner Arbeit.

Abstract

Einleitung

Es ist bekannt, dass Sport wichtig für unser Wohlbefinden ist. Darüber hinaus ist körperliche Aktivität gesundheitsfördernd. Bei fast allen kardiovaskulären Erkrankungen wird ergänzend zur medikamentösen Behandlung ein Gehtraining verschrieben („Sport auf Rezept“). Bedauerlicherweise wird Sport in der heutigen Gesellschaft viel zu sehr vernachlässigt. Ein konsequentes Gehtraining ist speziell im Bereich der peripheren Durchblutungsstörungen ausschlaggebend. In Anbetracht dessen ist es erforderlich, dass die Durchführung des Trainings mittels geeigneter Software von Ärzten engmaschig überwacht werden kann. In diesem Zusammenhang wird in Zusammenarbeit mit Herrn Prof. Dr. med. Thomas Dengler, Direktor der medizinischen Klinik I im Klinikum am Plattenwald, ein Dokumentations- und Überwachungssystem für medizinisches Gehtraining bei Patienten mit peripherer arterieller Verschlusskrankheit entwickelt.

Methoden

Für die Umsetzung des Dokumentations- und Überwachungssystems ist strukturiertes Vorgehen erforderlich. Ein bekanntes Modell hierfür ist das Wasserfallmodell. Dabei müssen zunächst die Anforderungen an das System analysiert und in Form von Anwendungsfällen, sogenannten UseCases, festgehalten werden. Parallel dazu muss ein Prototyp für die grafische Oberfläche erstellt werden. Im Anschluss daran erfolgt der Entwurf des Systems mittels Klassen- und Sequenzdiagrammen. Sobald diese Schritte abgeschlossen sind, kann mit der Realisierung begonnen werden.

Ergebnisse

Die Software gestattet, je nachdem wer sich anmeldet, einen unterschiedlichen Funktionsumfang. Prinzipiell können Ärzte und Patienten mit dem System interagieren. Aus Gründen der Datensicherheit läuft das System über eine passwortgeschützte Internetseite.

Diskussion und Ausblick

Das Dokumentations- und Überwachungssystem für medizinisches Gehtraining bei Patienten mit peripherer arterieller Verschlusskrankheit konnte im Rahmen der vorliegenden Arbeit als Webanwendung umgesetzt werden. Darüber hinaus konnten die gewünschten Anforderungen entsprechend realisiert werden. Bisher sind Methoden zur fortlaufenden Überwachung des Therapieerfolges noch nicht flächendeckend implementiert. Mit dieser Software konnte jedoch eine solide Grundlage geschaffen werden, die grundsätzlich für weitere medizinische Fragestellungen genutzt werden kann.

Inhaltsverzeichnis

Aufbau und Gliederung der Arbeit

Die Diplomarbeit ist in drei Bereiche unterteilt. Der erste Teil dieser Arbeit (Kapitel 1) umfasst den Hintergrund sowie die Zielsetzungen dieser Arbeit. Darüber hinaus wird das nötige Wissen über die periphere arterielle Verschlusskrankheit vermittelt.

Im zweiten Teil (Kapitel 2 und 3) werden die theoretischen Grundlagen für die zu erstellende Software geschaffen. In Kapitel 2 werden zunächst die Anforderungen an eine grafische Oberfläche (GUI) erörtert, bevor mit der Vorstellung der einzelnen Technologien begonnen wird. Abgerundet wird das Kapitel mit einem Vergleich zur Auswahl einer geeigneten Technologie. Kapitel 3 hingegen beschreibt die Technologien, die für die Realisierung des Backends benötigt werden.

Im dritten und letzten Teil dieser Arbeit (Kapitel 4 bis 7) wird die Vorgehensweise zum Erstellen der Software und deren technische Umsetzung schrittweise erläutert. Die Diplomarbeit wird mit einem Fazit über die im Rahmen dieser Arbeit entstandene Software abgeschlossen. Außerdem wird ein Ausblick für mögliche Ansätze zur Weiterentwicklung gegeben.

Im Folgenden die Kapitel im Überblick:

Kapitel 1 beschreibt das Umfeld, in dessen Rahmen die Diplomarbeit entstanden ist. Darüber hinaus wird die Aufgabenstellung der Diplomarbeit erläutert und die damit verbundenen Anforderungen erörtert.

Kapitel 2 stellt die notwendigen Grundlagen für die Realisierung der GUI vor. Client-seitige Technologien wie JavaServer Faces (JSF), Spring Web Framework oder das Google Web Toolkit (GWT) werden kurz erläutert und ein Vergleich zur Auswahl der Technologie durchgeführt. Insbesondere werden die Anforderungen an eine GUI vorgestellt.

Kapitel 3 schildert die Grundlagen für die Realisierung des Backends und beschreibt den Einsatz des Spring Frameworks.

Kapitel 4 beschreibt detailliert die erforderlichen Schritte, um eine Software zu konzipieren.

Kapitel 5 erläutert die Implementierung der Software und deren Realisierung mit den bereits vorgestellten Technologien.

Kapitel 6 zeigt die im Rahmen dieser Arbeit entstandene Software. Des Weiteren werden mögliche Testverfahren vorgestellt. Das Kapitel wird mit einem Benutzerfeedback von Herrn Prof. Dr. med. Thomas Dengler abgeschlossen.

Kapitel 7 diskutiert die bereits implementierte Software und es wird ein Ausblick über ihre weitere Entwicklung gegeben.

Anhang I enthält sämtliche Anwendungsfälle, sogenannte UseCases. Diese beschreiben aus Sicht des Benutzers eine Abfolge von Arbeitsschritten, die durch die entwickelte Software unterstützt werden sollen.

Anhang II führt die verwendeten Lizenzen auf und erläutert diese kurz.

1. Einleitung

Im September 2010 startete der Ultramarathon „in 31 Tagen nach Athen“. Am 20. September 2010 trat Jürgen Mennel[1] die 2.200 Kilometer lange Strecke in Heilbronn an, bei der es pro Etappe 70 bis 80 Kilometer zurückzulegen galt. Jürgen Mennel zeigt seit Kindesalter großes Interesse am Langstreckenlauf. Bislang hat er in seinem Leben 300.000 Kilometer absolviert. Anlässlich des 2.500-jährigen Jubiläums des Marathonmythosses nahm der 50-jährige Läufer die Herausforderung an.

Begleitend zu diesem Lauf fand eine elektronische Wissensstafette (e-Wissensstafette) statt, welche vom Studiengang Medizinische Informatik der Hochschule Heilbronn und der Universität Heidelberg unterstützt wurde. Die ebenfalls 2.200 Kilometer lange Wissensstafette hatte ihren Start- und Endpunkt in Heilbronn und lief in einem dreimaligen Rundkurs über Heidelberg, Tübingen, Stuttgart und Ulm, siehe Abbildung 1. Parallel zum Lauf von Herrn Jürgen Mennel lieferten alle Teilnehmer im Rahmen der e-Wissensstafette kontinuierlich wissenschaftliche Beiträge zu den Themen Sport, Gesundheit, Medizin und Informatik. Mit dieser Wissensstafette sollte vermittelt werden, dass Bewegung sehr wichtig für die Gesundheit ist, um zum Einen ein erhöhtes Wohlbefinden zu erreichen, zum Anderen aber vor allem Erkrankungen vorzubeugen.

1 Die Webseite des Athleten ist auf www.athenlauf.de zu finden.

Abbildung 1: Beteiligte Standorte der e-Wissensstafette [Webseite des Athleten]

Heutzutage wird körperliche Aktivität bei fast allen kardiovaskulären Erkrankungen ergänzend zur medikamentösen Behandlung verordnet („Sport auf Rezept“). Jedoch wird in der Praxis die Bewegungstherapie von Patienten nicht konsequent genug eingesetzt, so dass eine fortlaufende Überwachung dieser als erforderlich erscheint. Speziell im Bereich der peripheren Durchblutungsstörungen (periphere arterielle Verschlusskrankheit) ist der medizinische Nutzen eines Gehtrainings besonders gut nachgewiesen.

1.1 Periphere arterielle Verschlusskrankheit

Bei der peripheren arteriellen Verschlusskrankheit (pAVK) handelt es sich um eine krankhafte Verengung an den Arterien der Beine. Aufgrund dieser Verengungen kann es zu Durchblutungsstörungen in den Beinen kommen, die die Betroffenen zu Gehpausen zwingen. Aus diesem Grund wird die Krankheit auch als Schaufensterkrankheit bezeichnet [NetDoktor, a]. Die Verengungen können an unterschiedlichen Stellen auftreten. Die Ärzte unterscheiden dabei zwischen dem Oberschenkel-, Becken- und Unterschenkel- sowie dem Mischtyp. Des Weiteren muss zwischen links-, rechts- oder beidseitig kategorisiert werden. Die Prävalenz der Durchblutungsstörung ist altersabhängig und steigt mit höherem Lebensalter an. In Deutschland leiden ungefähr 15 bis 20 Prozent an dieser Krankheit, wobei Männer fünfmal häufiger als Frauen betroffen sind [NetDoktor, a].

Die Ursache für diese Krankheit ist in den meisten Fällen auf eine Arteriosklerose[2] zurückzuführen. Eine weitere bedeutende Ursache, neben Bluthochdruck, Diabetes Mellitus[3] und einem erhöhten Cholesterinspiegel, ist das Rauchen, welches keinesfalls unterschätzt werden darf [NetDoktor, b]. Bei der Krankheit werden Gefäßbereiche, die hinter der verengten Stelle liegen, nicht mehr gut durchblutet. Aufgrund der schlechten Versorgung können Beschwerden auftreten, die jedoch erst später von den Betroffenen wahrgenommen werden, da der Körper über bestimmte Mechanismen verfügt, die versuchen, die Mangelversorgung auszugleichen. Oftmals treten Schmerzen bei Belastung auf, die als erstes Zeichen zu sehen sind und die die Betroffenen zu Gehpausen zwingen [MedizInfo®].

2 Umgangssprachlich als *Arterienverkalkung* bekannt.

3 Umgangssprachlich als *Zuckerkrankheit* bekannt.

Nach *Fontaine-Ratschow* [NetDoktor, c] lässt sich die Krankheit in vier Stadien unterteilen, die auf nachfolgender Tabelle ersichtlich sind.

Stadium	Beschreibung
1	Keine Beschwerden
2a	Belastungsschmerzen ab einer Strecke von 200 Metern
2b	Belastungsschmerzen ab einer Strecke unter 200 Metern
3	Ruheschmerz
4	Zusätzliche Gewebeschädigungen, Entzündungen und Geschwüre als Zeichen, dass das Gewebe abstirbt

Tabelle 1: Stadien der pAVK [NetDoktor, c]

Zu den Maßnahmen um die Ursachen zu bekämpfen zählen vorwiegend der frühzeitige Stopp des Rauchens sowie eine gesunde Ernährung und ein Gehtraining. Insbesondere ab Stadium 2 stellt das Gehtraining einen wichtigen Bestandteil der Therapie dar. Dabei wird zum Einen die Strecke ermittelt, die die Betroffenen schmerzfrei bewältigen können. Zum Anderen wird die Strecke analysiert, die sie zurücklegen können, ehe sie abbrechen müssen [NetDoktor, d]. Die Bewertung der Gehstrecke ist die am häufigsten verwendete Methode, um die Wirkung der Behandlung bei diesen Patienten zu untersuchen. Darüber hinaus können zur Beurteilung „patientenbasierte validierte erkrankungs-spezifische Fragebögen herangezogen werden“ [AWMF], wie z.B. der Walking Impairment Questionare (WIQ) oder der Medical Outcome Short Form SF-36.

Im Rahmen einer Validierungsstudie wurden Patienten mit peripherer arterieller Verschlusskrankheit zu einer überwachten Bewegungstherapie herangezogen. Dazu wurden an die Studienteilnehmer Fragebögen verteilt, jeweils zu Beginn und nach drei Monaten, am Ende der Studie. Die ausgewerteten Daten der Studie zeigen, dass die Fragebögen ein geeignetes Instrument sind, um Verbesserungen oder Verschlechterungen in der täglichen Gehfähigkeit des Patienten mit pAVK zu erkennen [Nicola 2009].

Die vorläufigen Ergebnisse einer weiteren Studie, die das Laufverhalten zwischen krankenhaus-überwachtem Gehtraining und selbst gestaltetem Gehtraining analysieren und vergleichen, zeigen, dass die überwachten, krankenhausspezifischen Übungen wesentlich mehr zur Verbesserung beitragen, als die unüberwachten, heim-spezifischen Übungen [Regensteiner 1997].

In einer weiteren Studie wurde untersucht, ob ein heim-basiertes Programm für Gehtraining mit strukturiertem Coaching eine Verbesserung der Laufleistung bei Patienten mit pAVK erreichen kann. Dazu wurden 31 Patienten ausgewählt, die an einem 24-wöchigen Gehtraining-Programm in ihrer häuslichen Umgebung teilnahmen. Die Teilnehmer mussten unter anderem ein Lauftagebuch führen und Fragebögen ausfüllen. Die Studie zeigte, dass die Patienten ihren durchschnittlichen Knöchel-Arm-Index, welcher zur Feststellung einer peripheren arteriellen Verschlusskrankheit errechnet wird, verbessern konnten. Des Weiteren lassen die Ergebnisse darauf schließen, dass ein solches Programm innerhalb einer häuslichen Umgebung eine viel versprechende Intervention darstellt [Wullink 2001].

Die Studien zeigen, dass sich eine überwachte Durchführung positiv auf die Gehfähigkeit der Patienten auswirkt. In Anbetracht dessen wird in Zusammenarbeit mit Herrn Prof. Dr. med. Thomas Dengler, Direktor der Medizinischen Klinik I im Klinikum am Plattenwald, eine Software entwickelt, mit welcher die Möglichkeit besteht, die Durchführung und den Erfolg des medizinischen Gehtrainings, speziell bei Patienten mit pAVK, zu überwachen und zu kontrollieren. Diese Aufgabenstellung entstand im Rahmen der e-Wissensstafette.

1.2 Aufgabenstellung und Zielsetzung

Im Rahmen dieser Diplomarbeit soll das vorgenannte computer-gestützte, internet-basierte Dokumentations- und Überwachungssystem für medizinisches Gehtraining bei Patienten mit pAVK entwickelt werden. Welche Ziele mit diesem Vorhaben verbunden sind, wird nachfolgend aufgeführt.

Erwünschte Funktionalitäten des Dokumentations- und Überwachungssystems sind...

- ...die Benutzerverwaltung, die Ärzten und Patienten einen unterschiedlich großen Funktionsumfang gewährleistet
- ...die Möglichkeit als Arzt neue Benutzerkonten hinzuzufügen, bereits bestehende zu bearbeiten und zu löschen
- ...die Möglichkeit als Patient seine Messwerte zu erfassen und sich diese in Form von Diagrammen grafisch darstellen zu lassen
- ...die Möglichkeit als Patient seine Messwerte bei fehlerhafter Eingabe zu bearbeiten
- ...die Möglichkeit als Arzt die Kontrolle über die Messwerte eines einzelnen Patienten aufrecht zu halten
- ...die Möglichkeit einen Alarm auszulösen, wenn einer der Parameter einen der zuvor festgelegten Grenzwerte überschritten hat
- ...die Möglichkeit für Patienten und Ärzte, Nachrichten untereinander auszutauschen

Die beschriebenen Rahmenbedingungen sollen in Form von UseCases festgehalten werden.

Ferner soll die Software den folgenden Anforderungen genügen:

Software-ergonomische Aspekte

Insbesondere bei der Bedienung der Software soll auf ergonomische Aspekte geachtet werden, um die Benutzerakzeptanz zu steigern.

Erweiterbarkeit

Das Dokumentations- und Überwachungssystem soll leicht erweiterbar sein, falls zukünftig weitere Funktionalitäten erwünscht sind. Dabei soll darauf geachtet werden, dass die Änderungen an möglichst wenigen Stellen durchgeführt werden müssen.

Wiederverwendbarkeit

Der grundlegende Aufbau der Software soll für weitere medizinische Fragestellungen genügen.

Browserunabhängigkeit

Die Software soll auf allen gängigen Browsern, wie beispielsweise Mozilla Firefox, Internet Explorer oder Opera laufen.

Sicherheit

Es muss jederzeit Datensicherheit und -integrität gewährleistet werden, da medizinische Daten über das Internet zwischen allen Beteiligten ausgetauscht werden.

1.3 Eingesetzte Ressourcen und Software

Im Laufe der Erstellung des Dokumentations- und Überwachungssystems werden verschiedene Programme und Frameworks eingesetzt, die als Open-Source bezeichnet werden und somit für jeden frei zugänglich und verfügbar sind, um die laufenden Kosten möglichst gering zu halten.

Eine Übersicht über die verwendeten Lizenzen befindet sich im Anhang II.

2. GUI-Technologien

In diesem Kapitel werden zunächst die Anforderungen zur Erstellung von grafischen Benutzeroberflächen erörtert. Anschließend erfolgt die Vorstellung von drei möglichen Client-Technologien. Das Kapitel wird mit einem Vergleich zur Auswahl einer Technologie abgeschlossen.

2.1 Anforderungen an eine GUI

Bei der Erstellung von interaktiven Systemen ist es besonders wichtig, die software-ergonomischen Aspekte zu berücksichtigen. Allerdings ist der Entwurf von grafischen Oberflächen nicht trivial. Um ein effizientes Arbeiten zu gewährleisten, müssen die Benutzeroberflächen an die Bedürfnisse ihrer Benutzer angepasst werden. Das GUI-Design ist ein sehr umfangreiches Thema, so dass im Rahmen dieser Arbeit nur grundlegende Aspekte angesprochen werden können. Generell werden Prinzipien zur Gestaltung von Benutzeroberflächen konzipiert und deren Ergebnisse in Form von Normen festgehalten. Eine bekannte Norm im Bereich der Software-Ergonomie ist die DIN EN ISO 9241 [ISO]. Seit 2006 trägt sie den deutschen Namen „Ergonomie der Mensch-System-Interaktion“. Diese Norm umfasst 17 Teile, welche Anforderungen an die Bereiche Hardware, Software und Arbeitsumgebung beschreiben. Dabei werden die Aspekte der Software-Ergonomie in den Teilen 11-17 und 10 bzw. 110 behandelt.[4]

4 Der Abschnitt 10 wurde im Jahr 2006 überarbeitet und in 110 geändert.

Essentiell für die Gestaltung von interaktiven Systemen sind die sieben Grundsätze der Dialoggestaltung, die nachstehend kurz allgemein erläutert und jeweils mit einem Beispiel bestückt werden.

1. Aufgabenangemessenheit

   „Ein Dialog ist aufgabenangemessen, wenn er den Benutzer unterstützt, seine Arbeitsaufgabe effektiv und effizient zu erledigen.“ [ISO]

   - **Vorgabe von Standardwerten**

     Den Benutzern sollten Standardwerte als Vorgabe angeboten werden, die bei Bedarf überschrieben werden können.

2. Selbstbeschreibungsfähigkeit

   „Ein Dialog ist selbstbeschreibungsfähig, wenn jeder einzelne Dialogschritt durch Rückmeldung des Dialogsystems unmittelbar verständlich ist oder dem Benutzer auf Anfrage erklärt wird.“ [ISO]

   - **Feedback**

     Benutzer sollten über den Erfolg oder Misserfolg einer Aktion informiert werden.

3. Erwartungskonformität

   „Ein Dialog ist erwartungskonform, wenn er konsistent ist und den Merkmalen des Benutzers entspricht, z.B. den Kenntnissen aus dem Arbeitsgebiet, der Ausbildung und der Erfahrung des Benutzers sowie den allgemein anerkannten Konventionen.“ [ISO]

   - **Konsistenz**

     Verwendung von bekannten Mustern. Innerhalb einer Anwendung sollten gleichartige Informationen immer gleich benannt und an den gleichen Stellen dargestellt werden.

4. Fehlertoleranz

   „Ein Dialog ist fehlertolerant, wenn das beabsichtigte Arbeitsergebnis trotz erkennbar fehlerhafter Eingaben entweder mit keinem oder mit minimalem Korrekturaufwand durch den Benutzer erreicht werden kann.“ [ISO]

   - **Korrekturmöglichkeiten**

     Es sollte die Möglichkeit bestehen gespeicherte Datensätze, die verbesserungswürdig erscheinen, zu korrigieren.

5. Steuerbarkeit

   „Ein Dialog ist steuerbar, wenn der Benutzer in der Lage ist, den Dialogablauf zu starten sowie seine Richtung und Geschwindigkeit zu beeinflussen, bis das Ziel erreicht ist.“ [ISO]

   - **Abbruchmöglichkeiten**

     Die Benutzer sollten zu jedem Zeitpunkt die Vorgänge abbrechen können.

6. Individualisierbarkeit

   „Ein Dialog ist individualisierbar, wenn das Dialogsystem Anpassungen an die Erfordernisse der Arbeitsaufgabe, individuelle Vorlieben des Benutzers und Benutzerfähigkeiten zulässt.“ [ISO]

   - **Browserunabhängigkeit**

     Die Software sollte auf allen gängigen Browsern anwendbar sein, damit die Endanwender selbst entscheiden können, welchen Browser sie verwenden möchten.

7. Lernförderlichkeit

   „Ein Dialog ist lernförderlich, wenn er den Benutzer beim Erlernen des Dialogsystems unterstützt und anleitet.“ [ISO]

   - **Hilfeseiten**

     Benutzern sollte die Möglichkeit einer Hilfe eingeräumt werden.

2.2 Technologien

Allein im Java-Bereich gibt es etliche, als Open-Source veröffentlichte Frameworks zur Entwicklung von Benutzeroberflächen für Webanwendungen. Bekannte moderne Technologien dafür sind, neben Apache Struts[5], beispielsweise JavaServer Faces oder das Google Web Toolkit. Auf Empfehlung von Prof. Dr. Martin Haag soll das Google Web Toolkit als Technologie für die Oberfläche verwendet werden. Alternativ bringt allerdings auch das im Backend der Software verwendete Spring Framework ein eigenes Modul zur Erstellung grafischer Oberflächen mit, das sogenannte Spring Web Framework (vgl. Zeitner, 2008). In der Diplomarbeit von Frank Hess des Studiengangs Medizinische Informatik kommt außerdem die schon erwähnte Java-basierte Oberflächentechnologie JavaServer Faces erfolgreich zum Einsatz (vgl. Hess, 2006). Die drei genannten Technologien werden daher im Folgenden vorgestellt.

2.2.1 JavaServer Faces

JSF[6] ist ein Framework, welches mittels User Interface (UI)-Komponenten Unterstützung bei der Entwicklung von grafischen Benutzeroberflächen bietet [Jacobi 2006, S. 22]. Die JSF-Spezifikation wurde im Rahmen des Java Community Process von der Expertengruppe Java Specification Request (JSR) entwickelt, an der bekannte Unternehmen wie beispielsweise IBM beteiligt waren. Die Spezifikation beruht auf dem Model-View-Controller (MVC)-Pattern, welches eine Applikation in ein Datenmodell, eine Präsentationsschicht und eine Geschäftslogik unterteilt und somit für eine strukturelle Trennung dieser unterschiedlichen Anwendungsaspekte sorgt.

5 Ein Open-Source-Framework zur Erstellung von Benutzeroberflächen.

6 Die Projektseite von JavaServer Faces ist auf https://javaserverfaces.dev.java.net/ zu finden.

Eine ausführliche Beschreibung zum Thema JSF ermöglicht das Fachbuch von Marinschek et al., 2007. Darüber hinaus sei der Leser auf das Grundlagenwerk „Design Patterns“ von Gamma et al. verwiesen, sofern weiteres Interesse an Informationen über das MVC-Entwurfsmuster besteht.

2.2.2 Spring Web Framework

Das Spring Web Framework bietet bei der Entwicklung von Weboberflächen nutzbare Standardimplementierungen für viele Situationen an. Dazu zählen beispielhaft ein Validierungsmechanismus, welcher auf unkorrekte Dateneingaben reagiert, Tag-Bibliotheken für den Einsatz auf JavaServer Pages (JSP)-Seiten und Controller-Klassen für die typische Datenverarbeitung von Formularen [Auer 2010, S. 13]. Des Weiteren achtet dieses Framework ebenfalls darauf, dass das MVC-Pattern konsequent eingesetzt wird. Dieses Framework basiert auf dem Java Servlet Application Programming Interface (API). Die Konfiguration der einzelnen Spring-Komponenten erfolgt über eine spring-eigene Extensible Markup Language (XML)-Datei. Genauere Erläuterungen zum Spring Web Framework können im Fachbuch (Zeitner et al., 2008) in Kapitel 6 nachgelesen werden.

2.2.3 Google Web Toolkit

Google veröffentlichte im Jahre 2006 das Google Web Toolkit[7]. Mit dieser quelloffenen API können Rich Internet Applications (RIA) im allgemeinen und Asynchronous JavaScript and XML (AJAX)-Anwendungen im Speziellen entwickelt werden [Hanson 2007, S. 3].

AJAX ist einer der wichtigsten Entwickler-Trends im Bereich des Internets und entwickelt sich immer mehr zum Standard. Das Konzept von AJAX basiert auf einer asynchronen Datenübertragung zwischen Webbrowser und -server. Dieses Konzept ermöglicht, dass Hypertext Transfer Protocol (HTTP)-Anfragen durchgeführt werden können, während eine Hypertext Markup Language (HTML)-Seite angezeigt wird. Des Weiteren können HTML-Seiten verändert werden, ohne dass diese komplett neu geladen werden müssen [Marinschek 2007, S. 220]. Somit verbessert sich nicht nur das Nutzererlebnis sondern auch die Akzeptanz von Webanwendungen.

Ein bedeutender Unterschied von GWT zu anderen Technologien besteht darin, dass der client-seitige Code in Java anstatt in JavaScript (JS) geschrieben werden kann [Hanson 2007, S. 3]. Zentrales Element ist der GWT-Compiler, der Java in JS konvertiert und Code generiert, welcher browserunabhängig ausgeführt werden kann [Hanson 2007, S. 6, 7].

7 Die Projektseite von GWT ist unter http://code.google.com/intl/de-DE/webtoolkit/ zu finden.

Unterstützt werden dabei nachfolgende Browser, für die je nach Bedarf eine andere JS-Datei generiert wird [Hanson 2007, S. 7]:

- Firefox
- Internet Explorer
- Safari
- Opera

Der Compiler läuft unsichtbar im Hintergrund, wenn Entwickler ihre GWT-Anwendung zu Testzwecken erst einmal im Hosted Mode[8] auf dem Server ausführen. Mehr zum Thema Google Web Toolkit kann im Fachbuch (Hanson et al., 2007) nachgelesen werden.

8 Eine Umgebung zum Testen und Debuggen einer GWT-Anwendung.

2.2.4 Vergleich der Client-Technologien

Nach den einführenden Worten zu den Technologien werden nun jeweils die wichtigsten Vor- und Nachteile dieser erörtert. Im Anschluss daran wird die im Projekt getroffene Entscheidung zur Wahl einer der Technologien begründet.

JavaServer Faces	
Vorteile:	• leistungsfähige Webanwendung [Jacobi 2006, S. 11] • Open-Source • MVC-Pattern → Flexibilität, Austauschbarkeit[9] [Turau 2004, S. 158] • bietet Unterstützung bei der Entwicklung von AJAX-Anwendungen an [Marinschek 2007, S. 250]
Nachteile:	• Kenntnisse von vielen verschiedenen Techniken (HTML, Cascading Style Sheets (CSS), JSP) notwendig [Turau 2004, S. 181f.]

Tabelle 2: Vor- und Nachteile von JavaServer Faces

Spring Web Framework	
Vorteile:	• MVC-Pattern [Zeitner 2008, S. 215] • Open-Source [Zeitner 2008, S. 18] • leichte Integration mit anderen Technologien [Zeitner 2008, S. 19]
Nachteile:	• intensive XML-Konfiguration [Zeitner 2008, S. 214]

Tabelle 3: Vor- und Nachteile vom Spring Web Framework

9 Siehe S.14.

Google Web Toolkit	
Vorteile:	• Open-Source • client-seitiger Code in Java[10] • einfache Installation • als Eclipse Plugin verfügbar • effektiv, effizient und robust zu entwickeln • desktopähnliches Design [Hanson 2007, S. 34] • kein JS-Wissen nötig • browserunabhängig[11] • Integration mit anderen Technologien möglich [Hanson 2007, S. 3] • vollständige Debug-Umgebung [Hanson 2007, S. 4] • bietet Unterstützung für das Testframework JUnit an [Hanson 2007, S. 4]
Nachteile:	• JS muss im Browser aktiviert sein • viel Einarbeitungszeit • Integration in bestehende Infrastrukturen schwierig (Servlets sind eng in den Remote Procedure Call (RPC)- Mechanismus eingebunden)

Tabelle 4: Vor- und Nachteile vom Google Web Toolkit

10 Siehe S. 16.

11 Siehe S. 17.

Die Entscheidung fiel auf das Google Web Toolkit, weil damit Benutzeroberflächen im Java-Umfeld erstellt werden können, die ein desktopähnliches Design aufweisen, ohne dafür weiteres Wissen über JS, HTML und CSS zu benötigen. Des Weiteren lässt sich der Code in Java schreiben, was sich für die Entwicklung und Nutzung von bewährten Entwicklungsumgebungen, wie beispielsweise Eclipse, als vorteilhaft erweist. Darüber hinaus gestattet das Google Web Toolkit die Erstellung von Webanwendungen mit server-seitigen Technologien. Dadurch können einige Eigenschaften wie Sicherheit und Performance, die Java bietet, in den Web-Client übernommen werden. Zudem ist mir das GWT aus dem Praktikum *Internet-Applikationen* bekannt und dadurch konnte die Einarbeitungszeit zur Verwendung einer anderen Oberflächentechnologie eingespart werden.

Zusätzlich zu GWT wird eine Java-Bibliothek namens Ext GWT (GXT)[12] verwendet. Diese Bibliothek bringt vorgefertigte Widgets[13] mit und gestattet somit das Erstellen von Webanwendungen mit dem GWT.

12 Die Projektseite von GXT ist auf http://www.sencha.com/products/gwt zu finden.

13 Widgets sind Komponenten, welche auf der Browserseite zu sehen sind.

3. Technologien für das Backend

In diesem Kapitel werden die Technologien für das Backend beschrieben. Insbesondere wird das Spring Framework, welches sehr vielfältig eingesetzt werden kann, vorgestellt.

3.1 Java Servlets

Als Servlets werden Softwarekomponenten bezeichnet, die mit Hilfe der Java Servlet API eine HTTP-Anfrage entgegennehmen, eine Verarbeitung ausführen und eine HTTP-Antwort zurückliefern [Moczar 2005, S. 124]. Dabei dient der Web-Container der Java-Enterprise-Edition (JavaEE)-Spezifikation als Laufzeitumgebung für Servlets. Diese Komponenten müssen immer die Schnittstelle `javax.servlet.Servlet` oder eine davon abgeleitete Schnittstelle implementieren. Alle Metainformationen zu Servlets müssen im sogenannten Deployment Descriptor, in Form einer XML-Konfigurationsdatei, der `web.xml`, festgehalten werden.

3.2 Spring Framework

Das frei zugängliche Spring Framework[14], kurz Spring, soll nach seinem Gründer *Rod Johnson* die Entwicklung von Java- und Java-EE-Anwendungen vereinfachen und gute Programmierpraktiken fördern [Rod Johnson]. Spring ist unter der Apache-Lizenz 2.0 frei verfügbar und steht unter der Leitung der Firma *SpringSource*, welche im September 2009 von *VMware* übernommen wurde. Das Framework bietet mit seinen zahlreichen Modulen fertige Basisimplementierungen in Form von Schnittstellen an und basiert auf einem Plain Old Java Object (POJO)[15]-Programmiermodell.

14 Die Projektseite von Spring ist auf http://www.springsource.org/ zu finden.

15 Einfache Java-Klassen mit Getter- und Setter-Methoden.

Ein weiteres Architekturmerkmal von Spring ist Dependency Injection, welches das gegenseitige Injizieren von Java-Objekten ermöglicht. Dependency Injection findet seine Anwendung in der *BeanFactory*, die im Folgenden *ApplicationContext* genannt wird. Bei Spring werden alle Objekte, die von der Factory verwaltet werden sollen, als *Beans* bezeichnet. Es müssen XML-Konfigurationsdateien angelegt werden, die dem *ApplicationContext* mitteilen, welche *Beans* er zu verwalten hat. Eine ausführliche Beschreibung zum Thema *ApplicationContext* und Dependency Injection kann in (Zeitner et al., 2008), Kapitel 2 nachgelesen werden.

3.2.1 Spring JDBC

Die Schnittstellen aus dem Java Database Connectivity (JDBC)[16]-Paket bieten Unterstützung beim Datenbankzugriff an, sofern ein bestimmtes Architekturmuster verwendet wird. Ein wichtiges Muster hierfür stellen die Datenzugriffsobjekte dar. Ein Data Access Object (DAO) bündelt an einer Stelle sowie pro Objekt alle Zugriffe auf die Datenbank. Dieser Ansatz hat den Vorteil, dass der Datenzugriff an einer zentralen Stelle geändert werden kann, ohne dass der Code an mehreren Stellen angepasst werden muss. Das Spring Framework gewährt den Datenbankzugriff mittels JDBC, Hibernate[17] oder der Java Persistence API (JPA). Dabei wird der „Code zum Initialisieren, Öffnen und Schließen von Datenbankverbindungen, zum Connection Pooling, zum Vorbereiten von Query-Objekten, zum Übertragen der Ergebnisse in Java-Objekte und zur Fehlerbehandlung“ [Auer 2010, S.12] bereits von den Spring-Klassen abgedeckt.

16 Java-Datenbankschnittstelle

17 Hibernate ist ein Open-Source-Persistenz-Framework für Java.

Der Entwickler muss lediglich Structured Query Language (SQL)[18]-Statements definieren und ausführen und einmalig beschreiben, „wie Felder von Java-Domänenobjekten auf Datenbankspalten abgebildet werden“ [Auer 2010, S. 12] sollen. Wie bereits bekannt, müssen die dafür benötigten Parameter in einer Konfigurationsdatei notiert werden. Grundlagen zu Spring-JDBC werden in (Zeitner et al., 2008), Kapitel 4 erörtert.

3.2.2 Spring Mail

Spring bietet in seinem E-Mail-Modul nützliche Hilfsklassen zum Senden und Empfangen von E-Mails an. Dieses Modul verwendet dafür die JavaMail API. Alle notwendigen Parameter müssen, wie üblich, in einer spring-eigenen Konfigurationsdatei vermerkt werden. Für weitere Informationen sei der Leser auf (Zeitner et al., 2008), Kapitel 8 verwiesen.

18 Eine Datenbanksprache zur Abfrage von Daten in einer Datenbank.

4. Konzeption

Um das Dokumentations- und Überwachungssystem in die Tat umsetzen zu können, bedarf es einer strukturierten Planung. Dabei sind für die Vorgehensweise zur Entwicklung von Anwendungssystemen verschiedene Ansätze denkbar. Ein bekanntes Modell im Umfeld der Softwareentwicklung ist das Wasserfallmodell, siehe Abbildung 2.

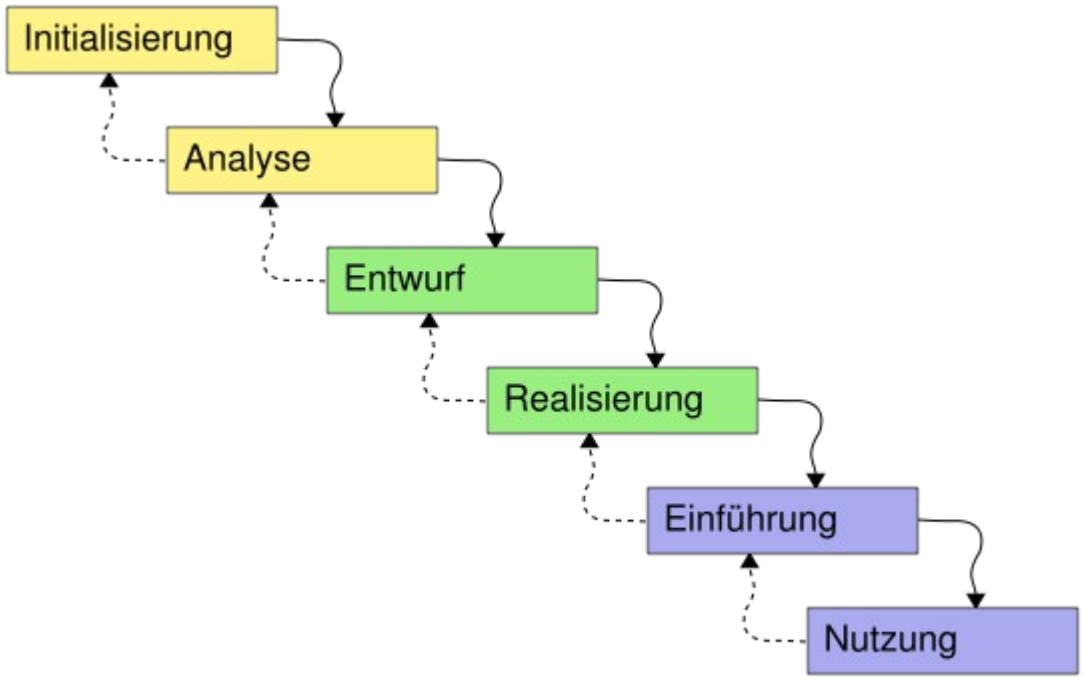


Abbildung 2: Wasserfallmodell [Wikipedia]

Herr Prof. Dr. med. Thomas Dengler hatte seine Vorstellungen zu Beginn des Projektes sehr konkretisiert, so dass auf ein iteratives Vorgehen verzichtet und das Wasserfallmodell angewendet werden konnte. Bei diesem Modell wird der Entwicklungsprozess in Phasen unterteilt, deren Ergebnisse zwingend für die darunter liegenden Phasen sind. Somit erlaubt das Modell eine klare Abgrenzung der jeweiligen Phasen.

Zu Beginn werden die Anforderungen erhoben, d.h. es werden alle Fakten zum betreffenden Projekt-Kontext gesammelt, dargestellt und überprüft. Die Anforderungsanalyse dient dazu, um ein gemeinsames Verständnis zwischen allen Beteiligten herzustellen. Typisch für diese Phase ist das Beschreiben von Anwendungsfällen, den sogenannten UseCases. Aus Sicht des Anwenders sollen diese eine Reihe von Arbeitsschritten beschreiben, welche durch das zu entwickelnde System unterstützt werden. Des Weiteren sollen damit die funktionalen Anforderungen aufgenommen und die Interaktion zwischen dem Anwender und dem System beschrieben werden. Die für die geplante Webanwendung erstellten UseCases befinden sich im Anhang I. Das Kapitel Konzeption bezieht sich immer wieder auf die UseCases. Aus diesem Grund sei an dieser Stelle angemerkt, dass für das weitere Vorgehen das Durchlesen der UseCases zwingend ist.

Begleitend zu den Anwendungsfällen wird meistens ein Prototyp für die grafische Oberfläche entwickelt. Der Prototyp besitzt noch keinerlei Funktionalität, sondern bildet die Bedienungsabläufe mit beispielhaften Daten ab. Mittels des Prototyps kann erkannt werden, mit welchen Datensätzen das System interagieren muss und welche Operationen dafür nötig sind. Letztendlich kann aus den gewonnenen Erkenntnissen ein Domänenmodell erstellt werden, welches im nachfolgenden Kapitel näher beschrieben wird.

4.1 Domänenmodell

Ein Domänenmodell besteht aus mehreren Domänenobjekten, jenen Klassen, die die Datenbankstruktur abbilden. In den meisten Fällen enthalten diese Domänenobjekte nur Attribute und eine Ansammlung von `Getter-` und `Setter-` Methoden. Aus den Anforderungen an die geplante Software geht hervor, dass Patienten ihre Messwerte, bestehend aus Parametern, die unter anderem das Gehtraining betreffen, in das System eintragen müssen. Aus diesem Grund stellt ein Domänenobjekt namens `MeasuredDataDto`, wobei DTO für Data Transfer Object steht, eine sinnvolle Datenstruktur dar. Allein aus dieser Anforderung wird ersichtlich, dass ein Messwert Merkmale wie schmerzfreie Gehstrecke, Abbruch, Dauer oder das Datum besitzen muss.

Des Weiteren kann aus den UseCases entnommen werden, dass verschiedene Personen, in den UseCases durch unterschiedliche Akteure verdeutlicht, mit dem Anwendungssystem interagieren können. Beispielsweise werden dem betreuenden Arzt die Messwerte eines Patienten in Form einer Tabelle und in Form von Diagrammen angezeigt. Somit erhält der Arzt die Möglichkeit der fortlaufenden Kontrolle über Durchführung und Therapieerfolg. Aus diesen Gründen ist es sinnvoll ein Domänenobjekt `UserDto` mit den gemeinsamen Merkmalen, wie Vor- und Nachname, Benutzername, Passwort und der E-Mailadresse anzulegen. Alle weiteren Merkmale, die charakteristisch für die jeweilige Person sind, werden in einer eigenen Klasse notiert. Deswegen werden zwei weitere Klassen namens `PatientDto` und `PhysicianDto` benötigt.

Die Klasse *UserDto* ist eine Oberklasse und die beiden Klassen *PatientDto* und *PhysicianDto* sind Unterklassen. Die beiden Unterklassen können über die Attribute und Operationen der Oberklasse implizit verfügen, d.h. sie können diese erben. In diesem Fall müssen die Attribute und Operationen in den Unterklassen nicht mehr explizit deklariert werden. Im Kontext der UML ist die Rede von einer Generalisierungsbeziehung.

Des Weiteren stellt die Klasse *UserType* einen Aufzählungstyp dar, welcher Benutzer repräsentiert. Innerhalb dieser Klasse wird festgelegt, dass nur Ärzte und Patienten als Benutzer zugelassen sind.

Außerdem können laut den Anforderungen Nachrichten unter beiden Benutzern ausgetauscht werden. Dafür wird eine weitere Domänenklasse namens *MessageDto* benötigt.

Zusammenfassend werden die auftretenden Daten für das Dokumentations- und Überwachungssystem in die folgenden sieben Domänenobjekte kategorisiert (siehe Abbildung 3).

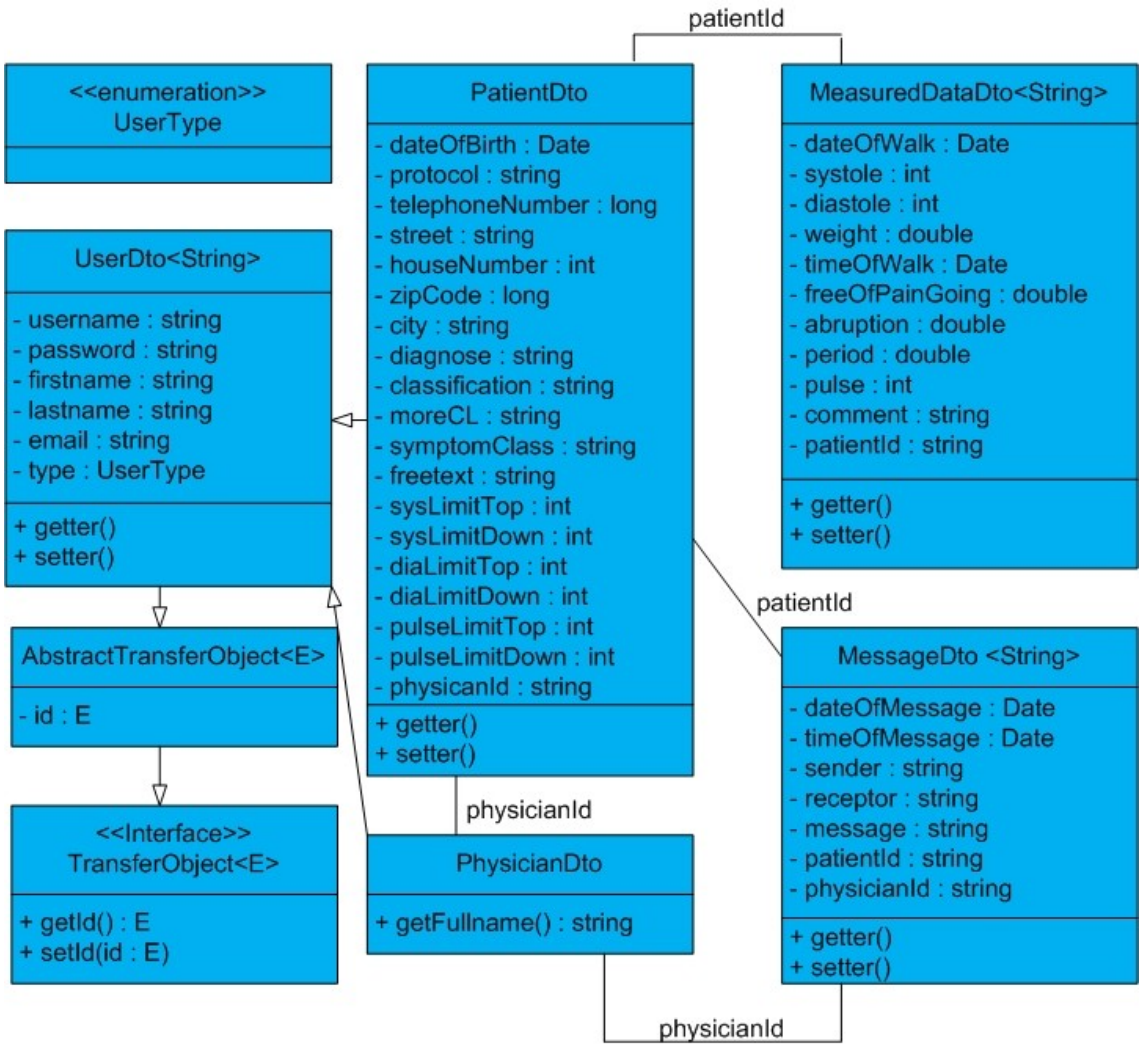


Abbildung 3: Domänenmodell

Alle Domänenobjekte besitzen ein eindeutiges Identifikationsmerkmal, die sogenannte ID. Aus Redundanzgründen wurde ein abstraktes Transferobjekt (`AbstractTransferObject`) eingeführt, welches als einziges Merkmal diese ID besitzt.

Darüber hinaus stellt das Interface `TransferObject` ein Interface für alle DTOs dar. Die Zuordnung von Messwerten zu einem Patienten wird mittels der `patientID` vorgenommen. Außerdem erfolgt die Zuordnung von einem Patienten zu seinem behandelten Arzt über die `physicianID`. Zu guter Letzt erfolgt äquivalent dazu die Zuordnung von Nachrichten zu Ärzten oder Patienten über deren ID.

4.2 Dienstmodell

Bisher wurden ausschließlich die Domänenobjekte sowie deren Beziehungen untereinander ermittelt und in Form eines Domänenmodells festgehalten. Die Funktionalitäten, um damit die in den UseCases spezifizierten Aufgaben erledigen zu können, wurden bislang noch nicht umgesetzt. Aus den Anforderungen ist bekannt, dass Domänenobjekte gespeichert, geändert und ggf. gelöscht werden müssen. Dafür ist ein Bezugsrahmen verpflichtend, der die Verwaltung der Domänenobjekte organisiert. Solch ein Rahmen kann durch ein geeignetes Zusammenspiel von Diensten erreicht werden. Für eine Dienstarchitektur werden mehrere Schnittstellen benötigt, welche die Aufgaben von Datentransferobjekten beschreiben. Aus den UseCases geht hervor, dass einige Methoden, die ein solcher Dienst besitzen muss, für mehrere DTOs erforderlich sind. Aus diesem Grund ist es sinnvoll, diese Methoden möglichst allgemein, in einem generischen Dienst, zu entwerfen. Analog zum abstrakten Transferobjekt können die Methoden der generischen Schnittstelle von anderen Dienstschnittstellen geerbt werden.

Im Fall des Dokumentations- und Überwachungssystems wurde für fast jedes vorhandene Domänenobjekt ein Dienst definiert, siehe Abbildung 4. Einzige Ausnahme bildet in diesem Fall das Domänenobjekt `UserDto`. An dieser Stelle sei erwähnt, dass der Grund dafür mit dem Datenbankschema zusammenhängt. Eine ausführliche Erklärung dazu erfolgt in Kapitel 5.1.[19]

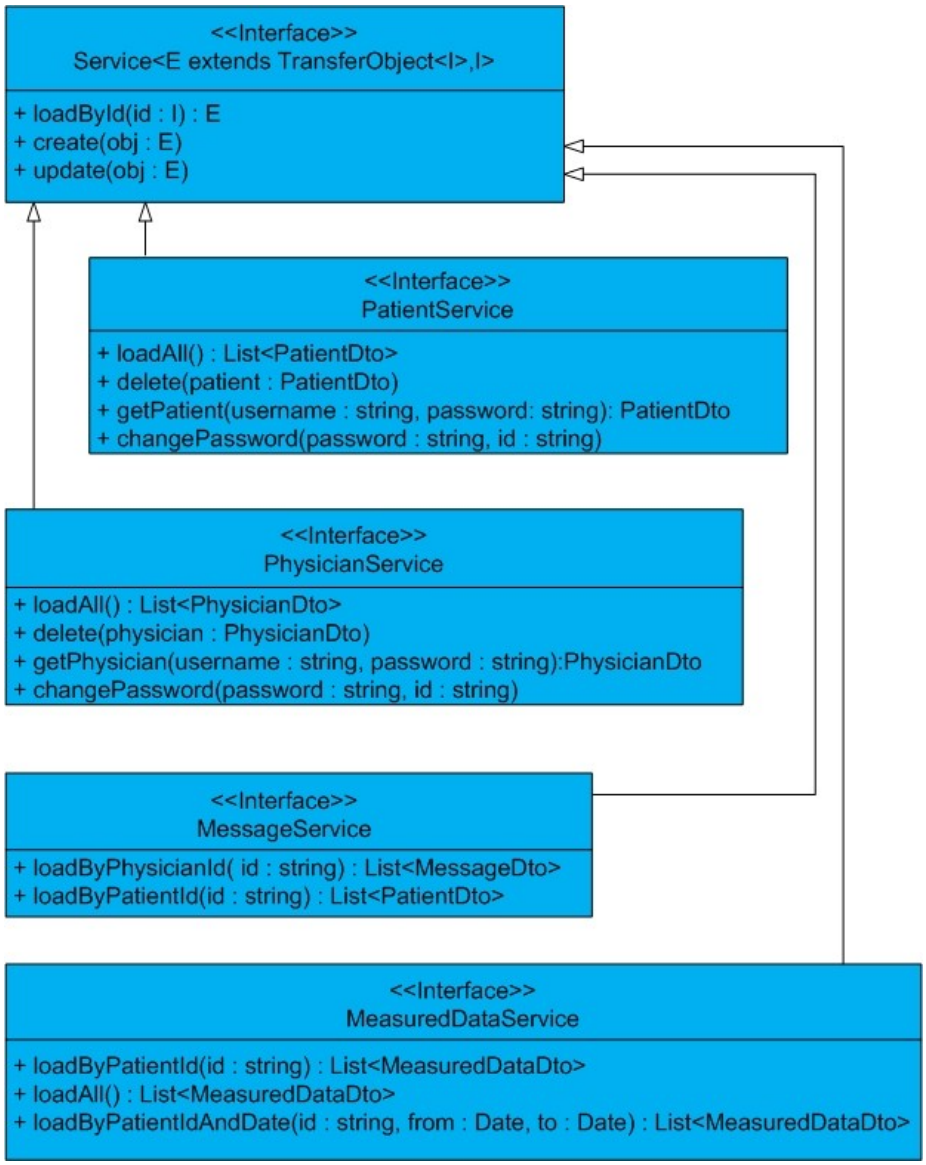


Abbildung 4: Dienstmodell

19 Siehe S. 42.

Das Interface *Service* beinhaltet die Methoden, die alle Domänenobjekte betreffen. Mittels der Methode *loadById()* können Objekte anhand ihrer ID aus der Datenbank geladen werden. Mit den Operationen *create()* und *update()* können Domänenobjekte dauerhaft in der Datenbank gespeichert und bei Bedarf geändert werden.

Das Interface *MeasuredDataService* muss zusätzlich zu den drei Grundfunktionen eine weitere Funktion anbieten, die es ermöglicht, die Messwerte eines bestimmten Patienten anhand dessen ID zu laden. Des Weiteren können ähnlich wie bei der Methode *loadByPatientId()* mittels der Methode *loadByPatientIdAndDate()* die Messwerte eines bestimmten Patienten anhand dessen ID geladen werden, die innerhalb eines gewünschten Zeitintervalls liegen. Mit der Methode *loadAll()* können alle Messwerte geladen werden.

Darüber hinaus muss das Interface *PatientService* zusätzlich eine Methode *delete()* anbieten, mittels derer ein vorhandener Patient vom System entfernt werden kann. Außerdem wird eine Methode namens *loadAll()* benötigt, die das Laden aller vorhandenen Patienten ermöglicht. Hinzu kommt die Methode *getPatient(String username, String password)*, die es gestattet, einen Patienten anhand seines Benutzernamens und Passwortes zu laden. Zusätzlich enthält das Interface *PatientService* die Methode *changePassword()* zum Ändern von Passwörtern.

Äquivalent zum *PatientService* gibt es den *PhysicianService*, welcher analoge Methoden zur Verfügung stellt.

Zu guter Letzt sei der `MessageService` mit den beiden Methoden `loadByPatientId()` und `loadByPhysicianId()` erwähnt, die es gestatten, eine Liste von Nachrichten anhand der jeweiligen ID zu laden. Mit der ID als Parameter wird sichergestellt, dass nur jene Nachrichten geladen werden, die die Person, entweder als Empfänger oder Absender, betreffen.

4.3 Verhaltensdiagramm

Bislang wurden die Domänenobjekte in Form eines Domänenmodells und deren Aufgaben in Form eines Dienstmodells festgehalten. Die Kommunikation zwischen den Objekten wurde bisher jedoch außer Acht gelassen. Es werden Verhaltens- oder auch Sequenzdiagramme für die Darstellung der Kommunikation von Objekten verwendet. Diese Diagramme sind ein Bestandteil der Unified Modeling Language (UML)[20].

Für das Dokumentations- und Überwachungssystem wurden drei Sequenzdiagramme erstellt, die typische erforderliche Szenarien des Systems beschreiben. Welche Objekte an einem Szenario beteiligt sind und welche Informationen diese untereinander austauschen, soll beispielhaft am Szenario „Messwerte eingeben“ gezeigt werden, siehe Abbildung 5.

20 Die Projektseite von UML ist unter http://www.uml.org/ zu finden.

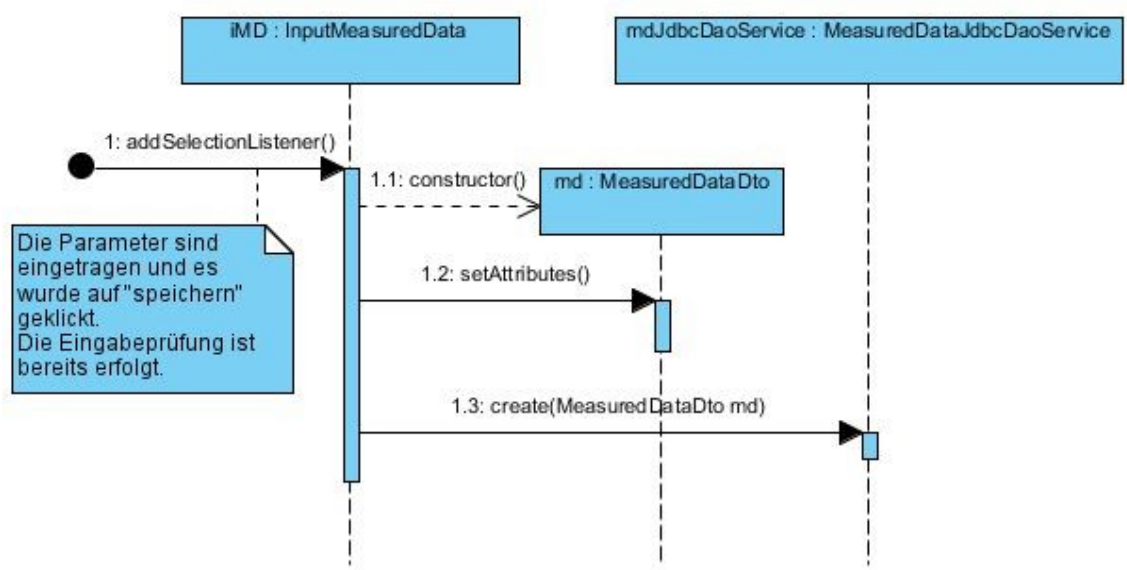


Abbildung 5: Sequenzdiagramm „Messwerte eingeben“

Bei diesem Diagramm haben Patienten bereits ihre Messwerte in das System eingetragen und mit Klick auf den Button `speichern` bestätigt. Des Weiteren erfolgte die Prüfung der Daten auf ihre Korrektheit. Die Methode `addSelectionListener()` fängt Ereignisse ab, die durch den Button ausgelöst werden. Innerhalb dieser Methode wird ein neues Objekt vom Typ `MeasuredDataDto` erzeugt. Im Anschluss daran werden die eingegebenen Daten mit den entsprechenden `setter`-Methoden der Klasse `MeasuredDataDto` gesetzt. Daraufhin erfolgt der Aufruf der Methode `create(MeasuredDataDto md)` aus der Klasse `MeasuredDataDtoJdbcDaoService`, um das eben gesetzte Objekt dauerhaft in der Datenbank zu speichern. Im Erfolgsfall ist das Objekt in der Datenbank gespeichert und kann jederzeit über eine `load()`-Methode geholt und auf der Oberfläche angezeigt werden.

Der Aufbau der anderen beiden Sequenzdiagramme ist ähnlich, so dass auf eine weitere detaillierte Beschreibung verzichtet werden kann.

5. Implementierung

In diesem Kapitel erfolgt die Beschreibung der Implementierung. Inwiefern die Technologien, welche im Grundlagenkapitel erläutert wurden, eingesetzt werden, wird hier beschrieben. Die Vorgehensweise bei der Implementierung ist ähnlich zur Systematik der Konzeption. Jedoch werden zuerst die Panels[21] und Widgets für die Realisierung der Oberfläche erstellt, damit Herrn Prof. Dr. med. Thomas Dengler erste Fortschritte gezeigt werden können. Die Ablauflogik der Oberfläche kann erst implementiert werden, wenn die server-seitige Implementierung abgeschlossen ist. Folglich wird zunächst die Implementierung der Domänenobjekte beschrieben. Im Anschluss daran erfolgt die Implementierung des Dienstmodells und der grafischen Oberfläche. Das Kapitel wird mit einer Beschreibung zum Build-Prozess abgeschlossen.

Die Architektur aus dem Praktikum *Internet-Applikationen* des Wintersemesters 09/10 unter der Leitung von Herrn Prof. Dr. Daniel Pfeifer dient als Referenzarchitektur für das Dokumentations- und Überwachungssystem. Abbildung 6 zeigt die Architektur für das Dokumentations- und Überwachungssystem.

21 Panels sind Bausteine zur Strukturierung der Anwendungsoberfläche.

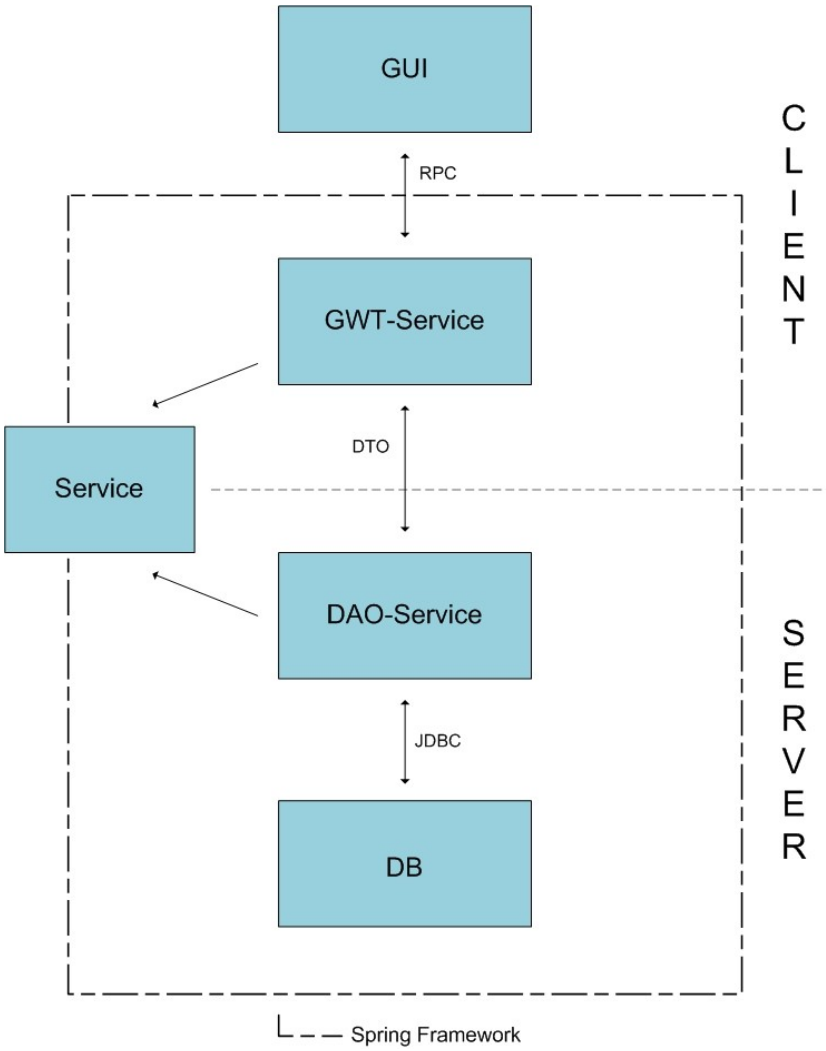


Abbildung 6: Architektur des Dokumentations- und Überwachungssystems

Seit Kapitel 2.2.4 steht fest, dass für die Realisierung der GUI das Google Web Toolkit verwendet wird. Diese Technologie basiert auf dem RPC-Mechanismus, um Daten vom Server nachzuladen. Um einen Remote Procedure Call zu erstellen müssen zwei Java-Interfaces (GWT-Service) und eine Java-Klasse definiert werden. Eines der beiden Interfaces erweitert eine Serviceschnittstelle, die wiederum von einem Datentransferobjekt abhängt. Server-seitig erfolgt der Zugriff auf die Datenbank über Datenzugriffsobjekte, welche ihrerseits wiederum eine der Serviceschnittstellen implementieren.

5.1 Implementierung des Domänenmodells

Jedes Domänenobjekt wird durch eine Java-Klasse repräsentiert, die allesamt im Package `de.walkndoc.server.transferObjects` abgelegt werden. Das gemeinsame Objekt, die ID, wird zunächst in einer Schnittstelle namens `TransferObject` definiert, wie in nachfolgender Abbildung zu sehen ist.

```
public interface TransferObject<E> extends
    Serializable, BeanModelTag {

    // Getter und Setter
    public E getId();

    public void setId(E id);

}
```

Abbildung 7: Schnittstelle für alle DTOs

Der generische Parameter `E` steht dabei für den Typ der ID des Transferobjektes und es bleibt jeder konkreten Domänenklasse selbst überlassen, durch welchen Typ die ID realisiert wird. Das Interface `TransferObject` erweitert die `java.io.Serializable`-Schnittstelle, um Objekte zu serialisieren und damit deren Persistenz zu erreichen. Zusätzlich erweitert das Interface die Schnittstelle `com.extjs.gxt.ui.client.data.BeanModelTag`, um JavaBeans zu identifizieren. Diese werden benötigt, um client-seitig BeanModel-Instanzen erstellen zu können.

Folgende Abbildung zeigt die abstrakte Klasse *AbstractTransferObject*, die die Schnittstelle *TransferObject* implementiert.

```
public abstract class AbstractTransferObject<E>
    implements TransferObject<E> {

    // gemeinsames Attribut zur Identifikation
    private E id;

    // Getter und Setter Methoden zum Attribut Id
    public E getId() {
        return id;
    }

    public void setId(E id) {
        this.id = id;
    }

}
```

Abbildung 8: Implementierung der Basisklasse für alle DTOs

Alle Domänenobjekte, die von *AbstractTransferObject* erben, haben von nun an einen funktionierenden Mechanismus, der ihnen das Speichern einer ID sowie deren Bereitstellung und Festlegung gestattet. Die Auswirkungen der abstrakten Basisklasse auf konkrete Domänenklassen sollen beispielhaft an der Klasse *MeasuredDataDto* gezeigt werden, siehe Abbildung 9.

```
public class MeasuredDataDto extends AbstractTransferObject<String>
    implements TransferObject<String> {

    // Attribute
    private Date dateOfWalk;
    private int systole;
    private int diastole;
    private double weight;
    private Date timeOfWalk;
    private double freeOfPainGoing;
    private double abruption;
    private double period;
    private int pulse;
    private String comment;
    private String patientID;

    // Getter und Setter Methoden zu den Attributen
```

Abbildung 9: Die Klasse MeasuredDataDto

In der *extends*-Klausel ist zu erkennen, dass die Klasse *MeasuredDataDto* von der abstrakten Basisklasse erbt und die ID durch den Typ *String* realisiert wird. Durch diese Angabe existieren in der Klasse *MeasuredDataDto* die beiden Methoden *String getId()* und *setId(String id)*. Darüber hinaus enthält die Klasse weitere Attribute sowie entsprechende *getter-* und *setter*-Methoden dazu.

Ferner gilt es besonders zu erwähnen, dass in der Klasse *UserDto* (siehe Abbildung 10) mittels *enum* Aufzählungstypen festgelegt werden können. Bei der Deklaration des Typs werden alle Werte des Aufzählungstyps definiert. In der Klasse *UserDto* wird somit festgelegt, dass ein Benutzer entweder Arzt oder Patient sein kann.

```
public class UserDto extends AbstractTransferObject<String>
    implements TransferObject<String> {

    public enum UserType {
        PATIENT, PHYSICIAN
    }

    // Attribute
    private String username;
    private String password;
    private String firstName;
    private String lastName;
    private String email;
    private UserType type;
```

Abbildung 10: Die Klasse UserDto

Zusätzlich werden hier die gemeinsamen Attribute sowie deren *getter-* und *setter*-Methoden beider Benutzer festgehalten.

In Abbildung 11 ist zu sehen, dass in der Klasse *PhysicianDto* der *UserType* auf Physician gesetzt wird.

```
public class PhysicianDto extends UserDto {

    /**
     * Konstruktor
     */
    public PhysicianDto() {

        setType(UserType.PHYSICIAN);

    }
```

Abbildung 11: Die Klasse PhysicianDto

Außerdem erweitert diese die Klasse *UserDto* und verfügt somit über deren Attribute und Methoden.

Datenbankmodell

Es ist bekannt, dass Java-Objekte nur zur Laufzeit verwendet werden können. Für das Dokumentations- und Überwachungssystem ist es jedoch Voraussetzung, dass Objekte dauerhaft gespeichert und zu späteren Zeitpunkten bereitgestellt werden müssen. Demnach wird eine Datenbankanbindung benötigt.

Für das Anwendungssystem wird eine PostgreSQL[22]-Datenbank verwendet. PostgreSQL ist ein objektrelationales Datenbankmanagementsystem und unterstützt sowohl die SQL92- als auch die SQL99-Standards.

Da relationale Datenbanksysteme nicht direkt mit den Java-Objekten arbeiten können, müssen diese, zusätzlich zum Domänenmodell, durch ein Schema beschrieben werden. Dabei wird für jedes Domänenobjekt eine Datenbanktabelle, auch Relation genannt, angelegt. Außerdem werden für die Attribute eines Domänenobjektes Spalten innerhalb der Relation angelegt. Weiterhin können Verknüpfungen genutzt werden, hier Fremdschlüsselbeziehungen[23], um Beziehungen zwischen einzelnen Relationen auszudrücken.

22 Die Projektseite von PostgreSQL ist auf http://www.postgresql.org zu finden.

23 Im Schema mit *FK* für *foreign key* gekennzeichnet.

Die folgende Abbildung zeigt das Datenbankschema:

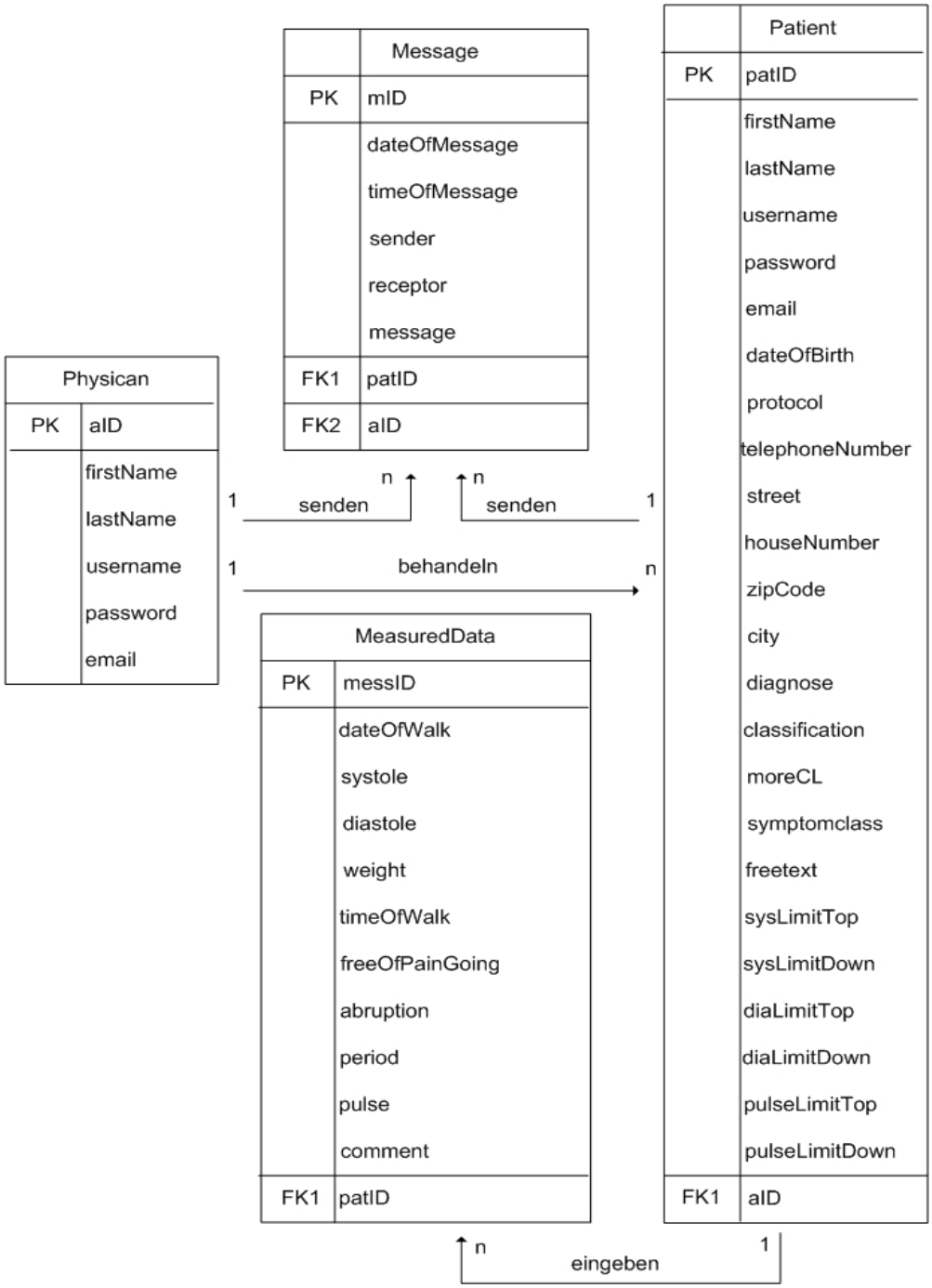


Abbildung 12: Datenbankschema

Jede Relation besitzt einen Primary Key (PK) zur Identifikation. Im Domänenmodell bestand zwischen *UserDto*, *PatientDto* und *PhysicianDto* eine Vererbungsbeziehung. Diese Beziehungen können in der Datenbankwelt auf unterschiedliche Art und Weise abgebildet werden. Im Falle des Dokumentations- und Überwachungssystems wurden nur die Unterklassen *PatientDto* und *PhysicianDto* als Datenbanktabellen realisiert. In diesem Fall werden alle Merkmale der Oberklasse *UserDto* in die Relationen der erbenden Klassen übernommen (siehe Abbildung 12).

Aus dem Domänenmodell ist bekannt, dass zwischen *MeasuredDataDto* und *PatientDto* über die *patientID* eine Verbindung besteht. Es handelt sich dabei um eine 1:n-Beziehung, d.h. ein Patient kann einen oder mehrere Messwerte in das System eingeben. Infolgedessen erhält die Relation *MeasuredData* einen Fremdschlüssel.

Ferner ist bekannt, dass ein Arzt mehrere Patienten behandeln kann und somit existiert eine weitere Fremdschlüsselbeziehung, bei der die Relation *Patient* den Fremdschlüssel *aID* bekommt.

Darüber hinaus können Ärzte und Patienten beliebig viele Nachrichten untereinander austauschen. Aufgrund dessen erhält die Relation *Message* die beiden Fremdschlüssel *aID* und *patID*.

5.2 Implementierung des Dienstmodells

Wie bereits bei den Domänenklassen gesehen, gibt es hier ebenfalls eine generische Schnittstelle (siehe Abbildung 13), welche Methoden umfasst, die von jeder weiteren konkreten Serviceschnittstelle geerbt werden können.

```
public interface Service<E extends TransferObject<I>, I> {

    public E loadById(I id);

    public void create(E object);

    public void update(E object);

}
```

Abbildung 13: Das Interface Service

Jeder Service hängt dabei von den beiden Parametern `E` und `I` ab und befindet sich im Package `de.walkndoc.server.service`. Der Parameter `E`, wie bereits aus Kapitel 5.1. bekannt, steht für das Transferobjekt und `I` für den Typ der ID. Die Auswirkungen der Basisserviceschnittstelle auf konkrete Dienstschnittstellen sollen beispielhaft am Interface `MeasuredDataService` erläutert werden.

Folgende Abbildung zeigt das Interface *MeasuredDataService*.

```
public interface MeasuredDataService extends
    Service<MeasuredDataDto, String> {

    public List<MeasuredDataDto> loadByPatientId(String id);

    public List<MeasuredDataDto> loadAll();

    public List<MeasuredDataDto> loadByPatientIdAndDate
        (String id, Date from, Date to);

}
```

Abbildung 14: Das Interface MeasuredDataService

Die Schnittstelle *MeasuredDataService* erweitert die grundlegende Dienstschnittstelle *Service* und somit stehen dieser die Methoden *create(MeasuredDataDto object)*, *update(MeasuredDataDto object)* sowie *MeasuredDataDto loadById(String id)* zur Verfügung. Darüber hinaus besitzt die Schnittstelle weitere Methoden, um alle Messwerte, patientenindividuelle Messwerte oder die Messwerte eines Patienten, die innerhalb eines angegebenen Zeitintervalls liegen, zu laden.

Analog dazu werden die Dienstschnittstellen für die anderen Domänenobjekte formuliert. Der Aufbau der anderen Dienstschnittstellen unterscheidet sich nicht nennenswert zu dieser, so dass auf eine weitere Beschreibung verzichtet werden kann. Nach der Schnittstellendefinition erfolgt eine konkrete Implementierung dieser.

Implementierung der Datenzugriffsobjekte

Für jede Domänenklasse muss ein Datenzugriffsobjekt definiert werden, wobei im Fall des Dokumentations- und Überwachungssystems für das *UserDto* kein Datenzugriffsobjekt definiert wird, sondern für die beiden Unterklassen *PatientDto* und *PhysicianDto*, da nur diese als Datenbanktabellen realisiert wurden. Anhand des MeasuredDataJdbcDaoService sollen im nachfolgenden Text die Mechanismen erklärt werden, die notwendig sind, um mittels Spring und JDBC auf eine Datenbank zugreifen zu können.

```
@Transactional
public class MeasuredDataJdbcDaoService extends
    SimpleJdbcDaoSupport implements
        MeasuredDataService {
```

Abbildung 15: Data Access Object für MeasuredDataDto

In Abbildung 15 fällt zunächst die Annotation *@Transactional* auf, die durch Spring bereitgestellt wird und den Service als transaktionsfähig markiert. Eine Transaktion ist als eine Folge von Operationen, die als eine logische Einheit betrachtet wird, zu verstehen. Diese Operationen überführen die Datenbank von einem konsistenten Zustand in einen neuen, konsistenten Zustand, wobei das Atomicity-Consistency-Isolation-Durability-(ACID)-Prinzip beachtet werden muss [Saake 2005, S. 499]. Für weitere Informationen zu den ACID-Eigenschaften sei der Leser auf das Fachbuch (Saake et al., 2005) Kapitel 8 verwiesen.

Darüber hinaus wird vor jeder Methode innerhalb der Klasse *MeasuredDataJdbcDaoService* eine weitere *@Transactional*-Annotation mit dem Zusatz, ob nur gelesen werden darf oder nicht, vorgenommen. Zusätzlich zum Marker *@Transactional* muss dem Framework die Transaktionsfähigkeit in einer Konfigurationsdatei mitgeteilt werden [Zeitner 2008, S. 182].

Des Weiteren implementiert die Klasse *MeasuredDataJdbcDaoService* die Dienstschnittstelle *MeasuredDataService* und somit stehen dieser Klasse wiederum die Methodensignaturen *create(MeasuredDataDto object)*, *update(MeasuredDataDto object)* und *MeasuredDataDto loadById(String id)* zur Verfügung, welche hier letztendlich umgesetzt werden. Für deren Umsetzung wird das durch Spring bereitgestellte Interface *org.springframework.jdbc.core.simple.SimpleJdbcDaoSupport* benötigt. Dieses Interface stellt ein *SimpleJdbcTemplate* zur Verfügung, welches alle gängigen Abfragen als Template abbildet [Zeitner 2008, S. 147].

Für die Umsetzung der Standardoperationen muss der DAO-Klasse zunächst mitgeteilt werden, auf welcher Datenbanktabelle operiert werden soll. Darüber hinaus müssen Konstanten für die Operationen angelegt werden, die entsprechende SQL-Anweisungen enthalten. Abbildung 16 zeigt dies beispielhaft an der Methode *create(MeasuredDataDto object)*.

```
private static final String TABLE = "measuredData";

// Konstante für create
private static final String INSERT = "insert into "
        + TABLE
        + " (messID, dateOfWalk, systole, diastole, " +
                "weight, timeOfWalk, freeOfPainGoing, " +
                "abruption, period, pulse, comment, " +
                "patID)"
        + "values (:messID, :dateOfWalk, :systole, " +
                ":diastole," +
                " :weight, :timeOfWalk, :freeOfPainGoing," +
                " :abruption," +
                " :period, :pulse, :comment, :patID)";

@Transactional(readOnly = false)
public void create(MeasuredDataDto object) {

    String id = UUID.randomUUID().toString();
    object.setId(id);

    Map<String, Object> parameters =
        new HashMap<String, Object>();

    parameters.put("messID", object.getId());
    parameters.put("dateOfWalk", object.getDateOfWalk());
    parameters.put("systole", object.getSystole());
    parameters.put("diastole", object.getDiastole());
    parameters.put("weight", object.getWeight());
    parameters.put("timeOfWalk", object.getTimeOfWalk());
    parameters.put("freeOfPainGoing",
            object.getFreeOfPainGoing());
    parameters.put("abruption", object.getAbruption());
    parameters.put("period", object.getPeriod());
    parameters.put("pulse", object.getPulse());
    parameters.put("comment", object.getComment());
    parameters.put("patID", object.getPatientID());

    getSimpleJdbcTemplate().update(INSERT, parameters);

}
```

Abbildung 16: Anlegen eines Datensatzes

Innerhalb der *create(MeasuredDataDto object)*-Methode wird zunächst der primäre Schlüssel mittels *randomUUID* erzeugt und direkt im Objekt abgelegt. Dannach wird eine *HashMap* erzeugt, die für das Setzen der jeweiligen Parameter benötigt wird.

Der eigentliche Aufruf erfolgt mittels *getSimpleJdbcTemplate().update(INSERT, parameters)*. *Update* ist dabei eine Methode aus dem *SimpleJdbcTemplate* und wird für Abfragen zum Anlegen, Ändern und Löschen verwendet. Die Methode *update(MeasuredDataDto object)* aus der Klasse *MeasuredDataJdbcDaoService* ist im Wesentlichen ähnlich aufgebaut, aber mit dem Unterschied, dass hier keine ID-Generierung mehr benötigt wird und im Aufruf von *getSimpleJdbcTemplate().update(UPDATE, parameters)* die Konstante *UPDATE* anstatt *INSERT* verwendet wird, die in Abbildung 17 definiert ist.

```
// Konstante für update
private static final String UPDATE = "update "
        + TABLE
        + " set dateOfWalk = :dateOfWalk, systole = :systole," +
                " diastole = :diastole, weight = :weight, " +
                "timeOfWalk = :timeOfWalk, " +
                "freeOfPainGoing = :freeOfPainGoing, " +
                "abruption = :abruption, period = :period, " +
                "pulse = :pulse, comment = :comment "
        + "where messID = :messID";
```

Abbildung 17: SQL-Anweisung zum Ändern eines Datensatzes

Besonders beim Ändern eines Datensatzes müssen viele Spalten geändert werden. Infolgedessen empfiehlt es sich, in der SQL-Anweisung benannte Parameter zu verwenden. Bei Spring ist es jedoch auch erlaubt für Platzhalter >>?<< anstatt Namen zu verwenden. Allerdings bietet sich das aus leserlichen Gründen vor allem bei einem update-Statement nicht an. In Abbildung 17 werden die benannten Parameter durch einen Doppelpunkt eingeführt. Dies ist eine Konvention, die im Datenbankbereich sehr weit verbreitet ist [Zeitner 2008, S. 150, 151].

Eine weitere Methode aus dem *SimpleJdbcTemplate* ist *query()*, welche für Abfragen aller Art verwendet wird. Diese Methode wird bei *loadByPatientId()* benötigt, siehe Abbildung 18.

```
// Konstante für loadByPatientId
private static final String SELECT_BY_PATIENT_ID = "select * from "
    + TABLE
    + " where patID = ? order by dateOfWalk ASC";

@Transactional(readOnly = true)
public List<MeasuredDataDto> loadByPatientId(String id) {

    return getSimpleJdbcTemplate().query(SELECT_BY_PATIENT_ID,
            new MeasuredDataRowMapper(), id);
}
```

Abbildung 18: Die Methode loadByPatientId()

Innerhalb der SQL-Anweisung wird zum ersten Mal eine SELECT-Abfrage verwendet, welche in JDBC ein *ResultSet* zurück liefert. Dabei muss jede Zeile aus dem *ResultSet* auf ein Objekt vom Typ *MeasuredDataDto* abgebildet werden. Für das sogenannte Mapping wird eine weitere Hilfsklasse von Spring benötigt, die diesen Vorgang unterstützt. Dafür muss zunächst das Interface `org.springframework.jdbc.core.RowMapper` implementiert werden, welches die Methodensignatur `MeasuredDataDto mapRow(ResultSet rs, int rowNum)` bereitstellt [Zeitner 2008, S. 148].

Nachstehende Abbildung zeigt die Klasse *MeasuredDataRowMapper*, welche sich innerhalb der Klasse *MeasuredDataJdbcDaoService* befindet.

```
private static class MeasuredDataRowMapper implements
        RowMapper<MeasuredDataDto> {

    public MeasuredDataDto mapRow(ResultSet rs, int rowNum)
            throws SQLException {

        MeasuredDataDto md = new MeasuredDataDto();

        md.setId(rs.getString("messID"));
        md.setDateOfWalk(rs.getDate("dateOfWalk"));
        md.setSystole(rs.getInt("systole"));
        md.setDiastole(rs.getInt("diastole"));
        md.setWeight(rs.getDouble("weight"));
        md.setTimeOfWalk(rs.getTime("timeOfWalk"));
        md.setFreeOfPainGoing(rs.getDouble("freeOfPainGoing"));
        md.setAbruption(rs.getDouble("abruption"));
        md.setPeriod(rs.getDouble("period"));
        md.setPulse(rs.getInt("pulse"));
        md.setComment(rs.getString("comment"));
        md.setPatientID(rs.getString("patID"));

        return md;
    }

}
```

Abbildung 19: Die Klasse MeasuredDataRowMapper

Innerhalb dieser Methode muss zuerst ein Objekt vom Typ *MeasuredDataDto* erzeugt werden. Mit den *get*-Methoden aus dem *ResultSet* können die Ergebnisse abgefragt und direkt über die *setter*-Methoden der Klasse *MeasuredDataDto* gesetzt werden.

Die Klasse *MeasuredDataJdbcDaoService* besitzt weitere Methoden, deren Aufbau sich jedoch nicht nennenswert zu den bereits vorgestellten Methoden unterscheidet. Analog dazu werden die Datenzugriffsobjekte für die anderen Domänenklassen erstellt, wobei jedes dieser Objekte in einer spring-eigenen XML-Konfigurationsdatei, dem *ApplicationContext*, bekannt gemacht werden muss.

ApplicationContext

Es muss ein Tag namens `bean` definiert werden, welches durch den Parameter `id` eindeutig den Namen der Bean, unter dem sie von nun an ansprechbar ist, festlegt. Des Weiteren fixiert der Parameter `class`, wo die Klasse zu finden ist, siehe Abbildung 20.

```
<!-- Datenzugriffsobjekte -->
<bean id="measuredDataJdbcDaoService"
    class="de.walkndoc.server.dataAcessObjects.
    MeasuredDataJdbcDaoService"
    autowire="byName"/>
<bean id="patientJdbcDaoService"
    class="de.walkndoc.server.dataAcessObjects.
    PatientJdbcDaoService"
    autowire="byName"/>
<bean id="physicanJdbcDaoService"
    class="de.walkndoc.server.dataAcessObjects.
    PhysicanJdbcDaoService"
    autowire="byName"/>
<bean id="messageJdbcDaoService"
    class="de.walkndoc.server.dataAcessObjects.
    MessageJdbcDaoService"
    autowire="byName"/>
```

Abbildung 20: Bekanntmachung der DAOs im ApplicationContext

Spring unterstützt XML-Autowiring, um Beans automatisch miteinander zu verknüpfen. Dabei erfolgt die Verknüpfung der Beans über deren Namen und wird in der Spring-Konfiguration mit dem Tag `autowire` festgehalten [Zeitner 2008, S. 67f.]. Bislang wurde innerhalb der Datenzugriffsobjekte von `SimpleJdbcDaoSupport` abgeleitet, wobei dem `ApplicationContext` bisher nicht mitgeteilt wurde, dass er dieses Template zu verwenden hat. Dafür muss eine Bean mit dem Namen `simpleJdbcTemplate` angelegt werden, siehe Abbildung 21.

```
<!-- JDBC Template fuer Datenzugriffsobjekte -->
<bean id="simpleJdbcTemplate"
    class="org.springframework.jdbc.core.simple.SimpleJdbcTemplate">
    <constructor-arg ref="dataSource"></constructor-arg>
</bean>
```

Abbildung 21: Angabe des JDBC-Template im ApplicationContext

Ebenso weiß das Template nicht, gegen welche Datenbank die Abfragen laufen sollen. Dazu referenziert die Bean `simpleJdbcTemplate` innerhalb ihres `constructor-arg`-Tags die Schnittstelle `DataSource`.

```
<bean id="dataSource"
    class="org.apache.commons.dbcp.BasicDataSource"
    dependency-check="none" destroy-method="close">
    <property name="driverClassName"
        value="org.postgresql.Driver"></property>
    <property name="url"
        value="jdbc:postgresql:WalkNDoc"></property>
    <property name="username"
        value="salang"></property>
    <property name="password"
        value="salang"></property>
</bean>
```

Abbildung 22: Angabe der DataSource im ApplicationContext

In Abbildung 22 wird die `BasicDataSource` aus dem „Apache Jakarta Commons“[24]-Projekt verwendet. Diese Schnittstelle übernimmt die Verwaltung der Datenbankverbindungen in einem Pool [Zeitner 2008, S. 156]. Mit `dependency-check="none"` wird die Abhängigkeitsprüfung für Beans deaktiviert. Außerdem können Beans mit der Definition einer Destroy-Methode, in Abbildung 22 mit `destroy-method="close"` angegeben, zerstört werden, die jedoch nur für Beans, welche als Singleton[25] verwaltet werden, gilt. Vorteilhaft ist, dass Spring standardmäßig alle definierten Beans als Singletons verwaltet. Die Angaben zur Datenbank werden innerhalb der `property`-Tags notiert.

24 Die Projektseite von Apache Commons ist unter http://commons.apache.org zu finden.

25 Ein Entwurfsmuster, das verhindert, dass von einer Klasse mehrere Objekte erzeugt werden.

Innerhalb der Bean-Klassen erlaubt Spring die Verwendung von verschiedenen Annotationen. Bereits in Kapitel 5.2 wurde die Annotation `@Transactional` für die Transaktionsverwaltung eingesetzt. Dort wurde erwähnt, dass zusätzlich zu dieser Annotation die Transaktionsfähigkeit in einer Konfigurationsdatei bekannt gemacht werden muss.[26] Abbildung 23 zeigt, wie ein Transaktionsmanager innerhalb des `ApplicationContexts` festgelegt wird.

```
<context:annotation-config/>
<tx:annotation-driven
    transaction-manager="transactionManager"/>

<!-- Transaktionsmanager -->
<bean id="transactionManager"
    class="org.springframework.jdbc.datasource.
    DataSourceTransactionManager">
    <constructor-arg ref="dataSource"></constructor-arg>
</bean>
```

Abbildung 23: Festlegung des Transaktionsmanagers

Um allgemein Annotationen verwenden zu können, muss zunächst im `ApplicationContext`, wie in Abbildung 23 zu sehen, ein Tag mit dem Namen `context` angelegt werden. Darüber hinaus wird eine Bean namens `transactionManager` festgelegt, die die Schnittstelle `DataSource` referenziert [Zeitner 2008, S. 159].

Versenden von E-Mails mit Hilfe der JavaMail API und Spring

Das Dokumentations- und Überwachungssystem unterstützt den Versand einer E-Mail, wenn gewisse Parameter einen der zuvor festgelegten Grenzwerte überschritten haben. Dazu wird die spring-spezifische Schnittstelle `MailSender` benötigt, welche eine konkrete Implementierung (`JavaMailSenderImpl`) erfordert.

26 Siehe S. 46.

Der Abbildung 24 kann entnommen werden, dass innerhalb der Bean `mailSender` notwendige Einstellungen, wie beispielsweise der Host- und Benutzername, festgehalten werden.

```
<bean id="mailSender"
    class="org.springframework.mail.
    javamail.JavaMailSenderImpl">
    <property name="host"
        value="${mail.host}"></property>
    <property name="username"
        value="${mail.username}"></property>
    <property name="password"
        value="${mail.password}"></property>
    <property name="javaMailProperties">
        <props>
            <prop key="mail.smtp.auth">true</prop>
            <prop key="mail.smtp.starttls.enable">true</prop>
            <prop key="mail.from">${mail.from}</prop>
            <prop key="mail.to">${mail.to}</prop>
        </props>
    </property>
</bean>
```

Abbildung 24: Definition der Bean mailSender im ApplicationContext

Falls der Mailserver eine SMTP-Authentifizierung erfordert, ist innerhalb der `JavaMailProperties` ein Parameter zu setzen [Zeitner 2008, S. 333]. Weitere Parameter werden für den Empfänger und Sender gesetzt, die jedoch als Variablen in der Konfigurationsdatei verwendet werden. Um allgemein Variablen im `ApplicationContext` verwenden zu können, bietet Spring „die Klasse `PropertyPlaceholderConfigurer` an, die es erlaubt, externe Properties-Dateien einzulesen und direkt in der Konfiguration zu verwenden.“ [Zeitner 2008, S. 47]

Dafür muss innerhalb des XML-Namespace *context* ein *Property-Placeholder* definiert werden, siehe Abbildung 25.

```
<context:property-placeholder
    location="/WEB-INF/mail.properties"/>
```

Abbildung 25: Definition des Property-Placeholder

Ferner muss eine Textdatei mit der Endung *.properties* angelegt werden, die die Properties definiert, welche dann in der Konfigurationsdatei mit der gängigen Schreibweise *${property-Name}* verwendet werden können [Zeitner 2008, S. 48]. Abbildung 26 zeigt die Textdatei *Mail.properties*, die unter anderem die Sender- und Empfängerdaten enthält.

```
mail.host = mailhost.rz.hs-heilbronn.de
mail.from = no-answer@hs-heilbronn.de
mail.to = salang86@googlemail.com
```

Abbildung 26: Die Textdatei mail.properties

Zusätzlich wird eine Bean namens *mailGwtServiceImpl* definiert, die eine Instanz der Klasse *MailGwtServiceImpl* erstellt. Dieser wird, wie in Abbildung 27 zu sehen, die Bean *mailSender* zur weiteren Verwendung übergeben.

```
<bean id="mailGwtServiceImpl"
    class="de.walkndoc.server.MailGwtServiceImpl">
    <property name="mailSender" ref="mailSender"></property>
</bean>
```

Abbildung 27: Definition der Bean mailGwtServiceImpl

Die Klasse *MailGwtServiceImpl* enthält die Methode *sendMail(String subject, String text)*, um E-Mails zu versenden. Im Inneren dieser Methode werden zunächst die Beans *mailSender* und *mailGwtServiceImpl* aus dem *ApplicationContext* geholt. Darüber hinaus wird eine Instanz der Klasse *SimpleMailMessage*, welche alle relevanten Informationen zu einer E-Mail enthält, erzeugt. Im Anschluss daran erfolgt das Setzen der Informationen, wobei der Sender und Empfänger aus dem *ApplicationContext* mittels *getJavaMailProperties().getProperty()* geholt wird. Zu guter Letzt kann mit dem Aufruf der Methode *send(simpleMailMessage)* die E-Mail versendet werden. All diese Vorgänge sind in Abbildung 28 dargestellt.

```
public void sendMail(String subject, String text) {

    // hole Bean über Spring
    context = new FileSystemXmlApplicationContext
        ("WEB-INF/application-context.xml");
    sender = (JavaMailSenderImpl)
        context.getBean("mailSender");
    MailGwtServiceImpl mailImpl = (MailGwtServiceImpl)
        context.getBean("mailGwtServiceImpl");

    // speichert alle relaventen Infos über eine e-mail
    SimpleMailMessage simpleMailMessage = new SimpleMailMessage();

    simpleMailMessage.setFrom(sender.getJavaMailProperties()
            .getProperty("mail.from"));
    simpleMailMessage.setTo(sender.getJavaMailProperties()
            .getProperty("mail.to"));
    simpleMailMessage.setSubject(subject);
    simpleMailMessage.setText(text);

    try {

        mailImpl.mailSender.send(simpleMailMessage);

    } catch(MailException ex) {

    }

}
```

Abbildung 28: Implementierung der Methode sendMail()

Nachdem nun server-seitig alles implementiert ist, kann mit der Umsetzung der grafischen Oberfläche begonnen werden. Zuvor muss jedoch die Integration von Spring mit GWT realisiert werden. Früher erfolgte die Anbindung zum Einen über das *DispatcherServlet*, welches im Deployment Descriptor definiert werden musste und zum Anderen über den *GWTSpringController*. Seit Spring 3.0.3 und GWT 2.0.4 bietet Spring für die Anbindung die Klasse *ContextLoaderListener* an, welche dem Framework mitteilt, wo sich der *ApplicationContext* innerhalb einer Webanwendung befindet [Zeitner 2008, S. 63].

Folgende Abbildung zeigt die Spring-Konfiguration im Deployment Descriptor.

```
<context-param>
  <param-name>contextConfigLocation</param-name>
  <param-value>/WEB-INF/application-context.xml</param-value>
</context-param>

<listener>
  <listener-class>org.springframework.web.context.
      ContextLoaderListener</listener-class>
</listener>
```

Abbildung 29: Spring-Konfiguration in der web.xml

Dabei wird festgelegt, dass die Datei *application-context.xml* den *ApplicationContext* repräsentiert und sich direkt im Unterverzeichnis *WEB-INF* befindet.

5.3 Implementierung der GUI

Jedes GWT-Projekt enthält eine Modul-Datei namens `[Name der Anwendung].gwt.xml`. Die Modul-Datei enthält einen „Satz von Konfigurationsdaten, die sich auf ein bestimmtes GWT-Projekt beziehen." [Hanson 2007, S. 296] Sie beinhaltet unter anderem Verweise auf Stylesheets und Eintrittspunkte. Darüber hinaus können weitere Module, beispielsweise von Drittanbietern, eingebunden werden [Hanson 2007, S. 296]. Abbildung 30 zeigt die Modul-Datei `WalkNDoc.gwt.xml` für das Dokumentations- und Überwachungssystem.

```
<module rename-to='walkndoc'>

    <inherits name='com.google.gwt.user.User' />
    <inherits name='com.google.gwt.user.theme.standard.Standard' />
    <inherits name='com.extjs.gxt.ui.GXT' />
    <inherits name='com.extjs.gxt.charts.Chart' />
    <inherits name='com.google.gwt.i18n.I18N' />
    <inherits name='com.google.gwt.junit.JUnit' />

    <entry-point class='de.walkndoc.client.both.WalkNDoc' />

    <source path='client' />
    <source path='server/service' />
    <source path='server/transferObjects' />

</module>
```

Abbildung 30: Die GWT-Modul-Datei WalkNDoc.gwt.xml

Durch den Tag `inherits` können allgemein Module in die Anwendung eingebunden werden. Zunächst wird vom Benutzermodul `user`, welches Kernfunktionalitäten beinhaltet, geerbt.

Aus den UseCases ist ersichtlich, dass das System Messwerte grafisch darstellen können muss. Aus diesem Grund wird das Modul `Chart` benötigt, welches von dem Drittanbietermodul `GXT` bereitgestellt wird. Im nächsten Schritt wird mittels `entry-point` der Eintrittspunkt festgelegt. Die `WalkNDoc.gwt.xml` ist im Verzeichnis `de.walkndoc` abgelegt, wobei der Compiler den Quellcode im Verzeichnis `de.walkndoc.client` erwartet. Gelegentlich werden weitere Speicherplätze benötigt. In diesem Fall können Quellpfade durch den Tag `source path` festgelegt werden [Hanson 2007, S. 302]. Im `client`-Package befindet sich unter anderem die Klasse `WalkNDoc`, die als Eintrittspunkt fungiert. Diese Klasse muss die Schnittstelle `EntryPoint` implementieren, welche wiederum die Implementierung der Methode `onModuleLoad()` erfordert. `OnModuleLoad()` erfüllt den gleichen Zweck wie beispielsweise die `main()`-Methode in einer Java-Anwendung.

Abbildung 31 demonstriert die Eintrittsklasse `WalkNDoc`.

```
public class WalkNDoc implements EntryPoint {

    public void onModuleLoad() {

    }
}
```

Abbildung 31: Eintrittspunkt der Anwendung

RPC-Mechanismus

Bereits zu Beginn des fünften Kapitels wurde kurz erwähnt, dass GWT auf dem RPC-Mechanismus basiert.[27] Dieser Mechanismus gestattet das Senden und Empfangen echter Java-Objekte, sowohl client- als auch server-seitig.

Der RPC-Mechanismus setzt sich aus folgenden drei Teilen zusammen [Hanson 2007, S. 324]:

- den Datenobjekten, die zwischen Client und Server ausgetauscht werden
- dem Dienst, der auf dem Server läuft
- dem Client, welcher den Dienst aufruft

Die Kommunikation geht dabei immer vom Client aus, indem er ein von GWT bereitgestelltes Proxy-Objekt übergibt. Dieses Objekt serialisiert die Anforderung als Textdatenstrom und leitet diesen an den Server weiter. Dort wird die Anforderung von einem GWT-spezifischen Java-Servlet empfangen, deserialisiert und an den entsprechenden Dienst weitergeleitet. Sobald dieser Dienst „einen Wert an das GWT-Servlet zurückgibt, wird das resultierende Objekt serialisiert und wieder zum Client zurückgeschickt.“ [Hanson 2007, S. 324] Daraufhin wird die Antwort vom Proxy-Objekt client-seitig empfangen, welches die Daten wiederum in ein Java-Objekt deserialisiert und an den aufrufenden Code zurückgibt [Hanson 2007, S. 324].

27 Siehe S. 35.

Der RPC-Mechanismus ist an sich relativ simpel, doch aufgrund vieler Details kann schnell der Überblick verloren gehen. Aus diesem Grund soll vorweg die nachstehende Abbildung den Mechanismus am Beispiel des `AccessService` anschaulich darstellen. Der `AccessService` wird zum Ein- und Ausloggen von Benutzern benötigt.

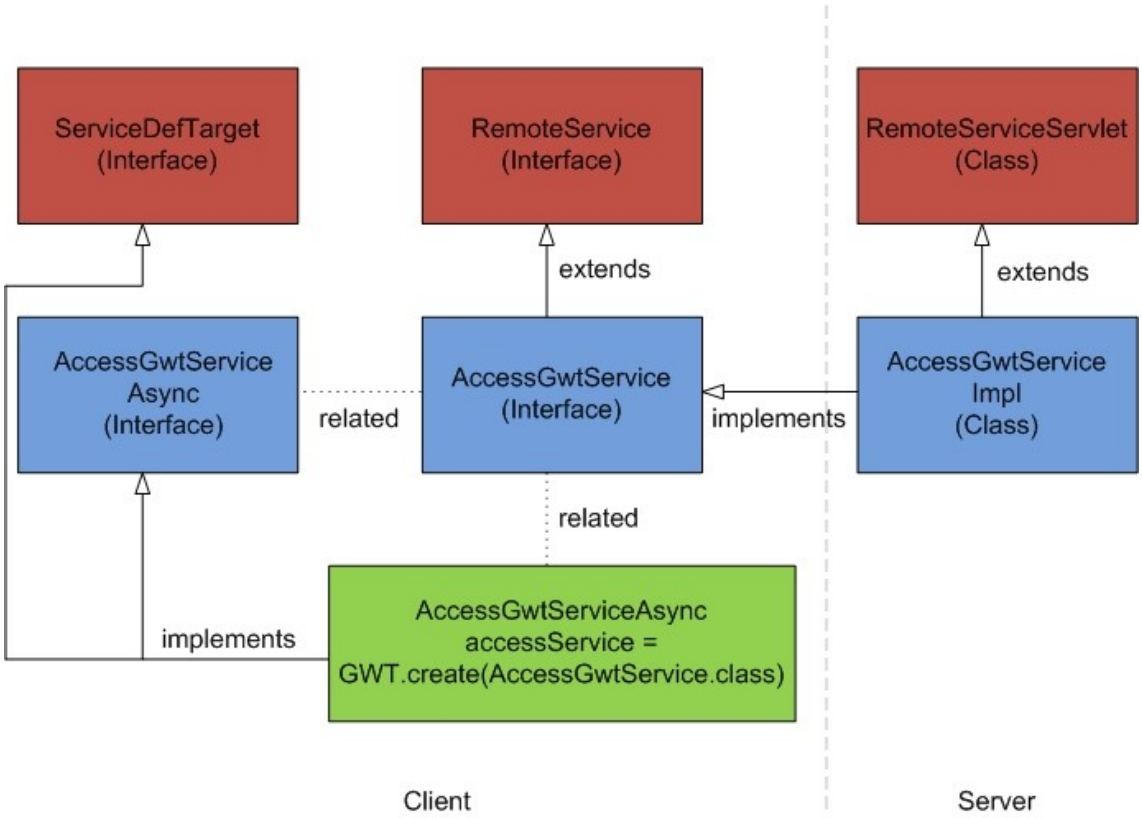


Abbildung 32: Der Aufbau des RPC-Mechanismus [Google Code]

Um den RPC-Mechanismus zu ermöglichen, müssen zunächst die Datenobjekte bereitgestellt werden, was bereits in Kapitel 5.1 geschah. GWT kann im client-seitigen Code nur die Verwendung bestimmter Java-Klassen bewilligen. Dazu zählen unter anderem primitive Java-Typen und Java-Wrapper, sowie Teile der Java Runtime Environment (JRE)-Objekte und Klassen, die die GWT-spezifische Schnittstelle `IsSerializable` implementieren. Jedoch besteht seit der Version 1.4 von GWT die Möglichkeit, serialisierbaren GWT-Klassen die Implementierung der Schnittstelle `java.io.Serializable` zu gestatten [Hanson 2007, S. 331]. Darüber hinaus wird für die Verwendung des RPC-Mechanismus sowohl eine Dienstdefinition als auch eine Dienstimplementierung benötigt. In den nachfolgenden Abbildungen wird das Prinzip des RPC-Mechanismus näher erläutert. Um einen Dienst definieren zu können, muss zunächst ein Java-Interface erstellt werden, welches die GWT-Schnittstelle `com.google.gwt.user.client.rpc.RemoteService` erweitert, siehe Abbildung 33.

```
@RemoteServiceRelativePath("accessService")
public interface AccessGwtService extends RemoteService {

    public UserDto login(String username, String password);

    public boolean logout();

}
```

Abbildung 33: Definition eines RPC-Dienstes

Die Methode `login(String username, String password)` wird aufgerufen, wenn sich ein Benutzer am System anmelden möchte. Diese gibt ein `UserDto` zurück, wobei in der entsprechenden Client-Klasse unterschieden werden muss, ob sich ein Arzt oder ein Patient einloggt.

Je nach Berechtigung wird dann die entsprechende Oberfläche angezeigt. Zum Abmelden wird die Methode *logout()* benötigt, welche im Erfolgsfall *true* zurück liefert. Es wird vorausgesetzt, dass sich diese Schnittstelle im *client*-Package befindet, da diese vom GWT-Compiler in JavaScript übersetzt werden muss. Die Bedeutung der Annotation *@RemoteServiceRelativePath* wird später, beim konkreten Aufruf einer entfernten Methode, erklärt.[28] Bevor ein entfernter Methodenaufruf vom Client aus durchgeführt werden kann, muss dazu eine weitere Schnittstelle, das sogenannte asynchrone Interface, erstellt werden. Dieses Interface sorgt für die asynchrone Kommunikation und basiert auf der originalen Dienstschnittstelle. Der Name des Interfaces ist explizit an den Namen der Dienstschnittstelle gebunden, jedoch wird dem Bezeichner das Token *Async* nachgestellt, wie nachstehende Abbildung demonstriert.

```
public interface AccessGwtServiceAsync {

    public void login(String username,
            String password,
            AsyncCallback<UserDto> callback);

    public void logout(AsyncCallback<Boolean>
        callback);

}
```

Abbildung 34: Asynchrones Interface zu AccessGwtService

Die Schnittstelle in Abbildung 34 enthält dieselben Methoden wie die Dienstschnittstelle, allerdings werden sie um den Parameter vom Typ `com.google.gwt.user.client.rpc.AsyncCallback` ergänzt. Die Schnittstelle *AsyncCallback* stellt zwei Methoden zur Verfügung. *onSuccess()* wird aufgerufen, wenn ein entfernter Methodenaufruf erfolgreich verlaufen ist, andernfalls *onFailure()* [Hanson 2007, S. 343].

28 Siehe S. 66.

Zudem fällt auf, dass die Methoden im asynchronen Interface den Rückgabewert *void* haben. Der Grund dafür hängt mit den beiden Methoden der *AsyncCallback*-Schnittstelle zusammen. Im Erfolgsfall wird über das Callback-Objekt die gesamte Antwort der entfernten Methode an den Aufrufer weitergereicht. Nach der Dienstdefinition erfolgt die Implementierung dessen. An dieser Stelle sei angemerkt, dass die Implementierung des RPC-Dienstes, welche eigentlich server-seitig hätte beschrieben werden müssen, hier erfolgt, um den Mechanismus an sich vollständig erklären zu können. Für die Dienstimplementierung muss ein Servlet erstellt werden, das die GWT-spezifische Schnittstelle *com.google.gwt.user.server.rpc.RemoteServiceServlet* erweitert. Zudem muss das Servlet, wie in Abbildung 35 zu sehen, die Dienstschnittstelle *AccessGwtService* implementieren [Hanson 2007, S. 337].

```
public class AccessGwtServiceImpl extends
    RemoteServiceServlet implements
        AccessGwtService {

    public UserDto login
        (String username, String password) {

        // TODO
        return null;

    }

    public boolean logout() {

        // TODO
        return false;
    }

}
```

Abbildung 35: Implementierung eines RPC-Dienstes

Hierbei muss sichergestellt werden, dass sich das Servlet im *server*-Package befindet, da es Java-Code beinhaltet, der nicht in JavaScript kompiliert werden muss. Nachdem die Dienstschnittstelle definiert und implementiert, die asynchrone Schnittstelle erstellt und ein serialisierbares Objekt angelegt wurde, muss der Dienst aus dem Client heraus aufgerufen werden.

Der Aufruf erfolgt immer nach dem hier vorgestelltem Schema [Hanson 2007, S. 341 - 343]:

1. Erstellen eines Proxy-Objektes

   Dazu wird die Methode *GWT.create()*, wie in Abbildung 36 zu sehen, verwendet. Dieser wird die Dienstschnittstelle als Argument übergeben. *create()* gibt selbst ein Proxy-Objekt zurück, welches in die asynchrone Schnittstelle umgewandelt werden muss.

```
private final AccessGwtServiceAsync accessService = GWT
.create(AccessGwtService.class);
```

   Abbildung 36: Instanziierung eines Proxy-Objektes

2. Umwandlung eines Proxy-Objektes in *ServiceDefTarget*

   Diese Umwandlung wird vorgenommen, um die URL für den entfernten Dienst festlegen zu können. Darüber hinaus muss mittels der Methode *getModuleBaseURL()* die Stelle angegeben werden, an der das Servlet zu finden ist. Zuletzt muss die URL der Serviceimplementierung mit der Methode *setServiceEntryPoint(url)* gesetzt werden [Hanson 2007, S. 342].

Diese Schritte können eingespart werden, wenn die Dienstschnittstelle mit der Annotation *@RemoteServiceRelativePath*, wie in Abbildung 33 zu sehen war, versehen wird. Dennoch muss die Stelle, an der das Servlet zu finden ist, angegeben werden. Dafür muss, wie in Abbildung 37 dargestellt, das Servlet in der *web.xml* angegeben werden. Die *web.xml* befindet sich im Verzeichnis *WEB-INF*.

```
  <servlet>
    <servlet-name>accessServlet</servlet-name>
    <servlet-class>de.walkndoc.server.AccessGwtServiceImpl</servlet-class>
  </servlet>

  <servlet-mapping>
    <servlet-name>accessServlet</servlet-name>
    <url-pattern>/walkndoc/accessService</url-pattern>
  </servlet-mapping>

  <!-- Default page to serve -->
  <welcome-file-list>
    <welcome-file>WalkNDoc.html</welcome-file>
  </welcome-file-list>

</web-app>
```

Abbildung 37: Servletangabe in der web.xml

Das Element *servlet* beschreibt den Namen und die Klasse des Servlets. Im nächsten Tag werden den Servlets bestimmte Url-Patterns zugeordnet. Es muss beachtet werden, dass der Name im Tag *url-pattern* mit dem Argument in *RemoteServiceRelativePath* übereinstimmt.

3. Durchführung eines entfernten Methodenaufrufs und Erstellen eines Callback-Objektes.

   In Abbildung 38 ist zu erkennen, dass für die Erstellung eines Callback-Objektes immer die beiden Methoden *onSuccess()* und *onFailure()* implementiert werden müssen.

```
accessService.login(username.getValue(), password.getValue(),
        new AsyncCallback<UserDto>() {

    @Override
    public void onFailure(Throwable caught) {

        //TODO

    }

    @Override
    public void onSuccess(UserDto result) {

        //TODO

    }
});
```

Abbildung 38: Entfernter Methodenaufruf

5.4 Build-Prozess

Bislang wurde die Implementierung des Backends und der GUI erläutert. Sobald alle relevanten Bestandteile fertig implementiert sind, kann das Anwendungsprogramm automatisch erzeugt werden. Dieser Vorgang wird in der Informatik als Build-Prozess bezeichnet. Bei diesem Prozess bilden Skripte die Basis. Innerhalb dieser Skripte werden die durchzuführenden Aufgaben in einer speziellen Skriptsprache formuliert. Die Skripte werden dabei mit Hilfe eines Interpreters, wie beispielsweise Ant oder Maven, ausgeführt [Popp 2006, S. 47].

Für den Build-Prozess des Dokumentations- und Überwachungssystems wird das Java-Programm Apache Maven[29] verwendet. Maven ist ein quell-offenes Projekt der Apache Software Foundation. Für das Anwendungssystem wird die Version 2.0 verwendet. Dieses Tool bietet eine hervorragende Lösung zur Verwaltung der Abhängigkeiten zu externen Bibliotheken an. Maven basiert auf einem deklarativen Ansatz. Bei einem solchen Ansatz wird der Build-Prozess nicht selbst implementiert, sondern mit dem Project Object Model (POM) beschrieben. Das Projektmodell enthält die Metainformationen zu einem Projekt und wird standardmäßig in Form einer XML-Datei, der `pom.xml`, erfasst [Popp 2006, S. 234].

29 Die Projektseite von Maven ist unter http://maven.apache.org zu finden.

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
    http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <groupId>WalkNDoc</groupId>
    <artifactId>WalkNDoc</artifactId>
    <version>0.0.1-SNAPSHOT</version>
    <packaging>war</packaging>
```

Abbildung 39: Der Kopf der pom.xml

Abbildung 39 zeigt den Kopf der Datei *pom.xml*. Dieser enthält bereits allgemeine Angaben, wie beispielsweise die Version des Projektmodells oder die eindeutige Identifikation des Produktes über die *groupId*. Durch die Angabe der *artifactId* kann der Name des Artefaktes festgelegt werden. „Maven versteht unter einem Artefakt eine in sich abgeschlossene (Teil-) Funktionalität eines Projektes, die eigenständig ausgeliefert werden kann.“ [Popp 2006, S. 235] Für das Dokumentations- und Überwachungssystem ist die Unterscheidung zwischen Projekt und Artefakt nicht zwingend relevant, da die zu erstellende Software nur aus einem einzigen Artefakt besteht. Das Artefakt ist in diesem Fall vom Typ Web Application Archive (WAR). Ein *war-file* stellt dabei einen Container dar, welcher alle Dateien einer Webanwendung bündelt.

Gleichzeitig können innerhalb des Projektmodells Properties definiert werden, um beispielsweise Quellelemente zu ermitteln oder die Version von externen Bibliotheken festzulegen. Nachstehende Abbildung zeigt die Definition der Properties.

```
<properties>
    <gwt.version>2.0.4</gwt.version>
    <maven.compiler.source>1.5</maven.compiler.source>
    <maven.compiler.target>1.5</maven.compiler.target>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    <spring.version>3.0.3.RELEASE</spring.version>
</properties>
```

Abbildung 40: Definition der Properties innerhalb der pom.xml

An dieser Stelle wird festgehalten, dass für GWT die Version 2.0.4, für Spring die Version 3.0.3 und für den Compiler die Version 1.5 verwendet wird. Darüber hinaus können mittels des Elementes *<dependencies>* externe Bibliotheken eingebunden werden. Dabei muss für jede externe Bibliothek ein eigenes *<dependency>*-Subelement angelegt werden [Popp 2006, S. 239]. Innerhalb dieser Subelemente werden durch *groupId*, *artifactId* und *version* eindeutige Schlüssel für die erzeugten Artefakte festgelegt. Zusätzlich kann ein weiterer Eintrag für den Gültigkeitsbereich (*scope*) angegeben werden. Sofern diese Angabe entfällt, wird automatisch der Gültigkeitsbereich *compile* angenommen.

Abbildung 41 zeigt, wie externe Bibliotheken innerhalb der *pom.xml* festgelegt werden.

```
<dependencies>
    <!-- GWT dependencies -->
    <dependency>
        <groupId>com.google.gwt</groupId>
        <artifactId>gwt-servlet</artifactId>
        <version>${gwt.version}</version>
        <scope>compile</scope>
    </dependency>
    <dependency>
        <groupId>com.google.gwt</groupId>
        <artifactId>gwt-user</artifactId>
        <version>${gwt.version}</version>
        <scope>provided</scope>
    </dependency>
```

Abbildung 41: Festlegung externer Bibliotheken innerhalb der pom.xml

Maven verwendet für die Verwaltung der externen Bibliotheken eigene Repositorys. Um Zugriff auf externe Bibliotheken zu erhalten, sucht Maven zunächst nach diesen im lokalen Repository. Sind dort die gewünschten Bibliotheken nicht vorhanden, wird die Suche im Remote-Repository fortgesetzt [Popp 2006, S. 240].

Letztendlich wird innerhalb des *<build>*-Elementes der Vorgang zur Erstellung des Produktes vorgenommen, siehe Abbildung 42. Wie bereits im Kopf der *pom.xml* festgelegt, erfolgt die Auslieferung des Dokumentations- und Überwachungssystems in Form einer *war*-Datei.[30] Dafür wird das *Maven-War-Plugin* benötigt.

```
<!-- Build Process -->
<build>
    <plugins>
        <plugin>
            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-war-plugin</artifactId>
            <version>2.0.2</version>
            <configuration>
                <webappDirectory>./war/</webappDirectory>
            </configuration>
        </plugin>
```

Abbildung 42: Definition des war-Plugins in der pom.xml

Laut dem Maven-Build-Lifecycle[31] muss die Phase *package* ausgeführt werden, um eine *war*-Datei zu erstellen. Im Erfolgsfall wird diese erstellt und kann nun von Tomcat[32] entpackt und ausgeführt werden.

30 Siehe S. 69.

31 Der Build-Prozess von Maven umfasst mehrere Build-Phasen. Ein Überblick ist unter http://maven.apache.org/guides/introduction/introduction-to-the-lifecycle.html zu finden.

32 Tomcat bezeichnet einen Webserver, welcher Java-basierte Webanwendungen ausführen kann. Die Projektseite von Tomcat ist auf http://tomcat.apache.org/ zu finden.

6. Ergebnisse

Im Rahmen dieser Diplomarbeit wurde eine Webanwendung mit dem Namen *WalkNDoc* zur Kontrolle und Überwachung des medizinischen Gehtrainings bei Patienten mit pAVK entwickelt. Die beiden vorangehenden Kapitel haben den Leser schrittweise bei der Konzeption und Implementierung des Systems begleitet. Im jetzigen Kapitel soll dem Leser das Bild vervollständigt werden, indem an dieser Stelle auf Ergebnisse eingegangen wird, die durch die vorherigen Kapitel nicht erfasst wurden.

Das Dokumentations- und Überwachungssystem *WalkNDoc* kann von folgenden zwei Benutzergruppen verwendet werden.

- Ärzte
- Patienten

Dabei gestattet die Software, je nachdem wer sich anmeldet, einen unterschiedlichen Funktionsumfang. Dieser soll in den nachfolgenden beiden Unterkapiteln demonstriert werden. Zunächst werden jedoch die Funktionen, welche beide Benutzer gemeinsam haben, vorgestellt.

Login

Beim ersten Aufruf von *WalkNDoc* wird dem Benutzer eine Loginseite angezeigt. Wie bei Web-Auftritten üblich, wird den Benutzern eine Kontaktadresse für eventuelle Rückfragen angeboten. Die Benutzer haben auf der Loginseite die Möglichkeit sich mit ihrem Benutzernamen und dem Passwort anzumelden. Abbildung 43 zeigt den Logindialog des Dokumentations- und Überwachungssystems.

Abbildung 43: Logindialog

Bei der Bestätigung der Eingaben werden diese mit der eigenen PostgreSQL-Datenbank verglichen, um die Daten zu überprüfen. Sind die Eingaben gültig, ist der Benutzer am System angemeldet und befindet sich auf der Startseite.

Startseite

Die Startseite, in Abbildung 44 zu sehen, bietet den Benutzern die Möglichkeit, Nachrichten untereinander auszutauschen. Dafür müssen zunächst die Sendedaten, wie Datum, Uhrzeit, Absender und Empfänger, festgelegt werden. Im Anschluss daran kann die Nachricht geschrieben werden. Das System überprüft automatisch während der Dateneingabe, ob alle Felder mit Werten belegt sind. Ist dies nicht der Fall, erscheint direkt eine Meldung, die den Benutzer darauf hinweist, dass dieses Feld erforderlich ist.

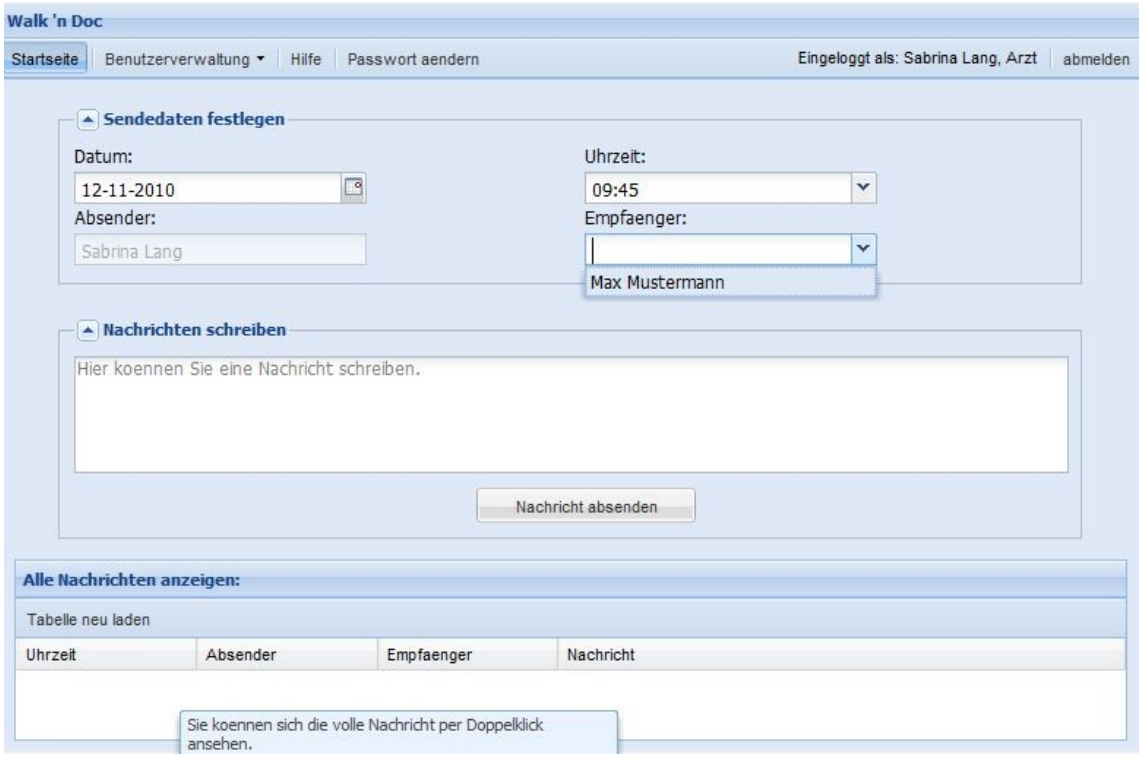


Abbildung 44: Startseite

Sobald alle Felder mit Werten versehen sind, kann die Nachricht abgeschickt und dauerhaft in der Datenbank gespeichert werden. Die Nachrichten werden nach ihrem jeweiligen Datum gruppiert und in der Tabelle angezeigt. Es besteht jederzeit die Möglichkeit sich eine Nachricht per Doppelklick ausführlicher anzusehen, siehe Abbildung 45.

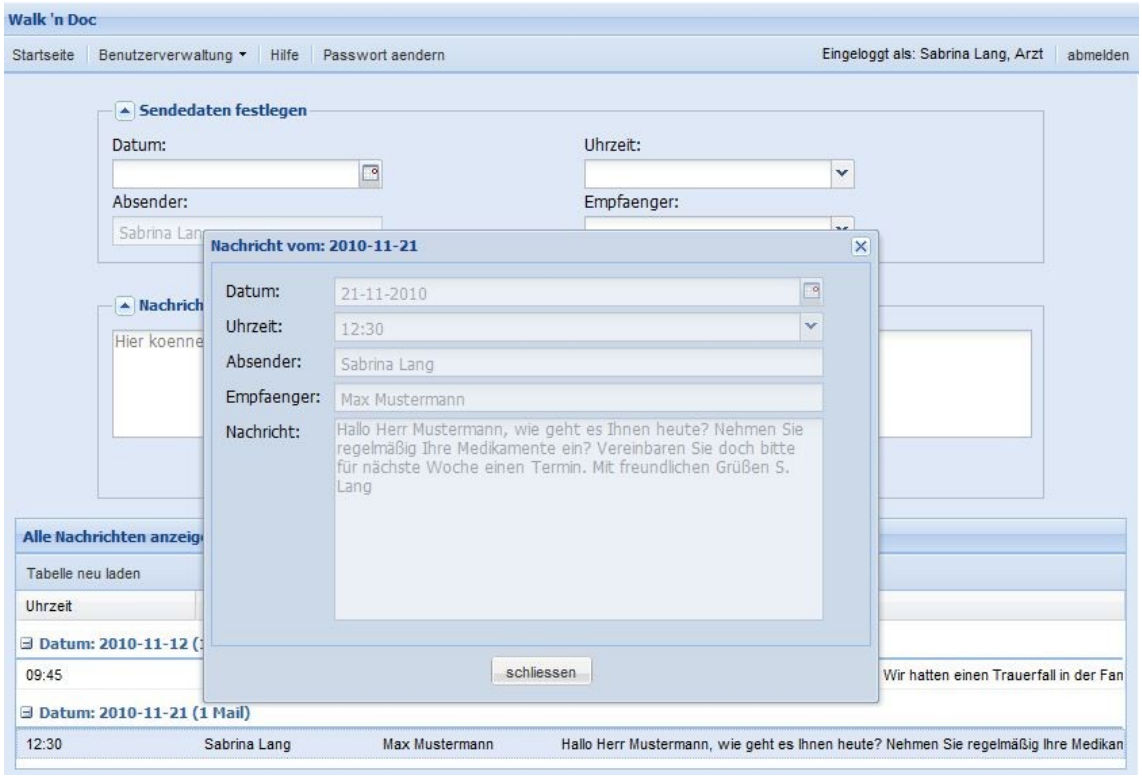


Abbildung 45: Beispiel einer Nachricht

Die Benutzer können sich jederzeit über den Button `abmelden` ausloggen. In diesem Fall erscheint erneut der Logindialog. Der Name des Benutzers und dessen Rolle, die ihm bereits beim Login zugewiesen wurde, erscheint direkt daneben.

Außerdem besteht zu jedem Zeitpunkt die Möglichkeit sein Passwort zu ändern. Dafür muss der Menüpunkt `Passwort aendern` gewählt werden, wie in Abbildung 46 ersichtlich ist.

Abbildung 46: Ändern des Passwortes

Es erscheint ein Fenster, in dem zunächst das alte und dann das neue Passwort eingegeben werden muss. Die Änderung wird mit der Bestätigung der Daten vorgenommen.

Darüber hinaus wird den Benutzern eine individuelle Hilfe zur Verfügung gestellt, welche die Lernförderlichkeit[33] des Systems unterstützt. Generell können bei Bedarf jederzeit alle Vorgänge abgebrochen werden. Diese Möglichkeit ist besonders wichtig, um dem Benutzer das Gefühl der Steuerbarkeit[34] zu vermitteln.

Im folgenden Kapitel wird der Funktionsumfang für Ärzte beschrieben.

33 Siehe S. 13.

34 Siehe S. 13.

6.1 WalkNDoc aus der Sichtweise des Arztes

Das Dokumentations- und Überwachungssystem gestattet den Ärzten zusätzlich nachstehende Funktionen, die im Anschluss entsprechend der Reihenfolge dargestellt werden:

- Eine getrennte Benutzerverwaltung zum Anlegen, Bearbeiten und Löschen von Ärzten und Patienten
- Das patientenindividuelle Festlegen von Grenzwerten für Blutdruck und Puls. Bei Überschreitung erfolgt die automatische Generierung einer E-Mail
- Eine grafische und tabellarische Darstellung der patientenindividuellen Messwerte zur Kontrolle

Benutzerverwaltung

Die Verwaltung von Ärzten und Patienten wird getrennt voneinander vorgenommen, obwohl die Funktionsweise im Wesentlichen dieselbe ist. Aus diesem Grund wird beispielhaft die Patientenverwaltung erläutert. Dazu muss im Menü der Punkt *Benutzerverwaltung -> Patientenverwaltung* gewählt werden.

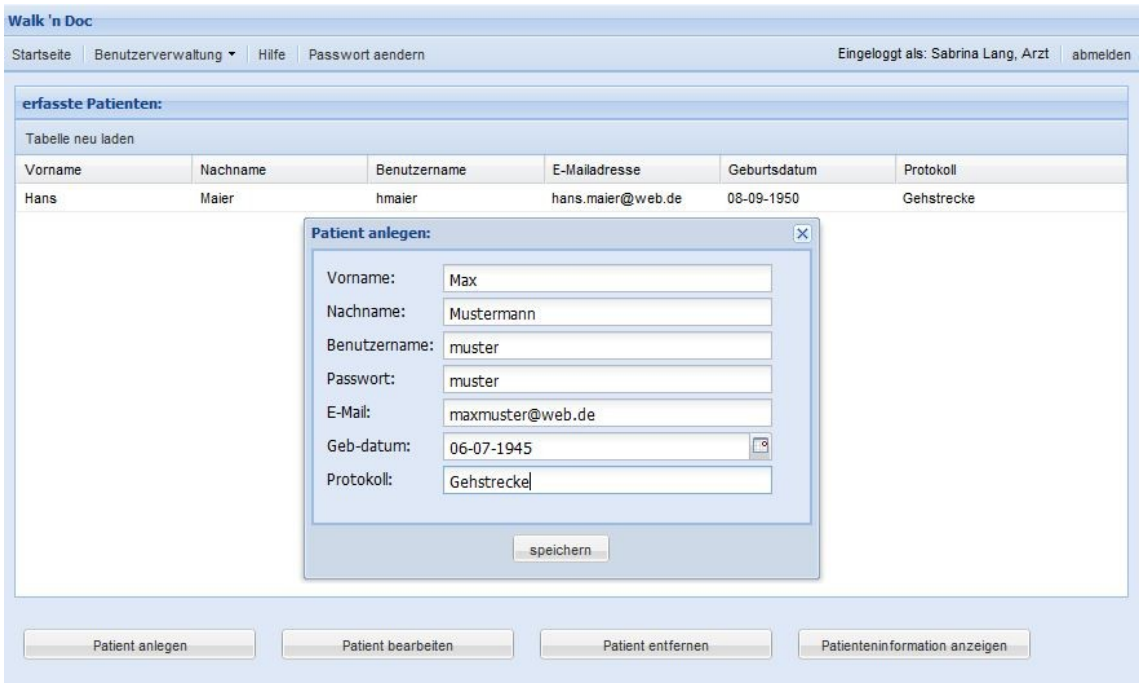


Abbildung 47: Anlegen eines Patienten

Abbildung 47 zeigt die Verwaltung der Patienten. Dabei werden alle bisher erfassten Patienten in der Tabelle aufgelistet. Mittels `Patient anlegen` können weitere Patienten erfasst werden. Dazu öffnet sich ein Dialog, in dem relevante Informationen, wie u.a. der Vor- und Nachname sowie das Geburtsdatum eines Patienten eingegeben werden müssen. Die Validierung der Daten erfolgt direkt bei der Dateneingabe („on the fly“). Wird beispielsweise in einem Textfeld, das nur für den Typ String[35] vorgesehen ist, eine Zahl eingetragen, erscheint sofort eine Meldung, die besagt, dass nur Buchstaben erlaubt sind. Falls eines der Textfelder nicht ausgefüllt wird, erscheint ebenfalls ein Warnung, die darauf hinweist, dass dieses Textfeld erforderlich ist. Erst wenn alle Textfelder korrekt ausgefüllt sind, kann der neue Patient dauerhaft in der Datenbank gespeichert werden.

35 Ein String ist im Java-Umfeld eine Aneinanderreihung von Buchstaben.

Aus Gründen der Fehlertoleranz[36] besteht jederzeit die Möglichkeit, bereits angelegte Datensätze zu überarbeiten. Abbildung 48 demonstriert die Bearbeitung eines Patienten.

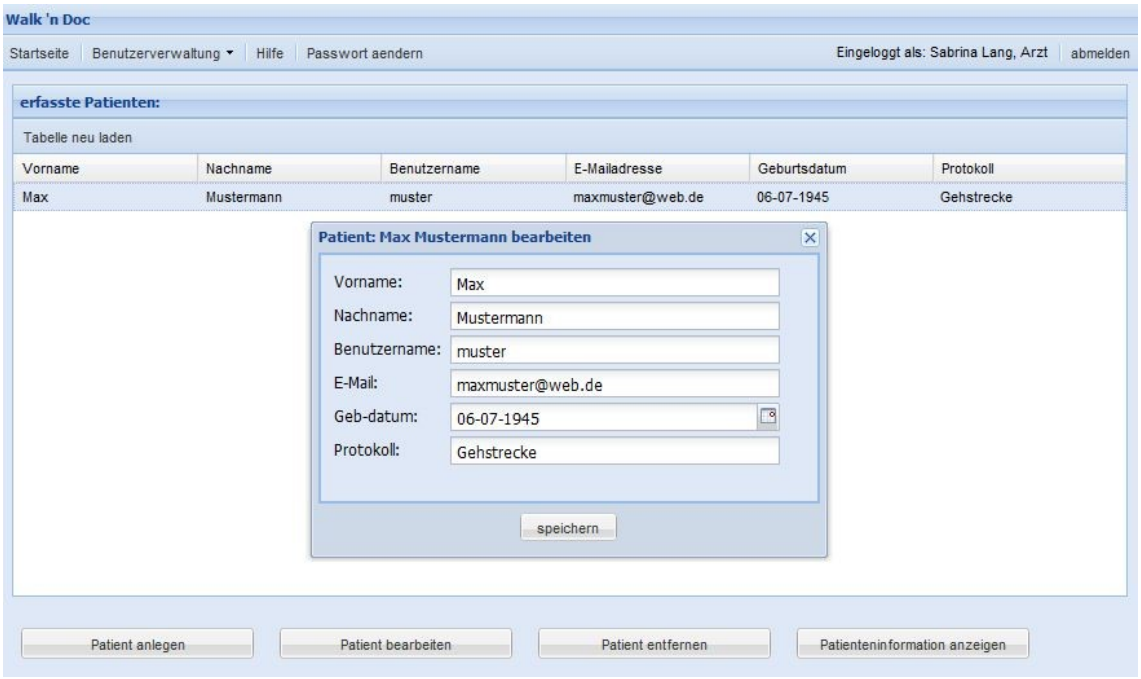


Abbildung 48: Bearbeiten eines Patienten

Dazu muss zunächst der gewünschte Patient in der Tabelle selektiert werden. Mit Klick auf `Patient bearbeiten` öffnet sich ein Dialog, welcher bereits mit Werten in den Textfeldern belegt ist. Dies hat den Vorteil, dass Änderungen nur an den entsprechenden Stellen vorgenommen werden müssen. Auch in diesem Fall erfolgt die Validierung on the fly.

36 Siehe S. 12.

Darüber hinaus besteht die Möglichkeit, bereits erfasste Patienten vom System zu entfernen. Hierfür muss ebenfalls der gewünschte Patient selektiert und der Button *Patient entfernen* betätigt werden.

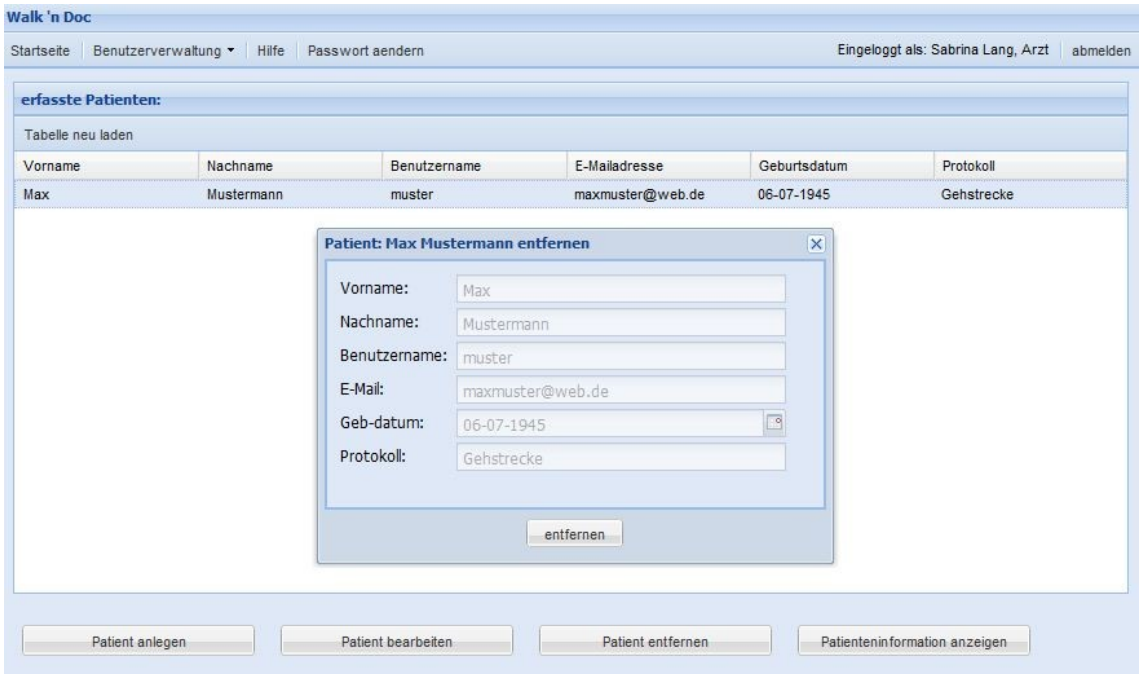


Abbildung 49: Entfernen eines Patienten

Wie in Abbildung 49 zu sehen, öffnet sich ein Dialogfenster, welches mit Werten vorbelegt ist, wobei diese grau hinterlegt sind und somit nicht editiert werden können. Der Patient wird mit der Bestätigung seiner Daten aus der Datenbank genommen.

Ärzte können innerhalb der Patientenverwaltung weitere Informationen zu einem Patienten festlegen. Dazu selektieren sie diesen und betätigen den Button *Patienteninformation anzeigen*. Anschließend öffnet sich ein Dialog, der sich aus mehreren Reitern zusammensetzt, wie Abbildung 50 demonstriert.

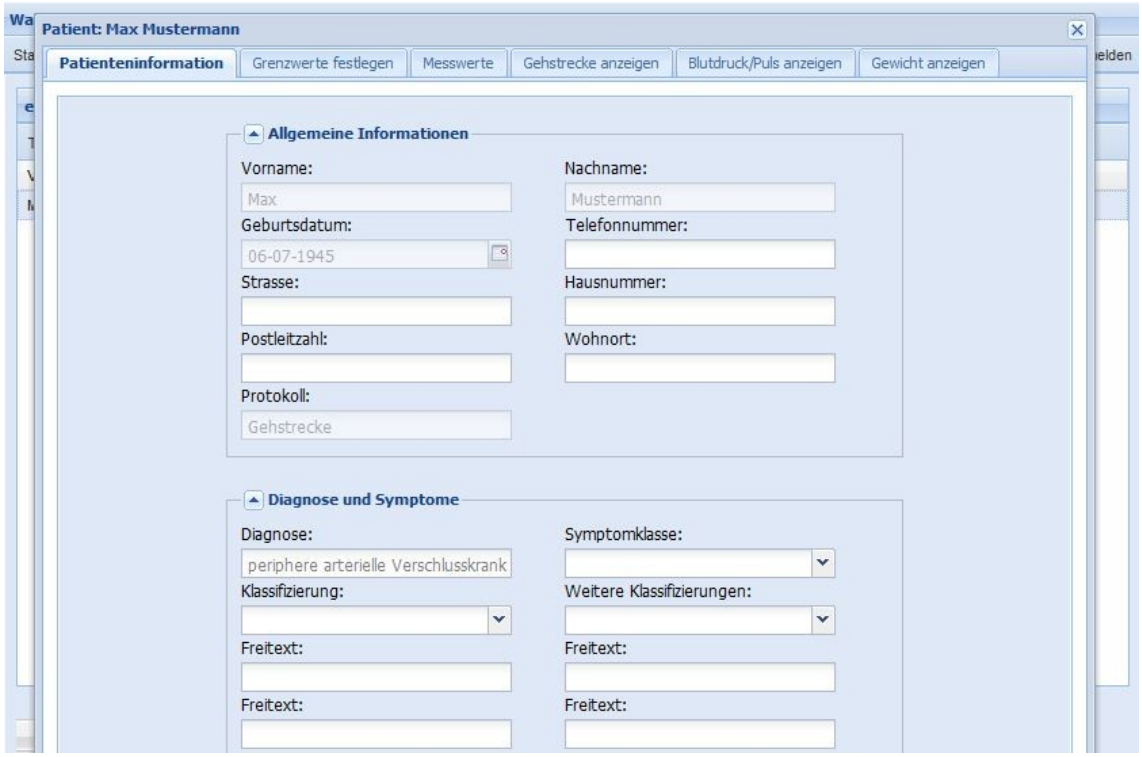


Abbildung 50: Weitere Informationen zu einem Patienten

Der erste Reiter enthält dabei sämtliche Informationen zur Anschrift und Diagnose des Patienten, wobei standardmäßig die Textfelder für Vor- und Nachname, Geburtsdatum und Protokoll, welche seit dem Anlegen existieren, mit Werten vorbelegt sind, die nicht mehr editiert werden können. Es bleibt dem Arzt selbst überlassen, ob er die weiteren Textfelder belegen möchte. Ist dies der Fall, erscheinen beim erneuten Öffnen des Dialogs diese Felder ebenfalls mit Werten.

Festlegung der Grenzwerte

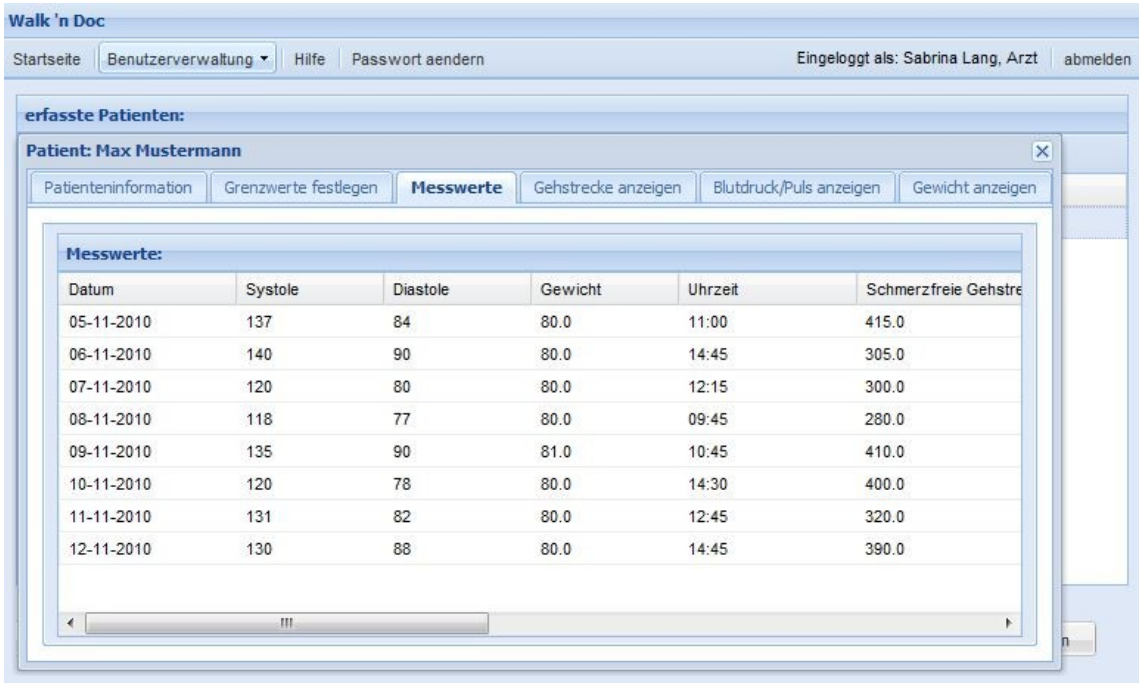
Im nächsten Reiter können die Grenzwerte für Systole, Diastole und Puls patientenindividuell festgelegt werden, siehe Abbildung 51. Sobald entsprechende Grenzwerte eingegeben wurden, überprüft das System zukünftig bei der Eingabe der Messwerte, ob diese unter- bzw. oberhalb der Grenzwerte liegen.

Abbildung 51: Festlegung von Grenzwerten für Blutdruck und Puls

Sobald die Messwerte außerhalb des gültigen Bereiches liegen, wird vom System automatisch eine E-Mail generiert und an den behandelnden Arzt versendet. Dieser kann daraufhin Kontakt mit dem Patienten aufnehmen.

Tabellarische und grafische Darstellung der Messwerte zur Kontrolle

Mit den nachfolgenden Reitern besteht für Ärzte die Möglichkeit, den Therapieerfolg zu kontrollieren. Dabei listet der dritte Reiter die patientenindividuellen Messwerte in Form einer Tabelle auf, wie in Abbildung 52 zu erkennen ist.



Datum	Systole	Diastole	Gewicht	Uhrzeit	Schmerzfreie Gehstre
05-11-2010	137	84	80.0	11:00	415.0
06-11-2010	140	90	80.0	14:45	305.0
07-11-2010	120	80	80.0	12:15	300.0
08-11-2010	118	77	80.0	09:45	280.0
09-11-2010	135	90	81.0	10:45	410.0
10-11-2010	120	78	80.0	14:30	400.0
11-11-2010	131	82	80.0	12:45	320.0
12-11-2010	130	88	80.0	14:45	390.0

Abbildung 52: Tabellarische Darstellung der patientenindividuellen Messwerte

Die letzten Reiter hingegen stellen die Messwerte in Form von Diagrammen grafisch dar. An dieser Stelle sei angemerkt, dass es diese Funktion auch innerhalb der Patienten-Oberfläche gibt und die Möglichkeiten dort ausführlicher beschrieben werden. Abbildung 53 zeigt beispielhaft die grafische Darstellung für den Parameter Gewicht eines individuellen Patienten.

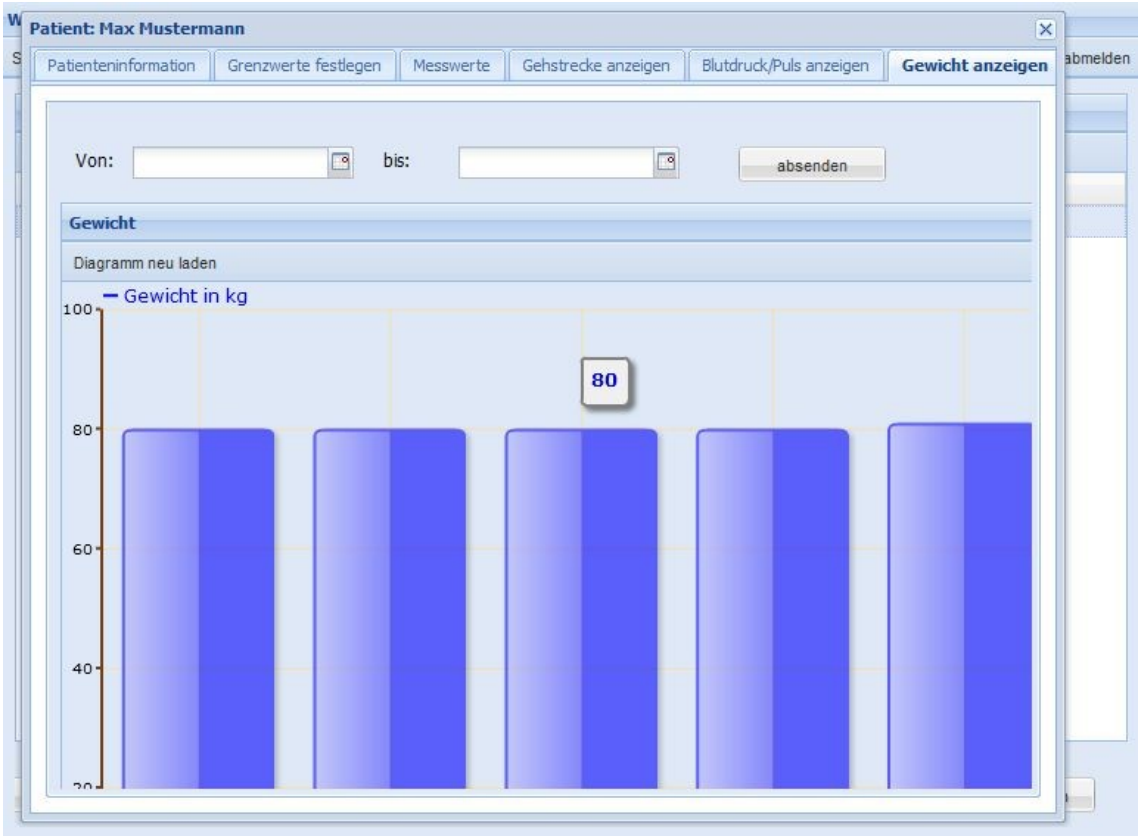


Abbildung 53: Grafische Darstellung des Gewichtes eines individuellen Patienten

Im nachfolgenden Kapitel wird nun die Oberfläche aus der Sichtweise eines Patienten erläutert.

6.2 WalkNDoc aus der Sichtweise des Patienten

Die Benutzergruppe Patient erhält folgende Zusatzfunktionen:

- Die Eingabe ihrer Messwerte in das System
- Das Bearbeiten vorhandener Messwert-Datensätze
- Die tabellarische und grafische Darstellung ihrer Messwerte

Eingabe der Messwerte

Patienten sollen im Idealfall ihre Messwerte täglich in das System eintragen, siehe Abbildung 54. Um die Eingabemaske zu erhalten, muss der Menüpunkt *Messwerte -> Eingabe* gewählt werden. Dabei sind folgende Parameter verpflichtend für die Eingabe:

- aktuelles Datum
- Systole
- Diastole
- Gewicht
- Uhrzeit
- schmerzfreie Gehstrecke
- gesamte Gehstrecke, ehe abgebrochen werden muss
- Dauer
- Puls

Die Angabe eines Kommentars hingegen ist freiwillig. Zum größten Teil müssen numerische Werte in ein `NumberField` eingetragen werden. Dieses Widget garantiert direkt bei der Dateneingabe, dass nur Zahlen zulässig sind.

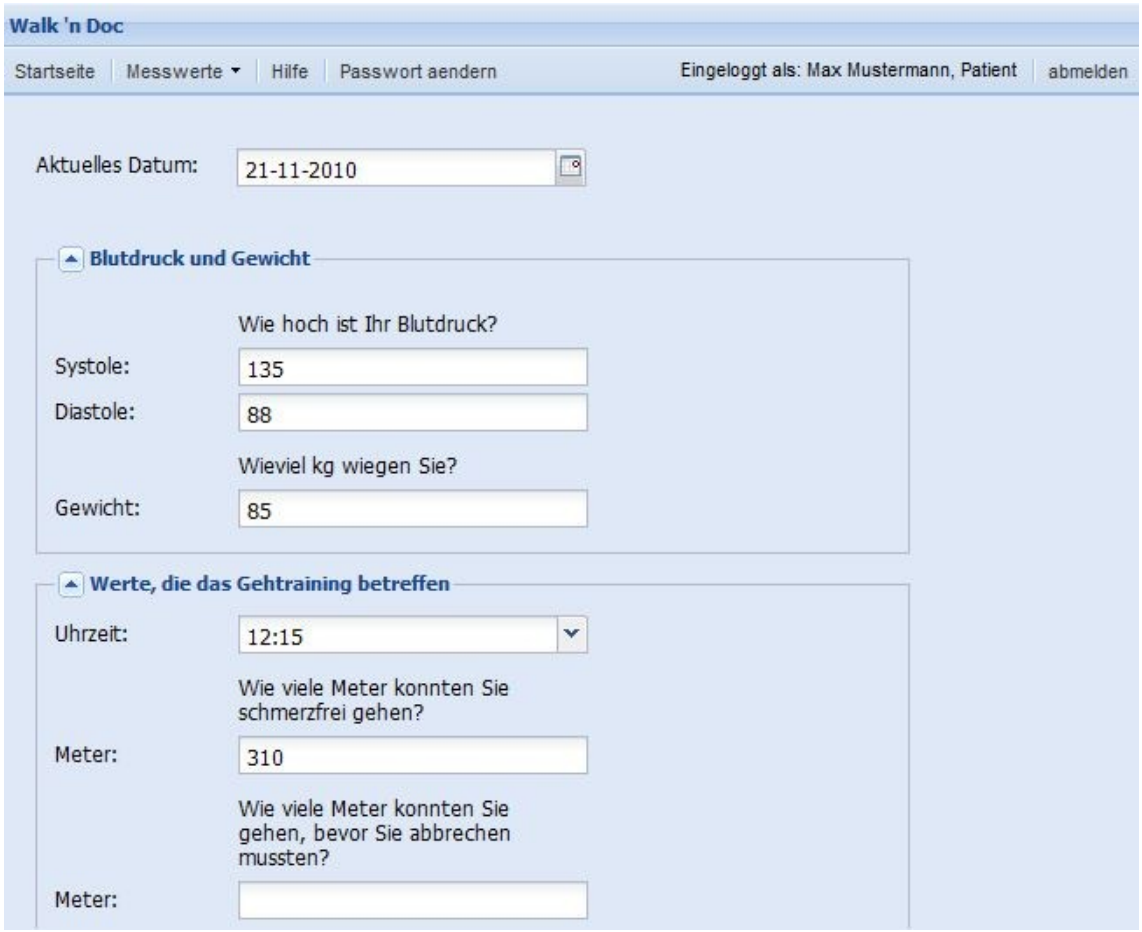


Abbildung 54: Ausschnitt aus der Eingabemaske

Ferner wird automatisch überprüft, ob alle Pflichtfelder mit Werten versehen sind. Sofern die Daten korrekt sind und bestätigt werden, erscheint im Erfolgsfall eine Meldung auf dem Bildschirm, die dem Patienten mitteilt, dass die Messwerte erfolgreich in der Datenbank gespeichert werden konnten. Eine Feedback-Meldung, wie beispielhaft in Abbildung 55 zu sehen, spricht für eine gute Selbstbeschreibungsfähigkeit[37] des Systems.

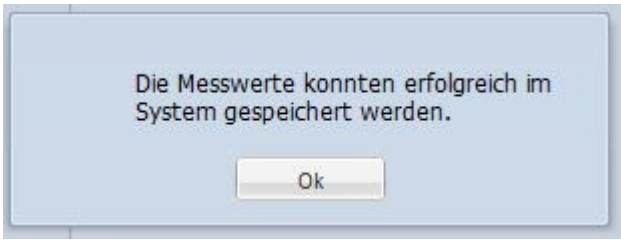


Abbildung 55: Mitteilung über die erfolgreiche Aufnahme eines Datensatzes

Bearbeiten vorhandener Messwert-Datensätze

Für den Fall, dass sich ein Patient bei der Eingabe seiner Messwerte vertippt hat, besteht die Möglichkeit zur Korrektur. Dafür muss im Menü der Punkt *Messwerte -> Anzeigen in Form einer Tabelle* gewählt werden.

37 Siehe S. 11.

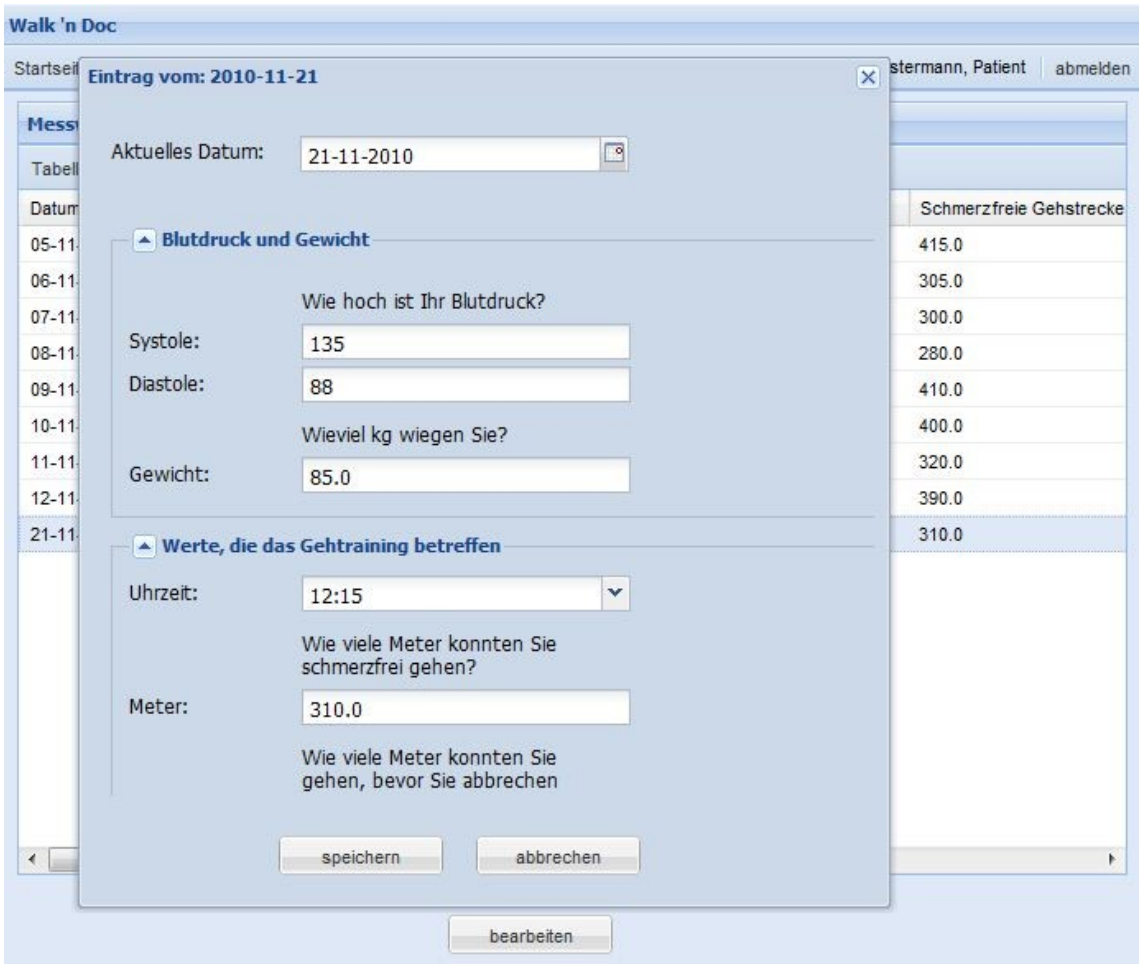


Abbildung 56: Bearbeiten eines Messwert-Datensatzes

Daraufhin wird eine Tabelle mit allen bisher gespeicherten Messwert-Datensätzen angezeigt. Der verbesserungswürdige Datensatz muss in der Tabelle selektiert werden. Danach stehen dem Patienten zwei Möglichkeiten für das weitere Vorgehen zur Verfügung. Zum Einen kann er direkt im Anschluss einen Doppelklick auf den selektierten Datensatz ausführen oder zum Anderen kann der Button `bearbeiten` betätigt werden.

In beiden Fällen öffnet sich daraufhin ein Dialog zum Bearbeiten des Datensatzes, siehe Abbildung 56. Dieser Dialog ist bereits mit den Werten aus dem Datensatz vorbelegt. Die Änderungen müssen demnach nur an den gewünschten Stellen erfolgen, wobei die `NumberFields` wiederum für die Validierung sorgen. Die Änderung wird mit der Bestätigung der Daten vorgenommen. Nach der Bestätigung schließt sich der Dialog automatisch und es wird erneut die Tabelle mit den Messwerten angezeigt. Die Verwendung von bekannten Mustern erweist sich als vorteilhaft. Für die Benutzerakzeptanz ist es besonders wichtig, dass der Aufbau der Eingabe- und Bearbeitenmaske ähnlich strukturiert ist, um Verwirrungen zu vermeiden.[38]

Grafische Darstellung der Messwerte

Für Patienten besteht gleichermaßen die Möglichkeit einer grafischen Darstellung der Messwerte. Diese können unter dem Menüpunkt `Messwerte -> Anzeigen in Form von Diagrammen` eingesehen werden. Unterschieden wird dabei zwischen den nachfolgenden Darstellungsarten für:

- Gehstrecke und Dauer
- Blutdruck und Puls
- Gewicht

In allen drei Fällen werden zunächst von den entsprechenden Parametern alle Existierenden angezeigt.

38 Siehe S. 12.

Zusätzlich kann ein Zeitraum, innerhalb dessen die Messwerte liegen sollen, gewählt werden. Somit können beispielhaft die Messwerte eines bestimmten Monats geplottet werden. Abbildung 57 zeigt die grafische Darstellung der Gehstrecken-Parameter.

Abbildung 57: Diagrammdarstellung für die Gehstrecke

Bei allen drei Diagrammen steht das Datum auf der x-Achse und die entsprechende Einheit auf der y-Achse.

Bei der Darstellung für den Blutdruck und Puls wird zusätzlich eine Tabelle mit Werten für Datum, Blutdruck, Puls und Kommentar angezeigt, die indirekt mit dem Diagramm verbunden ist, siehe Abbildung 58. Daraus ergibt sich die Möglichkeit, dass auffällige Werte direkt im Diagramm markiert werden können.

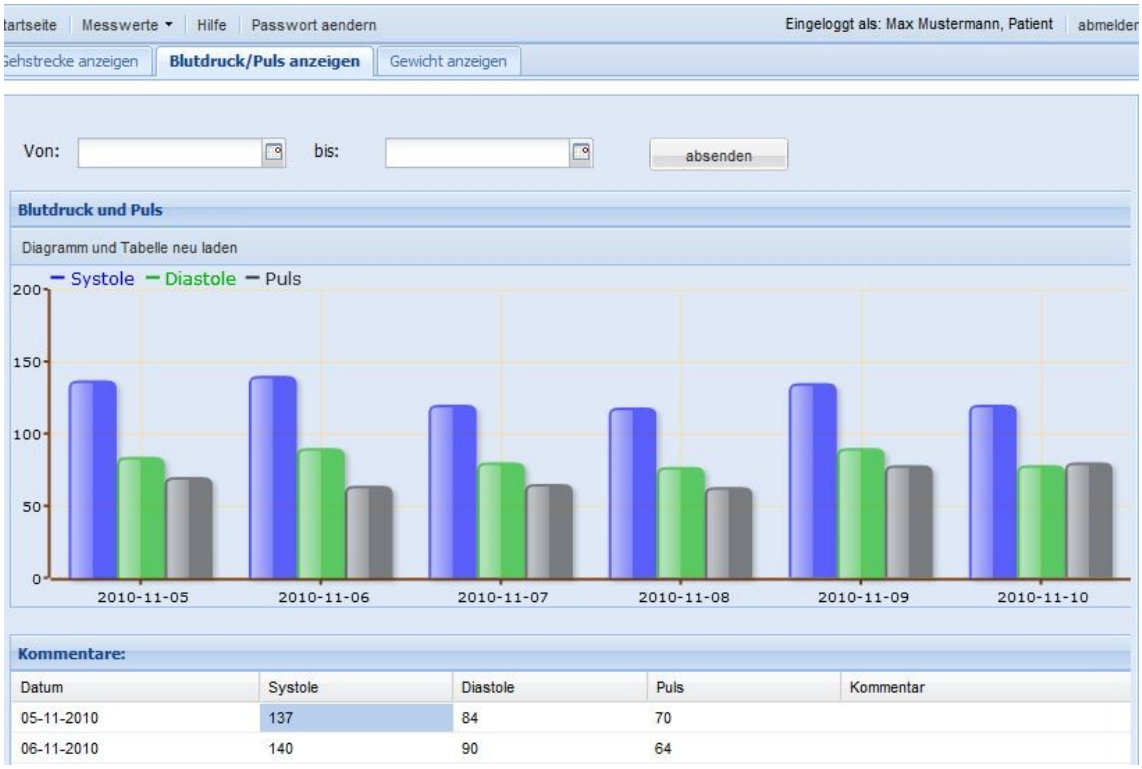


Abbildung 58: Diagrammdarstellung für Blutdruck und Puls

In der Tabelle wird entsprechend die dazugehörige Zelle selektiert. Sobald eine Zelle selektiert ist, kann in derselben Zeile nach dem Kommentar, welcher ggf. Aufschluss über das Verhalten bietet, gesucht werden.

Zu guter Letzt ist in Abbildung 59 das Diagramm für die Gewichtsdarstellung zu sehen.

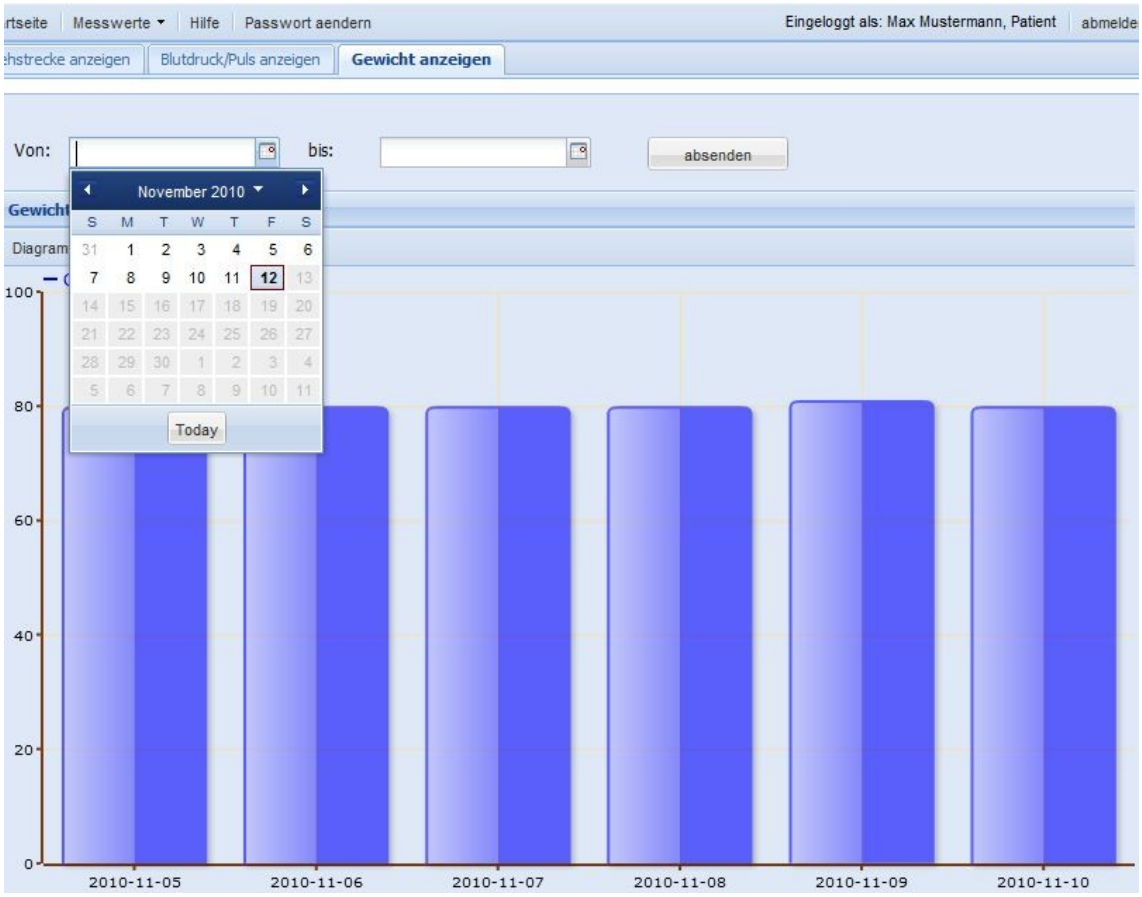


Abbildung 59: Diagrammdarstellung für Gewicht

6.3 Tests

Im Bereich der Software-Entwicklung ist das Testen einer Software eine bedeutende Methode zur Qualitätssicherung. Tests werden durchgeführt, um zum Einen die Qualität einer Software zu messen und zum Anderen um Fehler zu entdecken und diese bestmöglich zu beheben. Deshalb soll in diesem Kapitel auf mögliche Testverfahren eingegangen werden.

In der Regel werden Softwaretests in die vier folgenden Stufen unterteilt [Spillner 2007, S. 40]:

- Komponententest / Unit-Test
- Integrationstest
- Systemtest
- Abnahmetest

Je nach Umfang der Software können diese Merkmale beliebig weiter untergliedert werden. Speziell für das Dokumentations- und Überwachungssystem wurden Komponenten- und Systemtests durchgeführt. Auf Integrationstests konnte aufgrund der vergleichsweise geringen Komplexität des Gesamtsystems verzichtet werden. Aus diesem Grund wird an dieser Stelle nicht weiter darauf eingegangen. Mehr zum Thema Integrationstest kann im Buch (Spillner et al., 2007) in Kapitel 3.3 nachgelesen werden. Der Abnahmetest erfolgte in Form einer vierwöchigen Testphase zusammen mit Herrn Prof. Dr. med. Thomas Dengler.

Komponententest

Diese Tests finden auf unterster Ebene statt und werden von Entwicklern selbst durchgeführt. Im Java-Bereich gibt es das Framework JUnit, mit dem automatisch Klassen getestet werden können. Mit JUnit lassen sich Testklassen erstellen, die mehrere Testmethoden umfassen. Die Umsetzung einer solchen Testklasse soll anhand der Klasse *PatientServiceTest* gezeigt werden. Bei Komponententests werden einzelne Klassen auf ihre korrekte Funktionalität getestet. In diesem Fall wird mit der Klasse *PatientServiceTest* getestet, ob die Datenbankzugriffe der Standard-Operationen für ein Objekt vom Typ *PatientDto* korrekt ausgeführt werden.

In Abbildung 60 ist ersichtlich, dass Spring Unterstützung für JUnit anbietet. Mit der Annotation *@RunWith(SpringJUnit4ClassRunner.class)* wird Spring erlaubt den Test zu konfigurieren. Des Weiteren wird mittels der Annotation *@Context-Configuration* festgelegt, welche Konfigurationsdatei herangezogen werden soll [Zeitner 2008, S. 190].

```
//Erlaube Spring den Test zu konfigurieren
@RunWith(SpringJUnit4ClassRunner.class)
@ContextConfiguration(locations=
{"classpath:walkndoc-test.xml"})
@TransactionConfiguration(transactionManager =
    "transactionManager", defaultRollback = false)
@Transactional
public class PatientServiceTest extends
    ServiceTest<PatientService, PatientDto, String>{
```

Abbildung 60: Die Testklasse PatientServiceTest

Heutzutage sind zahlreiche Anwendungen datenbankgesteuert und folglich muss die Transaktion auch in Testklassen berücksichtigt werden. Der Spring-Class-Runner sorgt standardmäßig dafür, dass nach jeder Testmethode ein *rollback*[39] ausgeführt wird. Dieses Verhalten lässt sich mittels *@TransactionConfiguration* festlegen, wobei hierfür ein Transaktionsmanager in der Konfigurationsdatei vorhanden sein muss [Zeitner 2008, S. 200].[40] Zusätzlich erhält jede Testmethode die Annotation *@Transactional*. Darüber hinaus erweitert *PatientServiceTest* die Klasse *ServiceTest*, welche wiederum vom Interface *Service* abhängig ist, wie nachstehende Abbildung zeigt.

```
public abstract class ServiceTest<
    S extends Service<E, I>, E extends
    TransferObject<I>, I> {

    protected abstract S getService();

    @Test
    public void testService() {

        assertNotNull(getService());

    }

}
```

Abbildung 61: Die Testklasse Service

Innerhalb dieser Klasse wird die JUnit-Klasse *Assert* verwendet. Mit dieser Klasse können Werte und Bedingungen getestet werden, die jeweils erfüllt sein müssen, damit der Test in Ordnung ist. Dabei verifiziert die Methode *assertNotNull*, ob der entsprechende Service *null* ist. Die Methode *getService* wird entsprechend in der Klasse *PatientServiceTest* überschrieben, um den *PatientJdbcDaoService* zu erhalten, dessen Funktionalität getestet werden soll (siehe Abbildung 62).

39 Mit Rollback werden die einzelnen Schritte einer Transaktion zurückgesetzt.

40 Siehe Abbildung 23.

```
// Injiziere über den Namen der Bean
@Autowired
private PatientService patientJdbcDaoService;
private PatientDto patient;
private PatientDto newPatient;
private PatientDto updatePatient;

@Override
protected PatientService getService() {

    return patientJdbcDaoService;
}
```

Abbildung 62: Die Methode getService() in der Testklasse

Innerhalb der Klasse `PatientServiceTest` müssen Variablen vom Typ `PatientDto` angelegt werden. Die Variablen werden benötigt, um Instanzen vom Typ `PatientDto` zu erzeugen. Eine Instanz wird innerhalb der Methode `testCreate()` erzeugt, siehe Abbildung 63. Die Methode `testCreate()` aus Abbildung 63 ist mit der Annotation `@Test` versehen und somit erkennt JUnit diese als Testfall an.

```
@Test
@Transactional
@Rollback(false)
public void testCreate() {

    patient = new PatientDto();
    patient.setFirstName("Max");
    patient.setLastName("Mustermann");
    patient.setUsername("mMustermann");
    patient.setPassword("muster");
    patient.setEmail("m@m.de");
    patient.setDateOfBirth(new Date());
    patient.setProtocol("Gehstrecke");
    patient.setTelephoneNumber(987654);
    patient.setStreet("Mustermannstraße");
    patient.setHouseNumber(3);
    patient.setZipCode(73550);
    patient.setCity("Waldstetten");
    patient.setDiagnose("pAVK");
    patient.setClassification("Oberschenkel");
    patient.setMoreClassification("beidseitig");
    patient.setSymptomClass("III");
    patient.setFreeTextOne("Hypertonie");
    patient.setSysLimitTop(190);
    patient.setSysLimitDown(90);
    patient.setDiaLimitTop(120);
    patient.setDiaLimitDown(50);
    patient.setPulseLimitTop(100);
    patient.setPulseLimitDown(40);

    patientJdbcDaoService.create(patient);

    newPatient = patientJdbcDaoService.getPatient("mMustermann", "muster");
    assertEquals(patient.getId(), newPatient.getId());
    assertEquals(patient.getFirstName(), newPatient.getFirstName());
    assertEquals(patient.getLastName(), newPatient.getLastName());
    assertEquals(patient.getUsername(), newPatient.getUsername());
    assertEquals(patient.getPassword(), newPatient.getPassword());
    assertEquals(patient.getProtocol(), newPatient.getProtocol());
    assertEquals(patient.getCity(), newPatient.getCity());
    assertEquals(patient.getFreeTextOne(), newPatient.getFreeTextOne());

}
```

Abbildung 63: Die Methode testCreate() in der Testklasse

Die Parameter werden mit den *setter*-Methoden im Inneren der *testCreate()* gesetzt. Daraufhin erfolgt der Aufruf der Methode *create()* aus dem *PatientJdbcDaoService*.

Im Anschluss daran wird mittels der `getPatient()`-Methode nach einem Patienten-Objekt mit den entsprechenden Parametern in der Datenbank gesucht. Das gefundene Objekt wird direkt in einer neuen Variable gespeichert. Daraufhin wird mit der Methode `assertEquals()` überprüft, ob die ID des zuvor erstellten Objektes mit der ID des gefundenen Objektes übereinstimmt. Ferner wird überprüft, ob weitere Parameter der beiden Objekte übereinstimmen. Sofern kein `AssertionError` erscheint, kann sichergestellt werden, dass das richtige Objekt überprüft wurde.

Eine weitere Testmethode ist die Methode `testUpdate()`, die testet, ob das Bearbeiten eines Patienten-Datensatzes korrekt erfolgt. Dazu wird innerhalb dieser Methode zunächst mittels der `getPatient()`-Methode aus dem `PatientJdbcDaoService` ein Objekt vom Typ `PatientDto` geholt und in der Variable `newPatient` gespeichert, siehe Abbildung 64.

```
@Test
@Transactional
@Rollback(false)
public void testUpdate() {

    newPatient = patientJdbcDaoService.getPatient
        ("mMustermann", "muster");
    newPatient.setFirstName("Ali");
    newPatient.setLastName("G");

    patientJdbcDaoService.update(newPatient);

    updatePatient = patientJdbcDaoService.getPatient
        ("mMustermann", "muster");

    assertEquals(updatePatient.getId(), newPatient.getId());
    assertEquals(updatePatient.getFirstName(),
            newPatient.getFirstName());
    assertEquals(updatePatient.getLastName(), newPatient.getLastName());

}
```

Abbildung 64: Die Methode testUpdate() in der Testklasse

Daraufhin werden mit den *setter*-Methoden der Klasse *PatientDto* neue Parameter gesetzt. Anschließend erfolgt der Aufruf der Methode *update()*. Danach wird erneut mit der Methode *getPatient()* ein Objekt vom Typ *PatientDto* geholt und wiederum in einer Variable gespeichert. Zu guter Letzt wird mit *assertEquals* geprüft, ob bestimmte Parameter der beiden Objekte übereinstimmen. Die Klasse *PatientServiceTest* enthält weitere Testmethoden, die jedoch an dieser Stelle nicht zusätzlich aufgelistet werden, da sie entsprechend zu den bereits vorgestellten Methoden aufgebaut sind. Darüber hinaus werden die Testklassen für die weiteren DAO-Services nicht vorgestellt, weil der Aufbau ähnlich zur Klasse *PatientServiceTest* ist. Der einzige bedeutende Unterschied besteht darin, dass in den beiden Testklassen *MessageServiceTest* und *MeasuredDataServiceTest* innerhalb der Test-Methoden zunächst sichergestellt werden muss, dass ein Patient und/oder Arzt vorhanden ist, da zwischen diesen Klassen Beziehungen über Fremdschlüssel bestehen. Zudem muss gewährleistet werden, dass der Test dieser Klassen erst nach dem Ablauf der Testklassen *PatientServiceTest* und *PhysicianServiceTest* erfolgt.

Systemtest

Bei Systemtests wird das gesamte System gegen alle Anforderungen, sowohl funktional als auch nicht-funktional, getestet [Spillner 2007, S. 59]. Diese Tests finden in der Regel auf einer Testumgebung, wie beispielsweise Selenium[41], statt und laufen mit Testdaten ab. Selenium ist ein quell-offenes Testframework für Webanwendungen, welches von der Firma *ThoughtWorks* entwickelt wurde. Im Wesentlichen besteht das Framework aus den vier Modulen Core, Integrated Development Environment (IDE), Remote Control und Grid. Für das Dokumentations- und Überwachungssystem wurde das Modul IDE verwendet, welches als Firefox-Plugin installiert werden kann. Mit diesem Plugin können Testfälle aufgenommen, wieder abgespielt und beliebig oft wiederholt werden. Dies hat den Vorteil, dass sich Entwickler Tipparbeit einsparen. Dadurch wird das Testen schneller und flexibler.

Abbildung 65 zeigt beispielhaft die Aufnahme zur Überprüfung des Logins.

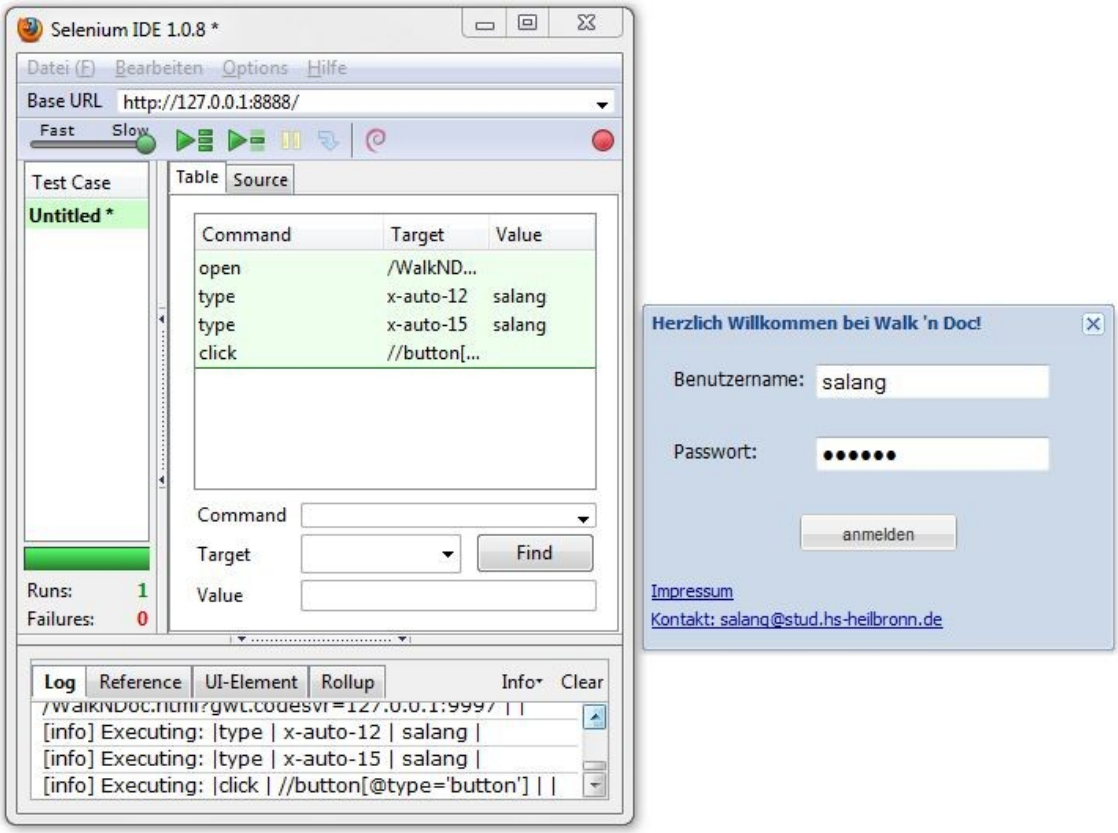


Abbildung 65: Beispiel eines Systemtests mit Selenium

Dafür muss zunächst die Webanwendung im Browser gestartet werden. Sobald der Login erscheint, wird Selenium aufgerufen. Um den Login-Vorgang aufnehmen zu können muss innerhalb der IDE der Record-Button gewählt werden. Dieser muss erneut gedrückt werden, um die Aufnahme zu stoppen. Im Anschluss daran kann der Testfall im Browser abgespielt werden. In Abbildung 65 ist zu erkennen, dass der Testfall fehlerfrei durchgeführt werden konnte.

Abnahmetest

Diese Art von Test wird von Kunden durchgeführt. Für solche Tests wird das sogenannte Blackbox-Verfahren angewendet. Bei diesem Verfahren betrachtet der Kunde nicht den Code selbst, sondern das Verhalten der Software bei typischen Abläufen. Das Dokumentations- und Überwachungssystem *WalkNDoc* wurde über einen Zeitraum von vier Wochen hinweg von Herrn Prof. Dr. med. Thomas Dengler und mir getestet. Für die Testphase wurde ein fiktiver Patient angelegt, dessen Messwerte täglich von mir in das System eingetragen wurden. Die Sichtweise von Herrn Prof. Dr. med. Thomas Dengler soll im nachfolgenden Unterkapitel dargestellt werden.

6.4 Benutzerfeedback

Herr Prof. Dr. med. Thomas Dengler ist zufrieden mit dem entwickelten Dokumentations- und Überwachungssystem als Webanwendung. Die gewünschten Funktionalitäten konnten entsprechend seinen Vorstellungen umgesetzt werden. Das System wird wohl speziell für Patienten mit peripherer arterieller Verschlusskrankheit keine Verwendung finden, da aufgrund des erhöhten Alters der Umgang mit dem Computer erschwert ist. Dennoch konnte mit diesem System aus seiner Sicht eine Grundlage geschaffen werden, die zukünftig eventuell für andere medizinische Fragestellungen eingesetzt werden soll. Ferner konnte mit dem System demonstriert werden, dass eine telemetrische Überwachung des Therapieerfolges ohne enorme Kosten denkbar ist. Das System wurde bewusst zu technisch gehalten, da eine Weiterentwicklung als Smartphone-Anwendung angestrebt werden soll. Zu Beginn des Projektes war ein praktischer Feldversuch mit zwei bis drei Testpatienten vorgesehen. Die Rekrutierung der Patienten sollte über das Klinikum am Plattenwald erfolgen. Der Feldversuch fand nicht statt, da Herr Prof. Dr. med. Thomas Dengler zum Zeitpunkt der Testphase keine passenden Patienten vorweisen konnte.

7. Diskussion und Ausblick

In diesem Kapitel sollen die erarbeiteten Ergebnisse in Hinblick auf die anfangs gesetzten Anforderungen kritisch bewertet werden. Des Weiteren werden Ansätze zur Weiterentwicklung des Dokumentations- und Überwachungssystems gegeben.

Mit dieser Arbeit konnte erfolgreich gezeigt werden, dass sich das Dokumentations- und Überwachungssystem *WalkNDoc* mit all seinen Funktionalitäten als Webanwendung umsetzen lässt. Ferner konnten die folgenden Anforderungen realisiert werden.

Software-ergonomische Aspekte

Es ist bekannt, dass Personen, die an einer pAVK leiden, meist ältere Leute sind. Aus diesem Grund wurde großer Wert auf eine intuitive Bedienbarkeit gelegt. Außerdem wurde berücksichtigt, dass sich das System entsprechend an die Bedürfnisse der Benutzer anpasst.

Speziell bei der Implementierung der grafischen Oberfläche wurde darauf geachtet, dass die sieben Grundsätze der Dialoggestaltung eingehalten werden. Zusätzlich zu den sieben Grundsätzen gibt es weitere Richtlinien und Gestaltungsgesetze, die die Bedienbarkeit steigern können. Ein noch effektiveres und effizienteres Arbeiten hätte gewährleistet werden können, wenn diese ebenfalls berücksichtigt worden wären. An dieser Stelle sei für weitere Informationen auf das Handbuch der Software-Ergonomie von Rudlof verwiesen.

Erweiterbarkeit

Im Rahmen dieser Arbeit konnte dank dem Spring Framework und der Existenz von Dienstschnittstellen eine leicht erweiterbare Architektur erstellt werden. Spring bietet mit seinem `ApplicationContext`, welcher die gesamte Instanziierung der einzelnen Beans erlaubt, eine hervorragende Möglichkeit zur Erweiterbarkeit an. Speziell das Zusammenspiel von Dependency Injection mit POJO-Klassen führt zu einer leicht erweiterbaren Architektur. Darüber hinaus können mit Spring weitere Frameworks leicht integriert werden.

Als verbesserungswürdig erscheint die Verwendung einer weiteren Properties-Datei, die die notwendigen Datenbankparameter enthält. Bislang wurden die Einstellungen dafür direkt im `ApplicationContext` festgehalten. Im Laufe der Entwicklung kann es jedoch vorkommen, dass die verwendete Datenbank gegen eine andere ausgetauscht werden muss. In diesem Fall wäre es hilfreich, wenn die Änderungen nur in einer einzigen Datei vorgenommen werden müssen.

Wiederverwendbarkeit

Zum jetzigen Zeitpunkt kann das Dokumentations- und Überwachungssystem, wie gewünscht, für Patienten mit pAVK verwendet werden, um die Durchführung eines medizinischen Gehtrainings zu kontrollieren. Dieses System kann jederzeit als Grundlage zur Erweiterung für andere medizinische Fragestellungen dienen.

Sollte dies der Fall sein, so variieren zukünftig die Parameter, die für die entsprechende Erkrankung charakteristisch sind. Folglich müssen eventuell andere Diagramme, die sich als geeigneter erweisen, zur Darstellung der Parameter gewählt werden.

Browserunabhängigkeit

Diese Anforderung kann prinzipiell durch den Einsatz des Goolge Web Toolkits verwirklicht werden. Im Rahmen dieser Diplomarbeit wurde geprüft, ob die Webanwendung auf den Browsern Mozilla Firefox und Internet Explorer lauffähig ist. Die Software konnte auf beiden Browsern problemlos ausgeführt werden. Allerdings muss ein Flash Player, welcher für die Wiedergabe interaktiver Anwendungen benötigt wird, installiert werden, damit die Diagramme korrekt dargestellt werden können.

Darüber hinaus muss JavaScript im Browser aktiviert sein, andernfalls erscheint eine leere Seite. Problematisch wird es an dieser Stelle, wenn aus Sicherheitsgründen eine Deaktivierung des JavaScripts gefordert wird. Dies ist allerdings ein prinzipielles Problem, welches beim Einsatz von AJAX-Anwendungen immer in Betracht gezogen werden muss, da diese generell einen JS-Interpreter voraussetzen.

Sicherheit

Aus Gründen der Datensicherheit läuft das Dokumentations- und Überwachungssystem über eine passwort-geschützte Internetseite. Die Abfrage des Passworts spielt dabei für die Authentifizierung der Benutzer eine große Rolle. Beim Login-Vorgang wird dem System mitgeteilt, dass eine Sitzung (Session) beginnt, die nach erfolgreichem Logout wieder beendet wird. Jede Sitzung ist dabei mit einem Sitzungsobjekt, welches durch die Klasse `HttpSession` abgebildet wird, verbunden.

Bisher können Benutzer nur Buchstaben für ihr Passwort verwenden. Diese Möglichkeit ist für eine simulierte Testphase völlig ausreichend. Sollte das Dokumentations- und Überwachungssystem eingesetzt werden, müssen zusätzlich Zahlen und Sonderzeichen für Passwörter erlaubt sein, um eine höchstmögliche Sicherheit der Daten gewährleisten zu können.

Ausblick

Bislang sind Methoden zur kontinuierlichen Überwachung noch nicht flächendeckend implementiert, jedoch konnte mit dieser Arbeit eine Ausgangsbasis dafür geschaffen werden. Darauf aufbauend können im Rahmen von weiteren Studien- oder Bachelorarbeiten diverse Ergänzungen an der Software vorgenommen werden.

Folgende Ansätze wären denkbar:

- *Datenübertragung über eine Bluetooth-Schnittstelle*

  Bisher wurden die Werte von Hand in das System eingetragen. Inzwischen verfügen jedoch viele der Standard-Messgeräte über eine Bluetooth-Schnittstelle zur direkten Übertragung der Werte auf den PC.

- *Bereitstellung der Software auf Smartphones*

  Dieser Ansatz wäre vorstellbar, wenn das Dokumentations- und Überwachungssystem für eine andere medizinische Fragestellung, die auch jüngere Personen betreffen kann, eingesetzt werden soll. In diesem Fall wäre es sinnvoll, eine zusätzliche Anwendung zu entwickeln, die es ermöglicht, zu jeder Zeit und ortsunabhängig damit zu kommunizieren.

- *Ausbauen der Patienteninformation*

  Die Informationen, die zu Patienten gespeichert werden, könnten beispielsweise um weitere Angaben wie Krankenkasse, Krankenvorgeschichte oder Medikamentengabe ergänzt werden. Darüber hinaus könnte zu jedem Patient ein Passbild gespeichert werden.

- *Überarbeitung der Diagramme*

  Zum jetzigen Zeitpunkt erfüllen die Diagramme ihren Zweck. Allerdings konnten bislang die Achsen der Diagramme mit keiner aussagekräftigen Einheit realisiert werden. Ferner werden die Messwerte entsprechend mit dem Datum auf der x-Achse dargestellt. Problematisch wird es an dieser Stelle, wenn mehrere Messwerte präsentiert werden sollen, da sich aufgrund der Menge die Daten überlappen. Diese Problematik muss überarbeitet werden, wenn das Dokumentations- und Überwachungssystem eingesetzt wird.

Schlussfolgerung

Das Dokumentations- und Überwachungssystem für medizinisches Gehtraining bei Patienten mit pAVK konnte im Rahmen dieser Diplomarbeit erfolgreich realisiert werden. Zudem konnten alle Anforderungen zur Zufriedenheit des Kunden umgesetzt werden. Außerdem wurde mit der erstellten Software eine solide Grundlage geschaffen, die für weitere medizinische Fragestellungen genutzt werden kann.

Abkürzungsverzeichnis

ACID

Atomicity-Consistency-Isolation-Durability

AJAX

Asynchronous JavaScript and XML

API

Application Programming Interface

CSS

Cascading Style Sheets

DAO

Data Access Object

DTO

Data Transfer Object

FK

Foreign Key

GUI

Graphical User Interface

GXT

Ext GWT

GWT

Google Web Toolkit

HTML

Hypertext Markup Language

HTTP

Hypertext Transfer Protocol

IDE

Integrated Development Environment

Java EE

Java Enterprise Edition

JDBC

Java Database Connectivity

JPA

Java Persistence API

JRE

Java Runtime Environment

JS

JavaScript

JSF

JavaServer Faces

JSP

JavaServer Pages

JSR

Java Specification Request

MVC

Model-View-Controller

pAVK

periphere arterielle Verschlusskrankheit

PK

Primary Key

POJO

Plain Old Java Object

POM

Project Object Model

RIA

Rich Internet Application

RPC

Remote Procedure Call

SQL

Structured Query Language

UI

User Interface

UML

Unified Modeling Language

WAR

Web Application Archive

WIQ

Walking Impairment Questionaire

XML

Extensible Markup Language

Abbildungsverzeichnis

Tabellenverzeichnis

Literaturverzeichnis

Internetquellen:

Apache License: Apache License Version 2.0. Zugriff am 27.10.2010 unter http://www.apache.org/licenses/LICENSE-2.0

Arbeitsgemeinschaft der Wissenschaftlichen Medizinischen Fachgesellschaften (AWMF): AWMF online – S3-Leitlinie Angiologie; Periphere arterielle Verschlusskrankheit (pAVK), Kapitel 3.4. Zugriff am 06.10.2010 unter http://www.uni-duesseldorf.de/AWMF/ll/065-003.htm

GNU Operating System: The GNU General Public License. Zugriff am 27.10.2010 unter http://www.gnu.org/licenses/gpl.html

Google Code: Making Remote Procedure Calls - Google Web Toolkit - Google Code. Zugriff am 09.10.2010 unter http://code.google.com/intl/deDE/webtoolkit/doc/latest/tutorial/RPC.html

ISO: EN ISO 9241. Zugriff am 12.08.2010 unter http://www.iso.org/iso/home.html

MedizInfo®: Symptome der peripheren arteriellen Verschlusskrankheit. Zugriff am 15.09.2010 unter http://www.medizinfo.de/arterien/pavk/symptome.shtml

NetDoktor, a: Periphere arterielle Verschlusskrankheit. Zugriff am 15.09.2010 unter http://www.netdoktor.de/Krankheiten/Raucherbein

NetDoktor, b: Periphere arterielle Verschlusskrankheit – Ursachen. Zugriff am 15.09.2010 unter http://www.netdoktor.de/Krankheiten/Raucherbein/Ursachen/

NetDoktor, c: Periphere arterielle Verschlusskrankheit – Symptome. Zugriff am 15.09.2010 unter http://www.netdoktor.de/Krankheiten/Raucherbein/Symptome/

NetDoktor, d: Periphere arterielle Verschlusskrankheit – Therapie. Zugriff am 15.09.2010 unter http://www.netdoktor.de/Krankheiten/Raucherbein/Therapie/

Open-Source Initiative: Common Public License Version 1.0: Licensing. Zugriff am 27.10.2010 unter http://www.opensource.org/licenses/cpl1.0.php

PostgreSQL: PostgreSQL: License. Zugriff am 27.10.2010 unter http://www-master.postgresql.org/about/licence

Rod Johnson: Introduction to the Spring Framework 2.5, 2007. Zugriff am 14.10.2010 unter http://www.theserverside.com/news/1363858/Introduction-to-the-Spring-Framework

Webseite des Athleten: Marathon – Jürgen Mennel läuft in einem Ultramarathon nach Athen. Zugriff am 11.09.2010 unter http://www.athenlauf.de/index.-php?id=63

Wikipedia: Wasserfallmodel - Wikipedia. Zugriff am 10.11.2010 unter http://de.wikipedia.org/wiki/Wasserfallmodell

Literaturquellen:

Auer, Christoph, 2010: Entwicklung einer Webplattform für wissenschaftliche Dokumente, Studienarbeit, Hochschule Heilbronn, Universität Heidelberg, 2010

Gamma, Erich (Hrsg.), 2007: Design patterns: Elements of reusable object-oriented software. 34th print, Boston: Addison-Wesley, 2007

Hanson, Robert / Tacy, Adam (Hrsg.), 2007: GWT im Einsatz: Ajax-Anwendungen entwickeln mit dem Google Web Toolkit. München: Hanser, 2007

Hess, Frank, 2006: Neukonzeption und prototypische Implementierung des CAMPUS-Players als Rich Internet Application, Diplomarbeit, Hochschule Heilbronn, Universität Heidelberg, 2006

Jacobi, Jonas / Fallows, John R. (Hrsg.), 2006: JSF und Ajax: Rich-Internet-Komponenten entwickeln mit JavaServer Faces und Ajax; [Entwicklung Ajax-fähiger JSF-Komponenten, Mozilla XUL und Microsoft HTC, zahlreiche wiederverwendbare Praxisbeispiele]. Bonn: mitp, 2006

Marinschek, Martin / Schnabel, Andrea / Müllan, Gerald (Hrsg.), 2007: JSF at Work: JavaServer Faces und Apache MyFaces erfolgreich einsetzen. 1. Aufl., Heidelberg: dpunkt.verlag, 2007

Moczar, Lajos / Alm, Petra (Hrsg.), 2005: Tomcat 5: Einsatz in Unternehmensanwendungen mit JSP und Servlets: Autoris. Übers. der amerikan. Orig.-Ausg. München: Addison-Wesley, 2005

Nicola, A. et al., 2009: The walking impairment questionnaire: an effective tool to assess the effect of treatment in patients with intermittent claudication, in: Journal of vascular surgery : official publication, the Society for Vascular Surgery [and] International Society for Cardiovascular Surgery, North American Chapter 50 (2009), S. 89–94, 19563956 / http://www.gopubmed.org/search?q=19563956&t=endnote

Popp, Gunther (Hrsg.), 2006: Konfigurationsmanagement mit Subversion, Ant und Maven: Ein Praxishandbuch für Softwarearchitekten und Entwickler. 1. Aufl., Heidelberg: dpunkt.verl., 2006

Regensteiner, J. G. et al., 1997: Hospital vs home-based exercise rehabilitation for patients with peripheral arterial occlusive disease, in: Angiology 48 (1997), S. 291–300, 9112877 / http://www.gopubmed.org/search?q=9112877&t=endnote

Rudlof, Christiane (Hrsg.), 2006: Handbuch Software-Ergonomie: Usability Engineering. 2. überarbeitete Auflage, 2006. http://www.ukpt.de/pages/dateien/software-ergonomie.pdf

Saake, Gunter / Heuer, Andreas / Sattler, Kai-Uwe (Hrsg.), 2005: Datenbanken: Implementierungstechniken; [Architekturprinzipien, Dateiorganisation & Zugriffsstrukturen, massiv verteilte Datenverwaltung in P2P-Systemen, Illustration der Konzepte am Beispiel aktueller DBMS-Produkte]. 2. aktualisierte und erweiterte Auflage, Bonn: mitp-Verlag, 2005

Spillner, Andreas / Linz, Tilo (Hrsg.), 2007: Basiswissen Softwaretest: Aus- und Weiterbildung zum Certified Tester; Foundation Level nach ISTQB-Standard. 3. überarbeitete und aktualisierte Auflage, korrigierter Nachdruck, Heidelberg: dpunkt Verlag, 2007

Turau, Volker / Saleck, Krister / Lenz, Christopher (Hrsg.), 2004: Web-basierte Anwendungen entwickeln mit JSP2: Einsatz von JSTL, Struts, JSF, JDBC, JDO, JCA. 2. vollständig überarbeitete Auflage, Heidelberg: dpunkt Verlag, 2004

Wullink, M. et al., 2001: A primary care walking exercise program for patients with intermittent claudication, in: Medicine and science in sports and exercise 33 (2001), S. 1629–1634, 11581544 / http://www.gopubmed.org/search?q=11581544&t=endnote

Zeitner, Alfred (Hrsg.), 2008: Spring 2.5: Eine pragmatische Einführung. München: Addison-Wesley, 2008

Anhang I: UseCases

1. Login

Name des UseCase

Login

Subsystem

Dokumentations- und Überwachungssystem

Aktoren

Patient, Arzt

Kontext und Vorbedingungen

-

Normalablauf

1. Das System zeigt den Logindialog an.
2. Der Benutzer trägt seinen Benutzernamen sowie das Passwort in die Textfelder ein.
3. Der Benutzer bestätigt seine Eingaben.
4. Das System überprüft, ob der Benutzer ein Patient oder ein Arzt ist. Ferner wird geprüft, ob der eingegebene Benutzername existiert und das eingegebene Passwort dazu stimmt.

Normalergebnis

Der Benutzer ist erfolgreich eingeloggt und kann entsprechend seinen Rechten im System arbeiten.

Alternativablauf

5a. Das System erkennt, dass der Benutzername nicht korrekt ist oder dass das Passwort nicht mit dem Benutzernamen übereinstimmt.
In diesen Fällen wird eine Fehlermeldung mit dem Hinweis, dass der Benutzername oder das Passwort nicht korrekt eingegeben wurde, angezeigt. Weiter mit 2.

Nicht funktionale Anforderungen

-

Notiz

-

2. Eingabe der Messwerte

Name des UseCase

Eingabe der Messwerte

Subsystem

Dokumentations- und Überwachungssystem

Aktoren

Patient

Kontext und Vorbedingungen

1. Der Patient muss im System angemeldet sein.

Normalablauf

1. Der Patient gibt folgende 6 Parameter/Messwerte in die dafür vorgesehenen Textfelder ein: Schmerzfreie Gehstrecke, Abbruch, Dauer, Blutdruck, Pulsfrequenz und Gewicht
   Des Weiteren muss der aktuelle Tag und die Uhrzeit angegeben werden.
2. Der Patient bestätigt die Speicherung der Daten.
3. Das System prüft die Daten auf Redundanz.
4. Das System speichert die Messwerte, sofern diese nicht redundant sind, und zeigt sie in Form einer Tabelle an.

Normalergebnis

Die Messwerte sind im System gespeichert.

Alternativablauf

3a. Das System teilt dem Patient mit, dass die Messwerte bereits existieren.
    Weiter mit 1.

Nicht funktionale Anforderungen

Da es sich um medizinische Daten handelt, müssen Datensicherheit und -integrität jederzeit gewährleistet sein.

Notiz

1. Hinweis zu den Parametern schmerzfreie Gehstrecke und Abbruch:
   Diese Werte müssen in Meter [m] angegeben werden.
2. Hinweis zum Parameter Abbruch:
   Anzahl Meter von Beginn bis zum Abbruch der Gehstrecke.
3. Hinweis zum Parameter Dauer:
   Dieser Messwert muss in Minuten [min] angegeben werden und umfasst den Zeitraum des Gehens/Laufens.
4. Hinweis zum Parameter Blutdruck:
   Der Blutdruck muss morgens täglich gemessen werden. Systole und Diastole müssen getrennt voneinander eingegeben werden, dafür gibt es 2 Textfelder.
5. Hinweis zur Pulsfrequenz:
   Der Puls muss in „Schläge pro Minute“ angegeben werden.
6. Hinweis zum Gewicht:
   Das Gewicht muss in kg eingegeben werden.

3. Messwerte bearbeiten

Name des UseCase

Messwerte bearbeiten

Subsystem

Dokumentations- und Überwachungssystem

Aktoren

Patient

Kontext und Vorbedingungen

1. Der Patient muss im System angemeldet sein.
2. Es müssen Messwerte vorhanden sein.

Normalablauf

1. Das System zeigt eine Liste mit allen vorhandenen Messwerten an.
2. Der Patient selektiert den verbesserungswürdigen Messwert in der Tabelle.
3. Das System blendet ein PopUp mit vorbelegten Textfeldern ein.
4. Der Patient nimmt Änderungen an den entsprechenden Stellen vor und bestätigt die Änderung.
5. Das System speichert den geänderten Messwert-Datensatz.

Normalergebnis

Der Messwert wurde geändert.

Alternativablauf

5a. Das System speichert die Änderungen nicht. Weiter mit 1.

Nicht funktionale Anforderungen

-

Notiz

-

4. Ausgabe der Messwerte (Arzt)

Name des UseCase

Ausgabe der Messwerte

Subsystem

Dokumentations- und Überwachungssystem

Aktoren

Arzt

Kontext und Vorbedingungen

1. Der Arzt muss im System eingeloggt sein.
2. Im System müssen Messwerte vorhanden sein.

Normalablauf

1. Der Arzt muss einen Patienten aus der Liste selektieren.
2. Das System zeigt die Messwerte in Form von Diagrammen grafisch an.

Normalergebnis

Das System zeigt die Grafiken an.

Alternativablauf

-

Nicht funktionale Anforderungen

-

Notiz

-

5. Ausgabe der Messwerte (Patient)

Name des UseCase

Ausgabe der Messwerte

Subsystem

Dokumentations- und Überwachungssystem

Aktoren

Patient

Kontext und Vorbedingungen

1. Der Patient muss im System eingeloggt sein.
2. Im System müssen Messwerte vorhanden sein.

Normalablauf

1. Das System zeigt die Messwerte in Form von Diagrammen grafisch an.

Normalergebnis

Das System zeigt die Grafiken an.

Alternativablauf

-

Nicht funktionale Anforderungen

-

Notiz

-

6. Alarm auslösen

Name des UseCase

Alarm auslösen

Subsystem

Dokumentations- und Überwachungssystem

Aktoren

System (automatisch)

Kontext und Vorbedingungen

1. Es müssen Messwerte im System vorhanden sein.

Normalablauf

1. Das System parst die Regeln und löst ggf. einen Alarm aus, wenn die Messwerte über einen Schwellenwert liegen, indem es eine E- Mail an das Postfach des Arztes schickt.

Normalergebnis

Der Alarm wurde ausgelöst und der Arzt hat eine E- Mail erhalten.

Alternativablauf

-

Nicht funktionale Anforderungen

-

Notiz

-

7. Patient hinzufügen

Name des UseCase

Patient hinzufügen

Subsystem

Dokumentations- und Überwachungssystem

Aktoren

Arzt

Kontext und Vorbedingungen

1. Der Arzt muss im System angemeldet sein.

Normalablauf

1. Der Arzt trägt Vor- und Nachname, Benutzername, Passwort, E-Mailadresse, Geburtsdatum und Protokoll in die dafür vorgesehenen Textfelder ein.
2. Der Arzt bestätigt die Speicherung der Daten.
3. Das System prüft die Daten auf Redundanz, Vollständigkeit und Richtigkeit.
4. Das System speichert den neu erstellten Patienten.

Normalergebnis

Der neu erstellte Patient ist im System registriert.

Alternativablauf

3a. Das System teilt dem Arzt mit, dass der Patient bereits existiert. Weiter mit 1.

3b. Der Arzt hat nicht formgerechte oder unvollständige Eingaben gemacht. Das System wirft daraufhin eine geeignete Fehlermeldung. Weiter mit 1.

Nicht funktionale Anforderungen

-

Notiz

1. Die E-Mailadresse muss dem gängigen Format (max.mustermann@web.de) entsprechen.
2. Als Protokoll muss „Gehstrecke“ angegeben werden.

8. Arzt hinzufügen

Name des UseCase

Arzt hinzufügen

Subsystem

Dokumentations- und Überwachungssystem

Aktoren

Arzt

Kontext und Vorbedingungen

1. Der Arzt muss im System angemeldet sein.

Normalablauf

1. Der Arzt trägt Vor- und Nachname, Benutzername, Passwort und E-Mailadresse in die dafür vorgesehenen Textfelder ein.
2. Der Arzt bestätigt die Speicherung der Daten.
3. Das System prüft die Daten auf Redundanz, Vollständigkeit und Richtigkeit.
4. Das System speichert den neu erstellten Arzt.

Normalergebnis

Der neu erstellte Arzt ist im System registriert.

Alternativablauf

3a. Das System teilt dem Arzt mit, dass der Arzt bereits existiert.
Weiter mit 1.

3b. Der Arzt hat nicht formgerechte oder unvollständige Eingaben gemacht. Das System wirft daraufhin eine geeignete Fehlermeldung.
Weiter mit 1.

Nicht funktionale Anforderungen

-

Notiz

1. Die E-Mailadresse muss dem gängigen Format (max.mustermann@web.de) entsprechen.

9. Benutzerkonto bearbeiten

Name des UseCase

Benutzerkonto bearbeiten

Subsystem

Dokumentations- und Überwachungssystem

Aktoren

Arzt

Kontext und Vorbedingungen

1. Der Arzt muss im System angemeldet sein.
2. Es müssen Benutzer vorhanden sein.

Normalablauf

1. Das System zeigt eine Liste mit allen registrierten Benutzern an.
2. Der Arzt wählt einen Benutzer, dessen Konto er bearbeiten möchte, aus der Liste aus.
3. Das System blendet ein PopUp mit vorbelegten Textfeldern ein.
4. Der Arzt nimmt Änderungen an den entsprechenden Stellen vor und bestätigt die Änderung.
5. Das System speichert den geänderten Benutzer.

Normalergebnis

Der Benutzer wurde bearbeitet.

Alternativablauf

5a. Das System speichert die Änderungen nicht. Weiter mit 1.

Nicht funktionale Anforderungen

-

Notiz

-

10. Benutzerkonto löschen

Name des UseCase

Benutzerkonto löschen

Subsystem

Dokumentations- und Überwachungssystem

Aktoren

Arzt

Kontext und Vorbedingungen

1. Der Arzt muss im System angemeldet sein.
2. Es müssen Benutzer vorhanden sein.

Normalablauf

1. Das System zeigt eine Liste mit allen registrierten Benutzern an.
2. Der Arzt wählt einen Benutzer, dessen Konto er entfernen möchte, aus der Liste aus.
3. Das System blendet ein PopUp mit vorbelegten Textfeldern ein.
4. Der Arzt betätigt die Entfernung.
5. Das System löscht den Benutzer.

Normalergebnis

Der Benutzer wurde vom System entfernt.

Alternativablauf

5a. Das System entfernt das Benutzerkonto nicht. Weiter mit 1.

Nicht funktionale Anforderungen

-

Notiz

-

11. Nachricht schreiben

Name des UseCase

Nachrichten schreiben

Subsystem

Dokumentations- und Überwachungssystem

Aktoren

Arzt, Patient

Kontext und Vorbedingungen

1. Der Akteur muss im System angemeldet sein.

Normalablauf

1. Das System zeigt eine Seite zum Nachrichten schreiben und weitere Felder zum Festlegen der Sendedaten an.
2. Der Akteur legt die notwendigen Sendedaten, wie Datum, Uhrzeit, Absender und Empfänger fest und gibt einen Text ein und sendet diesen ab.
3. Das System zeigt die Nachricht, nach Datum gruppiert, in einer Tabelle auf derselben Seite an.

Normalergebnis

Die Nachricht wurde erfolgreich geschrieben und abgesendet.

Alternativablauf

-

Nicht funktionale Anforderungen

-

Notiz

-

12. Logout

Name des UseCase

Logout

Subsystem

Dokumentations- und Überwachungssystem

Aktoren

Patient, Arzt

Kontext und Vorbedingungen

1. Es muss ein Benutzer eingeloggt sein.

Normalablauf

1. Der Akteur wählt den „Logout“-Button auf der Startseite aus.
2. Das System loggt den Benutzer erfolgreich aus und sichert dessen Änderungen.
3. Das System zeigt den Logindialog an.

Normalergebnis

Der Benutzer ist erfolgreich abgemeldet und seine Änderungen gespeichert.

Alternativablauf

-

Nicht funktionale Anforderungen

-

Notiz

-

Anhang II: Verwendete Lizenzen

Das Dokumentations- und Überwachungssystem basiert größtenteils auf Open-Source Produkten. Dabei verwenden die verschiedenen Technologien unterschiedliche Lizenzen. An dieser Stelle sollen die verwendeten Lizenzen aufgeführt und kurz beschrieben werden.

Apache License

Apache License v 2.0	
Web:	http://www.apache.org/licenses/LICENSE-2.0

Tabelle 5: Apache License

Die Apache-Lizenz ist eine Lizenz für freie Software und wurde von der Apache Software Foundation entwickelt. Seit 2004 existiert diese in der Version 2.0 [Apache License]. Eine Software, die unter der Apache-Lizenz steht, kann in jedem Umfeld frei verwendet werden. Ferner können Modifizierungen am Quellcode vorgenommen werden, ohne dass diese zurückgegeben werden müssen.

Das Dokumentations- und Überwachungssystem verwendet folgende unter der Apache-Lizenz stehende Software:

- Google Web Toolkit
- Maven
- Spring Framework

Vorteilhaft ist, dass das Dokumentations- und Überwachungssystem trotz der Verwendung von Software, die unter der Apache-Lizenz steht, nicht selbst unter der Apache-Lizenz stehen muss. Sofern das Dokumentations- und Überwachungssystem verteilt wird, muss eindeutig darauf hingewiesen werden, welche Bestandteile unter der Apache-Lizenz stehen. In diesem Fall muss eine Kopie der Lizenz beiliegen.

GNU General Public License

GNU General Public License v 3.0	
Web:	http://www.gnu.org/licenses/gpl.html

Tabelle 6: GNU General Public License

Die GNU General Public License wurde von der Free Software Foundation, welche 1984 von Richard Stallmann gegründet wurde, herausgegeben. Diese ermöglicht die Lizenzierung freier Software und ist derzeit in der Version 3.0 verfügbar. Die Lizenz garantiert, dass eine Software ohne Einschränkung für jeden Zweck genutzt werden darf. Darüber hinaus können beliebig viele Kopien verbreitet werden. Programme, die solchen Code enthalten, müssen dieser Lizenz unterliegen. Eine Verwendung der GNU General Public License ist nur möglich, wenn der eigene Source Code offengelegt wird [GNU Operating System].

Das Ext GWT, welches zusätzlich für die Implementierung der Oberfläche verwendet wurde, steht unter dieser Lizenz.

Common Public License

Common Public License v 1.0	
Web:	http://www.opensource.org/licenses/cpl1.0.php

Tabelle 7: Common Public License

Diese Lizenz ist eine Lizenz für freie Software, die von der Eclipse Public License abgelöst wurde. Die Common Public License kann jedoch weiterhin verwendet werden. Ebenso wie ihre beiden Vorgänger erlaubt diese Lizenz, dass eine Software, die unter dieser Lizenz steht, frei genutzt, geändert und verteilt werden kann [Open-Source Initiative].

Das Dokumentations- und Überwachungssystem verwendet JUnit, das unter dieser Lizenz steht. Trotz der Verwendung einer solchen Lizenz ist es nicht erforderlich, dass das Dokumentations- und Überwachungssystem unter der Common Public License stehen muss.

PostgreSQL Licence

PostgreSQL Licence	
Web:	http://www.opensource.org/licenses/postgresql

Tabelle 8: PostgreSQL Licence

Die PostgreSQL Licence ist eine Open-Source-Lizenz. Diese Lizenz erlaubt, dass eine Software ohne Kosten verwendet, kopiert, geändert und verteilt werden kann, sofern eine Kopie dieser beiliegt [PostgreSQL].

Eidesstattliche Erklärung

Lang, Sabrina 164790

Thema der Diplomarbeit:

Computer-gestütztes, internet-basiertes Dokumentations- und Überwachungssystem für medizinisches Gehtraining bei Patienten mit peripherer arterieller Verschlusskrankheit

Ich erkläre hiermit an Eides Statt, dass ich die vorliegende Arbeit selbstständig und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt habe; die aus fremden Quellen (einschließlich elektronischer Quellen) direkt oder indirekt übernommenen Gedanken sind als solche kenntlich gemacht. Bei der Auswahl und Auswertung des Materials sowie bei der Herstellung des Manuskripts habe ich Unterstützungsleistung von folgenden Personen erhalten:

Die Arbeit wurde bisher weder im Inland noch im Ausland in gleicher oder ähnlicher Form einer anderen Prüfungsbehörde vorgelegt und ist auch nicht veröffentlicht.

Waldstetten, den 25.11.2010