Titel der Arbeit
ggf. etwas länger

Project Thesis

vorgelegt am 12. November 2021

Fakultät Wirtschaft

Studiengang Wirtschaftsinformatik

Kurs ...

von

Tim Struthoff

Betreuer in der Ausbildungsstätte:

⟨ Name des Unternehmens ⟩
⟨ Titel, Vorname und Nachname der Betreuerin ⟩
⟨ Funktion der Betreuerin/des Betreuers ⟩

Unterschrift der Betreuerin/des Betreuers

DHBW Stuttgart:

⟨ Titel, Vorname und Nachname ⟩
⟨ der/des wissenschaftlichen Betreuerin/Prüferin ⟩

LaTeX-Vorlage für Projekt-, Seminar- und Bachelorarbeiten

Bei dem vorliegenden Dokument handelt es sich um eine Vorlage, die für Projekt-, Seminar- und Bachelorarbeiten im Studiengang Wirtschaftsinformatik der DHBW Stuttgart verwendet werden kann.

Sie setzt die technischen Vorgaben der Zitierrichtlinien[1] des Studiengangs (Stand: 01/2020) um.

Hinweise: Bitte lesen Sie sich die Zitierrichtlinien unbedingt genau durch. Dieses Dokument ersetzt keine Anleitung oder Einführung in LaTeX, für die Nutzung sind daher gewisse Vorkenntnisse unerlässlich. Ein Einstieg in LaTeX ist aber weniger schwierig, als es vielleicht auf den ersten Blick scheint und lohnt sich für das Verfassen wissenschaftlicher Arbeiten in jedem Fall.[2] Als Hilfestellung beim Schreiben eines Dokuments habe ich einen zweiseitigen kompakten LaTeX-Spickzettel erstellt, der über Moodle verfügbar ist.

Ihre Rückmeldungen und Anregungen zu dieser Vorlage nehme ich gerne per E-Mail an die Adresse `tobias.straub@dhbw-stuttgart.de` entgegen.

— Prof. Dr. Tobias Straub

Versionshistorie

1.0	05.02.2015	erste Fassung
1.1	16.02.2015	siehe **??**
1.2	20.04.2015	siehe **??**
1.3	20.02.2016	siehe **??**
1.4	24.07.2017	siehe **??**
1.5	07.01.2018	siehe **??**
1.6	07.04.2018	siehe **??**
1.7	12.02.2019	siehe **??**
1.8	10.02.2020	siehe **??**

[1] Sie finden diese unter „Prüfungsleistungen“ im Studierendenportal (`https://studium.dhbw-stuttgart.de/winf/pruefungsleistungen/`).

[2] so auch `http://www.spiegel.de/netzwelt/tech/textsatz-keine-angst-vor-latex-a-549509.html`

Inhaltsverzeichnis

Abkürzungsverzeichnis

Ein Abkürzungsverzeichnis ist optional. Das Paket `acronym` kann weit mehr, als hier gezeigt.[3] Beachten Sie allerdings, dass Sie die Einträge selbst in sortierter Reihenfolge angeben müssen.

CRM Customer Relationship Management

Ergänzende Bemerkung: Eine im Text verwendete Abkürzung sollte bei ihrer ersten Verwendung erklärt werden. Falls Sie sich nicht selbst darum kümmern möchten, kann das das Paket `acronym` übernehmen und auch automatisch Links zum Abkürzungsverzeichnis hinzufügen. Dazu ist an allen Stellen, an denen die Abkürzung vorkommt, `\ac{ITIL}` zu schreiben.

Das Ergebnis sieht wie folgt aus:

- erstmalige Verwendung von `\ac{ITIL}` ergibt: **ITIL!** (**ITIL!**),
- weitere Verwendung von `\ac{ITIL}` ergibt: **ITIL!**

Wo benötigt, kann man mit dem Befehl `\acl{ITIL}` wieder die Langfassung ausgeben lassen: **ITIL!**.

Falls man die Abkürzungen durchgängig so handhabt, kann man durch Paket-Optionen (in `_dhbw_praeambel.tex`) erreichen, dass im Abkürzungsverzeichnis nur die tatsächlich verwendeten Quellen aufgeführt werden (Option: `printonlyused`) und zu jedem Eintrag die Seite der ersten Verwendung angegeben wird (Option: `withpage`).

[3] siehe `http://ctan.org/pkg/acronym`

Abbildungsverzeichnis

Tabellenverzeichnis

1 Einleitung

Die Hewlett Packard Enterprise (HPE) Business Unit Aruba Networks bietet neben anderen Netzwerkgeräten auch Wireless Local Area Network (WLAN) Zugangspunkte bzw. Access Points (APs) an. Mit diesen Zugangspunkten ist es möglich, Endgeräte drahtlos an ein Netzwerk anzubinden[4]. Ein Kunde der Aruba Networks setzt mehrere hundert dieser APs ein. Er verwaltet alle seine IT Geräte, also auch die Aruba APs, mit einer Inventarisierungssoftware. Mit dieser Software sollen nun periodisch Inventardaten zu den APs abgerufen werden. Zur Einsparung von Arbeitskosten soll das Abrufen dieser Daten automatisiert werden. Die Aruba APs werden zentral über den SNMP basierten cloud-managed Service Aruba Central Cloud verwaltet. Die Aruba Central Cloud stellt eine sog. Representational State Transfer (REST) Schnittstelle zur Verfügung, über die Statusdaten der APs abgerufen werden können. Ein JavaScript Programm soll sich nun mit der Aruba REST Schnittstelle verbinden und so die Statusdaten der APs automatisiert abrufen. Somit fungiert das in dieser Arbeit konzipierte Programm als eine Middleware zwischen Aruba Cloud Central und der Inventarisierungssoftware des Kunden.

1.1 Zielsetzung

Das betriebliche, praktische Ziel dieser Arbeit stellt die Umsetzung eines Prototyps für die zuvor gezeigte Middleware dar. Das theoretische, bzw. akademische Ziel dieser Arbeit liegt in der formalisierten Aufbereitung des Recherche- und Implementierungsprozesses für die Interaktion der Middleware mit der REST API. Konkret sollen folgende **Forschungsfragen** beantwortet werden:

1. Wie ist der Representational State Transfer Architekturstil von Roy Fielding aufgebaut?
2. Wie kann überprüft werden, ob eine API den REST Prinzipien entspricht?
3. Entspricht die API der Managementsoftware Aruba Cloud Central den Grundsätzen von Roy Fieldings REST Architektur und werden auch andere Empfehlungen aus Fachliteratur beachtet?
4. Kann sich ein JavaScript Programm mit der Aruba Schnittstelle verbinden und mittels REST HTTP Abfragen Informationen über Aruba Geräte in der Netzwerkumgebung des Kunden sammeln?

Zusammengefasst liegt das Ziel der Arbeit in der Aufbereitung, Analyse und examplarischen Anwendung des REST Architekturstils am Beispiel der Aruba Central API.

[4] Vgl. LP 2021

1.2 Verwandte Arbeiten

Zu dem REST Architekturstil wurden bereits zahlreiche Arbeiten veröffentlicht. Speziell der Analyse von REST APIs haben sich die folgenden Autoren gewidmet. Bei allen unten aufgeführten Werken lag der Fokus jedoch nicht auf einer speziellen API oder gar einem Endpunkt, sondern in der Analyse einer Vielzahl von APIs.

- **Renzel et. al** untersuchen in einer Arbeit von 2012 eine Reihe von populären REST Webservices hinsichtlich ihrer Konformität mit Best Practices[5].
- **Maleshkova et. al** analysieren die Dokumentation von 222 APIs und ziehen Rückschlüsse auf gängige Beschreibungsformen, Ausgabetypen, die Verwendung von API-Parametern sowie weiteren Merkmalen[6]. In einer anderen Arbeit werden von Maleshkova 45 APIs von der Website ProgrammableWeb analysiert[7].
- **Neumann et al** sammelt Regeln zum Aufbau von REST APIs und analysiert 500 APIs der Alexa.com 4000 populärsten Websites. Neumann et al referenzieren in der Auswahl der Best Practices Renzel, Maleshkova 2014 und Rodriguez et. al 2016.
- In **Rodriguez et al 2016** wurden 78 GB internet traffic der Telecom Italia analysiert. Die Autoren stellen einige Regeln für REST APIs zusammen und analysieren den Traffic hinsichtlich der Konformität mit diesen Regeln[8].

1.3 Forschungsbeitrag

Der akademische Mehrwert dieser Arbeit begründet sich aus dem Zusammenstellen von konkreten Handlungsrichtlinien zur Implementierung von REST APIs aus vorhandener Literatur und der exemplarischen Anwendung dieser Regeln auf die Aruba Central API. Durch die exemplarische Anwendung dieses heuristischen Regelwerks wird das Regelwerk einerseits validiert und andererseits ein Beispiel gegeben, welches es zukünftigen Anwendern ermöglicht, die Regeln besser anzuwenden. Abschließend wird die Relevanz der Regeln mit einem Prototyp validiert. Zusammengefasst ist es Lesern dieser Arbeit möglich, REST APIs zu klassifizieren und auf Basis dieser Einordnung eine Entscheidung zu der Machbarkeit der Implementierung von Software mit der API zu treffen. Auch ist die Arbeit als firmeninternes Referenzwerk gedacht.

In der Problemdomäne ist bereits eine Vielzahl an Regelwerken zu REST APIs vorhanden. Diese Arbeit unterscheidet sich jedoch von vorhandener Literatur in ihrer Spezifität zu dem betrieblichen Kontext der Aruba Networks, sowie der Validierung der Forschungsergebnisse mittels eines Prototyps.

[5] Vgl. Renzel/Schlebusch/Klamma 2012
[6] Vgl. Maleshkova/Pedrinaci/Domingue 2010
[7] Vgl. Bülthoff/Maleshkova 2014
[8] Vgl. Rodríguez u. a. 2016

Neben der positiven Eingrenzung des Themenbereichs dieser Arbeit muss auch eine negative Abgrenzung stattfinden: Der Fokus dieser Arbeit liegt allein auf dem REST Architekturstil mit der konkreten Anwendung im HTTP Protokoll. Es werden keine alternativen Architekturstile in Betracht gezogen. Auch können die, in dieser Arbeit gewonnenen Ergebnisse nicht als absolute, unlimitiert gültige Wahrheit gesehen werden. Die Umsetzung des REST Stils entwickelt sich dynamisch und ist weitgehend frei von regulatorischer Kontrolle. So können die hier erlangten Ergebnisse lediglich als eine Heuristik auf Basis von aktueller, verbreiteter Literatur und etablierter Unternehmenspraxis gesehen werden. Eine weitere Einschränkung der Arbeit liegt in der Analyse von lediglich einem Endpunkt der Aruba Central API. Zwar ist anzunehmen, dass alle anderen Endpunkte der API ähnlich zu dem hier analysierten sind, jedoch kann dies aufgrund des beschränkten Umfangs der Arbeit nicht validiert werden. Zusätzlich liegt der Fokus bei der Umsetzung des Prototyps nicht auf konkreten Implementierungsdetails sondern auf dem Beweis der Machbarkeit einer Integration mit der Aruba Central API.

1.4 Aufbau der Arbeit

Zunächst wird das Thema in Kapitel 2 aus theoretischer Sicht beleuchtet. Zur Beantwortung der ersten Forschungsfrage nach dem Aufbau einer REST API wird in Kapitel 2.1 vorhandene Fachliteratur ausgewählt und die Auswahl begründet. In Kapitel 2.2 wird ein Grundwortschatz aus Fachbegriffen vorgestellt. Auf diesen Wortschatz wird in der weiteren Arbeit zurückgegriffen. Weiter wird in Kapitel 2.3 der REST Architekturstil in seiner Ausführung von Fielding erläutert. Zur späteren Klassifizierung der Aruba Central REST API wird in Kapitel 2.4 ein Gütemodell von Richardson um weitere Quellen erweitert und so ein Referenzmodell geschaffen.

Im Praxisteil (Kapitel 3) der Arbeit werden die erlangten Erkenntnisse angewendet und so validiert. Zunächst wird die betriebliche Problemstellung erläutert (Kapitel 3.1). Dann wird der Aruba Central Monitoring Endpunkt analysiert (Kapitel 3.2) und in das in Kapitel 2.4 erstellte Referenzmodell eingeordnet. Diese Einordnung soll eine grobe Einschätzung der Machbarkeit des Middleware Programmes ermöglichen. Das Analyseergebnis wird in Kapitel 3.3 mittels eines Prototyps validiert: Wurde die API in Kapitel 3.2 als RESTful eingestuft, sollte die Umsetzung des Prototyps möglich sein.

Im Fazit der Arbeit werden die erlangten Forschungsergebnisse zusammengefasst (Kapitel 4) und kritisch hinterfragt (Kapitel 4.1). Abschließend wird in Kapitel 4.2 ein Ausblick auf weitere Entwicklungen der Arbeit gegeben.

2 Theoretische Betrachtung des REST Architekturstils

Bei der Umsetzung einer Middleware zur REST Kommunikation stellt sich zunächst die erste Forschungsfrage nach dem Aufbau einer solchen REST Kommunikation. Der REST Architekturstil wird im Folgenden aus der Primärliteratur von Roy Fielding sowie anderer Fachliteratur herausgearbeitet und abstrahiert zusammengefasst:

2.1 Quellendiskussion

Der REST Architekturstil wurde von Roy Fielding erfunden und zuerst in Fieldings Dissertation *"Architectural Styles and the Design of Network-based Software Architectures"* beschrieben[9]. Somit kann die Dissertation von Fielding als erste und wichtigste Primärquelle angesehen werden[10]. Der REST Architekturstil ist eine Abstraktion des ebenfalls zu großen Teilen von Fielding konzipierten HTTP Protokolls[11]. Somit sollte das HTTP Protokoll in der Fassung von 1999 auch als Primärquelle aufgenommen werden. Auch andere Request For Comments (RFC) der IETF werden als Primärquelle angesehen.

Die Arbeit von Fielding befasst sich nicht mit der Umsetzung der Regel des REST Stils und auch nicht mit der Priorisierung dieser Regeln[12]. Diese beiden Aspekte werden jedoch für diese Arbeit benötigt. Für diese Informationen wird auf Sekundärliteratur zurückgegriffen.

2.1.1 Quellen zur konkreten Umsetzung des REST Architekturstils

Es wurde eine Vielzahl von Quellen untersucht. Diese Quellen sind in Abbildung 1 aufgeführt.

Die Kreise stellen die einzelnen Quellen dar. Ein Pfeil von Quelle A zu Quelle B symbolisiert eine Referenz von Quelle A auf Quelle B. Alle Quellen beziehen sich zusätzlich zu den hier aufgeführten Referenzen auf Roy Fieldings Doktorarbeit. Zur weiteren Synthese der Quellen wurde versucht, die Qualität der Quellen zu beurteilen. Alle als qualitativ beurteilten Quellen sind in der Abbildung in grün dargestellt. Diese werden in der folgenden Arbeit hauptsächlich verwendet. Die Beurteilung der Quellen wird im Folgenden begründet:

Zunächst werden die Quellen von *Richardson* von 2007, 2008 und 2013 betrachtet: Alle Quellen werden vielfach zitiert. Da andere Autoren ebenfalls auf die Auswahl hochwertiger Quellen achten sollten, lässt sich annehmen, dass eine viel zitierte Quelle von hoher Qualität ist. Gerade das Lehrbuch Richardson/Ruby 2007 wurde besonders oft zitiert. Zusätzlich sind Richardsons Werke

[9] Vgl. Tilkov 2015, S. 1
[10] Vgl. Richardson/Ruby 2007, S. XVI
[11] Vgl. Tilkov 2015, S. 1 und Richardson/Ruby 2007, S. XVII
[12] Vgl. Richardson/Ruby 2007, S. XVI

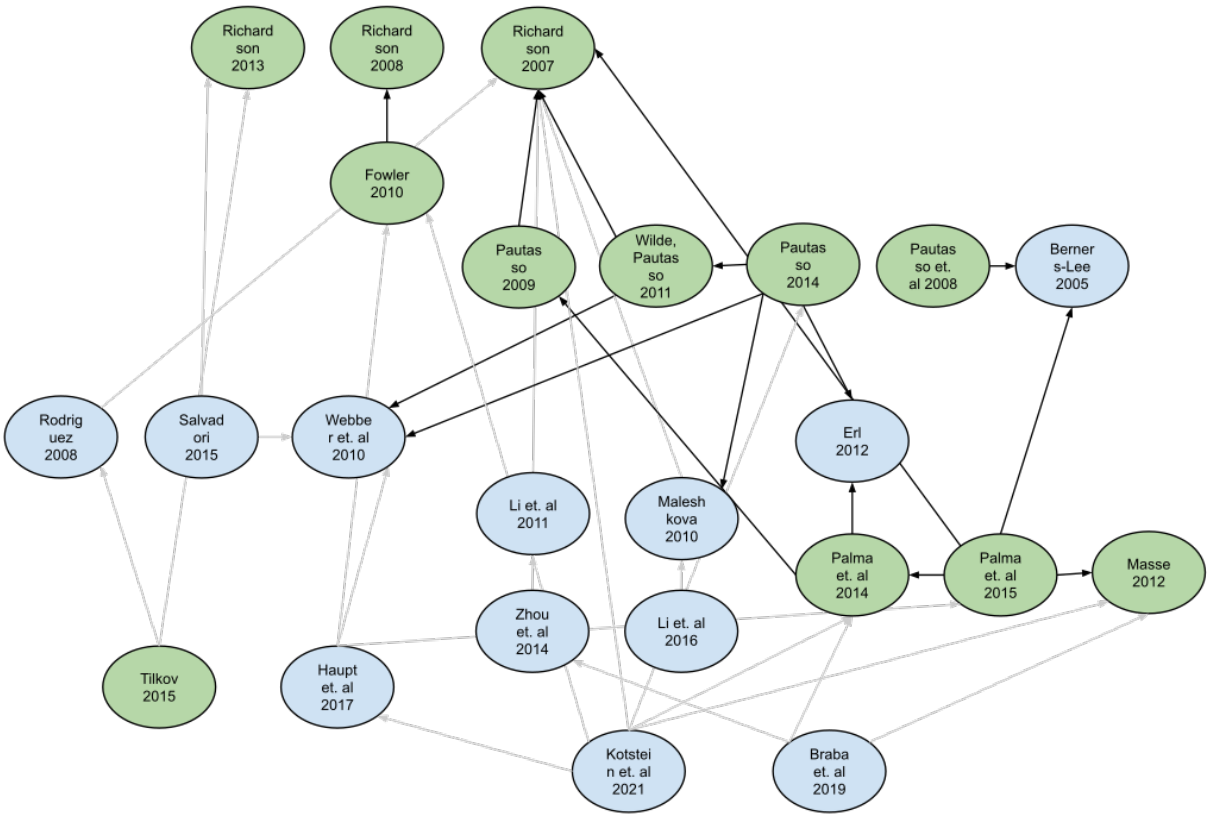

Abb. 1: Quelleneinordnung

bei dem 1978 gegründeten, anerkannten Verleger O'Reilly publiziert[13]. Weitergehend ist Leonard Richardson der Autor der populären Bibliothek Beautiful Soup zum Parsen von XML und HTML Dokumenten [14] und arbeitete bereits als Software Architekt bei der New York Public Library, sowie bei Canonical, der Firma hinter Ubuntu[15]. Insgesamt wurden seine Werke über 1500 mal zitiert[16].

Auch die Werke von Pautasso wurden vielfach zitiert[17]. Cesare Pautasso is full professor at the Software Institute of the Faculty of Informatics at the University of Lugano, Switzerland. Previously he was a researcher at the IBM Zurich Research Lab (2007) and a senior researcher at ETH Zurich (2004-2007). He completed his graduate studies with a Ph.D. from ETH Zurich in 2004 and his undergraduate studies at Politecnico di Milano, Italy with a Computer Science Engineering Degree (cum laude) in 2000[18].

Francis Palma wird ebenfalls als ein Spezialist auf seinem Fachgebiet angesehen und weißt einen h-Index von 12 auf[19]. Seine Werke werden unter kritischer Betrachtung ebenfalls als Quellen aufgenommen.

[13]Vgl. Levy 2005, S. 1
[14]Vgl. Richardson 2021a und Richardson 2021b
[15]Vgl. Richardson 2021d
[16]Vgl. Richardson 2021c
[17]Vgl. Pautasso, Cesare 2021a
[18]Vgl. Pautasso, Cesare 2021b
[19]Vgl. Palma 2021

Ein Lehrbuch von Mark Masse wird auch als Quelle aufgenommen. Masse verfügt über fünfzehn Jahre Erfahrung in Technik, Management und Architektur der Informatik und entwarf das Content Management System (CMS) von allen Websites der Walt Disney Company[20]. Das vorliegende Lehrbuch ist jedoch teils subjektiv gehalten und sollte somit besonders kritisch hinterfragt werden.

Eine weitere betrachtete Quelle ist ein Lehrbuch von Stefan Tilkov. Das Lehrbuch wurde aufgrund einer mündlichen Empfehlung der Bibliothek der Dualen Hochschule Baden-Württemberg (DHBW) sowie der Universität Stuttgart ausgewählt. Auch ist Stefan Tilkov Berater bei INNOQ, Autor zahlreicher Fachartikel und Referent auf zahlreichen internationalen Konferenzen[21].

2.1.2 Quellen zur Priorisierung der Rest Regeln

Johanna Barzen und Sebastian Kotstein erstellten eine Umfrage unter Experten, welche REST Regeln am wichtigsten sind[22]. Die Meinungen dieser Experten werden auch in der Priorisierung der REST Regeln in dieser Arbeit berücksichtigt.

2.2 Definitionen und Begriffe

Fielding legte bei der formulierung seines REST Architekturstil einen besonderen Fokus auf die Definition und korrekte Verwendung von speziellen Begriffen. Diese werden im folgenden zusammengefasst.

2.2.1 Ressourcen und Repräsentationen

Da es sich bei REST um einen Stil zur Umsetzung von verteilten *Informations*systemen handelt, liegt es auf der Hand, zunächst die Modellierung von Informationen zu betrachten. Die Abstraktion einer jeden Information in REST stellt eine *Ressource* dar[23]. Zu jedem Abfragezeitpunkt entspricht eine *Ressource* einem bestimmten Set von *Ressourcen Repräsentationen* und/-oder *Ressourcen Metadaten*. *Ressourcen Repräsentationen* sind konkrete Daten in einer konkreten Darstellungsform, wie zum Beispiel ein Bild oder ein Text[24]. Die Metadaten beschreiben diese Daten, sodass sie angemessen verarbeitet werden können[25].

[20] Vgl. Massé, M. 2021
[21] Vgl. Tilkov 2021
[22] Vgl. Kotstein/Bogner 2021
[23] Vgl. Richardson/Amundsen 2013, S. 30 und Richardson/Ruby 2007, S. 81
[24] Vgl. Vgl. Fielding 2000, S. 88ff, Richardson/Ruby 2007, S. 91 und Richardson/Amundsen 2013, S. 30
[25] Vgl. Fielding 2000, S. 87

2.2.2 Verbindungselemente

Neben Ressourcen und Repräsentationen sieht die REST Architektur von Roy Fielding eine Reihe von Verbindungselementen vor. Er beschreibt Verbindungselemente als abstrakte und generalisierte Konstrukte zwischen den anderen Komponenten. Diese abstrakte Betrachtung der Verbindungselemente hat den Vorteil, dass die Komponenten des Netzwerkes, wie z.B. der Client und der Server keine konkreten Informationen über die Implementierung der Verbindungselemente benötigen, so lange sie ebenfalls dem einheitlichen Protokoll folgen. Der generische Aufbau eines Verbindungselementes wird in der folgenden Abbildung gezeigt:

Die Schnittstelle des Verbindungselementes ähnelt dem prozeduralen Aufruf, jedoch mit einigen wichtigen Unterschieden: Die Eingangsparameter bestehen aus Anfragekontrolldaten, einer Ressourcenkennung, die das Ziel der Anfrage angibt, und einer optionalen Repräsentation der Ressource. Die Out-Parameter bestehen aus Antwort-Kontrolldaten, optionalen Ressourcen-Metadaten und einer optionalen Darstellung. Aus abstrakter Sicht ist der Aufruf synchron, aber sowohl Eingangs- als auch Ausgangsparameter können als Datenströme übergeben und somit asynchron implementiert werden[26].

Die primären Verbindungselemente sind der Client und der Server. Der Unterschied zwischen beiden besteht darin, dass der Client eine Verbindungsanfrage initiiert und der Server diese Anfrage beantwortet. Die drei zuletzt genannten Verbindungselemente werden im Rahmen dieser Arbeit nicht weiter erläutert, da sie von geringer Relevanz für die implementierung der Middleware sind[27].

2.2.3 Komponenten

Ein Useragent verwendet das Client Verbindungselement, um Abfragen abzuschicken und Antworten zu empfangen. Ein Beispiel eines Useragents ist ein Web Browser oder, wie in dieser Arbeit die Middleware[28].

2.3 Der REST Architekturstil

Nach der Definition der Grundvokabeln kann nun der REST Architekturstil selbst beschrieben werden.

[26] Vgl. Fielding 2000, S. 92ff
[27] Vgl. Fielding 2000, S. 92ff
[28] Vgl. Fielding 2000, S. 96ff

2.3.1 Überblick über den REST Stil

Der REST Architekturstil ist eine grundlegende Methode zum Bau von verteilten Hypermedia-Informationssystemen[29]. Er stellte eine wesentliche Grundlage für das heutige Internet dar, da z.B. das HTTP Protokoll zusammen mit dem REST Stil entwickelt wurde[30]. Der amerikanische Informatiker Roy Fielding[31] konzipierte den REST Architekturstil in den frühen Jahren des Internets, um die anfänglichen Skalierungsprobleme[32] des damals ersten verteilten Informationssystems zu lindern[33]. Der Architekturstil wurde 1993 als Teil von Roy Fieldings Dissertation veröffentlicht. Das Ziel der Arbeit war es, das damalige Internet robuster und skalierbarer zu machen[34]. Nach Tilkov bildete Fielding die Grundprinzipien des damaligen Internets abstrahiert und optimiert ab. Neben anderen Anwendungsbereichen können auch HTTP-basierte Programmierschnittstellen (APIs) nach dem REST Stil entworfen werden. Diese Schnittstellen werden informell auch als RESTful bezeichnet[35].

2.3.2 Grundziele des REST Architekturstils

Bei der Konzeption des REST Architekturstils hat Roy Fielding die Anforderungen und Einschränkungen bestehender Internetsysteme betrachtet und diese abstrahiert zusammengefasst[36]. Er stellt die These auf, dass es möglich sei, durch eine fortschreitende Standardisierung des Internets dessen Zustand und Benutzerfreundlichkeit zu erhöhen[37]. Das Ziel seiner Dissertation und damit auch das Ziel von REST beschreibt er wie folgt:

> „REST provides a set of architectural constraints that, when applied as a whole, emphasizes scalability of component interactions, generality of interfaces, independent deployment of components, and intermediary components to reduce interaction latency, enforce security, and encapsulate legacy systems.“[38]

REST soll also eine Reihe an architekturellen Einschränkungen zur Verfügung stellen, die folgende sechs Ziele unterstützen:

1. Die Skalierbarkeit von Interaktionen zwischen Komponenten verbessern
2. Die Generalität von Schnittstellen
3. Unabhängige Deployments von Komponenten

[29] Vgl. Fielding 2000, S. 76
[30] Vgl. Fielding u. a. 1999 und Tilkov 2015, S. 9
[31] Vgl. Fielding 2021
[32] Erwähnt in Fielding 2000 und erklärt in Berners-Lee/Cailliau u. a. 1994
[33] Vgl. Fielding 2000, S. 1 und Tilkov 2015, S. 9
[34] Vgl. Fielding 2000, S. 3
[35] Vgl. Rodriguez 2008 und Tilkov 2015, S. 10
[36] Vgl. Fielding 2000, S. 76
[37] Vgl. Fielding 2000, S. 73f
[38] Vgl. Fielding 2000, S. 75

4. Eine Reduktion der Interaktions-Latenz
5. Das Verstärken der Sicherheit und
6. Das Einbetten älterer Systeme

Während diese Ziele ursprünglich zur Verbesserung des Webs gedacht waren, sind sie auch für Programmierschnittstellen auf HTTP Basis erstrebenswert.

2.3.3 Konstruktive Randbedingungen

Die zuvor genannten sechs Ziele der REST Architektur sollen durch eine Reihe von Einschränkungen in der Kommunikation zwischen Komponenten des Webs erreicht werden.

1. **Client-Server**: Zunächst stellt Fielding die Regel auf, dass jedes Internet System aus Clients und Servern bestehe. Server seien dabei für die Verwaltung der Daten zuständig und Clients stellen eine Art Schnittstelle für Interaktionen dar[39].
2. **Zustandslosigkeit**: Jede Anfrage des Clients an den Server müsse sämtliche Informationen zur Bearbeitung dieser beinhalten[40]. Laut Fielding, Tilkov und Richardson soll diese Regel die Kommunikation im Internet durchschaubarer, zuverlässiger und skalierbarer machen[41]. Richardson bezeichnet die Zustandslosigkeit als eines der vier wichtigsten Features seiner Resource Oriented Architecture, einer Erweiterung der REST Architektur[42] und auch Pautasso nimmt Zustandslosigkeit als ein Grundkriterium für REST auf[43].
3. **Caching** Zu jeder Ressource müsse angegeben werden, ob diese cachebar ist oder nicht. Caches könnten die Skalierbarkeit und Leistungsfähigkeit der Kommunikation signifikant erhöhen[44].
4. **Einheitliche Schnittstellen**: Alle Komponenten im Internet sollten laut Fielding das gleiche Interface zur Verfügung stellen[45]. Richardson unterstützt diese Regel, indem er die Regel der einheitlichen Schnittstellen in seine ROA aufnimmt[46]. Auch Pautasso unterstützt diese Annahme[47]. Durch diese Einschränkung wird eine lose Kopplung von Systemen erreicht, was wiederum die Flexibilität bei Änderungen am Gesamtsystem erhöht. Insgesamt verringere dies zwar die Effizienz der Kommunikation, stelle jedoch gleichzeitig auch sicher, dass eine möglichst große Anzahl an Teilnehmern an der Kommunikation partizipieren können[48].

[39] Vgl. Fielding 2000, S. 78
[40] Vgl. Tilkov 2015, S. 10
[41] Vgl. Fielding 2000, S. 78, Tilkov 2015, S. 4, Tilkov 2015, S. 119 und Richardson/Ruby 2007, S. 86
[42] Vgl. Richardson/Ruby 2007, S. 86
[43] Vgl. Pautasso, Cesare 2014, S. 3
[44] Vgl. Tilkov 2015, S. 4 und Tilkov 2015, S. 127
[45] Vgl. Fielding 2000, S. 81f
[46] Vgl. Richardson/Ruby 2007, S. 105
[47] Vgl. Pautasso, Cesare/Zimmermann/Leymann 2008, S. 3
[48] Vgl. Tilkov 2015, S. 2 und Fielding 2000, S. 81f

5. **Mehrschichtige Systeme**: Die sog. Mehrschichtigkeitseinschränkung besagt, dass die Nachrichten in der Kommunikation aus unabhängigen Schichten bestehen sollen. Jede Komponente im Netzwerk müsse nur mit einer Schicht interagieren. So könne die Funktionalität der Komponenten wiederum begrenzt werden. Der hierdurch entstehende Mehraufwand könne auch hier durch Caching beseitigt werden[49].

6. **Code-On-Demand**: Die letzte Einschränkung für Internet-Kommunikation ist optional. So könne es möglich gemacht werden, zusätzliche Programmteile für den Client zur Laufzeit in Form von Applets oder Skripten nachzuladen. Das erhöhe zwar die Flexibilität der Clients, hindere jedoch gleichzeitig auch die Visibilität der Kommunikation[50].

Neben diesen Randbedingungen stellt Fielding die zusätzliche Regel auf, dass Dokumente untereinander verbunden sein sollen[51]. Dies geschieht über sog. Hyperlinks. Durch diese Verlinkung der Dokumente wird die Zustandslosigkeit der Kommunikation verbessert[52]. Er nennt dieses Prinzip[53]. Neben der Zustandslosigkeit würde hierdurch die lose Kopplung von Server und Client unterstützt werden[54].

2.4 Klassifizierung von REST APIs mit Richardsons Gütemodell

Die Anwendung des REST Architekturstil ist keine binäre Entscheidung; der Stil kann auch inkrementell übernommen werden[55]. Mithilfe des Reifegradmodells von Richardson kann festgestellt, werden, inwiefern eine Schnittstelle mit dem REST Architekturstil übereinstimmt. Zwar steht auch eine Vielzahl von anderen Reifegradmodellen für REST APIs zur Verfügung, jedoch wurde in der Quellendiskussion Leonard Richardson als einflussreichster Autor auf diesem Gebiet identifiziert und somit wird sein Reifegradmodell vorgezogen. Das Modell wird auch in verschiedenen Arbeiten [56] von Pautasso referenziert.

Richardsons Reifegradmodell besteht aus vier Leveln (0 bis 3), welche jeweils aufeinander aufbauen. Also muss zur Erfüllung eines höheren Levels zunächst das niedrigere erreicht werden. Level 1 setzt z.B. die Erfüllung von Level 0 voraus. Gleichzeitig spricht ein höheres Level für eine bessere Konformität mit dem REST Stil.

In Tabelle 1 auf Seite 13 wird eine Übersicht über den Zusammenhang der zuvor genannten Quellen gegeben

[49] Vgl. Fielding 2000, S. 82f
[50] Vgl. Fielding 2000, S. 84f
[51] Vgl. Fielding 2000, S. 76
[52] Vgl. Pautasso, Cesare/Zimmermann/Leymann 2008, S. 807 und Palma/Dubois u. a. 2014, S. 86
[53] Vgl. Pautasso, Cesare 2014, S. 3
[54] Vgl. Tilkov 2015, S. 4
[55] Vgl. Pautasso, Cesare 2014, S. 4
[56] Vgl. Pautasso, Cesare/Zimmermann/Leymann 2008, S. 807 und Pautasso, Cesare 2014, S. 4

2.4.1 Level 0 — HTTP als Tunnel

Der Ausgangspunkt des Reifegradmodells ist die Nutzung von HTTP als Transportsystem für verteilte Interaktionen. Auf diesem Level werden keine der speziellen Features von HTTP verwendet und HTTP wird lediglich als Tunnel für eigene Kommunikationsmechanismen verwendet[57]. Durch die Verwendung von HTTP als Protokoll werden eine Reihe der konstruktiven Randbedingungen des REST Architekturstils bereits erfüllt: HTTP wird grundsätzlich als Client-Server System verwendet (erfüllt Randbedingung 1), zusätzlich ist es ein mehrschichtiges System (erfüllt Randbedingung 5) mit einheitlichen Schnittstellen (erfüllt Randbedingung 4).

2.4.2 Level 1 — Ressourcen

Level 1 führt als zusätzliche Einschränkung die Identifizierbarkeit von Ressourcen ein. Interaktionen mit einzelnen Ressourcen (Definition siehe Kapitel) müssen über separate URLs realisiert werden[58]. Die Trennung einzelner Ressourcen ist eine Grundvoraussetzung für das Caching (erfüllt Randbedingung 3). In einem anderen Lehrbuch bezeichnet Richardson die Differenzierung von Ressourcen als Scoping Information und besagt, dass dies eine Grundvoraussetzung für REST sei[59]. Auch Tilkov ist der Meinung, dass die eindeutige Identifikation von Ressourcen die wichtigste Grundvoraussetzung für REST ist[60]. Er beruft sich dabei auf einen Blog Artikel von David Megginson[61]. Zusätzlich wird diese Annahme auch von Pautasso unterstützt[62].

Nach dem HTTP Protokoll sollen einzelne Ressourcen mit URLs identifiziert werden[63]. URLs stellen eine spezielle Art von Uniform Ressource Identifiern (URIs) dar[64]. Bei URLs wird der URI noch das Abfrageprotokoll hinzugefügt. In RFC 3986 werden die Bestandteile einer URI angegeben. Im Folgenden werden die einzelnen Bestandteile aufgelistet[65]:

1. Das Schema. In der Regel wird hier das Protokoll, wie z.B. https oder ftp angegeben.
2. Ein Doppelpunkt gefolgt von zwei Schrägstrichen (://)
3. Die Autorität. Dies ist in der Regel der Domainname des Server, sowie der Port.
4. Der Pfad zu der angefragten Ressource auf dem Server. Wird kein Pfad angegeben, wird automatisch der Root Pfad angesprochen.
5. Optional: Ein oder mehrere Query Parameter
6. Optional: Ein Fragment

[57] Fowler 2010, Vgl.
[58] Vgl. Fowler 2010, Richardson/Ruby 2007, S. 81, Berners-Lee 1996 und Pautasso, C. 2009, S. 22
[59] Vgl. Richardson/Ruby 2007, S. 79 und Richardson/Ruby 2007, S. 105
[60] Vgl. Tilkov 2015, S. 40
[61] Megginson 2007, Vgl.
[62] Vgl. Pautasso, Cesare 2014, S. 3
[63] Vgl. Fielding u. a. 1999, S. 18
[64] Berners-Lee/Fielding/Masinter 2005, Vgl.
[65] Vgl. Berners-Lee/Fielding/Masinter 2005, S. 6ff

Abbildung .. zeigt Beispielhaft den Aufbau einer URL[66].

Gemäß des Standards symbolisiert ein URI einen hierarchischen, gerichteter Graph [67]. Das folgende Beispiel aus Richardson 2007 verdeutlicht diesen Zusammenhang; Paris ist Teil von Frankreich, Frankreich Teil der Erde[68].

http://maps.example.com/Earth/France/Paris

2.4.3 Level 2 — HTTP-Verben und Response Codes

Zusätzlich werden auf dem zweiten Level HTTP-Verben und Response-Codes eingeführt. Sie stellen einen uniformen Mechanismus zur Verfügung, um die Intention einer HTTP Anfrage und das Resultat in einer HTTP Antwort zu beurteilen [69]. In seinem Lehrbuch ordnet Richardson das Ausweisen der HTTP Methoden als eines der vier wichtigsten Grundprinzipien von RESTful Web Services ein[70]. Es erfülle Fieldings Prinzip der einheitlichen Schnittstelle[71].

Request Methoden zeigen die Intention den Abfrage. Zum Beispiel wird mit GET eine Intention zum Abrufen von Informationen ausgewiesen [72].

Response Codes stellen eine Kurzfassung der Antwort des Servers dar[73]. Der Code 200 ("OK") zeigt an, dass ein Server erfolgreich eine Antwort zurückgibt[74]. Für Fehler sind die Codes mit 3, 4 und 5 als erster Ziffer vorgesehen[75]. Weitere individuelle Codes können den Kapiteln 10.1.1 bis 10.5.6 des RFC 2616 entnommen werden.

Die Methoden und Response Codes sind in ihrer Bedeutung bei allen REST konformen Schnittstellen gleich. Somit stellen sie eine einheitliche Schnittstelle dar, sodass eine API mit diesen Request Methoden und Response Codes Randbedingung 4 von Fielding entspricht. Tilkov erwähnt Request Methoden und Response Codes direkt nach den Ressourcen und ordnet ihnen eine "zentrale Rolle"[76] zu. Mithilfe dieser beiden Begriffe kann sowohl die Intention als auch das Resultat einer Operation in ihren Grundzügen automatisiert erfasst werden[77]. Das HTTP Verb GET weist z.b. den Abruf von Informationen aus. Wenn diese Informationen sich nicht verändern, können Ressourcen aus der Antwort im Client oder anderen Komponenten zwischengespeichert werden[78].

[66] Vgl. Berners-Lee/Fielding/Masinter 2005, S. 16
[67] Vgl. Richardson/Ruby 2007, S. 118 und Palma/Gonzalez-Huerta u. a. 2017, S. 175
[68] Vgl. Richardson/Ruby 2007, S. 118
[69] Vgl. Haupt/Leymann/Pautasso, Cesare 2015, S. 15, Palma/Dubois u. a. 2014, S. 237 und Massé, M. H./Massé, M. 2012, S. 23
[70] Vgl. Richardson/Ruby 2007, S. 105
[71] Vgl. Richardson/Ruby 2007, S. 79
[72] Vgl. Richardson/Ruby 2007, S. 6 und Massé, M. H./Massé, M. 2012, S. 28
[73] Vgl. Palma/Dubois u. a. 2014, S. 237 und Massé, M. H./Massé, M. 2012, S. 28
[74] Vgl. Richardson/Ruby 2007, S. 137
[75] Vgl. Richardson/Ruby 2007, S. 139
[76] Vgl. Tilkov 2015, S. 53
[77] Vgl. Richardson/Amundsen 2013, S. 36
[78] Vgl. Fowler 2010 und Richardson/Ruby 2007, S. 219

Richardson	**Fielding**	**Weitere Nachweise**
Level 0: HTTP als Transportmechanismus	1 Client-Server / 4 Einheitliche Schnittstellen / 5 Mehrschichtiges System	Fielding 2000
Level 1: Ressourcen	3 Caching	Fowler 2010 / Richardson/Ruby 2007, S. 79 / Berners-Lee 1996 / Richardson/Ruby 2007, S. 105 / Tilkov 2015, S. 40 Megginson 2007 / Pautasso, Cesare 2014, S. 3 / Fielding u. a. 1999
Level 2: Request Methoden und Response Codes	4 Einheitliche Schnittstellen	Pautasso, C. 2009, S. 15 / Palma/Dubois u. a. 2014, S. 237 / Massé, M. H./Massé, M. 2012, S. 23 / Tilkov 2015, S. 53 / Richardson/Ruby 2007
Level 3: Hypermedia	2 Zustandslosigkeit	Fowler 2010 / Richardson 2008 / Pautasso, Cesare 2014 / Richardson/Ruby 2007 / Tilkov 2015

Tab. 1: Einordnung von Sekundärliteratur in das Richardson Maturity Model

2.4.4 Level 3 — Hypermedia Controls

Level 3 des Richardson Gütemodells führt Hypermedia As The Engine Of State (HATEOS) ein. HATEOS besagt, dass eine Ressource Hypermedia-Referenzen zu anderen Ressourcen beinhalten soll[79]. Auch in Richardsons Lehrbuch bezeichnet er die "Connectedness"[80], sprich die Verbindung zwischen verschiedenen Ressourcen als Grundpfeiler seiner ROA. Diese Behauptung unterstützt er auch in einem weiteren Lehrbuch[81]. In Tilkov wird diese Randbedingung ebenfalls nach den HTTP Verben eingeordnet, was die Bedeutung von Richardsons Priorisierung weiter unterstützt[82]. Wie in Kapitel 2.3.3 beschrieben wurde wird mit HATEOS die Zustandslosigkeit und damit Randbedingung 2 von Fielding erfüllt[83].

[79] Vgl. Vgl. Fowler 2010, Vgl. Richardson 2008 und Vgl. Pautasso, Cesare 2014, S. 5
[80] Vgl. Richardson/Ruby 2007, S. 105
[81] Vgl. Richardson/Amundsen 2013, S. XV
[82] Vgl. Tilkov 2015, S. 75
[83] Vgl. Tilkov 2015, S. 75

3 Praxisteil

Im Folgenden werden die erlangten theoretischen Erkenntnisse praktisch angewendet und so validiert.

3.1 Problemstellung der betrieblichen Praxis

Im Rahmen dieser Arbeit soll für einen Kunden der Aruba Networks eine Middleware Anwendung realisiert werden. Diese Middleware soll der Inventarisierung von Teilen der Netzwerkumgebung des Kunden dienen.

3.1.1 Die Aruba Cloud Central API

Der Kunde verwendet in seiner Netzwerkumgebung WLAN APs der Aruba Networks. Diese APs stellen über das Internet eine Verbindung zu der zentralen Verwaltungs- und Orchestrationseinheit Aruba Central her. Aruba Central ist ein Softwareprodukt der Aruba Networks und bietet die Möglichkeit, eine große Anzahl an APs und anderer Netzwerkgeräte in einem einzigen visuellen Benutzerinterface zu verwalten. Dabei können die Geräte weltweit an verschiedenen Standorten verteilt sein. Aruba Central kann sowohl vor Ort (on-premise) installiert werden, als auch aus zentralen Rechenzentren bereitgestellt werden. Wichtige Features der Software sind u. a. die Möglichkeit, entferntes Netzwerkequipment über das Internet zu provisionieren, mit künstlicher Intelligenz Fehler zu finden und das Netzwerk aktiv zu überwachen. Vor allem das Überwachen, sprich Monitoring, ist in dieser Arbeit von Relevanz. Viele der Funktionen von Aruba Central lassen sich auch mittels einer Programmierschnittstelle bedienen. Für die Entwicklung der Middleware soll diese Schnittstelle verwendet werden. Die Schnittstelle wird als REST Schnittstelle bereitgestellt.

3.1.2 Integration der Middleware in den Ivanti Endpoint Manager

Der Kunde möchte die Inventardaten der APs in die Inventarisierungssoftware Ivanti Endpoint Manager importieren. Der Ivanti Endpoint Manager dient dem Inventarisieren, Verwalten, Schützen und Warten von IT Geräten.

Mit dem Kunden wurde die Vereinbarung getroffen, dass lediglich ein Prototyp der Middleware für die JavaScript Laufzeitumgebung V8 entwickelt wird und der Kunde diesen Prototyp selbst in die Software integriert. Die Software Ivanti Endpoint Manager findet aus diesem Grund keine weitere Betrachtung.

3.1.3 Anforderungen an die Middleware

Das praktische Ziel dieser Arbeit ist ein Middleware Skript zu entwickeln, welches sich mit der REST API von Aruba Cloud Central verbindet und Inventardaten über die Aruba Access Points abfragt. Die Daten soll das Skript dann in einer verkürzten Form ausgeben.

Im Folgenden werden nach ISO 25010 sowohl funktionale als auch nicht-funktionale Anforderungen an das Programm aufgestellt. Aufgrund einer langjährigen Vertrauensbasis mit dem Kunden konnten die Anforderungen informell festgelegt werden. So besteht keinerlei schriftliche Dokumentation der Anforderungen. Im Folgenden werden diese stichwortartig aufgelistet:

Funktionale Anforderungen

1. Das Skript soll sich mittels Kundendaten mit der Aruba Cloud Central API authentifizieren.
2. Daten über alle Access Points der Netzwerkumgebung sollen gesammelt werden. Diese Daten werden zunächst im Arbeitsspeicher zwischengespeichert.
3. Ein Auszug der Daten soll auf dem Bildschirm ausgegeben werden.

Nicht-Funktionale Anforderungen

1. Die Anwendung muss mit der Programmiersprache JavaScript umgesetzt werden und auf der JavaScript Runtime V8 ausführbar sein.
2. Sie soll umfangreich kommentiert, dokumentiert und so wenig komplex wie möglich sein.
3. Die Anwendung soll Plattformunabhängig sein.

3.2 Analyse des Access Point Monitoring API Endpunktes

Für das Umsetzen der Middleware muss ein API Endpunkt angesprochen werden. In mehreren Studien wurde jedoch festgestellt, dass in der Praxis viele APIs nicht komplett dem REST Architekturstil entsprechen. Dies könnte also auch bei der Aruba Central API der Fall sein. Da es für die Umsetzung der Middleware unabdinglich ist, dass die Aruba API REST konform ist, wird im Folgenden analysiert, ob die API den REST Regeln entspricht. Hierbei wird auf die in Kapitel definierten Regeln im Gütemodell von Richardson zurückgegriffen.

Zunächst ist es erforderlich, sich mit der API zu authentifizieren. Die Authentifizierung folgt dem OAuth2 Standard und erfordert drei Abfragen. Der genaue Ablauf der Authentifizierungsroutine ist auf einer öffentlichen Dokumentationsseite der Aruba Networks dokumentiert und wird hier nicht weiter betrachtet.

Nach der erfolgreichen Authentifizierung können Daten von der API abgerufen werden. Die Statusdaten der APs werden mit einem Endpunkt bzw. einer Ressource der API angeboten. Im Folgenden wird dieser Endpunkt genauer untersucht. Sowohl das Abrufen der Daten vom Endpunkt, als auch der Rückgabewert werden beschrieben und argumentativ deduktiv analysiert.

Für die Beschreibung des API Endpunktes wurden mit dem HTTP Client Postman HTTP Abfragen an die API gesendet. Das Resultat dieser Abfragen wurde gespeichert und dem Anhang dieser Projektarbeit beigefügt. Im Folgenden werden lediglich Ausschnitte der Abfrage betrachtet. Die Zeilennummern am linken Rand der Abfrage im Text verweisen dabei auf die Zeilennummern aus der ursprünglichen kompletten Abfrage im Anhang.

3.2.1 Abfrage des Monitoring Endpoints

Zunächst wird die Abfrage der Statusinformationen von der API betrachtet. Im Folgenden werden Auszüge der Abfrage als unveränderter Text gezeigt.

```
1 GET /monitoring/v2/aps HTTP/1.1
2 Host: internal-apigw.central.arubanetworks.com
```

Die einzelnen Bestandteile der Abfrage werden im Folgenden analysiert.

Protokollversion (Zeile 1): Am Ende von Zeile 1 wird angegeben, dass die Abfrage der Version 1.1 des HTTP-Protokolls entspricht. Diese Version des HTTP-Protokolls wird in RFC 2616 beschrieben. Stimmt der Rest der Abfrage ebenfalls mit HTTP/1.1 überein, ist Level 0 des Gütemodells von Richardson erfüllt.

Analyse der URL (Zeile 1 und 7): Aus Zeile 1 und Zeile 7 lässt sich die Uniform Ressource Locator (URL) der Ressource erkennen. Zusammengesetzt ist sie hier zu sehen:

https://internal-apigw.central.arubanetworks.com/monitoring/v2/aps

Nun wird untersucht, ob der Aruba Monitoring Endpunkt dem generellen Aufbau einer URL folgt. Hierzu wird versucht, die einzelnen Bestandteile der URL den generischen Bestandteilen von URLs zuzuweisen. Abbildung .. illustriert diese Zuordnung

In der Abbildung ist erkennbar, dass alle obligatorischen Elemente (1-4) syntaktisch korrekt vorhanden sind. Somit entspricht die URL des Endpunktes dem RFC 3986.

Der Aufbau der URL lässt auch auf die Struktur der API schließen. Alle Autoren der zuvor genannten Quellen sind sich einig, dass es möglich sein muss, Ressourcen eindeutig über URLs zu identifizieren . In der URL folgen die Begriffe monitoring, v2 und aps aufeinander. Nach Tilkov ergibt sich aus der URL eine natürliche, hierarchische Anordnung. Nach Hesse stellt monitoring eine Collection, sprich eine Sammlung von Elementen dar. Der Pfad-Bestandteil v2 ist ebenfalls eine Collection und ist ein Teil von monitoring. Zuletzt ist aps eine Ressource in der v2 collection. Aus der obigen URL ist somit eine klare Struktur der API erkennbar.

Analyse des HTTP Verbes (Zeile 1): In Zeile 1 ist zu erkennen, dass der Request Methoden Token GET ist.

```
1 GET /monitoring/v2/aps HTTP/1.1
```

Der Zweck des Endpunktes ist es, Informationen zu den APs im Netzwerk abzurufen. Es werden keine Informationen auf dem Server verändert und keine Aktionen ausgelöst. Laut RFC 2616 (Das HTTP Protokoll) ist genau für eine solche Operation die GET Methode sinnvoll. Durch adäquate Request Methoden kann der Zweck einer Operation geschlussfolgert werden. Zum Beispiel kann so gesagt werden, dass der vorliegende Endpunkt idempotent ist und theoretisch gecacht werden dürfte. Auch Hesse unterstützt die Aussage, dass GET in diesem Fall die richtige Request Methode ist.

Zusammengefasst konnten in der HTTP Abfrage eine korrekte Request Methode, eine korrekte URL, sowie HTTP/1.1 als Protokollversion festgestellt werden. Durch das Verwenden von HTTP besteht die Abfrage Level 0 des Gütemodells. Durch die korrekte URL wird Level 1 bestanden und durch den korrekten Pfad Level 2.

3.2.2 Antwort des Monitoring Endpoints

Auf die HTTP-Abfrage des Endpunktes sollte der Monitoring Endpunkt die Inventardaten der WLAN-APs zurückgeben. Der Rückgabewert der Abfrage ist im folgenden Code zu sehen.

```
12 HTTP/1.1 200 OK
13 Content-Type: application/json
14 Transfer-Encoding: chunked
15 Date: Sat, 31 Jul 2021 11:36:35 GMT
16 cache-control: no-cache, no-store, must-revalidate, private
17 Content-Encoding: gzip
18
19 {
20         "aps": [
21                 {
22                         "ap_deployment_mode": "IAP",
23                         "ap_group": null,
24                         "cluster_id": "",
25                         "controller_name": "",
26                         "group_name": "dw_AP-Group",
27                         "ip_address": "192.168.0.14",
28                         "labels": [],
29                         "last_modified": 1627371453,
30                         "name": "ctrl-dw01",
31                         "public_ip_address": "95.91.239.249",
```

```
32                                  "status": "Up",
33                          },
34 ...
```

Der Status Code (Zeile 12): Die erste Zeile der HTTP Antwort kann dem folgenden Code entnommen werden.

```
12 HTTP/1.1 200 OK
```

Laut dem HTTP Protokoll sollte die erste Zeile einer HTTP Antwort aus der Protokollversion, einem dreistelligen Status Code und einer kurzen textuellen Beschreibung des Status Codes bestehen. In der vorliegenden Antwort wurde der Status Code 200 zurückgegeben. Dieser besagt, dass die angefragten Informationen erfolgreich abgerufen werden konnten. Wie in den folgenden Textpassagen erkenntlich wird, konnten die Informationen tatsächlich erfolgreich zurückgegeben werden. Der Statuscode ist somit an dieser Stelle angebracht.

Content Type und Encoding (Zeile 13, 14 und 27):

In der Zeile 13, 14 und 16 der Antwort werden genauere Informationen über die Formatierung des Inhalts der Antwort übergeben (Abbildung ..).

```
13 Content-Type: application/json
14 Transfer-Encoding: chunked
15 Content-Encoding: gzip
```

Nach dem HTTP Protokoll gehören die Header Content-Type (Zeile 13) und Content-Encoding zu den Entitäts Kopfdaten. Diese Kopfdaten sollen zusätzliche Metainformationen über den Inhalt der Antwort geben. In Fieldings Vokabular beschrieben, bezeichnen sie den Typ der Repräsentation einer Ressource. Der Content-Type Header spezifiziert den genauen Typ des Inhalts. Hier wird ein JavaScript Object Notation (JSON) Dokument erwartet. Weitere Typen wurden in weiteren RFCs definiert. Der Content-Encoding Header spezifiziert eine zusätzliche Encodierung des Inhalts. In diesem Beispiel wird das JSON Dokument zusätzlich mit der GZIP Komprimierung encodiert. Mit dem Transfer-Encoding Header wird festgelegt, ob Veränderungen an dem Inhalt der Nachricht vorgenommen wurden, um sie zu transportieren. So kann, wie in diesem Beispiel, der Inhalt aufgeteilt werden, um ihn schneller und sicherer zu transportieren.

Date und Cache Control (Zeile 18 und 21)

In den Zeilen 18 und 21 werden der Date und der cache-control Header gesetzt (im folgenden Code zu sehen).

```
13 Date: Sat, 31 Jul 2021 11:36:35 GMT
14 cache-control: no-cache, no-store, must-revalidate, private
```

Bis auf wenige Ausnahmefälle (beschrieben in Kapitel 14.18 des RFC 2616) muss jede Antwort eines Servers den genauen Absendezeitpunkt beinhalten. Die Zeitangabe muss dabei nach RFC 1123 formatiert sein. Sie kann später von Server und Client zu Entscheidungen bezüglich des Cachings der Ressource genutzt werden.

Der Header cache-control beschreibt, ob und wie eine Antwort zwischengespeichert werden soll. Der Header kann sowohl vom Client, als auch vom Server gesetzt werden. Bei einem Konflikt soll die restriktivste Caching Methode verwendet werden. In der vorliegenden Antwort des Servers werden die cache-control directives no-cache, no-store, must-revalidate und private gesetzt. Diese besagen, dass die Ergebnisse der Abfrage nicht wiederverwendet (no-cache) dun nicht zwischengespeichert (no-store) werden dürfen. Sie müssen stetig neu validiert werden (must-revalidate) und dürfen nicht in einem geteilten Cache abgelegt werden (private). Die Informationen sind hochsensibel, also scheint eine solche Caching Richtlinie sinnvoll. Durch die zuvor genannten Cache Header wird genau angegeben, ob die Ressource cachebar ist oder nicht. Somit wird die dritte Randbedingung Fieldings erfüllt. Auch Richardson empfiehlt klar, dass ein Server angeben sollte, dass eine Ressource nicht cachebar ist.

Der Body der Antwort (Ab Zeile 32)

Wie zuvor beschrieben, sieht der Status Code 200 einer Antwort vor, dass die Antwort einen Inhalt enthält. Wie zuvor erläutert, sollte es nach dem HATEOS Prinzip möglich sein, allein mittels Hyperlinks durch eine API zu navigieren. In dem vorliegenden Auszug des Nachrichteninhalts sind mehrere Bezüge zu anderen Entitäten klar erkennbar: So könnten ap_group (Zeile 36) und controller_name (Zeile 38) auf andere Ressourcen verweisen. Anstatt von Hyperlinks finden sich hier jedoch die Namen der jeweiligen Ressourcen wieder. Ohne vorherige Kenntnis der API Struktur ist es somit nicht möglich, mehr Informationen über z.B. die Gruppe des Access Points abzurufen.

Zusammengefasst erfüllt die Antwort des Monitoring Endpunktes mit korrekten HTTP Headern Level 0 des Gütemodells. Level 1 ist hier nur für die Abfrage relevant. Level 2 wird erfüllt dar ein korrekter Response Status Code zurückgegeben wird. Level 3 wird nicht erfüllt, da in der Antwort keine Hypermedia Referenzen verwendet werden.

3.2.3 Einordnung in das Richardson Reifegradmodell

Nachdem der Monitoring Endpunkt zuvor analysiert wurde, wird er nun in das Reifegradmodell von Richardson eingeordnet. Level 0 (HTTP) ist bestanden, da sowohl Abfrage, als auch Antwort nach HTTP/1.1 korrekte beschrieben sind. Level 1 (Ressourcen) wird durch die korrekte URL in der Abfrage erfüllt. Level 2 durch korrekte HTTP Verben und Status Codes. Level 3 wird nicht erfüllt, da das HATEOS Prinzip nicht beachtet wird. Aufgrund dieser Beobachtungen ist anzunehmen, dass eine Implementierung der Middleware eingeschränkt möglich ist (Level 0 bis 2). Es müssen jedoch Informationen zur Struktur der API in den Client eingebaut werden, dar

diese nicht durch Hypermedia Referenzen automatisiert abrufbar sind. Das Forschungsergebnis wird im folgenden Kapitel mittels Prototyping validiert.

3.3 Evaluierung des Analyseergebnisses durch Prototyping

Mittels einer argumentativ deduktiven Analyse konnte festgestellt werden, dass die Aruba Cloud Central API überwiegend den REST Best Practices entspricht. Dieses Forschungsergebnis soll validiert werden. Best Practice Empfehlungen verfolgen i.d.R. den Zweck, die Geschwindigkeit und Qualität einer Implementierung zu erhöhen. Entspricht die Aruba Cloud Central API den Best Practices, ist folglich anzunehmen, dass eine Umsetzung der Middleware mit vergleichsweise geringen Aufwand vollzogen werden kann.

3.3.1 Genereller Programmablauf

Das Middleware Programm besteht im Wesentlichen aus HTTP Anfragen an die Aruba Cloud Central REST API. HTTP Anfragen sind in ihrer Natur asynchron. JavaScript ermöglicht es ebenfalls, in einer asynchronen Art zu programmieren. Demnach liegt es auch auf der Hand, das Inventarisierungsprogramm asynchron zu gestalten. In JavaScript stehen eine Reihe von Mechanismen zur asynchronen Programmierung zur Verfügung. Ein sehr verbreiteter Ansatz ist der der Promises. Dieser Ansatz wird auch in dem Skript verfolgt. Speziell werden in der Anwendung verschiedene HTTP Anfragen in einer sog. Promise Chain der Reihe nach abgearbeitet . Auch die verwendete Bibliothek Axios unterstützt das Arbeiten mit einer Promis Chain.

Zunächst wird der Authentifizierungsendpunkt aufgerufen (Zeile 12 bis 20). Dieser Aufruf gibt ein Promise zurück. In einem nächsten Funktionsaufruf werden die zuvor abgerufenen Authentifizierungsinformationen abgespeichert (Zeile 21 bis 41). Die komplette Authentifizierungsprozedur wird hier nicht weiter erklärt, da sie unabhängig von dem analysierten Endpunkt geschieht. Mit den Authentifizierungsinformationen werden dann die Inventardaten der Access Points abgerufen (Zeile 42 bis 51, Kapitel). Zuletzt werden die gesammelten Daten ausgegeben (Zeile 52 bis 62, Kapitel). In Zeile 63 bis 65 wird über die Catch Methode ein Error Handler an alle Promises gehängt. Sollte einer der zuvor genannten Schritte mit einer Exception abgebrochen werden, wird die Promise Chain abgebrochen und der Fehler wird mit dem catch Block behandelt.

3.3.2 Umgebungsvariablen

Für die Authentifizierung mit der API sind Zugangsdaten erforderlich. Die Zugangsdaten werden dem Programm in sogenannten Umgebungsvariablen zur Verfügung gestellt. So ist garantiert, dass keine sensiblen Daten in dem Programmcode aufzufinden sind. Tabelle .. zeigt eine Erklärung der Umgebungsvariablen:

Variable	Erklärung
ARUBA_CENTRAL_API_ROOT_PATH	Der Basis Pfad der Aruba Central API
ARUBA_CENTRAL_CLIENT_ID	Die Client ID als Sicherheitsmerkmal
ARUBA_CENTRAL_CLIENT_SECRET	Das Client Secret als Sicherheitsmerkmal

Tab. 2: Umgebungsvariablen.

3.3.3 Abrufen der Inventardaten

Nach der erfolgreichen Authentifizierung mit der Aruba API können die Inventardaten abgerufen werden (Zeile 42 bis 51). Alle vom Kunden geforderten Daten lassen sich mittels eines API Endpunktes abrufen. Dieser Endpunkt wurde bereits in beschrieben.

```
13          return axios.get(process.env.ARUBA_CENTRAL_API_ROOT_PATH + 'monitor
14          {
15              headers: {
16                      Authorization: `Bearer ${tokenStore.readTokenObject(
17              }
18          })
```

Die Abfrage besteht aus dem Funktionsaufruf der axios.get Funktion. Die axios.get Funktion wird in der offiziellen Dokumentation von Axios erläutert. Mit ihr ist es möglich, eine GET Anfrage an eine bestimmte URL zu senden.

Als erster Parameter wird bei der axios.get Methode die URL angegeben (Zeile 45). Diese URL entspricht der URL des zuvor in Kapitel analysierten Monitoring Endpunktes. Neben der URL ist für eine erfolgreiche Abfrage lediglich der Access Token notwendig. Dem OAuth2 Standard entsprechend wird dieser Token in einem Authorization Header mit der Abfrage an den Server geschickt. Header können mit Axios mit der headers Konfigurationsoption gesetzt werden. In Zeile 47 bis 49 wird ein Authorization Header mit dem Access Token an die Abfrage angehängt. Mit dem Aufrufen der Funktion wird die HTTP Anfrage abgeschickt und ein Promis für die Erfüllung der Abfrage zurückgegeben. Im Promise wird danach mit der then Funktion eine Funktion hinterlegt, welche die abgerufenen AP Daten in der Konsole ausgibt.

3.3.4 Ausgeben der Inventardaten

Ist die Abfrage der Monitoring Informationen erfolgreich werden diese in der Programm Konsole ausgegeben. Da die vorliegende Version der Software lediglich ein Prototyp ist, genügt diese Ausgabe den Anforderungen (siehe Kapitel). In der fertiggestellten Middleware werden die Informationen an dieser Stelle in das Inventarisierungsprogramm des Kunden geladen.

In diesem Beispiel werden die Access Point Daten zunächst aus der Antwort des Servers extrahiert (Zeile 55). Dann wird mit einer Schleife über die Objekte der einzelnen Access Points iteriert (Zeile 58) und jeweils beispielhaft der Name des APs ausgegeben (Zeile 59). Sind die Daten

erfolgreich ausgegeben hat die Middleware an dieser Stelle alle Anforderungen erfüllt. Trat bei einem der zuvor genannten Schritte ein Fehler auf, tritt die in Kapitel erklärte Fehlerbehandlung in Kraft.

4 Fazit

In dieser Arbeit konnte erfolgreich ein Regelwerk zur Konzeption von REST APIs aufgestellt, angewendet und validiert werden.

Es wurden zunächst die theoretischen Grundlagen des REST Architekturstils untersucht (Kapitel 2). Dabei konnten durch Primärquellen von Fielding ein Grundwortschatz (Kapitel 2.2), die Grundziele (Kapitel 2.3.2), sowie sieben Randbedingungen der REST Architektur identifiziert werden (Kapitel 2.3.3). Weiter konnte mit Sekundärliteratur von Richardson, Pautasso, Palma, Masse und Tilkov ein Gütemodell als narratives Referenzmodell zur Beurteilung von REST APIs erstellt werden. Auf Basis des Richardson Gütemodells wurden vier Ebenen begründet (Kapitel 2.4). Jede REST API kann in eine der Ebenen eingeordnet werden. Zur Erfüllung einer höheren Ebene muss erst die jeweils darunter liegende erfüllt werden. Eine höhere Ebene bedeutet eine genauere Übereinstimmung mit den Prinzipien des REST Architekturstils. Auf der vierten Ebene entspricht eine API dem REST Architekturstil komplett.

Im Praxisteil (Kapitel 3) konnten die gewonnenen Erkenntnisse im Rahmen einer betrieblichen Problemstellung angewendet und validiert werden. Für einen Kunden der Business Unit Aruba Networks der Hewlett Packard Enterprise sollte ein Middleware Programm erstellt werden. Diese Middleware sollte sich mit der API der Aruba Access Points verbinden und Informationen von der API abrufen. Diese Informationen sollten einem Inventarisierungsprogramm zur Verfügung gestellt werden.

Die Umsetzung der Middleware wurde in drei Schritten vollzogen: Zunächst wurde die betriebliche Problemstellung erläutert (Kapitel 3.1) und eine Reihe von Anforderungen an die Middleware informell festgehalten (Kapitel 3.1.3). Als zweiter Schritt zur Umsetzung der Middleware wurde die Machbarkeit der Implementierung der Middleware untersucht. Es wurde mittels des im Theorieteil erstellten Gütemodells untersucht, ob die API von Aruba Central der REST Architektur folgt (Kapitel 3.2). Die API wurde in die dritte Ebene des Gütemodells eingeordnet. Für die vierte Ebene fehlten der API Hypermedia Referenzen. Somit sollte eine Umsetzung der Middleware grundsätzlich möglich sein. Jedoch konnte angenommen werden, dass die Struktur der API in der Middleware hinterlegt werden muss. Dies wäre nicht der Fall, wenn die API die vierte Ebene des Gütemodells erfüllt hätte. Diese Erkenntnis wurde in einem dritten Schritt validiert. Hierzu wurde ein Prototyp der Middleware erstellt. In Kapitel 3.3 wird die Implementierung des Prototyps schemenhaft beschrieben. Der Prototyp konnte mittels einer verbreiteten Bibliothek zum Versenden von HTTP-Abfragen mit einer vergleichsweise geringen Komplexität umgesetzt werden. Dies spricht für die Erfüllung der dritten Ebene. Jedoch musste die genaue Lokation der verwendeten API Ressource in die Middleware codiert werden. Hätte die API Ebene vier des Gütemodells erfüllt, wäre es möglich, die Lokation der Ressource mittels Hypermedia Referenzen zur Laufzeit zu ermitteln. Somit entspricht die Realität der Umsetzung der Middleware genauestens dem zuvor formulierten Modell. Dies beweist, dass das Gütemodell zur Beurteilung von APIs geeignet ist.

Zusammenfassend konnte in dieser Arbeit ein narratives Referenzmodell zur Klassifizierung von REST APIs deduktiv aus Primär- und Sekundärliteratur herausgearbeitet werden; dieses Modell konnte in einem betrieblichen Kontext angewendet werden; weiter wurde es konstruktivistisch mit einem Prototyp validiert; es konnte induktiv geschlussfolgert werden, dass die erfolgreiche Anwendung des Modells bei einer API auch auf eine weitere erfolgreiche Anwendung mit anderen APIs schließen lässt.

4.1 Kritische Evaluierung

Die gewonnenen Forschungsergebnisse werden nun kritisch hinterfragt. Dabei werden sowohl positive als auch negative Aspekte der Arbeit reflektiert dargestellt.

Zunächst spricht die erfolgreiche Anwendung im betrieblichen Kontext für diese Arbeit; das entstandene Programm wurde letztendlich vom Kunden leicht angepasst und in die Inventarisierungssoftware integriert. So kann der Kunde mit dem Arbeitsergebnis automatisiert die Aruba Access Points überwachen. Dieser Prozess würde bei einer manuellen Umsetzung deutlich mehr Arbeitszeit kosten. So konnte mit der Umsetzung und Integration des vorliegenden JavaScript-Programmes ein ökonomischer Nutzen erzielt werden. Nach der erfolgreichen Integration wurde vom Kunde exzellentes Feedback geäußert.

Gleichzeitig müssen auch negative Punkte in Betracht gezogen werden: Die Arbeit kann lediglich als eine Heuristik angesehen werden. Zwar ist die Quellenlage in vielerlei Hinsicht eindeutig, jedoch gibt es zum Erstellungszeitpunkt der Arbeit keinen allgemeinen Standard zu dem Aufbau von REST APIs. Weiter wurde das hier vorgestellte Referenzmodell lediglich mit einem Endpunkt einer API validiert. Zwar ist anzunehmen, dass andere Endpunkte der gleichen Firma auf ähnliche Weise strukturiert sind, jedoch konnte dies nicht abschließend validiert werden.

Zusammenfassend sprechen gerade Aspekte wie der ökonomische Nutzen des Kunden für den Erfolg der Arbeit. Es konnten alle Forschungsziele erfüllt werden und der Kunde empfand das Arbeitsergebnis als sehr zufriedenstellend.

4.2 Ausblick

Gerade die negativen Punkte der kritischen Evaluierung legen noch Potenzial zur Verbesserung der Arbeit offen:

Zunächst sollte die das hier erstellte Referenzmodell auf weitere Endpunkte von verschiedenen Programmierschnittstellen von verschiedenen Herstellern angewendet werden. Zusätzlich könnte es mit weiteren Prototypen weiter validiert werden.

Das Skript könnte generalisiert werden und als Beispiel in die öffentliche Dokumentation der Aruba Programmierschnittstelle aufgenommen werden. Das würde es anderen Kunden von Aruba ermöglichen, die gleiche Problemstellung zu bewältigen.

Anhang

Anhangverzeichnis

Anhang 1: So funktioniert's

Um den der Zitierrichtlinien nachzukommen, wird das Paket `tocloft` verwendet. Jeder Anhang wird mit dem (neu definierten) Befehl **anhang**{Bezeichnung} begonnen, der insbesondere dafür sorgt, dass ein Eintrag im Anhangsverzeichnis erzeugt wird. Manchmal ist es wünschenswert, auch einen Anhang noch weiter zu unterteilen. Hierfür wurde der Befehl **anhangteil**{Bezeichnung} definiert.

In Anhang 1/1 finden Sie eine bekannte Abbildung und etwas Source Code in Anhang 1/2.

Anhang 1/1: Wieder mal eine Abbildung

Abb. 2: Mal wieder das DHBW-Logo.

Anhang 1/2: Etwas Source Code

```
13 public class HelloWorld {
14   public static void main(String[] args) {
15     if(args.length == 0) {
16       System.out.println("Hallo Sie!");
17     } else {
18       System.out.println("Hallo " + args[0] + "!");
19     }
20   }
21 }
```

Literaturverzeichnis

Berners-Lee, T. (1996): Univeral Resource Identifiers – Axioms of Web architecture. URL: https://www.w3.org/DesignIssues/Axioms (Abruf: 12. 11. 2021).

Berners-Lee, T./Cailliau, R./Luotonen, A./Nielsen, H. F./Secret, A. (1994): The World-Wide Web. en. In: *Commun. ACM* 37.8, S. 76–82. ISSN: 0001-0782, 1557-7317. DOI: 10.1145/179606.179671. URL: https://dl.acm.org/doi/10.1145/179606.179671 (Abruf: 25. 10. 2021).

Berners-Lee, T./Fielding, R. T./Masinter, L. (2005): Uniform Resource Identifier (URI): Generic Syntax. STD 66. Backup Publisher: RFC Editor ISSN: 2070-1721 Published: Internet Requests for Comments. RFC Editor. URL: http://www.rfc-editor.org/rfc/rfc3986.txt.

Bülthoff, F./Maleshkova, M. (2014): „RESTful or RESTless – Current State of Today's Top Web APIs". en. In: *The Semantic Web: ESWC 2014 Satellite Events*. Hrsg. von Valentina Presutti/Eva Blomqvist/Raphael Troncy/Harald Sack/Ioannis Papadakis/Anna Tordai. Bd. 8798. Series Title: Lecture Notes in Computer Science. Cham: Springer International Publishing, S. 64–74. ISBN: 978-3-319-11954-0 978-3-319-11955-7. DOI: 10.1007/978-3-319-11955-7_6. URL: http://link.springer.com/10.1007/978-3-319-11955-7_6 (Abruf: 31. 10. 2021).

Fielding, R. T. (2000): Architectural Styles and the Design of Network-based Software Architectures"; Doctoral dissertation. In: ISBN: 978-0-599-87118-2.

– **(2021)**: Roy T. Fielding. URL: https://roy.gbiv.com/ (Abruf: 11. 11. 2021).

Fielding, R. T./Gettys, J./Mogul, J. C./Nielsen, H. F./Masinter, L./Leach, P. J./Berners-Lee, T. (1999): Hypertext Transfer Protocol – HTTP/1.1. RFC 2616. Backup Publisher: RFC Editor ISSN: 2070-1721 Published: Internet Requests for Comments. RFC Editor. URL: http://www.rfc-editor.org/rfc/rfc2616.txt.

Fowler, M. (2010): Richardson Maturity Model: Steps toward the glory of REST. In: URL: http://martinfowler.com/articles/richardsonMaturityModel.html.

Haupt, F./Leymann, F./Pautasso, Cesare (2015): A Conversation Based Approach for Modeling REST APIs. In: *2015 12th Working IEEE/IFIP Conference on Software Architecture*. Montreal, QC, Canada: IEEE, S. 165–174. ISBN: 978-1-4799-1922-2. DOI: 10.1109/WICSA.2015.20. URL: http://ieeexplore.ieee.org/document/7158518/ (Abruf: 20. 10. 2021).

Kotstein, S./Bogner, J. (2021): „Which RESTful API Design Rules Are Important and How Do They Improve Software Quality? A Delphi Study with Industry Experts". en. In: *Service-Oriented Computing*. Hrsg. von Johanna Barzen. Bd. 1429. Series Title: Communications in Computer and Information Science. Cham: Springer International Publishing, S. 154–173. ISBN: 978-3-030-87567-1 978-3-030-87568-8. DOI: 10.1007/978-3-030-87568-8_10. URL: https://link.springer.com/10.1007/978-3-030-87568-8_10 (Abruf: 20. 10. 2021).

Levy, S. (2005): The Trend Spotter | WIRED. URL: https://www.wired.com/2005/10/oreilly/ (Abruf: 10. 11. 2021).

LP, H. P. E. D. (2021): Access Points | Aruba. URL: https://www.arubanetworks.com/de/products/drahtlos/access-points/ (Abruf: 10. 11. 2021).

Maleshkova, M./Pedrinaci, C./Domingue, J. (2010): Investigating Web APIs on the World Wide Web. In: *2010 Eighth IEEE European Conference on Web Services*. Ayia Napa, Cyprus: IEEE, S. 107–114. ISBN: 978-1-4244-9397-5. DOI: 10.1109/ECOWS.2010.9. URL: http://ieeexplore.ieee.org/document/5693251/ (Abruf: 20. 10. 2021).

Massé, M. (2021): Mark Masse. en. URL: https://www.oreilly.com/pub/au/4998 (Abruf: 11. 11. 2021).

Massé, M. H./Massé, M. (2012): REST API design rulebook: designing consistent RESTful Web Service Interfaces. eng. Beijing Köln: O'Reilly. ISBN: 978-1-4493-1050-9.

Megginson, D. (2007): REST: the quick pitch | Quoderat. URL: https://quoderat.megginson.com/2007/02/15/rest-the-quick-pitch/ (Abruf: 02. 11. 2021).

Palma, F. (2021): Francis Palma, PhD. URL: https://scholar.google.com/citations?user=hzmzRsEAAAAJ&hl=en (Abruf: 10. 11. 2021).

Palma, F./Dubois, J./Moha, N./Guéhéneuc, Y.-G. (2014): „Detection of REST Patterns and Antipatterns: A Heuristics-Based Approach". en. In: *Service-Oriented Computing*. Hrsg. von Xavier Franch/Aditya K. Ghose/Grace A. Lewis/Sami Bhiri. Bd. 8831. Series Title: Lecture Notes in Computer Science. Berlin, Heidelberg: Springer Berlin Heidelberg, S. 230–244. ISBN: 978-3-662-45390-2 978-3-662-45391-9. DOI: 10.1007/978-3-662-45391-9_16. URL: http://link.springer.com/10.1007/978-3-662-45391-9_16 (Abruf: 20. 10. 2021).

Palma, F./Gonzalez-Huerta, J./Founi, M./Moha, N./Tremblay, G./Guéhéneuc, Y.-G. (2017): Semantic Analysis of RESTful APIs for the Detection of Linguistic Patterns and Antipatterns. en. In: *Int. J. Coop. Info. Syst.* 26.02, S. 1742001. ISSN: 0218-8430, 1793-6365. DOI: 10.1142/S0218843017420011. URL: https://www.worldscientific.com/doi/abs/10.1142/S0218843017420011 (Abruf: 31. 10. 2021).

Pautasso, C. (2009): Some rest design patterns (and anti-patterns). In.

Pautasso, Cesare (2014): RESTful Web services: principles, patterns, emerging technologies. Techn. Ber.

– **(2021a)**: Cesare Pautasso. URL: https://scholar.google.com/citations?user=r7ISNNYAAAAJ&hl=en (Abruf: 11. 11. 2021).

– **(2021b)**: Prof. Dr. Cesare Pautasso. URL: http://www.pautasso.info/ (Abruf: 10. 11. 2021).

Pautasso, Cesare/Zimmermann, O./Leymann, F. (2008): Restful web services vs. "big" web services. In: *Proceeding of the 17th international conference on World Wide Web - WWW '08*. ACM Press. ISBN: 978-1-60558-085-2. DOI: 10.1145/1367497.1367606.

Renzel, D./Schlebusch, P./Klamma, R. (2012): „Today's Top "RESTful" Services and Why They Are Not RESTful". In: *Web Information Systems Engineering - WISE 2012*. Hrsg. von David Hutchison/Takeo Kanade/Josef Kittler/Jon M. Kleinberg/Friedemann Mattern/John C. Mitchell/Moni Naor/Oscar Nierstrasz/C. Pandu Rangan/Bernhard Steffen/Madhu Sudan/Demetri Terzopoulos/Doug Tygar/Moshe Y. Vardi/Gerhard Weikum/X. Sean Wang/Isabel Cruz/Alex Delis/Guangyan Huang. Bd. 7651. Series Title: Lecture Notes in Computer Science. Berlin, Heidelberg: Springer Berlin Heidelberg, S. 354–367. ISBN: 978-3-642-35062-7 978-3-642-35063-4. DOI: 10.1007/978-3-642-35063-4_26. URL: http://link.springer.com/10.1007/978-3-642-35063-4_26 (Abruf: 20. 10. 2021).

Richardson, L. (2008): Justice Will Take Us Millions Of Intricate Moves. Act Three: The Maturity Heuristic. In: URL: http://www.crummy.com/writing/speaking/2008-QCon/act3.html.

– **(2021a)**: beautifulsoup4: Screen-scraping library. URL: http://www.crummy.com/software/BeautifulSoup/bs4/ (Abruf: 10. 11. 2021).

– **(2021b)**: Code : Leonard Richardson. en. URL: https://code.launchpad.net/%7Eleonardr/+branches (Abruf: 10. 11. 2021).

– **(2021c)**: L. Richardson | Semantic Scholar. en. URL: https://www.semanticscholar.org/author/L.-Richardson/153443626 (Abruf: 10. 11. 2021).

– **(2021d)**: Leonard Richardson - Software Architect - The New York Public Library | LinkedIn. en. URL: https://www.linkedin.com/in/leonardr (Abruf: 10. 11. 2021).

Richardson, L./Amundsen, M. (2013): RESTful Web APIs. First edition. OCLC: ocn827841775. Beijing: O'Reilly. ISBN: 978-1-4493-5806-8.

Richardson, L./Ruby, S. (2007): RESTful web services. OCLC: ocm82671871. Farnham: O'Reilly. ISBN: 978-0-596-52926-0.

Rodriguez, A. (2008): RESTful Web services: The basics. URL: http://www.ibm.com/developerworks/webservices/library/ws-restful/.

Rodríguez, C./Baez, M./Daniel, F./Casati, F./Trabucco, J. C./Canali, L./Percannella, G. (2016): REST APIs: A large-scale analysis of compliance with principles and best practices. In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. Bd. 9671. ISSN: 16113349. Springer Verlag, S. 21–39. ISBN: 978-3-319-38790-1. DOI: 10.1007/978-3-319-38791-8_2. URL: https://link.springer.com/chapter/10.1007/978-3-319-38791-8_2.

Tilkov, S., Hrsg. **(2015)**: REST und HTTP: Entwicklung und Integration nach dem Architekturstil des Web. ger. 3., aktualisierte und erw. Aufl. Heidelberg: dpunkt-Verl. ISBN: 978-3-86490-120-1.

– **(2021)**: Stefan Tilkov – O'Reilly Author(s) – O'Reilly. URL: https://www.oreilly.com/people/stefan-tilkov/ (Abruf: 11. 11. 2021).

Erklärung

Ich versichere hiermit, dass ich meine Project Thesis mit dem Thema: *Mein Titel* selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe. Ich versichere zudem, dass die eingereichte elektronische Fassung mit der gedruckten Fassung übereinstimmt.

(Ort, Datum) (Unterschrift)