Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Bachelorarbeit

Nicolas With

Testbett für eine kriteriengestützte Analyse von Webframeworks

Fakultät Technik und Informatik
Department Informatik

Faculty of Engineering and Computer Science
Department of Computer Science

Nicolas With

Testbett für eine kriteriengestützte Analyse von Webframeworks

Bachelorarbeit eingereicht im Rahmen der Bachelorprüfung
im Studiengang Angewandte Informatik
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer : Prof. Dr. Olaf Zukunft
Zweitgutachter : Prof. Dr. Stefan Sarstedt

Abgegeben am 26. August 2010

Nicolas With

Thema der Bachelorarbeit

Testbett für eine kriteriengestützte Analyse von Webframeworks

Stichworte

JavaServer Faces, Google Web Toolkit, PHP, Metriken, Softwarequalität, Webframework, Maintainability Index

Kurzzusammenfassung

In dieser Arbeit werden drei Webframeworks mit Hilfe einer Testumgebung analysiert, ausgewertet und verglichen. Anhand eines erstellten Szenarios werden Anwendungen von den jeweiligen Frameworks realisiert und mit den vorher definierten Kriterien gemessen. Am Ende werden die Frameworks anhand der Ergebnisse der Tests in bestimmte Anforderungsbereiche eingeordnet.

Nicolas With

Title of the paper

Testbed for a criteria based Analysis of Web Application Frameworks

Keywords

JavaServer Faces, Google Web Toolkit, PHP, Metrics, Software quality, Web Application Framework, Maintainability Index

Abstract

In this paper three Web Application Frameworks will be tested, evaluated and compared via a test environment. The applications created with the frameworks will be realised on the basis of an constructed scenario and measured with previously defined criterias. At the end the frameworks will be classified in certain areas of specifications on the basis of the results that occured from the tests.

Danksagung

An dieser Stelle möchte ich allen Menschen meinen Dank aussprechen, die mir geholfen und mich dabei unterstützt haben diese Arbeit zu erstellen.

Als erstes möchte ich mich bei meinem Betreuer Prof. Dr. Olaf Zukunft bedanken, der mich während meiner Bachelorarbeit unterstützt und mir wertvolle Tipps gegeben hat.

Weiterer Dank geht an meine Brüder und meine Eltern für ihre Unterstützung.

Nicolas With, August 2010

Inhaltsverzeichnis

Tabellenverzeichnis

Abbildungsverzeichnis

1 Einleitung

1.1 Motivation

Das World Wide Web ist der am meisten genutzte Dienst im Internet. Wurden vor knapp 15 Jahren nur simple statische Seiten über die Internetleitung übertragen und das WWW vor allem nur für Austausch in Foren und dem eigenen Präsentieren über Home Pages gebraucht, so gibt es heute schier überwältigende Nutzungsmöglichkeiten. War der Browser in früheren Zeiten nur eines von vielen Tools auf dem PC, dessen Zweck es war Internetseiten anzuzeigen, so ist er heutzutage wohl das wichtigste Programm auf dem Computer. Mit ihm lassen sich mittlerweile Videos angucken, Stadpläne und Routenplaner aufrufen und sogar komplexe Desktopanwendungen, wie z.B. Ego Shooter, darstellen. Es ist nicht zu übersehen, dass das WWW immer dynamischer wird. Mit den Standardwerkzeugen HTML, CSS und JavaScript lassen sich solche dynamischen und komplexen Webanwendungen nur sehr umständlich umsetzen. So wurden zahlreiche Frameworks entwickelt, welche auf verschiedenen Wegen dem Programmierer helfen, dynamische und komplexe Webseiten zu erstellen.

Webframeworks geben Entwickler ein Programmiergerüst, sie geben Entwurfsmuster vor und stellen Schnittstellen und Bibliotheken bereit, mit denen der Entwickler seine Anwendung realisieren kann. Der Anwendungsaufbau wird dem Programmierer über das Framework vorgegeben, er wird sozusagen über Schienen ans Ziel geleitet. Dabei ist der Weg, der vorgegeben wird, von Framework zu Framework unterschiedlich. Verschiedene Frameworks geben unterschiedliche Entwurfsmuster vor, die eine andere Architektur der Anwendung voraussetzen. Es werden unterschiedliche Schnittstellen und Bibliotheken bereit gestellt, die eine andere Herangehensweise an bestimmte Probleme zur Folge haben. Manche Frameworks geben einen eher starren Entwicklungsablauf vor, der wenig Freiheiten lässt, andere sind flexibler und geben einen größeren Raum zur Entfaltung. Und vor allem unterscheiden sich die Frameworks sehr stark in der Sprache auf der sie aufbauen, dies können neue, eigens entwickelte Websprachen sein oder ältere Programmiersprachen, die um neue Aspekte erweitert wurden.

Diese Vielfalt an Framworks sorgt für ein Überangebot und die Auswahl des Frameworks für eine bestimmte Webanwendung hängt meist nur von den persönlichen Präferenzen ab. Doch

diese Vielfalt bedeutet auch, dass manche Frameworks für bestimmte Anwendungsbereiche besser geeignet sind müssen und manche schlechter, da ansonsten ein Framework reichen würde, um das gesamte Gebiet abzudecken. Um zu sehen, wo ein Framework eingeordnet werden kann, müssen die Frameworks nach bestimmten Gesichtspunkten analysiert und die Ergebnisse verglichen werden. Einen Ansatz dazu wird in der folgenden Arbeit vorgestellt.

1.2 Aufgabenstellung

Die Arbeit bietet einen Ansatz, Anwendungen eines Frameworks mit bestimmten Kriterien zu testen und auszuwerten. Dabei müssen Metriken erstellt werden, um die Kriterien messen zu können. Ein Szenario hilft dabei, alle Frameworks unter den gleichen Voraussetzungen zu testen. Anhand der ausgewählten Kriterien wird am Ende eine Testumgebung erstellt, in der die erstellten Anwendungen analysiert und die verwendeten Frameworks eingeordnet werden.

1.3 Aufbau der Arbeit

Die Arbeit gliedert sich grob in 5 Kapitel. Im ersten Kapitel werden die einzelnen Kriterien vorgestellt, nach denen ein Framework bewertet werden kann. Dabei sind manche Kriterien besser geeignet als andere. Vor allem sind die Kriterien gut geeignet, die sich leicht messen lassen und ein Ergebnis haben, welches sich vergleichen lässt, um es besser einordnen zu können.
Im zweiten Kapitel wird das verwendete Szenario schematisch dargestellt. Das Szenario hilft dabei eine Basis für das Testbett zu haben, um eine Analyse unter gleichen Voraussetzungen zu schaffen.
Im dritten Kapitel werden die Frameworks vorgestellt, die analysiert werden sollen. Dabei wurden die Frameworks so ausgewählt, dass möglichst neue, weit verbreitete und typische, dass heißt keine exotischen, Technologien vertreten sind, um ein möglichst wirklichkeitsgetreues Bild wiederzugeben.
Im vierten Kapitel wird eine Anwendung, angelehnt an das beschriebene Szenario, erstellt. Dabei wird auf die verwendete Technologie, die Implementierung und auf etwaige aufgetretene Probleme eingegangen.
Zuletzt werden die Frameworks anhand des erstellten Anwendung und der gewählten Kriterien mit Hilfe eines erstellten Testbetts analysiert. Den Schlusspunkt stellt die Auswertung der Ergebnisse der Analyse dar.

2 Metriken

2.1 GQM - Goal-Question-Metric

Um Kriterien sinnvoll messen und bewerten zu können braucht man Metriken, die helfen eine quantifizierte Aussage zu treffen. Ein Ansatz, um Metriken herzuleiten, ist der Goal-Question-Metric Ansatz. Der Ansatz wurde schon 1984 von Basili und Weis entwickelt und in den Folgejahren immer weiter erprobt und weiterentwickelt. Wie der Name schon vermuten lässt, besteht der GQM aus drei Schritten: Ludewig und Lichter (2010)

1. Definiere die für ein Projekt oder für eine Organisation relevanten Ziele (Goal)
2. Leite aus jedem Ziel Fragen ab (Question), die geklärt werden müssen, um überprüfen zu können, ob das Ziel erreicht ist
3. Lege für jede Frage Metriken fest (Metric), die dazu beitragen, die Antworten auf die Fragen zu finden

Als erster Schritt wird ein Ziel festgelegt. Für dieses Ziel werden möglichst konkrete und einfache Fragen aufgestellt, wenn die Frage zu abstrakt ist, muss die Frage weiter verfeinert werden. Schlussendlich wird zu jeder Frage eine Metrik festgelegt, dabei kann natürlich eine Metrik mehrere Fragen abdecken, wie auch eine Frage mehrere Metriken abdecken kann, quasi ein n <-> n Beziehung. Wie das aussehen kann, veranschaulicht die Abbildung 2.1.

Nachdem die Metriken zu den einzelnen Fragen definiert worden sind, werden die Daten, die durch die Metrik gesammelt werden, systematisch erfasst und ausgewertet. Im besten Fall wird es anhand der gesammelten Daten möglich sein Vergleiche anzustellen.

2.2 Kriterien

Um eine geeignete Basis aufzustellen, die Frameworks zu untersuchen und einzuordnen, werden verschiedene Kriterien zur Hilfe genommen, an denen man die einzelnen Frameworks testen kann. Ein Framework kann ein Kriterium gut oder weniger gut erfüllen und wird

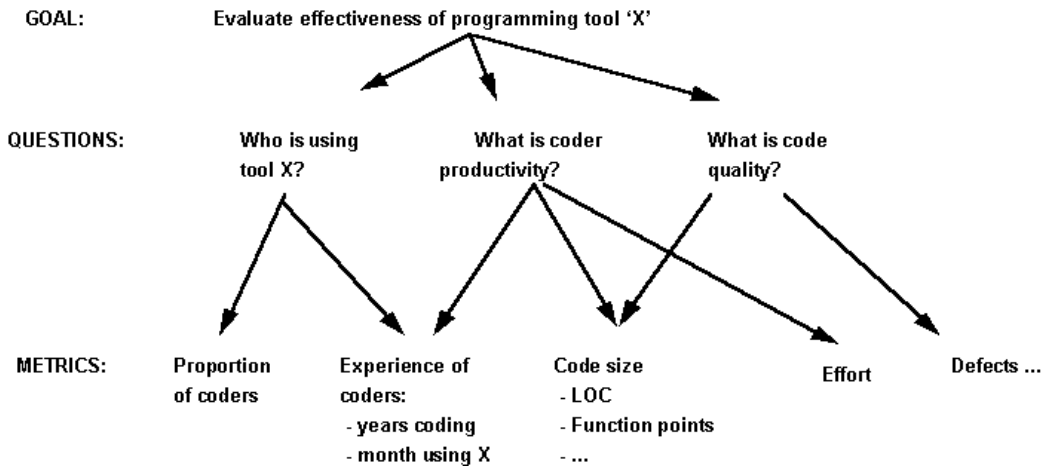


Abbildung 2.1: Beispiel eines GQM-Baums Fenton (1999)

so für eine bestimmte Aufgabe besser oder schlechter einzusetzen sein. Danach ist es möglich für die einzelnen Frameworks eine Art „Schablone“ zu erstellen, um dann einfacher zu erkennen, ob das Framework für die bevorstehende Arbeit geeignet ist.

Während die Qualität eines Endproduktes aus Benutzersicht und Entwicklersicht bewertet werden kann, können die behandelten Frameworks fast ausschließlich aus Entwicklersicht den einzelnen Kriterien gut oder weniger gut zugeordnet werden, da die Frameworks im Webbereich alle die gleiche Ausgabe liefern sollten und „nur“ die Art der Programmierung sich teilweise stark voneinander unterscheidet.

Im Folgenden werden die für die Analyse der Frameworks benutzten Kriterien aufgelistet, dabei wird jedes Kriterium in drei Punkten erläutert werden. Erstens eine allgemeine Definition und Einordnung, zweitens wird auf die Relevanz und die Frage „Warum dieses?“ eingegangen und drittens wird mit Hilfe des GQM-Ansatzes versucht eine geeignete Metrik zu entwickeln.

2.2.1 Änderbarkeit

Änderbarkeit fällt in den Bereich der Wartbarkeit. Änderbarkeit beschreibt das Maß an Aufwand, den man betreiben muss, um einen Fehler zu beheben oder eine bestimmte Sache neu zu gestalten oder zu verändern. Genauer gesagt beschreibt der Aufwand die Geschwindigkeit der Änderung, den Umfang der Änderung (Lines of Code) und die Reichweite der Änderung, was besagt wie viele Bereiche geändert werden müssen, um eine Änderung durchzuführen. Unterbereiche sind SwtWiki (2008c)

- **Analysierbarkeit:** Aufwand, um Mängel oder Fehler zu diagnostizieren und den Änderungsumfang zu bestimmen
- **Modifizierbarkeit:** Aufwand, zur Verbesserung oder Umgebungsanpassung
- **Stabilität:** Wahrscheinlichkeit des Auftretens unerwarteter Wirkung von Änderungen
- **Prüfbarkeit:** Aufwand, der zur Prüfung der Änderung notwendig ist (siehe auch Testbarkeit)

Änderbarkeit ist für jede Art von Programmierarbeit von höchster Wichtigkeit. Änderungen am Code müssen ständig vollbracht werden, seien es kleine Änderungen, die kaum Einfluss auf die Umgebung haben oder größere Änderungen, bei der komplette Teile des Codes umgearbeitet werden müssen. Eine gutes Framework zeichnet sich dadurch aus, dass Änderungen nur die Teile des Programms beeinflussen, die auch wirklich mit der Änderung zu tun haben und der Aufwand auf ein Minimum reduziert werden muss.

Änderungen zu messen ist nicht ganz einfach, da sich Änderungen stark voneinander unterscheiden können. Trotzdem wurde eine Metrik entwickelt, welche die Änderbarkeit einer Software in einer Zahl ausdrückt. Der Maintainability Index (MI) wurde vor einiger Zeit in den USA definiert. Er verwendet die Zeilenmetriken (LOC), die zyklomatische Zahl, die von McCabe entwickelt wurde und die Halstead-Metrik. Aus diesen Grundlagen heraus wird, im besten Fall, eine Zahl zwischen 0 - 100 berechnet, welche die Güte der Änderbarkeit beschreibt, wobei alles unter 65 (sogar negative Zahlen sind möglich) für eine schlechte Änderbarkeit steht und eine Zahl über 85 für eine gute Änderbarkeit. Verifysoft Technology (2010)

Die erste und gebräuchlichste Messung zur Komplexität einer Software liegt in der Zeilenmetrik, bei welchem man einfach die Lines of Code (LOC) des Programms zählt. Es gibt aber verschiedene Zeilenmetriken, als da wären

- **LOCphy:** Anzahl der physikalischen Zeilen (Number of physical lines);
- **LOCpro:** Anzahl der Programmzeilen (Number of program lines): Deklarationen, Definitionen, Direktiven und Code;
- **LOCcom:** Anzahl der Zeilen mit Kommentaren (Number of commented lines);
- **LOCbl:** Anzahl der Leerzeilen (blank lines), Hinweis: Leerzeilen innerhalb eines Kommentarblocks werden als Kommentarzeilen gezählt.

Eine weitere Metrik ist die von Thomas J. McCabe, Sr. im Jahre 1976 entwickelte zyklomatische Zahl. Die Zahl, abgekürzt mit v(G), zeigt die Komplexität des Kontrollflusses des Programms an. Für eine rein aus sequentiellen Befehlen bestehendes Programm beträgt v(G) = 1, für jede weitere Verzweigung erhöht sich v(G) um 1. Sprachkonstukte, welche v(G) erhöhen sind if (...), for (...), while (...), case ...:, catch (...), &&, ||, ?. Als grobe Richtlinie gilt, dass eine Funktion kein v(G) > 15 haben sollte.

1977 hat Maurice Howard Halstead eine weiteres Maß für die Komplexität von Software entwickelt. Die Metrik beruht direkt auf der Anzahl der Operatoren und Operanden und steht daher direkt im Bezug zum Programmcode. Es gibt verschiedene Halstead-Metriken, hier eine Auflistung:

- N - Programmlänge (Program length)
- n - Vokabulargröße (Vocabulary size)
- V - Volumen des Programms (Program volume)
- D - Schwierigkeitsgrad (Difficulty level)
- L - Programmniveau (Program level)
- E - Implementierungsaufwand (Effort to implement)
- T - Implementierungszeit (Implementation time)
- B - Anzahl der ausgelieferten Bugs (Number of delivered bugs)

So wird zum Beispiel die Programmlänge N über die Summe von der Gesamtzahl der Operatoren N1 und der Gesamtzahl der Operanden N2 berechnet, also N = N1 + N2. Die Vokabulargröße n wird über de Summe der verschiedenen Operatoren n1 und der verschiedenen Operanden n2 berechenet, also n = n1 + n2. aus der Programmlänge N und der Vokabulargröße n lässt sich nun das Programmvolumen berechnen, der auf der Anzahl der durchgeführten Operationen und der bearbeiteten Operanden beruht. Er berechnet sich mit V = N * $\log_2(n)$.

Aus den drei oben beschriebenen Metriken lässt sich nun der Maintainability Index berechnen. Zuerst wird der MI ohne Kommentare berechnet:

$$MIwoc = 171 - 5.2 * ln(aveV) - 0.23 * aveG - 16.2 * ln(aveLOC) \tag{2.1}$$

Hier ist aveV das durchschnittliche Halstead-Volumen (V), aveG ist die durchschnittliche zyklomatische Komplexität v(G), aveLOC ist die durchschnittliche Anzahl der Programmcodezeilen (LOCphys). Dann wird der MI der Kommentare berechnet:

$$MIcw = 50 * sin(sqrt(2.4 * perCM)) \tag{2.2}$$

Hier ist perCM die durchschnittliche Anzahl der Zeilen mit Kommentaren. Schlussendlich wird der eigentliche MI berechnet, der die Summe der beiden vorhergehenden Metriken bildet: MI = MIwoc + MIcw.

2.2.2 Funktionalität

Die Funktionalität einer Anwendung beschreibt inwiefern sie die geforderten Funktionen erfüllt. Funktionalität definiert sich in den folgenden Punkten: Wikipedia (2010d)

- **Angemessenheit:** Eignung von Funktionen für spezifizierte Aufgaben, z. B. aufgabenorientierte Zusammensetzung von Funktionen aus Teilfunktionen.
- **Richtigkeit:** Liefern der richtigen oder vereinbarten Ergebnisse oder Wirkungen, z. B. die benötigte Genauigkeit von berechneten Werten.
- **Interoperabilität:** Fähigkeit, mit vorgegebenen Systemen zusammenzuwirken.
- **Sicherheit:** Fähigkeit, unberechtigten Zugriff, sowohl versehentlich als auch vorsätzlich, auf Programme und Daten zu verhindern.
- **Ordnungsmäßigkeit:** Merkmale von Software, die bewirken, dass die Software anwendungsspezifische Normen oder Vereinbarungen oder gesetzliche Bestimmungen und ähnliche Vorschriften erfüllt.

Funktionalität ist wichtig für eine Anwendung, da unterschiedliche Sprachen verschiedene Möglichkeiten geben eine Funktion zu implementieren. Dadurch ist auch das erzielte Ergebnis ein anderes und je nachdem, wie die Anforderung an die Funktionsweise war, ist die Erfüllung der Anforderung von unterschiedlicher Güte.

Um die Funktionalität einer Anwendung zu messen, muss man das Anforderungsprofil in die Messung mit einbeziehen. Umso genauer das Anforderungsprofil erfüllt wird, umso größer ist der gemessene Wert. Es wird mit einbezogen, ob die Anforderungen allgemein überhaupt erfüllt werden können und wie genau sie erfüllt werden können. Weiter muss berücksichtigt werden, ob die Funktionen in dem geforderten Maße mit den schon vorhandenen Einstellungen des Systems erfüllt werden können, das heißt, dass die Messung mit verschiedenen Einstellungen des Systems vorgenommen werden muss, um sicherzustellen, wie groß die Interoperabilität der Anwendung mit dem System ist.

2.2.3 Effizienz

Effizienz beschreibt eine Kosten/Nutzen-Relation und demzufolge ein Verhältnis zwischen dem erzieltem Resultat und den eingesetzten Mitteln. Bei dem Einsatz von Effizienz ist es das Ziel bei hoher Wirtschaftlichkeit einen möglichst geringen Aufwand zu verursachen. Somit erfolgt eine Bewertung der Angemessenheit und man muss sich fragen: „Mache ich die Dinge richtig?“. SwtWiki (2008a)

Merkmale von Effizienz:

- **Zeitverhalten:** Benötigte Reaktions- und Bearbeitungszeiten der Software, wie z.B. Laufzeiten von Algorithmen
- **Verbrauchsverhalten:** Anzahl der verwendeten Ressourcen zur Ausführung der Software, wie z.B. Speicherverbrauch oder Festplattenzugriffe
- **Konformität:** Grad, in dem die Software Normen oder Vereinbarungen zur Effizienz erfüllt.

Nicht zu verwechseln, aber eng verbunden, ist Effizienz („Die Dinge richtig tun“) mit Effektivität („Die richtigen Dinge tun“). Demnach wäre es Verschwendung die falschen Dinge effizient durchzuführen.

In einer Informatikwelt in der schneller immer besser bedeutet, ist es natürlich auch bei Webanwendungen von größter Wichtigkeit, schnell Ergebnisse auf den Browser zu bringen und Anfragen schnell zu bearbeiten. Niemand will lange auf eine Antwort warten müssen, deswegen muss eine gute Balance zwischen Funktionalität und Schnelligkeit der Anwendung erreicht werden. Genauso verhält es sich mit dem Ressourcenverbrauch. Ein großes Plus von Webanwendungen besteht darin, dass sie auf jeder Hardware aufgerufen werden können, solange eine Internetverbindung besteht. Diese Geräte können von großen Computern, mit viel Speicher und CPU-Takt, bis hin zu kleinsten Smartphones, mit nur begrenzter Hardwareleistung, reichen und somit muss sichergestellt sein, dass die Anwendung auf allen Geräten zufriedenstellend läuft. Wiederum gilt es eine gutes Gleichgewicht zwischen Umfang der Anwendung und Ressourcenanspruch zu erreichen.

Für den ersten Punkt bietet es sich an einen Benchmark zur Messung der Geschwindigkeit der Anwendung und des damit verbundenen Frameworks anzuwenden. Ein Benchmark ist ein Programm, welches eine Anwendung oder nur einen einzelnen Algorithmus mit verschiedenen Einstellungen prüfen kann, damit man eine Grundlage zum Vergleich hat. Um zu ermitteln, wie schnell ein Programm auf ein Request reagiert und das gewünschte Ergebnis angezeigt wird, ist es erforderlich den Zeitraum von der Bearbeitung der Anfrage bis zum Anzeigen des Response des Servers zu messen.

Als Metrik zum Verbrauchsverhalten der Anwendung bietet es sich an Lasttests als Messgrundlage zu nehmen. Lasttests messen, wie bei ausgewählten Algorithmen die Ressourcen in Anspruch genommen werden. Anders als beim Benchmark, wo die Geschwindigkeit der Algorithmen gemessen wird, wird bei den Lasttests unter anderem der Speicherverbrauch durch Stacks und Heaps und die CPU-Auslastung durch gute oder schlechte Prozessausnutzung gemessen. Ähnlich wie beim Benchmark wird dann beim Lasttest eine Messlatte angelegt, um die Ergebnisse als einfache Integerzahlen zu präsentieren. Wikipedia (2010b)

Beide Messungen erfolgen mit unterschiedlichen Einstellungen, die es uns ermöglicht, eine geeignte Messgrundlage zu erreichen, diese sind

- Worst-Case - das schlechtmöglichste Verhalten
- Average-Case - das durchschnittliche Verhalten
- Best-Case - das bestmöglichste Verhalten

Aufrgrund dieser Einstellungen kann man sehen, wie sich das Programm bei unterschiedlichen Einstellungen verhält.

2.2.4 Testbarkeit

Testbarkeit ist ein sehr wichtiges Kriterium für jede Art von Programmiersprache und Software, da immer Tests geschrieben werden müssen, um den reibungslosen Ablauf eines Programms, auch nach häufiger Änderung, schnell und sicher zu gewährleisten.

Darunter fallen die Verfügbarkeit von Tests in dem Framework.
Gute Einbindung der Tests in dem Framework.
Schnelle und unkomplizierte Ausführung der Tests.
Schnelle und einfache Änderung der Tests, wenn nötig.
Übersichtlich und klar gestaltetes Ergebnis der Tests, um wenn nötig Fehler schnell finden und beheben zu können.

Testen nimmt in der modernen Softwareentwicklung eine primäre Rolle ein. Wenn auch schon früher als wichtig erachtet, werden Tests für Software heutzutage als notwendiges Kriterium gesehen. Die moderne Softwareentwicklung sieht vor, Tests schon vor der Anwendung zu formulieren, um dann den eigentlichen Algorithmus nach den Anforderungen des Tests zu implementieren, damit der Test erfolgreich verläuft. Testbarkeit einer Anwendung zu messen ist schon eine wesentlich komplexere Aufgabe. Aufgrund der noch nicht vollkommenen Unterstützung von Tests in verschiedenen Programmiersprachen ist es erstmal schwierig eine gleichmäßige Messgrundlage für den Vergleich der Frameworks zu bekommen. Da verschiedene Frameworks unterschiedliche Sprachen benutzen und die jeweiligen Sprachen unterschiedlich stark ausgeprägte Testumgebungen haben, erweist sich eine einheitlich angewandte Metrik als von vornherein schwierig. Als zweites ist die Testbarkeit von Webframeworks in der Hinsicht erschwert, als dass das Frontend der Anwendung, in den meisten Fällen HTML/CSS/JavaScript, schwer oder gar nicht zu testen ist. Also muss man sich bei der Testbarkeit eher auf das Backend konzentrieren, welches durch die einzelne Testunterstützung besser zur Messung dient. Doch um eine einheitliche Messgrundlage zu gewährleisten, werden nicht die Anwendbarkeit der Tests, sondern nur die Funktionalität der Tests zur Metrik hinzugezogen.

Es gibt unterschiedliche Faktoren, um die Testbarkeit von Software zu bestimmen. Wikipedia (2010h)

- **Kontrollierbarkeit:** Das Testobjekt kann in den für den Test erforderlichen Zustand gebracht werden.
- **Beobachtbarkeit:** Das Testergebnis kann beobachtet werden.
- **Isolierbarkeit:** Das Testobjekt kann isoliert getestet werden.
- **Trennung der Verantwortlichkeit:** Das Testobjekt hat eine wohldefinierte Verantwortlichkeit.
- **Verständlichkeit:** Das Testobjekt ist selbsterklärend bzw. gut dokumentiert.
- **Automatisierbarkeit:** Die Tests lassen sich automatisieren.
- **Heterogenität:** Unterschiedliche Technologien erfordern den gleichzeitigen Einsatz von unterschiedlichen Testverfahren und -Werkzeugen.

2.2.5 Zuverlässigkeit

Zuverlässigkeit ist der Umfang, in dem von einem System erwartet werden kann, dass es die beabsichtigte Funktion mit der erforderlichen Genauigkeit ausführt. Sie umfaßt Korrektheit, Genauigkeit und Ausfallsicherheit. Ludewig und Lichter (2010)

Eine Metrik, um eine quantifizierte Aussage über Zuverlässigkeit zu treffen, wäre die mittlere Dauer zwischen zwei Fehlern zu messen. Dieses Maß wird auch *Mean Time Between Failures* (MTBF) genannt. Einfach ausgedrückt berechnet sich die MTBF wie folgt:

$$MTBF = \frac{Betriebsdauer}{Fehleranzahl} \tag{2.3}$$

Allerdings gibt es mehrere Arten von Fehlern, es gibt schwerwiegende und leichte Fehler, das heißt nicht jeder Fehler wiegt gleich schwer. Darüber hinaus spricht man eher von der Unzuverlässigkeit eines Systems, wenn man die Zuverlässigkeit messen will. Im Groben bedeutet das, dass verschiedene Fehler in unterschiedlicher Gewichtung in die Rechnung einfließen und das am Ende eine Wert für die Unzuverlässigkeit der Software (S) heraus kommt:

$$U_S = \frac{Fehlerkosten_S}{Betriebsdauer_S} \tag{2.4}$$

Wobei die Zuverlässigkeit den Kehrwert der Rechnung darstellt:

$$Z_S = \frac{1}{U_S} \tag{2.5}$$

2.2.6 Übertragbarkeit

Übertragbarkeit ist die Eignung, von einer Umgebung in eine andere übertragen zu werden. Diese Umgebung kann eine organisatorische Umgebung, Hardware- oder Software-Umgebung einschließen.

Beispiele für gute Übertragbarkeit sind Virtual Machines, wie bei Java (JVM) oder das .NET-Framework. Unterbereiche sind

Anpassbarkeit: Fähigkeit der Software, sich an die verschiedenen Umgebungen anzupassen

Installierbarkeit: Aufwand, der zum Installieren der Software in einer festgelegten Umgebung notwenig ist

Koexistenz: Fähigkeit, einer Software neben einer anderen mit ähnlichen oder gleichen Funktionen zu arbeiten

Austauschbarkeit: notwendiger Aufwand, die Software in einer anderen Umgebung zu verwenden

Konformität: Grad der Erfüllung der Software-Normen zur Übertragbarkeit

Für Webframeworks ist die Übertragbarkeit elementar, da sie eins der Hauptunterschiede von Webframeworks gegenüber einfacher statischer Software bildet. Eine gute Übertragbarkeit heißt, ohne großen Aufwand eine Anwendung auf zwei unterschiedlichen Umgebungen laufen zu lassen, ohne dass die beiden Anwendungsinstanzen sich vom Ergebnis (Funktionalität, Bedienbarkeit) unterscheiden. SwtWiki (2008b)

Für die Übertragbarkeit muss gemessen werden, wie groß der Aufwand ist eine Anwendung auf zwei unterschiedlichen Umgebungen laufen zu lassen. Der Aufwand kann gemessen werden, in dem die Unterschiede in der Ausführung der Anwendung berücksichtigt werden. Umso mehr Unterschiede auftreten bei einer eigentlich komplett gleichen Anwendung, umso größer ist die Komplexität und der Aufwand der Übertragbarkeit der Anwendung auf zwei unterschiedliche Systeme. Darüber hinaus kann auch die Änderungen die an einem System vorgenommen werden gemessen werden. Mehr Änderungen in den Einstellungen eines Systems bedeutet mehr Aufwand, um die Anwendung auf dem System auszuführen. Einfach und

sinnvoll zu messen ist die Installierbarkeit der Anwendung, die in Minuten oder Mannstunden gemessen werden kann. Nicht ganz zur Übertragbarkeit gehörend, aber auf der gleichen Messgrundlage aufbauend, kann der Implementierungsaufwand gemessen werden.

2.3 Anwendung auf die einzelnen Technologien

Um die Webframeworks bewerten zu können, müssen die Kriterien angewandt werden, um Unterschiede oder Gemeinsamkeiten der Webframeworks herauskristallisieren zu können. Dabei werden nicht alle Kriterien benutzt, sondern nur relevante und gut messbare Kriterien. Relevant ist Änderbarkeit und die Übertragbarkeit, dabei insbesondere die Installierbarkeit, das heißt, der Aufwand, der zur Installation der Umgebung notwendig ist und passend dazu der Implementierungsaufwand, der angibt, wie lange es dauerte die Anwendung aufzubauen. Nicht weniger relevant und gut messbar ist die Effizienz und die Zuverlässigkeit der Anwendung.

Die Anwendungen der einzelnen Frameworks werden mit bestimmten Tools gemessen, dabei kommen eigens geschriebene und schon vorhandene Tools zum Einsatz. Weiteres folgt in Kapitel 6.1.

3 Szenario

Um die einzelnen verwendeten Webframeworks gleichberechtigt nebeneinander zu stellen und einen wissenschaftlichen Vergleich zu ziehen, ist es von Nöten ein exemplarisches Szenario zu erstellen, welches gestattet alle Webframeworks unter den gleichen Gesichtspunkten zu testen.

Dieses Szenario muss die Anforderungen erfüllen, ein gewisses Grad an Komplexität aufzuweisen, um eine höhere Divergenz der einzelnen Technologien zu ermöglichen. Denn in den einfachen Anwendungen zeigen sich meist keine großen Unterschiede, erst mit der Ausnutzung jeweiliger sprachinterner Features, welches sie von den anderen Sprachen unterscheidet, gelingt es sie besser einzuordnen und voneinander zu trennen.

Zu diesem Zweck wurde ein Szenario ausgewählt, welches aus dem medizintechnischen Bereich kommt, jedoch leicht übertragbar in andere Bereiche ist und dennoch genügend Komplexität aufweist.

3.1 Anforderungen an ein Szenario

Das Beispiel muss einerseits ein sehr allgemeines Beispiel sein, damit die Allgemeingültigkeit gewährleistet ist, genauso muss es ein mehr abstrakt gehaltenes Beispiel sein, um die Übertragung in ein anderes Szenario einfacher zu gestalten. Trotzdem muss das Beispiel auch konkret genug sein, um auch konkrete Funktionen implementieren zu können, welche die Differenz zwischen den einzelnen Frameworks aufzeigt.

Da das Szenario zur Untersuchung von Webframeworks dient, muss es ein Beispiel sein, bei dem es darum geht, es also nicht nur sinnvoll sondern notwendig ist, einen Mehrbenutzerbetrieb über das Internet zu ermöglichen. Es ist also eine zentrale Anwendung, auf die über das Internet von authorisierten Personen zugegriffen werden kann. Diese können Daten verändern, die entweder nur persönliche Auswirkung oder auch Auswirkung für die anderen Personen hat. Dabei stehen Aspekte wie Sicherheit und Mehrbenutzerbetrieb ganz oben.

Funktionen, die das Beispiel unterstützen sollte sind welche, die bei jeder modernen Web-Anwendung zum Tragen kommen. Darunter fallen dynamische Änderungen, wie es unter anderem von JavaScript ermöglicht wird.

3.2 Einführung ins Szenario 'Physiotherapie'

Das Szenario „Physiotherapie“ ist ein einfaches Beispiel, welches theoretisch in jeder Physiotherapiepraxis zum Einsatz kommen kann. Es handelt sich um ein Programm, welches in der Lage ist die Praxen zu vernetzen und gemeinsam gesammelte Daten zu erheben und so ein verbessertes Behandlungsergebnis zu erreichen. Sinn des Programms ist es, dass die Therapeuten Daten des zu behandelnden Patienten einbinden und Befunde erstellen können. Die zur Behandlung angewandten Methoden und Übungen komplettieren den Befund. Nachdem ein Befund erstellt wurde, werden in definierten Zeitabständen die Werte des Patienten überprüft und in eine Verlaufstabelle eingetragen. Dabei können Übungen, die sich zum Beispiel als nicht wirksam gezeigt haben, durch andere ersetzt werden. Der Verlauf gibt das Krankheitsbild und den Krankheitsverlauf des Patienten wieder. Anhand der durchgeführten Behandlung und dem Krankheitsverlauf lässt sich nun eine bestimmte Wirksamkeit der Behandlung ablesen und so für nachfolgende Befunde eine bessere Behandlung ermöglichen.

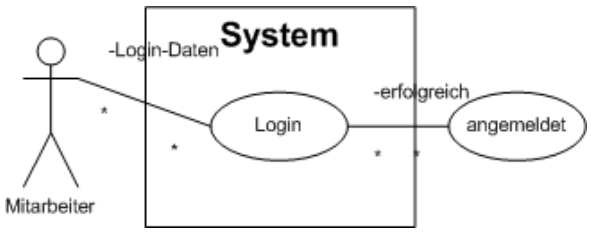
Das Programm und die Daten sind zentral auf einem Server verfügbar und die behandelnden Therapeuten haben die Möglichkeit, nur über einen Browser ihre Patienten zu verwalten und Befunde zu erstellen, dessen Daten in der zentralen Datenbank verwaltet und für andere Therapeuten zur Verfügung gestellt werden kann. Alle Daten des Patienten bleiben nur dem ihm zugewiesenen Therapeuten enthalten, nur die Ergebnisse der Befunde wird der Allgemeinheit zur Verfügung gestellt.

Für das Erstellen des Befundes und der Übungen werden Pseudodaten verwendet, um die Abstraktion zu wahren und sich nicht zu sehr in medizinischen Begriffen zu verirren.

3.2.1 Use Cases

Use Cases helfen typische Szenarien durchzugehen, wie das Programm benutzt werden kann. Die einfachsten Objekte eines Use-Case Diagramms sind Akteure, die direkt oder auch nur indirekt mit dem Programm zu tun haben. Es gibt Anwendungsfälle, die ein bestimmten Teil des Programms schematisch darstellen und die Beziehungen zueinander.

Folgend sind ein paar einfache Use-Cases beschrieben, die aufzeigen sollen, wie das Programm benutzt wird.



Use-Case-Name	**Login ins System**
Vorbedindung	Der Mitarbeiter ist nicht im System angemeldet.
Nachbedingung	Der Mitarbeiter ist angemeldet und kann alle Funktionen des Programms benutzen.
Akteure	Ein Mitarbeiter, der die Anwendung benutzen will und sich anmeldet.
Beschreibung	Der Mitarbeiter will die Anwendung benutzen und meldet sich mit dem richtigen Benutzernamen und Passwort an. Danach ist der Mitarbeiter eingeloggt und kann die Funktionen des Programms benutzen.

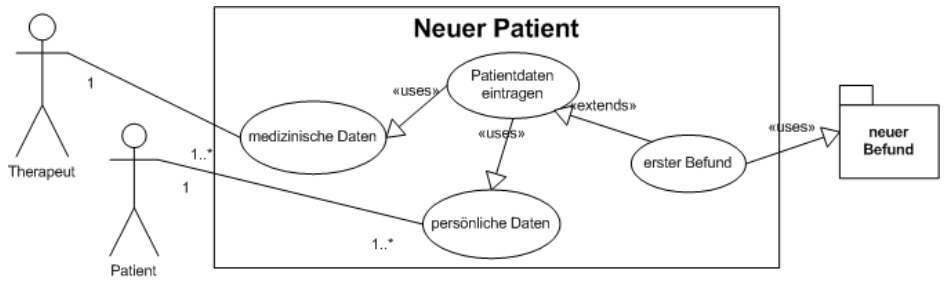
Tabelle 3.1: Anwendungsfall 1: Login

Der erste Fall der Auftritt ist, dass ein Mitarbeiter das Programm benutzen will und sich dafür einloggen muss (Tabelle 3.1). Dafür muss der richtige Benutzername und das Passwort des Mitarbeiters benutzt und in die Login-Maske eingetragen werden. Nach erfolgreicher Anmeldung können alle Funktionen des Programms benutzt werden. Sind die Daten, falsch wird ein Fehler ausgegeben und der Mitarbeiter muss nochmals die richtigen Daten eingeben.

Der nächste Fall der auftritt, ist dass ein neuer Patient, der zum ersten Mal zur Behandlung kommt, in die Datenbank eingefügt werden muss (Tabelle 3.2). Die Patientendaten müssen eingegeben werden, dabei kann der Patient das Einfüllen der persönlichen Daten übernehmen, der Therapeut füllt die medizinischen Daten ein. Darauf folgend wird ein erster Befund angelegt, der eine erste Einstufung und Bewertung des Patienten vornehmen soll.

Beim Befund müssen alle grundlegenden Untersuchungen durchgeführt werden. Es geht vor allem darum festzustellen, wo die Probleme des Patienten liegen. Es werden medizinische Daten eingetragen und eine Reihe von Übungen ausgewählt, welche dem Patienten helfen sollen. Am Ende des Befunds steht eine Diagnose und eine Auswertung des Befundes, welches sich einerseits aus der Diagnose des Therapeuten ergibt, andererseits aus den Daten, die aus Befunden von anderen Patienten erstellt wurden (Tabelle 3.3).

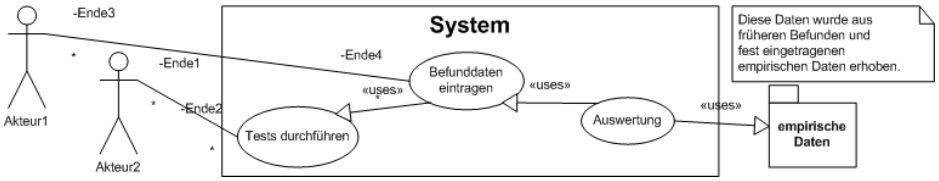
Nachdem der erste Befund erstellt wurde, wird in definierten zeitlichen Abständen, zum Beispiel einen Monat, die Daten aktualisiert. Die Tests während des Befundes, die beim Patienten auffällig waren, werden nun kontrolliert und in einer Verlaufstabelle wird notiert, wie sich



Use-Case-Name	**Neuer Patient**
Vorbedindung	Der Patient ist noch nicht im System vorhanden.
Nachbedingung	Der Patient ist im System eingetragen und es können nun Befunde für den Patienten erstellt werden.
Akteure	Ein Mitarbeiter, der die Daten des Patienten in das System eitnrägt.
Beschreibung	Der Mitarbeiter ruft die „Neuer Patient“-Maske des Systems auf und trägt die Daten, die sie vom Patienten bekommen hat in das System ein und bestätigt.

Tabelle 3.2: Anwendungsfall 2: Neuer Patient

die Daten verändert haben. Es können neue Übungen zugewiesen werden und die Daten werden in die Datenbank aufgenommen (Tabelle 3.4).



Use-Case-Name	**Neuer Befund**
Vorbedindung	Der Patient, für den der Befund erstellt wird, ist im System vorhanden.
Nachbedingung	Der Befund wurde dem Patienten hinzugefügt und kann bearbeitet werden und der Verlauf kann verfolgt werden.
Akteure	Ein Mitarbeiter, der die Daten des Befundes in das System einträgt und der Patient, welcher die Übungen durchführt, die nötig sind die erforderlichen Testdaten zu bekommen.
Beschreibung	Der Therapeut ruft die „Neuer Befund“-Maske auf und trägt die Daten ein, welche aus den Tests, die der Patient macht herauskommen.

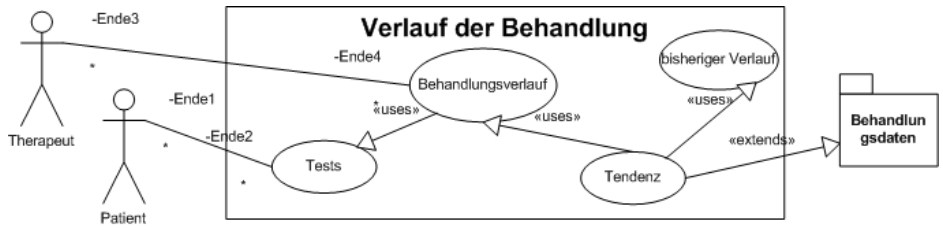
Tabelle 3.3: Anwendungsfall 3: Neuer Befund

3.2.2 Aufbau der Anwendung

Der technische Aufbau der Anwendung sieht aus wie folgt:

Es gibt drei grundlegende Teile in dem Aufbau der Anwendung. Zuallererst das Model, dieser ist der wohl grundlegendste Teil, der sich von Framework zu Framework nicht verändern wird und somit allen hier programmierten Anwendungen gemein ist. Das Model ist in dieser Zeichnung aufgeteilt in die Anwendungslogik, mit dem die fachliche Implementierung der Anwendung gemeint ist (s.u.) und der „technische“ Teil des Models, hiermit sind die Klassen gemeint, die z.B. die Verbindung zur Datenbank herstellen. Die Datenbank, zum Besipiel MySQL, befindet sich ganz unten und speichert alle Daten die anfallen. Auf den Aufbau der Datenbank werden wir hier nicht näher eingehen, im Beispiel wird dieser Teil simuliert sein. Der zweite Teil ist der Controller, welcher für den Austausch zwischen der Logik und dem Frontend zuständig ist, dieser Teil ist je nach benutztem Framework mal mehr und mal weniger ausgeprägt, aber immer wichtig, um den reibungslosen Austausch zwischen Benutzer und Anwendung zu garantieren. Der dritte Teil ist das Frontend oder View, dies ist der Teil den der Benutzer sieht und mit dem er interagiert.

Von der Mainclass wird der fachliche Teil initialisiert, welcher die Logik beinhaltet. Das fachliche Model ist nicht bindend und soll nur eine schematische Darstellung des Aufbaus des



Use-Case-Name	**Eintrag im Verlauf**
Vorbedindung	Der Patient und der Befund, zu dem der Verlauf niedergeschrieben wird, sind im System vorhanden.
Nachbedingung	Der Eintrag ist im System vorhanden und wurde dem entsprechenden Patienten und Befund zugeordnet und wird nun im Verlauf-Fenster angezeigt.
Akteure	Ein Therapeut, der die Daten in das System einträgt und der Patient, welcher die verordneten Übungen durchführt.
Beschreibung	Es wird ein Eintrag im Verlauf vorgenommen, der die Entwicklung des Patienten in Bezug auf den entsprechenden Befund zeigt.

Tabelle 3.4: Anwendungsfall 4: Eintrag im Verlauf

Programms liefern, da hier unterschiedliche Web-Frameworks zum Einsatz kommen und dadurch es auch unterschiedliche Herangehensweisen an das Problem gibt.

Die Patientenliste wird in der Datenbank gehalten und per statischer Methode geholt, von dort erhält der Controller Zugriff auf die Patienten. Jeder Patient kann mehrere Befunde haben, für jeden Befund gibt es wiederrum einen Verlauf, der aus beliebig vielen Einträgen besteht. Ein Eintrag im Verlauf gibt Auskunft über die veränderten Daten und neuen Werte des Patienten und auch Auskunft über die ausgeführten Übungen.

Für den Verlauf werden nur die Daten aktualisiert die von Interesse sind. Dazu werden diese Daten als solche gekennzeichnet, entweder automatisch, wenn nicht der Standardwert eingetragen ist oder manuell durch den Therapeuten.

Das Frontend der Anwendung besteht aus einer Hauptseite, wo auf die einzelnen Patienten zugegriffen werden kann. Beim ausgewählten Patienten werden dann die einzelnen Befunde angezeigt zum bearbeiten, löschen oder auch neu erstellen. Dort kann auch auf den Verlauf zugegriffen werden. Auf der Verlaufsseite ist es dann möglich neue Einträge hinzuzufügen oder Einträge zu löschen oder zu bearbeiten. Zusätzlich kommen noch Seiten zum Erstellen eines Befundes und zum Erstellen eines neuen Patienten. Es sind somit ungefähr fünf verschiedene Seiten, das Frontend wird also sehr klein gehalten, um mögliche Komplexität zu vermeiden (Abb. 3.3).

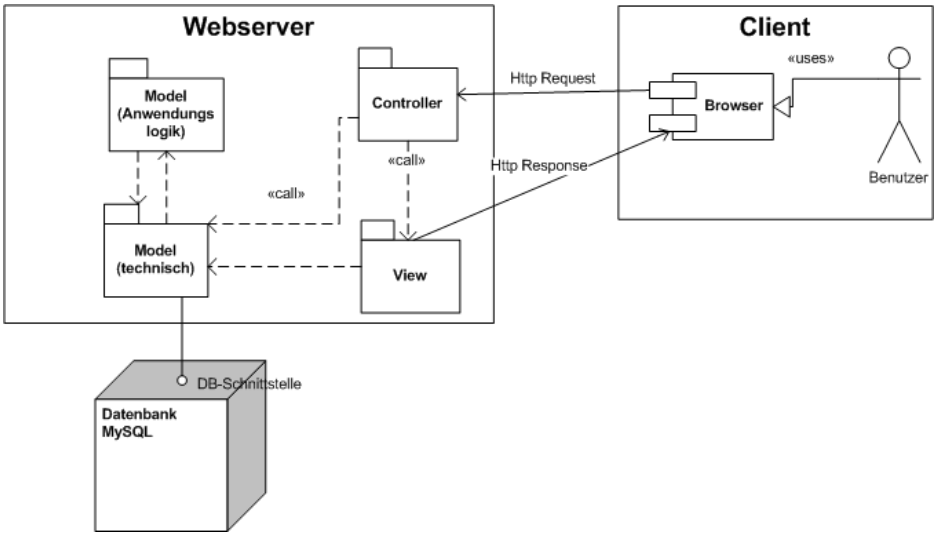


Abbildung 3.1: technischer Aufbau der Anwendung

3.3 Übertragbarkeit in andere Bereiche

Das Szenario soll helfen eine Grundlage zu geben, um die Frameworks mit den definierten Kriterien messen zu können. Daher wurde ein eher abstraktes Beispiel genommen, um die Bewertung der Frameworks leichter in ein anderes Gebiet übertragen zu können. Die Ergebnisse, die bei der Messung der Frameworks entstehen, können also nicht nur im Bereich der Physiotherapie benutzt werden, sondern bei jeder Art von Anwendung, bei der Daten erhoben werden, die gespeichert und fortlaufend aktualisert und verglichen werden, um dann zu sehen, wie sich die Daten in welcher Umgebung entwickeln.

Dies wäre zum Beispiel ein Programm zur Kundenzufriedenheit. Kunden werden gefragt, die Daten, in diesem Falle die Meinung der Kunden, werden gespeichert und mit später erhobenen Daten verglichen. Nun können bestimmte Aktionen ausgelöst werden, die die Daten verändern und die Veränderung wird mit der Aktion assoziiert und in der Datenbank festgehalten. Nun profitieren alle, die mit der Datenbank verbunden sind, von den so empirisch erhobenen Daten.

Allgemein gesagt könnte man dieses Szenario in alle Bereiche übertragen, wo nach bestimmten Kriterien erhobene Daten in einer zentralen Datenbank gespeichert werden, um Prozesse zu optimieren oder bestimmte Vorgehensweisen zu erstellen oder auch alteingesessene Wege zu streichen.

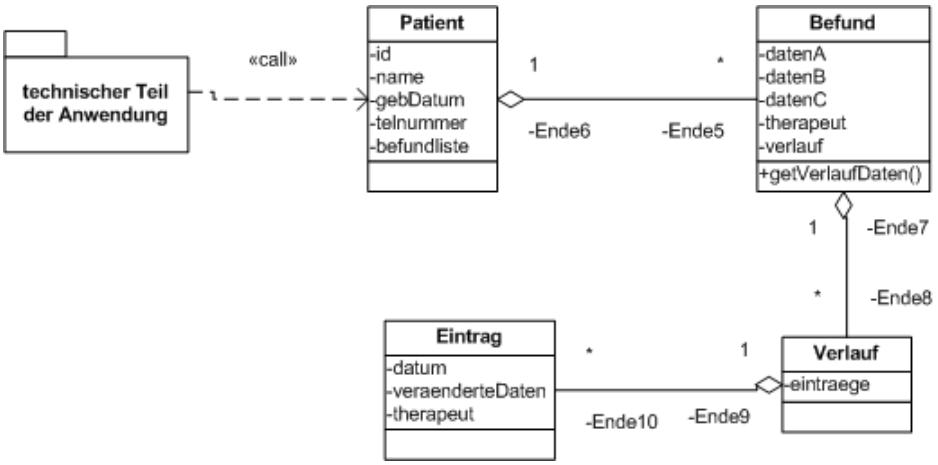


Abbildung 3.2: Aufbau der fachlichen Seite

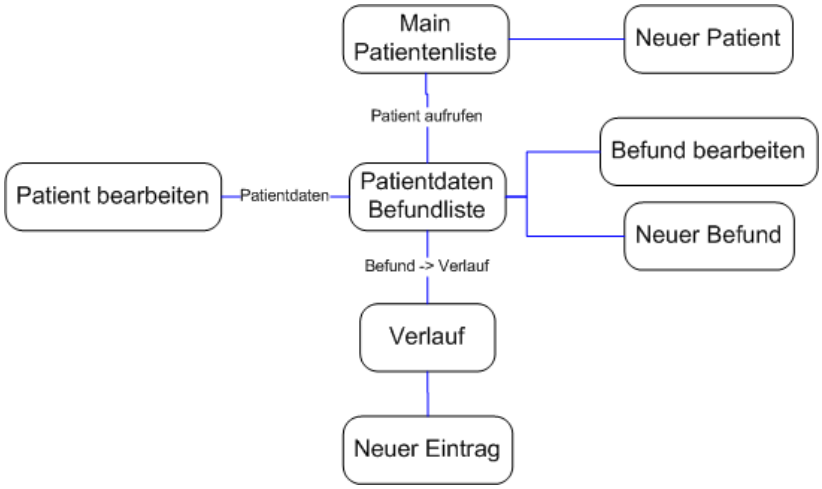


Abbildung 3.3: Diagramm vom Frontend

4 Technologien

4.1 Einführung in die Web-Frameworks

Mit dem Aufkommen des World Wide Webs war es möglich, Inhalte über einen Browser mittels dem von Tim Berners-Lee entwickelten HTML darzustellen. Es konnte formatierter Text und Bilder angezeigt werden. Das wichtigste Element an HTML war jedoch die Verlinkung (Tag: <a>) zwischen zwei Seiten, die es möglich machte durch das Netz zu „surfen“. Durch das Klicken auf ein Link wird die Seite, die sich als URL hinter dem href-Atribut verbirgt, an den Client gesendet und angezeigt, wobei immer die komplette Seite geladen wurde. Desweiteren konnten mit HTML nur statische Inhalte wiedergegeben werden, das heißt, dass die geladene Seite nicht verändert werden konnte, da HTML keine Programmiersprache, sondern eine reine Beschreibungssprache ist. In den Anfängen des WWW hat dies kein Problem dargestellt, da es eigentlich nur zur Informationsbeschaffung diente oder um sich mit anderen Usern über Foren auszutauschen.

Doch mit der steigenden Nutzerzahl stiegen auch die Anforderungen an das Web, so wurde die Layoutsprache CSS integriert und JavaScript machte das WWW dynamischer. Doch die ganze Technologie wurde nur auf dem Client ausgeführt. Der Server verschickte nur die angeforderten Seiten an den Client, der dann über den Browser mit CSS das Layout der dargestellten Seite veränderte oder mit JavaScript dynamische Inhalte anbot. Das Problem liegt auf der Hand, da der Client alle Aufgaben übernahm, konnten keine komplexen serverseitigen Web-Anwendung realisiert werden. Anwendungen wie Versandhäuser, Videoportale oder soziale Netzwerke sind ohne eine serverseitige Anwendung, welche auf große Datenmengen zugreifen und diese auswerten kann, kaum realisierbar. Serverseitige Webframeworks machen also einen Großteil des heutigen Web 2.0 aus.

Doch nachdem zuerst Anwendungen nur auf dem Client ausgeführt worden sind und später immer mehr auf dem Server ausgelagert wurde, hat sich inzwischen ein Gleichgewicht entwickelt, so dass nicht mehr die Frage besteht, ob bestimmte Bereiche auf den Server ausgelagert werden, sondern wieviel vom Server erledigt wird und wieviel vom Client.

4.2 Vergleich von „Fat Client“ zu „Thin Client“

Je nachdem, wieviel von der Webarchitektur der Server übernimmt oder der Client spricht man von einem „Thin Client“, das bedeutet, dass der Client nur die Präsentation der Daten übernimmt oder von einem „Fat Client“, welcher dann Teile oder auch die ganze Anwendungslogik mit übernimmt. Eine Auflistung der einzelnen Stufen gibt es in Abbildung 4.1.

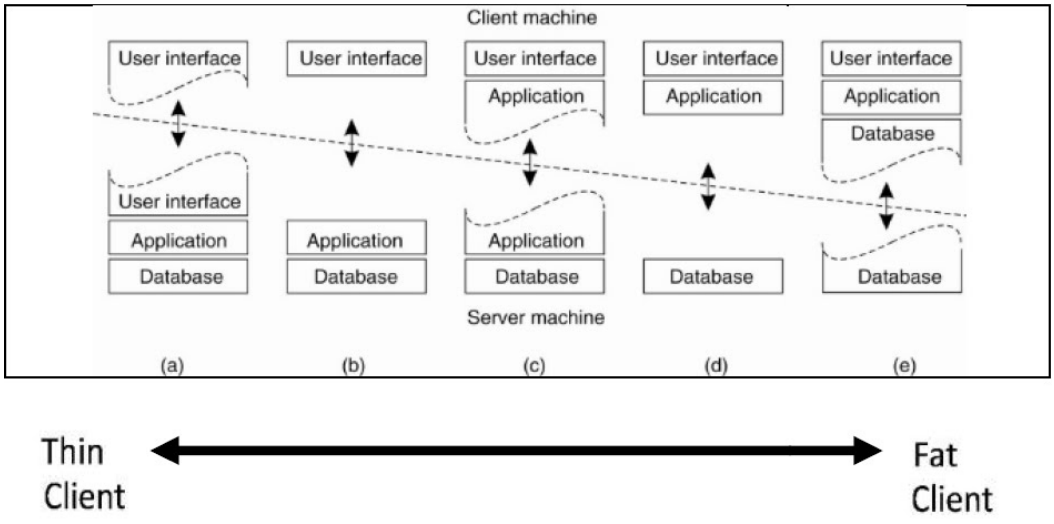


Abbildung 4.1: Verschiedene Stufen der Client-Architektur

Eine Anwendung, in der der Server die komplette Logik verwaltet und die Ausgabe, quasi „vorgekaut“, dem Client zur Verfügung stellt, der es dann über den Browser ausgibt, wäre ein Beispiel für die Architektur unter (b). Bei dieser Architektur wird auf dem Client keine zusätzlichen Programme vorausgesetzt, da nur HTML, JavaScript und CSS zum Einsatz kommt. Technologien, die nach dem Thin-Client-Prinzip arbeiten, sind zum Beispiel JSP oder PHP. Den Gegensatz dazu bildet der „Fat Client“. Hierbei handelt es sicht um eine Anwendung, welche auf dem Client etwaige Zusatzprogramme erwartet, wie eine installierte JVM oder plattformabhängig ist. Horn (2007) Der Server arbeitet hier meist nur als Datenspeicher, der Client holt die Daten vom Server, verarbeitet sie und macht dann eine Ausgabe. Ein konkretes Beispiel hierfür wäre ein Java-Applet oder eine Desktop-Applikation, welche über ein Web-Service mit dem Server kommuniziert, wie zum Beispiel Google Earth.

Die Vorteile der Thin-Client-Architektur sind, dass sie auf jedem Rechner aufgerufen werden können, der einen Browser hat. Jeder Browser ist in der Lage HTML, JavaScript und CSS zu verarbeiten und somit können Inhalte schnell und unkompliziert abgerufen werden. Im Gegensatz dazu müssen beim Fat Client meistens Programme installiert werden, die es dem Rechner ermöglichen die Inhalte, die von der Anwendung angeboten werden, anzuzeigen. Für Computerlaien kann dies eine Hürde darstellen und somit den Zugang zu den

Anwendungen verwehren. Anderenfalls ist die Möglichkeit der Inhaltsdarstellung bei Fat-Client-Anwendungen mächtiger, da man hier nicht an das HTML-Layout gebunden ist. Mit Java-Applets zum Beispiel kann die Interaktion und die Bedienungsfreundlichkeit mit dem Benutzer erhöht werden. Jedoch können beim Thin Client solche Einschränkungen mit dem Einsatz von JavaScript zum Teil ausgeglichen werden. Große Abhilfe verschafft hier die Nutzung von Ajax, welches durch JavaScript und anderen Technologien ansprechende und flexible Webanwendungen ermöglicht. Mittlerweile unterstützen auch alle bekannten Browser die Ajax-Technologie.

Ein weiteres Problem bei Fat Clients ist, dass der Anwendungscode, der auf dem Client gelagert ist, dem Benutzer sichtbar und zugänglich ist, dies kann unerwünscht sein und ein mögliches Sicherheitsrisiko darstellen.Wikipedia (2009) Genauso verhält es sich bei der Speicherung von Daten. Temporäre Datenspeicherung auf dem Client (z.b. über einen Cookie) kann von schädlicher Spyware ausgelesen werden. Bei der eher serverseitigen Webarchitektur kann ein Sessionmanagement auf dem Server die temporären Daten verwalten und so besser vor fremdem Zugriff schützen.Wikipedia (2009)

4.3 Beschreibung einzelner Technologien

In dieser Arbeit werden die Webframeworks JavaServer Faces von Sun, das Google Web Toolkit von Google und PHP von der PHP Group zum Einsatz kommen. Alle sind frei erhältlich unter einer Open-Source-Lizenz. Alle drei Technologien sind serverseitige Webframeworks, sind aber im Aufbau grundverschieden. PHP ist das wohl weit verbreiteste Webframework und wird von fast allen Webservern unterstützt. JSF ist eine Fusion von Servlet-Technologie und Facelets und recht populär, wird jedoch nur von ein paar Webservern unterstützt (z.B. Tomcat). GWT ist ein relativ neues Webframework, dass auf Ajax-Applikationen spezialisiert und daher nur JavaScript Seiten produziert und somit auch auf fast allen Webservern läuft.

Eine genauerer Vergleich der einzelnen Webframework folgt unter Kapitel 4.4.

4.3.1 JavaServer Faces

Im Jahre 1997 entwickelte Sun Microsystems einen ersten Ansatz zur serverseitigen Webentwicklung, den Servlets. Servlet ist ein sogenanntes Kofferwort, also ein Wort welches aus zwei Begriffen zusammengesetzt ist, einerseits „Server“ und andererseits „Applet“. Wikipedia (2010g) Servlets waren Java-Klassen, welche Anfragen an den Client entgegen nahmen und beantworteten, indem in den Java-Code Befehle eingebunden waren, welche HTML-Code erzeugten. So erzeugt der Befehl

```
println("<html>text</html>");
```

dynamisch den HTML-Code und gibt ihn über einen OutputStream an den Client aus. So konnte sehr einfach auf Anfragen des Clients der HTML-Code dynamisch erzeugt und ausgegeben werden, jedoch wurden die Servlet-Klassen bei etwas komplexeren HTML-Seiten sehr schnell sehr unübersichtlich und somit auch schlecht wartbar, so dass eine einfache HTML-Seite, welche einen einachen Satz ausgab, zu einem beachtlichen Texthaufen anwuchs. irian (2010)

```
1  public class BeispielServlet extends HttpServlet {
2
3    public void doGet (
4      HttpServletRequest req, HttpServletResponse res)
5        throws ServletException, IOException {
6
7        //Vom HTTP-Request kann der Entwickler sich die
8        //Parameter für die Verarbeitung der Anfrage holen
9        String name = req.getParameter("name");
10       String text = req.getParameter("text");
11
12       //Es wird HTML-Text an den Browser zurückgesendet
13       res.setContentType("text/html");
14
15       //Ein OutputStream macht es möglich, den
16       //eigentlichen Text zu senden.
17       ServletOutputStream out = res.getOutputStream();
18
19       //Einzelne Elemente des Texts werden versendet
20       out.println("<html><head><title>");
21       out.println(name);
22       out.println("</title></head><body>");
23       out.println(text);
24       out.println("</body>");
25       out.println("</html>");
26
27       out.flush();
28     }
29 }
```

In diesem Beispiel wird ein bestimmter Text, welcher über die Get-Funktion abgerufen wird, in einer neuen Seite ausgegeben. Diese sehr simple Funktion kommt schon zu einen Umfang von knapp 30 Zeilen und ist durch den permanenten Einsatz von out.println(““) sehr unübersichtlich gestaltet.

Die nächste Stufe in der Entwicklung stellten die JavaServer Pages (JSP) dar. Hierbei wurden erstmals die Java-Befehle als XML-Tag direkt in die HTML-Seite eingebettet. Diese Tags wurden dann vom Server bearbeitet, ehe die fertige HTML-Seite an den Client gesendet wurde. Im Folgenden das obige Beispiel in JSP:

```
<@ page language="java" >
<html>
  <head>
    <title><=request.getParameter("Seitentitel");></title>
  </head>
  <body>
    <=request.getParameter("AusgabeText");>
  </body>
</html>
```

Durch die Einbindung in die XML-Syntax ist die JSP-Variante deutlich übersichtlicher und somit besser wartbar. Jedoch ist bei komplexeren Anwendungen der Bogen schnell überspannt und durch zuviel Java-Code, der in die HTML-Seite eingebettet wird, geht auch hier die Übersichtlichkeit schnell verloren. Zusätzlich dazu traten viele Fehler, die normalerweise bei der Erstellung von Java-Klassen beseitigt worden wären, erst zur Laufzeit auf, da der Quellcode erst bei Anwendungsstart vom Webserver kompiliert wurde. irian (2010)

Um solcher Probleme Herr zu werden, hat man angefangen Frameworks zu entwickeln, welche die Entwickler angehalten haben, möglichst viel Anwendungslogik auszulagern und somit möglichst nur die Layoutbeschreibungen in den JSP-Seiten zu belassen. Eine Möglichkeit dazu ist das Model-View-Controller-Prinzip, welches vorsieht die Anwendung in drei Teile zu spalten. Zum Ersten das Model, welches die Anwendungslogik beinhaltet, zweitens die Sicht (engl.: View), in der das Layout und die Ausgabe festgehalten wird und drittens der Controller, welcher einerseits die Sicht erzeugt, andererseits das Model initialisiert. Speziell für den Webbereich und insbesondere in der Entwicklung mit Java hat sich eine spezielle Variante des MVC-Prinzips etabliert, das das Model2-Prinzip genannt wird.

Das Model2-Prinzip handelt nach den Prinzipien des MVC-Prinzips. Der Client sendet eine Anfrage an den Server (1), diese wird vom Controller entgegengenommen, der nun die erforderliche Sicht (4) und falls nötig das benötigte Model (2) erzeugt. Die Sicht bearbeitet nun gemäß der Anfrage das Model (5), welches auf die dahinter liegende Anwendungslogik und die Datenbank zugreifen kann (3) und sendet eine Antwort an den Server zurück (6). irian (2010)

Anfang 2004 trat dann JavaServer Faces (JSF) auf den Plan. JSF baut auf JSP auf, beruht also auf dem Model2-Prinzip, bietet aber, vor allem was die Präsentation angeht, eine Reihe Vorzüge, die unter anderem eine bessere Wiederverwendbarkeit der Komponenten

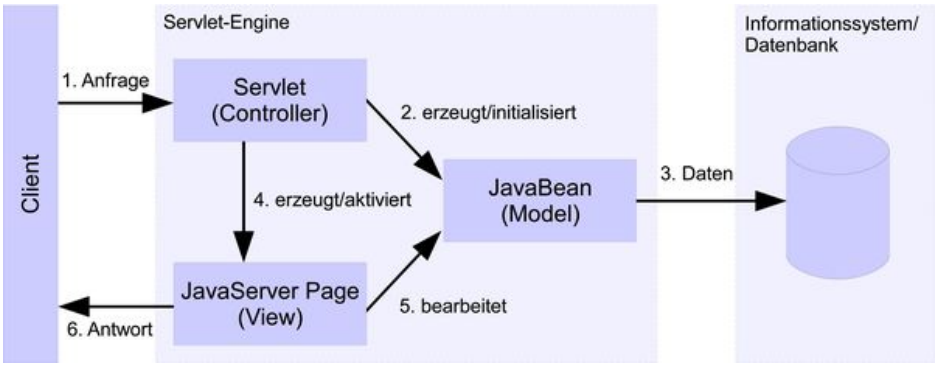


Abbildung 4.2: Das Model2-Prinzip

bereit stellt. JSF führt die zum Teil weit auseinander gelaufenen Entwicklungsstränge zusammen und standardisiert die Webanwendungsentwicklung. irian (2010) Die Vorzüge von JSF gegenüber reiner JSP-Programmierung ist die vollkommene Trennung von Java-Code und HTML-Code. Die Verbindung zwischen den zwei Seiten stellen bestimmte JSF-Tags sicher, die über bestimmte Attribute und einer eigens entwickelten Expression Language (EL) mit dem Model kommunizieren. Das obige Beispiel sieht in JSF dann wie folgt aus:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
    xmlns:h="http://java.sun.com/jsf/html">
<h:head>
  <title>Hello World</title>
</h:head>
<h:body>
  <h:outputText value="#{exampleBean.ausgabe}"/>
</h:body>
</html>
```

Wie man sieht wird der Textfluss der JavaServer Page durch die Nutzung der JSF-Tags konsequent eingehalten und wird nicht durch Java-Code Einbettungen gestört. Das Wichtigste hier ist die Nutzung der JSF-Komponenten, in diesem Falle <h:outputText>. Dem Value-Attribut wird über die Expression Language ein Wert zugewiesen, der in der Bean von der Funktion mit der Signatur *String getAusgabe()* zurückgegeben wird. Der Tag wird dann vom Server in reinen HTML- und JavaScript-Code kompiliert und an den Client gesendet. Vollständigkeitshalber hier die Implementierung der BackingBean:

```
package bsp;

import javax.faces.bean.ManagedBean;
```

```
import javax.faces.bean.RequestScoped;

@ManagedBean
@RequestScoped
public class ExampleBean {
  private String ausgabe;

  public String getAusgabe() {
    return ausgabe;
  }
  public void setAusgabe(String s) {
    ausgabe = s;
  }
}
```

Dies ist eine einfache Umsetzung einer ManagedBean. Sie wird von dem Controller initialisiert, nachdem sie von dem <h:outputText/>-Tag aufgerufen wurde. Die Bean hat ein einfaches Datenfeld „ausgabe“, welches durch einen Getter und Setter erreicht werden kann. Wichtig hier ist, dass in der JSP-Seite nur #{exampleBean.**ausgabe**} aufgerufen wurde, in der Bean jedoch getAusgabe() und setAusgabe(String) stehen, das bedeutet, dass die Get- und Setmethoden auf jeden Fall gleich heißen müssen, da es sonst in der JSP-Seite zu Fehlern kommt. Auffällig sind auch die Annotationen über der Klasse (*@ManagedBean* und *@RequestScoped*), welche erstens die Klasse als ManagedBean kennzeichnet, so dass sie vom Controller auch initialisiert werden kann und zweitens die Lebenszeit der Bean angibt. RequestScope sagt aus, dass die Klasse für die Länge einer Http-Anfrage erstellt und danach wieder gelöscht wird. Beans lassen sich in weitere folgende Scopes zuweisen: Marinschek u. a. (2010)

- *@NoneScoped*: die Bean wird bei jedem Aufruf neu erstellt
- *@ViewScoped*: die Bean ist an die Seite in der Ansicht geknüpft und lebt solange, wie die Seite aufgerufen ist
- *@SessionScoped*: die Bean hat die Lebensdauer einer Sitzung
- *@ApplicationScoped*: die Bean lebt für die gesamte Dauer der Anwendung und es wird nur eine Instanz für alle Benutzer erstellt

Die ManagedBean und Scope Zuweisung per Annotation wurde mit JSF 2.0 eingeführt, welches im Juli 2009 erschien, vorher mussten die Klassen in der *faces-config.xml* als ManagedBean deklariert und dem richtigen Scope zugeordnet werden.

Mit JSF können so einfach und schnell JSF-Tags in dem HTML-Layout deklariert werden, die dann über einen einfachen Aufruf an die Bean über die Expression Language Werte

zugewiesen bekommen. Jedoch können mit der EL nicht nur Werte abgerufen oder verändert, sondern es können auch komplexere Anweisungen ausgeführt werden, wie z.B.: Müller (2006)

- Zugriff auf ein Array: #{someBean.array[0]}
- Zugriff auf ein assoziatives Array: #{someBean.array['key']}
- auf leeren String testen: #{empty someBean.string}
- ternären Operator anwenden: #{someBean.value == null ? 'true' : 'false'}

Neu bei JSF-Version 2.0 ist auch, dass Ajax in den Standard integriert worden ist. Es kann nun über einen <f:ajax> Tag, der in einen <h:commandButton> Tag eingebettet ist, auf die Bean zugegriffen und ein Wert in einem <h:outputText> ausgegeben werden. Das ganze würde dann so aussehen:

```
1  <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
       "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
3  <html xmlns="http://www.w3.org/1999/xhtml"
       xmlns:h="http://java.sun.com/jsf/html"
5      xmlns:f="http://java.sun.com/jsf/core">
   <h:head>
7    <title>Ajax Beispiel</title>
   </h:head>
9  <h:body>
       <h:commandButton value="Press"
11         action="#{someBean.action}">
           <f:ajax render="exampleText"/>
13     </h:commandButton><br/>
       <h2><h:outputText value="#{someBean.value}"
15         id="exampleText"/></h2>
   </h:body>
```

Wenn der CommandButton gedrückt wird, kann über die action-Methode der Bean der Wert *value* verändert und über das Textfeld ausgegeben werden. Dabei wird durch den <f:ajax> Tag nur die Daten des commandButtons und des Textfelds an den Server geschickt und aktualisiert, somit treten kaum Ladezeiten auf und der Rest der Seite bleibt unberücksichtigt. Marinschek u. a. (2010) Leider ist der <f:ajax> Tag bis jetzt die einzige Möglichkeit Ajax-Funktionalität einzufügen und die Möglichkeiten sind noch stark begrenzt.

4.3.2 Google Web Toolkit

Das Google Web Toolkit (GWT) ist im Gegensatz zu JavaServer Faces ein ziemlich neues Web-Framework. Entwickelt wurde es von der Firma Google im Jahr 2006 und wird seither als freie Software unter der Apache-Lizenz veröffentlicht. Genau wie JSF benutzt GWT Java, um den serverseitigen Teil der Webapplikation zu realisieren. Das Besondere ist, dass auch der clientseitige Code komplett in Java geschrieben werden kann, es ist in GWT also möglich ajaxbasierende Webanwendungen zu erschaffen, ohne eine einzige Zeile JavaScript zu schreiben. Möglich macht dies der GWT-Compiler, welcher einfachen Java-Code in JavaScript umwandelt und es dem Entwickler somit ermöglicht, eine Webanwendung komplett in Java zu schreiben. Wikipedia (2010c)

Im Grunde wird GWT vorwiegend dafür benutzt Ajax-Anwendungen zu erstellen. Ajax (abgk.: Asynchronous JavaScript and XML) bezeichnet ein Konzept zur asynchronen Datenübermittlung in einer Webanwendung zwischen Client und Server. Im Kern wird Ajax dazu benutzt, bei einer Anfrage an den Server nicht die ganze Webseite neu zugeschickt zu bekommen, sondern nur einen Teil der Webseite. Der Vorteil ist, dass dadurch nur die für den Benutzer relevanten Daten übertragen wird und somit nicht soviel „Datenmüll“ übertragen wird, somit werden zum Beispiel Formulare die schon ausgefüllt sind nicht gelöscht, wenn eine Anfrage an den Server gestellt wird. Ein weiterer Vorteil ist, dass mit Ajax schon etablierte und seit Jahren genutzte Technologien zum Einsatz kommen, wie JavaScript oder XML und somit die Browser und die Webserver keine zusätzlichen Features brauchen, um mit Ajax-Technologie Webanwendungen darstellen zu können. Jedoch gibt es auch ein paar Probleme, die Ajax mit sich bringt. So muss der Client in jedem Fall JavaScript akriviert haben, um die Vorteile von Ajax benutzen zu können. Darüber hinaus gibt es Probleme mit der XML-Verarbeitung im Client, da die Browser sich teilweise massiv bei der XML-Verarbeitung unterscheiden und es somit noch schwieriger wird Webapplikationen für alle Browser gleichermaßen anzubieten. Steyer (2007)

Eine wichtige Funktion, die es möglich macht die nahezu komplette Anwendung in Java zu erstellen, ist der GWT Compiler. Beim Kompiliervorgang werden alle Klassen, welche von der Applikation benutzt werden, in die entsprechenden JavaScript-Gegenstücke übersetzt. Das Ergebnis ist dann eine Mischung aus HTML-/CSS- und JavaScript Seiten. Der Compiler übersetzt dabei den Java-SourceCode nicht nur, sondern er versucht auch eine möglichst hohe Optimierung des JavaScript-Codes zu erreichen. Laut Google soll die dabei entstehenden JavaScript-Code mindestens so performant sein wie handgeschriebene Gegenstücke. Steyer (01/2007)

Vorteile von der reinen Java-Programmierung sind, dass es für Java wesentlich mehr und bessere Entwicklungsumgebungen gibt als für JavaScript. Da Java im Gegensatz zu JavaScript statisch typisiert ist, ist es für den Entwickler einfacher Fehler zu erkennen und der

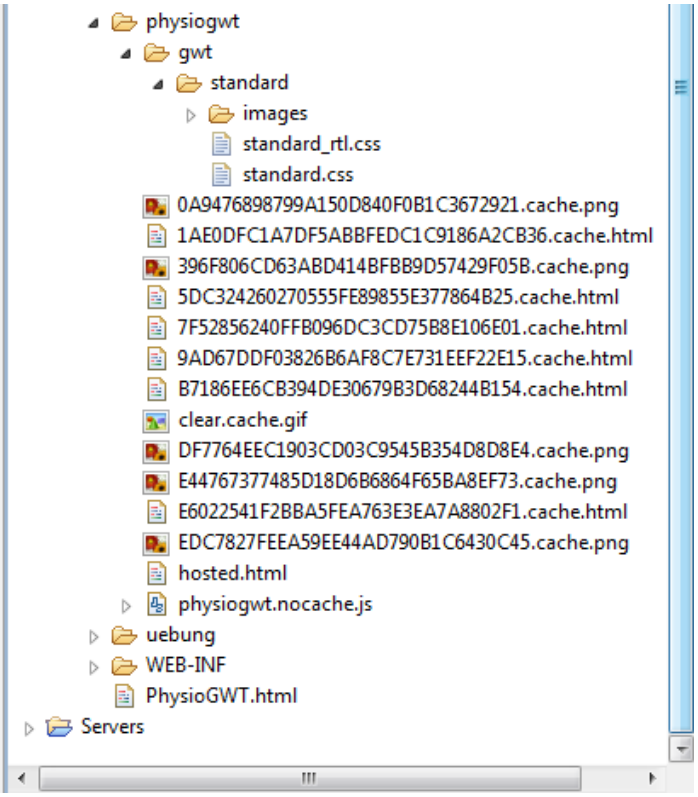


Abbildung 4.3: Auszug von den vom GWT-Compiler übersetzten Dateien

Compiler findet schon zur Kompilierungsphase die meisten Fehler. Sommers (12/2006) Steyer (01/2007) Für Entwickler, die viel Desktop-Anwendungen mit Java erstellt haben, wird so der Einstieg in die Webentwicklung leichter gemacht.

Nachteile dieser Art der Webentwicklung ist, nicht hinter den GWT-Compiler blicken zu können und so nicht genau zu wissen, wie der Compiler die HTML-Seite genau aufbaut und welche Tags benutzt werden. So kann es zu Komplikationen kommen, wenn zu den übersetzten HTML-Tags das eigens geschriebene CSS-Layout passen soll.

Um eine GWT-Applikation zu erzeugen braucht es also nicht mehr als ein paar Java- und HTML-Kenntnisse, wobei auch die HTML-Kenntnisse nicht unbedingt von Nöten sind. Für eine GWT-Anwendung ist normalerweise nur eine HTML-Seite nötig, in der dann per Java-Code die HTML-Elemente eingefügt werden. Dabei sieht die HTML-Seite meist immer gleich aus.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
  <head>
```

```
4     <meta http-equiv="content-type"
              content="text/html; charset=UTF-8">
6     <title>Hello</title>
      <script type="text/javascript" language="javascript"
8         src="hello/hello.nocache.js"></script>
    </head>
10  <body bgcolor="white">
      <iframe id="__gwt_historyFrame"
12      style="width:0;height:0;border:0"></iframe>
      <noscript>
14      <div style="...">
          Your web browser must have JavaScript enabled
16        in order for this application to display correctly.
        </div>
18    </noscript>
    </body>
20  </html>
```

Zu beachten ist hier die Angabe der JavaScript-Seite, die vom GWT-Compiler erstellt wird. Diese heißt immer gleich („<Modulname>.nocache.js“) und beinhaltet unter anderem eine Tabelle die abhängig von dem benutztem Browser oder der Lokalisierung auf verschiedene *.cache.html Seiten hinweist. Für jeden unterstützten Browser und Sprachen werden verschiedene *.cache.html Seiten erstellt, die einen optimierten JavaScript-Code für den jeweiligen Browser bereit stellt. „Firefox in German“ oder „Safari in English“ wären zum Beispiel verschiedene Einträge in der Tabelle, die zu den unterschiedlichen *.cache.html Seiten hinweisen. Die Endung .nocache.js weist den Browser darauf hin, die Seite jedes Mal, wenn die Anwendung aufgerufen wird, neu vom Server anzufordern, da der GWT Compiler diese Datei jedes Mal erneuert und somit sichergestellt ist, dass die neueste Version benutzt wird. Izabel (12/2007b)

Die *.cache.html Dateien beinhalten die Anwedungslogik der Applikation. Sie beinhalten reinen JavaScript-Code, der jedoch in eine HTML-Seite eingebettet ist. Dies hat den Grund, dass manche Browser Probleme mit reinen JavaScript-Dateien haben und durch die Einbettung in eine HTML-Seite dieses Problem gelöst wird. Der Name der cache-Datei ensteht aus der MD5-Summe des Inhalts (zB. 421E38C2351CE5EC433B92059D018123.cache.html). Jede .cache.html Datei hat somit einen einzigartigen Namen und kann vom Browser im Cache gehalten werden, da bei einer Änderung am Inhalt der Name der Datei auch geändert werden würde. Izabel (12/2007b)

Eine weitere Besonderheit ist die Angabe eines iframes mit der id=“__gwt_history-Frame“. Ein Problem mit Ajax-Applikation ist die Benutzung der Vor- und Zurück-Buttons der Browser, die die „History“ des Browser (oder eines Browsertabs) durchblättern. Wenn einer

der Knöpfe gedrückt wird, wird die vorherige oder die nachfolgende URL aufgerufen, je nachdem in welcher Reihenfolge sie aufgerufen wurden. Ajax-Applikationen laden jedoch immer nur ein Teil der Webseite und die URL der Webseite bleibt somit unverändert und die History-Buttons des Browsers nutzlos. GWT bietet jedoch Möglichkeiten an, die History des Browsers zu verwalten und somit zu bestimmen, ob sie benutzt werden soll und was genau passieren soll, wenn sie benutzt wird. Dazu verwaltet GWT die History als ein Stack von Tokens, das sind Identifier, die der Entwickler selbst benennt. Jedesmal wenn ein Hyperlink (ein GWT-Widget) aufgerufen oder ein History-Button benutzt wird, werden vom Stack Tokens entfernt oder hinzugefügt. Wenn das passiert, wird der HistoryListener benachrichtigt und die onHistoryChange() Methode aufgerufen, in der betimmt werden kann, was anhand des überlieferten Tokens passieren soll. Welcher Token übergeben wurde wird auch in der URL notiert, als <URL>#<tokenid>. So kann auch ein entsprechender Zustand der Applikation als Lesezeichen gespeichert und später wieder aufgerufen werden. Nazmul (01/2008)

In den Java-Dateien steckt die eigentliche Programmlogik. Wenn die Anwendung gestartet wird, wird zuerst die Klasse aufgerufen, die das EntryPoint-Interface implementiert hat. Dies steht in der Datei mit dem Namen <Modulname>.gwt.xml.

```
<module rename-to="hello">
    <inherits name="com.google.gwt.user.User"/>
    <entry-point class="com.gwt.sample.client.Hello"/>
</module>
```

Diese Datei beinhaltet den genauen Klassenpfad zu der Startklasse, daneben werden auch bestimmte Module geladen oder weitere Informationen für den Compiler festgehalten, wie zum Beispiel die Namen der Servlet-Klassen. Die Entry-Klasse bildet den Anfang des Programms und wird als erstes aufgerufen.

```
...
public class Hello implements EntryPoint {

  public void onModuleLoad() {
    Button b = new Button("Click me", new ClickHandler() {
      public void onClick(ClickEvent event) {
        Window.alert("Hello, AJAX");
      }
    });
    RootPanel.get().add(b);
  }
}
```

Das EntryPoint-Interface deklariert die Methode OnModuleLoad(), die von der implementierenden Klasse definiert werden muss. Dort können nun sogenannte Widgets erstellt und dem Komponentenbaum hinzugefügt werden. Dies passiert über die RootPanel-Klasse, welche den <body>-Tag symbolisiert. Über die RootPanel.get().add() werden die Widgets direkt dem <body>-Tag angefügt, wenn in der get()-Methode eine Id angegeben wird (RootPanel.get("id").add(widget)), so wird das Widget an das Element mit der jeweiligen Id angefügt. Steyer (2007)

Beim Widget können EventHandler über den Konstruktor oder die addClickHandler()-Methode realisiert werden, wobei die EventHandler-Methoden variieren (z.B. KeyUpHandler oder MouseDownHandler), je nachdem welches Widget den EventHandler hinzufügt. Steyer (2007)

Bei GWT-Applikationen findet die gesamte Kommunikation meist nur auf dem Client statt. Der clientseitige Code wird in JavaScript und JavaScript enthaltenen HTML-Seiten kompiliert und der Browser kann so die Informationen direkt vom Client nehmen. Dies hat den Vorteil, dass die Anwendung wesentlich schneller wird, da keine Webseiten oder andere Dateien vom Server nachgeliefert werden müssen. Jedoch kommt es vor, dass bestimmte Informationen nur vom Server geliefert und bestimmte Funktionen nur vom Server ausgeführt werden können. GWT stellt dafür mehrere Möglichkeiten zur Verfügung, um eine Client-Server-Kommunikation zu realisieren. Zum Einen können eigene Http-Request-Objekte erstellt und an den Server versendet werden, weitaus einfacher und populärer ist jedoch die Server-Client-Kommunikation über Remote Procedure Calls (RPC).

Die GWT-Bibliothek stellt dafür mehrere Klassen zur Verfügung. Um einen RPC auszuführen müssen drei Klassen definiert werden: Google Inc. (2010a)

1. Ein Interface, das die GWT-Klasse RemoteService erweitert und die RPC-Methoden deklariert.
2. Eine serverseitige Implementation des oben genannten Interfaces, die die Klasse RemoteServiceServlet erweitert.
3. Ein asynchrones Gegenstück zu dem oben genannten Interface, welches letztendlich vom Client aufgerufen wird.

Als Beispiel dient eine einfache Implementierung eines GWT-Service, in der der Client einen String an den Server sendet, der diesen umdreht und zurückschickt. Wichtig zu beachten ist, dass der GWT-„Service“ nichts mit dem allgemeinen Term „Web Service“ zu tun hat. Google Inc. (2010a) Im ersten Schritt wird das clientseitige Interface erstellt, welches die RPC-Methoden zur Verfügung stellt. Das Interface erweitert das *RemoteService*-Interface der GWT-Bibliothek.

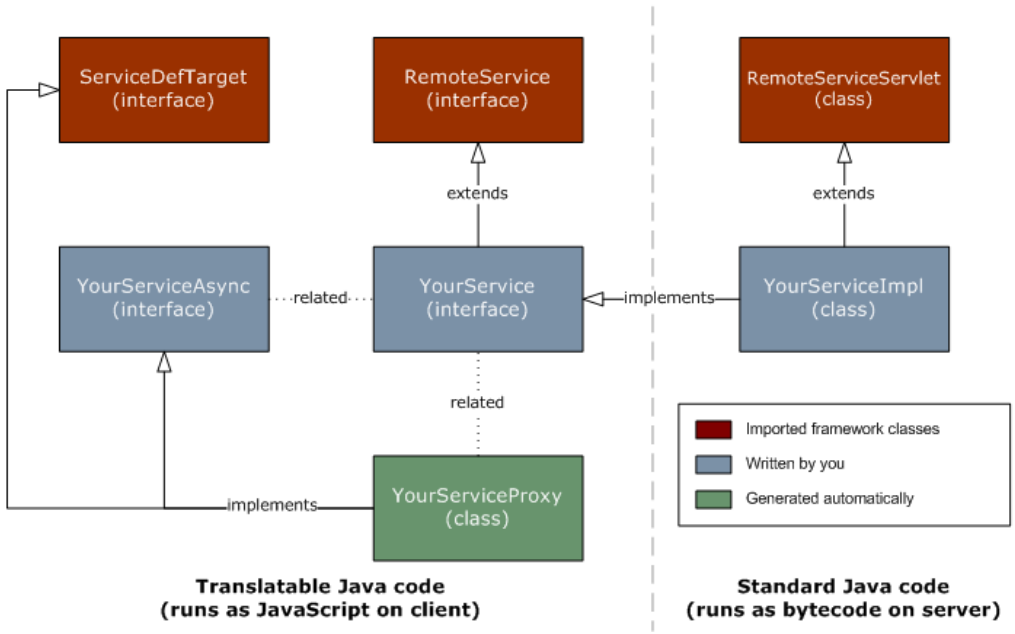


Abbildung 4.4: Architektur eines RPC-Aufrufs

```
@RemoteServiceRelativePath("StringReverserService")
public interface StringReverserService extends RemoteService {
  public String reverseString(String stringToReverse);
}
```

Die Annotation *RemoteServiceRelativePath* zeigt an, wo sich die Implementation des Interfaces auf dem Server befindet. Ein entsprechender Eintrag wird in der .gwt.xml Datei vorgenommen:

<servlet path="/StringReverserService"class="com.gwt.sample.server.

StringReverserServiceImpl"/>

Im zweiten Schritt wird auf der Serverseite die RPC-Methode implementiert. Diese Klasse erweitert die RemoteServiceServlet-Klasse aus der GWT-Bibliothek und implementiert das *StringReverserService*-Interface des Clients.

```
public class StringReverserServiceImpl
  extends RemoteServiceServlet implements StringReverserService{

  public String reverseString(String sRev) {
    StringBuffer reverse = new StringBuffer(sRev.length());
    for (int i = (sRev.length() - 1); i >= 0; i--)
      reverse.append(sRev.charAt(i));
```

```
8       return reverse.toString();
    }
10  }
```

Im dritten Schritt wird das asynchrone Interface des *StringReverserService* erstellt. Dazu muss zum Einen das Kürzel „Async“ hinter den Interfacenamen geschrieben werden, zum Anderen als zusätzlicher Parameter in den RPC-Methoden ein Objekt des Typs AsyncCallback übergeben werden. Dies sind Konventionen von GWT und müssen eingehalten werden.

```
public interface StringReverserServiceAsync {
2   void reverseString(String sRev, AsyncCallback async);
}
```

Dieses Interface ist notwendig, da bei GWT nur asynchrone Netzwerkoperationen erlaubt sind, das heißt nur Operationen, bei denen sofort etwas zurückgegeben wird. Der Grund dafür ist, dass in den meisten Browsern JavaScript nur „single-threaded“ läuft, also nur einen einzigen Thread zur Verfügung hat und dadurch bei dem Senden eines XMLHTTPRequest-Objekts die Oberfläche des Browsers kurz „einfriert“, bis eine Antwort erhalten wurde. Izabel (12/2007a) Da GWT mit Ajax auf extrem schnelle Webanwendungen baut, werden bei GWT keine synchronen Netzwerkoperationen erlaubt. Zu beachten ist hier noch, dass die asynchrone Methode keinen Rückgabewert hat, vielmehr wird der String über das callback-Objekt zurückgeliefert, welches auch den Client signalisiert, dass eine Rückgabe erfolgt ist.

Beim Aufruf des GWT-Services wird über die Methode GWT.create() aus der GWT-Bibliothek eine Instanz des synchronen Interfaces erstellt und in die asynchrone Variante gecastet. An der Instanz wird dann die RPC-Methode aufgerufen und der String, sowie ein AsyncCallback-Objekt übergeben.

```
1   private StringReverserServiceAsync getReverserServiceInstance(){
      if (reverserService == null)
3       reverserService = (StringReverserServiceAsync)
            GWT.create(StringReverserService.class);
5     return reverserService;
    }
7
    public void callStringReverserService(final String sRev) {
9     getReverserServiceInstance().reverseString(sRev,
        new AsyncCallback(){
11      public void onFailure(Throwable caught) {
          Window.alert("Failed to get response from server");
13      }
        public void onSuccess(Object result) {
```

```
15          String reverse = (String) result;
            Window.alert(reverse);
17        }
      });
19  }
```

Beim Erstellen des AsyncCallback-Objekts müssen zwei Methoden definiert werden. Bei onFailure() wird auf ein Fehlschlagen des RPC-Aufrufs reagiert, um zum Beispiel eine Fehlermeldung auszugeben; onSuccess() wird aufgerufen, wenn der Aufruf erfolgreich war.

Es können neben einfachen Strings auch jedes andere Objekt über einen Remote Procedure Call verschickt werden, sofern es serialisierbar ist. Dabei ist zu beachten, dass die GWT-Serialisierung eine andere ist als die Serialisierung, wie Java sie anbietet mit java.io.Serialization. Seit GWT 1.4 können zwar beide Serialisierungsarten benutzt werden, um ein Objekt zu serialisieren, jedoch wird empfohlen die von GWT bereit gestellte Serialisierung zu benutzen. Google Inc. (2010b) Folgende Objekte oder Typen können in einem RPC versendet werden:

- primitive Typen (int, float...)
- String, Date und Wrapper (Integer, Short, Float...)
- Enumerationen
- Einen Array von serialisierbaren Objekten
- Objekte, welche mit IsSerializable markiert oder Unterklassen davon sind

Mittlerweile ist GWT in der Version 2.0.4 erhältlich. Mit der Version 2.0 sind ein paar neue Features hinzugekommen. So gibt es nun einen **In-Browser Development Mode** mit dem es möglich ist mittels eines Plugins für den Browser direkt im Browser das Programm zu debuggen. Dabei wird über eine TCP-/IP-Verbindung direkt mit der Entwickler-Shell kommuniziert.Wikipedia (2010c) Bis jetzt werden aber noch nicht alle Browser unterstützt. Desweiteren kann jetzt die grafische Oberfläche über eine XML-Datei definiert werden, vorher war dies nur über Java-Code möglich. Wikipedia (2010c)

4.3.3 PHP

PHP ist eine von Rasmus Lerdorf enwickelte Programmiersprache. Ursprünglich die Abkürzung für „Personal Home Page Tools“ steht PHP nun für das rekursive Akronym „PHP: Hypertext Preprocessor“. Ab 1997 entwickelten Andi Gutmans und Zeev Suraski die PHP Version 3, welche als Startschuss für die große Verbreitung von PHP angesehen werden kann. Das

von Gutmans und Suraski gegründete Unternehmen *Zend Technologies* ist seitdem maßgeblich an der Entwicklung von PHP beteiligt. Die aktuellste Version ist PHP 5.3.3, mit der 5.0 Version wurden unter anderem objektorientierte Sprachkonstrukte, wie Überladung oder Sichtbarkeits-Schlüsselwörter (public/private/protected), Exceptions und Reflections ermöglicht.Wikipedia (2010e)

PHP wird vor allem für das Erstellen von dynamischen Webseiten und Webanwendungen benutzt. PHP-Dateien weisen immer die Endung .php auf und können PHP, sowie HTML-Code beinhalten. PHP-Codestücke müssen von dem Tag <?php ... ?> umschlossen sein. Der PHP-Code wird dabei vom Server ausgeführt, danach wird der fertige HTML-Code an den Client gesendet. Das Hello-World-Beispiel sieht in PHP folgendermaßen aus:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
    "http://www.w3.org/TR/html4/loose.dtd">
<html>
 <head>
   <title>Hallo-Welt-Beispiel</title>
 </head>
 <body>
   <?php
     echo "<p style='color:red;' align=\"center\">Hallo Welt!</p>";
   ?>
 </body>
</html>
```

Der PHP-Code ist direkt im HTML-Code eingebettet. Durch den Befehl **echo** wird der darauf folgende String in den HTML-Code übernommen. Um dadurch Strings in HTML-Code darzustellen, können entweder einfache Apostrophe verwendet werden oder mit dem Escapezeichen versehende Anführungsstriche.

Durch diese Einbettung des PHP-Codes in die HTML-Umgebung können auch komplexere Anweisungen ausgeführt werden, die über HTML nicht möglich sind, zum Beispiel Schleifen und bedingte Anweisungen.

```
<?php
    $zahl = 2;
?>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
    "http://www.w3.org/TR/html4/loose.dtd">
<html>
 <head>
   <title>Hallo-Welt-Beispiel</title>
 </head>
 <body>
```

```
    <ul>
12   <?php
         for($i = 0; $i < 5; $i++){
14       if($i == $zahl) echo "<li style='color:red;'>Zahl: ".$i."</li>";
         else "<li>Zahl: ".$i."</li>";
16     }
     ?>
18  </ul>
   </body>
20 </html>
```

In diesem Beispiel wird eine HTML-Liste erstellt, deren Inhalt dynamisch über eine for-Schleife eingefügt wird. Variablen und Strings können mit einem Punkt konkateniert werden. PHP ist eine dynamisch typisierte Sprache, daher müssen keine Typen angegeben werden, wenn Variablen deklariert werden. Wenn der Compiler eine Variable nicht kennt, wird sie automatisch initialisiert.

Der große Vorteil von PHP ist die direkte Einbindung in den HTML-Code. Dadurch hat man die volle Kontrolle über die HTML-Ausgabe und hat somit auch keine automatisch erstellten HTML-Codezeilen. Ausserdem geht es mit PHP sehr einfach und schnell eine simple dynamische HTML-Anwendung zu erstellen, dies liegt auch an der großen Verbreitung und damit guten Dokumentation und daran, dass PHP von so gut wie jedem Webserver unterstützt wird. Ein Nachteil und eine häufige Problemquelle kann die schwache Typisierung von PHP darstellen. Wie oben erwähnt initialisiert der Compiler sofort eine Variable, wenn er die Variable nicht kennt. Durch Tippfehler beim Variablennamen kann es so zu sehr schwer auffindbaren Fehlern kommen. Allerdings bietet PHP eine Möglichkeit eine umfangreichere Fehlerberichterstattung über den Befehl *error_reporting()* zu erreichen, womit solche Fehler besser gefunden werden können. Wikipedia (2010e)

4.4 Vergleich der angewendeten Technologien

Obwohl JSF als auch GWT die Sprache Java als Grundlage benutzen, sind sie trotzdem in der Art der Entwicklung einer Applikation grundverschieden. Das fängt schon bei dem Programmaufbau an. Bei JSF arbeitet der Entwickler ganz nach dem Model-View-Controller Prinzip, die drei Einheiten sind strikt getrennt, sowohl logisch als auch physisch. In GWT verschwimmen die Grenzen des MVC-Prinzips, der Entwickler arbeitet fast komplett in Java und hat somit die physische Trennung der einzelnen Teile überwunden. Stattdessen rückt vielmehr die Client-Server-Trennung in den Vordergrund. Bei JSF befindet sich die gesamte Anwendungslogik immer auf dem Server, nur die Präsentation des Programms ist auf dem Client vorhanden. Bei GWT jedoch wird durch den Ajax-Ansatz und den GWT-Compiler die

komplette Anwendungslogik per JavaScript auf den Client übertragen. Dadurch rückt für den Entwickler die Frage in den Vordergrund, welche Teile des Programms schnell auf dem Client erreichbar sein sollen und welche Teile auf dem Server belassen werden, um diese dann per RPC an den Client zu senden. Bei beiden Frameworks kann der Aufbau des Models, also die Implementation der fachlichen Architektur, identisch gehalten werden. Den grundlegendsten Unterschied gibt es in der Implementation des Frontends.

Im Gegensatz zu den beiden javabasierten Sprachen benutzt PHP eine eigene auf den Webbereich ausgelegte Sprache. Aufgrund dessen ist PHP eine sehr spezialisierte Technologie, die für die Entwicklung von dynamischen Webanwendungen ausgelegt ist. Darüber hinaus bietet PHP kaum eine Trennung zwischen Logik und Präsentation, da alles in .php Dateien geschrieben wird und somit Teile der Logik neben die HTML-Präsentation geschrieben werden könnten.

	JSF	**GWT**	**PHP**
Typisierung	statisch	statisch	dynamisch
Sprachparadigma	objektorientiert	objektorientert	imperativ/objektorientiert
Trennung Logik - Präs.	ja	schwach	schwach
JavaScript-abhängig	wenig	sehr stark	nein

Tabelle 4.1: Vergleich der Technologien

4.5 Weitere wichtige Web-Frameworks

Weitere Webframeworks, die sich als Standard etabliert haben sind RubyOnRails, welches auf Ruby basiert und ebenfalls dem MVC-Prinzip folgt und mittlerweile in der Version 2.3 erhältlich ist. RubyOnRails arbeitet nach den Prinzipien „Don't repeat yourself“ (DRY) und „Convention over Configuration“ (COC), welches eine sehr schnelle und agile Softwareentwicklung möglich macht. Das DRY-Prinzip besagt, dass Informationen nur ein einziges Mal vorhanden sein sollten, zum Beispiel sollten Tabelleneinträge nur ein einziges Mal und zwar in der Datenbank vorhanden sein, von dort aus können sie direkt ausgelesen und ausgegeben werden ohne die Daten im Quellcode explizit in Variablen zu speichern. Das COC-Prinzip erwartet, dass bestimmte Namenskonventionen eingehalten werden, die es dem Programm möglich machen ohne explizite Angaben die richtigen zusammengehörigen Daten auszugeben. Dadurch ist es möglich in RubyOnRails sehr schnell einfache Web-Anwendungen zu erstellen. Wikipedia (2010f)

Weiter gibt es noch ASP.NET welches C# als Sprache benutzt. ASP.NET hat sich aus der veralteten Technologie Active Server Pages (ASP) entwickelt und baut nun auf dem .NET-Framework auf. Neu ist zum Beispiel, dass durch das „Code-Behind-Prinzip“ bei der jeder Web-Datei eine Klasse zugeordnet wird, eine vollständige Trennung von Programmcode und HTML-Layout erreicht wird. Durch die Anbindung an das .NET-Framework können alle davon unterstützten Sprachen benutzt werden, so zum Beispiel VisualBasic oder J#. Wikipedia (2010a).

Daneben gibt es noch Frameworks, die auf Sprachen aufbauen, die eher im kleinen Kreis populär sind, wie Django, welches auf Python aufbaut oder Catalyst, welches die Sprache Perl benutzt.

5 Umsetzung

5.1 Umsetzung mit JSF

Als Erstes wurde das Szenario mit der JavaServer Faces Technologie umgesetzt, da der Autor dieser Arbeit mit Java und auch mit JSF vertraut ist und so eine schnelle Referenzanwendung erstellen werden konnte. Jedoch stellte sich die Entwicklung des Szenarios mit JSF schwieriger heraus, als zu Anfang gedacht, da mit vielen Problemen gekämpft werden musste, welche mit der Entwicklungsumgebung, aber auch mit der Technologie zu tun hatte. Mehr dazu gibt es unter dem folgenden Abschnitt.

5.1.1 Technologien und Probleme

Als Entwicklungsumgebung wurde Eclipse benutzt, welches sich mit der Entwicklung von Java-Anwendung bewährt hat und eine sehr umfangreiche Funktionspalette vorweisen kann. Dies ist auch nicht verwunderlich, war Eclipse ursprünglich eine reine Entwicklungsumgebung für Java. Eclipse ist frei erhältlich und läuft unter der Eclipse Public License (früher Common Public License), die von der Free Software Foundation anerkannt wird und die Möglichkeit bietet, unter der Lizenz auch Plugins zu veröffentlichen, aber wenn gewollt, auch unter einer anderen Lizenz. Es wurde die neueste Version Eclipse Helios IDE for Java EE Developers verwendet.

Für die JSF Implementierung wurde die neueste Version von JSF benutzt, Mojarra JSF 2.0, die von Sun Microsystems/Oracle zur Verfügung gestellt wird. Nötig sind dabei die beiden JSF jar-Dateien (jsf-api.jar und jsf-impl.jar) und die JSTL jars (jstl-api.jar und jstl-impl.jar). JSTL, Abkürzung für JavaServer Pages Standard Tag Library, ist eine Sammlung von Custom-Tag-Bibliotheken, welche einige nützliche Tags bieten, wie zum Beispiel eine Iteration mit <c:forEach>...</c:forEach> oder bedingte Anweisungen mit <c:if test=“Bedingung“>...</c:if>.

Als Testwebserver wurde der Apache Tomcat 6.0 benutzt, der einfach in Eclipse integriert werden kann und Dateien, die geändert worden sind, sofort auch aktualisiert. Somit kriegt man einen sofortigen Rückschluss auf die Veränderung, die gemacht worden ist und es

gewährleistet eine schnelle Entwicklung.

Wie oben erwähnt gab es bei dem Aufsetzen der Entwicklungsumgebung einige Schwierigkeiten. So ist die neueste Version des Java Development Kits (JDK v. 1.6.0_21) nicht mit der neusten Eclipse Version kompatibel und es kommt nach einiger Zeit zum Absturz von Eclipse. Mit der vorigen Version jdk1.6.0_20 gibt es hingegen keine Probleme. Zusätzlich kommt es zu einigen Kompabilitätsproblemen mit der neuesten JSF-Version. Um die neuesten Funktionen nutzen zu können (wie Ajax Integration), müssen die Dateien die Endung *.xhtml vorweisen. Jedoch ist es in Eclipse nicht möglich XHTML-Dateien zu editieren und gleichzeitig Unterstützung in der JSF-Entwicklung, wie Content Assist etc., zu bekommen. Somit ist man in der Zwickmühle, entweder auf die neuen JSF-Features zu verzichten, dafür aber Unterstützung vom Editor zu bekommen oder andererseits die neuen Features zu benutzen, aber dafür auf die Konformitäten des Editors zu verzichten. Natürlich ist es möglich zwischen den Dateiendungen zu wechseln, jedoch kann das bei einer größeren Anwendung nicht der Stein der Weisen sein.

5.1.2 Model

Das Model wurde gemäß den UML-Klassendiagrammen aus dem Abschnitt 4.2.2 hin implementiert. Das Objekt, welches von der Anwendung als erstes erstellt wird, ist der Patient. Nachdem der Benutzer über die Loginmaske die Anwendung gestartet hat, werden alle Patienten aus der Datenbank geladen. In diesem Beispiel wird die Datenbank nur simuliert, das übernimmt die Klasse DBSimulator. Hier werden alle Patienten in einer Liste gehalten, zusätzlich können Patienten aktualisiert, gelöscht oder neue Patienten hinzugefügt werden über die entsprechenden Methoden. Dabei übernimmt der DBSimulator auch die Vergabe der IDs, die in aufsteigender Reihenfolge vergeben werden.

Auf den DBSimulator kann über die Klasse DBConnector zugegriffen werden, diese Klasse würde auch den Zugang zur echten Datenbank übernehmen. Dafür wird über eine Klassenmethode eine Instanz der Klasse mit einer *lazy initialization* erzeugt. Über dieses Objekt wird dann der Zugang zur Datenbank erzeugt.

Die Klasse Patient besteht aus ein paar Datenfeldern wie Name, Vorname, Geburtsdatum etc. mit den entsprechenden Gettern und Settern und hält eine Befundsliste. Die Klasse Befund hat die Datenfelder Diagnose, Erstellungsdatum und Therapeut und hat als Verlauf eine Liste mit Einträgen, die das Datum, den Therapeuten und Bemerkungen enthält.

Der Aufbau der Klassenhierarchie ist sehr javakonform und man könnte an das Model auch ein Swing-Frontend bauen, da die Anwendungslogik von der Ansicht abgekoppelt ist, wie es auch vom MVC-Prinzip verlangt wird. Als Schnittstelle zwischen der Logik und der Ansicht

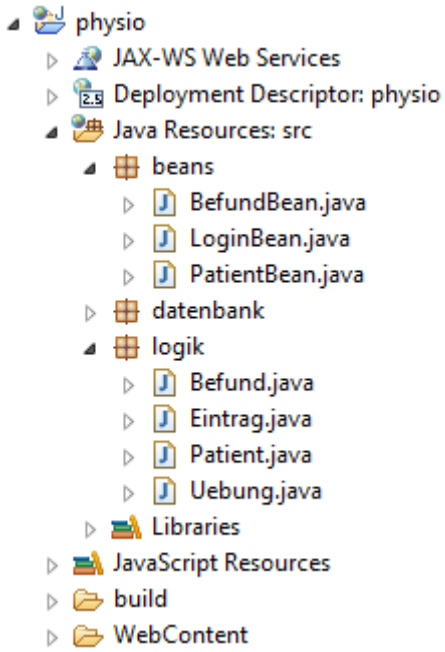


Abbildung 5.1: Ordnerstruktur in Eclipse (Model)

dienen in JSF die ManagedBeans, welche gerade aktive Objekte in Variablen hält und somit für die Ansicht bereit stellt. Die einfachste und auch die erste Bean, die der Anwender benutzt, ist die LoginBean, welche für den Login-Screen aufgerufen wird. Die LoginBean ist für diese Anwendung sehr einfach gehalten. Sie besteht aus zwei statischen Feldern, welche den Benutzernamen und das Password darstellen. Daneben noch zwei Variablen, um die Eingabe des Benutzers zu speichern und eine Actionmethode, welche nach Klicken des Login-Buttons aufgerufen wird.

```
package beans;

import javax.faces.bean.ManagedBean;
import javax.faces.bean.RequestScoped;

@ManagedBean
@RequestScoped
public class LoginBean {

    private final String LOGIN = "User";
    private final String PASSWORD = "geheim";

    private String _login;
```

```
14      private String _password;

16      public LoginBean(){}

18      public String getBenutzer(){
           return _login;
20      }
        public void setBenutzer(String login){
22         _login = login;
        }
24
        public String getPassword(){
26         return "";
        }
28      public void setPassword(String password){
           _password = password;
30      }

32      public String login(){
           if(_login.equals(LOGIN) && _password.equals(PASSWORD))
34               return "main";
           else return null;
36      }
    }
```

Die Bean ist nur als RequestScoped gekennzeichnet, sie wird also nach einer Http-Anfrage gelöscht und danach für die nächste Anfrage wieder neu erstellt. Die Methode *login()* ist die Actionmethode welche nach Betätigung des Login-Buttons aufgerufen wird. Sie überprüft, ob der eingegebene Benutzername und das Passwort mit dem in der Klasse gespeicherten Variablen übereinstimmt. Ist das der Fall, wird eine String zurückgegeben, der besagt, zu welcher Seite der Benutzer weitergeleitet werden soll. Der Inhalt des Strings ist ein eigens kreierter Name zur Kennzeichnung des Navigationspfades. Der entsprechende Eintrag wird in der *faces-config.xml* festgehalten. Dort wird über eine Navigationsregel der entsprechende Name auf den Navigationspfad abgebildet.

```
1   ...
    <navigation-rule>
3      <from-view-id>/login.jsf</from-view-id>
       <navigation-case>
5         <from-outcome>main</from-outcome>
          <to-view-id>/main.jsf</to-view-id>
7         <redirect />
```

```
    </navigation-case>
<navigation-rule>
 ...
```

Nachdem sich der Benutzer eingeloggt hat, kommt er auf die Hauptseite, dort wird auf die PatientBean zugegriffen. In dieser wird eine Liste mit allen Patienten gehalten, sowie der momentan „aktive“ Patient. Daneben gibt es noch eine Variable des Typs *HtmlDataTable*. In dieser Variable wird ein Objekt gespeichert, das an ein <h:dataTable>-Tag aus dem View gebunden ist. Es ist also eine javainterne Repräsentation eines JSF-Tags mit den Werten der Attribute des Tags. An das DataTable wird die Patientenliste übergeben, über die dann iteriert wird und die angegebenen Patientendaten ausgegeben werden. Wenn nun in der Tabelle in der Ansicht ein Patient angeklickt wird, so wird das an die BackingBean übertragen und man kann über das HtmlDataTable-Objekt Zugriff auf den entsprechenden „aktivierten“ Patienten bekommen.

```
...
@ManagedBean
@SessionScoped
public class PatientBean {

    HtmlDataTable _patientTable;
    List<Patient> _patientliste;
    Patient _aktiverPatient;
    ...
    public HtmlDataTable getPatientTable(){
        return _patientTable;
    }
    public void setPatientTable(HtmlDataTable hdt){
        _patientTable = hdt;
    }
    public String choosePatient(){
        _aktiverPatient = (Patient)_patientTable.getRowData();
        return "patient";
    }
}
```

Über die Get- und Setmethode wird das HtmlDataTable-Objekt an die Variable gebunden. Wenn ein Patient aktiviert wurde, so wird die choosePatient() Methode aufgerufen, dies ist wieder eine Actionmethode mit dem entsprechenden Navigationseintrag in *der faces-config.xml*. Über die getRowData()-Methode kann der entsprechende Patient extrahiert werden und wird in der Variablen *_aktiverPatient* gespeichert, um dann auf den Patienten direkten Zugriff zu haben.

Auf der folgenden Seite wird eine Tabelle mit allen Befunden des Patienten angezeigt. Hier kommt die BefundBean ins Spiel, welche ähnlich zur PatientBean die Befundliste, das HtmlDataTable-Objekt für die Befundtabelle und eine Variable für den „aktivierten“ Befund hält. Jedoch muss die BefundBean erst die Befundliste von dem aktivierten Patienten erhalten. Und da weder die PatientBean auf die BefundBean noch umgekehrt Zugriff haben, muss die Information andersweitig übermittelt werden. Die PatientBean ist allerdings als SessionScoped markiert und wird deswegen in einer Sitzung gehalten. Es gibt also direkten Zugriff über das Sessionobjekt auf die PatientBean und somit auf den Patienten und auf die Befundliste. Gemacht wird das über den FacesContext, welche alle Informationen enthält über den momentanen Request und Response und die Sessions.

```
...
@ManagedBean
@SessionScoped
public class BefundBean {
  ...
  private List<Befund> _befunde;

  public BefundBean(){
    FacesContext ctx = FacesContext.getCurrentInstance();
    HttpSession session = (HttpSession)ctx.getExternalContext().
      getSession(false);
    PatientBean bean = (PatientBean)session.
      getAttribute("patientBean");
    _befunde = new ArrayList<Befund>
      (bean.getPatient().getBefundeAll());
  }
    ...
}
```

5.1.3 View

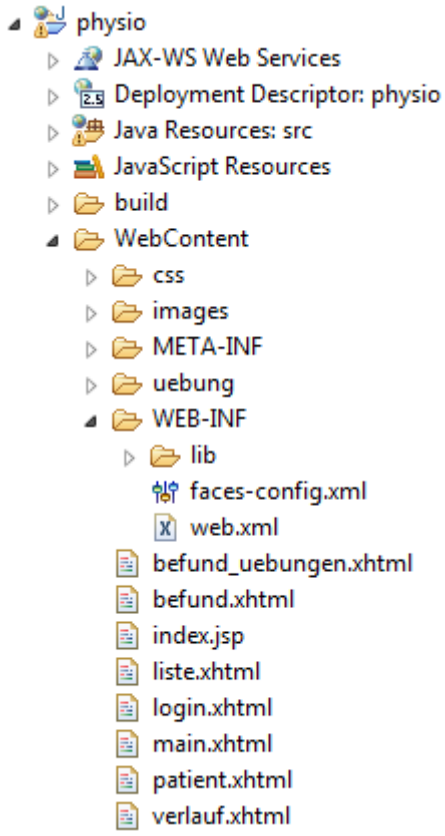


Abbildung 5.2: Ordnerstruktur in Eclipse (View)

Die Ansicht startet auf dem LoginScreen. Über eine Eingabemaske kann sich der Benutzer in das System einloggen. Dabei wird der Benutzername und das Password über ein <h:inputField> und ein <h:inputSecret> gespeichert und das Formular über ein <h:commandButton> abgeschickt.

```
...
<h:form id="loginForm">
 <h:panelGrid id="grid" columns="2">
   <h:outputLabel value="Login:" for="login"/>
   <h:inputText id="login" value="#{loginBean.benutzer}"/>
   <h:outputLabel value="Password:" for="password"/>
   <h:inputSecret id="password" value="#{loginBean.password}"/>
  <h:outputText/>
   <h:commandButton id="loginButton" value="Login"
      action="#{loginBean.login}" style="float:right;"/>
```

```
    </h:panelGrid>
12  </h:form>
    ...
```

<h:outputLabel> sind spezielle Textausgaben, die über das *for*-Attribut an ein Textfeld gebunden werden können. Durch einen Klick auf das Label wird dann das entsprechende Textfeld fokussiert.

Nach der Loginmaske wird der Benutzer auf die Hauptseite weitergeleitet, wo die Patientenliste ausgeben wird. Die Patiententabelle wird über ein <h:dataTable>-Tag realisiert, an dieses wird über das *value*-Attribut die Patientenliste aus der ManagedBean zugewiesen. Über die Liste wird dann automatisch iteriert, auf das einzelne Element wird über einen Variablennamen zugegriffen, der in dem *var*-Attribut deklariert wird. Die einzelnen Spalten der Tabelle werden mit <h:column> gekennzeichnet. Mit <f:facet name="header"> kann eine Spaltenüberschrift ausgegeben werden. Der <f:verbatim>-Tag bewirkt, dass das enthaltene Element innerhalb des JSF Komponentenbaums gerendert wird, da nur JSF-Tags dem Baum hinzugefügt werden und der Text sonst nicht fester Bestandteil des HtmlDataTables würde.

```
1   ...
    <h:dataTable id="patTable" binding="#{patientBean.patientTable}"
3       value="#{patientBean.patientliste}" var="patient"
        style="border-right:1px solid #a4a4a4">
5       <h:column id="nameColumn">
            <f:facet name="header">
7               <f:verbatim>Name</f:verbatim>
            </f:facet>
9           <h:form id="befundLink">
                <h:commandLink id="patientLink"
11                  value="#{patient.name}, #{patient.vorname}"
                    action="#{patientBean.choosePatient}">
13              </h:commandLink>
            </h:form>
15      </h:column>
        <h:column id="gebColumn">
17          <f:facet name="header">
                <f:verbatim>geb. am</f:verbatim>
19          </f:facet>
            <h:outputText id="patientGeb"
21              value="#{patient.gebDatum}">
            </h:outputText>
23      </h:column>
```

```
    </h:dataTable>
25  ...
```

Die Seite der Befundtabelle ist ähnlich aufgebaut wie die der Patiententabelle. Wenn ein neuer Befund erstellt werden soll, wird man auf die erste Seite der zweiseitigen NeueBefund-Seite wetiergeleitet. Die erste Seite ist aufgebaut aus einer Form und vielen InputFeldern. Die zweite Seite hingegen dient zum Auswählen von jeweiligen Übungen, die der Patient durchführen sollte. Dabei wird eine Auswahl aus einigen Übungen zur Verfügung gestellt. Die Übungen werden über ein DataTable aufgelistet und es können über ein Klick auf die jeweilige Übung weitere Informationen abgerufen werden. Dabei wird die Informationsseite zur Übung dynamisch in die Seite eingebunden. Dies passiert über ein <ui:include>-Tag, mit der eine seperate Seite in die vorhandene Seite integriert werden kann. Wenn der Info-Button gedrückt wird, wird der Seitenname der entsprechenden Informationsseite in der Bean gespeichert und mit <ui:include> kann auf den String zugegriffen werden und die entsprechende Seite eingebunden werden.

```
1   ...
    <div class="zelle" style="right:1px;">
3       <ui:include src="/uebung/${befundBean.uebungGross}.xhtml" />
    </div>
5   ...
```

Über Checkboxen können die gewollten Übungen ausgewählt und über den CommandLink bestätigt werden, wonach die Daten in dem Model gespeichert werden und der neue Befund in der Befundtabelle erscheint.

5.2 Umsetzung mit GWT

5.2.1 Technologien und Probleme

Wie bei der JSF-Umsetzung wurde auch bei der Umsetzung mit GWT Eclipse Helios als Entwicklungsumgebung eingesetzt. Mit Version 2.0.4 wurde die neueste Version des Google Web Toolkits benutzt (Stand: 03.08.2010). Als Browser für den Development Mode wurde Firefox benutzt.

Das GWT-Software Development Kit wurde per Plugin in Eclipse eingebettet, so wurde das Interface von Eclipse an die GWT-Entwicklung angepasst. Nach der Erstellung eines Web Application Projects wird die komplette GWT-Ordnerstruktur hergestellt mit allen notwendigen Dateien und Bibliotheken. Zusätzlich dazu wird auch ein kleines GWT-Beispiel erzeugt,

die als Starterklassen benutzt werden können.

Ein großes Problem von GWT ist die mangelnde Unterstützung für Java-Bibliotheken. Standardmäßige Bibliotheken wie java.text, java.io oder java.util fallen teilweise oder auch komplett raus und können für die Entwicklung der GWT-Applikation nicht benutzt werden. Besonders fällt dies auf, zum Beispiel, bei einem einfachen und häufigen Problem, wie das Formatieren und Ausgeben eines Datums. Da die Klassen java.text.SimpleDateFormat und java.util.Calendar nicht unterstützt werden, muss auf die *Deprecated*-Methoden von java.util.Date zurückgegriffen werden, die eigentlich nicht mehr benutzt werden sollen, da sie nur mangelhaft implementiert worden sind.

5.2.2 Client

Die fachliche Architektur ist gegenüber der Implementierung von JSF gleich geblieben. Lediglich die Anfrage der Patientenliste wird nun von einem Remote Procedure Call vom Client an den Server übernommen.

Die Startklasse der GWT-Applikation ist die Entry-Point-Klasse. Diese Klasse implementiert das Interface Entry-Point und wird in der .gwt.xml Datei notiert. Die EntryPoint-Klasse initialisiert die Patiententabelle. Die Patiententabelle wird letzten Endes von der Klasse PatientTable erstellt, die die nötigen HTML-Objekte erstellt und sie dem RootPanel zufügt. Die Patientenliste wird der Klasse PatientTable übergeben, nachdem diese über einen RPC vom Server geholt worden ist.

```
1  ...
   public class PhysioGWT implements EntryPoint {
3
       private PatientTable _patientTable;
5      private DBConnectServiceAsync _dbservice;

7      public static void setContent(String id, Widget w){
           RootPanel.get(id).clear();
9          RootPanel.get(id).add(w);
       }
11
       public void onModuleLoad() {
13         _patientTable = new PatientTable(new ArrayList<Patient>());
             getPatientList();
15         RootPanel.get("patient").add(_patientTable);
       }
17     //Service wird über Lazy Initialization erstellt
```

```
    private DBConnectServiceAsync getDBServiceInstance(){...}

    public void getPatientList(){
        getDBServiceInstance().getPatientList(
            new AsyncCallback<List<Patient>>() {
            @Override
            public void onFailure(Throwable caught) {
                caught.printStackTrace();
                Window.alert(caught.getMessage());
            }
            @Override
            public void onSuccess(List<Patient> result) {
                _patientTable.setInput(result);
            }
        });
    }
}
```

Die Methode setContent() dient dazu, von überall durch eine einzige Methode in die von der HTML-Seite definierten Bereiche dynamisch neue Inhalte einzubinden. Dazu werden in diesem Bereich durch RootPanel.get(id).clear() alle Kindsobjekte des Html-Objekts mit der ID „id“ gelöscht und danach ein neues Objekt hinzugefügt.

Die Patientenliste wird über die Methode getPatientList() geholt. Mit einer Lazy Initialization wird das Service-Objekt erstellt und die getPatientList()-Methode des Services aufgerufen. Da es sich um die Async-Variante des Services handelt, muss ein AsyncCallback-Objekt übergeben werden. Über die onSuccess()-Methode wird die Patientenliste gefüllt.

Die Patientenliste wird dem PatientTable-Objekt übergeben. PatientTable ist vom Typ *FlexTable*, kann demnach als Widget dem RootPanel hinzugefügt werden.

```
...
    public class PatientTable extends FlexTable {

    public PatientTable(List<Patient> patienten){
        ... //Style-Änderungen werden vorgenommen
        this.setInput(patienten);
    }
    public void setInput(List<Patient> patienten){
        this.removeAllRows();
        this.setText(0, 0, "Name");
        this.setText(0, 1, "geb am");
        if(patienten.isEmpty()) return;
```

```
14          int row = 1;
            for(final Patient p : patienten){
16              Anchor link = new Anchor(p.getName()+
                    ", "+p.getVorname(), "#");
18              link.addClickHandler(new ClickHandler(){
                    @Override
20                  public void onClick(ClickEvent event) {
                        PhysioGWT.setContent("befund",
22                          new BefundTable(p));
                    }
24              });
                this.setWidget(row, 0, link);
26              this.setText(row, 1, p.getGebDatum());
                row++;
28          }
        }
30      }
```

In der Methode setInput() wird die HTML-Tabelle gefüllt. Dazu kann über setText(zeile, spalte, text) oder setWidget(zeile, spalte, widget) entweder reiner Text oder ein Widget einer Zelle hinzugefügt werden. Als Link, um die Befundtabelle eines Patienten aufzurufen, wird ein Anchor-Widget verwendet, da die Linkoptik beibehalten werden soll und das Hyperlink-Widget nur für die Historyfunktionen benutzt werden sollte.Google Inc. (2010c) Über den ClickHandler werden dann die Befunde des Patienten mit einer BefundTable-Instanz geladen.

Die Klasse BefundTable ist analog zu der Klasse PatientTable aufgebaut. Durch einen Klick auf einen Befund wird die Klasse BefundView instanziiert, die das Erstellen oder das Bearbeiten eines Befundes regelt. Da das Befundformular zwei Seiten umfasst, hat die Klasse BefundView eine Methode showPage1() und showPage2().

```
    ...
2   public class BefundView extends FlowPanel{
        ... //Datenfelder und Konstruktor
4       private void showPage1(){...}
        private void showPage2(){
6           this.clear();
            FlexTable table = new FlexTable();
8           this.add(table);

10          int row = 0;
            for(final Uebung u : _befund.getAlleUebungen()){
12              table.setWidget(row, 0, new InlineLabel(u.getName()));
```

```
            Image i = new Image("images/uebung/magnifier.png");
14          i.addClickHandler(new ClickHandler(){
               @Override
16             public void onClick(ClickEvent event) {
                  HTML html = new HTML(u.getDescription());
18                if(getWidgetCount() > 2) remove(2);
                  add(html);
20             }
            });
22          table.setWidget(row, 1, i);
            CheckBox check = new CheckBox();
24          check.setValue(u.getChoosed());
            check.addClickHandler(new ClickHandler(){
26             @Override
               public void onClick(ClickEvent event) {
28                u.setChoosed(((CheckBox)
                     event.getSource()).getValue());
30             }
            });
32          table.setWidget(row, 2, check);
            row++;
34       }
         ... //Submit-Button wird erstellt
36    }
   }
```

BefundView erbt von der Klasse FlowPanel und kann somit dem HTML-Komponentenbaum hinzugefügt werden. In der Methode showPage1() werden ein paar Labels und Inputfelder erstellt und dem BefundView hinzugefügt und abschließend ein SubmitButton, welcher beim onClick() die eingegebenen Daten an das Model weitergibt und showPage2() aufruft.

Auf der zweiten Seite werden die Übungen für den Patienten ausgewählt. Dazu werden die Übungen in einer Liste präsentiert und können über eine Checkbox ausgewählt oder es können nähere Informationen zur Übung eingesehen werden. Pro Übung wird ein Image-Widget erstellt, welches einen ClickHandler bereitstellt, der auf Knopfdruck die Beschreibung der Übung dem BefundView hinzufügt und ausgibt. Die Beschreibung wird vom Model als String mit eingebauten HTML-Tags übergeben. Mit dem Html-Objekt kann dieser HTML-String verareitet und das Ergebnis ausgegeben werden. Über die Checkbox kann eine Übung ausgewählt werden, dazu wird über den ClickHandler der Übung sofort mitgeteilt, dass sie ausgewählt wurde. Der Grund, dass das sofort gemacht wird ist, dass so bis zur Bestätigung des Submit-Buttons die booleschen Werte der Checkboxen nicht temporär lokal gespeichert werden müssen, sondern nur in der Übung selbst.

Eine Übersicht der GWT-Ordnerstruktur ist unter Abbildung 5.3 zu finden.

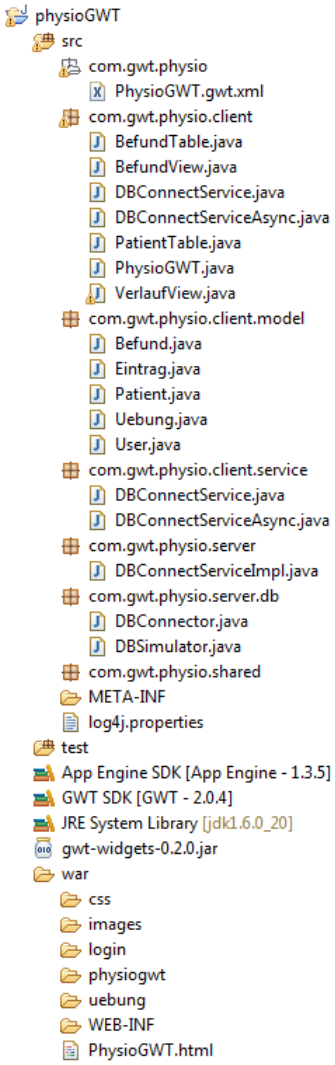


Abbildung 5.3: Ordnerstruktur von GWT (Eclipse)

5.2.3 Server und Service

Da die Datenbank auf dem Server liegt, muss die Patientenliste, welche nur aus der Datenbank erhältlich ist, über einen Remote Procedure Call an den Client gesendet werden. Dazu wird der Service DBConnectService erstellt, der analog zum Service unter Kapitel 4.3.2 erstellt wird. Die Implementierung des Services *DBConnectServiceImpl* hat eine Methode *getPatientList()*, bei der die Datenbank kontaktiert wird und die Patientenliste als Rückgabewert zurück gegeben wird.

```
...
public class DBConnectServiceImpl
    extends RemoteServiceServlet
        implements DBConnectService {
    @Override
    public List<Patient> getPatientList() {
        return DBConnector.getConnector().
            getConnection().getPatientListe();
    }
}
```

Bei einer realen Datenbank im Backend würde diese Methode durch andere ersetzt werden, die als Argument einen SQL-Query-String überreicht bekommen und die entsprechenden Objekte dann als Rückgabewert liefert.

5.3 Umsetzung mit PHP

Die PHP-Implementation wird in dieser Arbeit nur kurz angerissen, da der Hauptaugenmerk auf JSF und GWT liegt und PHP nur als Vergleich zu einer alteingesessenen und weitverbreiteten Sprache herangezogen wird.

5.3.1 Technologien und Probleme

Es wurde die PHP-Version 5.3.1 verwendet. Als Webserver fungierte der Apache Server mit XAMPP Version 1.7.3. Als Editor wurde Notepad++ in der Version 5.6.8 benutzt. Bei der Implementation und Installation traten keine Probleme auf.

5.3.2 Implementierung

Wie bei den anderen Umsetzungen startet die Anwendung mit der Login-Seite. Die Login-Seite wurde von der JSF-Umsetzung übernommen und um die PHP-Codestücke erweitert.

```
...
<div style="margin: 5% auto; width:200px;">
    <h1>Physiotherapie</h1>
    <h2>Bitte einloggen</h2>
    <?php
```

```
6       if(isset($_GET["success"])){
            if($_GET["success"] == 0) echo "<p style='color:#ff0000;'>Die
                Eingabedaten sind falsch!</p>";
8           else echo "<p style='color:#50cd59'>Sie haben sich erfolgreich
                ausgeloggt!</p>";
        }
10  ?>
    <form id="form" method="post" action="auth.php">
12      <label for="benutzerInput">Benutzer</label>
        <input type="text" id="benutzerInput" name="benutzerInput"/>
14      <label for="passwortInput">Passwort</label>
        <input type="password" id="passwortInput" name="passwortInput"/>
16      <input type="submit" value="Login"/>
    </form>
18 </div>
   ...
```

Mit *isset()* kann überprüft werden, ob bestimmte Variablen oder wie in diesem Beispiel ein Key-Value-Paar gesetzt sind. Mit $_GET[„key“] können Werte benutzt werden, die über die URL angehängt worden sind, zum Beispiel „.../seite.php?key=value“. Die Eingaben des Formulars werden dann mit der Post-Methode an eine nächste Seite übergeben, die die eingegebenen Daten auswertet und den Benutzer authentifiziert. Im Gegensatz zur Get-Methode werden bei der Post-Methode keine Werte an die URL angehängt, sondern die Werte werden über eine spezielle Post-Anfrage an den Webserver verschickt und können somit nicht vom Benutzer eingesehen werden.

Die nächste Seite, welche die Daten des Benutzers authentifiziert, ist eine reine PHP-Seite, die beim fehlgeschlagenen Login den Benutzer auf die Login-Seite zurückwirft oder beim erfolgreichen Login den Benutzer auf die Hauptseite der Anwendung leitet. Da es sich um eine reine PHP-Seite handelt, die nur zur Weiterleitung benutzt wird, bekommt der Benutzer davon nichts mit.

```
1  <?php
       class User{
3          private $name = "";
           private $password = "";
5
           function __construct($login, $pass){
7              $this->name = $login;
               $this->password = $pass;
9          }
           function auth($login, $pass){
11             return $login == $this->name and $pass == $this->password;
           }
```

```
13      }

15      $user = array(new User("user1", "pw"), new User("user2", "pwd"));
        $login = $_POST["benutzerInput"];
17      $pass = $_POST["passwortInput"];

19      foreach($user as $each){
            if($each->auth($login, $pass)){
21              session_start();
                $_SESSION["user"] = $login;
23              header("Location: main.php?sid=".session_id());
            }
25      }
        header("Location: login.php?success=0");
27  ?>
```

Es wird eine Klasse User erstellt, welche die Datenfelder name und password beinhaltet. Die Funktion __construct ist der Konstruktor des Objekts. Über die Funktion **auth(name, password)** wird der Benutzer authentifiziert. Über $_POST[„inputname“] wird der eingegebene Benutzername und das Passwort aus dem Formular ausgelesen. Wenn der richtige Benutzer gefunden ist, wird eine Session gestartet, welche den Benutzernamen beinhaltet. Auf diese Session kann über eine Session-ID auf jeder Seite zugegriffen werden und somit der richtige Benutzer augelesen werden. Bei einem Logout wird die Session gelöscht. Über die Funktion **header(“Location: seite.php“)** wird der Benutzer auf eine andere Seite weitergeleitet, dabei wird die Session-ID mit übergeben, um die Session auf der nächsten Seite abrufen zu können.

Auf der Hauptseite werden die Patienten aufgelistet. Dabei wird eine HTML-Tabelle dynamisch mit Patienten gefüllt.

```
1   <?php
        session_start();
3       include "model/Patient.php";
        $patientliste = Patient::getPatientList();
5   ?>
    ...
7   <div class="logout">
        Hallo, <?php echo $_SESSION["user"]; ?> | <a href="logout.php">Logout
            </a>
9   </div>
    <div class="patientWindow">
11      <a href="neuerPatient.php">Neuer Patient</a><br>
        <table id="patientTable">
13          <?php
```

```
            foreach($patientliste as $patient){
15              echo "<tr>";
                echo "<td><a href='befundliste.php?sid=".session_id()."&pid="
                    .$patient->id."'>".$patient->name.", ".$patient->vorname."
                    </a></td>";
17              echo "<td>".$patient->gebDatum."</td>";
                echo "</tr>";
19          }
        ?>
21  </table>
</div>
23  ...
```

Zuerst wird die Session gestartet, dabei wird automatisch die Session-ID genommen, die entweder über die GET- oder über die POST-Methode gesendet wurde. Mit **Patient::getPatientList()** wird die statiche Methode *getPatientList()* aufgerufen und in einer Variablen gespeichert. Über eine foreach-Schleife werden die relevanten Patientendaten in die HTML-Tabelle übernommen, dabei wird über die GET-Methode die Patienten-ID des gewählten Patienten an die Seite mit der Befundliste übergeben.

Im Gegensatz zu den anderen Umsetzungen gibt es in PHP keine Möglichkeiten HTML-Seiten dynamisch zu verändern, ohne die komplette Seite neu zu laden. Um das zu erreichen, muss in PHP auf einfaches Java-Script zurück gegriffen werden. In GWT wird dieser Part vom GWT Compiler übernommen, bei PHP muss dies wenn nötig selbst codiert werden. Das dynamische Einbinden einer HTML-Seite kann jedoch ohne Ajax-Unterstützung oder Listenern erreicht werden. Hierzu kann der Tag <iframe></iframe> benutzt werden.

```
1   ...
    <table class="uebung_table">
3   <?php
        foreach($befund->uebungen as $uebung){
5           echo "<tr><td>".$uebung->name."</td>";
            echo "<td><a><img src='images/uebung/magnifier.png' onclick=\"
                document.getElementById('uebung_big').src='uebung/".$uebung->
                pagename.".html';\"/></a></td>";
7           echo "<td><input type='checkbox' id='uebungCheck' name='uebungCheck
                []' value='".$uebung->id."'/></td></tr>";
        }
9   ?>
    </table>
11  ...
    <div class="zelle" style="right:1px;">
13      <iframe class="uebung_big" id="uebung_big"  src=""></iframe>
    </div>
```

```
15  ...
```

Der <iframe>-Tag erlaubt es über das src-Attribut eine Seite einzubinden. Über JavaScript kann nun über die ID des iFrames eine Seite dynamisch gesetzt werden.

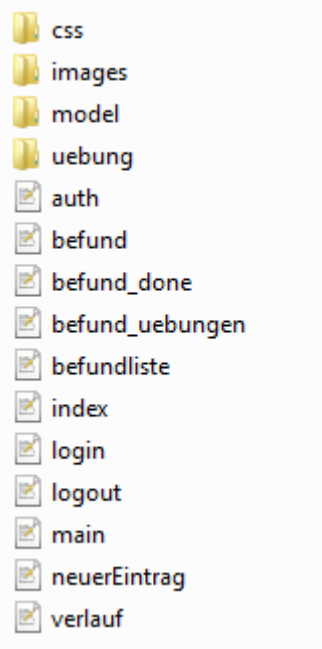


Abbildung 5.4: Ordnerstruktur von PHP (Windows)

6 Bewertung des Endproduktes

Nachdem das Szenario mit mehreren Webframeworks realisiert worden ist, geht es darum, die einzelnen Webframeworks mit den unter Abschnitt 2.2 erläuterten Kriterien zu messen.

6.1 Das Testbett

Das Testbett oder auch die Testumgebung besteht einerseits aus einem eigens erstellten Analysetool zur Messung der Änderbarkeit, andererseits aus einem Performancetool zur Messung der Effizienz und Zuverlässigkeit. Die Ergebnisse für die Übertragbarkeit und den Aufwand sind subjektive Krtierien, die keiner objektiven Analyse bedürfen.

Das Tool zur Messung der Änderbarkeit ist ein simples Werkzeug, welches den Maintainability Index eines Projekts ausgibt. Zur Auswahl stehen Messungen von JSF-/GWT- und PHP-Projekten. Angegeben werden muss, neben der Auswahl an Frameworks, der Projektordner und das Tool analysiert alle relevanten Dateien, das heißt im Falle von JSF alle .java Dateien und gibt den entsprechenden Teil-MI zurück, der am Ende zum schlussendlichen Index zusammen gerechnet wird.

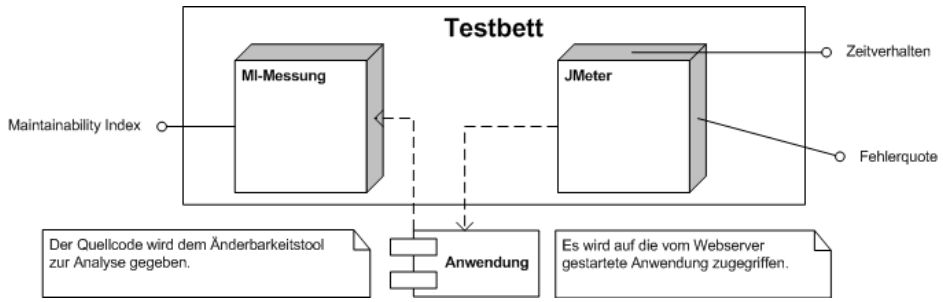


Abbildung 6.1: Aufbau des Testbetts

Das Werkzeug, um die Effizienz und die Zuverlässigkeit der Applikation zu messen ist das frei erhältliche Tool JMeter, welches über die Apache Jakarta Project Seite runtergeladen werden kann.Apache Software Foundation (2010) Das Tool bietet die Möglichkeit einen Webserver zu überwachen, um Messungen durchzuführen, wie zum Beispiel die Ladezeit oder den Ressourcenverbrauch.

6.2 Analyse

Die Kriterien, die für die Analyse in Betrachtung gezogen werden, sind Änderbarkeit, Effizienz, Zuverlässigkeit und Übertragbarkeit. Diese Kriterien wurden ausgewählt, da sie gut zu messen sind und aussagekräftig genug sind, um die Webframeworks aufgrund der Kriterien bewerten zu können.

6.2.1 Änderbarkeit

Wie unter Abschnitt 2.2.1 formuliert, wird die Änderbarkeit mit Hilfe des Maintainability Index (MI) gemessen. Der MI benutzt Zeilenmetriken, zyklomatische Zahl und das Halstead Volumen, um auf eine Zahl zu kommen, welche die Güte der Änderbarkeit des Programms anzeigt. Dabei steht ein MI über 85 für eine gute Änderbarkeit und ein MI unter 65 für eine schlechte.

Es wurden für die javabasierten Sprachen (JSF und GWT) der MI pro .java-Datei, also im Endeffekt pro Java-Klasse, berechnet. Die jsp-Dateien blieben unberücksichtigt, da es sich hierbei nicht um eine Programmiersprache im klassischem Sinn handelt. Am Ende wird für den gesamten MI der Durchschnitt der vorher berechneten MIs gebildet. Dabei wurde das fachliche Model seperat gemessen, da JSF und GWT das gleiche Model benutzen.

Maintainability Index

	JSF	**GWT**	**PHP**
MIwoc	63.35	65.67	87.89
MIcw	24.68	25.28	3.36
MI	88.03	90.95	91.25
fachlich	78.72	78.72	93.75
MI gesamt	84.16	84.84	92.5

Tabelle 6.1: Messung der Änderbarkeit - Maintainability Index

Wie aus der Tabelle entnommen werden kann weisen die javabasierten Sprachen eine ähnliche Komplexität und damit Änderbarkeit auf. Es muss noch gesagt werden, dass nicht der einfache Durchschnitt des MI berechnet wurde, sondern ein gewichteter MI, das bedeutet, für größere Klassen wurde der MI stärker in die Rechnung einbezogen. Dies hat den Grund, dass kleinere Klassen, die nur aus ein paar Codezeilen bestehen, den Index „verfälschen“ könnten, da sie nur einen Bruchteil des Programms ausmachen, sich dies jedoch nicht im MI

widerspiegeln würde. Wie der MI theoretisch berechnet wird, steht in Abschnitt 2.2.1. Konkret würde das für die Klasse BefundBean folgendermaßen aussehen:

Zuerst wird das Halstead-Volumen berechnet. Die Klasse BefundBean hat 14 unterschiedliche Operatoren (n1) und 63 unterschiedliche Operanden (n2). Operatoren sind Keywords (private, static, throws) und mathematische und logische Operatoren (+, -, /, &&, ||), Operanden sind Literale, Strings, primitive Typen und Identifier. Insgesamt werden 211 Operatoren (N1) und 164 Operanden (N2) gezählt. Um das Volumen (aveV) zu berechnen, gilt der Term:

$$aveV = (211 + 164) * \lg(14 + 63) \tag{6.1}$$
$$aveV = 2350.04 \tag{6.2}$$

Die zyklomatische Zahl wird durch die Anzahl der Schleifen und bedingten Anweisungen bestimmt. BefundBean hat eine bedingte Anweisung und somit eine zyklomatísche Zahl von 2. Die physikalische Anzahl der Zeilen ist 89 (locPhys). Der MI ohne Kommentare (MIwoc) wird demnach berechnet durch:

$$MIwoc = 171 - 5.2 * \ln(2350.04) - 0.23 * 2 - 16.2 * \ln(89) \tag{6.3}$$
$$MIwoc = 57.46 \tag{6.4}$$

Für den MI der Kommentare muss die prozentuale Kommentarzeilenanzahl bestimmt werden. Es gibt in der Klasse BefundBean 9 Kommentarzeilen, bei 89 Programmzeilen insgesamt macht das 0.101, also knapp 10%. Demnach ist der MI Kommentare (MIcw):

$$MIcw = 50 * \sin(\sqrt{2.4 * 0.101}) \tag{6.5}$$
$$MIcw = 23.65 \tag{6.6}$$

Für die Klasse BefundBean.java ist der MI: 57.46 + 23.65 = 81.11.

6.2.2 Effizienz

Gemessen wird das **Zeitverhalten** der Anwendungen beim Start und bei Anfragen an den Server oder bei einfachen Interaktion mit der Anwendung im idle-Betrieb und bei häufigen Zugriff von mehreren simulierten Personen.

Das Zeitverhalten wurde mit dem Tool JMeter gemessen. Es wurden 1000 Anfragen von einem Benutzer gestellt. Es wird die minimale, die maximale und die durschnittliche Antwortzeit angegeben: Um das Zeitverhalten der Anwendung bei Last zu messen wurden mehrere Benutzer simuliert, die gleichzeitig Anfragen an die Applikation senden. Die Benutzer wurden über parallel laufende Threads simuliert. Eine Auflistung zeigt die Tabelle 6.3.

	JSF	GWT	PHP
Wiederholungen	1000	1000	1000
Benutzer	1	1	1
min (ms)	3	1	1
max (ms)	295	11	64
avg (ms)	5	1	2
Fehler %	0.0	0.0	0.0

Tabelle 6.2: Zeitverhalten - ein Benutzer

	JSF				GWT				PHP			
Wdh.	50	50	50	50	50	50	50	50	50	50	50	50
Benutzer	10	50	100	200	10	50	100	200	10	50	100	200
min (ms)	3	2	2	1	1	1	1	2	2	2	2	2
max (ms)	384	587	1702	4290	5	142	225	767	416	4768	15654	52423
avg (ms)	32	54	156	302	1	39	73	138	5	40	141	501

Tabelle 6.3: Zeitverhalten - mehrere Benutzer

6.2.3 Lasttest

Mit einem Lasttest wird die Fehleranzahl bei laufendem Betrieb festgestellt. Dabei werden bei steigender Last die auftretenden Fehler gemessen. Daraus ergibt sich dann eine Fehlerquote, welche genauen Aufschluss darüber gibt, wie zuverlässig die einzelnen Applikationen und darüber hinaus die benutzten Frameworks sind.

Das Tool JMeter hilft bei steigender Last die Fehlerquote der Anwendung zu ermitteln. Es wurde neben der Fehlerquote auch der Datendurchsatz ermittelt. Es wurde die Zuverlässigkeit bei steigender Patientenzahl und Befundanzahl in der Datenbank gemessen. Es wurden immer 10 Wiederholungen von 10 Benutzern getestet.

6.2.4 Installierbarkeit

Es wird die unter Abschnitt 2.2.6 definierte Installierbarkeit gemessen. Dazu wird überprüft, wie lange es gedauert hat das Web-Framework zu installieren, das heißt, die Dauer bis die erste Ausgabe des eigenen Programms im Browser erscheint. Bei JSF wurde das Projekt durch zahlreiche Fehler merhmals neu aufgesetzt, bis schließlich ein blankes JSF-Projekt

	JSF			
Patienten	10	50	100	100
Befunde	5	10	10	50
kb/sek	1028.4	2651.5	3373.6	4079.5
Fehler %	0.60	2.0	3.0	3.0

Tabelle 6.4: Lasttest - JSF

	GWT			
Patienten	10	50	100	100
Befunde	5	10	10	50
kb/sek	252.0	256.0	260.5	261.0
Fehler %	0.0	0.0	0.0	0.0

Tabelle 6.5: Lasttest - GWT

benutzt wurde, welches alle wichtigen Einstellungen vorgenommen hatte, so dass die Anwendung implementiert werden konnte. Durch den GWT Application Wizard wurde das GWT-Projekt direkt mit den richtigen Einstellungen geladen und eine Beispielanwendung angelegt, welche auch auf Anhieb funktioniert hat. Bei der Implementierung der eigenen Applikation mussten noch ein paar Änderungen in den Einstellungen vorgenommen werden, wodurch die Dauer bis zur ersten Ausgabe verzögert wurde. Die Installation von PHP ging sehr schnell, da außer dem Server keine zusätzlichen Pakete oder Programme installiert werden mussten, auch die Implementierung der Anwendung hat nur wenige Stunden gedauert. Zusammengefasst lassen sich die folgenden Messungen vornehmen:

6.3 Auswertung

Um die Frameworks in bestimme Bereiche einordnen zu können, müssen die Ergebnisse ausgewertet und verglichen werden, um eine relative Einschätzung der Frameworks zu erreichen. Dabei werden die Frameworks in den verschiedenen Kriteriumskategorien verglichen, um sie am Ende grob einordnen zu können und zu sehen, für welche Bereiche die Frameworks besser geeignet sind und für welche nicht.

Bei der Änderbarkeit liegen GWT und JSF gleichauf, PHP hingegen weist ein wesentlich höheren Maintainability Index auf. Da GWT und JSF beide auf Java basieren, ist die Ähnlichkeit des MI nicht verwunderlich. Der höhere MI von PHP führt wahrscheinlich darauf zurück,

	PHP			
Patienten	10	50	100	100
Befunde	5	10	10	50
kb/sek	662	218.8	3656.0	6082.1
Fehler %	0.0	0.0	0.0	0.0

Tabelle 6.6: Lasttest - PHP

	JSF	GWT	PHP
Installation - Dauer (min)	300	210	60
Implementierung - Dauer (min)	360	510	270
Entwicklungsaufwand (min)	660	720	330

Tabelle 6.7: Übersicht zur Messung der Übertragbarkeit

dass PHP dynamisch typisiert ist, wodurch viele Operanden wegfallen und somit der Code generell übersichtlicher wird. Darüber hinaus sind immer nur sehr kleine „Codehäppchen“ im HTML eingebettet und da weniger meistens besser zu warten ist, hat PHP einen höheren MI.

Die Effizienz der Frameworks ist sehr unterschiedlich ausgefallen. PHP hat ein sehr schlechtes Zeitverhalten, vor allem bei Mehrbenutzerbetrieb. Bei 200 gleichzeitigen Benutzern stieg die maximale Ladezeit der Seite auf knapp über 50000 ms an, durchschnittlich wurde eine Seite 0.5 Sekunden geladen. JavaServer Faces hat ein durchschnittliches Zeitverhalten mit einer durschnittlichen Ladezeit von 300 Millisekunden im Mehrbenutzerbetrieb. Sehr schnell hingegen ist Google Web Toolkit mit einer durchschnittlichen Ladezeit von 138 ms. Zurückzuführen ist dies auf die Benutzung von Ajax, durch welches bei wiederholtem Seitenaufruf nicht die ganze Seite neu geladen wird und somit die Ladezeiten sehr niedrig bleiben.

Bei PHP war der Installationsaufwand am geringsten, da nur der Webserver aufgesetzt werden musste, wo alle nötigen Einstellungen schon vorgenommen waren. Mit JSF und GWT gab es Probleme beim Einrichten der Umgebung, wodurch sich die Installationsdauer deutlich in die Höhe geschraubt hat. Der Implementationsaufwand war bei allen Frameworks ungefähr ähnlich, wobei die Implementationsdauer bei GWT etwas höher war, dadurch dass auch der komplette HTML Komponentenbaun mit Java erstellt werden musste. Anstatt nur das nötige Tag niederzuschreiben mussten erst Objekte deklariert, Werte zugewiesen und schließlich das Objekt dem RootPanel hinzugefügt werden.

Nach der Analyse der einzelenen Webframeworks lassen sich folgende Schlüsse ziehen. Mit PHP lassen sich eher Anwendungen realisieren, wo schnell Ergebnisse gebraucht werden

und für Webseiten, die schnell aufgezogen werden müssen. Da der Installations- und Implementationsaufwand sehr gering ist, lässt sich schon in wenigen Stunden eine dynamische Webseite hochziehen. Jedoch ist PHP nicht für größer angelegte Projekte zu gebrauchen, wo viele Benutzer auf die Webressourcen zugreifen. Für diese Art von Webanwendung lässt sich GWT gut benutzen. Der Aufwand ist größer, jedoch haben die Anwendungen auch bei Mehrbenutzerbetrieb eine stets gute Performace. JSF bildet den Durschnitt der getesteten Frameworks und lässt sich sowohl bei größeren Projekten von Firmen, aber auch von Projekten von Privatleuten gut benutzen.

Desweiteren wurde festgestellt, dass GWT ein eher starres Programmiergerüst zur Verfügung stellt, welches dem Entwickler wenig Spielraum in der Entwicklung der Anwendung gibt, da wenig Einfluss in die tatsächliche Ausgabe des Programms genommen werden kann. PHP hingegen lässt sich flexibel einsetzen, da direkt am HTML-Code angesetzt werden kann und PHP viele Möglichkeiten gibt, die Anwendung nach den Wünschen des Programmieres zu gestalten. JSF bildet auch hier wieder den Durchschnitt.

Die getesteten Frameworks lassen sich nach der Analyse der Anwendungen folgendermaßen grob einordnen:

- **JSF:** Java-Framework, welches eine durschnittliche Abdeckung in allen getesteten Bereichen zeigt
- **GWT:** auf Java basierendes Framework, welches leistungsstark ist, jedoch eingeschränkte Entfaltungsmöglichkeiten bietet
- **PHP:** auf Webbereich spezialisierte Sprache, die flexibel einsetzbar ist, präzise benutzt werden kann, jedoch in dieser Testumgebung Leistungseinbußen hatte

Die vorgenommenen Tests stellen nur einen Ansatz zur Analyse und Bewertung von Frameworks dar. Da die Frameworks nur anhand eines Szenarios getestet und nur einige Kriterien zur Analyse herangezogen wurden, bieten die Ergebnisse keine fundierte Grundlage und es müssen noch weitere Tests herangezogen werden, um die Frameworks endgültig in bestimmte Bereiche enordnen zu können. Da der Autor dieser Arbeit mit manchen Frameworks schon Erfahrung hatte und mit anderen nicht, könnten die Ergebnisse bei anderen Testern verschieden ausfallen. Genauso könnten weitere Einstellungen gemacht oder eine andere Entwicklungsumgebung benutzt werden, die in der Analyse unterschiedliche Ergebnisse zur Folge hätte. Somit müsste für jede Technologie die bestmöglichste Einstellung erreicht werden, um fundierte Ergebnisse zu bekommen und letzten Endes eine treffende Einordnung der Frameworks zu erreichen.

7 Zusammenfassung

In dieser Arbeit wurden verschiedene Frameworks anhand von ausgewählten Kriterien analysiert. Es wurden verschiedene Kriterien vorgestellt und es wurde versucht Metriken zu entwickeln, um diese Kriterien messen zu können. Dabei wurde festgestellt, dass es für viele Kriterien, welche in der Softwareentwicklung benutzt werden, um die Qualität von Software zu beschreiben, sehr schwer ist, passende Metriken zu finden, so dass ein eindeutiges vergleichbares Ergebnis herauskommt. Die gewählten Kriterien waren Änderbarkeit, der Installations- und Implementierungsaufwand, die Effizienz der Anwendung und die Zuverlässigkeit bei Belastung, auch Lasttest genannt.

Um die Frameworks unter gleichen Bedingungen testen zu können wurde ein Szenario erstellt, welches mit den einzelnen Frameworks realisiert werden sollte. Da das Szenario dabei nur ein Mittel zum Zweck darstellt, wurde ein einfaches Szenario gewählt, welches schnell realisiert werden konnte, sich jedoch leicht in andere Bereiche übertragen lässt. Es wurde das Szenario „Physiotherapie“ entwickelt, welches dazu benutzt werden kann Patienten in das System einzutragen, Befunde für Patienten zu erstellen und den Behandlungsverlauf des Patienten zu dokumentieren, um so über die Zeit Erkenntnisse über die verwendeten Methoden zu erhalten.

Als Technologien wurden JSF, GWT unt PHP gewählt. JSF und GWT basieren auf Java, PHP basiert auf einer imperativen/objektorientierten Sprache, die speziell für den Webbereich entwickelt wurde. Da PHP eine weit verbreitete und bekannte Sprache ist, wurde die Sprache nur kurz angerissen, auf JSF und GWT wurde genauer eingegangen und anhand eines Beispiels das jeweilige Framework vorgestellt. Danach wurde das Szenario mit den einzelnen Frameworks implementiert und das Ergebnis vorgestellt. Dabei wurden die unterschiedlichen Architekturmuster der Frameworks aufgezeigt, so benutzt JSF das Model-View-Controller-Prinzip, GWT hingegen orientiert sich eher am Client-Server-Modell mit Remote-Procedure-Calls.

Zum Schluss wurden die Anwendungen in der Testumgebung analysiert und ausgewertet. In der Testumgebung konnte jede Implementation anhand der gewählten Kriterien untersucht werden. Die Ergebnisse wurden verglichen und die Frameworks konnten so bewertet und eingeordnet werden. Es kam heraus, dass GWT ein sehr effizientes Framework ist, welches jedoch einen eher starren Entwicklungsablauf vorgibt und somit für Bereiche geeignet, die auf geringe Antwortzeiten setzen. PHP hat in dem Test leistungsschwach abgeschnitten,

hatte jedoch einen sehr hohen Änderbarkeitswert und einen geringen Installationsaufwand und ist somit für das schnelle und einfache Erstellen von Webanwendungen geeignet. JSF stellte den Durchschnitt der getesteten Frameworks dar und kann somit für den einen und den anderen Bereich genutzt werden.

7.1 Ausblick

Diese Arbeit hat die Problematik, die bei der Analyse von Webframeworks auftreten kann, angerissen und einen Ansatz gegeben, wie eine Testumgebung aussehen kann, um Frameworks zu analysieren. Dabei wurde nur ein grober Überblick über die Möglichkeiten gegeben. So müssen neue Metriken entwickelt werden, um Kriterien zu messen, mit denen am Ende die Frameworks getestet werden können. Qualitätskriterien sind schon reichlich aus dem Bereich der Softwarequalität bekannt und die Kriterien dieser Arbeit wurden diesem Bereich entliehen. Nur müssen eindeutige, vergleichbare Ergebnisse erzeugt werden.

Genauso müssen weitere Szenarien erstellt werden, um so eine größere Bandbreite an Anforderungen abdecken zu können. In dieser Arbeit wurde ein simples Szenario genommen, in weiteren Tests müssen verschiedene Szenarien zum Einsatz kommen, die es ermöglichen die Frameworks unter anderen Gesichtspunkten zu testen, z.B. komplexere Anwendungen, wie Online-Shops, Bilderarchive oder ähnliches. So kann am Ende auch berücksichtigt werden, ob ein Framework bestimmte Anforderungen, die durch ein Szenario gestellt wurden, gut abdecken kann. Und natürlich muss die Breite der getesteten Technologien vergrößert werden, so dass möglichst alle wichtigen Sprachen vertreten sind. Auf diesem Wege wird die Testumgebung stetig erweitert, so dass die Ergebnisse besser eingeordnet werden können und somit die Frameworks konkretere Anforderungsbereiche abdecken.

Am Ende können diese Ergebnisse eine Hilfestellung darstellen, um für eigene Projekte das passende Framework zu finden.

Literaturverzeichnis

[Apache Software Foundation 2010] APACHE SOFTWARE FOUNDATION: *Apache JMeter*. 2010. – URL http://jakarta.apache.org/jmeter/. – [Online; Stand 14. Juli 2010]

[Fenton 1999] FENTON, Norman: *Measurement Frameworks and Standards*. 1999. – URL http://www.eecs.qmul.ac.uk/~norman/papers/qa_metrics_article/section_7_standards.html. – [Online; Stand 28. Juli 1999]

[Google Inc. 2010a] GOOGLE INC.: *Communicating with a Server*. 2010. – URL http://code.google.com/intl/de-DE/webtoolkit/doc/1.6/DevGuideServerCommunication.html

[Google Inc. 2010b] GOOGLE INC.: *Communicating with a Server*. 2010. – URL http://code.google.com/intl/de-DE/webtoolkit/doc/1.6/FAQ_Server.html

[Google Inc. 2010c] GOOGLE INC.: *Hyperlink (Google Web Toolkit Javadoc)*. 2010. – URL http://google-web-toolkit.googlecode.com/svn/javadoc/2.0/com/google/gwt/user/client/ui/Hyperlink.html. – [Online; Stand 03. Februar 2010]

[Horn 2007] HORN, Torsten: *Architektur für Webanwendungen*. 2007. – URL http://www.torsten-horn.de/techdocs/webanwendungen.htm

[irian 2010] IRIAN: *JavaServer Faces*. 2010. – URL http://jsfatwork.irian.at/semistatic/introduction.html. – [Online; Stand 29. Juni 2010]

[Izabel 12/2007a] IZABEL: *GWT Tutorial - Building a GWT RPC Service*. 12/2007. – URL http://developerlife.com/tutorials/?p=125

[Izabel 12/2007b] IZABEL: *GWT Tutorial - Introduction to GWT*. 12/2007. – URL http://developerlife.com/tutorials/?p=80

[Ludewig und Lichter 2010] LUDEWIG, Jochen ; LICHTER, Horst: *Software Engineering 2.Aufl.* dpunkt.verlag, 2010

[Marinschek u. a. 2010] MARINSCHEK, Martin ; KURZ, Michael ; MÜLLAN, Gerald: *JavaServer Faces 2.0*. dpunkt.verlag, 2010

[Müller 2006] MÜLLER, Bernd: *JavaServer Faces*. Hanser, 2006

[Nazmul 01/2008] NAZMUL: *GWT Tutorial - Managing History and Hyperlinks*. 01/2008. – URL http://developerlife.com/tutorials/?p=232

[Sommers 12/2006] SOMMERS, Frank: *Compiling Java to JavaScript*. 12/2006. – URL http://www.artima.com/lejava/articles/java_to_javascript.html

[Steyer 01/2007] STEYER, Ralph: *Das Google Web Toolkit*. 01/2007. – URL http://www.contentmanager.de/magazin/artikel_1292_google_web_toolkit_gwt_ajax_java.html

[Steyer 2007] STEYER, Ralph: *Google Web Toolkit*. entwickler.press, 2007

[SwtWiki 2008a] SWTWIKI: *Effizienz*. 2008. – URL http://www.imn.htwk-leipzig.de/~weicker/pmwiki/pmwiki.php/Main/Effizienz

[SwtWiki 2008b] SWTWIKI: *Effizienz*. 2008. – URL http://www.imn.htwk-leipzig.de/~weicker/pmwiki/pmwiki.php/Main/%dcbertragbarkeit

[SwtWiki 2008c] SWTWIKI: *Änderbarkeit*. 2008. – URL http://www.imn.htwk-leipzig.de/~weicker/pmwiki/pmwiki.php/Main/%C4nderbarkeit

[Verifysoft Technology 2010] VERIFYSOFT TECHNOLOGY: *Verifysoft - Maintainability Index*. 2010. – URL http://www.verifysoft.com/de_maintainability.html. – [Online; Stand 5. Juni 2010]

[Wikipedia 2009] WIKIPEDIA: *Clientseitige Anwendung — Wikipedia, Die freie Enzyklopädie*. 2009. – URL http://de.wikipedia.org/w/index.php?title=Clientseitige_Anwendung&oldid=62360662. – [Online; Stand 6. August 2010]

[Wikipedia 2010a] WIKIPEDIA: *ASP.NET — Wikipedia, Die freie Enzyklopädie*. 2010. – URL http://de.wikipedia.org/w/index.php?title=ASP.NET&oldid=77452236. – [Online; Stand 08. August 2010]

[Wikipedia 2010b] WIKIPEDIA: *Effizienz (Informatik) — Wikipedia, Die freie Enzyklopädie*. 2010. – URL http://de.wikipedia.org/w/index.php?title=Effizienz_(Informatik)&oldid=75262713. – [Online; Stand 07. Juli 2010]

[Wikipedia 2010c] WIKIPEDIA: *Google Web Toolkit — Wikipedia, Die freie Enzyklopädie*. 2010. – URL http://de.wikipedia.org/w/index.php?title=Google_Web_Toolkit&oldid=76120252. – [Online; Stand 1. August 2010]

[Wikipedia 2010d] WIKIPEDIA: *ISO/IEC 9126 — Wikipedia, Die freie Enzyklopädie*. 2010. – URL http://de.wikipedia.org/w/index.php?title=ISO/IEC_9126&oldid=77673268. – [Online; Stand 11. August 2010]

[Wikipedia 2010e] WIKIPEDIA: *PHP — Wikipedia, Die freie Enzyklopädie*. 2010. – URL http://de.wikipedia.org/w/index.php?title=PHP&oldid=78022985. – [Online; Stand 22. August 2010]

[Wikipedia 2010f] WIKIPEDIA: *Ruby on Rails — Wikipedia, Die freie Enzyklopädie*. 2010. – URL http://de.wikipedia.org/w/index.php?title=Ruby_on_Rails&oldid=76802723. – [Online; Stand 08. August 2010]

[Wikipedia 2010g] WIKIPEDIA: *Servlet — Wikipedia, Die freie Enzyklopädie*. 2010. – URL http://de.wikipedia.org/w/index.php?title=Servlet&oldid=78025571. – [Online; Stand 25. August 2010]

[Wikipedia 2010h] WIKIPEDIA: *Testbarkeit — Wikipedia, Die freie Enzyklopädie*. 2010. – URL http://de.wikipedia.org/w/index.php?title=Testbarkeit&oldid=73485202. – [Online; Stand 05. Juli 2010]

Versicherung über Selbstständigkeit

Hiermit versichere ich, dass ich die vorliegende Arbeit im Sinne der Prüfungsordnung nach §24(5) ohne fremde Hilfe selbstständig verfasst und nur die angegebenen Hilfsmittel benutzt habe.

Hamburg, 26. August 2010	
Ort, Datum	Unterschrift