IHK-Abschlussprüfung Sommer 2018

Dokumentation der betrieblichen Projektarbeit
Für die Abschlussprüfung zum Fachinformatiker - Anwendungsentwicklung

Network Monitoring Tool

Eine Desktop-Applikation
zur Überwachung der Verfügbarkeit von Servern

Prüfungsbewerber
Christoph Kiank
Lämmersieth 54
22305 Hamburg

Prüflingsnummer
131 54036

Abgabetermin: Hamburg, den 13.05.2018

Praktikumsbetrieb: BITMARCK Technik GmbH
Hammerbrookstraße 38
20097 Hamburg

Ausbildungsstätte: CBW College Berufliche
Weiterbildung GmbH
Frankenstraße 3
20097 Hamburg

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1 Einleitung

Im Rahmen einer Umschulung zum Fachinformatiker für Anwendungsentwicklung bei der CBW (College Berufliche Weiterbildung GmbH), absolvierte der Autor ein Praktikum bei der BITMARCK Technik GmbH (BMT) in Hamburg und fertigte dort diese Dokumentation über die betriebliche Projektarbeit an.

Die Dokumentation ist mit Microsoft Word 2016 erstellt worden. Für die Erstellung der Präsentation wird Apples Keynote in der Version 8.0 verwendet.

Die Daten und Informationen in diesem Dokument sind Eigentum der BMT, eine Weitergabe an Dritte ist nicht gestattet.

1.1 Projektumfeld

Die BITMARCK Technik GmbH ist eine der Tochtergesellschaften der BITMARCK Unternehmensgruppe und spaltete sich aus dem ehemaligen IT-Bereich der DAK-Gesundheit (Deutsche Angestellten Krankenkasse) ab. Die BITMARCK ist ein „[…] Full-Service-Dienstleister im IT-Markt der gesetzlichen Krankenversicherung und realisiert IT-Lösungen für die Betriebs- und Innungskrankenkassen sowie für die DAK-Gesundheit und weitere Ersatzkassen – 30.000 Mitarbeiter und 20 Millionen Versicherte in der GKV profitieren von den IT-Dienstleistungen der BITMARCK, 85 Prozent der Krankenkassen sind Kunden der Unternehmensgruppe.“[1]

1.2 Projektbegründung

Ein Produkt der BITMARCK ist die bitGo_Suite. Sie ist eine Zusatzsoftware mit Anbindung an das Kernsystem BITMARCK_21c|ng und besteht aus drei Komponenten:

Das jüngste Mitglied der bitGo_Suite, ist die bitGo_App. Sie bietet ein mobile Lösungen für Versicherte und Krankenkassen an.

Mit der Softwarelösung bitGo_KV (KV=Krankenversicherung) haben Krankenkassen die Möglichkeit, mit ihren Versicherten, die nicht in der Onlinekasse registriert sind, Schriftverkehr digital zu führen. Über einen personenbezogenen Zugang mit einem Einmalpasswort, haben Versicherte die Möglichkeit über einen Webbrowser Formulare, Anträge oder auch Umfragen auszufüllen und an die Krankenkassen zurück zu senden.

Die Software bitGo_GS (GS=Online-Geschäftsstelle) gilt als direkter Verbindungspunkt zwischen Krankenkassen und den registrierten Versicherten. Krankenkassen präsentieren hierüber ihr Unternehmen, in Form einer Internetpräsenz. Versicherte haben über einem Webbrowser auf eben dieser Webpräsenz die Möglichkeit, beispielsweise ihre Adressdaten zu ändern oder eine Arbeitsunfähigkeitsbescheinigung hochzuladen.

Die Komponenten von bitGo_GS liegen auf unterschiedlichen Servern. Auf dem Auslieferungsserver, liegt die Version der Software, mit der Versicherte direkt in Kontakt kommen. Die Redakteure der Krankenkassen wiederum stellen den Inhalt ihrer Webpräsenz mit dem Content-Management-System FirstSpirit, auf einem separaten Server, zusammen. Dieser ist nicht

[1] Vgl. https://www.bitmarck.de

direkt mit dem Internet verbunden. Jede Änderung der Produktivsoftware wird zuerst in FirstSpirit umgesetzt. Anschließend wird diese Änderung in Form einer .war-Datei zusammengefasst und an den Auslieferungsserver gesendet. Dort wird die Datei wieder entpackt und die Änderungen werden veröffentlicht.

Bei der BMT werden die Vorlagen, welche die Krankenkassen in FirstSpirit für das Zusammenstellen ihrer Inhalte benötigen, entwickelt. Um die Funktionsweisen der Vorlagen testen zu können, ist ein Testsystem im BMT eingerichtet. Der Aufbau der Server für dieses System entspricht dem Aufbau der Server bei den Krankenkassen. Zusätzlich sind weitere Server für Entwicklungszwecke eingerichtet.

Tritt während der Entwicklung ein Fehler auf Grund eines ausgefallenen Servers auf, lässt dieser sich, wegen einer ungenauen Fehlerausgabe, nicht exakt identifizieren. Mühsam wird jeder Server auf seine Verfügbarkeit hin überprüft. Ist die fehlende Verbindung gefunden, wird sie über die Befehlseingabe in PuTTY[2] wiederhergestellt.

1.3 Projektziel

Aufgabe ist es, eine Desktop-Applikation in der Programmiersprache Java zu entwickeln. In dieser sollen die Server und deren Verfügbarkeit tabellarisch aufgelistet sein. Bei einem Ausfall eines Servers soll der Benutzer durch die Anwendung gewarnt werden. Die Verfügbarkeit von einem oder gleich aller Server soll der Benutzer während der Laufzeit, entweder manuell oder automatisch in bestimmten Zyklen, durch wiederholtes Senden von Testdaten an einen Server ermitteln können. Das Ergebnis soll auf einer grafischen Oberfläche ausgegeben werden. Für die Anfragen notwendige Server- und Porteinträge sollen aus einer externen Quelle gelesen werden. Sie sollen von der Anwendung aus bearbeitet, gelöscht oder neu erstellt werden können.

1.4 Zielgruppe

Zielgruppe der Anwendung sind die Mitarbeiter des bitGo_GS-Teams innerhalb der BMT.

[2] Es ist hierüber möglich Befehle, die auf einem fernen System abgesetzt werden sollen, einzugeben.

2 Projektplanung

2.1 Projektphasen

Für die Umsetzung des Projektes standen dem Autor 70 Stunden zur Verfügung. Diese wurden vor Projektbeginn grob in sechs Phasen aufgeteilt und im Laufe der Entwicklung auf Grund der agilen Vorgehensweise umstrukturiert. Eine genauere Zeitplanung der einzelnen Phasen lassen sich der Tabelle 3 im Abschnitt 7.1 entnehmen.

Phase	Geplante Zeit
Analysephase	2 h
Entwurfsphase	11 h
Implementierungsphase	36 h
Test und Korrektur	6 h
Projektübergabe	3 h
Dokumentation	12 h
Gesamt	**70 h**

Tabelle 1: Zeitplanung der Projektphasen

2.2 Ressourcenplanung

Die für das Projekt verwendeten Ressourcen werden im Anhang A.1 aufgelistet. Damit sind sowohl Hard- und Softwareressourcen als auch das Personal gemeint. Bei der Auswahl der verwendeten Software wurde darauf geachtet, dass diese kostenfrei (z.B. als Open Source) zur Verfügung steht oder die BMT bereits Lizenzen für diese besitzt. Dadurch sollen anfallende Projektkosten möglichst gering gehalten werden.

2.3 Entwicklungsprozess

Bei der BMT wird die Software nach dem agilen Vorgehensmodell Scrum entwickelt. Für das Abschlussprojekt musste sich der Autor demnach nicht selbst für ein passendes Vorgehensmodell entscheiden.

Scrum verfolgt den Ansatz empirisch, inkrementell und iterativ zu sein. Das bedeutet im Groben, dass das langfristige Ziel (das Product Backlog) für die Fertigstellung des Projektes kontinuierlich verfeinert und verbessert wird. Genaue Anforderungen werden in sogenannte Sprints unterteilt. Da das Projekt einen relativ kurzen Zeitraum umfasst, wurden kleinere Sprints im Abstand einer Woche geplant. Das Projekt wurde somit auf zwei Wochen aufgeteilt.

In der ersten Woche wurden die Analysephase, die Entwurfsphase und ein Teil der Implementierungsphase durchlaufen. In der Analysephase wurde der Ist-Zustand, der Soll-Zustand und die Wirtschaftlichkeit ermittelt. Während der Entwurfsphase wurde ein Mockup der Anwendung

erstellt und der Anwendungsablauf in einem Aktivitätsdiagramm (Siehe Anhang A.6) abgebildet. Im ersten Teil der Implementierungsphase wurden die Model-Klassen angelegt.

In der zweiten Woche wurde die grafische Oberfläche (View) in JavaFX mit Hilfe des Scene Builders umgesetzt. Anschließend wurde der Funktionsumfang der Anwendung mit den Controller-Klassen implementiert. Diese konnten danach in einem Klassendiagramm abgebildet (Siehe ab Anhang A.7) werden. Alle Klassen wurden während der Implementierungsphase nach dem MVC-Softwarearchitekturmodell erstellt. Ihre Attribute und Methoden wurden mittels der auf HTML basierenden Dokumentationssoftware Javadoc festgehalten. Das Anlegen eines Entwicklerhandbuches wurde dadurch überflüssig.

Da es sich um eine testgetriebene Entwicklung handelt, wurden im Vorfeld für Klassen und Methoden JUnit-Tests geschrieben. Dadurch konnte sichergestellt werden, dass bei einer Veränderung des Quellcodes das Programmverhalten nicht unbeabsichtigt geändert wird. Schlussendlich wurde die Anwendung dem Fachbereich übergeben und die Dokumentation über das Projekt verfasst.

3 Analysephase

3.1 Ist-Zustand

Alle Komponenten von bitGo_GS werden auf unterschiedlichen Servern entwickelt und getestet, um so die Auslastung eines einzelnen Servers zu reduzieren. Dennoch kann es vorkommen, dass ein Server ausfällt, so dass ein Weiterarbeiten unmöglich ist. Den ausgefallenen Server zu identifizieren, um ihn anschließend Neustarten zu können ist mühsam und kostet wertvolle Arbeitszeit.

3.2 Soll-Zustand

Die Anwendung wird dahingehend konzipiert, dass jeder Entwickler die Möglichkeit hat, sich eine Übersicht seiner Server anzulegen, um über einen Ausfall eines jeden Servers informiert zu werden. Alle Einstellungen, wie Server-Port-Verbindungen oder Anwendungseinstellungen, beispielsweise, ob eine Intervallabfrage stattfinden soll, werden in einer externen Datei im JSON-Format abgespeichert. Diese Datei ist für das Ausführen der Anwendung unabdingbar.

3.3 Wirtschaftlichkeitsanalyse

Aufgrund der im Abschnitt 3.1 und 3.2 genannten Probleme, müssen folgende Fragen geklärt werden. Existiert auf dem Markt bereits ein Produkt, welches die spezifischen Anforderungen erfüllt und zusätzlich kosteneffizient ist oder ist es sinnvoller die Anforderungen, in Form einer eigenen entwickelten Anwendung im Betrieb umzusetzen? Wie hoch ist die Zeitersparnis jedes Entwicklers, die erforderlich ist, um ausgefallene Server zu identifizieren und wieder neu zu starten? In den folgenden Abschnitten wird die Wirtschaftlichkeit der eben genannten Fälle geklärt.

3.3.1 „Make or Buy“-Entscheidung

Da das Projekt unternehmensspezifischen Anforderungen aufweist, eine stetige Funktionserweiterung, auch nach Beendigung des Projektes, erwartet wird und auf dem Markt keine relevante Software zur Verfügung steht, ist es sinnvoll eine Eigenentwicklung durchzuführen.

3.3.2 Projektkosten

Da es sich bei der Entwicklung um ein firmeninternes Projekt handelt, ist die BMT sowohl Kunde als auch Auftraggeber.

Die Kosten für die Entwicklung, werden in Personentagen(PT)[3] angegeben.

Der interne Stundensatz der BMT errechnet sich laut der Abteilung Controlling wie folgt:

$$\frac{520\ €/PT}{7{,}8\ Stunden/PT} \approx 66{,}67\ €/Stunde$$

Die investierte Zeit des Entwicklers für das Projekt, betrug 70 Stunden. Hinzu kommen die Kosten von Mitarbeitern die während eines Fachgespräches in der Analysephase und Entwurfsphase, für Code-Review in der Implementierungsphase und zum Schluss bei der Endabnahme der finalen Version der Anwendung zu Verfügung standen.

Die gesamten Projektkosten ergeben sich aus der Summe der einzelnen Vorgänge, welche der Tabelle 2 zu entnehmen sind.

Vorgang	Mitarbeiter	Zeit	Personal[4]
Entwicklungskosten	1x Umschüler	70 h	4666,90 €
Fachgespräch	1x Mitarbeiter	2 x 0,5 h	66,67 €
Code-Review	1x Mitarbeiter	2 x 1 h	133,34 €
Abnahme	1x Mitarbeiter	0,5 h	33,34 €
Projektkosten gesamt			**4900,25 €**

Tabelle 2: Übersicht der Projektkosten

Da für das Projekt auf bestehende Hard- und Software zurückgegriffen werden konnte, entstanden keine weiteren Kosten.

[3] Kosten für einen Mitarbeiter pro Tag. Ein Personentag wird mit einem Aufwand von 7,8 Stunden gezählt.
[4] Personalkosten pro Vorgang = Anzahl der Mitarbeiter * Zeit * Stundensatz

3.3.3 Amortisationsdauer

Im folgenden Abschnitt soll ermittelt werden, ab welchem Zeitpunkt sich die Entwicklung der Anwendung amortisiert hat. Anhand dieses Wertes kann dann beurteilt werden, ob die Umsetzung des Projektes aus wirtschaftlicher Sicht sinnvoll ist und sich auf Dauer Kostenvorteile ergeben. Die zeitliche Ersparnis ergibt sich aus der Zeit, die Entwickler nicht mehr aufwenden müssen, um einen ausgefallenen Server zu identifizieren, um eben diesen dann neu starten zu können. Eine Umfrage im Team ergab, dass ein Server durchschnittlich einmal alle zwei Tage ausfällt. Die Anzahl der Arbeitstage im Jahr 2017 betrugen 251. Aus den Angaben ergibt sich folgende Berechnung:

$$0{,}5\,\frac{\frac{Ausfälle}{Server}}{Arbeitstag} * 251\,\frac{Arbeitstag}{Jahr} = 125{,}5\,\frac{\frac{Ausfälle}{Server}}{Jahr}$$

Aktuell wird auf drei Servern entwickelt.

$$125{,}5\,\frac{\frac{Ausfälle}{Server}}{Jahr} * 3\ Server = 376{,}5\,\frac{Ausfälle}{Jahr}$$

Vom ungefähren Erkennen eines Ausfalls bis zum Neustarten eines Servers beträgt die verlorene Arbeitszeit ca. 5 Minuten. Die gesamte verlorene Arbeitszeit pro Jahr errechnet sich wie folgt:

$$376{,}5\,\frac{Ausfälle}{Jahr} * 5\,\frac{Minuten}{Ausfälle} = 1.882{,}5\ \frac{Minuten}{Jahr} = 31{,}375\,\frac{Stunden}{Jahr}$$

Der jährliche Verlust errechnet sich aus der verlorenen Arbeitszeit pro Jahr mit dem internen Stundensatz der BMT.

$$31{,}375\,\frac{Stunden}{Jahr} * 66{,}67\,\frac{€}{Stunde} = 2.091{,}77\,\frac{€}{Jahr}$$

Die Amortisationsdauer wird berechnet, indem man die Anschaffungskosten durch die laufende Kostenersparnis dividiert, die durch das neue Produkt erzielt wird.

$$\frac{4900{,}25\ €}{2.091{,}77\ €/Jahr} = 2{,}34\ Jahre\ \approx 2\ Jahre\ 4\ Monate$$

Anhand der Amortisationsrechnung ergibt sich für das Projekt eine Amortisationsdauer von 2 Jahren und 4 Monaten. Dies ist der Zeitraum über den die neue Anwendung mindestens eingesetzt werden muss, damit sich Anschaffungskosten und Kosteneinsparung ausgleichen. Da das Unternehmen die neue Anwendung langfristig einsetzen möchte, kann das Projekt auch unter wirtschaftlichen Gesichtspunkten als sinnvoll eingestuft werden.

4 Entwurfsphase

Um die neue Anwendung möglichst benutzerfreundlich bedienen zu können, soll eine intuitiv gestaltete grafische Oberfläche entwickelt werden. Mit Hilfe von Mockups wurden hierfür zunächst Prototypen der Oberfläche angefertigt und später dann ausgewählt. Dafür wurde der Aufbau der Anwendung mit einer 30-tägigen Testversion der Online Software Balsamiq[5] erstellt. Damit die Benutzeroberfläche am Ende den Anforderungen und Vorstellungen des Fachbereichs entspricht, wurde dieser bei der Entwurfsphase mit einbezogen. Für den Aufbau einer geeigneten Datenhaltung wurde Anhand einer Nutzwertanalyse (Anhang A.2) das Datenformat JSON ermittelt.

4.1 Datenstruktur

Die Datenstruktur beinhaltet für das Ausführen der Anwendung alle nötigen Informationen, wie beispielsweise Server- und Porteinträge, angelegte Verbindungsabfragen und gemachte Einstellungen. Der Grundaufbau dieser Datei befindet sich im Anhang A.3.

4.1.1 Die GSON-Bibliothek

Die Wahl einer geeigneten Bibliothek, welche zum Serialisieren und Deserialisieren von Java-Objekten zu JSON nötig ist, fiel auf die von Google entwickelte Open-Source-Bibliothek GSON, da sich diese im Unternehmen bereits etabliert hat.

4.2 Benutzeroberfläche

Im Folgenden werden die drei Hauptbereiche der Anwendung in ihrem Aufbau und ihrer Funktion kurz erläutert.

In dem oberen Bereich kommt eine Tool-Leiste. Diese beinhaltet eine Intervallabfrage als Slider und ein Ping-Button. Der Button ist für das Anstoßen der Verbindungsabfrage aller in der Tabelle (Anhang A.4 „Bereich Mitte“) eingetragenen Server verantwortlich. Weitere Befehle, wie zum Beispiel das Speichern der aktuellen Session oder das Laden einer neuen Session, werden in der oberen Menüleiste verfügbar sein.

Der Mittelteil bzw. der Hauptteil der Anwendung ist für die Verbindungsübersicht der Server reserviert. Die Übersicht wird tabellarisch dargestellt. Von links nach rechts kommen Server und -Porteinträge, ihr Erstellungsdatum mit Uhrzeit und eine erfolgte Verbindungsanfrage, ebenfalls mit aktuellem Datum und Uhrzeit.

Im unteren Bereich sind die Eingabemasken platziert. Sie sind für das Zusammenstellen von Serververbindungen in der Tabelle und zusätzlich für das Anlegen, Löschen oder Bearbeiten von Server- und Porteinträge aus der JSON-Datei erforderlich. Port- und Servereinträge sollen über eine Dropdown-Liste ausgewählt werden können.

[5] Vgl. https://balsamiq.cloud

Alle Informationen, die für das Ausführen der Anwendung nötig sind, werden in einer separaten Datei gespeichert (Anhang A.3). Diese beinhaltet alle Server- und Porteinträge, die zu der Tabelle (Anhang A.4 Bereich Mitte) hinzugefügten bzw. zu überwachenden Verbindungen und die Auswahl der Intervallabfrage. Am unteren Rand der Anwendung kommen eine Fortschrittsanzeige und eine Textausgabe, welche weitere Informationen einiger Befehle bereitstellen. Ein Mockup der Anwendung befindet sich im Anhang A.4.

4.2.1 Die JavaFX-Bibliothek

Für die Erstellung der grafischen Oberfläche besticht JavaFX durch moderne Design Elemente und gleichzeitig einer genaueren Trennung zwischen der View und dem Controller. Die komplette View wird in der Auszeichnungssprache XML erstellt und in einer einzigen Datei gespeichert. Diese liegt anschließend im FXML-Format vor und kann in dem Programm Scene Builder, grafisch durch das Zusammenstellen der einzelnen Elemente, bearbeitet werden. Ein Auszug befindet sich im Anhang A.5.

4.3 Anwendungsablauf

Direkt nach dem Starten der Anwendung, wird der Benutzer gefragt, ob er eine vorhandene Session laden oder eine neue anlegen möchte. Solange er keine Auswahl trifft, sind alle Elemente der Anwendung ausgegraut, d.h. deaktiviert.

Entscheidet er sich beispielsweise eine neue Session anzulegen, öffnet sich ein Datei-Manager mit der Aufforderung einen Speicherort auszuwählen und einen Dateinamen zu vergeben. Erst dann stehen alle Elemente der Anwendung zur Verfügung. Ausnahmen stellen die Elemente dar, welche an weitere Bedingungen geknüpft sind. Zum Beispiel bleibt der „Verbindung hinzufügen“-Button solange deaktiviert, wie kein Server und ein dazugehöriger Port aus dem Dropdown-Menü ausgewählt wurde.

Um der Tabelle (Anhang A.4 „Bereich Mitte“) Verbindungen, welche überwacht werden sollen, hinzufügen zu können, müssen im Dropdown-Menü mindestens ein Server und ein Port verfügbar sein. Sind keine Einträge verfügbar, kann der Benutzer in der Eingabemaske (Anhang A.4 „Bereich Unten“) Server- und Porteinträge anlegen und abspeichern. Diese Informationen werden dann in der JSON-Datei abspeichert. Danach sind sie im Dropdown-Menü verfügbar und können ausgewählt werden.

Sind jeweils ein Server und ein Port aus dem Dropdown-Menü markiert, wird der „Verbindung hinzufügen“-Button verfügbar. Nach betätigen des Buttons wird die Auswahl mit Erstellungsdatum und -uhrzeit der Tabelle hinzugefügt. Gleichzeitig wird einmalig eine Verbindungsanfrage gestartet und das Ergebnis, ebenfalls mit einem Zeitstempel versehen, der Zeile hinzugefügt.

Der Benutzer kann beliebig viele Kombinationen genau einmal anlegen, so, dass keine doppelten Einträge in der Tabelle möglich sind. Auch doppelte Server- oder Porteinträge können weder angelegt noch gespeichert werden. Sind alle Verbindung die überwacht werden sollen, in der Liste abgespeichert, kann entschieden werden ob, diese innerhalb eines bestimmen Intervall getestet werden sollen. Abschließend können alle in der Session gemachten Einstellungen abgespeichert werden.

Nach einem Neustart der Anwendung kann, durch die Auswahl „Session laden“ und dem anschließenden Auswählen der entsprechenden JSON-Datei im Datei-Manager, die gespeicherten Einstellungen wieder geladen werden. Eine Übersicht des Ablaufes befindet sich in Form eines Aktivitätsdiagramm im Anhang A.6.

5 Implementierungsphase

Das Abschlussprojekt soll als eigenständige Desktopapplikation umgesetzt werden. Der Quellcode wird mit Hilfe des kostenlosen Git-Clients Sourcetree verwaltet. Da der überwiegende Teil der Mitarbeiter im Unternehmen mit der Programmiersprache Java arbeitet, musste keine Auswahl anderer Sprachen in Betracht gezogen werden. Um Kosten für das Projekt zu sparen, fiel die Auswahl einer geeigneten Entwicklungsumgebung auf die Open-Source-Software Eclipse in der Version 4.7.3a (Oxygen 3A - April). Eclipse zeichnet sich durch eine sehr gute Erweiterbarkeit mittels Java-Bibliotheken in Form von .jar-Dateien aus. Im Anhang A.1 sind die eingesetzten Bibliotheken aufgelistet.

5.1 Das MVC-Architekturmodell

Bei der BMT erwies es sich als vorteilhaft, das Projekt auf Basis des MVC-Architekturmodells umzusetzen. Jede Komponente einer Software kann einem der drei Bestandteile – Model, View oder Controller – zugeordnet werden und ist damit weitestgehend unabhängig in seinem Aufgabenbereich. Durch die lose Verknüpfung der einzelnen Module erhöht sich die Wiederverwendbarkeit und Austauschbarkeit. Beispielsweise wäre es möglich das Erscheinungsbild der Anwendung auszutauschen, ohne eine Anpassung der Model-Klassen durchführen zu müssen. Zusätzlich können einzelne Komponenten durch eine striktere Trennung einfacher getestet, gewartet und flexibel erweitert werden. Aufgrund der genannten Vorteile soll das Architekturmuster für die Implementierung der Klassen verwendet werden.

5.1.1 Implementierung der Datenstrukturen (Model)

Das Model setzt sich aus den Daten und der entsprechenden Verarbeitungslogik zusammen. Anhand der Informationen, die in der JSON-Datei gespeichert werden sollen, wurden folgende Klassen realisiert.

Zuerst wurde jeweils die Server- und Portklasse erstellt. Für beide sollte sichergestellt werden, dass sich nur Objekte mit validem Inhalt erzeugen lassen. Daher wurde eine Validierungsklasse angelegt. Dies wird durch den Einsatz regulärer Ausdrücke erreicht. Sie stellen sicher, dass sich nur gültige IP-Adressen und fünfstellige Ports mit ganzen Zahlen erzeugen lassen. (Siehe Anhang A.7).

Es wurden dann die Klassen geschrieben, die für den Tabelleninhalt verantwortlich sind. Eine Zeile der Tabelle wird mit der `ServerPortTableContent` erzeugt. Diese steht in einer Ganzes-Teile-Beziehung zu zwei weiteren Klassen.

Eine, die zum Erstellen der zu überwachenden Verbindung (`ServerPortConnection`) verantwortlich ist und eine (`ServerPortConnectionQuery`), die die eben erstellte Verbindung auf ihre Konnektivität testet.Ebenfalls stehen die letztgenannten Klassen auch in einer Ganzes-Teile-Beziehung zur `DateStamp`-Klasse. Diese erzeugt wiederum einen Zeitstempel für die anderen Klassen.

Für das Speichern, Löschen oder Bearbeiten wurden die Klassen `JSONFileInitialisator`, `JSONContentHandler` und `JSONContenInAList` erstellt. Sie stehen in einer Vererbungshierarchie untereinander.Die Elternklasse `JSONFileInitialisator` ist für das Lesen und Schreiben, mittels eines `Input-/OutputStreams` und `BufferedReader-/Writer` verantwortlich. Weiter werden hier die Inhalte anhand der Struktur der JSON-Datei zur weiteren Verarbeitung in Listen zusammengestellt.Die erste Kindklasse, der `JSONContentHandler`, hält alle Methoden zum eigentlichen Schreiben, Löschen und Bearbeiten aller Informationen aus der JSON-Datei bereit.Die `JSONContentInAList`-Klasse ist die letzte Kindklasse und stellt die Information für die View zusammen. Eine Übersicht aller Klassen befindet sich im Anhang A.7.

5.1.2 Implementierung der Benutzeroberfläche (View)

Die View ist für die Präsentation bzw. Anzeige der Daten zuständig. JavaFX bietet die Möglichkeit alle Elemente einer GUI in einer FXML-Datei zu deklarieren. Mit Hilfe des Scene Builders werden die Elemente per Drag and Drop zusammengestellt. Dieser erzeugt automatisch den Inhalt der FXML-Datei in Form der erweiterbaren Auszeichnungssprache XML. Ein Ausschnitt befindet sich im Anhang A.5.

5.1.3 Implementierung der Geschäftslogik (Controller)

Über die Controller-Klasse erfolgt die Steuerung der Anwendung. Sie stellt das Bindeglied zwischen Model und View dar. Auch hier bietet JavaFX eine passende Lösung. Alle für die Anwendung erforderlichen Funktionen werden in einer Controller-Klasse geladen und über den Scene Builder mit dem jeweiligen Element der GUI verknüpft. Siehe im Anhang A.7 Abbildung 8: Aufbau der Controller-Klasse.

6 Projektabschluss

6.1 Abnahme

Nachdem die gesamte Anwendung fertig gestellt war, konnte diese dem Fachbereich zur Endabnahme vorgelegt werden. Auf Grund der agilen Softwareentwicklungsmethode wurde den Fachbereichen nach jedem Durchlauf die aktuelle Version der Anwendung präsentiert. Dadurch waren sie bei der Endabnahme bereits mit der Oberfläche und der Funktionsweise des Programmes vertraut. Außerdem konnten Anregungen und Kritik der Fachbereiche durch die stetigen Rücksprachen schon frühzeitig während der Entwicklungsphase berücksichtigt werden. Dadurch ergaben sich bei der Endabnahme keine Probleme oder Hindernisse mehr, sodass der Einführung der Anwendung nichts mehr im Wege stand.

Vor der Freigabe wurde zur Qualitätssicherung, zusätzlich zur Abnahme durch den Fachbereich, ein Code-Review durch einen anderen Entwickler durchgeführt.

6.2 Dokumentation

Die Dokumentation besteht aus drei Bestandteilen: der Projektdokumentation, dem Benutzerhandbuch und der Entwicklerdokumentation. In der Projektdokumentation beschreibt der Autor die einzelnen Phasen, die während der Umsetzung des Projektes durchlaufen wurden.

Das Benutzerhandbuch enthält Informationen über den Aufbau und die Funktionsweise der Anwendung. Es soll den Fachbereichen als Anhaltspunkt für Nachfragen zur Verfügung stehen und zur Einarbeitung neuer Mitarbeiter in die Anwendung dienen. Bei der Entwicklerdokumentation handelt es sich um eine detaillierte Beschreibung der Klassen, die in der Anwendung verwendet werden. Außerdem werden auch deren Attribute und Methoden sowie die Abhängigkeiten der Klassen untereinander erläutert. Diese Dokumentation soll dem Entwickler als Übersicht und Nachschlagewerk dienen. Mit Hilfe des Dokumentationstools Javadoc wurde diese Dokumentation automatisch generiert. Dazu werden aus den Kommentaren im Programmcode HTML-Dokumentationsdateien erzeugt. Ein Ausschnitt aus der Entwicklerdokumentation befindet sich im Anhang A.10. Zusätzlich zu der Entwicklerdokumentation wurde für jede Komponente ein Klassendiagramm aus dem Quellcode generiert.

7 Fazit

Rückblickend kann Festgehalten werden, dass alle Anforderungen an das Projekt erfolgreich vom Autor umgesetzt worden sind. Der gesetzte Projektplan, wie im Abschnitt 2.1 beschrieben wurden zeitlich eingehalten. Jedoch wurde der generelle Ablauf durch die agile Vorgehensweise umstrukturiert. Im Folgenden Abschnitt werden die detaillierten Phasen im Soll- /Ist-Vergleich tabellarisch dargestellt und die Gründe ihrer zeitlichen Differenz kurz erläutert.

7.1 Soll- und Ist-Vergleich

Wie bereits erwähnt haben sich auf Grund der agilen Softwareentwicklung und des damit einhergehenden stetigen Feedbacks des Fachbereiches die Phasen, wie die Analysephase und die Entwurfsphase deutlich verlängert.

Die verlorene Zeit konnte jedoch durch die Integration der Testphase in die Implementierungsphase und das Verschieben der Wirtschaftlichkeitsberechnung wieder aufgeholt werden. Diese Integration ist damit zu begründen, dass es sich als sinnvoll herausstellte, während der Softwareentwicklung und vor der Erstellung einer Methode einen Testfall zu schreiben. Es konnte dadurch sichergestellt werden, dass Änderungen in Klassen und Methode kein unerwünschtes Verhalten in der Anwendung mit sich bringen.

Projektphasen	Soll	Ist	Differenz
Analysephase	**2**	**7**	**+5**
Ist-Zustand ermitteln (Fachgespräch)	1	3	+2
Soll-Zustand ermitteln	1	2	+1
Wirtschaftlichkeit ermitteln	0	2	+2
Entwurfsphase	**11**	**14**	**+3**
Datenstruktur festlegen	3	4	+1
Benutzeroberfläche entwerfen	4	5	+1
Ablaufplan der Anwendung erstellen	4	5	+1
Implementierungsphase	**36**	**36**	**0**
JUnit-Tests erstellen	0	4	+4
Einrichten der Entwicklungsumgebung	2	1	-1
Implementierung der Model-Klassen	9	7	-2
Implementierung der View	10	9	-1
Implementierung der Controller-Klassen	12	15	+3
Schreiben einer Entwicklerdokumentation	3	0	-3
Test und Korrektur	**6**	**0**	**- 6**
JUnit Tests erstellen	4	0	-4
Fehlerbehebung	2	0	-2
Projektabschluss	**15**	**13**	**-2**
Projektabgabe	1	1	0
Wirtschaftlichkeit ermitteln	2	0	-2
Dokumentation	12	12	0
Gesamt	**70**	**70**	**0**

Tabelle 3: Detaillierter Soll- und Ist-Vergleich in Zeitstunden

7.2 Ausblick

Alle Anforderungen des Projektes konnten umgesetzt werden und doch ist der Funktionsumfang der Anwendung übersichtlich.

Es wäre daher angebracht für die Zukunft weitere Features, wie zum Beispiel einen Neustart eines Servers direkt von der Anwendung aus, einzuplanen. Entweder durch implementieren einer Befehlseingabemaske, welche den Zugriff über PuTTY ermöglicht oder durch Hinzufügen des Befehlssatzes bei der Erstellung einer Verbindung in die Tabelle.

Darüber hinaus wäre auch eine Anpassung der Benutzeroberfläche an das Corporate Design der BMT denkbar. Auch hier bietet JavaFX die Möglichkeit dieses Vorhaben mittels einer CSS-Datei zu realisieren. Durch die sorgfältige Dokumentation mittels Javadoc und einer strikten Trennung der Softwarearchitektur (MVC-Modell) wird eine gute Erweiterbarkeit und Wartbarkeit des Quellcodes durch andere Entwickler ermöglicht.

Literaturverzeichnis

(10.05.2018)

C. Kecher, A. Salvanos: UML 2.5 - Das umfassende Handbuch

© Rheinwerk Verlag GmbH, Bonn 2015 – ISBN 978-8362-2977-7

Wikipedia (10.05.2018)

- https://de.wikipedia.org/wiki/PuTTY
- https://de.wikipedia.org/wiki/Agile_Softwareentwicklung
- https://de.wikipedia.org/wiki/Scrum
- https://de.wikipedia.org/wiki/JavaScript_Object_Notation
- https://de.wikipedia.org/wiki/Git

Weitere Quellen (10.05.2018)

- https://balsamiq.cloud
- https://www.bitmarck.de
- https://www.duden.de/rechtschreibung/serialisieren
- https://www.sourcetreeapp.com

Abkürzungsverzeichnis

BMT, 1, 2, 3, 5, 6, 9 ..... BITMARCK Teachnik GmbH

FXML, 8, 10, xviii ..... JavaFX Extensible Markup Language

GKV, 1 ..... Gesetzliche Krankenversicherung

GS, 1, 2, 4 ..... Geschäftsstelle

GSON, 7, xvi ..... JSON Bibliothek für Eclipse

GUI, 10, xvi ..... Graphical User Interface

HTML, 4, 11 ..... Hypertext Markup Language

IP, 9 ..... Internet Protocol

JSON, 7, 8, 9, 10, xvii, xxi ..... JavaScript Object Notation

KV, 1 ..... Krankenversicherung

MVC, 4, 9, 12 ..... Model View Controller

A Anhang

A.1 Verwendete Ressourcen

Hardware

- ✓ Büroarbeitsplatz mit Fat-Client

Software

- ✓ Windows 7 Service Pack 1 – Betriebssystem
- ✓ Eclipse Oxygen 4.7.3a – Entwicklungsumgebung Java
  - Import von Bibliotheken: JavaFX, JUnit 4, GSON
- ✓ Microsoft Office 365 – Bürosoftware
- ✓ Sourcetree (Git) – Verteilte Versionsverwaltung
- ✓ JavaFX Scene Builder 9.0.1 – Tool zum erstellen der GUI
- ✓ UMLet 14.2 Eclipse – Tool zum erstellen von UML-Diagrammen
- ✓ Balsamiq.cloud– Webbasiertes Tool zum erstellen von Mockups

Personal

- ✓ Entwickler - Projektverantwortlicher und -ersteller
- ✓ Anwendungsentwickler – Review des Codes, Festlegung der Anforderungen und Abnahme des Projektes

A.2 Nutzwertanalyse zur Auswahl eines Datenmodells

Eigenschaft	Gewich-tung[6]	JSON		XML	
		Punkte[7]	Bewertung[8]	Punkte	Bewertung
Lesbarkeit	15%	5	0,75	3	0,45
Kosten	20%	1	0,2	1	0,2
Performance	25%	5	1,25	2	0,5
Kenntnisstand	20%	3	0,6	1	0,2
Verwendbarkeit	15%	5	0,75	3	0,45
Entwicklungsumgebung	5%	3	0,15	3	0,15
Gesamt	**100%**		**3,7**		**1,95**

Tabelle 4: Nutzwertanalyse einer geeigneten Datenhaltung

A.3 Grundaufbau der JSON-Datei


```
{
    "settings":[
        {
            "interval":100
        }
    ],
    "server":[
        {
            "name":"...",
            "host":"a.b.com",
            "ip":"1.2.3.4"},
        {
            "name":"...",
            "host":"e.f.google",
            "ip":"123.23.34"
        }
    ],
    "ports":[
        {
            "name":"...",
            "port":8080
        },
        {
            "name":"...",
            "port":90909
        }
    ],
    "querys":[
        {
            "ip":"1.2.3.4",
            "port":"8080"
            ,"creationDate":"14.05.2018[11:59:58]"
        },
        {
            "server":"1.2.3.4",
            "port":"90909",
            "creationDate":"14.05.2018[11:59:59]"
        }
    ]
}
```


Abbildung 1: Screenshot einer beispielhaften JSON-Datei

[6] Anteile der Gesamtgewichtung in Prozent aufgeteilt.
[7] Punkteskala: Von 1 = schlecht bis 5 = sehr gut.
[8] Bewertung ergibt sich aus der Multiplikation des Anteils der Gewichtung und der Anzahl der Punkte.

A.4 Oberflächenentwurf

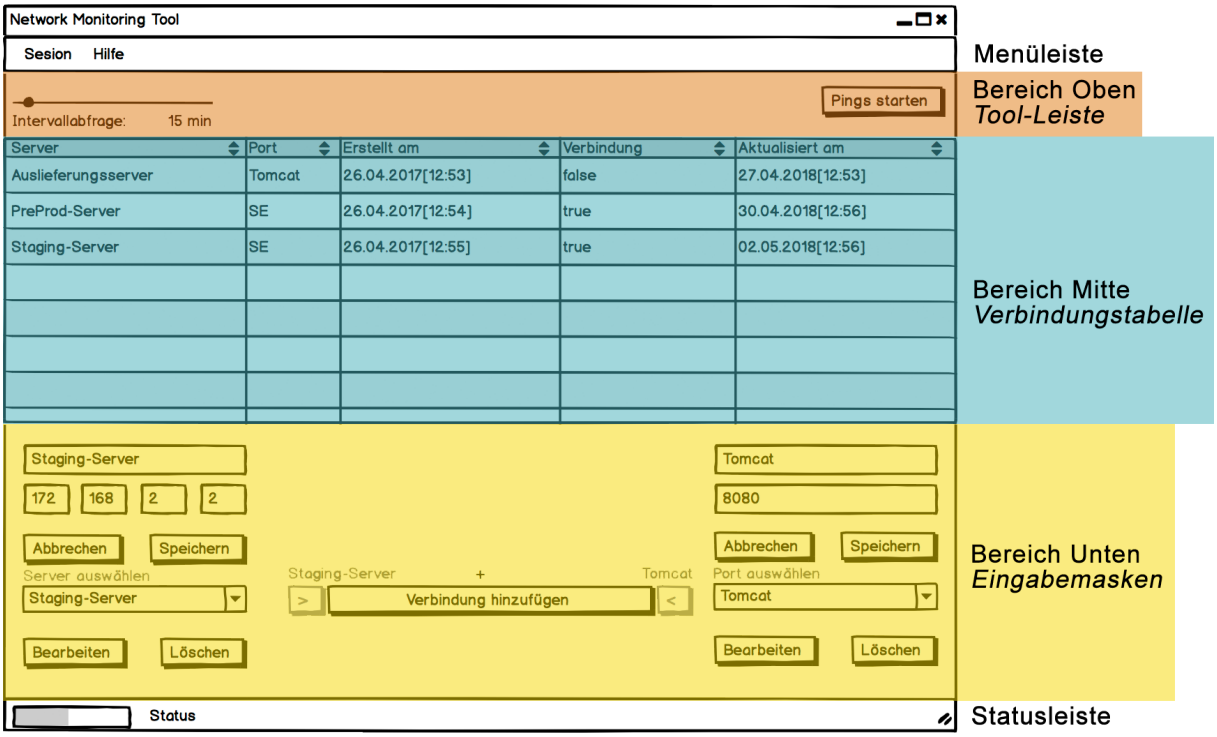

Abbildung 2: Oberflächenentwurf der Hauptansicht (aufgeteilt)

A.5 Auszug der FXML-Datei

Abbildung 3: Ausschnitt der FXML-Datei für die View

A.6 Aktivitätsdiagramm

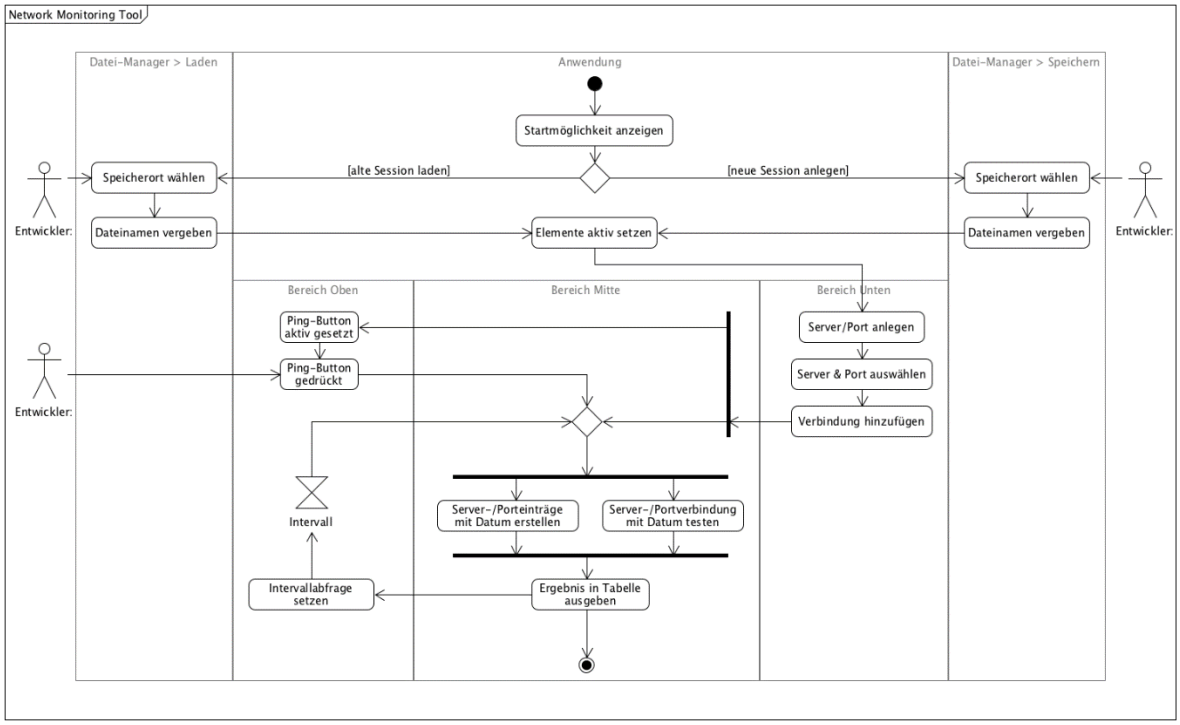

Abbildung 4: Anwendungsfall als Aktivitätsdiagramm

A.7 Klassendiagramme

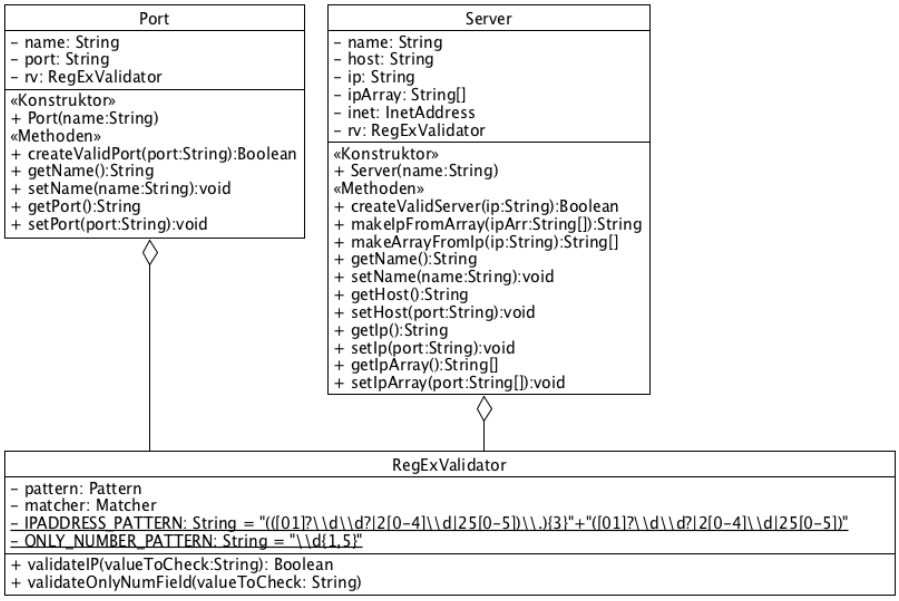

Abbildung 5: Klassendiagramm - Aufbau der Server- und Portklassen

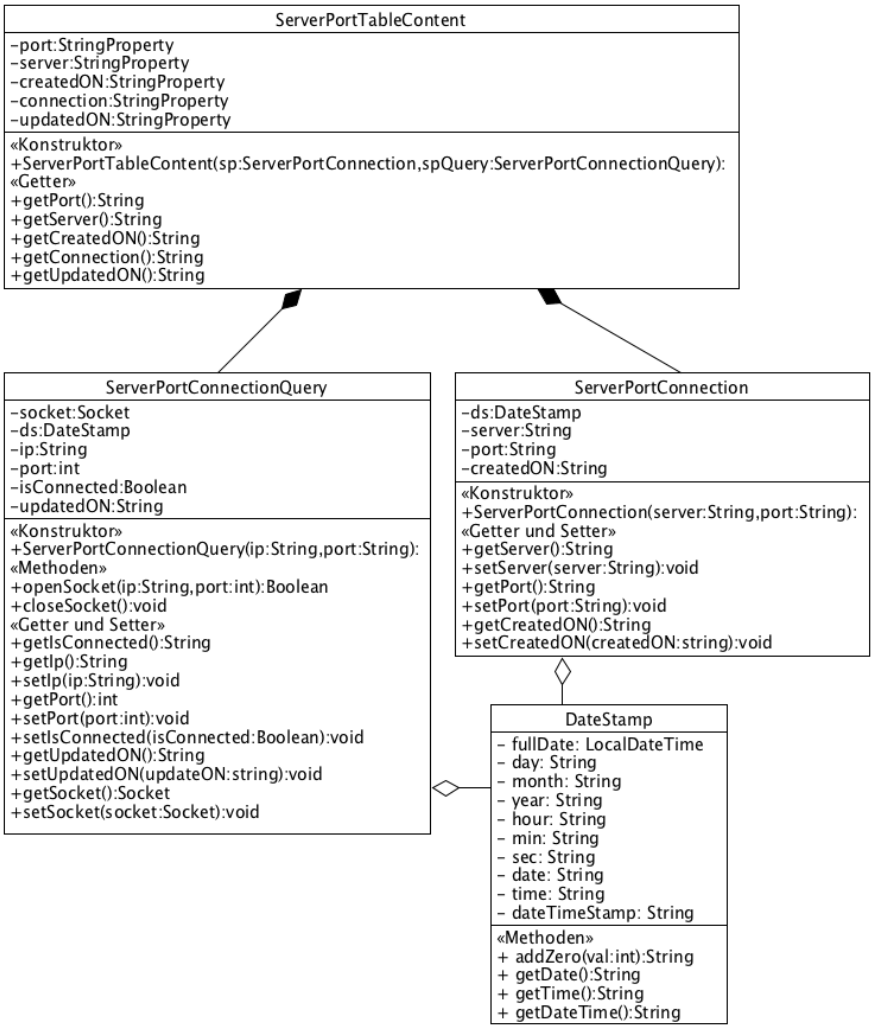

Abbildung 6: Klassendiagramm - Aufbau des Tabelleninhaltes

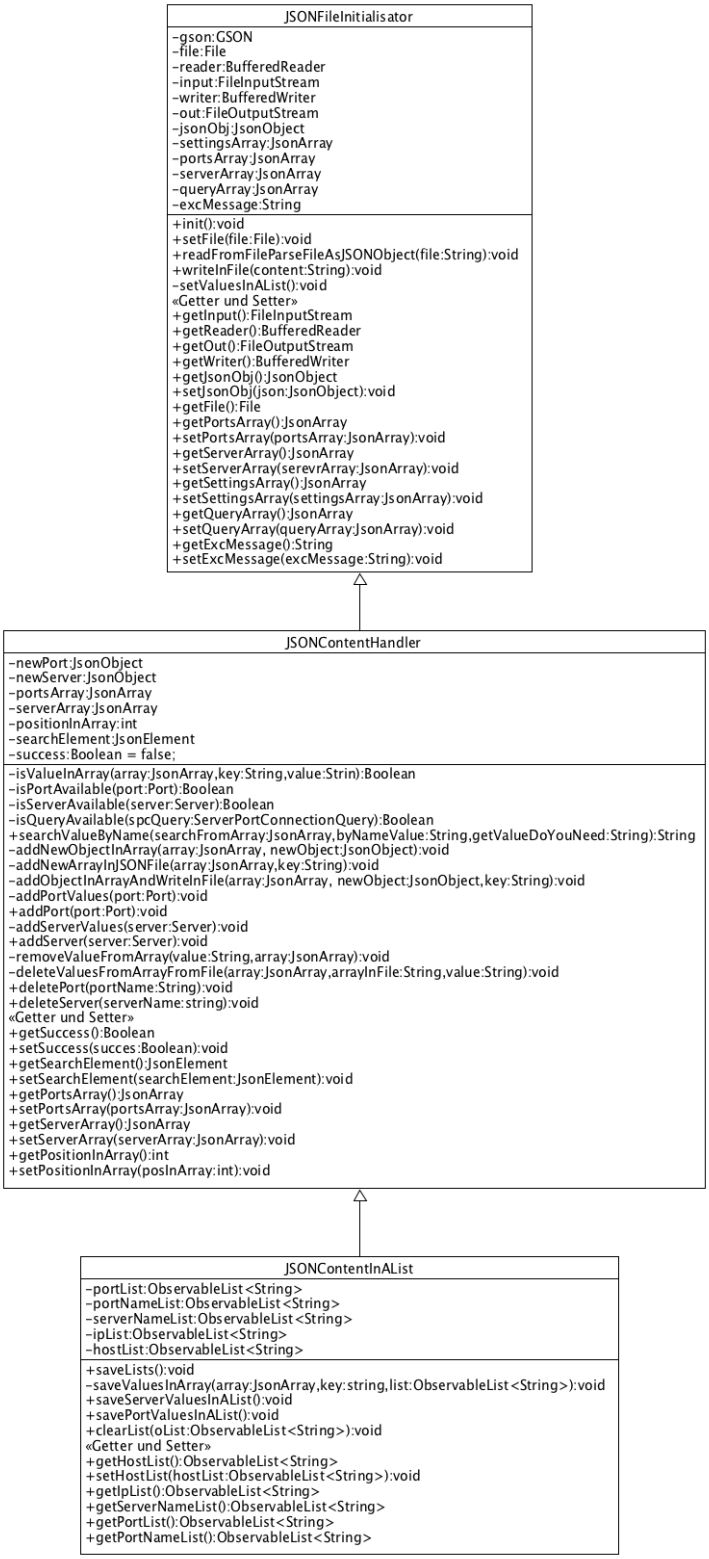

Abbildung 7: Klassendiagramm - Lesen des Inhalts einer JSON-Datei

Abbildung 8: Aufbau der Controller-Klasse

A.8 Screenshot Scene Builder

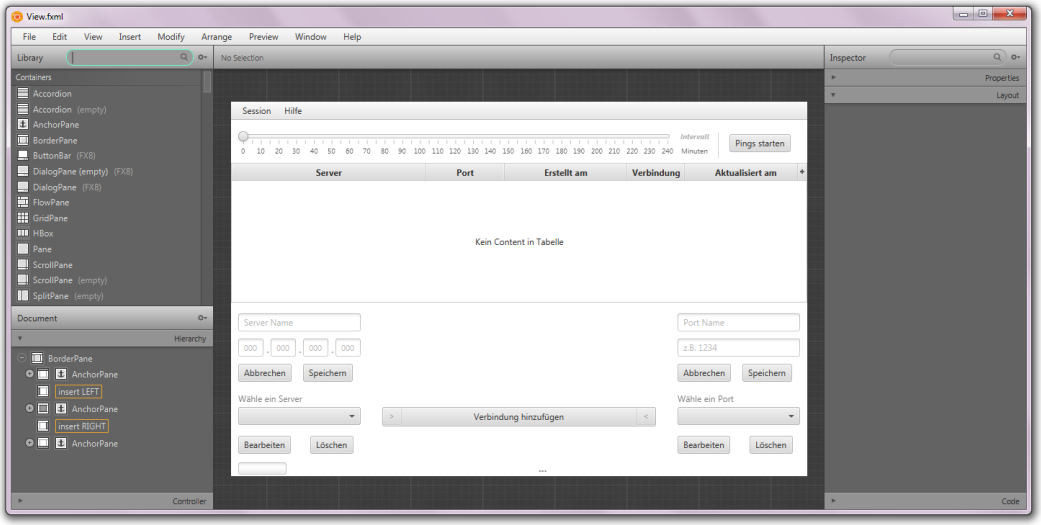

Abbildung 9: Benutzeroberfläche des Scene Builders

A.9 Screenshot der fertigen Anwendung

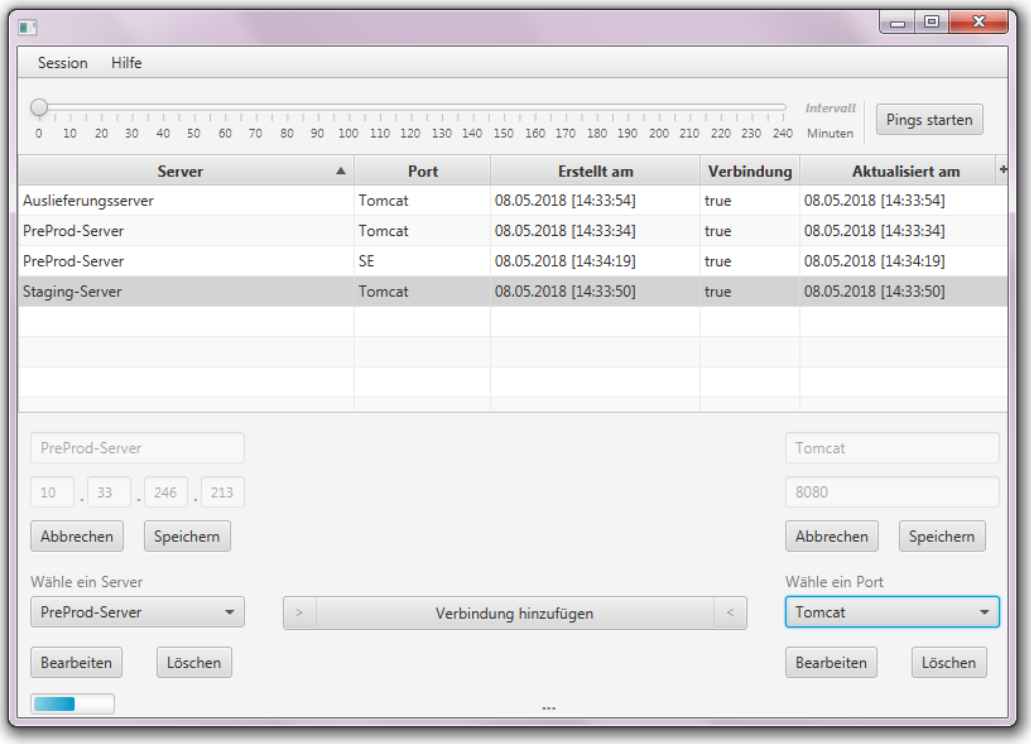

Abbildung 10: Screenshot der Anwendung

A.10 Auszug der Entwicklerdokumentation

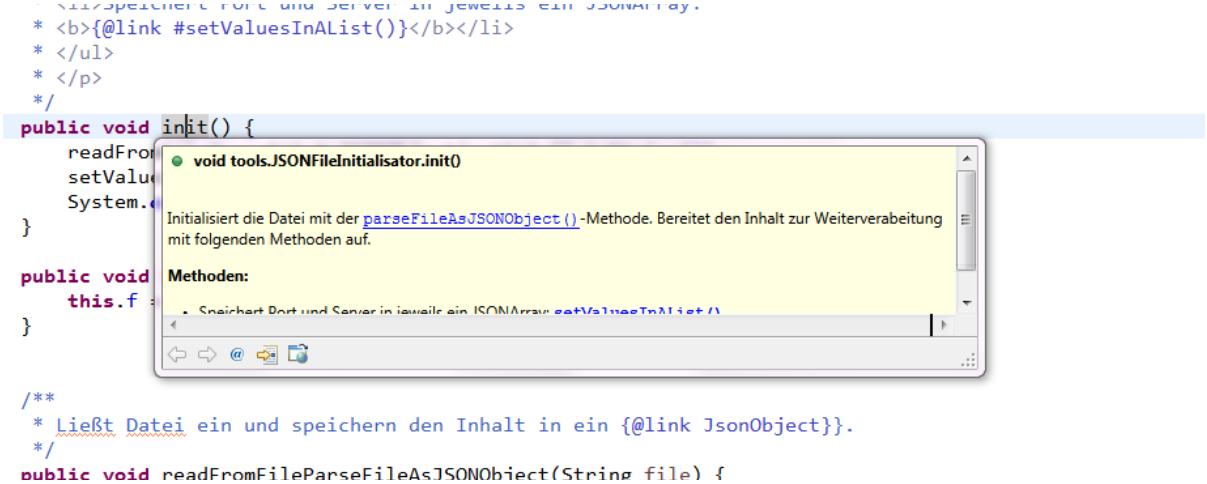

Abbildung 11: Ausschnitt einer Javadoc-Ausgabe