HOCHSCHULE
MITTWEIDA
UNIVERSITY OF
APPLIED SCIENCES

BACHELORARBEIT

Herr
Marcel Berger

Untersuchungen zur Verbesserung des Lehrmoduls Datenrepräsentation: Technologien und APIs

2018

Fakultät Angewandte Computer- und Biowissenschaften

BACHELORARBEIT

Untersuchungen zur Verbesserung des Lehrmoduls Datenrepräsentation: Technologien und APIs

Autor:

Herr Marcel Berger

Studiengang:

Angewandte Informatik – Softwareentwicklung

Seminargruppe:

IF15wS-B

Erstprüfer:

Herr Prof. Dr.-Ing. Wilfried Schubert

Zweitprüfer:

Herr Rico Beier, M.Sc.

Einreichung:

Mittweida, 1.10.2018

Faculty of Applied Computer Sciences and Biosciences

BACHELOR THESIS

Studies to improve the module data representation: technologies and APIs

Author:
Mr. Marcel Berger

Course of studies:
Applied Computer Sciences

Seminar group:
IF15wS-B

First examiner:
Mr. Prof. Dr.-Ing. Wilfried Schubert

Second examiner:
Mr. Rico Beier, M. Sc.

Submission:
Mittweida, 1.10.2018

Bibliografische Angaben:

Berger, Marcel:

Untersuchungen zur Verbesserung des Lehrmoduls Datenrepräsentation: Technologien und APIs

Studies to improve the module data representation: technologies and APIs

2018 – 49 Seiten

Mittweida, Hochschule Mittweida (FH), University of Applied Sciences,

Fakultät Angewandte Computer und Biowissenschaften, 2018

Abstract

Das Lehrmodul „Datenrepräsentation“ wird im 3 Semester in den Studiengängen der Informatik an der Hochschule Mittweida durchgeführt. In den Vorlesungen werden dabei verschiedene Datenrepräsentationstechnologien vorgestellt. Zur praktischen Umsetzung dieser Technologien finden außerdem eine Reihe von Praktikumsveranstaltungen statt. Diese Arbeit behandelt die Verbesserung und Erweiterung ausgewählter Praktikumsinhalte des Lehrmoduls Datenrepräsentation. Bestehende Praktikumseinheiten werden zunächst hinsichtlich der Struktur analysiert. Danach werden Verbesserungsmöglichkeiten erläutert und schließlich realisiert. Im Rahmen dieser Arbeit werden außerdem neue Praktikumseinheiten konzipiert, welche unter anderem die Technologien JSON und RESTful Webservices behandeln. Es werden neue Praktikumsanleitungen erstellt sowie Beispielprogramme implementiert, welche in den Praktikumsveranstaltungen des Lehrmoduls eingesetzt werden können.

Inhaltsverzeichnis

Abkürzungsverzeichnis

API	Application Programming Interface, Anwendungsprogrammierschnittstelle
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
JAR	Java Archive
JAX-RS	Java API for RESTful Webservices
JSON	JavaScript Object Notation
REST	Representational State Transfer
SAX	Simple API for XML
SOAP	Simple Object Access Protocol
StAX	Streaming API for XML
UDDI	Universal Description, Discovery and Integration
URI	Uniform Resource Identifier
WSDL	Web Services Description Language
XML	Extensible Markup Language
XSLT	Extensible Stylesheet Languages Transformation

Abbildungsverzeichnis

Tabellenverzeichnis

1. Einleitung

1.1. Motivation

Das Lehrmodul „Datenrepräsentation“ ist seit 2007 ein Teil der Informatik-Studiengänge an der Hochschule Mittweida und wird im 3. Semester eines Informatik-Studiengangs mit jeweils einer Vorlesung pro Woche durchgeführt. Weiterhin findet alle zwei Wochen eine Praktikumsveranstaltung in den PC-Pools der Hochschule statt. In den Praktikumsveranstaltungen erfolgt die technische Umsetzung der in den Vorlesungen vorgestellten Dateirepräsentationstechnologien. Teilnehmende der Praktikumsveranstaltungen führen dabei unter der Nutzung von Praktikumsanleitungen verschiedene Programmieraufgaben durch und implementieren Funktionen der jeweiligen Datenrepräsentationstechnologie. Die im Lehrmodul behandelten Technologien basieren dabei fast ausschließlich auf XML, da sich diese Auszeichnungssprache in vielen Bereichen der Datenrepräsentation etabliert hat. XML besitzt allerdings auch einige Nachteile. Als ein Beispiel dafür kann der große Overhead von XML genannt werden. Dieser kann z. B. die Effizienz der Übertragung von Daten zwischen einem Server und einem Client verringern. In solchen Bereichen haben sich andere Datenrepräsentationsformate durchgesetzt. Dabei steht vor allem JSON im Fokus, welches sich als Standardformat in vielen Webanwendungen etabliert hat.[1] Auch im Bereich der Webservices haben sich neue Technologien wie „RESTful Webservices“ durchgesetzt. Damit das Lehrmodul Datenrepräsentation weiterhin auf dem aktuellsten Stand bleibt, müssen somit einige Verbesserungen an den Inhalten des Lehrmoduls durchgeführt werden. Des Weiteren wird die Anzahl der Lehrveranstaltungen im Hinblick auf die Modernisierung des Lehrmoduls und der Umbenennung auf „Datenrepräsentation: Technologien und APIs“ weiter ansteigen, weshalb verbesserte und neue Praktikumsinhalte benötigt werden.

1.2. Aufgabenstellung und Ziel

Diese Arbeit beschäftigt sich mit der Verbesserung und Erweiterung von Inhalten der zu Beginn erwähnten Praktikumsveranstaltungen des Lehrmoduls Datenrepräsentati-

1 Vgl. Vogel, Lucas (2017): „JSON vs. XML: The battle for format supremacy may be wasted energy“.

on. Im Rahmen der Arbeit werden ausgewählte Inhalte bestehender Praktikumsveranstaltungen untersucht und hinsichtlich der Benutzbarkeit, des technologischen Standes sowie der Verständlichkeit verbessert. Die einzelnen Teilaufgaben für die Verbesserung bestehender Inhalte werden im folgenden Absatz erläutert.

Grundsätzlich werden für bestehende Praktikumsinhalte zunächst Recherchen hinsichtlich der technischen Entwicklung der jeweils behandelten Technologie ausgeführt. Danach werden die recherchierten Ergebnisse zusammengetragen und es wird eine Analyse der bestehenden Praktikumsinhalte durchgeführt. Dabei wird eine Übersicht über einzelne Vorgaben der Praktikumsinhalte sowie Abläufe in der Praktikumsdurchführung geschaffen. Schließlich werden mögliche Änderungen, Verbesserungen und Erweiterungen für bestehende Praktikumsinhalte erarbeitet und implementiert. Bei den bestehenden Praktikumsinhalten handelt es sich um die Technologien XML-SAX und Webservices mit SOAP. Für die Verbesserung des Praktikums zur Technologie XML-SAX steht eine bestimmte Hauptaufgabe im Vordergrund. Die im Praktikum verwendete grafische Benutzeroberfläche zur Verdeutlichung von Funktionen der Schnittstelle SAX soll von Java Swing auf Java FX umgestellt werden. Das Praktikum zum Thema Webservices mit SOAP soll durch Inhalte der Technologie RESTful Webservices erweitert werden. Zusätzlich soll betrachtet werden, ob mögliche Alternativen für die im Praktikum Webservices verwendeten Frameworks vorhanden sind. Für die Technologien XML-StAX und JSON werden komplett neue Praktikumsinhalte angefertigt. Beide Themen werden gegenwärtig noch nicht bzw. nicht vollständig in den Praktikumsveranstaltungen behandelt. Für die neuen Praktikumsinhalte werden zunächst theoretische Grundlagen erarbeitet. Danach werden Konzepte für den Aufbau neuer Praktikumsanleitungen erstellt. Schließlich werden die Entwürfe realisiert und es werden Praktikumsinhalte geschaffen und an Programmbeispielen erläutert. Nach der Realisierung der Praktikumsinhalte soll schließlich überprüft werden, ob sämtliche Inhalte, wie z. B. Programmbeispiele, in den PC-Pools der Hochschule Mittweida fehlerfrei verwendet werden können. Als Bezugspunkte dienen die PC-Pools im Haus 8 der Hochschule.

Eine weitere Aufgabe behandelt die Umstellung der in den Praktika verwendeten Beispielthemen auf ein neues Praxisbeispiel mit der Bezeichnung „Elektronikrechnungen“. Die Beispielthemen enthalten fiktive Datensätze, welche für den Test von Funktionen der in den Praktika behandelten Technologien verwendet werden. Zu den bestehenden Praxisbeispielen sind XML Dateien mit Beispieldaten sowie Beschreibungen in den Praktikumsanleitungen vorgegeben. Für die Umstellung werden neue Beispieldaten zum Thema „Elektronikrechnungen“ erzeugt und in die Praktikumsanleitungen integriert. Die im Rahmen dieser Arbeit durchzuführende Umstellung erfolgt aus Gründen

der Verständlichkeit und des fehlenden Bezugs zum alltäglichen Umfeld eines Praktikumsteilnehmers.

Das Ziel dieser Arbeit ist die Verbesserung und Erstellung von ausgewählten Praktikumsinhalten des Lehrmoduls Datenrepräsentation. Nach der Analyse, dem Entwurf und der Umsetzung werden zum Schluss der Arbeit überarbeitete bzw. neue Praktikumsanleitungen sowie Vorgaben (Bibliotheken, Programmbeispiele, Vorlagen, usw.) bereitstehen, welche im praktischen Unterricht es Lehrmoduls Datenrepräsentation eingesetzt werden können.

1.3. Abgrenzung

Die vorliegende Arbeit beschäftigt sich ausschließlich mit den Inhalten der Praktikumsveranstaltungen. Inhalte aus den Vorlesungen des Moduls Datenrepräsentation werden in dieser Arbeit nicht behandelt. Im ersten Teil dieser Arbeit werden theoretische Grundlagen zu Technologien erläutert, welche für diese Arbeit relevant sind. Bei den theoretischen Grundlagen liegt der Schwerpunkt vor allem auf den neueren Technologien, welche in den Praktikumsveranstaltungen noch nicht umfassend behandelt wurden. Grundlagen zu bereits behandelten Themen werden in kürzerer Fassung dargestellt. Ein Hauptteil dieser Arbeit beschäftigt sich auch mit der Programmiersprache Java. Anhand von Programmbeispielen werden bestimmte Funktionen der jeweils behandelten Technologie vorgestellt. Sämtliche im Rahmen dieser Arbeit erstellten Java-Anwendungen werden auf Basis der Java Standard Edition 8 sowie durch die Nutzung der Eclipse-IDE angefertigt. Auf die mögliche Einbindung der erstellten Java-Programme in Entwicklungsumgebungen wird in dieser Arbeit nicht weiter eingegangen. Zum Schluss der Arbeit werden sämtliche neue und überarbeitete Praktikumsinhalte mit entsprechenden Anleitungen und Vorgaben bereitstehen. Wie die Inhalte in das Lehrmodul „Datenrepräsentation“ eingeordnet werden, wird in dieser Arbeit nicht erläutert.

1.4. Kapitelübersicht

Nach der Einleitung werden im zweiten Kapitel theoretische Grundlagen zu den Begriffen aus dem Bereich der Datenrepräsentation erläutert. Dabei werden hauptsächlich Grundlagen zu moderneren Technologien wie JSON und REST hervorgehoben. Das dritte Kapitel behandelt Praxisbeispiele, welche in den bestehenden Praktika verwendet werden. Die Praxisbeispiele werden hinsichtlich der Verständlichkeit und der Komplexität bewertet. Auf Basis dieser Bewertung wird schließlich ein neues Praxisbeispiel

zum Thema „Elektronikrechnungen“ konzipiert. Das vierte Kapitel dreht sich um die Thematik der bestehenden Praktika. Eine einzelne Praktikumseinheit wird zunächst hinsichtlich der Struktur und des Ablaufs untersucht. Danach werden Verbesserungsmöglichkeiten zu den einzelnen Praktikumseinheiten erläutert. Im letzten Teil des vierten Kapitels erfolgt die Realisierung der Verbesserungen. Im fünften Kapitel werden neue Praktikumsthemen behandelt. Für die neuen Praktikumseinheiten zu den Technologien JSON, StAX und RESTful Webservices werden zu Beginn des Kapitels Konzepte für Abläufe und Strukturen angefertigt. Im Anschluss erfolgt die Realisierung dieser Konzepte. Bei der Realisierung stehen vor allem Beispielprogramme im Vordergrund, die im Rahmen der Entwicklung neuer Praktika implementiert wurden. Mithilfe von Programmausschnitten werden wichtige Funktionen der Beispielprogramme erläutert. Im letzten Kapitel werden die Ergebnisse dieser Arbeit betrachtet und es erfolgt ein Fazit.

2. Begriffe und theoretische Grundlagen

Das folgende Kapitel behandelt begriffliche und theoretische Grundlagen aus dem Gebiet der Datenrepräsentation. Hierbei stehen RESTful Web Services und JSON im Vordergrund, da diese Technologien seltener in den Praktikumsveranstaltungen behandelt wurden. Die Grundlagen zu bekannteren Technologien werden in verkürzter Form erläutert.

2.1. Begriffe

2.1.1. SOAP

SOAP (Simple Object Access Protocol) ist ein auf XML-basierendes plattformunabhängiges Netzwerkprotokoll. Es dient zum Austausch von strukturierten Informationen in verteilten Systemen. SOAP besitzt dazu eine erweiterbare Funktionalität für den Austausch von Nachrichten in einem dezentralisierten Computersystem.[2]

Eine SOAP Nachricht besteht aus mehreren verschachtelten XML-Elementen. Das Wurzelelement bildet der „Envelope“ (Umschlag). Dieses Element enthält zwei weitere Unterkomponenten. Die erste Komponente ist der SOAP „Header“, welches ein optionales Element der SOAP Nachricht ist. Das Element enthält dabei optionale Informationen, welche für die Verarbeitung durch eine Anwendung relevant sein könnten. Diese bestehen u. a. aus Anweisungen für die Übertragung und Kontextinformationen. Anweisungen in einem Header können als Pflichtinformation gesetzt werden. In einen solchen Fall müssen die Anweisungen von Anwendungen, welche den SOAP Header verarbeiten, umgesetzt werden. Der Header wird von allen zur Übertragung beitragenden Anwendungen verarbeitet. Der tatsächliche Inhalt einer SOAP Nachricht wird im zweiten Unterelement der Envelope angegeben, dem SOAP „Body“. Dieses Element enthält die Nachricht des Senders an den Empfänger sowie Informationen zu Fehlern, welche bei der Übertragung zwischen dem Sender und dem Empfänger aufgetreten sind. Der Body einer SOAP Nachricht wird i. d. R. nur vom Empfänger der SOAP Nachricht verarbeitet.[3]

2 Vgl. Mitra, Lafon (2007): „SOAP Version 1.2 Part 0: Primer (Second Edition)“, Abschnitt 1.1 „Overview“.
3 Vgl. Mitra, Lafon: Abschnitt 2.1 „SOAP Messages“.

2.1.2. WSDL

Zur Beschreibung von Web Services wird die auf XML aufbauende Technologie WSDL (Web Services Description Language) verwendet. WSDL beschreibt die Angebotsbreite von Funktionen eines Web Services.[4] Die Angebote von Funktionen eines Web Services werden dabei in einem WSDL Dokument dargestellt. Diese Angebote geben an, welche Funktionen von einem Client ausgeführt werden können. Sollte ein Client eine Funktion des Web Services ausführen wollen, dann beschreibt das WSDL Dokument den Vorgang für die Interaktion des Clients mit dem Web Service.[5] Für WSDL gibt es die Version 1.1 und 2.0. Die Version 2.0 wird durch einen W3C Standard beschrieben. Der Aufbau eines WSDL 2.0 Dokuments wird im folgenden Abschnitt erläutert.

Das Wurzelelement eines WSDL 2.0 Dokuments bildet die Komponente „Description“. Dieses Element enthält alle weiteren WSDL 2.0 Hauptkomponenten. Die erste Hauptkomponente bildet das „Interface“ Element. Dieses fasst Operationen des Web Services als einzelne Unterelemente zusammen. Ein Operationselement enthält dabei Beschreibungen für die Ein- und Ausgabe von Werten einer Funktion.[6] Weiterhin enthält das Wurzelelement die Komponente „Binding“. Diese Komponente definiert die nötigen Schritte zur Realisierung der Interaktion mit dem Web Service. Dabei werden die benötigten Übertragungsprotokolle definiert, welche bei der Interaktion mit dem Web Service genutzt werden können. Für einzelne Operation eines Interface-Elements können außerdem zu verwendende Übertragungsprotokolle (u. a.) festgelegt werden.[7] Im Service-Element eines WSDL Dokuments befindet sich eine Reihe von Endpunktelementen (Endpoints). In diesen Elementen werden URLs des Web Services angegeben. Auch hier können für verschiedene Übertragungsprotokolle mehrere URLs angegeben werden. Schließlich enthält die Description Komponente das Element „Types“. In diesem Element werden die Datentypen angegeben, welche vom Web Service verwendet werden. Diese Datentypen werden u. a. den Ein- und Ausgabewerten der Operationen des Web Services zugeordnet.[8]

Zusammengefasst beschreiben WSDL Dokumente die vorhandenen Funktionen eines Web Services, welche von einem Client verwendet werden können. Das Dokument enthält Informationen über URL und Operationen sowie einsetzbare Übertragungsprotokolle des Web Services.

4 Vgl. Chinnici et al. (2007): „Web Services Description Language (WSDL) Version 2.0 Part 1“, S.6.
5 Vgl. Chinnici et al., S.7.
6 Vgl. Chinnici et al., S.12 ff.
7 Vgl. Chinnici et al., S.37.
8 Vgl. Chinnici et al., S.53 ff.

2.1.3. Unterschiede zwischen StAX und SAX

Die Technologien SAX (Simple API for XML) und StAX (Streaming API for XML) sind zwei Programmierschnittstellen für die Verarbeitung von XML Dokumenten. Ein XML Dokument wird, im Vergleich zu DOM, bei SAX und StAX durch einen Datenstrom (Stream) eingelesen. Ein XML Dokument wird dazu seriell, von Anfang bis Ende, durchlaufen und während der Programmlaufzeit verarbeitet.[9] Die Unterschiede zwischen beiden APIs werden im folgenden Abschnitt erläutert.

Ein Hauptunterschied zwischen beiden Schnittstellen ist die Fähigkeit des schreibenden Zugriffs auf eine XML Datei. Die API StAX kann sowohl lesend als auch schreibend auf ein XML Dokument zugreifen. Bei der Schnittstelle SAX ist nur der lesende Zugriff auf ein XML Dokument möglich. Der zweite Hauptunterschied zwischen beiden Technologien liegt in der Funktionsweise der Verarbeitung von XML Dokumenten. SAX wird als eine „Push-Style“ API und StAX als eine „Pull-Style“ API bezeichnet. Die Begriffe „Push“ und „Pull“ werden aufgrund der Art und Weise des Einlesens von XML Elementen verwendet. Bei der Implementierung von SAX in der Programmiersprache Java wird ein „Handler“ erzeugt, welcher Ereignisse des SAX Parsers behandelt. Der SAX Parser durchläuft jedes Element eines XML Dokuments und löst Ereignisse aus, welche durch den genannten Handler verarbeitet werden. Der SAX Parser „drückt“ somit Inhalte der XML Datei auf den in Java implementierten Handler. Bei der Verwendung von StAX wird der Handler nicht durch den Parser aufgerufen, da der StAX-Parser selbst gesteuert werden kann. Das XML Dokument kann entweder mit einem Cursor oder mit einem Iterator für XML Ereignisse durchlaufen werden.[10]

2.2. JSON

2.2.1. Begriffsklärung

JSON ist die Abkürzung von „JavaScript Object Notation“ und beschreibt ein Textformat zum Austausch und zur Strukturierung von Daten. JSON ist eine Alternative zu dem bekannten XML Format. Wie auch XML wird JSON zum Austausch von Daten zwischen unterschiedlichen Systemen verwendet.[11]

Der Begriff „JavaScript“ in der Abkürzung JSON bezieht sich auf die Basiskomponen-

9 Vgl. ORACLE.COM: „Why StAX“, Überschrift: „Streaming versus DOM“.
10 Vgl. Jenkov: „Java StAX“.
11 Vgl. Bassett: „Introduction to JavaScript Object Notation“, S.1-2.

ten des JSON Formats. Diese werden durch die Skriptsprache JavaScript dargestellt. Dabei setzt JSON allerdings nur die JavaScript-Syntax zur Zuweisung von Werten auf Objekte um. JavaScript Funktionen werden in JSON nicht verwendet. Der Begriff „Object“ leitet sich aus dem Programmierkonzept der Objektorientierten Programmierung ab. Das Word „Notation“ steht für die Richtlinien der Verwendung von Textzeichen (z. B. Buchstaben und Zahlen) zur Repräsentation von Daten in Textform.[12]

2.2.2. Sytnax

Die JSON Syntax setzt ein Konzept um, welches in der Datenverarbeitung häufig verwendet wird. Dieses Konzept besteht aus der Bildung von Paaren aus Namen und Werten. Ein Name bezieht sich dabei auf die Bezeichnung einer Variable, welche einem Wert zugeordnet ist.[13]

```
1    {
2        "typ": "Bachelorarbeit",
3        "autor": "Marcel Berger",
4        "jahr": 2018
5    }
```

Im oben dargestellten Beispiel wird die Syntax von JSON dargestellt. Dabei sind die Paare aus Namen (Variablen) und Werten zu erkennen. Der Wert und der Name eines Paares werden dabei durch einen Doppelpunkt getrennt und die Paare selbst werden hintereinander angegeben und durch ein Komma getrennt. Die Namen der Paare werden immer in Anführungsstrichen dargestellt. Die Verwendung von Anführungsstrichen zur Darstellung der Werte ist abhängig vom Datentyp des Wertes (siehe Kapitel 2.2.3). Im oben dargestellten Beispiel wird der Wert „Bachelorarbeit“ des Namens „typ“ in Anführungsstrichen dargestellt, da es sich hierbei um eine Zeichenkette handelt. Alle Paare sind außerdem von zwei geschweiften Klammern umgeben. Die Klammern stellen ein Objekt dar, welches im Besitz der zwischen den Klammern stehenden Paare ist. Die geöffnete Klammer deutet den Beginn des Objekts an und die schließende Klammer das Ende des Objekts.[14]

Um die Korrektheit der Syntax eines JSON Textes (wie z. B. im oben dargestellten Beispiel) zu überprüfen, werden „JSON Validator“ verwendet. Ein Validator ist ein Tool, welches einen JSON Text einliest, die Syntax überprüft und das Ergebnis der Überprüfung anzeigt. Eine inkorrekte Syntax wird im Ergebnis der Validierung als Fehlermeldung angezeigt. Eine Fehlermeldung enthält i. d. R. einen Verweis auf die Stelle des

12 Vgl. Bassett, S.3.
13 Vgl. Bassett, S.6.
14 Vgl. Bassett, S.8-9.

Fehlers im JSON Text.[15]

2.2.3. Datentypen

JSON verwendet die primitiven Datentypen, welche in den meisten Programmiersprachen verwendet werden. Der Datentyp „Objekt“ wurde bereits im Kapitel 2.2.2 erwähnt. Dieser Datentyp besteht demnach aus einer Reihe von Paaren aus Namen und Werten. Da einem Namen auch ein Objekt zugeordnet werden kann, trägt dieser Datentyp auch zur Verschachtelung von weiteren Objekten bei. Im unten dargestellten Beispiel ist eine solche Verschachtelung zu erkennen. Dem Namen „bachelorarbeit“ wird dabei ein Objekt mit weiteren Namen und Werten zugeordnet. Diese Verschachtelung von Objekten dient zur Strukturierung von Daten.[16]

```
1    {
2       "bachelorarbeit": {
3          "titel": "Untersuchungen zur Verbesserung...",
4          "autor": "Marcel Berger",
5          "jahr": 2018,
6          "seiten": 50
7       }
8    }
```

Ein weiterer einfacher Datentyp unter JSON, welcher bereits erwähnt wurde, ist die Zeichenkette (String). Eine Zeichenkette wird durch doppelte Anführungszeichen umschlossen. In JavaScript können auch einfache Anführungszeichen zur Umschließung von Zeichenketten verwendet werden. JSON basiert zwar auf JavaScript Strukturen, dennoch sind nur doppelte Anführungszeichen bei der Umschließung von Zeichenketten erlaubt. Für die Darstellung von bestimmten Sonderzeichen in Zeichenketten muss ein Fluchtzeichen (Escape Sequence) verwendet werden. Das Fluchtzeichen ist ein umgekehrter Schrägstrich („\“) und wird vor dem jeweiligen Sonderzeichen eingesetzt.[17]

Für die Verwendung von Zahlen in JSON wird der Nummer-Datentyp verwendet. Ein Beispiel für einen solchen Datentyp ist der Ganzzahlwert „2018“ des Namens „jahr“ in der Zeile 5 im oben dargestellten JSON Text. Neben Ganzzahlen können auch Dezimalzahlen, negative Zahlen sowie Exponenten verwendet werden. Unter JSON ist es auch möglich, denn Datentyp „Boolean“ zu nutzen. Für die Verwendung dieses Datentyps muss einem Namen entweder „true“ oder „false“ als Wert zugeordnet werden. Die Zuweisung des Wertes „null“ auf einen Namen ist unter JSON ebenfalls möglich. Die-

15 Vgl. Bassett, S.10.
16 Vgl. Bassett, S.16.
17 Vgl. Bassett, S.18.

ser kann z. B. verwendet werden, wenn ein Name keinen Wert besitzt.[18]

Der letzte Datentyp unter JSON ist das Feld bzw. Array. Das Array ist eine Liste von Werten, welche einem Namen zugeordnet werden können. Die Werte selbst werden dabei von einem Komma getrennt und von eckigen Klammern umgeben. Ein besonders Merkmal von JSON Arrays ist die Möglichkeit, unterschiedliche Datentypen innerhalb eines Arrays zu verwenden. Die meisten Programmiersprachen unterstützen diese Funktionalität allerdings nicht, weshalb i. d. R. nur ein Datentyp pro Array verwendet wird.[19] Im unten dargestellten JSON Text ist ein Beispiel für ein Array mit dem Namen „kapitel“ dargestellt. Das Array ist von eckigen Klammern umgeben und enthält eine Liste von Zeichenketten („Kapitel 1“, „Kapitel 2“ usw.).

```
1     {
2        "kapitel": [
3           "Kapitel 1",
4           "Kapitel 2",
5           "Kapitel 3",
6           "Kapitel 4"
7        ]
8     }
```

2.2.4. Schema

Die einfache Verschachtelung und Erstellung von Paaren aus Namen und Werten ermöglicht eine flexible Strukturierung von Daten. Damit die Ordnung von Strukturen in einem JSON Dokument ersichtlich bleibt, sollte ein JSON Dokument hinsichtlich eines Schemas validiert werden. Ein Schema definiert dabei ein Grundgerüst einer Datenstruktur, welches den Aufbau eines JSON Dokuments festlegt. Für JSON wird gegenwärtig das JSON Schema entwickelt. Es befindet sich zum Zeitpunkt der Erstellung dieser Arbeit im Entwurf 7 (draft-07) auf der Plattform „json-schema.org“.[20] Die Verwendung von Schemata ist bereits von XML bekannt. XML nutzt die XML Schema Definition (XSD) zur Strukturierung von XML-Dokumenten.

Ein JSON Schema wird unter der Verwendung von JSON erstellt. Um eine Verwechselung mit einem JSON Datendokument auszuschließen, enthalten JSON Schemadokumente am Anfang des Textes den Namen „$schema“. Der Wert dieses Namens besteht aus einer Zeichenkette eines Links zur Plattform json-schema.org und dem jeweiligen Schema-Entwurf (z. B. „*http://json-schema.org/draft-06/schema#*“). Nach der Angabe eines Titels und einer Beschreibung des Schemas erfolgt danach die Festlegung der

18 Vgl. Bassett, S.19-20.
19 Vgl. Bassett, S.21
20 Siehe JSON-SCHEMA.ORG, URL: „http://json-schema.org/“ (Stand: 04.09.2018, 11:06)

Dokumentstrukturen. Zuerst werden i. d. R. die Namen festgelegt, welche in dem Dokument vorhanden sein sollen. Unter der Ebene „properties“ werden dazu Namensstrukturen angegeben, welche das Schema akzeptiert. Danach erfolgt die Festlegung der Datentypen sowie des Formats der jeweiligen Namen. Dafür stehen mehrere Optionen zur Verfügung. Zur Festlegung des Datentyps wird der Name „type“ verwendet. Der Wert dieses Namens ist eine Zeichenkette mit der Bezeichnung des Datentyps (z. B. „number“). Für das Format eines Wertes stehen mehrere Optionen zur Verfügung. Es können u. a. Maximal- und Minimalwerte festgelegt werden. Im Schema wird die Bezeichnung „required“ zur Festlegung der Namen verwendet, welche in einen zu validierenden JSON Dokument vorhanden sein müssen. Benötigte Namen aus der Ebene „properties“ werden dabei der Bezeichnung „required“ in einem Array zugeordnet.[21]

```
1     {
2        "$schema": "http://json-schema.org/draft-04/schema#",
3        "title": "Bachelorarbeit",
4        "description": "Angaben zu einer Bachelorarbeit",
5        "type": "object",
6        "properties": {
7           "titel": {
8              "type": "string",
9              "maxLength": 100
10          },
11          "autor": {
12             "type": "string",
13             "minLength": 4
14          },
15          "jahr": {
16             "type": "number",
17             "minimum": 2010
18          }
19       },
20       "required": [
21          "titel",
22          "autor",
23          "jahr"
24       ]
25    }
```

Im oben dargestellten JSON Text wird ein Beispielschema für ein JSON Dokument dargestellt, welches bibliografische Angaben einer Bachelorarbeit beschreibt. Die Namen „title“ und „description“ in den Zeilen 4 und 5 beschreiben, welchen Sachverhalt dieses Schema definiert. Unter dem Namen „properties“ in der Zeile 6 werden die Namen angegeben, welche das Schema kennt. Dazu werden die Namen „titel“, „autor“ und „jahr“ angegeben. Den Namen sind Objekte zugeordnet, welches Datenformate der zugeord-

21 Vgl. Bassett, S.31-33.

neten Werte festlegt. Der Name „titel“ in der Zeile 7 enthält z. B. die Elemente „type“ und „maxLength“. Unter „type“ wird der Datentyp des Wertes für den Namen „titel“ zugeordnet. Der Name akzeptiert durch die Angabe von „string“ nur Zeichenketten. Des Weiteren dürfen diese Zeichenketten durch die Angabe von „maxLength“ nur aus maximal 100 Zeichen bestehen. In der Zeile 20 des Schemas ist außerdem ein Array mit dem Namen „required“ vorhanden. Das Array gibt an, welche Namen in einem JSON Text vorhanden sein müssen, um im Schema gültig zu sein. Im oben dargestellten Schema enthält das Array Zeichenketten mit Namen aus dem „properties“ Element. Damit ein JSON Dokument hinsichtlich dieses Schemas validiert werden kann, muss es die Namen „titel“, “autor“ und „jahr“ besitzen. Im folgenden Beispiel wird ein Objekt dargestellt, welches im oben dargestellten Schema gültig ist.

```
1     {
2         "titel": "Untersuchungen zur Verbesserung...",
3         "autor": "Marcel Berger",
4         "jahr": 2018
5     }
```

Die Pflichtnamen „titel“, „autor“ und „jahr“ sind vollständig vorhanden. Der Wert das Namens „titel“ ist eine Zeichenkette, welche aus weniger als 100 Zeichen besteht. Die Werte der anderen Namen besitzen ebenfalls den richtigen Datentyp und das richtige Format.

2.2.5. Vergleich mit XML

Im folgenden Teil wird die Technologie JSON mit XML verglichen und es werden die Vor- und Nachteile dieses Datenaustauschformates gegenüber XML dargestellt. Ein Hauptunterschied sind die Einsatzgebiete beider Technologien. JSON wird hauptsächlich von Web-Anwendungen bei der Übertragung von Daten verwendet. XML ist dagegen schon länger etabliert und wird vor allem zur technischen Dokumentation verwendet. Die Bevorzugung von XML in diesem Bereich führt auf die kompaktere Formatierung von Text zurück, welche durch die Verwendung von Attributen in den XML-Tags ermöglicht wird. JSON ist im Vergleich zu XML keine Auszeichnungssprache, sondern eine reine Darstellung unformatierter Daten. Der Hauptvorteil von JSON liegt in der kompakten Syntax, da keine Tags verwendet werden. In XML werden i. d. R. vor und nach einem Datensatz Tags verwendet, was zu einem Anstieg von Zeichen in einem Dokument führt. Durch diesen Overhead steigt die Dateigröße mit zunehmender Menge und Komplexität von Daten schneller an. Das Datenvolumen ist bei Verwendung von JSON wesentlich geringer. Da keine Tags verwendet werden, steigt die Übersichtlichkeit bei der Strukturierung von einfachen Daten. In JSON gibt es allerdings keine Al-

ternative für die „<![CDATA]>“ Elemente. Diese Elemente werden von einem XML-Parser nicht verarbeitet, wodurch u. a. spezielle XML-Elemente oder Sonderzeichen einfacher dargestellt werden können. Da JSON auf JavaScript Elementen basiert, ist die Verwendung von JSON unter JavaScript allerdings einfacher als die Verwendung von XML. Ein Vorteil von XML ist hingegen die Möglichkeit zur Umwandlung von Dokumenten in verschiedene Formate durch XSLT. [22]

Auch in der Nutzung von Ressourcen gibt es Unterschiede zwischen XML und JSON. In einer Fallstudie der Montana State University wurden Übertragungsgeschwindigkeiten und Ressourcennutzung beider Technologien gemessen. Dazu wurde ein isoliertes lokales Netzwerk bestehend aus einem Client und einem Server aufgebaut. Der Client sendete verschlüsselte XML bzw. JSON Datensätze zu dem Server, welcher die Ressourcenauslastung erfasste. Dazu wurden zwei Szenerien erstellt.[23] Im ersten Szenario wurde eine große Anzahl von Objekten übertragen. Dazu wurden jeweils eine Million JSON bzw. XML Objekte übertragen und die Auslastung der Ressourcen sowie die Übertragungszeit wurden erfasst. Die Ergebnisse des ersten Szenarios werden in der Tabelle 1 dargestellt.

Messung der durchschnittlichen ...	**JSON**	**XML**
Übertragungsgeschwindigkeit pro Objekt	0,08 ms	4,55 ms
Auslastung der Nutzer-seitigen CPU	86,1 %	54,6 %
Auslastung der System-seitigen CPU	13,1 %	45,4 %
Auslastung des Arbeitsspeichers	27,4 %	29,7 %

Tabelle 1: gemessene Serverauslastung bei der Übertragung von 1 Mio. JSON bzw. XML Objekten.[24]

Dabei wurde festgestellt, dass in diesem Szenario die JSON Objekte wesentlich schneller übertragen werden als XML Objekte. Die durchschnittliche Auslastung der CPU liegt im ersten Szenario bei der Verarbeitung von JSON i. d. R. höher als bei Verarbeitung von XML. Die Auslastung des Arbeitsspeichers liegt nahezu im gleichen Bereich. Im Rahmen dieser Fallstudie wurde außerdem ein zweites Szenerio betrachtet. In fünf Versuchen wurden Übertragungen von 20000 bis 100000 JSON bzw. XML Objekten durchgeführt. Dabei wurde festgestellt, dass die durchschnittliche Übertragungsgeschwindigkeit von XML Objekten geringer ist, wenn die Anzahl der Objekte selbst gering bleibt. Bei JSON lag die durchschnittliche Übertragungsgeschwindigkeit in allen Szenarien gleich und.[25]

22 Vgl. Längle (2015): „XML vs. JSON“.
23 Vgl. Nurseitov et al. (o.D): „Comparison von JSON and XML Data Interchange Formats: A Case Study“, Abschnitt 4: „Methodology“.
24 Vgl. Nurseitov et al., Abschnitt 5.1 „Scenario 1“, Tabelle 1 und 2.
25 Vgl. Nurseitov et al., Abschnitt 5.3 „Discussion“.

Im Endeffekt hat JSON viele Vorteile gegenüber XML. Ein wichtiger Vorteil ist die kompaktere Syntax und die reinere Darstellung von Daten. Die Ergebnisse aus der Fallstudie zeigen, dass JSON bei der Übertragung von Objekten über ein Netzwerk große Geschwindigkeitsvorteile gegenüber XML besitzt. In mobilen Applikationen oder Webanwendungen ist JSON zur Einsparung von Ressourcen somit besser geeignet als XML. XML kann dafür Datenstrukturen komplexer darstellen.

2.3. RESTful Webservices

2.3.1. Einführung zu Webservices

Webservices dienen zur Interoperabilität zweier Maschinen in einem Computer-Netzwerk. Ein Web Service stellt dabei Funktionen zur Verfügung, welche von anderen Systemen durch eine Schnittstelle über das Netzwerk verwendet werden können.[26] Die meisten Webservices besitzen die Merkmale der Serviceorientierten Architektur (Abb. 1).[27] Die Serviceorientierte Architektur wird dabei als ein verteiltes System betrachtet.[28]

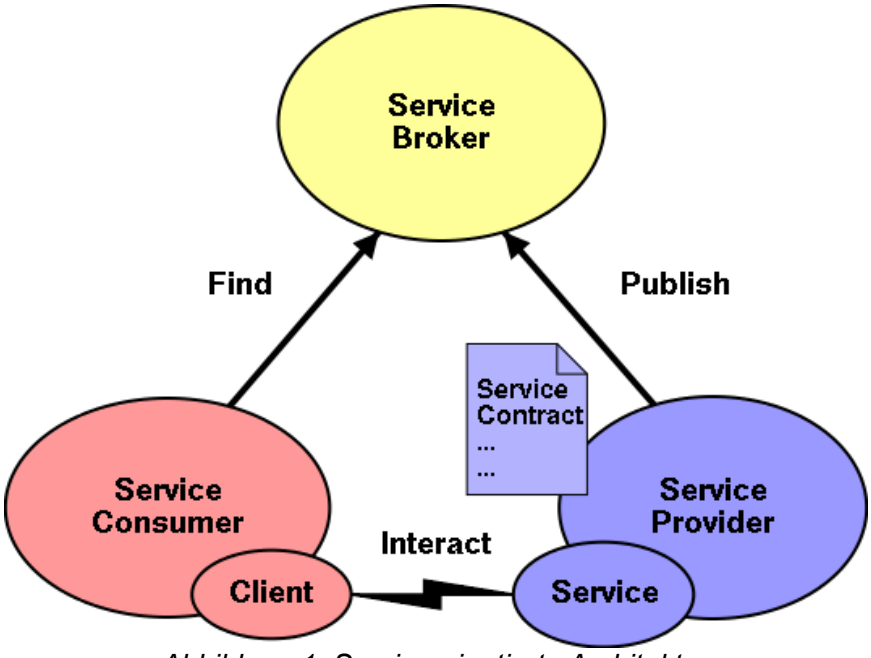


Abbildung 1: Serviceorientierte Architektur

Ein Hauptmerkmal besteht darin, dass ein Service dabei aus einer abstrahierten Sichtweise betrachtet wird. Der Service besteht dabei i. d. R. aus Anwendungen oder Ge-

26 Vgl. Booth, Haas et al. (2004): „Web Services Architecture“, S.7
27 Vgl. Booth, Haas et al., S.6.
28 Vgl. Booth, Haas et al., S.61.

schäftsprozessen, welche eine bestimmte Funktionalität zur Verfügung stellen. Weiterhin ist ein Service in dieser Architektur nachrichten-, beschreibungs- und netzwerkorientiert. Nachrichtenorientiertheit beschreibt in erster Linie den Austausch von Informationen durch Nachrichten („Messages“) zwischen dem Service Provider und dem Service Consumer (siehe Abbildung 1). Der interne Aufbau (z. B. Datenbankstruktur, Programmiersprache) eines Providers bzw. Consumers spielt dabei keine Rolle. Diese Plattformunabhängigkeit wird durch standardisierte Datenformate (vor allem XML) sichergestellt. Beschreibungsorientiertheit ist ein weiteres Merkmal der Architektur. Dabei werden maschinell verarbeitbare Metadaten dargestellt, welche wichtige Informationen für die Nutzung und Hintergründe des Services beinhalten.

Da die Komponenten in der Serviceorientierten Architektur i. d. R. über ein Computer-Netzwerk kooperieren, gilt das Merkmal der Netzwerkorientiertheit.[29] Der eigentliche Ablauf innerhalb der Serviceorientierten Architektur kann in 4 Schritten zusammengefasst werden. Im ersten Schritt beginnt die Interaktion zwischen dem Service Consumer und einem gewünschten Service Provider. Die Interaktion wird dabei durch den Service Consumer initiiert. Dieser muss dafür Informationen (z. B. Adresse) des Service Providers kennen, um die Verbindung herzustellen.[30] Will der Service Consumer ohne diese Informationen mit einem Service Provider interagieren, dann tritt ein Service zur Entdeckung (siehe Abbildung 1: „Service Broker“) eines geeigneten Providers ein. Unter der Nutzung der gestellten Kriterien des Service Consumers findet der Service Broker maschinell verarbeitbare Beschreibungen von Web Services sowie anzubietende Funktionen des Providers.[31] Der Broker agiert dabei entweder als Web Crawler oder als ein Register. Als Web Crawler sucht der Broker nach den Beschreibungen der Service Provider, ohne dabei direkt in Kontakt mit dem Provider zu treten. Agiert der Broker als ein Register, dann müssen Beschreibungen von Service Providern direkt in diesem Register veröffentlicht werden. Zur Implementierung eines solchen Registers wird z. B. die Technologie UDDI verwendet. Schließlich wählt der Broker passende Web Service Beschreibungen aus und teilt diese dem Service Consumer mit, welcher dann eine Beschreibung auswählt.[32]

Ist die Suche nach einem Web Service abgeschlossen, beginnt im zweiten Schritt des Ablaufs die Verständigung zwischen dem Consumer und dem Provider. Diese einigen sich auf die Servicebeschreibung, welche in einem WSDL Dokument vorgelegt ist (siehe Abbildung 1: „Service Contract“). Im dritten Schritt werden die Informationen der

29 Vgl. Booth, Haas et al., S.61.
30 Vgl. Booth, Haas et al., S.66.
31 Vgl. Booth, Haas et al., S.68.
32 Vgl. Booth, Haas et al., S.69 ff.

Servicebeschreibung innerhalb des Consumers bzw. des Providers als Input übergeben und eingesetzt. Im letzten Schritt tauschen der Service Consumer und der Service Provider SOAP Nachrichten aus.[33]

2.3.2. Geschichte von REST

Der Architekturstil REST wurde im Jahre 2000 von Roy Thomas Fielding im Rahmen seiner Dissertation dargestellt. Fielding war seit ca. 1994 in den Entwicklungsprozess der Standardisierung von HTTP eingebunden. Während der Weiterentwicklung auf HTTP 1.1 erstellte er einen Entwurf für ein einheitliches globales System zur Verarbeitung statischer und dynamischer Informationen. Dieses Modell wurde ursprünglich als „HTTP Object Model“ bezeichnet.[34] In seiner Dissertation wurde schließlich der „Representational State Transfer“ (REST) als ein Konzept für die Verwendungsmöglichkeit von HTTP vorgestellt. In der Umsetzung von Web Services wurden vor allem die Technologien WSDL und SOAP verwendet. Diese verwenden HTTP zwar ebenfalls, allerdings wird mit SOAP auch ein spezielles Protokoll verwendet. Der Architekturstil REST basiert hingegen nur auf den Konzepten von HTTP. Infolgedessen stieg die Beliebtheit von REST über die Jahre weiter an und neuere Web-APIs ohne SOAP wurden entwickelt.[35] In der Verwendung von Web Services mit REST trat dabei auch die Bezeichnung „RESTful“ auf. Dieser Begriff wird im Englischen als eine Bezeichnung für ein System verwendet, welches den Prinzipien von REST folgt.[36]

2.3.3. Grundprinzipien von REST

Ressourcen mit eindeutiger Identifikation, Hypermedia, die einheitliche Schnittstelle, unterschiedliche Repräsentationen und statuslose Kommunikation sind die fünf Grundprinzipien von REST.[37]

Ressourcen mit eindeutiger Identifikation: In REST besitzen Ressourcen bzw. Instanzen einer Anwendung eindeutige Identifikatoren. Diese Identifikatoren werden als URIs (Uniform Resource Identifier) bezeichnet und sind global eindeutig. Ein System mit REST besitzt i. d. R. Ressourcen, welche ein bestimmtes Datenobjekt darstellen und durch eine URI erhalten werden können. In der folgenden Liste sind Beispiele für URIs angegeben. In der Liste werden Schemata dargestellt, welche mögliche URIs für

33 Vgl. Booth, Haas et al., S.66.
34 Vgl. Tilkov et al. (2015): „REST und HTTP“, S.9.
35 Vgl. Tilkov et al., S.10.
36 Vgl. Kay (2007): „Representational State Transfer (REST)“.
37 Vgl. Tilkov et al., S.11.

abstrakte Ressourcen einer Beispielgruppe „Bachelorarbeit“ beschreiben. Hierarchien dieser Gruppe werden durch einen Schrägstrich getrennt. Die erste URI ist z. B. ein Identifikator für eine Ressource dieses Kapitels.[38]

- `http://bachelorarbeit.com/kapitel/2/3/3`
- `http://bachelorarbeit.com/seite/17`

Hypermedia: Das zweite Prinzip von REST ist das Konzept der Verknüpfungen, welches als Hypermedia bezeichnet wird. Diese Verknüpfungen werden bei REST von Applikationen für den Erhalt von weiteren Informationen bzw. Ressourcen verwendet. Die Verknüpfungen der Ressourcen basieren unter REST i. d. R. auf dem Schema der URIs. Da diese global eindeutig identifizierbar sind, können alle Ressourcen weltweit miteinander verknüpft werden. Einer der Hauptaufgaben von Verknüpfungen ist die Bereitstellung von Informationen über mögliche Aktionen in einem bestimmten Applikationszustand. Der Server teilt einem Client mit, welche Aktionen dieser in seinem jeweiligen Zustand ausführen kann. Dem Client werden dazu Verknüpfungen zu Ressourcen mitgeteilt, welche der Client zur Veränderung seines Applikationszustandes verwenden kann.[39] In Bezug auf das Beispiel „Bachelorarbeit“ kann ein Client z. B. die Ressource „/kapitel“ anfordern. Der Server würde dem Client die möglichen Verknüpfungen zu den Ressourcen der Unterebenen mitteilen (z. B. „/kapitel/2“). Diese Verknüpfungen könnte der Client dann zur Änderung seines Applikationszustandes benutzen.

Einheitliche Schnittstelle: Ein weiteres Prinzip von REST ist die einheitliche Schnittstelle, welche durch die Verwendung von HTTP Standardmethoden gewährleistet wird. Der Zugriff auf eine Ressource erfolgt dabei immer über Methoden wie z. B. GET, POST, PUT und DELETE.[40] Diese und andere Methoden werden im Kapitel 2.3.4 genauer dargestellt.

Ressourcen und Repräsentationen: Das Prinzip der Ressourcen und Repräsentationen beschreibt die Bereitstellung mehrerer Repräsentationsformate von Daten einer Ressource. Dazu wird das Verfahren der HTTP Content Negotiation verwendet. In diesem Verfahren können bei Aufruf von Standardmethoden bestimmte Anforderungen gestellt werden. Ein Client kann bei z. B. bei Aufruf der GET Methode ein bestimmtes Datenformat verlangen. Ressourcen, welche mehrere Repräsentationsformate bereitstellen, können mit einer Vielzahl von Clients mit unterschiedlichen Anforderungen interagieren.[41]

38 Vgl. Tilkov et al., S.12.
39 Vgl. Tilkov et al., S.13.
40 Vgl. Tilkov et al., S.15.
41 Vgl. Tilkov et al., S.17.

Statuslose Kommunikation: Das letzte Prinzip von REST ist die statuslose Kommunikation. Bei der statuslosen Kommunikation wird der Sitzungsstatus zwischen dem Server und dem Client abgeschafft. Die für eine Sitzung relevanten Zustände werden komplett auf der Seite des Clients verarbeitet. Bei einer Anfrage an den Server muss der Client alle wichtigen Informationen mit übergeben, da der Server keine Informationen aus vorherigen Anfragen des Clients speichert. Jede Anfrage wird von einer neuen Serverinstanz bearbeitet. Dadurch wird die Bindung zwischen dem Client und den Server verringert und die Skalierbarkeit vereinfacht.[42]

2.3.4. HTTP Standardmethoden

Unter REST wird eine Reihe von HTTP Standardmethoden als Schnittstelle zu Ressourcen verwendet. Für REST sind dabei 8 Methoden relevant. Im vorherigen Kapitel wurden bereits die Methoden GET, POST, PUT und DELETE erwähnt. Weitere Methoden sind HEAD, OPTIONS, TRACE und CONNECT. Die wichtigste Methode ist GET.[43]

GET: Mit GET können Informationen einer Ressource abgefragt werden. Mit einem bedingten GET können ebenfalls Bedingungen gestellt werden. Es kann z. B. eine Bedingung über den Verzicht der Rückgabe einer Ressource gestellt werden, wenn diese seit einem bestimmten Datum nicht mehr aktualisiert wurde. Dazu gibt der Client ein Datum im „If-Modified-Since“ Header der GET Anfrage an. Wurde die Ressource seit diesem Datum nicht mehr verändert, dann gibt der Server die Meldung 304 „Not Modified“ zurück. Durch die Angaben von Bedienungen wird der Datenverkehr somit optimiert und die Zahl der überflüssigen Anfragen wird minimiert.

HEAD: Die Methode HEAD ähnelt der Methode GET. Mit HEAD werden allerdings nur die Header angefragt und keine Repräsentationen einer Ressource. Die Header enthalten die Metadaten einer Ressource. Die Methode kann von einem Client z. B. zur Abfrage der Größe oder der Existenz einer Ressource verwendet werden, ohne dabei ein Repräsentationsformat der Ressource empfangen zu müssen.[44] GET und HEAD dienen zum lesenden Zugriff auf eine Ressource. Für den schreibenden Zugriff auf eine Ressource werden die Methoden PUT, POST und DELETE verwendet.

DELETE: Durch die Verwendung von DELETE können Ressourcen entfernt werden.

PUT: Mit der Methode PUT können Ressourcen modifiziert werden. Sollte eine Ressource nicht existieren, dann wird diese nach Aufruf von PUT erstellt.

42 Vgl. Tilkov et al., S.18-19.
43 Vgl. Tilkov, et al., S.53
44 Vgl. Tilkov, et al., S.54

POST: Auch die Methode POST kann zur Erstellung von Ressourcen benutzt werden. Der Unterschied zur Methode PUT liegt bei der Angabe der URI. Bei PUT zeigt die URI direkt auf die Ressource, welche modifiziert bzw. erstellt wird. Unter POST wird die URI einer Verwaltungsressource angegeben, welche für die Erstellung von Ressourcen zuständig ist.[45]

Im folgenden Beispiel wird der Unterschied zwischen PUT und POST am Bachelorarbeit-Beispiel dargestellt. Mit PUT wird die URI der Ressource „/seite/18“ direkt verändert bzw. erstellt. Mit POST wird die URI für die Erstellung zuständige Ressource „/seite“ aufgerufen.

- `PUT /seiten/19`
- `POST /seiten`

Ein weiterer Unterschied zwischen PUT und POST ist die Eigenschaft der Idempotenz der Methode PUT. Eine Methode wird als idempotent bezeichnet, wenn das Ergebnis des Aufrufs immer gleich ist. Dabei spielt es keine Rolle, ob die Methode einmal oder mehrfach aufgerufen wird. Zu den idempotenten Methoden gehören ebenfalls GET, HEAD, OPTIONS und DELETE.[46] Die Methoden GET, OPTIONS und HEAD gehören außerdem zu den „sicheren Methoden“. Eine Methode wird als sicher bezeichnet, wenn diese nur lesenden Zugriff ausübt und den Zustand einer Ressource nicht verändert.[47]

OPTIONS: Mit der Methode OPTIONS können Metadaten über Methoden einer Ressource angefordert werden. Diese Metadaten umfassen Informationen über mögliche Methoden einer Ressource, welche ein Client verwenden könnte.[48]

2.3.5. Caching

Im Kapitel 2.3.4 wurde bereits das bedingte GET erwähnt. Diese Methode ist eine wichtige Methode für die Implementierung von Cache in der REST Architektur. Der Cache dient zur Reduzierung des Datenverkehrs zwischen Server und Client. Für das Caching steht dem Client und dem Server das „Cache-Control“ Element zur Verfügung. Dieses Element kann im Header eines HTTP Response durch den Server verwendet werden. Der Client verwendet dieses Element im Header des HTTP-Requests. Bei der Antwort des Servers stehen für „Cache-Control“ eine Reihe von Anweisungen zur Ver-

45 Vgl. Tilkov, et al., S.56-57
46 Vgl. Tilkov, et al., S.59.
47 Vgl. Tilkov, et al., S.55.
48 Vgl. Tilkov, et al., S.58.

fügung. Es kann z. B. durch die Zuweisungen einer Ganzzahl auf das Element „max-age“ die zeitliche Gültigkeit einer Server-Antwort in Sekunden angegeben werden. Durch die Angabe von „no-cache“ wird die Cache-Fähigkeit einer Ressource ausgeschlossen.[49] Der Client kann ebenfalls Anweisungen für den „Cache-Control“ verwenden. In der Anweisung „min-fresh“ kann z. B. eine Zeit in Sekunden angegeben werden. Diese Zeit gibt an, wie lange eine Server-Antwort noch gültig sein muss. Durch die Angabe von „max-stale“ akzeptiert der Client auch Server-Antworten mit zeitlich abgelaufener Gültigkeit. Zur Überprüfung der Gültigkeit werden „ETags“ verwendet. Die ETags bestehen aus einer Zeichenkette, welche zur Identifikation von Zuständen einer Ressource dienen und werden in den HTTP Requests bzw. in den HTTP Respones verwendet. Durch den Vergleich zweier Zeichenketten eines ETags können unterschiedliche Zustände von Ressourcen identifiziert werden.[50]

49 Vgl. Tilkov, et al., S.134
50 Vgl. Tilkov, et al., S.135.

3. Entwicklung des neuen Praxisbeispiels

Im Zuge der Modernisierung des Lehrmoduls Datenrepräsentation wird für die Praktika ein neues Praxisbeispiel mit der Bezeichnung „Elektronikrechnungen“ entwickelt. Gegenwärtig werden in den Praktika hauptsächlich 3 verschiedene Praxisbeispiele zur Verdeutlichung von Funktionen einer Datenrepräsentationstechnologie verwendet. Ein Praxisbeispiel stellt dabei einen Sachverhalt aus realem Umfeld dar und repräsentiert diesen durch die Verwendung von Beispieldaten. Die im XML Format dargestellten Beispieldaten sollen dabei für die Teilnehmer eines Praktikums nachvollziehbar und leicht verständlich sein. Die Beispieldaten werden während der Durchführung eines Praktikums verwendet. Ein Praktikumsteilnehmer implementiert dabei z. B. ein Java Programm gemäß der Praktikumsanleitung. Zur Überprüfung Java Programms werden dann Beispieldaten als Eingabewerte für den Test der Funktionalität verwendet. Da die bisherigen Praxisbeispiele nicht mehr zeitgemäß sind, werden diese hinsichtlich der Verständlichkeit und der Komplexität neu bewertet werden. Auf Basis dieser Bewertung erfolgt die Entwicklung des neuen Praxisbeispiels mit der Bezeichnung „Elektronikrechnungen“.

3.1. Gegenwärtig verwendete Praxisbeispiele

3.1.1. Beschreibung

Firmenadressen: Das erste Beispiel beschäftigt sich mit dem Thema „Firmenadressen“. Das Beispiel besteht aus Datensätzen, welche Adressen und Informationen von Firmen aus Deutschland beinhalten. Ein Datensatz besitzt eine Identifikationsnummer, eine Zeichenkette für den Firmennamen, Adressen, eine Telefonnummer sowie die Anzahl der Mitarbeiter. Der Aufbau eines einzelnen Datensatzes dieses Praxisbeispiels wird in der Abbildung 2 auf der linken Seite (Klasse „Firmenadresse“) dargestellt.

Biblio: Das zweite Praxisbeispiel trägt die Bezeichnung „Biblio“. Die Komplexität dieses Themas ist von allen drei Beispielthemen am geringsten. Das Beispiel stellt eine Liste mit Angaben zu Büchern dar. Ein einzelner Datensatz besteht dabei aus einer Identifikationsnummer, einem Kürzel, dem Namen des Autors sowie Titel und Verlag des Buches. Der Aufbau eines einzelnen Datensatzes dieses Praxisbeispiels wird in der Abbildung 2 auf der rechten Seite (Klasse „Buch“) dargestellt.

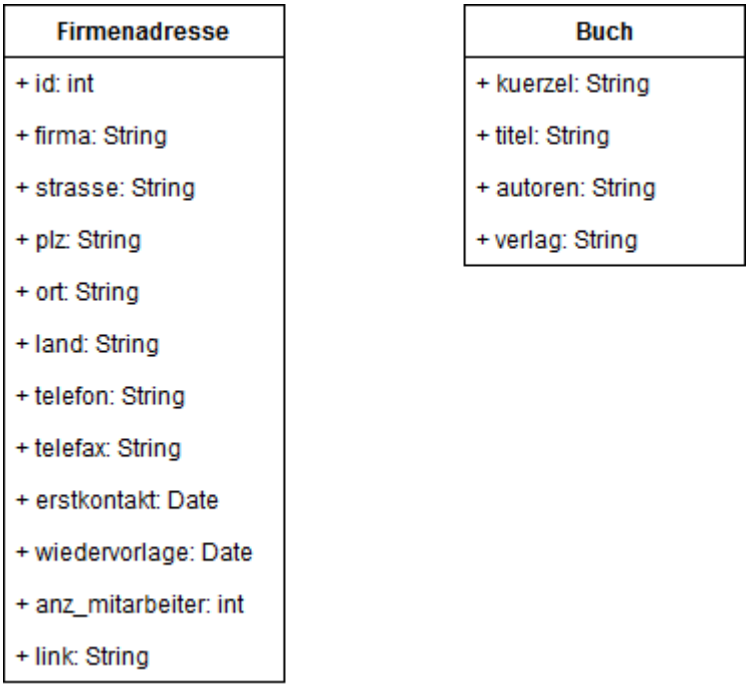


Abbildung 2: Daten der Praxisbeispiele „Firmenadressen“ (links) und „Biblio“ (rechts)

Messberichte: Das letzte Praxisbeispiel trägt die Bezeichnung „Messberichte“ und ist deutlich komplexer aufgebaut als die Praxisbeispiele „Firmenadressen“ und „Biblio“. Das Beispiel enthält Datensätze über Messwerte von Wareneingangsprüfungen eines fiktionalen Produktionsbetriebes. Diese Datensätze werden schließlich einen Beispielmessbericht über die Prüfung eines Artikels bilden. Die Daten des Themas „Messbericht“ werden in die Gruppen Kopfdaten, Messwertdaten und Prüfergebnisse unterteilt. In den Kopfdaten sind die Elemente „Artikeldaten“, „Lieferentendaten“ und „Prüferdaten“ enthalten. Die Gruppe der Messwertdaten enthält Angaben zu den Prüfmerkmalen. Bei den Prüfmerkmalen werden die Messwerte sowie berechnete Mittelwert von fiktionalen Wareneingangsprüfungen angegeben. Das letzte Element enthält Angaben zu den Prüfergebnissen der Wareneingangsprüfungen. Neben einer Einstufung der Ergebnisse kann optional eine Bemerkung und ein Termin für die Nachbearbeitung angegeben werden.[51] In der 3. Abbildung wird das soeben beschriebene Praxisbeispiel grob dargestellt.

51 Vgl. Schubert: „Praktikum 02 – Erstellen und Testen von DTDs zum Validieren von XML-Dokumenten am Beispiel eines Messwert-Prüfberichtes“, S.1-2.

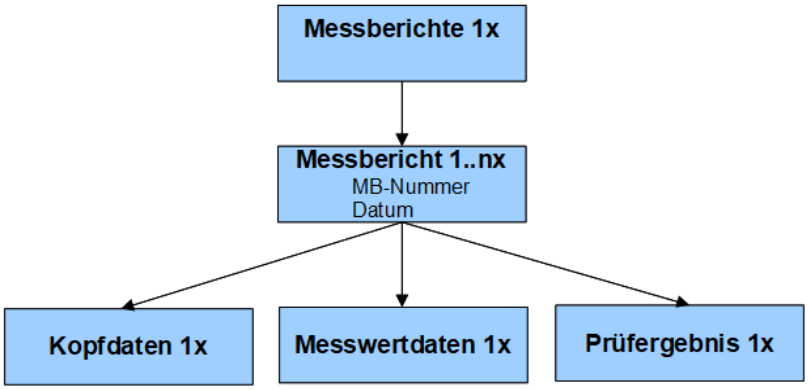


Abbildung 3: Das Praxisbeispiel „Messbericht“

3.1.2. Bewertung

Damit die Verständlichkeit eines Praxisbeispiels gewährleistet werden kann, sollte ein Bezug zum alltäglichen Umfeld eines Praktikumsteilnehmers hergestellt werden. Bei den bisher verwendeten Praxisbeispielen ist dieser Bezug nicht immer vorhanden. Das Praxisbeispiel „Messberichte“ befindet sich z. B. im Umfeld eines Produktionsbetriebes. Beispiele aus Betrieben sind i. d. R. sehr spezifisch und nicht alltäglich. Der fehlende Bezug zu diesem Thema könnte für Praktikumsteilnehmer zu Verständnisproblemen führen. Dazu tragen auch die Werte der Beispieldatensätze bei. Messwerte und firmenspezifische Daten sind durch den fehlenden Bezug zum Alltag nicht immer nachvollziehbar. Des Weiteren ist das Praxisbeispiel durch die Bezeichnung „Wareneingangsprüfung“ sehr abstrakt gewählt, wodurch die Nachvollziehbarkeit weiter verringert wird. Die Praxisbeispiele „Biblio“ und „Firmenadressen“ besitzen im Vergleich dazu einen besseren Bezug zum alltäglichen Umfeld. Sämtliche Datensätze des Beispiels „Firmenadressen“ beinhalten z. B. einfache Firmen- oder Städtenamen, welche für Praktikumsteilnehmer leicht verständlich sind. Die Datenstrukturen dieser Praxisbeispiele sind im Vergleich zum Thema „Messberichte“ sehr übersichtlich gestaltet. Ein Datensatz besitzt keine Abhängigkeiten zu anderen Daten. Die Firmenadressen werden z. B. nur aufgelistet und sind unabhängig von anderen Datensätzen. Die Komplexität der Datenstrukturen dieser Praxisbeispiele ist somit nicht immer ausreichend geeignet, um die gesamte Funktionalität einer Technologie darzustellen. Im Hinblick auf die Erstellung neuer Praktikumsinhalte ist ein komplexes Thema vor allem für das Praktikum „RESTful Webservices“ notwendig. Die Verwendung von einfachen Datenstrukturen würde z. B. die Erklärung der Funktionalität von Hypermedia wesentlich komplizierter machen, da keine wirklichen Verknüpfungen zwischen den Daten vorhanden sind.

Im Endeffekt besitzen die derzeitigen Praxisbeispiele der Praktika entweder keinen Bezug für Praktikumsteilnehmer zum alltäglichen Umfeld oder die Komplexität der Datenstrukturen ist im Hinblick auf die Erstellung neuer Praktikumsinhalte nicht ausreichend. Diese Nachteile der derzeitigen Praxisbeispiele müssen während des Entwurfs des neuen Hauptbeispiels berücksichtigt und behoben werden.

3.2. Das Praxisbeispiel „Elektronikrechnungen“

3.2.1. Entwurf

Für das neue Praxisbeispiel „Elektronikrechnungen“ wird eine Liste von fiktiven Rechnungselementen entworfen. Ein einzelnes Rechnungselement soll dabei Details über den Einkauf von Elektronikgeräten bei einem Elektronikfachhändler enthalten. In der unten dargestellten Abbildung ist die Datenstruktur eines Rechnungselementes dargestellt. Ein Rechnungselement besteht demnach aus einer Identifikationsnummer, eine Angabe für das Rechnungsdatum, einer Liste von gekauften Elektronikgeräten sowie Angaben zur Anschrift des Elektronikfachhändlers. Das Format dieser Daten ist fest definiert. Die Identifikationsnummer besteht aus einer dreistelligen Zeichenkette, welche in der gesamten Rechnungsliste zur Identifikation eines einzelnen Rechnungselementes verwendet wird. Die Zeichen der Identifikationsnummer bestehen dabei nur aus Zahlen zwischen 0 und 9.

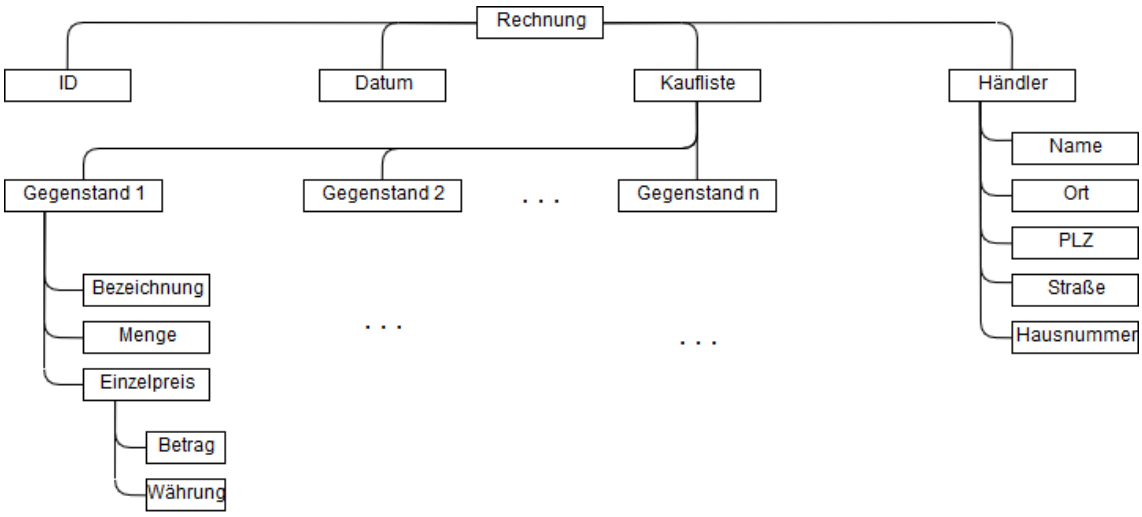


Abbildung 4: Struktur des Praxisbeispiels „Elektronikrechnungen“

Im Element „Datum“ wird ein Rechnungsdatum angegeben, welches im von der ISO 8601[52] spezifizierten „YYYY-MM-DD“-Format dargestellt wird. Das Element „Kaufliste“ enthält mehrere Unterelemente für die Strukturierung von Daten gekaufter Elektronikgeräte. Die Unterelemente werden in der oben dargestellten Abbildung als „Gegenstand“ bezeichnet. In der Kaufliste können beliebig viele Gegenstände angegeben werden, es muss aber mindestens ein Gegenstand enthalten sein. Für einen einzelnen Gegenstand wird schließlich die Bezeichnung, die Menge und der Einzelpreis angegeben. Die Bezeichnung besteht aus einer Zeichenkette und besteht aus Angaben wie z. B. Herstellername oder Produktname. Das Element „Menge“ besteht aus einer positiven Ganzzahl und gibt an, in welcher Anzahl der Gegenstand gekauft wurde. Das letzte Element enthält Information zum Einzelpreis des jeweiligen Gegenstands. Die Kosten eines Gegenstandes werden durch eine nicht-negative Dezimalzahl im Unterelement „Betrag“ angegeben. Zusätzlich wird bei der Angabe des Einzelpreises eine Enumeration für Währungen vorgegeben. In der Enumeration werden die Zeichenketten „Euro“, „USD“ und „GBP“ definitiv enthalten sein. Die Datenstruktur für die Bezeichnung, die Menge und den Einzelpreis sind für jeden einzelnen Gegenstand der „Kaufliste“ identisch. Das letzte Unterelement einer Rechnung enthält Informationen über den Elektronikfachhändler. Für jedes Rechnungselement wird immer exakt ein Händler angegeben. Die Informationen eines Händlers umfassen den Namen des Händlers sowie den Ortsnamen, die Postleitzahl, die Straße und die Hausnummer einer Filiale bzw. des Hauptsitzes des Händlers. Für die Darstellung von Händlerinformationen wird in allen Fällen eine Zeichenkette verwendet.

Das neue Praxisbeispiel „Elektronikrechnungen“ wird für den Ersatz der bisherigen Beispielthemen gewählt, da es einen besseren Bezug zum täglichen Umfeld für die Teilnehmenden der Praktika besitzt. Es ist anzunehmen, dass sich Teilnehmende der Praktika auch im täglichen Umfeld mit Rechnungen beschäftigen und somit bereits eine bestimmte Vorstellung über die Datenstrukturen von Rechnungen haben. Die Inhalte von Rechnungen des Praxisbeispiels werden auf Elektronikgeräte eingegrenzt, da Teilnehmende der Praktika hauptsächlich aus dem Umfeld der Informatik kommen und somit i. d. R. Kenntnisse über Elektronikgeräte, wie z. B. Computerzubehör, besitzen.

3.2.2. Umsetzung

Im Kapitel 3.2.1 wurde die Datenstruktur eines Rechnungselements erläutert. Damit bei der technischen Umsetzung dieser Struktur gültige Beispieldatensätze entstehen, müs-

52 Siehe Wolf et al. (1997): „Date and Time Formats“, Abschnitt „Formats“.

sen genaue Schemata für die Datenrepräsentationsformate JSON und XML angelegt werden. Neben der Datenstruktur definieren die Schemata auch Datentypen und Dimensionen einzelner Elemente. Für die Validierung der Identifikationsnummer einer Rechnung wird z. B. ein regulärer Ausdruck verwendet. Im JSON Schema (siehe Anlage 1) wird ein gültiges Muster der Zeichenkette „id“ durch eine entsprechende Festlegung im Feld „pattern“ definiert. In der XML Schema Definition (siehe Anlage 2) wird die Festlegung des Formats der Identifikationsnummer ähnlich definiert. Ein Hauptunterschied zwischen dem JSON Schema und der XML Schema Definition entsteht bei der Definition des Datum-Elements. Bei XSD ist es möglich, den Datentyp „Date“ für die Definition von Datumsangaben zu verwenden. Bei JSON ist dies nicht möglich, da JSON keine komplexen Datentypen verwendet. Für die Validierung der Datumsangabe wird im JSON Schema daher eine Zeichenkette und ein regulärer Ausdruck verwendet. Eine Datumsangabe im JSON Format ist im Rechnungsbeispiel somit nur gültig, wenn diese als Zeichenkette dargestellt und im „YYYY-MM-DD“ Format angegeben wird. Ein weiterer signifikanter Unterschied liegt bei Validierung der Eindeutigkeit von Identifikationsnummern. Bei der XML Schema Definition wird dazu das Element „xsd:unique“ verwendet. Dabei werden die Zeichenketten der Identifikationsnummern mithilfe von XPath abgefragt und auf Eindeutigkeit überprüft. Wenn in einem Datensatz Übereinstimmungen zwischen den Identifikationsnummern von Rechnungselementen existieren, dann ist dieser nicht gültig. Für das JSON Schema steht im gegenwärtigen Standard keine Möglichkeit zur Validierung der Eindeutigkeit einzelner Daten in einem Dokument zur Verfügung. Es ist allerdings möglich, die Inhalte eines JSON Arrays auf Eindeutigkeit zu überprüfen. Dafür kann das Feld „uniqueItems“ verwendet werden. Da Rechnungselemente im JSON Schema in einem Array strukturiert werden, können somit zumindest exakt gleiche Rechnungsobjekte erkannt werden. Die Schemata werden von Teilnehmenden der Praktika zur Überprüfung der Gültigkeit von vorgegeben oder selbsterstellten Beispieldaten verwendet, um Fehler bei der Verarbeitung der Daten zu vermeiden.

4. Verbesserung bestehender Praktika

Dieses Kapitel behandelt bestehende Praktikumsinhalte des Lehrmoduls Datenrepräsentation. Dabei wird zuerst ein Überblick über die bestehenden Praktika geschaffen. Die Praktika besitzen dabei Praktikumsanleitungen, Vorlagen, Beispiele und sonstige für relevante Dokumente. Nachdem ein Überblick geschaffen wurde, werden ausgewählte Praktika untersucht. Ein Schwerpunkt der Untersuchung liegt dabei auf Strukturen der Praktika sowie den Praktikumsanleitungen. Bei der Untersuchung stehen aufgrund der Aufgabenstellung vor allem die Praktika zu den Technologien XML-SAX und Webservices im Vordergrund. Nach den Untersuchungen werden die Verbesserungsmöglichkeiten für die Praktika erläutert, welche im Schlussteil des Kapitels realisiert werden.

4.1. Organisation von Praktikumsvorgaben

In diesem Teil des Kapitels wird ein Überblick über die Organisation von Vorgaben der Praktika geschaffen. Vorgaben umfassen dabei sämtliche Dateien, die für die Praktika des Lehrmoduls Datenrepräsentation zur Verfügung stehen. Ein Hauptteil der Vorgaben befindet sich dabei im Intranet der Hochschule Mittweida auf den Seiten zum Thema „Datenrepräsentation mit XML“. Die Vorgaben im Intranet sind thematisch untergliedert.

Praktikum	wird behandelt
PR01 – Wohlgeformte und validierte XML-Dokumente	
PR02 – XML-Messbericht erstellen und mit DTD validieren	
PR03 – XML-Messbericht mit XML-Schema validieren	
PR04 – Programmierschnittstelle XML-DOM	
PR05 – Programmierschnittstelle XML-SAX	**X**
PR06 – Transformation mit XSLT, Navigieren mit XPath	
PR07 – Transformation mit XSL-FO, PDF erzeugen	
PR08 – Zusatzpraktikum: XML-Webservices	**X**

Tabelle 2: Übersicht über zu verbessernde Praktika

Für jede im Lehrmodul behandelte Technologie steht eine einzelne Sektion zur Verfügung. In den einzelnen Sektionen befinden sich schließlich Links für den Download von

jeweiligen Lehrmaterialien und es befinden sich außerdem die Anleitungen und Vorlagen für die Praktikumsveranstaltungen. Die Vorlagen umfassen verpackte Dateien, welche z. B. Beispieldaten im XML Format, Java Dateien oder Softwarebibliotheken beinhalten. In der Tabelle 2 wird dargestellt, welche Praktikumseinheiten im Rahmen dieser Arbeit untersucht und behandelt werden. Als Referenz wurde das Wintersemester 2016/2017 verwendet.

4.2. Praktikum XML-SAX

4.2.1. Überblick

Das Praktikum XML-SAX beschäftigt sich mit der Erstellung einer Konsolen-Anwendung sowie einer grafischen Anwendung in Java. Beide Anwendungen beinhalten dabei Funktionen zum Einlesen und Anzeigen von Inhalten einer XML Datei unter der Verwendung eines SAX Parsers. Für die Bearbeitung des Praktikums XML-SAX werden die Praktikumsanleitungen und Vorgaben des Praktikums „PR05“ aus dem Wintersemester 2017/2018 verwendet. Es stehen die originalen Praktikumsvorgaben sowie eine Musterlösung zur Verfügung. Die originalen Praktikumsvorgaben sind in der Abbildung 5 dargestellt. Bei der PDF-Datei „Fach_XML_PR06“ handelt es sich um die Praktikumsanleitung. Das Praktikum verwendet die Praxisbeispiele „Biblio“ und „Firmenadressen“. Die XML Dateien beinhalten Beispieldaten der jeweiligen Themen und die XSD Dateien beinhalten Beschreibungen für die Schemata der jeweiligen XML Dateien. Die Java Datei „MainView“ implementiert eine grafische Anzeige unter der Verwendung von Java-Swing Elementen, welche im Praktikum als Vorlage für die Erstellung der grafischen Anwendung verwendet. Zur Implementierung von Funktionen des SAX Parsers wird die Softwarebibliothek „Xerces“ von Apache verwendet.

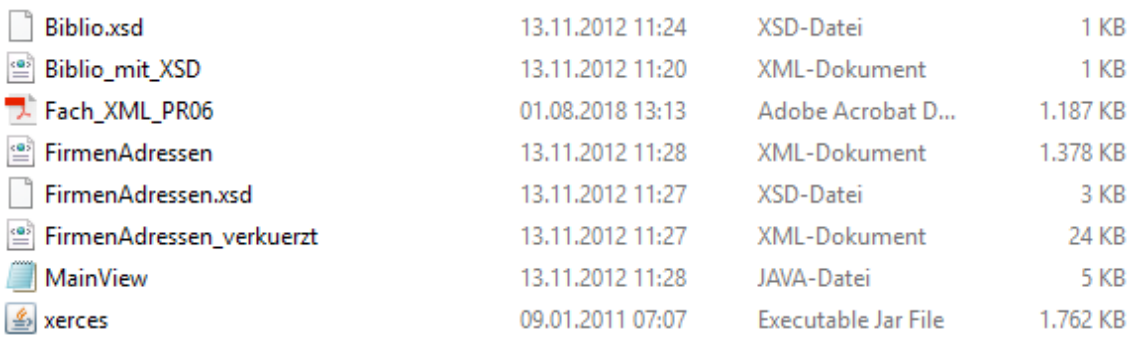


Abbildung 5: Praktikumsvorgaben XML-SAX

4.2.2. Ablauf

Das Praktikum ist in zwei Teilaufgaben untergliedert. In der ersten Aufgabe wird eine Konsolen-Anwendung in Java erstellt. Die Anwendung soll Inhalte einer XML Datei mit Funktionen von SAX einlesen und anzeigen. Für die Aufgabe wird eine XML Datei mit Beispieldaten des Themas „Biblio“ verwendet. Beide Teilaufgaben werden dabei unter der Verwendung der Entwicklungsumgebung Eclipse bearbeitet. In Eclipse wird zuerst ein neues Projekt angelegt. Die benötigte Softwarebibliothek „Xerces“ wird in das Projekt eingebunden. Danach werden grundlegende Funktionen des SAX Parsers anhand von Quellcode erklärt. Zu Beginn wird der Prozess des Einlesens anhand der Java-Klasse „SimpleSaxReader“ erstellt. Danach erfolgt eine Anleitung über die Implementierung des Content- und Error-Handlers. Nach der Implementierung der Funktionen werden diese getestet. Dazu werden Inhalte der Datei „Biblio_mit_XSD.xml“ (siehe Abbildung 5) eingelesen. Bei erfolgreicher Durchführung der Aufgabe werden entsprechende Inhalte der XML-Datei in der Konsole angezeigt.

Die zweite Teilaufgabe des Praktikums beschäftigt sich mit der Erstellung einer grafischen Anwendung in Java. Inhalte einer XML-Datei werden durch Funktionen des SAX Readers und durch die Verwendung von Java-Swing grafisch dargestellt. Als Vorlage für die grafische Ansicht wird dazu die Java-Klasse „MainView“ (siehe Abbildung 5) verwendet. Die Klasse erzeugt ein Fenster mit einem Grundgerüst für die Anzeige von Inhalten einer XML Datei. Dabei werden speziell die Daten des Praxisbeispiels „Firmenadressen“ angezeigt. Als Vorgabe für die Beispieldaten wird die Datei „Firmenadressen.xml“ (siehe Abbildung 5) einbezogen. Für die Bearbeitung der Teilaufgabe wird das Eclipse Projekt der ersten Teilaufgabe weiter verwendet und um eine Reihe von Java-Klassen erweitert. Während der Durchführung des Praktikums werden u. a. Klassen für die Erzeugung des JTable-Modells sowie Funktionen für das Hinzufügen von Tabellenzeilen erzeugt. Schließlich wird ein Content-Handler für das Füllen der Tabelle implementiert. Zum Schluss der Praktikumsdurchführung wird zusätzlich noch eine Funktion für das Filtern von Inhalten erstellt. Nach erfolgreicher Beendigung der Teilaufgabe steht eine grafische Anwendung für die Anzeige von Beispieldaten des Themas „Firmenadressen“ in einer JTable zur Verfügung.

4.2.3. Änderungen und Verbesserungen

Die Änderung der grafische Benutzeroberfläche von Java-Swing auf JavaFX steht bei der Verbesserung des Praktikums XML-SAX im Vordergrund. Für die Änderung der Oberfläche müssen alle Swing-Elemente in der vorgegebenen Java Datei „MainView“

vollständig durch JavaFX Elemente ersetzt werden. Bei der Praktikumsdurchführung werden Java Klassen erstellt, welche teilweise auf Swing-Elemente zugreifen. Dabei sind vor allem die Klassen „AdrTableModel“, „AdrRow“ und „AdrContentHandler“ betroffen. Im folgenden Teil werden Veränderungen in den einzelnen Java Klassen erläutert.

ProgramMain: Die Klasse „ProgramMain“ enthält die Main-Methode zum Starten der Anwendung. Die Main-Methode führt den SAX Parser aus und öffnet die grafische Java-Swing-Benutzeroberfläche. Für die Verwendung von JavaFX muss die Klasse durch javafx.application.Application erweitert werden und die Methode „start()“ der Elternklasse implementieren. In der Methode „start()“ wird dann der SAX Parser und die JavaFX Benutzeroberfläche gestartet.

MainView: In der Klasse „MainView“ müssen alle Swing-Elemente durch JavaFX Elemente ersetzt werden. Die meisten Swing-Elemente können direkt durch ein äquivalentes JavaFX Objekt ersetzt werden. Für nicht direkt ersetzbare Objekte muss eine gleichwertige Lösung implementiert werden. Bisher vererbt die Klasse „MainView“ von der Klasse „JFrame“. In der JavaFX Anwendung wird die Klasse „MainView“ von der Klasse „javafx.stage.Stage“ vererben. „MainView“ enthält außerdem eine Reihe von Get-Methoden, welche Swing-Objekte zurückgeben. Auf Get-Methoden zugreifende Klassen müssen dementsprechend angepasst werden. Den größten Änderungsaufwand erzeugt die Umstellung der Java-Swing JTable auf die JavaFX TableView, da die Implementierung der TableView grundlegende Unterschiede hat.

AdrRow: Die Klasse „AdrRow“ repräsentiert eine einzelne Zeile der JTable und spielt bei in der Implementierung des SAX ContentHandlers eine Rolle. Wird im ContentHandler z. B. ein neues Element einer XML-Datei gestartet, dann wird eine Instanz dieser Klasse erzeugt. Die durch den ContentHandler eingelesenen Daten werden schließlich an die Variablen dieser Klasse übergeben. Die Klasse wird in der umgestellten Anwendung nicht mehr benötigt, da für die Elektronikrechnungen ein anderes Datenmodell verwendet wird, mit welchem auch das Einlesen von Daten im SAX ContentHandler ermöglicht wird.

AdrTableModel: Die Klasse „AdrTableModel“ beschreibt den Aufbau bzw. das Model der JTable. Für die Anzeige von Datensätzen in einer JavaFX TableView wird zunächst ein vollständiges Datenmodell für Elektronikrechnungen benötigt. Die Klasse AdrTableModel wird schließlich als Adapterklasse des Datenmodells für die TableView verwendet. Die Verwendung dieser Klasse als Adapter ist notwendig, da für die standardmäßige Implementierung der JavaFX TableView i. d. R. Properties Objekte anstatt einfacher Datentypen (String, Integer usw.) verwendet werden.[53] Java-Objekte des

53 Vgl. ORACLE.COM (Hrsg.): „12 Table View“.

Datenmodells werden mithilfe dieser Adapterklasse somit in Properties-Objekte umgewandelt, welche schließlich für die Darstellung in der TableView eingesetzt werden.

AdrContentHandler: Die Struktur der Klasse „AdrContentHandler“ wird für die Umstellung leicht verändert. Die Klasse wird weiterhin von der Klasse „ContentHandler“ des SAX Parsers vererben. Sämtliche Änderungen finden in den überschriebenen Methoden der Klasse „ContentHandler“ statt. In den Methoden werden neue Instanzen des Datenmodells für Elektronikrechnungen erzeugt und mit eingelesenen Daten gefüllt. Für die Anzeige in der TableView werden die Instanzen an die Adapterklasse weitergegeben.

In der Praktikumsanleitungen werden Ausschnitte aus dem Quellcode der Java Klassen dargestellt. Die Ausschnitte dienen für Praktikumsteilnehmer als Vorlage für die Erstellung der Klassen. Da sich die Struktur der Klassen bei der Umstellung auf JavaFX ändert, muss die Praktikumsanleitung ebenfalls angepasst werden. Texte mit Beschreibungen und Anweisungen zu den Listings werden auf den aktuellen Stand angepasst. Die Praktikumsanleitung enthält außerdem einige Abbildungen, welche Java-Swing-Elemente grafisch darstellen. Die Abbildungen werden durch gleichwertige Darstellungen von JavaFX Elementen ausgetauscht. Abschließend wird das Praktikum XML-SAX auf das neue Praxisbeispiel „Elektronikrechnungen“ umgestellt. Für die Umstellung werden Änderungen an der Praktikumsanleitung durchgeführt. In die erste Teilaufgabe müssen wenige Änderungen durchgeführt werden. Die Angaben von Dateipfaden in den Listings müssen angepasst werden und einige Abbildungen von Konsolen-Ausgaben des SAX Parsers müssen ausgetauscht werden. Die Einbindung des neuen Praxisbeispiels in die zweite Aufgabe wird bei der Umstellung der grafischen Benutzeroberfläche auf JavaFX mit berücksichtigt.

4.2.4. Realisierung der Verbesserungen

Die Benutzeroberfläche des bestehenden Praktikums XML-SAX wird von Java-Swing auf JavaFX umgestellt. Die Benutzeroberfläche war für das alte Praxisbeispiel „Firmenadressen“ ausgelegt. Damit die Benutzeroberfläche für das Praxisbeispiel „Elektronikrechnungen“ verwendet werden kann, wurden auch einige Änderungen an den Steuerungselementen und der Haupttabelle der Benutzeroberfläche durchgeführt. In der Abbildung 6 ist eine Bildschirmaufnahme der auf JavaFX umgestellten Benutzeroberfläche dargestellt. Der allgemeine Grundaufbau der Oberfläche wurde beibehalten. Im oberen Bereich wurden weitere Elemente für die Tabellenfilterung eingesetzt. In der Java-Swing-Benutzeroberfläche stand dazu ein Textfeld für die Filterung der gesamten

Tabelle zur Verfügung. In der neuen Benutzeroberfläche sind zwei weitere Textfelder für die Filterung vorhanden. Zusammengefasst können in der Tabelle Elektronikrechnungen nach Rechnungs-ID, Kaufdatum und Händlerdetails gefiltert werden. Die Haupttabelle dient, wie auch die Tabelle der alten Java-Swing-Benutzeroberfläche, zur Darstellung von Daten aus einer XML-Datei. Der Unterschied liegt darin, dass in der neuen Benutzeroberfläche nun Daten des Praxisbeispiels „Elektronikrechnungen“ dargestellt werden. Des Weiteren besitzt die JavaFX Tabelle nur noch 4 Spalten, damit die komplexe Datenstruktur der Elektronikrechnungen kompakter dargestellt wird. Die Spalte „ID“ zeigt die Identifikationsnummer einer einzelnen Rechnung an. In der Spalte „Kaufliste“ werden die gekauften einzelnen Gegenstände einer Rechnung untereinander dargestellt. In der Spalte „Händlerdaten“ wird der Händlername sowie die Anschrift des Händlers in einer einzelnen Zeile zusammengefasst.

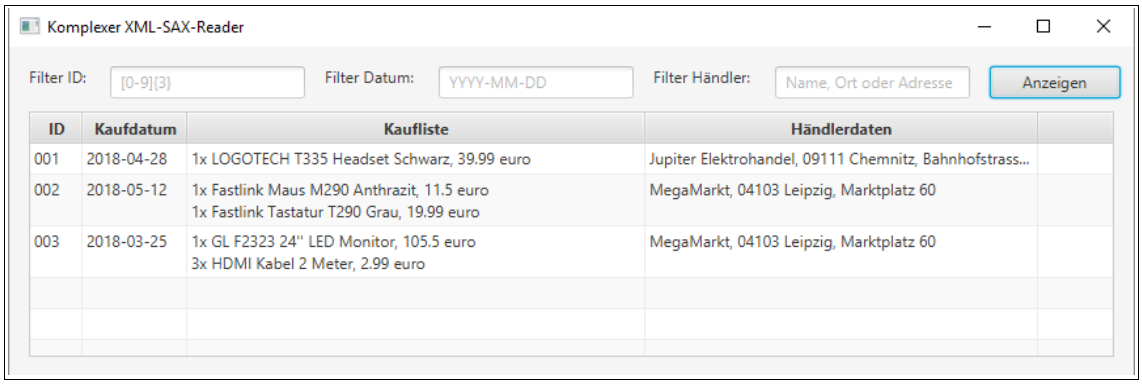


Abbildung 6: JavaFX Benutzeroberfläche für das Praktikum XML-SAX

4.3. Praktikum Webservices

4.3.1. Überblick

Im Lehrmodul Datenrepräsentation stehen zum Praktika Webservices mehrere Themen zur Verfügung. Dabei gibt es ein Praktikum zum Thema Webservices mit SOAP sowie ein Praktikum zum Thema RESTful Webservices. Das Praktikum zum Thema RESTful Webservices wird erweitert, da dieses im Lehrmodul kaum verwendet wird. Bisher stand zum Thema Webservices nur die Technologie SOAP im Vordergrund. Für die Erweiterung des Praktikums stehen Dateien aus dem Intranet des Lehrmoduls Datenrepräsentation des Wintersemesters 2016/2017 sowie für die Bearbeitung der Praktika bereitgestellte Inhalte zur Verfügung. Im Lehrmodul wird das Praktikum „PR10 – Webservices“ durchgeführt. Für das Praktikum ist eine Anleitung sowie eine Reihe von Softwarebibliotheken vorgegeben. Zu den Vorgaben gehören 3 JAR Dateien, wel-

che für die Implementierung von Java Funktionen im Zusammenhang mit Webservices benötigt werden. Des Weiteren werden 2 Archivdateien vorgegeben. Die erste Datei enthält die Software „Tomcat“ von Apache und wird im Praktikums für die Erstellung eines Webservers benötigt. In der zweiten Archivdatei wird das Framework „Axis“ von Apache zur Verfügung gestellt. Axis wird im Praktikum zur Erstellung von auf SOAP basierenden Webservices verwendet.

Da die Technologie SOAP bisher den Hauptteil des Praktikums Webservices bildet, sind nur wenige Vorgaben im Zusammenhang mit REST vorhanden. Für die Bearbeitung des Praktikums sind lediglich JAR Dateien für die Implementierung von Java-Funktionen im Zusammenhang mit REST sowie die Quelldateien des Eclipse-Projektes „DemoSimpleREST“ vorgegeben. Eine Übersicht über die Vorgaben ist in der folgenden Abbildung dargestellt.

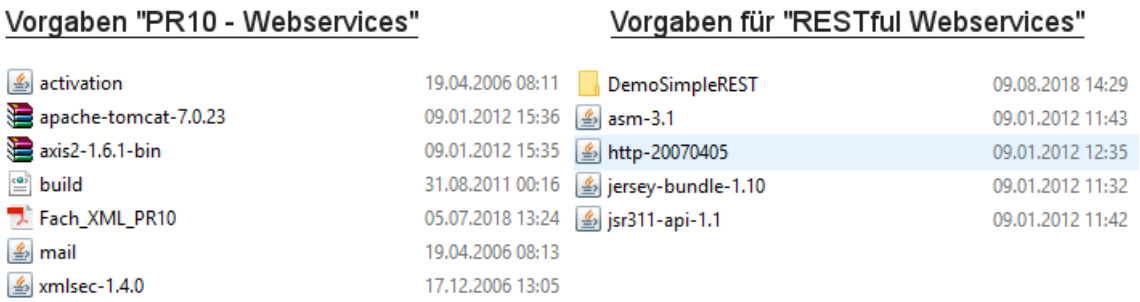


Abbildung 7: Vorgaben für das Praktikum Webservices

4.3.2. Ablauf

Der Ablauf des Praktikums wird in der Praktikumsanleitung „Fach_XML_PR10“ festgelegt, welche den Titel „Java-Webservices mit Tomcat und Axis“ trägt. Im Praktikum wird die Erstellung eines Webservices durch die Verwendung von Apache Tomcat und dem Framework Axis durchgeführt. Das Praktikum ist dabei in mehrere Teilaufgaben unterteilt. In der ersten Teilaufgabe wird ein Server mit Tomcat erstellt und konfiguriert. Im Praktikum werden dafür die Vorgaben aus dem Archiv „apache-tomcat-7.0.23“ verwendet. Nach der Konfigurierung wird der Server durch den Aufruf der Datei „startup.bat“ gestartet. Für die Überprüfung der Funktionsfähigkeit des Server soll zum Schluss der ersten Teilaufgabe eine HTML-Datei erstellt werden. Die zweite Teilaufgabe des Praktikums behandelt die Installation des Frameworks Axis von Apache. Als Vorgabe für die Teilaufgabe wird das Archiv „axis2-1.6.1-bin“ verwendet. Für die Installation werden Inhalte des Archivs in entsprechende Unterorder des Installationspfades von Tomcat kopiert. In der dritten Teilaufgabe ist die Implementierung einer Webservice-Funkti-

on in Java vorgesehen. Dazu wird eine JWS Datei mit einer Java-Funktion für das Verketten von zwei Zeichenketten erstellt und in das System von Axis hinzugefügt. Am Ende der dritten Teilaufgabe wird in der Praktikumsanleitung noch die automatische Erstellung des WSDL Dokuments von Axis demonstriert. Im vorletzten Abschnitt wird ein weiterer Webservice unter der Verwendung eines „Webservice Deployment Descriptors“ (WSDD) implementiert. Dazu wird eine Java Datei mit einer Webservice Funktion sowie WSDD Dateien für das Aktivieren und Deaktivieren des Webservices erstellt. Die letzte Aufgabe beschäftigt sich mit der Erstellung eines Java-Clients für das Aufrufen des Webservices. Der Java-Client wird dabei auf der Basis von JAX-WS angefertigt. In der Praktikumsanleitung werden die Schritte für die Erstellung des Clients unter der Verwendung von Eclipse dargestellt. Zuerst wird ein neues Projekt mit dem Titel „DemoWebserviceClient“ erstellt. Danach werden die JAR Dateien aus den Vorgaben zum Projekt hinzugefügt. Schließlich erfolgt die Implementierung einer Java Klasse für das Aufrufen und das Testen der Webservices. Die Implementierungsdetails werden in Listings innerhalb der Praktikumsanleitung abgebildet.

Zusammengefasst wird in der Praktikumsdurchführung ein Webserver mit Tomcat erstellt, um Beispiel-Webservices mithilfe von Apache Axis zu implementieren. Für die Implementierungen der Webservices werden verschiedene Möglichkeiten verwendet. Die erstellten Webservices werden zum Schluss des Praktikums durch einen Java-Client getestet, welcher mithilfe der Entwicklungsumgebung Eclipse erstellt wird.

4.3.3. Möglichkeiten zur Modernisierung

Für das Praktikum zum Thema Webservices mit SOAP sind keine umfangreichen Änderungen vorgesehen, da die Erweiterung des Praktikums mit Inhalten zur Technologie RESTful Webservices im Vordergrund steht. Dennoch besteht ein Teil der Aufgabenstellung darin, mögliche, zur Modernisierung beitragende, Alternativen für die im Praktikum verwendeten Frameworks zu nennen. Wie im Kapitel 4.3.2 festgestellt wurde, handelt es sich bei der im Praktikum verwendeten Software um das Framework Axis sowie den Server Tomcat von Apache. 2011 erschien im Magazin „LinuxUser“ ein Artikel über 4 mögliche Alternativen für Server von Apache. Als Alternativen wurden dabei die Server „Monkey HTTP Daemon“, „Hiawatha“, „Lighttpd“ und „Thttpd“ genannt, deren Funktionsweise eine Ähnlichkeit mit dem Server von Apache besitzt.[54] Eine mögliche Alternative speziell für den im Praktikum verwendeten Server Tomcat von Apache ist die „JBoss Enterprise Application Platform“ des Unternehmens Red Hat, welche zu-

54 Vgl. Schürmann (2011): „Vier alternative Webserver im Vergleich“.

sätzlich die Spezifikation der Java Enterprise Edition 6 unterstützt.[55] Auch für das Framework Axis von Apache gibt es mögliche Alternativen. Von Apache selbst gibt es das Framework CXF, welches aus den Projekten Celtix und XFire entstanden ist.[56]

55 Vgl. Sauvé (2015): „Introduction to Red Hat JBoss Middleware“, S.8.
56 Vgl. Balani et al.(2009): „Apache CXF Web Service Development“, S.16.

5. Entwicklung neuer Praktika

Das vierte Kapitel behandelt die Entwicklung neuer Praktikumsinhalte. Im Mittelpunkt stehen dabei die Themen „JSON“, „StAX“ und „RESTful Webservices“. Zu Beginn werden die einzelnen Abläufe der neuen Praktika konzipiert sowie die Vorgaben (Software, Frameworks, usw.), welche für die Durchführung des jeweiligen Praktikums benötigt werden, vorgestellt. Danach erfolgt die Realisierung der Praktika. Dabei stehen vor allem die Beispielprogramme im Vordergrund, welche während der Entwicklung der Praktika implementiert wurden. Anhand von Ausschnitten aus dem Quellcode werden wichtige Funktionen erläutert.

5.1. Praktikum JSON

5.1.1. Konzept

Der Praktikumsablauf für das Praktikum JSON wird in einer PDF-Anleitung dargestellt. Die Inhalte des Praktikums sollen Praktikumsteilnehmern grundlegende Kenntnisse von JSON vermitteln. Dafür wird zu Beginn eine Einführung zu JSON durchgeführt. In der Praktikumsanleitung werden dazu Grundlagen zur Syntax, zu Datentypen und zum Schema von JSON dargestellt. Für die Einführung wird eine JSON Datei mit Datensätzen des neuen Hauptbeispiels sowie ein JSON Schema für die Validierung dieser Datensätze einbezogen. Die Daten- und Schema-Dateien werden auch im weiteren Verlauf des Praktikums in den jeweiligen Teilaufgaben verwendet. Die Durchführung der Teilaufgaben erfolgt direkt nach Abschluss der Einführung. In der ersten Teilaufgabe wird die Verwendung von JSON in JavaScript dargestellt. Dabei wurde bewusst JavaScript für die Implementierung gewählt, da JSON auf Objekten von JavaScript basiert und somit die Komplexität des Beispiels gering bleibt. Zu Beginn der Teilaufgabe wird eine HTML-Datei und eine Java-Skript-Datei angelegt. In der Java-Skript-Datei wird die JSON Datei mit Beispieldatensätzen zuerst eingelesen und danach mit der standardmäßigen Methode „JSON.parse()“[57] verarbeitet. Die Skriptdatei wird dann in der HTML-Datei eingebunden, um die verarbeiteten Inhalte der JSON Datei in einem Webbrowser darzustellen. Für die Darstellung wird ein Texteingabefeld und eine Tabelle verwendet. Über das Eingabefeld wird ein Schlüssel für die eindeutige Identifizierung

57 Vgl. Bassett (2015): „Introduction to JavaScript Object Notation“, S.44.

eines Datensatzes eingegeben. In der Tabelle werden dann die zugehörigen Daten des Datensatzes angezeigt.

Nach der Darstellung von JSON Daten mit HTML und JavaScript erfolgt in der zweiten Teilaufgabe des Praktikums die Implementierung einer Konsolen-Anwendung mit Funktionen einer JSON-API in Java. Für die Implementierung wird die Bibliothek „Jackson“ als Vorgabe verwendet, da sich diese als eine standardmäßige JSON-API für Java etabliert hat.[58] Mit Jackson können JSON Dokumente verarbeitet und generiert werden. Die Entwicklungsversionen dieser Bibliothek werden auf der entsprechenden Projektseite unter Github veröffentlicht. Zu Beginn der Teilaufgabe wird ein neues Java-Projekt in Eclipse erstellt und es wird ein neues Paket mit der Hauptklasse der Anwendung angelegt. Auch in dieser Teilaufgabe wird die JSON Datei mit Beispieldaten des neuen Hauptbeispiels „Rechnungen“ verwendet. Damit die Jackson-API die Beispieldaten verarbeiten kann, werden verschiedene Datenmodellklassen im Eclipse Projekt angelegt. Eine einzelne Datenmodellklasse implementiert Get-Methoden und Set-Methoden für ein jeweiliges JSON Objekt der Beispieldaten. In der Hauptklasse wird dann ein vorgegebenes JSON Dokument mit Datensätzen des neuen Praxisbeispiels eingelesen und verarbeitet. Das Dokument wird auf Gültigkeit überprüft und die Inhalte des Dokuments werden in der Konsole angezeigt. Im nächsten Schritt werden mithilfe der Datenmodellklassen neue Elemente angelegt und unter Nutzung der JSON-API zum JSON Datendokument hinzugefügt. Zum Schluss der zweiten Teilaufgabe wird das Beispieldokument noch hinsichtlich des vorgegebenen JSON Schemas validiert. Für die Validierung eines JSON Datendokuments auf ein JSON Schema wird die Bibliothek „json-schema-validator-x.x.x“ verwendet. Die Bibliothek, welche ebenfalls öffentlich auf Github zur Verfügung steht, enthält Funktionen für Schema-Validierungen unter Java.[59] Im Eclipse Projekt werden dazu Validierungsfunktionen in einer separaten Klasse implementiert und schließlich in der Hauptklasse unter Nutzung des vorgegebenen Schemas getestet. Zusätzlich wird eine Anzeige für mögliche Fehler während der Validierung erstellt.

Zusammengefasst wird das Praktikum JSON grundlegende Kenntnisse über Elemente von JSON sowie die Verarbeitung von JSON mit JavaScript und Java darstellen. Für das Praktikum wird ein Dokument mit JSON Datensätzen sowie ein JSON Schema vorgegeben. Die Daten werden während der Verarbeitung mit JavaScript in einem Webbrowser dargestellt. Für die Verarbeitung mit Java wird eine Konsolen-Anwendung angelegt, welche unter Nutzung einer JSON-API JSON Daten einliest und erzeugt. Die Implementierung einer JSON Schema-Validierung in Java bildet den Schlusspunkt des Praktikums.

58 Siehe GITHUB.COM: „Jackson Project Home @github“.
59 Siehe GITHUB.COM: „Json-schema-validator“.

5.1.2. Realisierung des JavaScript Beispiels

Den ersten Teil des JSON Praktikums bildet die Erstellung einer HTML Seite, welche Daten aus der JSON Datei „Elektronikrechnungen_Daten.json“ darstellen soll. Die Praktikumsteilnehmer erstellen dabei unter Nutzung der Praktikumsanleitung schrittweise eine Skript-Datei und eine HTML-Datei. In der JavaScript Datei wird zu Beginn eine Funktion „ladeJSON()“ implementiert. In der Funktion wird ein HTTP Get Request auf die entsprechende JSON Datei durchgeführt und es wird ein Callback mit JSON Text aufgerufen, sobald der GET Request erfolgreich durchgeführt wurde. Im folgenden JavaScript Programmausschnitt ist der soeben genannte Ablauf dargestellt.

```
1     var httpObj = new XMLHttpRequest();
2     httpObj.overrideMimeType("application/json");
3     httpObj.open('GET', 'Elektronikrechnungen_Daten.json');
4     httpObj.onreadystatechange = function() {
5        if (httpObj.readyState == 4 && httpObj.status == "200") {
6           callback(httpObj.responseText);
7        }
8     }
```

Beim Aufruf der Methode „ladeJSON()“ wird ein Objekt für den HTTP Request erzeugt und es wird der Medientypen festgelegt. Danach erfolgt der GET Request auf die angegebene JSON Datei. Sobald der GET Request beendet wurde und sich der Zustand des HTTP Objektes ändert, wird ein Callback mit dem durch HTTP-GET erhaltenen JSON Text durchgeführt (siehe Zeile 6). Die Funktion „ladeJSON“ wird schließlich von einer weiteren Funktion mit dem Namen „verarbeiteJSON()“ aufgerufen. In dieser Funktion wird der Erhaltene JSON Text schließlich mit der JavaScript Standardfunktion „JSON.parse()“ verarbeitet. Die Verarbeitung wird eingeleitet, sobald der Callback der Funktion „ladeJSON()“ aufgerufen wird. Der unten dargestellte Programmausschnitt stellt den Aufruf diese Funktion dar. Der Callback übergibt den JSON Text in der Variable „jsoninhalt“, welcher schließlich mit der JavaScript Standardfunktion verarbeitet wird (siehe Zeile 2). Das daraus erzeugte JSON Objekt wird an eine weitere Funktion übergeben, welche schließlich Inhalte dieses JSON Textes in einer HTML-Seite darstellt.

```
1     ladeJSON(function(jsoninhalt) {
2        jsonObj = JSON.parse(jsoninhalt);
3        zeigeRechnungenAn(jsonObj);
4     });
```

Im Praktikum wird als Vorlage für die HTML-Seite die Datei „Rechnungsanzeiger.html“ verwendet. Diese Seite wird während der Praktikumsdurchführung schrittweise erweitert. Bei der Anzeige handelt es sich um eine HTML-Tabelle, welche Daten einer einzelnen Rechnung darstellen soll. Weiterhin wird auf der HTML-Seite ein Texfeld und

ein Button verwendet. Im Textfeld wird die ID einer Rechnung eingegeben. Durch die Interaktion mit dem Button wird die JavaScript Funktion „verarbeiteJSON()“ ausgeführt. Bei der Verarbeitung wird die Eingabe aus dem Textfeld gelesen und für die Suche nach der Rechnung mit dieser ID verwendet. Die Daten der gefundenen Rechnung werden schließlich in die Zellen der HTML-Tabelle geschrieben. Der Quelltext der HTML-Datei ist in der Anlage 4 dieser Arbeit beigefügt. Der gesamte Quelltext des der JavaScript Funktionen wird in der Anlage 5 dargestellt.

5.1.3. Realisierung der Java Anwendung

Im zweiten Teil der Praktikumsdurchführung wird die Konsolen-Anwendung für die Verarbeitung von JSON in Java erstellt. Die einzelnen Vorgänge für die Implementierung der Anwendung werden in der Praktikumsanleitung etappenweise dargestellt. Den Kern der Anwendung bildet das im Kapitel 3.2.1 beschriebene Datenmodell. Das Datenmodell spielt vor allem während des Einlesens von JSON mithilfe der API „Jackson“ eine wichtige Rolle. Jackson-Funktionen verarbeiten JSON Daten, indem sie diese Daten direkt auf ein Modell und dessen Java-Objekte abbildet. Die für dieses Verfahren wichtigste Klasse ist der „ObjectMapper“ aus dem Paket „com.fasterxml.jackson.databind“. Dieser liest zu Beginn des Verfahrens den Text eines JSON Dokuments ein, dessen Dateipfad mit einer Zeichenkette angegeben wird. Die Feldnamen dieses JSON Textes werden nun auf die Wurzelklasse des Datenmodells abgebildet. Die Namen der Java-Objekte müssen dabei exakt den Feldnamen des JSON Textes entsprechen. Durch die Verwendung von Set-Methoden werden schließlich Werte des JSON Textes auf die korrespondierenden Java-Objekte zugeordnet.[60] Im folgenden Beispiel wird dargestellt, wie der ObjectMapper im Praktikum unter der Nutzung des Praxisbeispiels „Elektronikrechnungen“ verwendet wird.

```
1     Rechnungsliste liste = new Rechnungsliste();
2     liste.setRechnungen(new ArrayList<Rechnung>());
3     ObjectMapper obm = new ObjectMapper();
4     liste = obm.readValue(new File(json_pfad),Rechnungsliste.class);
5     return liste;
```

Im oben dargestellten Ausschnitt bildet die Modellklasse „Rechnungsliste“ die Wurzel des Datenmodells. Nachdem eine neue Instanz des ObjectMappers angelegt wurde, wird der Dateipfad des JSON Dokuments sowie die Klassenbezeichnung des Modells für die Rechnungsliste an die Instanz des ObjectMappers übergeben. Nach der Durchführung der Methode „readValue()“ wird schließlich eine Instanz der Rechnungsliste zu-

60 Vgl. Jenkov (2018): „Jackson ObjectMapper“, Überschrift: „How Jackson ObjectMapper Matches JSON Fields to Java Fields“.

rückgegeben. Sofern das Einlesen von JSON und die Abbildung der Feldnamen auf Java-Objekte erfolgreich war, wird die Instanz der Rechnungsliste sämtliche Rechnungsdaten der eingelesenen JSON Datei enthalten. Wenn das Einlesen von Rechnungen nicht erfolgreich sein sollte, dann könnte eine mögliche „NullPointerException“ durch die fehlende Instanz des Listen-Objekts für Rechnungsobjekte erscheinen. Damit solche Fehler verringert werden, verwendet man in der Java-Programmierung i. d. R. „leere“ Listen anstatt nicht-instanziierter Listen.[61] Dieser Ansatz wird in dieser Java-Anwendung ebenfalls umgesetzt. So wird (siehe zweite Zeile des oben dargestellten Beispiels) der Klasse „Rechnungsliste“ die Instanz einer leeren Liste übergeben. Für die Implementierung der Liste wird die Klasse „java.util.List“ aus der Java-Standardbibliothek verwendet. Das gesamte Klassendiagramm des Datenmodells für Elektronikrechnungen wird in der Anlage 3 dargestellt.

Ein weiterer Teil der Java-Anwendung implementiert Funktionen der Softwarebibliothek „json-schema-validator“ eine Funktion für das Validieren von JSON Daten des Elektronikrechnungs-Beispiels. Die Bibliothek nutzt dabei selbst Funktionalität des Jackson-API. Im JSON Praktikum wird für die Java-Anwendung zunächst eine neue Klasse für die Validierung angelegt. Die Praktikumsteilnehmer implementieren schließlich unter der Nutzung von Jackson und der Bibliothek für die Schema-Validierung eine Methode, mit welcher zu Beginn JSON Texte aus den vorgegeben Dateien „Elektronikrechnungen_Daten.json“ und „Elektronikrechnungen_Schema.json“ separat eingelesen werden. Schließlich wird der eingelesene Text der Datendatei hinsichtlich des eingelesenen Textes der Schemadatei validiert. Für die Auswertung des Ergebnisses der Validierung wird die Klasse „ProcessingReport“ der Schema-Bibliothek verwendet. Im unten dargestellten Programmausschnitt wird der Prozess der Validierung dargestellt.

```
1     ObjectMapper obm = new ObjectMapper();
2     sNode = obm.readTree(new File(schemaPfad));
3     dNode = obm.readTree(new File(datenPfad));
4     schema = JsonSchemaFactory.byDefault().getJsonSchema(sNode);
5     ProcessingReport report = schema.validate(dNode);
6     if (!report.isSuccess()) {
7           for (ProcessingMessage message : report) {
8                 System.out.println("Nachricht: " + message);
9           }
10    }
```

Dazu wird wieder die Klasse „ObjectMapper“ der Jackson-API verwendet. Dabei ist zu beachten, dass der ObjectMapper die Daten in diesem Fall nicht auf das Datenmodell abbildet. Für die Abbildung werden sogenannte „JSON Knoten“ verwendet, welche eingelesene JSON Texte in einer Baumstruktur abbilden. Diese Art und Weise des Verarbeitens von JSON Texten ist in etwa vergleichbar mit dem Document Object Model

61 Vgl. Bloch (2018): „Effective Java“, S.247.

Parser für XML.[62] Aus dem Schema Knoten wird schließlich ein Schema Objekt erzeugt, mit welchem der Prozess des Validierens gestartet werden kann. Der Prozess wird im oben dargestellten Beispiel in der fünften Zeile durch den Aufruf der Funktion „validate()“ ausgeführt. Die Ergebnisse der Validierung werden schließlich durch die Nutzung der Klasse „ProcessingReport“ der Bibliothek „json-schema-validator“ verarbeitet. Sollte die Validierung nicht erfolgreich sein, so können bestimmte Nachrichten mit Hinweisen auf die Ursachen der fehlerhaften Validierung ausgegeben werden. Die letzte Funktion der Java-Anwendung dient zum Schreiben von Daten in eine JSON Datei. Im Praktikum wird dazu mithilfe des Datenmodells zuerst ein neues Rechnungsobjekt erzeugt und zur Instanz der eingelesenen Rechnungsliste hinzugefügt. Die aktualisierte Liste besitzt nach der Erzeugung somit die derzeitigen Daten der eingelesen JSON Datei und zusätzlich das in Java erzeugte Rechnungsobjekt mit neuen Daten. Das Java-Objekt der aktualisierten Rechnungsliste wird zum Schluss in einen JSON Text konvertiert, welcher schließlich in eine JSON Datei geschrieben wird. Das Umwandeln von Java-Objekten zu JSON Texten sowie das Schreiben in eine JSON Datei wird in der Anwendung wieder mithilfe der Jackson-API durchgeführt. Die API stellt dafür die Klasse „ObjectWriter“ zur Verfügung. Dem „ObjectWriter“ wird dazu das Java-Objekt sowie der Dateipfad einer JSON Datei übergeben. Schließlich wird der Schreibprozess durchgeführt. Im folgenden Programmausschnitt wird der eben genannte Prozess dargestellt.

```
1     r.getRechnungen().add(erzeugeRechnung());
2     ObjectMapper mapper = new ObjectMapper();
3     ObjectWriter writer = mapper.writer(new DefaultPrettyPrinter());
4     try {
5           writer.writeValue(new File(pfad), r);
6     } catch (Exception e) {
7           e.printStackTrace();
8     }
```

Im oben dargestellten Beispiel wird bei der Erzeugung der Klasse „ObjectWriter“ zusätzlich eine Instanz der Klasse „DefaultPrettyPrinter“ (siehe Zeile 3) überführt. Die Klasse formatiert den in Java erzeugten JSON Text und formatiert diesen in eine lesbare Darstellung. Ohne die Formatierung würde der gesamte in eine Datei geschriebene JSON Text in einer einzigen Zeile dargestellt werden. Die Klasse „DefaultPrettyPrinter“ verändert den JSON Text somit nicht, sondern stellt diesen lediglich in einer für den Menschen lesbaren Form dar.

Da es sich bei dieser Java-Anwendung um eine Konsolen-Anwendung handelt werden sämtliche Ausgaben, die während des Einlesens, der Validierung des JSON Schemas oder des Schreibvorgangs auftreten, auf der Konsole dargestellt. Anzeigen für die Kon-

62 Vgl. Kumar (2018): „Jackson JSON Java Parser API Example Tutorial“, Überschrift: „Read Specific JSON Key“.

sole werden eigenständig durch die Praktikumsteilnehmer erzeugt oder mithilfe der Praktikumsanleitung implementiert.

5.2. Praktikum XML-StAX

5.2.1. Konzept

Zur Thematik der XML-Parser wird ein neues Praktikum zur Technologie XML-StAX angelegt. Die im Praktikum durchzuführenden Implementierungen von Funktionen des StAX Parsers erfolgen unter der Nutzung des Pakets „javax.xml.stream“ der Java-Standardbibliothek. StAX wurde mit der Einführung von Java 6.0 in die Standardbibliothek integriert.[63] Für die Programmierung wird im Praktikum wieder die Eclipse IDE verwendet, in welcher zu Beginn ein neues Java-Projekt angelegt wird. Das Praktikums XML-StAX verläuft ähnlich wie das bereits bestehende Praktikum XML-SAX. Dabei wird wieder eine Konsolen-Anwendung erstellt, in welcher Ergebnisse von Funktionsaufrufen des StAX Parsers ausgegeben werden. Das Praktikum wird 2 Teilaufgaben besitzen. In der ersten Teilaufgabe wird ein XML-Dokument mit Daten des Praxisbeispiels „Elektronikrechnungen“ eingelesen und mit dem StAX Parser verarbeitet. In der zweiten Teilaufgabe werden mithilfe von StAX neue Rechnungsdaten in eine vorgegeben XML Datei geschrieben. Auch für dieses Praktikum wird eine Anleitung erstellt, in welcher die Implementierung der genannten Funktion schrittweise dargestellt wird.

Zusammengefasst steht nach erfolgreicher Durchführung des Praktikums eine Konsolen-Anwendung in Java zur Verfügung, in welcher Funktionen von StAX der Java Standardbibliothek beispielhaft dargestellt werden. Die Anwendung enthält Funktionen für das Einlesen von XML-Dokumenten, für die Abfrage bestimmter Datensätze aus einem XML-Dokument und für das Einfügen neuer Datensätze in ein bestehendes XML-Dokument.

5.2.2. Realisierung

Für das neue konzipierte Praktikum zum Thema XML-StAX wird eine Java Konsolen-Anwendung erstellt, in welcher Funktion von StAX unter der Nutzung der Java Standardbibliothek dargestellt werden. Die einzelnen Schritte für die Implementierung dieser Anwendung wird auch in diesem Praktikum in einer Praktikumsanleitung dargestellt. Da für die Verwendung von StAX die Standardbibliothek von Java verwendet

63 Vgl. Mikhalenko (2008): „StAX: So parst man XML-Code mit Java“.

wird, werden keine weiteren Vorgaben benötigt. Für dieses Praktikum wird ausschließlich eine XML mit Daten des Praxisbeispiels „Elektronikrechnungen“ sowie eine XSD Datei vorgegeben. Die Java-Klassen für das Datenmodell von Elektronikrechnungen werden aus vorherigen Praktika weitergenutzt. Während der Praktikumsdurchführung wird eine Java-Klasse erstellt, in welcher Funktionen für die Verarbeitung von XML mit StAX implementiert wird. Für den Leseprozess mit StAX werden dazu die Klassen „XMLInputFactory“ und „XMLEventReader“ aus dem Paket „javax.xml.stream“ der Standardbibliothek verwendet. Mit einer Instanz der Input Factory kann eine XML Datei durch die Angabe eines Dateipfads geladen werden. Die Input Factory erzeugt schließlich eine Instanz für den Event Reader.

```
1     XMLInputFactory factory = XMLInputFactory.newInstance();
2     XMLEventReader eventReader =
            factory.createXMLEventReader(new FileReader(dateipfad));
```

Die XML Datei wird nun zeilenweise, als Datenstrom, eingelesen. Während des Einleseprozesses begegnet der StAX Parser verschiedene Ereignisse, welche als XML Events bezeichnet werden. Bei einem solchen Ereignis kann es sich z. B. um den Start eines XML Elementes handeln. Mit dem Event Reader werden diese Ereignisse nacheinander abgearbeitet.

5.3. Praktikum RESTful Webservices

5.3.1. Konzept

Die Praktikumsinhalte zum Thema Webservices werden durch ein neues Praktikum zur Technologie RESTful Webservices ergänzt. Auch für dieses Praktikum wird eine neue Anleitung erstellt. Das Praktikum wird sich mit der Implementierung eines Beispiel-Webservices auf der Basis von REST beschäftigen. Für die Praktikumsdurchführung wird wieder die Programmiersprache Java verwendet. In der Java Enterprise Edition steht durch JAX-RS eine standardmäßige Schnittstelle für die Implementierung von RESTful Webservices zur Verfügung[64]. Da für alle Java-Praktika die Java Standard Edition verwendet werden soll, muss eine Implementierung von JAX-RS verwendet werden. Für das Praktikum wird dazu die Bibliothek „Jersey“ als Implementierung von JAX-RS verwendet. Für die Ausführung und das Testen von erstellten Webservices wird wie auch im SOAP Praktikum ein Webserver benötigt. Im Praktikum RESTful Webservices wird dafür das HTTP-Server Framework „Grizzly“ verwendet. Das Frame-

64 Vgl. Pericas-Geertsen et al. (2013): „JAX-RS: Java API for RESTful Web Services“, S.2.

work stellt Funktionen für die Erstellung einfacher Webserver sowie zur Kommunikation zwischen Server und Client über HTTP zur Verfügung[65] und bietet sich somit für die Implementierung von RESTful Webservices an. Ein Webserver kann dabei direkt in Java implementiert werden, wodurch der Aufwand der Erzeugung eines externen Webservers mit z. B. Tomcat nicht notwendig ist. Die Nutzung von Jersey und Grizzly erfolgt durch die Einbindung entsprechender JAR Dateien in das Java-Projekt. Für den Erhalt dieser Dateien werden in der Praktikumsanleitung entsprechende Verweise und Links auf die Projektseiten von Jersey und Grizzly angegeben. Für die Praktikumsdurchführung wird wieder die Eclipse-IDE verwendet. In dieser wird zu Beginn ein neues Java-Projekt angelegt und alle benötigten Bibliotheken werden in das Projekt eingebunden. Für die Vorbereitung auf die Erstellung der Webservices wird im zweiten Schritt ein einfacher HTTP-Server mithilfe von Grizzly erstellt und gestartet. Danach erfolgt die Implementierung eines Beispiel-Webservices und eines Clients. Unter Einbeziehung des Praxisbeispiels „Elektronikrechnungen“ wird ein Webservice angelegt, welcher bestimme Funktionen (z. B. Berechnung der Summe aller Rechnungsbeträge) durchführt. Die Implementierung des Clients erfolgt im gleichen Java-Projekt innerhalb der Eclipse-IDE. Mit dem Client wird die Funktionalität des Webservices getestet. Im Client werden Zugriffe auf den Webservice durch die Nutzung von HTTP-Standardmethoden durchgeführt. Die Verwendung von HTTP-Standardmethoden ist ein Grundprinzip von REST (siehe Kapitel 2.3.3). Die Verdeutlichung von Grundprinzipien von REST wird bei der Praktikumsdurchführung im Vordergrund stehen. Für das Prinzip der unterschiedlichen Repräsentationen von Ressourcen wird ein Webservice über verschiedene Wege, wie z. B. durch die Verwendung des Java-Clients oder durch Aufruf über HTML, angesprochen. Auch das Prinzip von Hypermedia wird bei der Durchführung des Praktikums genauer betrachtet.

Zusammengefasst wird im Praktikum RESTful Webservices ein unter der Nutzung von JAX-RS erstellter Webservice implementiert und ein Java-Client für den Test des Webservices erstellt. Unter Einbezug des Rechnungsbeispiels werden die Grundprinzipien von REST während der Durchführung des Praktikums verdeutlicht.

5.3.2. Realisierung des Servers

Im Entwurf dieses Praktikums wurde das Framework „Grizzly“ für die Implementierung eines Webservers gewählt. Mit dem Framework kann ein HTTP-Server direkt in Java implementiert werden. Im Rahmen des Praktikums wird dafür eine Klasse mit der Bezeichnung „TestServer.java“ implementiert. In dieser Klasse steht eine Methode für die

65 Siehe Project Grizzly: „Http Server Framework Overview“.

Erzeugung und den Start eines HTTP-Servers zur Verfügung. Im folgenden Programmausschnitt wird der soeben beschriebene Ablauf dargestellt.

```
1     ResourceConfig rc = new ResourceConfig().packages("rest");
2     HttpServer server =
            GrizzlyHttpServerFactory.createHttpServer(
            URI.create("http://localhost:4434"), rc);
3     server.start();
4     System.out.println("Server gestartet");
```

In der ersten Zeile wird eine Konfiguration für Ressourcen angelegt. Die Konfiguration legt fest, in welchem Paket die Ressourcen für den Webservice zur Verfügung stehen. In der Anwendung wird dazu auf das Paket „rest“ verwiesen, welches während der Durchführung des Praktikums durch Praktikumsteilnehmer erzeugt werden. In diesem Paket werden diverse Java-Klassen angelegt, welche bestimmte Ressourcen repräsentieren (z. B. Rechnungen). Eine einzelne Ressourcenklasse ist dabei durch die Bezeichnung einer URI eindeutig identifizierbar. Die URI wird vor der Klassendefinition durch die Annotation „@Path“ dargestellt, welche den Pfad der Ressourcenklasse angibt (z. B. „/rechnung“). Durch die Angabe dieser Annotation wird die Ressourcenklasse als ein JAX-RS Webservice kenntlich gemacht.[66] In der zweiten Zeile des oben dargestellten Programmausschnittes erfolgt die Erstellung einer Instanz der Klasse „HTTPServer“. Mit der Verwendung von Grizzly wird dazu ein lokaler Server erzeugt. Die Instanz der Konfiguration für Ressourcen wird während der Erzeugung als Parameter übergeben, wodurch die festgelegten URIs der Ressourcen im Server festgelegt werden. Nach der Erzeugung und Konfiguration des Servers wird dieser schließlich direkt gestartet.

5.3.3. Realisierung der Clients

Für den Test des Webservers sowie Funktionen von Webservices werden im Praktikum verschiedene Clients erstellt und verwendet. Für die Erzeugung eines Clients in Java steht die Client-API von JAX-RS zur Verfügung. Mit dieser Schnittstelle können HTTP-Verbindungen zum Webserver erzeugt und verwaltet werden.[67] In der Praktikumsdurchführung wird dazu eine Java-Klasse „TestClient“ angelegt, in welcher verschiedene Methoden für den Test des Webservices erstellt werden. Die Funktionstests dienen im Praktikum in erster Linie zur Verdeutlichung der Funktionsweise von JAX-RS Ressourcen im Zusammenspiel mit HTTP-Requests. Im folgenden Programmausschnitt ist ein Beispiel für eine solche Testmethode dargestellt. Dabei wird mithilfe der Client-API von JAX-RS ein HTTP-GET Request auf die Rechnungsressource durchgeführt.

66 Vgl. Burke (2014): „RESTful Java with JAX-RS 2.0“, S.29.
67 Vgl. Burke, S.40.

```
1     Client client = ClientBuilder.newClient();
2     String antwort =
            client.target("http://localhost:4434/rechnung").
            request(MediaType.TEXT_PLAIN).get(String.class);
```

Als Ziel wird dabei die Adresse des Grizzly Webservers mit der URI der Rechnungsressource angegeben. Danach wird ein GET-Request auf die Rechnungsressource mit einer Anforderung auf die Repräsentation für „Plain Text“ durchgeführt. Als Antwort dieser Anforderung würde der Client in diesem Beispiel eine Zeichenkette mit einer Textrepräsentation der Ressource „/rechnung“ erhalten.

5.4. Kompatibilitätsprüfungen

Für die technische Umsetzung der in den Praktika behandelten Technologien wurde bereits in der Entwurfsphase eine spezielle Arbeits- und Entwicklungsumgebung eingerichtet. In dieser Umgebung wurden sämtliche Praktikumsinhalte, wie z. B. Java Anwendungen oder Dokumente, implementiert und angelegt. In der Aufgabenstellung dieser Arbeit wurden eine Reihe von Einschränkungen bezüglich der Arbeits- und Entwicklungsumgebung vorgegeben. Die wichtigste Einschränkung sieht vor, dass alle Praktikumsinhalte mit der Arbeitsumgebung in den PC-Pools der Hochschule Mittweida kompatibel sein müssen. Als Bezugspunkte wurden die PC-Pools im Haus 8 vorgegeben. Da die technische Umsetzung außerhalb dieser PC-Pools erfolgt, müssen Prüfungen der Kompatibilität durchgeführt werden. Die Prüfungen sollen sicherstellen, dass die Praktika ohne Kompatibilitätsprobleme in den PC-Pools durchgeführt werden können. Für beide Arbeitsumgebungen ist das Betriebssystem Windows 10 vorgesehen. Die Implementierung aller Java-Anwendungen erfolgt unter der Nutzung der Entwicklungsumgebung Eclipse Neon 3 sowie Eclipse Photon. In den PC-Pools der Hochschule wird Eclipse Mars 2 verwendet, einer Vorgängerversion von Neon und Photon. Zur Prüfung der Kompatibilität wurden die in Neon und Photon erstellten Eclipse Projekte in die Entwicklungsumgebung Eclipse Mars importiert. Die Java Anwendung wurden schließlich in der Entwicklungsumgebung ausgeführt und die Funktionen der Anwendungen wurden getestet. Während der Ausführung und der Tests konnten keine Unterschiede zwischen den verschiedenen Eclipse Entwicklungsumgebungen Neon, Photon und Mars festgestellt werden. In allen Entwicklungsumgebungen wurden jeweils Versionen des Java SE Development Kit 8 verwendet.

In der JavaScript Teilaufgabe des Praktikums JSON wird zusätzlich ein Webbrowser während der Praktikumsdurchführung verwendet. Im Webbrowser wird eine HTML-Seite angezeigt, welche eine in JavaScript implementierte Anzeige für Elektronikrechnun-

gen darstellt. Die technische Umsetzung dieses Praktikums erfolgte unter der Verwendung des Webbrowsers Mozilla Firefox Version 62. Zur Prüfung der Kompatibilität wurde das Praktikum in den PC-Pools testweise durchgeführt. Dazu wurde die HTML Datei mit den Webbrowsern Microsoft Edge und Google Chrome angezeigt und die Funktionen der JavaScript Datei wurden getestet. Die Ausführung in Microsoft Edge erfolgte ohne Fehler. Unter der Nutzung von Google Chrome ist die ordnungsgemäße Darstellung der Anzeige allerdings nur bedingt möglich. Die im Praktikum vorgegebene JSON Datei konnte nicht lokal geöffnet werden, da das Protokoll „file:///“ von Chrome blockiert wird. Damit der lokale Aufruf der Datei funktioniert, müssten entsprechende Zugriffe durch Veränderungen in den Einstellungen von Chrome ermöglicht werden. Das Praktikum kann in den PC-Pools dennoch durchgeführt werden, da mit Microsoft Edge mindestens ein kompatibler Webbrowser zur Verfügung steht.

6. Zusammenfassung

6.1. Ergebnisbetrachtung

Im folgenden Kapitel wird betrachtet, welche Ergebnisse im Verlauf dieser Arbeit entstanden sind und ob die Ziele der Aufgabenstellung erreicht wurden. Als allgemeines Ziel dieser Arbeit wurde die Verbesserung und Erstellung ausgewählter Praktikumsinhalte des Lehrmoduls Datenrepräsentation gewählt. Unter dem Stichwort „Verbesserungen“ wurden die bestehenden Praktika zu den Themen XML-SAX und Webservices analysiert und hinsichtlich spezieller Kriterien verändert. Die Analyse der Strukturen und Abläufe dieser zwei Praktika wurden im vierten Kapitel dieser Arbeit ausführlich dargestellt. Für die Veränderung der Praktika XML-SAX und Webservices wurden in der Aufgabenstellung verschiedene spezielle Kriterien genannt. Für das Praktikum XML-SAX stand dabei die Umstellung der im Praktikum verwendeten Benutzeroberfläche von Java-Swing auf JavaFX im Vordergrund. Die Umstellung dieser Benutzeroberfläche wurde erfolgreich durchgeführt. Sowohl das Konzept für die Umstellung als auch die technische Realisierung konnten im Rahmen dieser Arbeit umgesetzt werden. Die Kriterien für die Veränderung des Praktikums zum Thema Webservices waren weniger speziell gewählt. Im Vordergrund stand dabei die Erweiterung des Praktikums mit Inhalten zur Technologie RESTful Webservices. Aufgrund der Unterschiede zur Technologie SOAP wurde die Technologie RESTful Webservices allerdings als ein gesondertes Praktikumsthema betrachtet. Für das bestehende Praktikum zum Thema Webservices mit SOAP wurde somit lediglich eine Untersuchung zu möglichen Alternativen für die im Praktikum verwendeten Frameworks durchgeführt. Eine Auflistung möglicher Alternativen erfolgte im vierten Kapitel. Für das Praktikum zum Thema RESTful Webservices wurden zufriedenstellende Ergebnisse erreicht. Eine Praktikumsanleitung sowie eine Java-Anwendung für grundlegende Funktionen von RESTful Webservices konnten erstellt werden. Für das Praktikum war vorgesehen, dass die Grundprinzipien von REST anhand des Praxisbeispiels „Elektronikrechnungen“ und der Java-Anwendung dargestellt werden. Die Darstellung der meisten Grundprinzipien, wie z. B. die Verwendung von HTTP Standardmethoden oder das Prinzip der unterschiedlichen Repräsentationen, konnten in den Praktikumsinhalt aufgenommen werden. Für die Darstellung von Hypermedia kann die Java-Anwendung sowie die Praktikumsanleitung allerdings umfangreicher ausgebaut werden. Dazu könnten auch wesentlich mehr Beispieldaten-

sätze von Elektronikrechnungen verwendet werden, damit die Verknüpfungen zwischen Daten und Ressourcen und somit das Prinzip Hypermedia besser dargestellt werden kann.

In der soeben erfolgten Ergebnisbetrachtung zur Erstellung des Praktikums RESTful Webservices wurde das Praxisbeispiel „Elektronikrechnungen“ genannt. Der Entwurf dieses Praxisbeispiels und die Erzeugung von Beispieldaten waren ein wichtiger Teil der Aufgabenstellung. Diese Aufgaben konnten im Rahmen der Arbeit erfolgreich durchgeführt werden. Als Ergebnis konnte eine Reihe von Beispieldatensätze erzeugt werden, welche in JSON und XML repräsentiert werden können. Für das Praxisbeispiel wurden jeweils ein JSON Schema sowie eine XML Schema Definition erzeugt. Diese Schemata werden in den meisten Praktika als Vorgabe eingesetzt und können auch für zukünftige Praktikumsinhalte verwendet werden. Im Rahmen dieses Entwurfs des Praxisbeispiels konnte auch ein Java-Datenmodell erstellt werden. Die Ergebnisse dazu können in den Anhängen 1 bis 3 betrachtet werden.

Für die Praktika zu den Technologien JSON und XML-StAX stehen ebenfalls eine Reihe von Ergebnissen bereit. Das Praktikum JSON konnte, wie im Entwurf dargestellt, umgesetzt werden. Für das Praktikum konnten zwei Anleitungen sowie Programmbeispiele in JavaScript und Java erstellt werden. Die Praktika verdeutlichen die Grundlagen der JSON Syntax sowie des JSON Schemas. Für das Praktikum XML-StAX stehen eine Anleitung sowie ein Java Programmbeispiel als Ergebnisse bereit, in welchen die Grundlagen der Technologie StAX unter Nutzung der Java Standardbibliothek dargestellt werden.

6.2. Ausblick und Fazit

Im Rahmen dieser Arbeit konnte eine Reihe von Praktikumsinhalten für das Lehrmodul Datenrepräsentation verbessert und erstellt werden. Die in der Aufgabenstellung definierten Ziele dieser Arbeit konnten größtenteils umgesetzt werden. Auch die einzelnen Arbeitsschritte (Analyse, Entwurf, Realisierung) konnten im Rahmen dieser Arbeit gut dargestellt werden. Die Ergebnisse dieser Arbeit könnten theoretisch bereits in die Praktikumsveranstaltungen des Lehrmoduls Datenrepräsentation eingebunden werden. Es sollten allerdings weitere ausführliche Tests der Ergebnisse durchgeführt werden, damit auch zeitliche Abläufe und die Verständlichkeit der Praktikumsinhalte optimiert werden können. Die Frage, inwiefern die neuen Praktika später in das Lehrmodul eingebunden werden, bleibt somit weiterhin offen. Sicher ist, dass die behandelten Praktikumsinhalte auch in Zukunft weiter ausgebaut werden können.

Literaturverzeichnis

Literaturquellen:

Basset, Lindsay (2015): „Introduction to JavaScript Object Notation. A to-the-point guide to JSON",
1. Auflage, O'Reilly Media, Sebastopol, CA.

Bloch, Joshua (2018): „Effective Java",
3. Auflage, Addison-Wesley Professional, USA.

Burke, Bill (2014): „RESTful Java with JAX-RS 2.0, Designing and Developing Distributed Web Services",
2. Auflage, O'Reilly Media, Sebastopol, CA.

Tilkov, Stefan; Eigenbrodt, Martin; Schreier, Silvia; Wolf, Oliver (2015): „REST und HTTP. Entwicklungen und Integration nach dem Architekturstil des Web",
3. Auflage, dpunkt Verlag, Heidelberg.

Internetquellen:

Balani, Naveen; Hathi, Rajeev (2009): „Apache CXF Web Service Development",
Packt Publishing,
URL: „https://doc.lagout.org/programmation/tech_web/Apache/Apache%20CXF%20Web%20Service%20Development.pdf" (Stand: 13.09.2018, 19:56).

Booth, David; Haas, Hugo; McCabe, Francis; Newcomer, Eric; Champion, Michael; Ferris, Chris; Orchard, David (2004): „Web Services Architecture",
URL: „https://www.w3.org/TR/ws-arch/wsa.pdf" (Stand: 11.07.2018, 13:07).

Chinnici, Roberto; Moreau, Jean-Jaques; Ryman, Arthur; Weerawarana, Sanjiva (2007): „Web Servies Description Language (WSDL) Version 2.0, Part 1: Core Language",
URL: „https://www.w3.org/TR/wsdl/wsdl20.pdf" (Stand: 11.07.2018, 14:41).

GITHUB.COM (o.D): „Jackson Project Home @github",
URL: „https://github.com/FasterXML/jackson" (Stand: 19.09.2018, 20:25)

GITHUB.COM (o.D): „json-schema-validator",
URL: „https://github.com/java-json-tools/json-schema-validator" (Stand: 19.09.2018, 20:29)

Jenkov, Jakob (2018): „Jackson ObjectMapper",
URL: „http://tutorials.jenkov.com/java-json/jackson-objectmapper.html" (Stand: 03.09.2018, 14:44).

Jenkov, Jakob: „Java StAX“,
URL: „http://tutorials.jenkov.com/java-xml/stax.html“ (Stand: 23.08.2018, 14:30).

JSON-SCHEMA.ORG,
„http://json-schema.org/“ (Stand: 04.09.2018, 11:06).

Kay, Russell (2007): „Representational State Transfer (REST)“,
URL: „https://www.computerworld.com/article/2552929/networking/representational-state-transfer--rest-.html“ (Stand: 25.07.2018, 14:44).

Kumar, Pankaj (2018): „Jackson JSON Java Parser API Example Tutorial“,
URL: „https://www.journaldev.com/2324/jackson-json-java-parser-api-example-tutorial“ (Stand: 04.09.2018, 17:01).

Längle, Mariell (2015): „XML vs. JSON“,
URL: „http://mobdok.de/2015/05/xml-vs-json/“ (Stand: 24.07.2018, 13:55).

Mikhalenko, Peter (2008): „StAX: So parst man XML-Code mit Java“,
URL: „https://www.zdnet.de/39194401/stax-so-parst-man-xml-code-mit-java/“ (Stand: 22.08.2018, 19:21).

Mitra, Nilo; Lafon, Yves (2007): „SOAP Version Part 1.2 Part 0: Primer“,
Second Edition,
URL: „https://www.w3.org/TR/soap12-part0/“ (Stand: 13.08.2018, 14:17).

Nurseitov, Nurzhan; Paulson, Micheal; Reynolds, Randall; Izurieta, Clemente (o.D): „Comparison of JSON and XML Data Interchange Formats: A Case Study“,
URL: „https://www.cs.montana.edu/izurieta/pubs/caine2009.pdf“ (Stand: 24.07.2018, 13:56).

ORACLE.COM (o.D.): „12 Table View“,
URL: „https://docs.oracle.com/javafx/2/ui_controls/table-view.htm“ (Stand: 09.08.2018, 10:56).

ORACLE.COM (o. D.): „Why StAX?“,
URL: „https://docs.oracle.com/javase/tutorial/jaxp/stax/why.html“ (Stand: 16.09.2018, 19:12).

Pericas-Geertsen, Santiago; Potociar, Marek (2013): „JAX-RS: Java API for RESTful Web Services“,
Version 2.0 Final Release,
URL: „http://download.oracle.com/otn-pub/jcp/jaxrs-2_0_rev_A-mrel-eval-spec/jsr339-jaxrs-2.0-final-spec.pdf“ (Stand: 24.08.2018, 14:52).

Project Grizzly: „Http Server Framework Overview“,
URL: „https://javaee.github.io/grizzly/httpserverframework.html“ (Stand: 22.09.2018, 20:04)

Sauvé, Martin (2015): „Introduction to Red Hat JBoss Middleware“,
URL:
„https://people.redhat.com/mlessard/qc/presentations/fev2015/IntrotoRedHatJBoss.pdf“ (Stand: 13.09.2018, 13:09).

Schubert, Wilfried (2014): „Praktikum 02 – Erstellen und Testen von DTD's zum Validieren von XML-Dokumenten am Beispiel eines Messwert-Prüfberichtes“,
URL: „https://www.staff.hs-mittweida.de/~wschub/intranet/ws1718/Fach_XML/Fach_XML_PR02.pdf“ (Stand: 22.09.2018, 17:28)

Schürmann, Tim (2011): „Vier alternative Webserver im Vergleich“,
in: LinuxUser (2011), Ausgabe 04/11,
URL: „http://www.linux-community.de/ausgaben/linuxuser/ 2011/04/vier-alternative-webserver-im-vergleich/“ (Stand: 13.09.2018, 15:29).

Vogel, Lucas (2017): „JSON vs. XML: The battle for format supremacy may be wasted energy“,
URL: „https://sdtimes.com/communication-protocols/json-vs-xml-battle-format-supremacy-may-wasted-energy/“ (Stand: 19.08.2018, 14:35).

Wolf, Misha; Wicksteed, Charles (1997): „Date and Time Formats“,
URL: https://www.w3.org/TR/NOTE-datetime (Stand: 14.08.2018, 13:50).

Bildquellen:

Abbildung 1: **Hass, Hugo (o.D): „Service-Oriented Architectire“,**
URL: „https://www.w3.org/2003/Talks/0521-hh-wsa/soa.png“ (Stand: 11.07.2018, 14:57).

Abbildung 3: **Schubert, Wilfried (2014): „Praktikum 02 – Erstellen und Testen von DTD's zum Validieren von XML-Dokumenten am Beispiel eines Messwert-Prüfberichtes“,**
URL: „https://www.staff.hs-mittweida.de/~wschub/intranet/ws1718/Fach_XML/Fach_XML_PR02.pdf“ (Stand: 22.09.2018, 17:28).

Anlagen

Anlage 1: JSON Schema für Elektronikrechnungen

```
{
  "type": "object",
  "$schema": "http://json-schema.org/draft-04/schema#",
  "title": "Elektronikrechnungen",
  "description": "Eine Liste von Elektronikrechnungen",
  "properties": {
    "rechnungen": {
      "type": "array",
      "items": {
        "type": "object",
        "properties": {
          "id": {
            "type": "string",
            "pattern": "[0-9]{3}"
          },
          "datum": {
            "type": "string",
            "pattern":
                "[12][0-9]{3}-(0[1-9]|1[0-2])-(0[1-9]|[12][0-9]|3[01])"
          },
          "kaufliste": {
            "type": "array",
            "items": {
              "type": "object",
              "properties": {
                "bezeichnung": {
                  "type": "string"
                },
                "menge": {
                  "type": "integer",
                  "minimum": 1
                },
                "einzelpreis": {
                  "type": "object",
                  "properties": {
                    "betrag": {
                      "type": "number",
                      "minimum": 0
                    },
                    "waehrung": {
                      "type": "string",
                      "enum": [
                        "euro",
                        "usd",
                        "gpb"
                      ]
                    }
                  },
                  "required": [
                    "betrag",
                    "waehrung"
                  ],
                  "additionalProperties": false
                }
              },
```

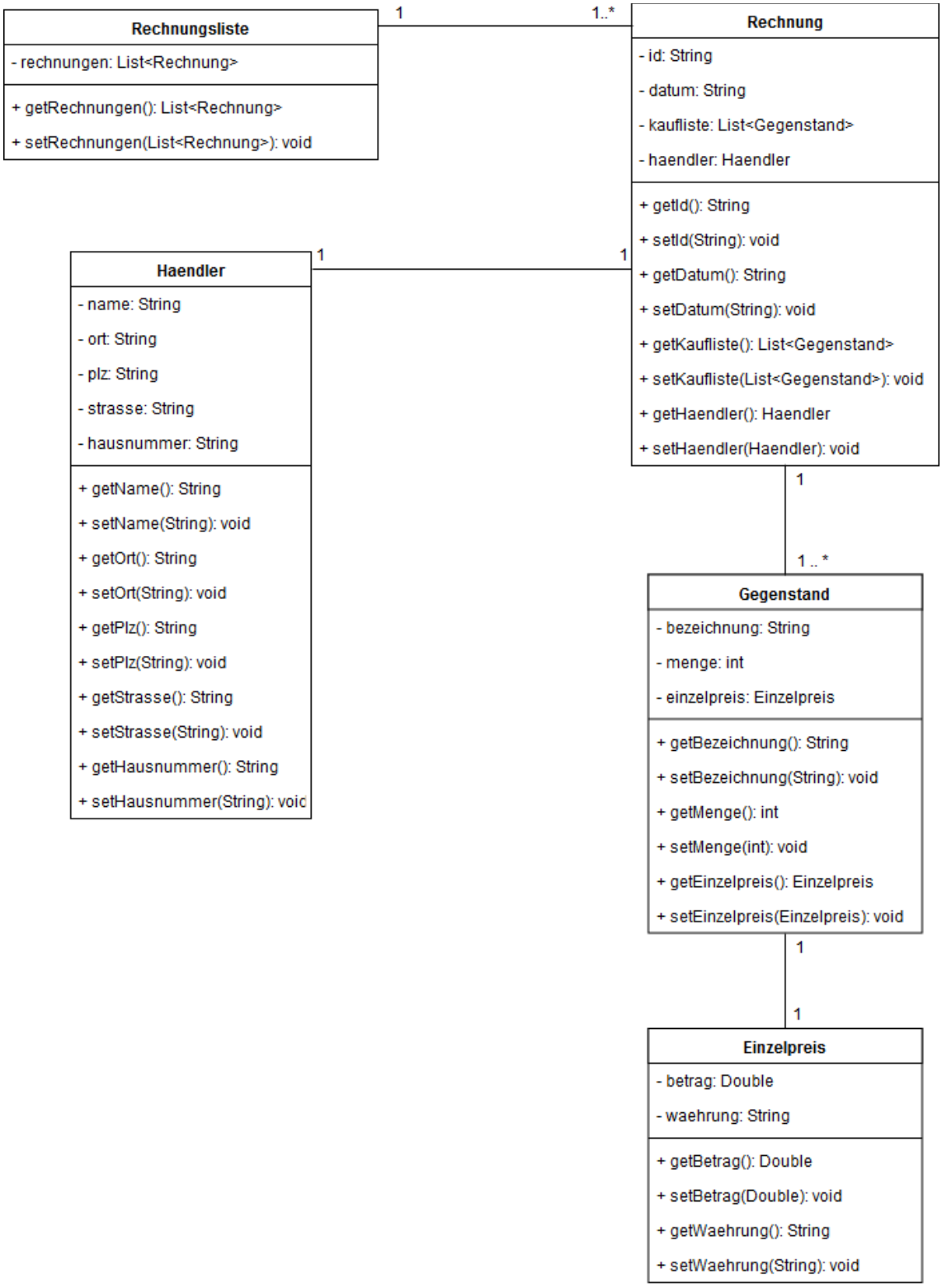
```
            "required": [
              "bezeichnung",
              "menge",
              "einzelpreis"
            ],
            "additionalProperties": false
          },
          "minItems": 1,
          "uniqueItems": true
        },
        "haendler": {
          "type": "object",
          "properties": {
            "name": {
              "type": "string"
            },
            "ort": {
              "type": "string"
            },
            "plz": {
              "type": "string"
            },
            "strasse": {
              "type": "string"
            },
            "hausnummer": {
              "type": "string"
            }
          },
          "required": [
            "name",
            "ort",
            "plz",
            "strasse",
            "hausnummer"
          ],
          "additionalProperties": false
        }
      },
      "required": [
        "id",
        "datum",
        "kaufliste",
        "haendler"
      ],
      "additionalProperties": false
    }
  }
},
"required": [
 "rechnungen"
],
"additionalProperties": false
}
```

Anlage 2: XSD für Elektronikrechnungen

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">

<!--Hauptelement: Rechnungsliste fuer Rechnungselemente-->
<xs:element name="rechnungsliste" type="rechnungslistentyp">
    <!--Attribut "id" der Rechnungselement eindeutig setzen-->
    <xs:unique name="eindeutigeIDs">
        <xs:selector xpath="rechnung"/>
        <xs:field xpath="@id"/>
    </xs:unique>
</xs:element>

<!--Typendefinition fuer die Rechnungsliste-->
<xs:complexType name="rechnungslistentyp">
    <xs:sequence>
        <xs:element name="rechnung" type="rechnungstyp"
            maxOccurs="unbounded"/>
    </xs:sequence>
</xs:complexType>

<!--Typendefinition fuer ein Rechnungselement-->
<xs:complexType name="rechnungstyp">
    <xs:sequence>
        <!--Kaufdatum-->
        <xs:element name="kaufdatum" type="xs:date" maxOccurs="1"/>

        <!--Kaufliste-->
        <xs:element name="kaufliste" type="kauflistentyp"
            maxOccurs="1"/>
        <!--Haendler-->
        <xs:element name="haendler" type="haendlertyp" maxOccurs="1"/>
    </xs:sequence>

   <!--ID der Rechnung als Attribut-->
    <xs:attribute name="id" type="idtype"/>
</xs:complexType>

<!--Typendefinition fuer die Kaufliste-->
<xs:complexType name="kauflistentyp">
    <xs:sequence>
        <xs:element name="gegenstand" type="gegenstandstyp"
            maxOccurs="unbounded"/>
    </xs:sequence>
</xs:complexType>

<!--Typendefinition fuer Gegenstaende-->
<xs:complexType name="gegenstandstyp">
    <xs:sequence>

      <!--Bezeichnung-->
        <xs:element name="bezeichnung" type="xs:string"
            maxOccurs="1"/>

      <!--Menge-->
      <xs:element name="menge" type="xs:positiveInteger"
            maxOccurs="1"/>
```

```
        <!--Einzelpreis-->
        <xs:element name="einzelpreis" type="einzelpreistyp"
                maxOccurs="1"/>
    </xs:sequence>
</xs:complexType>

<!--Typendefinition fuer Einzelpreis-->
<xs:complexType name="einzelpreistyp">
    <xs:sequence>

        <!--Betrag-->
          <xs:element name="betrag" type="betragstyp" maxOccurs="1"/>

        <!--Waehrung-->
        <xs:element name="waehrung" type="waehrungstyp" maxOccurs="1"/>
    </xs:sequence>
</xs:complexType>

<!--Typendefinition fuer Haendleradressen-->
<xs:complexType name="haendlertyp">
    <xs:sequence>
        <xs:element name="name" type="xs:string" maxOccurs="1"/>
        <xs:element name="ort" type="xs:string" maxOccurs="1"/>
        <xs:element name="plz" type="xs:string" maxOccurs="1"/>
        <xs:element name="strasse" type="xs:string" maxOccurs="1"/>
        <xs:element name="hausnummer" type="xs:string" maxOccurs="1"/>
    </xs:sequence>
</xs:complexType>

<!--Dimensionen des Betrags-->
<xs:simpleType name="betragstyp">
    <xs:restriction base="xs:decimal">
      <xs:minExclusive value="0"/>
      <xs:fractionDigits value="2"/>
    </xs:restriction>
</xs:simpleType>

<!--Enumeration fuer Waehrungen-->
<xs:simpleType name="waehrungstyp" final="restriction">
    <xs:restriction base="xs:string">
        <xs:enumeration value="euro"/>
        <xs:enumeration value="usd"/>
        <xs:enumeration value="gpb"/>
    </xs:restriction>
</xs:simpleType>

<!--Muster fuer Rechungs-ID-Attribute-->
<xs:simpleType name="idtype" final="restriction">
    <xs:restriction base="xs:token">
        <xs:pattern value="[0-9]{3}"/>
    </xs:restriction>
</xs:simpleType>
</xs:schema>
```

Anlage 3: Klassendiagramm für Elektronikrechnungen



Anlage 4: HTML für die Anzeige von Elektronikrechnungen

```
<?xml version="1.0" encoding="ISO 8859-1"?>
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <!--Laedt das JavaScript fuer die Verarbeitung von JSON-->
    <script type="text/javascript" src="JSONParser.js"></script>

    <!--Einige Festlegung fuer den Style-->
    <style>
      table {border-collapse: collapse;}
      table, th, td {border: 0.1em solid black;}
      td {min-width: 100px; padding: 0.3em;}
    </style>

  </head>
  <body>

    <!--Inputfelder fuer Anzeige-Button und ID-Eingabefeld-->
    <p>
      <input id="rechnung_anzeigen" type="button" value="Rechnung
anzeigen" onclick="verarbeiteJSON()"/>
      <input id="angabe_rechnungsid" type="text"
placeholder="Rechnungs-ID angeben"/>
    </p>

    <!--Tabelle fuer die Anzeige von Rechnungsdaten-->
    <table>
      <tr>
        <td><b>Kaufdatum</b></td>
        <td id="td_kaufdatum"></td>
      </tr>
      <tr>
        <td><b>Kaufliste</b></td>
        <td id="td_kaufliste">
          <ul id="liste_gegenstaende"></ul>
        </td>
      </tr>
      <tr>
        <td><b>Haendler</b></td>
        <td id="td_haendler"></td>
      </tr>
    </table>

    <!--Informationsfeld-->
    <p id="infofeld"></p>

  </body>
</html>
```

Anlage 5: JavaScript für die Anzeige von Elektronikrechnungen

```
function ladeJSON(callback) {

   /* XMLHttpRequest wird als API fuer das Uebertragen
      von Daten ueber das HTTP-Protokoll verwendet */
   var httpObj = new XMLHttpRequest();

   /* Das Format des Dokuments wird auf JSON festgelegt */
   httpObj.overrideMimeType("application/json");

   /* Die Inhalte des JSON-Dokuments "Rechnungen.json"
      werden mit einem HTTP-GET Request geholt. Dazu
      wird Dateiname (Pfadangabe moeglich) angegeben */
   httpObj.open('GET', 'Elektronikrechnungen_Daten.json');
    /* Die Inhalte werden unter open() standardmaessig
      im Asynchronen Modus eingelesen und es muss somit
       eine Callback-Funktion definiert werden */
    httpObj.onreadystatechange = function() {

      /* Die 4 signalisiert den Abschluss der Operation open()
         Die 200 ist ein HTTP-Statuscode ("200 = OK")
           Gibt httpObj 4 und 200 an, dann war der Erhalt
           der JSON-Inhalte erfolgreich */
      if (httpObj.readyState == 4 && httpObj.status == "200") {
         console.log("Laden erfolgreich");
         /* Die Callback-Funktion wird aufgerufen
            und uebergibt die JSON-Inhalte */
         callback(httpObj.responseText);
      }
    }
    httpObj.send(null);
}

function verarbeiteJSON() {
   /* Die Funktion loadJSON() wird aufgerufen
      und loest einen Callback nach einem erfolgreichen
      Ladevorgang aus. Der Inhalt der geladenen JSON-Datei
      wird uebergeben */
   ladeJSON(function(jsoninhalt) {

      /* Die JavaScript Standardfunktion JSON.parse
          verarbeitet den JSON-Inhalt und erzeugt ein
         JSON-Objekt */
      jsonObj = JSON.parse(jsoninhalt);
       findeIndexVonRechnungDurchID(jsonObj);


    });
}

function findeIndexVonRechnungDurchID(jsonObjekt) {
   /* RechnungsID aus dem Eingabefeld der HTML-Seite holen */
   var gesuchteID = document.getElementById("angabe_rechnungsid").value;

   /* Inhalte der Tabellenzellen der HTML-Seite zuruecksetzen */
   document.getElementById("liste_gegenstaende").innerHTML = "";
   document.getElementById("infofeld").innerHTML = "";
   document.getElementById("td_kaufdatum").innerHTML = "";
   document.getElementById("td_haendler").innerHTML = "";

   /* jedes Rechnung der Rechnungsliste durchlaufen */
   for (let i = 0; i<jsonObjekt.rechnungen.length; i++) {
```

```
        /* Wenn die ID einer Rechnung mit der Eingabe in HTML
         * uebereinstimmt, dann wird diese Rechnung behandlet */
        if (jsonObjekt.rechnungen[i].id == gesuchteID) {

            /* Datum in Tabellenzelle setzen */
            document.getElementById("td_kaufdatum").innerHTML =
                jsonObjekt.rechnungen[i].datum;

            /* Kaufliste anzeigen als HTML-Liste */
            for (let j = 0; j<jsonObjekt.rechnungen[i].kaufliste.length; j++) {
               document.getElementById("liste_gegenstaende").innerHTML +=
                  "<li>" +
                  jsonObjekt.rechnungen[i].kaufliste[j].menge + "x " +
                  jsonObjekt.rechnungen[i].kaufliste[j].bezeichnung + ", " +
                  jsonObjekt.rechnungen[i].kaufliste[j].einzelpreis.betrag +" "+
                  jsonObjekt.rechnungen[i].kaufliste[j].einzelpreis.waehrung+" "+
                  "</li>";
            }

            /* Haendler in verkuerzter Form anzeigen */
            document.getElementById("td_haendler").innerHTML =
               jsonObjekt.rechnungen[i].haendler.name + ", " +
                        jsonObjekt.rechnungen[i].haendler.ort;
            break;
        }

        /* Rechnung mit dieser ID nicht vorhanden */
        if (i == jsonObjekt.rechnungen.length-1) {
            document.getElementById("infofeld").innerHTML = "Es gibt keine
                Rechnung mit dieser ID";
        }
    }
}
```

Eigenständigkeitserklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig und nur unter Verwendung der angegebenen Literatur und Hilfsmittel angefertigt habe. Stellen, die wörtlich oder sinngemäß aus Quellen entnommen wurden, sind als solche kenntlich gemacht. Diese Arbeit wurde in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegt.

_____

Ort, den TT. Monat JJJJ Vorname Nachname