Michael Lorenzoni, BSc

Migration eines etablierten Campus-Management-Systems zu einer ressourcenorientierten Single-Page-Architektur

Masterarbeit

zur Erlangung des akademischen Grades

Master of Science

Masterstudium: Telematik

eingereicht an der

Technischen Universität Graz

Betreuer

Assoc.Prof. Dipl.-Ing. Dr.techn. Denis Helic

Institute of Interactive Systems and Data Science

Graz, April 2018

This document is set in Palatino, compiled with pdfLaTeX2e and Biber.

The LaTeX template from Karl Voit is based on KOMA script and can be found online: https://github.com/novoid/LaTeX-KOMA-template

Eidesstattliche Erklärung

Ich erkläre an Eides statt, dass ich die vorliegende Arbeit selbstständig verfasst, andere als die angegebenen Quellen/Hilfsmittel nicht benutzt, und die den benutzten Quellen wörtlich und inhaltlich entnommenen Stellen als solche kenntlich gemacht habe. Das in TUGRAZonline hochgeladene Textdokument ist mit der vorliegenden Masterarbeit identisch.

Datum	Unterschrift

Abstract

The migration of information systems to a new technology is a major technical and organizational challenge in software development. An approach to developing a new architectural style for the target system is illustrated by means of the migration of an established campus management system (CMS). On the basis of an abstract product vision, a flexible information architecture is developed. Subsequently, a resource-oriented single page architecture (ROSPA) based on 25 principles is derived. A special focus of this thesis is placed on the flexible integration of the current system based on Oracle PL/SQL into a new resource-oriented target architecture based on Java EE and Angular. During the iterative migration time of several years the entire functionality of the system must be continuously provided to the users, while at the same time guaranteeing a consistent user experience. In addition, the subject-based modularization of the system into segments, areas and applications as well as the technical modularization into layers are described in detail. The thesis closes with an introduction to a contextual role-based access control allowing to assign dynamic roles depending on the current user context.

Kurzfassung

Die Migration von Informationssystemen auf eine neue Technologie ist eine sehr große technische und organisatorische Herausforderung in der Softwareentwicklung. Anhand der Migration eines etablierten Campus-Management-System (CMS) wird eine mögliche Vorgehensweise illustriert, wie ein neuer Architekturstil für das Zielsystem entwickelt werden kann. Ausgehend von einer abstrakten Produktvision wurde eine flexible Informationsarchitektur entwickelt, um anschließend einen ressourcenorientierten Single-Page-Architekturstil (ROSPA) abzuleiten, der auf 25 Grundsätzen beruht. Ein spezieller Fokus dieser Arbeit ist die flexible Integration des auf Oracle PL/SQL basierenden Altsystems in eine neue ressourcenorientierte auf Java EE und Angular basierende Zielarchitektur, um während der iterativen und mehrjährigen Migrationszeit dem Benutzer stets die Gesamtfunktionalität des Systems mit einheitlicher User Experience anbieten zu können. Zusätzlich wird die fachliche Modularisierung des Systems in Segmente, Bereiche und Applikationen und die technische Modularisierung in Schichten näher beleuchtet. Den Abschluss bildet die Vorstellung einer kontextuellen rollenbasierten Zugriffskontrolle (CORBAC), die es mittels dynamischer Rollen erlaubt, diese in Abhängigkeit vom aktuellen Benutzerkontext zu vergeben.

Inhaltsverzeichnis

Abbildungsverzeichnis

Abkürzungsverzeichnis

ABAC Attribute-Based Access Control

Ajax Asynchronous JavaScript and XML

API Application Programming Interface

CDI Context and Dependency Injection

CMS Campus-Management-System

coData Collection Data Protocol

CORBAC kontextuelle rollenbasierte Zugriffskontrolle

CSS Cascading Style Sheets

CSV Comma-separated values

DOM Document-Object-Model

DRY Don't repeat yourself

DTO Data transfer object

ECB Entity Control Boundary

E-GovG E-Government-Gesetz

HAL Hypertext Application Language

HATEOAS Hypermedia as the Engine of Application State

HRSM Hochschulraum-Strukturmittel

HTML Hypertext Markup Language

HTTP Hypertext Transfer Protocol

HTTPS Hypertext Transfer Protocol Secure

IA Informationsarchitektur

JAAS Java Authentication and Authorization Service

Java EE Java Platform Enterprise Edition

JAXB Java Architecture for XML Binding

JAX-RS Java API for RESTful Web Services

JDBC Java Database Connectivity

JPA Java Persistence API

JSF JavaServer Faces

JSON JavaScript Object Notation

JSON-P Java API for JSON Processing

KISS Keep it short and simple

LeSS Large-Scale Scrum

PDF Portable Document Format

PL/SQL Procedural Language/Structured Query Language

RBAC Role-Based Access Control

REST Representational State Transfer

ROA Ressourcenorientierte Architektur

ROCA Ressourcenorientierte Client Architektur

ROSPA Ressourcenorientierter Single-Page-Architekturstil

SLC Student Lifecycle

SOAP Simple Object Access Protocol

SoD Separation of Duties

SPA Single-Page-Webanwendung

SQL Structured Query Language

SSL Secure Sockets Layer

TU Graz Technische Universität Graz

URL Uniform Resource Locator

WAI Web Accessibility Initiative

WWW World Wide Web

XML Extensible Markup Language

XSL Extensible Stylesheet Language

Gender Erklärung

Aus Gründen der besseren Lesbarkeit wird in dieser Masterarbeit die gewohnte männliche Sprachform bei personenbezogenen Substantiven und Pronomen verwendet. Dies impliziert jedoch keine Benachteiligung des weiblichen Geschlechts, sondern soll im Sinne der sprachlichen Vereinfachung als geschlechtsneutral zu verstehen sein.

Danksagung

Ein unendlicher Dank geht an meine beiden technischen Mitstreitert Joachim Lippnegg und Lucas Reeh. Außerdem möchte ich mich ganz herzlich beim Leiter des Zentraler Informatikdienst der Technischen Universität Graz Herr Dipl. Ing. Isidor Kamrat bedanken, welcher dieses Projekt ins Leben gerufen hat und mir die ehrenvolle Aufgabe der technischen Leitung übergab.

Ein ganz besonderes Dankeschön geht an meine Frau Kerstin Lorenzoni, welche mich immer motiviert hat und viel Geduld aufgebracht hat.

1. Einleitung

Ein Campus-Management-System (CMS) ist ein integriertes Anwendungssystem im Bereich von Universitäten und Hochschulen. Dabei deckt es im engeren Sinne laut Alt und Auth (2010) eine operative Unterstützung aller relevanten Geschäftsprozesse, sowie Planungs- und Kontrollfunktionalitäten im Bereich Studium und Lehre ab. Ein wesentlicher Aspekt dabei ist die Auffassung des Studierenden als Kunden, der im Zuge seines Student Lifecycle (SLC) bestmöglich unterstützt werden soll. Die *Bologna-Reform*, eine dezentrale Organisationsstruktur und der immerwährende Wettbewerb um Studierende, stellen die Verwaltung einer Hochschule vor große Herausforderungen, die mittels eines integrierten CMS bewältigt werden können. (Lämmerhirt, Franssen und Becker, 2013) Um dies zu ermöglichen sollen alle Daten und Prozesse direkt, konsistent und in Echtzeit durch die prozessteilnehmenden Personen verfügbar sein. Für die im Internetzeitalter aufgewachsenen Studierenden aber auch Lehrenden ist die Verwaltung der Daten an der Quelle, im Sinne von *Self-Service*, und ein breites mobiles Applikationsangebot selbstverständlich.

Diese Arbeit handelt von der technischen Migration des in Österreich und Deutschland etablierten CMS CAMPUSonline auf einen neuen Technologie-Stack. Die webbasierte Softwarelösung CAMPUSonline wird von der Technischen Universität Graz (TU Graz) seit 2004 als CMS angeboten und bisher evolutionär für verschiedenste Bildungseinrichtungen weiter entwickelt. Aufgrund des über die Jahre hinweg veralteten Technologie-Stacks, traf man 2014 zusammen mit vielen österreichischen Bildungseinrichtungen die Entscheidung das System zu modernisieren. Der neue Technologie-Stack dient zur Abbildung einer modularen, zukunftssicheren ressourcenorientierten Architektur (ROA) und wird in weiterer Folge als CAMPUSonline EE bezeichnet. Zur klaren Unterscheidung wird die aktuelle Technologie in

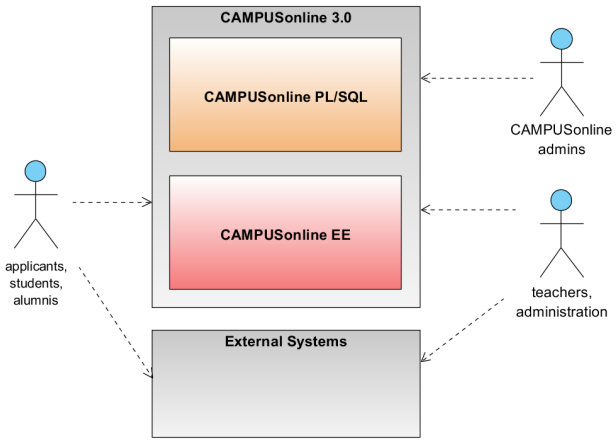

Abbildung 1.1.: CAMPUSonline soll vom bestehenden CAMPUSonline PL/SQL Technologie-Stack auf den neuen in dieser Arbeit vorgestellten CAMPUSonline EE Technologie-Stack migriert werden.

dieser Arbeit als CAMPUSonline PL/SQL bezeichnet. Es ist eine Gesamtmigration des bestehenden Technologie-Stacks bis zum Jahre 2025 geplant. Wie man der Abbildung 1.1 entnehmen kann, koexistieren bis zu diesem Datum beide Technologie-Stacks und bilden nach außen hin für die Benutzer ein einziges System, das weiterhin als CAMPUSonline bezeichnet wird. Eine besondere Herausforderung dieses Vorhabens ist die sehr lange Laufzeit und die enorme Funktionsvielfalt des bestehenden Systems, die während der mehrjährigen Migrationszeit erhalten bleiben muss. Dabei sollen unwirtschaftliche Faktoren, wie die doppelte Wartung und ein langwieriger organisatorischer Veränderungsprozess, minimiert werden, während gleichzeitig eine sukzessive Qualitätssteigerung des Gesamtsystems CAMPUSonline erreicht werden soll.

1.1. Motivation

Der Architekturstil Representational State Transfer (REST) definiert die fundamentalen Prinzipien des World Wide Web (WWW) und dient dazu ein lose gekoppeltes Netzwerk an verlinkten Ressourcen - eine sogenannte

ressourcenorientierte Architektur (ROA) - aufzubauen. REST ist für viele wichtige Qualitätsattribute des WWW verantwortlich, wie zum Beispiel die langfristige Beständigkeit und die uneingeschränkte Skalierbarkeit. (X. Xu u. a., 2008) Die Verbreitung von REST hat in den letzten Jahren aufgrund seiner Leichtgewichtigkeit und Flexibilität stark zugenommen und konkurrierende Ansätze wie Simple Object Access Protocol (SOAP) im Bereich Web Application Programming Interfaces (APIs) zunehmend verdrängt. Einer aktuellen Studie zufolge (Geene, R. Garrett und Lane, 2017) ist der Verbreitungsgrad von REST basierenden APIs auf 83% gegenüber SOAP basierenden Web APIs (15%) angestiegen.

Die Kernidee hinter dem neuen Technologie-Stack von CAMPUSonline EE ist die Umstellung des Systems auf eine große und homogen aufbereitete ROA, die sowohl interne als auch externe Clients in gleicher Weise bedienen kann. Die Motivationsfaktoren zum Aufbau einer ROA konformen RESTful-Servicelandschaft für ein CMS sind vielseitig. Es sollen Endgeräte jeglicher Art, zur Abwicklung aller Aktivitäten von Studierenden und Lehrenden im SLC, durch den Aufbau präsentationsneutraler Schnittstellen unterstützt werden. Dabei soll die Gesamtfunktionalität des Systems über diese Schnittstellen zugänglich gemacht werden. Gerade die fehlende Unterstützung von mobilen Endgeräten ist ein großer Kritikpunkt an CAMPUSonline, der damit sehr einfach lösbar wäre. Die neue Benutzeroberfläche von CAMPUSonline EE wird als eine Single-Page-Webanwendung (SPA) realisiert und bietet dank dem *Responsive Design* einen benutzerfreundlichen Zugriff mit unterschiedlichen Endgeräten. Aufsetzend auf die RESTful-Servicelandschaft könnten Studierende sogar selbst eine Vielzahl an nützlichen Zusatzangeboten entwickeln. Die für ein externes System zur Verfügung gestellten Daten und möglichen Aktionen sollen für Benutzer dabei komplett transparent und selektiv freischaltbar im Sinne des Datenschutzes sein. Selektiv freischaltbar bedeutet, der Benutzer soll selbst bestimmen können, welche Daten von welchen externen Systemen eingesehen bzw. manipuliert werden dürfen. Die zunehmende Studierenden- und Lehrenden-Mobilität im Zeitgeist der Internationalisierung bedingt die Einrichtung interuniversitärer Studiengänge. Daraus resultiert ein großer Bedarf an kooperativen Studiengängen, welche die Lehre interuniversitär und kostengünstig abbilden können. Für eine Effizienzsteigerung im Verwaltungsprozess dieser Studiengänge ist eine enge Vernetzung der beteiligten Informationssysteme

unumgänglich. Es sollen systemübergreifende personalisierte Sichten für Studierende und Lehrende angeboten werden, mit dem Ziel die vernetzten Systeme als ein einziges System zur Abwicklung der Geschäftsprozesse zu präsentieren. Zusätzlich besteht ein immer größer werdender Bedarf an zertifizierten Schnittstellen, über die externe Module lose gekoppelt an CAMPUSonline angedockt werden können. Ein strategisches Ziel von CAMPUSonline ist die Abkehr von einem allumfassenden monolithischen System. Stattdessen soll ein auf den studentischen Lebenszyklus konzentriertes Kernsystem aufgebaut werden, welches über Schnittstellen modular erweiterbar ist.

In dieser Arbeit soll ein Architekturstil zum Aufbau einer entsprechenden RESTful-Servicelandschaft erarbeitet werden. Durch die parallele Umsetzung des hier vorgestellten Architekturstils wird der Praxisbezug sichergestellt. Da es oft schwierig ist getroffene technische und architektonische Entscheidungen zu argumentieren, kann der Leser diese Arbeit als einen in der Praxis funktionierenden Leitfaden nutzen, wenn er selbst vor ähnlichen Herausforderungen steht.

1.2. Ausgangssituation

Um eine erfolgreiche Migration eines bestehenden Systems durchzuführen, ist es wichtig dessen aktuellen Zustand in Form einer kurzen Umfeldanalyse zu erheben. Diese erleichtert projektfremden Personen eine erste Einschätzung zur Genese und zur Größenordnung des Gesamtsystems zu entwickeln.

CAMPUSonline wird seit 1997 an der TU Graz entwickelt und hat dabei einige Evolutionsstufen durchlaufen. Das heutige System stellt noch immer eine Weiterentwicklung des Systems von 1997 dar. Eine Ablöse oder Migration von Technologien fand in diesem Zeitraum nur in geringem Ausmaß statt. CAMPUSonline ist an zahlreichen Universitäten, Fachhochschulen und pädagogischen Hochschulen im Einsatz, die im Sinne der Zusammenarbeit als *Kooperationspartner* bezeichnet werden. Das System ist

über 20 Jahre gewachsen und dementsprechend komplex. Derzeit arbeiten rund 100 Personen in verschiedenen Positionen an der Weiterentwicklung und Wartung. Daraus ergibt sich eine jährliche Aufwandsinvestition von ca. 80 Personenjahren. Im Rahmen des Hochschulraum-Strukturmittel-Projektes Hochschulraum-Strukturmittel (HRSM) *Campusmanagement an Österreichischen Universitäten* wurde von 15 österreichischen Universitäten eine Rundumerneuerung des Gesamtsystems angestoßen. Bisher wurde das System in einer proprietären, von Oracle angebotenen Programmiersprache namens Procedural Language/Structured Query Language (PL/SQL), entwickelt. Dabei erfolgte die Entwicklung ausschließlich direkt in der Datenbank und führte zu einem sehr großen, in sich geschlossenen, monolithischen System. Im Abschnitt 3.2 wird die Architektur des Altsystems genauer beschrieben.

1.2.1. Kritikpunkte

Einige Kritikpunkte am bestehenden System werden von unterschiedlichen Vertretern der Kooperationspartner kontinuierlich genannt. Sie stellen die Basis zur Entwicklung einer Produktvision (Abschnitt 1.3) für die neue Zielarchitektur dar.

- Das System ist über viele Jahre gewachsen und stößt nun an Komplexitätsgrenzen, die sich sowohl bei der Implementierung als auch bei der Parametrisierung negativ bemerkbar machen. Die Auswirkung von Änderungen sind oft nicht oder nur mit sehr großem Aufwand abschätzbar.
- Die Benutzeroberfläche wirkt veraltet und gestaltet sich uneinheitlich in den verschiedenen Bereichen des Systems. Die grafische Ausgabe ist nicht für diverse neue Endgeräte optimiert, die im Alltag durch Studierende und Bedienstete benutzt werden.
- Durch die aktuell verwendete Programmiersprache PL/SQL besteht ein *Locked In Syndrom* gegenüber eines einzelnen Software-Anbieters. Ein zukunftssicheres System muss im Einklang mit technologischen Innovationen entwickelt werden können und darf nicht an die Entwicklungsgeschwindigkeit von Oracle Lösungen gekoppelt sein.

- Bestehende Schnittstellen unterliegen keinem einheitlichem Konzept, welches von Haus aus eine einfache Integration in eine bestehende Systemlandschaft gewährleistet bzw. Erweiterung- und Anknüpfungspunkte anbietet.

1.2.2. Verbreitung

CAMPUSonline wird aktuell an 37 Universitäten und Hochschulen betrieben und befindet sich bei einer weiteren gerade in der Einführungsphase. Insgesamt werden dabei über 400.000 Studierende und 124.000 Mitarbeiter in ihrer täglichen Arbeit unterstützt. Das System wurde bereits 1998 an der TU Graz in Betrieb genommen und seit 2004 auch an anderen österreichischen Universitäten eingesetzt. Im Jahre 2008 hat sich die Technische Universität München für CAMPUSonline entschieden. Diesem Beispiel folgten seit 2011 viele andere deutsche Universitäten, wie die Universität zu Köln, die Universität Stuttgart und aktuell auch die RWTH Aachen.

1.2.3. Technische Fakten

In Zahlen, die durch eine Quellcodeanalyse erhoben wurden, stellt sich CAMPUSonline technisch wie folgt dar:

- Rund 4.500.000 Zeilen Quellcode
- Rund 3.600 Masken
- Rund 880 Listen
- Rund 1.250 Parameter
- Rund 60.000 Texte
- Rund 46.000 Datenabfragen
- Rund 3.600 Tabellen
- Rund 410.000.000 Datenbankfelder
- Rund 3.200 Datenbank-Views

1.3. Produktvision

CAMPUSonline EE ist eines der größten und herausforderndsten Vorhaben in der Geschichte von CAMPUSonline. Aus diesem Grund ist eine kommunikative und vor allem agile Vorgehensweise für den Projekterfolg unumgänglich. Laut Fowler und Highsmith (2001) ist der erste Schritt in einem agilen Projekt die Definition einer Produktvision in Form einer prägnanten Formulierung der Ziele. Sie geben die Leitplanken des Vorhabens vor. In weiterer Folge werden die einzelnen Punkte der Produktvision in dieser Arbeit genutzt, um den abgeleiteten Architekturstil zu begründen und zu argumentieren.

1. **Zukunftssichere Architektur ($Z_{ARCHITEKTUR}$)**
   CAMPUSonline EE ist eine modulare Internetanwendung und setzt konsequent auf die Grundprinzipien und den Architekturstil des World Wide Webs. Die Architektur ist extern leicht erweiterbar, homogen und vor allem zukunftssicher (Perspektive: mindestens 15 Jahre) gestaltet. Das CAMPUSonline-EE-Framework folgt konsequent den DRY- und KISS-Prinzipien und setzt möglichst auf etablierte Standards und Open-Source-Lösungen.
2. **Ausgereifte Schnittstellentechnologie ($Z_{INTERFACE}$)**
   Eine ausgereifte Schnittstellentechnologie gewährleistet die einfache und flexible Integration von CAMPUSonline EE in die bestehenden Systemlandschaften heutiger Universitäten und Hochschulen. Nicht durch CAMPUSonline abgedeckte Geschäftsprozesse höherer Bildungseinrichtungen können durch die nahtlose Anbindung von ausgewählten Partnerprodukten abgebildet werden. Durch externe Entwicklungen kann der Funktionsumfang von CAMPUSonline EE nach Bedarf der Kooperationspartner in definierten Bereichen und mit definierten technischen Mitteln erweitert und angepasst werden, ohne das die Basisfunktionalität von CAMPUSonline beeinflusst wird.
3. **Anbindung CAMPUSonline PL/SQL ($Z_{INTEGRATION}$)**
   Die gewählte Technologie ermöglicht eine einfache Anbindung an bestehende Strukturen von CAMPUSonline PL/SQL. Dabei liegt der Fokus auf einer schrittweisen Migration nach CAMPUSonline EE:

- Wiederverwendung und Integrierbarkeit bestehender Applikationen
- Wiederverwendung von zentraler Geschäftslogik aus CAMPUSonline PL/SQL
- Gleiche Datenbasis in CAMPUSonline PL/SQL und CAMPUSonline EE

4. **Ausgezeichnete Usability ($Z_{\text{USABILITY}}$)**
   Die mehrsprachige Benutzeroberfläche gestaltet sich für alle Benutzergruppen konsistent, einfach, barrierefrei und intuitiv – auch auf unterschiedlichen Endgeräten. Sie kann einfach sowohl an die Bedürfnisse und an das Corporate Design der Universitäten/Hochschulen als auch an persönliche Bedürfnisse der Benutzer angepasst werden.
5. **Hohe Qualität ($Z_{\text{QUALITÄT}}$)**
   Der Einsatz moderner Werkzeuge und einer hochautomatisierten, ökonomischen Entwicklungsstraße, steigern die Qualität des entwickelten Systems signifikant. Ziel ist eine nach internationalen Standards betriebssichere, wartbare, gereifte und fehlerfreie Funktionalität auszuliefern.
6. **Informationssicherheit ($Z_{\text{SICHERHEIT}}$)**
   Die Korrektheit und Sicherheit aller in CAMPUSonline verwalteten Daten, muss stets in allen Aspekten der Informationssicherheit (Integrität, Vertraulichkeit, Verfügbarkeit) gewährleistet sein.

1.4. Ziele und Grenzen

Viele wissenschaftliche Arbeiten stellen jeweils Teilaspekte zur Umsetzung eines sehr großen Informationsmanagementsystems zur Verfügung. Selten hingegen werden die einzelnen Architekturentscheidungen, die zur Umsetzung eines so großen Systems notwendig sind, einander gegenübergestellt und deren Auswirkung untereinander analysiert. Auch die speziellen Herausforderungen und Lösungen für eine mehrjährige, evolutionäre Migration eines bestehenden Systems auf einen komplett neuen Technologie-Stack, werden kaum in wissenschaftlichen Arbeiten in einem größeren Zusammenhang dargestellt. Ziel dieser Arbeit ist es den Lesern eine Vorgangsweise zu liefern, wie ein solches Vorhaben präsentiert, argumentiert und

umgesetzt werden kann. Aufgrund des beschränkten Umfangs einer Masterarbeit können leider viele Themen nicht detailliert beschrieben und nur im größeren Kontext dargestellt werden. Auch die wirtschaftliche Komponente und die Organisationsentwicklung hin zu einer nach Large-Scale Scrum (LeSS) funktionierenden Produktionswerkstatt, werden im Rahmen dieser Arbeit nicht dargestellt.

Konkret wird in dieser Arbeit ein Architekturstil zur Umsetzung eines CMS entwickelt. Die Grundsätze des Architekturstils bauen dabei auf bestehenden Architekturstile und auf eine zuvor beschriebene gewünschte Informationsarchitektur (IA) auf. Dabei werden RESTful-Prinzipien im Kontext moderner Softwarearchitektur-Paradigmen und deren nahtlose Integration in eine SPA beleuchtet. Der beschriebene Architekturstil wurde in Zusammenarbeit mit CAMPUSonline Mitarbeitern und vielen Stakeholdern der österreichischen und deutschen Universitätslandschaft erarbeitet, in die Praxis umgesetzt und bereits an einigen Bildungseinrichtungen erfolgreich ausgeliefert. Ein wesentlicher Neuwert des Architekturstils ist die Erweiterung einer klassischen ROA um den Aspekt des Kontextes in dem sich der Benutzer selbstbestimmt bewegen kann. Um den Anforderungen an die Zukunftssicherheit in CAMPUSonline gerecht zu werden, wurde ein neuer Media-Type namens coData (3.6.4) aufbauend auf bewährten Mustern im Zuge des Architekturstils erarbeitet. Einer der wichtigsten Erfolgsfaktoren eines CMS ist dessen Berechtigungssystem. Gerade in Zeiten in denen Datenschutz aufgrund zahlreicher Bedrohungen eine große Rolle spielt und in Anbetracht der EU Datenschutzgrundverordnung einen neuen Stellenwert erhält, ist ein besonders sicheres, dennoch flexibles und transparent konfigurierbares Berechtigungssystem unverzichtbar. Den Abschluss bildet eine Analyse und Erweiterung klassischer Zugriffskontrollmechanismen, um den Aspekt des Kontextes zur Ermittlung der Rollen und Rechte eines Benutzers.

1.5. Gliederung

In Kapitel 2 wird die neue Informationsarchitektur und das Navigationskonzept anhand von Benutzeranforderungen abgeleitet und mittels Studien

untermauert.

In Kapitel 3 wird der grundlegende Architekturstil des geplanten CMS beschrieben. Dabei wird besonders auf die Herausforderung der Integration und der Migration eines bestehenden Altsystems eingegangen. Die einzelnen architektonischen Elemente, begründet durch die in der Produktvision beschriebenen Zielvorgaben, werden anhand von wissenschaftlichen Arbeiten abgeleitet, vorgestellt und deren Auswirkung untereinander analysiert.

Die Architektur des neuen Systems wird mithilfe der Bausteinsicht in Kapitel 4 beschrieben. Dabei wird eine fachliche und auch technische Modularisierung des Systems vorgenommen.

Kapitel 5 erweitert Role-Based Access Control (RBAC) um das Konzept der kontextabhängigen Rollen, Privilegien und Objektrechte.

Das Kapitel 6 gibt einen Ausblick auf zukünftige Entwicklungen und Herausforderungen im Bereich von CMS, welche mit der vorgestellten Architektur strukturiert und effizient umgesetzt werden können.

2. Flexible Informationsarchitektur

Der wichtigste Aspekt eines CMS ist die benutzerfreundliche (Nielsen, 1999a, Norman und Draper, 1986) Bereitstellung von Informationen, die konkrete Bildungseinrichtung betreffend. Zusätzlich stellt ein CMS die Kommunikationsgrundlage dar, um alle Geschäftsprozesse einer Hochschule abzubilden. Die Aufgabe einer moderner Informationsarchitektur ist diese Konversation optimal zu gestalten und zu erleichtern. „Eine effektive IA war und ist also eine Struktur, die sich dem Nutzer nicht in den Weg stellt, sondern hilft innerhalb eines (meist digitalen) Informationssystems ein gewünschtes Ziel zu erreichen." (Wolf, 2010) Ausgehend von der Produktvision ZUSABILITY müssen eine Vielzahl unterschiedlicher Benutzergruppen bedient werden, die verschiedenste Bedürfnisse haben und unterschiedliche Zugänge zur Information benötigen (J. J. Garrett, 2010). Personen, die an einer Bildungseinrichtung studieren, durchlaufen im Rahmen des SLC (Abbildung 2.1) folgende Benutzergruppen. Interessenten wollen sich über das Bildungsangebot der Hochschule informieren. Aus Interessenten werden Bewerber sobald sie sich für eine Studienrichtung bewerben. Verläuft die Bewerbung positiv werden aus Bewerbern Studierende, die wiederum nach einem erfolgreich abgeschlossenen Studium zu Alumni werden. Auf der anderen Seite des SLC stehen die Mitarbeiter, welche unterschiedlichste Funktionen in einer Bildungseinrichtung einnehmen können. Dazu zählen Lehrende, Forschende, Fachabteilungspersonal und sonstige Mitarbeitende. Aber auch externe Personen der Hochschule, wie beispielsweise Gastdozenten, sind Benutzergruppen eines CMS. Zusätzlich kann ein und dieselbe Person auch gleichzeitig mehreren Benutzergruppen angehören. Als Beispiel seien hier studentische Mitarbeitende genannt.

Um dem Ziel der Benutzerfreundlichkeit (Abschnitt 1.3) gerecht zu werden, muss die IA für einzelne Benutzergruppen entsprechend aufbereitet (J. J. Garrett, 2010) und an den speziellen Kontext der Organisation angepasst

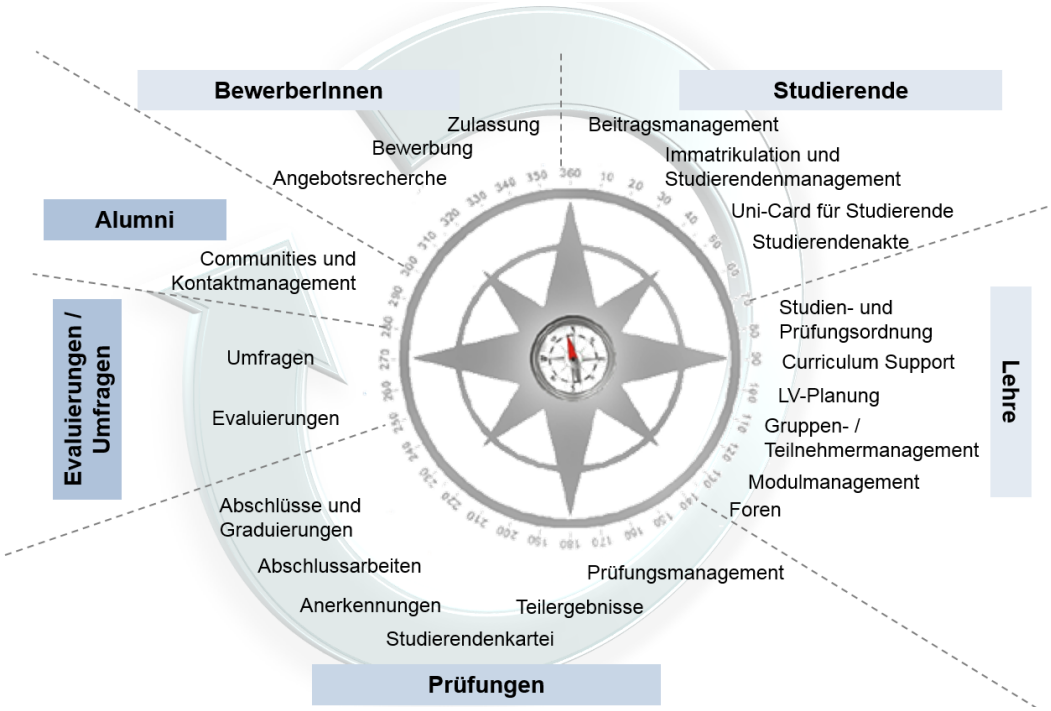

Abbildung 2.1.: Zur Abbildung des Student Lifecycle (SLC) werden in CAMPUSonline eine Vielzahl an Applikationen angeboten. Die Applikationen müssen sich zielgruppenorientiert an die Bedürfnisse der Benutzer anpassen.

werden (Rosenfeld und Morville, 2006, S. 26). Die IA definiert die Strukturierung und die Organisation des Informationssystems. Weitere zentrale Aspekte einer IA sind das Navigationskonzept, sowie die Wahl der entsprechenden Bezeichnungen der Inhalte und die Suchmöglichkeiten, die dem Benutzer angeboten werden. Die Definition einer vollständigen IA zu Beginn eines Projektes mit allen Navigationspfaden, allen Begriffen und Suchmöglichkeiten ist im Kontext eines großen Systems nur schwer zu bewerkstelligen. Im Sinne einer agilen Entwicklung ist dies auch wirtschaftlich nicht vertretbar. Aufgrund der geforderten Dynamik bei der Anpassung eines CMS an die Anforderungen unterschiedlicher Bildungseinrichtungen muss das System hoch konfigurierbar gestaltet sein. Die konkrete Ausprägung der IA wird also erst bei der Produkteinführung bzw. beim Ausrollen neuer Module definiert. Zusätzlich ist es aufgrund des wachsenden Systems und der schrittweisen Migration notwendig, dass die IA iterativ erweiterbar ist. Beim Design einer IA ist eine enge Zusammenarbeit mit den eigentlichen Benutzern des Systems notwendig, um schon frühzeitig auf deren Bedürfnisse eingehen zu können. Im konkreten Fall wurden unterschiedliche Praktiken aus dem Bereich Mensch-Computer-Interaktion angewandt, u.a. Prototyping, Benutzerbefragungen und Benutzertests. (Andrews, 2009, J. J. Garrett, 2010)

2.1. Strukturelemente der Navigation

In einem hochflexiblen System ist es wichtig, dass die Konfiguration einfach und für alle unterschiedlichen Stakeholder transparent durchgeführt werden können. Neben den eigentlichen Benutzern, sind auch die Bedürfnisse der Administratoren, der Produkteinführung und vor allem auch der Entwickler des Systems im Sinne von Erweiterbarkeit, Testbarkeit und Dokumentation zu berücksichtigen. Aus diesem Grund ist es notwendig exakt definierte Strukturelemente für das System zu finden, die über alle Stakeholder hinweg bekannt sind und einheitlich in der Kommunikation verwendet werden können. In diesem Abschnitt werden nun die einzelnen Elemente, die für die Navigation in CAMPUSonline EE verwendet werden vorgestellt und definiert.

2.1.1. Einzelfenster ($IA_{FENSTER}$)

Die Benutzer interagieren standardmäßig mit CAMPUSonline über genau EIN Fenster. Das bedeutet in CAMPUSonline EE wird eine Einzelfenster-Navigation angestrebt. Prinzipiell erlaubt es ein Browser mehrere Fenster zu öffnen, was im bestehenden System auch intensiv genutzt wurde, um den aktuellen Seitenkontext nicht zu verlieren. Durch das Öffnen neuer Fenster konnten Benutzereingaben bzw. der Applikationszustand der aktuellen Seite erhalten bleiben. Usability-Experten empfehlen bei Webapplikationen im Regelfall keine neuen Fenster zu öffnen, sondern die Navigation im selben Fenster durchzuführen (V. Friedman, 2008). Nielsen, 1999b meinte sogar: „Opening new browser windows [...] is a user-hostile message implied in taking over the user's machine." Es finden sich zahlreiche Argumente gegen die Verwendung mehrerer Fenster. Zuallererst die intransparente Aktualisierung von in Relation stehenden Fenstern, die fehlende Zurück-Funktionalität und die Unübersichtlichkeit bei übereinander liegenden Fenstern. Oft wird der Browser auch im Vollbildmodus betrieben - gerade beim Einsatz auf mobilen Endgeräten - wodurch die Benutzer nicht einmal merken, dass mehrere Fenster geöffnet wurden. Benutzer haben durch die Browserfunktionalität ohnehin die Möglichkeit, gezielt bestimmte Seiten in neuen Fenstern oder Browsertabs zu öffnen. Dieser Ansatz ist in der ISO9241 (Norm, o.D., S. 11) unter der Überschrift „Steuerbarkeit" zu finden. Die Regel besagt im weitesten Sinne, dass es Benutzern möglich sein muss die aktuell verwendete Benutzeroberfläche kontrollieren zu können, und selbst zu entscheiden, womit sie interagieren. Ausnahmen bestätigen die Regel. Bei Links die Benutzer von ihrer eigentlichen Aufgabe wegführen wie Drucksorten oder andere externe Seiten kann es sinnvoll sein diese in einem eigenen Fenster oder Browsertab zu öffnen.

2.1.2. Seiten

Aus Benutzersicht besteht CAMPUSonline EE aus einer Vielzahl von miteinander verlinkten einzelnen Seiten, welche Benutzer ansteuern können, um ihre Arbeit zu erledigen. Da unterschiedliche Bildungseinrichtungen

unterschiedliche Seiten und auch Workflows verwenden wollen, setzt CAMPUSonline auf eine hochkonfigurierbare Lösung um die einzelnen Seiten optimiert für die jeweilige Bildungseinrichtung miteinander zu verbinden. Alle Seiten besitzen eine eindeutige Id und sind somit eindeutig referenzierbar.

Sichten

Der Begriff Sicht ist in CAMPUSonline klar zu differenzieren vom Begriff der Seite. In CAMPUSonline werden eindeutig identifizierbare Seiten verwaltet, die sich abhängig von Rechten unterschiedlich gestalten und verhalten. Unter Sicht ist das Gestalten und Verhalten einer Seite für die Zielgruppe zu verstehen. Sprich eine Sicht ist eine Seite in einem speziellen Kontext. Beispiel: Im angemeldeten Zustand präsentiert sich eine Seite anders als im unangemeldeten.

2.1.3. Zugänge

Eine Seite kann über einen Zugang aufgerufen werden. Zugänge entsprechen aus Sicht des Benutzers jeweils einem Einstiegspunkt, also der erste Schritt, zur Begleitung eines Anwendungsfalls, zur Erreichung eines vom Benutzer gesetzten Ziels. Ein Zugang ist ein konfigurierbarer Link und besitzt neben dem Uniform Resource Locator (URL) weitere definierte Metainformationen. Zugänge sind im System eindeutig erfasst und können zur Laufzeit durch Administratoren angepasst werden. In der Zugangsverwaltung (Abbildung 2.2) im System Management können im Rahmen von Programmmodulen neue Zugänge zu Seiten angelegt werden bzw. bestehende verwaltet oder gelöscht werden.

Die wichtigste Metainformation eines Zugangs ist die Zugangsbedingung. Die Zugangsbedingung entscheidet, ob ein Zugang von der aktuell angemeldeten Person im aktuellen Kontext aufgerufen werden kann. Ist die Zugangsbedingung nicht erfüllt, sollte den Benutzern auch kein Hyperlink oder nur ein deaktivierter Hyperlink präsentiert werden, um Links die zu einer Fehlerseite führen zu vermeiden. Das heißt die Zugangsbedingung

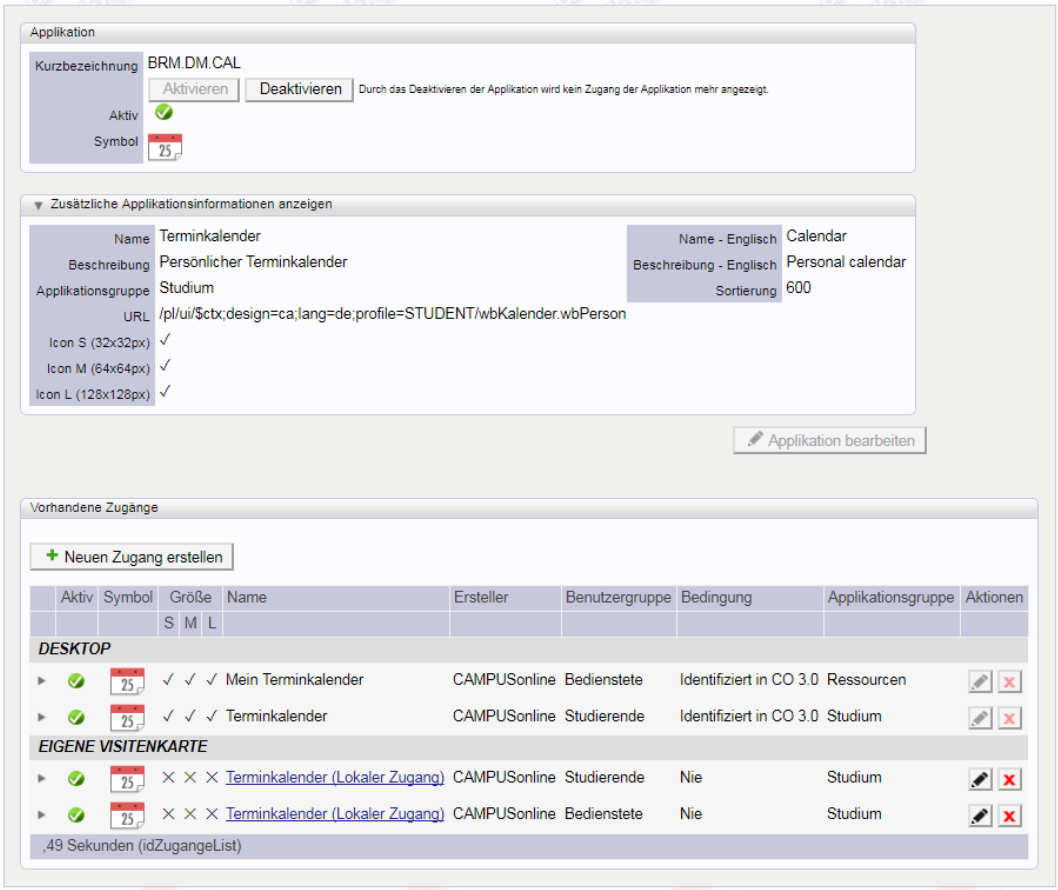

Abbildung 2.2.: Von einem Programmmodul können mehrere Zugänge abgeleitet werden. Die Attribute der Zugänge können von denen der Applikation übernommen und für die entsprechende Zielgruppe angepasst werden.

hängt von den Rollen der angemeldeten Person, dem Kontext und von den Prozessen der Bildungseinrichtung ab. Im Kapitel 5 wird beschrieben, wie die Auswertung von Zugangsbedingungen erfolgen kann.

2.1.4. Menüs

Klassischerweise wird die Navigation auf Webseiten über Menüs abgebildet. In CAMPUSonline werden Zugänge zu Menüs anhand von zusammengehörenden User Journeys gebündelt. „User Journeys sind alle Schritte, die ein Nutzer geht, um ein gewisses Ziel mit einem interaktiven System zu erreichen. Sie bestehen meist aus einer Anzahl von Website-Seiten und Entscheidungspunkten, die den Nutzer von einem Schritt zum nächsten bringen“ (Sthamer, 2016). Dem Gesetz der Gleichheit (Wertheimer, 1923, S. 309) entsprechend, können so Zugänge den Benutzern in einer homogenen Form präsentiert werden. Ziel ist es, sämtliche Funktionalitäten um eine Aufgabe zu erfüllen übersichtlich, konsistent und in der Sprache der aktuell angemeldeten Person anzubieten. Menüs begleiten die Anwendungsfälle der Benutzer auf intuitive Weise bis zur Zielerreichung. Funktionalitäten sollen dabei mit einer möglichst minimalen Anzahl an Klicks aus der Sicht der handelnden Person erreichbar sein. Die Menüeinträge werden dynamisch, je nach Rollen der angemeldeten Person und anhängig vom Kontext, ein bzw. ausgeblendet. Damit wird verhindert, dass für jeden Anwendungsfall ein eigenes Menü erstellt werden muss.

2.1.5. Applikationen

Aus der Sicht der Usability umfassen Applikationen mehrere, thematisch zusammenhängende und teilweise sich überschneidende Anwendungsfälle, rund um die Bearbeitung von im System verwalteten Objekten. Applikationen reduzieren sich auf einen technischen Begriff zur Ermöglichung von 1-m Applikationsmenüs. Die genaue Abgrenzung einer Applikation ist aus Benutzersicht weniger relevant, da der Einstiegspunkt bei Benutzern die Zugänge sind. Diese präsentieren sich nach mobilen Benutzeroberflächen Vorbildern als Links am Desktop in Form von *Apps*. Das bedeutet, eine

technisch abgebildete Applikation erscheint für den Benutzer als 1-n *Apps* am Desktop und können sich auf 1-m Applikationsmenüs erstrecken.

Somit entspricht die Strukturierung des Systems in Applikationen vielmehr der Sicht eines Entwicklers. Applikationen bieten Funktionalität an und bestehen aus Seiten, welche über Zugänge aufgerufen werden können. Im Abschnitt 4.4 wird der Begriff der Applikation in den Überbegriff des Programmmoduls eingebettet und genauer erläutert. Die Strukturierung in Applikationen ist eng verbunden mit der Strukturierung des Quellcodes und des darunterliegenden Datenmodells. Ziel ist es Applikationen für Entwickler übersichtlich, klein, wartbar und austauschbar zu halten. Applikationen sollen über schmale und definierte Schnittstellen mit anderen Applikationen kommunizieren. Im Gegensatz zur klassischen App-Entwicklung ist in einer dynamischen IA der Begriff der Applikation klar zu differenzieren. Eine klassische Anwendung entspricht aus Benutzersicht eher dem Begriff des Applikationsmenüs 2.3.4. Diese beiden Welten können sich aufgrund Conways Law (Conway, 1968) überschneiden, tun dies aber generell nicht. Besonders relevant wird der Unterschied, wenn eine Migration einer oder mehrere Applikationen durchgeführt wird. Dabei können neue Seiten und Seiten von bestehenden Applikationen zur Optimierung der Benutzerführung in einem Hauptmenü zusammengefasst werden. Sie erscheinen den Benutzern als eine *App*, obwohl sie im Hintergrund aus mehreren verschiedenen Applikationen bestehen.

2.2. Navigationsarten

In CAMPUSonline EE wird prinzipiell zwischen drei Arten der Navigation unterschieden.

- Menüübergreifende Navigation
- Menüinterne Navigation
- Seiteninterne Navigation

Die Durchmischung dieser drei grundlegend unterschiedlichen Navigationsarten und das Nichteinhalten einer strikten Trennung von Aktions-

mit Navigationselementen, sind oft geäußerter Kritikpunkt von Benutzern gegenüber dem CAMPUSonline PL/SQL System.

2.2.1. Menüübergreifende Navigation

Um Benutzer intuitiv durch die Prozesse zu führen, wird in CAMPUSonline immer genau ein dominantes Menü im aktuellen Kontext angezeigt. Menü übergreifende Navigation bedeutet, dass das aktuelle dominante Menü durch ein Neues abgelöst wird. Da das Wechseln des Menüs einen signifikanten Kontextwechsel darstellt, muss laut dem Prinzip der geringsten Überraschung (Geoffrey, 1987, S. 11) eine Menü übergreifende Navigation entsprechend gut ausgezeichnet bzw. vorhersagbar sein.

2.2.2. Menüinterne Navigation

Menü interne Navigation bedeutet, dass Benutzer von einer Seite zur nächsten Seite navigieren, dabei aber das aktuell dominante Menü bestehen bleibt. Benutzer bleiben im Kontext ihrer Aufgaben und wissen aufgrund des Seitenkopfes, wo sie sich aktuell gerade befinden.

2.2.3. Seiteninterne Navigation

Bei der Seiten internen Navigation bleiben Benutzer auf der aktuellen Seite. Das aktive Menü, der gewählte Menüeintrag und der Seitenkopf bleiben bestehen. Typische Seiten interne Navigation ist eine Sprungmarke zu einem bestimmten Eintrag in einer längeren Seite. Eine andere Seiten interne Navigation ist das Ausklappen oder auch die Anzeige einer Unterressource der Seite. Ein Beispiel hierfür wäre das Anzeigen von Detailinformationen zu einem spezifischen Eintrag in einer Übersichtsliste.

2.2.4. Navigation vs. Aktion ($IA_{AKTIONEN}$)

Aktionen sind zum Beispiel das „Hinzufügen“, „Löschen“ oder „Bearbeiten“ der auf einer Seite angezeigten Elemente. Aktionen die sich direkt auf Ressourcen beziehen, welche auf der aktuellen Seite angezeigt werden, sollen möglichst nahe der angezeigten Ressource präsentiert werden, um dem Gesetz der Nähe (Wertheimer, 1923, S. 310) zu entsprechen. Aktionen werden mittels Buttons dargestellt und dürfen zur klaren Differenzierung nicht in Navigationsmenüs vorkommen.

2.3. Menüarten

Zur Umsetzung der Navigation werden unterschiedliche Arten von Menüs verwendet. Jedes davon hat unterschiedliche Ansprüche bezüglich Konfigurierbarkeit und definierte Eigenschaften. Auf folgende Arten wird in diesem Abschnitt näher eingegangen:

- Statisches-Menü
- Desktop-Menü
- Applikationsmenü
- Detailmenü
- Breadcrumb-Menü

2.3.1. Entkopplung von Menüs und Seiten (IA_{MENU})

Ein ganz zentraler Aspekt der IA ist die Entkopplung der konkreten Seite vom tatsächlich angezeigten Menü. Eine konkrete Seite soll in unterschiedlichen Menüs integriert und wiederverwendet werden können. Nur dadurch können Menüs konfigurierbar gestaltet und an die Prozesse einer Bildungseinrichtung angepasst werden. In CAMPUSonline PL/SQL waren die Menüs fest mit der Seite verbunden und konnten von Administratoren nicht angepasst werden. Zusätzlich bestand der Wunsch, dass Menüs durch lokal bei den Kooperationspartnern entwickelten Seiten erweitert werden können. Bei klassischen Content-Management-Systemen ist die

flexible Zusammenstellung eines Menüs gelebte Praxis. Im Rahmen von Applikationsentwicklungen gestaltet sich diese Aufgabe allerdings wesentlich komplexer. Das Problem ist der über mehrere Seiten hinweg geteilte Applikationszustand und dass der Kontext der Seite Auswirkungen auf die Anzeige von Menüeinträgen hat. Im Abschnitt 2.4 wird beschrieben, wie diese Herausforderung aus der Sicht des Benutzers gelöst werden kann. Die technische Umsetzung des Kontextes wird unter Abschnitt 3.7.4 besprochen und das zugrundeliegende Berechtigungsmodell zur dynamischen Anzeige der Zugänge in Abhängigkeit eines Kontextes wird in Kapitel 5 dargelegt.

2.3.2. Statisches Menü

Das statische Menü bietet die Möglichkeit Navigationselemente jederzeit zugänglich zu machen. Es wird im Kopf- und Fußbereich jeder Seite angezeigt. Um die Benutzer nicht zu überfordern muss die Menge, der so zur Verfügung gestellten Navigationselemente, jedoch stark reduziert sein und sich auf die wichtigsten Funktionalitäten beschränken. Die wichtigsten statischen Navigationselemente sind in CAMPUSonline:

- die Sprachauswahl (wichtig für internationale Studierende)
- die An-/Abmeldung mit der Möglichkeit das Profil zu wechseln
- ein Start-Button in Form des Logos der Bildungseinrichtung
- die Möglichkeit die allgemeine Suche auszulösen
- das Impressum
- der Systemstatus

2.3.3. Desktop-Menü

Der Desktop bietet den Einstieg in diverse Anwendungsfälle (vgl. 2.1.3) und ist das zentrale Element für die Menü-übergreifende Navigation im System. Hier werden alle für die Benutzer wichtigen Zugänge als Icons mit einer textuellen Beschreibung zu weiterführenden Menüs dargestellt. Die Zugänge am Desktop sind typischerweise je Bildungseinrichtung sehr individuell für die unterschiedlichsten Benutzergruppen konfiguriert. Es

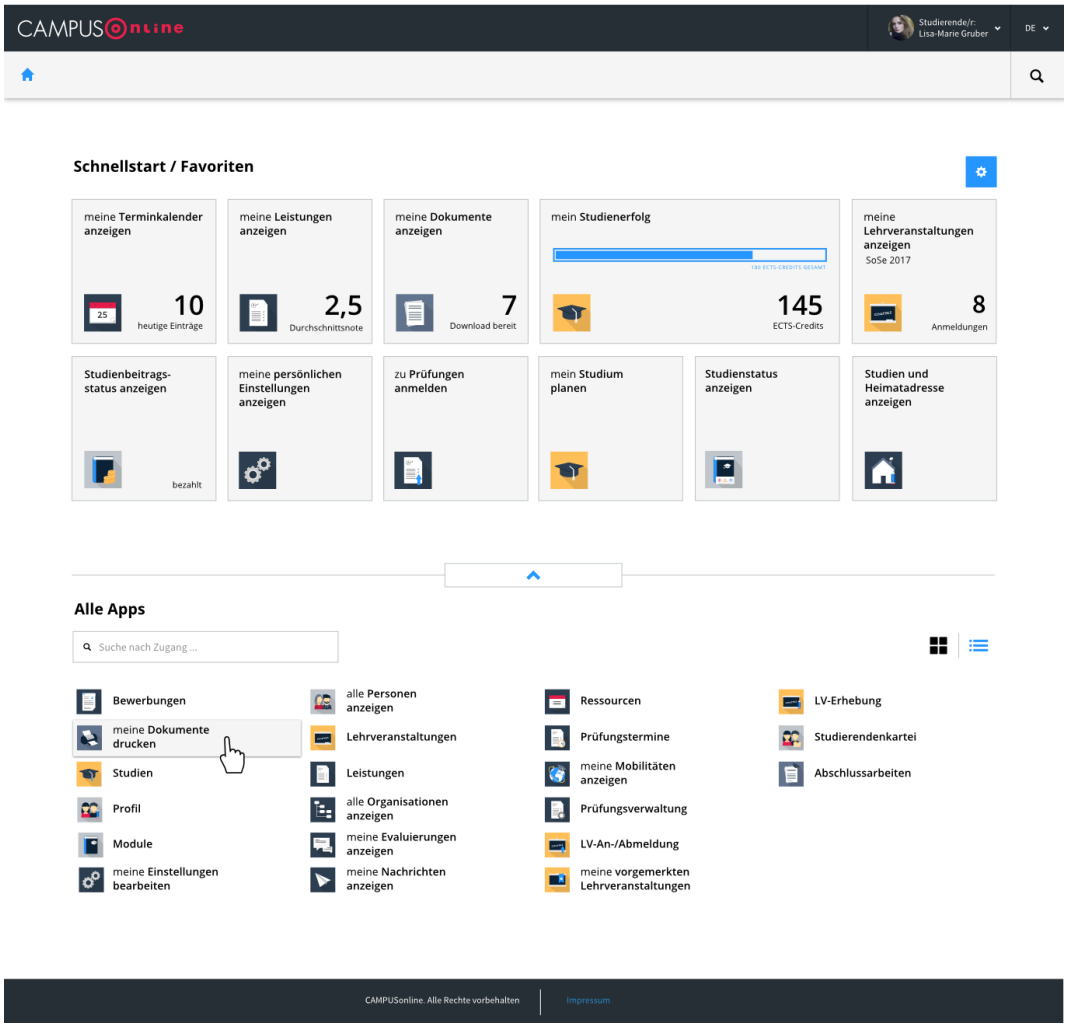

Abbildung 2.3.: Der Desktop bietet Zugänge zu den einzelnen Funktionalitäten von CAMPUSonline und stellt das klassische Beispiel für eine menüübergreifende Navigation dar.

werden auch nur jene Zugänge angezeigt zu denen die aktuell angemeldete Person tatsächlich auch berechtigt ist. Durch die Entkopplung von Seiten und Zugängen kann die Bezeichnung und das Icon des Zugangs entsprechend an die konkrete Zielgruppe angepasst werden. Zusätzlich können sich die Benutzer selbstständig zu jeder Seite im System einen Zugang im Favoriten-Bereich des Desktops anlegen. Dabei bleibt auch der aktuelle Grundzustand der Seite - wie ausgewählte Kontexte, Sortierungen, Filterungen usw. - erhalten. Im Unterschied zu klassischen Browser Lesezeichen bleibt bei persönlichen Zugängen die referentielle Verbindung zum ursprünglichen Zugang bestehen. Dadurch würde sich das Icon und der Text bei Änderung des zugrundeliegenden Zugangs mit ändern. Vor allem würde der persönliche Zugang nicht mehr angezeigt werden, wenn keine Berechtigung für die angemeldete Person mehr bestünde. Priorisiert durch die Analyse von Benutzerverhalten werden standardmäßig Favoriten bereits Zielgruppen orientiert mit ausgeliefert, um den unterschiedlichen Benutzern noch einen schnelleren Start für ihre Anwendungsfälle anbieten zu können.

Eine weitere Möglichkeit, die der Desktop anbietet, ist die Erstellung von Untermenüs, die zur Gruppierung von mehreren Menüeinträgen genutzt werden kann. Außerdem können Benutzer am Desktop auch Menüs von anderen Personen oder, allgemein gesprochen, Ressourcen im System anzeigen. Dazu zählen zum Beispiel die Visitenkarte eines Bediensteten oder die Informationsseite eines Institutes der Bildungseinrichtung. Natürlich stehen dabei nur jene Zugänge zur Verfügung zu denen die aktuelle angemeldete Person auch berechtigt ist. Zusammengefasst kann behauptet werden, dass das Desktop-Menü ein sehr individuelles auf die Bedürfnisse abgestimmtes Bild auf das System für jeden Benutzer anbietet, welches sowohl zentral als auch dezentral verwaltet werden kann.

2.3.4. Applikationsmenü

Ein Applikationsmenü wird klassischerweise über einen Zugang am Desktop aufgerufen. Es zeigt Zugänge zu Seiten an und bietet zusammengehörende Funktionalitäten zur Erfüllung von konkreten Aufgaben an. Aktionen, die auf die konkrete Seite wirken, dürfen nicht im Applikationsmenü angezeigt

werden. Das Applikationsmenü besteht aus maximal zwei Ebenen. Die Beschränkung auf zwei Ebenen fördert die Übersichtlichkeit und optimiert gleichzeitig die notwendigen Klicks um eine Seite aufzurufen. Die Anzeige des Applikationsmenüs passt sich an unterschiedliche Bildschirmgrößen an und verwendet je nach zur Verfügung stehenden Platz unterschiedliche Usability-Muster. Im Applikationsmenü können Zugänge direkt - auf oberster Ebene - angezeigt werden, die sich auf großen Bildschirmen dem Registerkarten Usability-Muster entsprechend verhalten. Zugänge können zu so genannten Zugangsgruppen zusammengefasst werden. In diesem Fall stellt sich der Menüpunkt auf großen Bildschirmen als aufklappbares Element dar. Steht nur wenig Bildschirmfläche zur Verfügung - wie etwa bei mobilen Browsern - so wird das gesamte Menü als eine aufklappbare gruppierte Liste von Links präsentiert.

Das Applikationsmenü dient überwiegend zur menüinternen Navigation. (Abbildung 2.5) Das bedeutet, dass bei der Aktivierung eines Zugangs das Applikationsmenü bestehen bleibt. Nur die Zugangsgruppe *Gehe zu* stellt eine Ausnahme im Applikationsmenü dar. Diese darf, aber dafür ausschließlich, Zugänge zur menüübergreifenden Navigation beinhalten. Da das Wechseln des Applikationsmenüs für die Benutzer einen großen Kontextwechsel in der Erledigung ihrer Aufgabe darstellt, wurde hier das bereits erwähnte Prinzip der geringsten Überraschung (Geoffrey, 1987, S. 11) angewendet.

Im System Management von CAMPUSonline können Zugänge flexibel zur Laufzeit angelegt und Menüs zugeordnet werden. Die Menüs sollen für eine Bildungseinrichtung sinnvolle Einheiten ergeben. Während das Desktop-Menü bei einem Neukunden typischerweise leer ausgeliefert wird, wird ein Satz von bereits vordefinierten Applikationsmenüs mitgeliefert, um Kooperationspartner bewährte Abläufe bei der Abbildung von Prozessen anbieten zu können. Bei Bedarf können Administratoren ein solches Menü allerdings ergänzen bzw. ein komplett neues Menü anlegen.

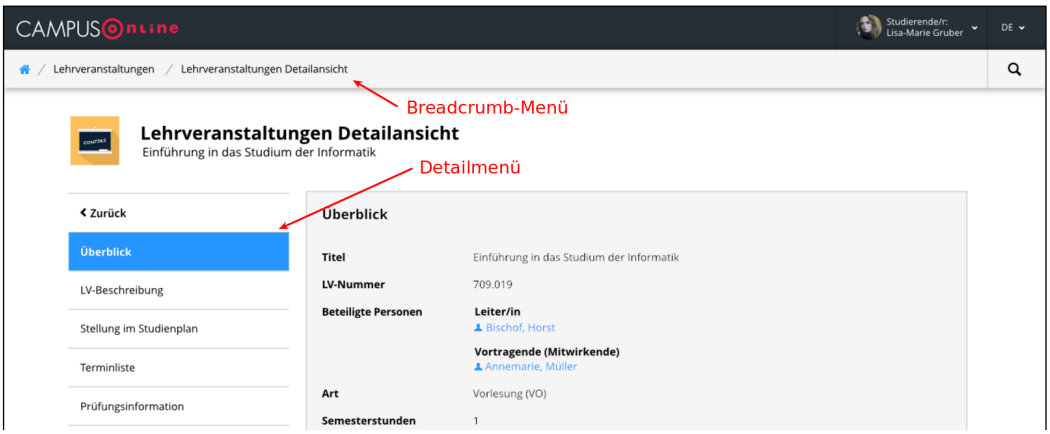

Abbildung 2.4.: Breadcrumbs zeigen, wo sich der Benutzer befindet und spiegeln Zugänge und die Seitenhierarchie wider

2.3.5. Detailmenü

Das Detailmenü gehört zu einer konkreten Seite und dient somit der Abbildung der seiteninternen Navigation. Dieses spezielle Menü wird für sehr lange Seiten verwendet (zum Beispiel Informationsseiten zu Lehrveranstaltungen), um den Benutzern Sprungmarken zu den einzelnen Sektionen der Seite anzubieten. Während alle anderen Menüarten konfiguriert werden können, ist das Detailmenü durch die Struktur der Seite vorgegeben.

2.3.6. Breadcrumb-Menü

Zur besseren Orientierung muss für den Benutzer zu jeder Zeit ersichtlich sein, wo er sich befindet. Breadcrumbs werden im Sinne von Location-Breadcrumbs (vgl. Instone, 2002) verwendetet, sind immer für eine URL eindeutig und stellen nicht den Klickpfad dar. (Abbildung 2.4) Sie informieren den Benutzer darüber, welcher Zugang am Desktop gewählt wurde und wie der aktuelle Seitentitel lautet. Darüber hinaus ist auch die Information interessant, zu welcher Zugangsgruppe die aktuelle Seite gehört. Aus diesem Grund wird diese Information im Breadcrumb-Menü stets

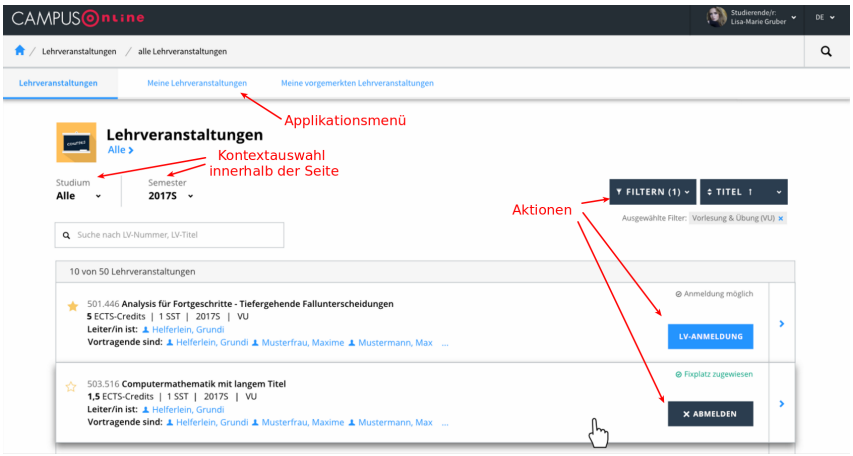

Abbildung 2.5.: Applikationsmenüs bestehen maximal aus zwei Ebenen und dienen zur menüinternen Navigation. Während Kontext, Auswahlmöglichkeiten und Aktionen direkt in der Seite angeboten werden, sind Applikationsmenüs von der Seite entkoppelt und stehen seitenübergreifend zur Verfügung.

angezeigt. Außer am Dekstop selbst, da man sich dort in der obersten Navigationsebene befindet. Die Breadcrumbs setzen sich aus diesem Grunde, je nachdem ob ein Zugang zu einer Zugangsgruppe gehört oder nicht, wie folgt zusammen:

- (Home) - (Menü-Namen) - (Zugangsgruppe-Namen) - (Seiten-Namen)
- (Home) - (Menü-Namen) - (Seiten-Namen)

Befindet man sich gerade in einem untergeordneten Detail der aktuellen Seite, so kann noch ein zusätzlicher Punkt zum Breadcrumb-Menü hinzugefügt werden.

2.4. Kontext ($IA_{KONTEXT}$)

In der Kommunikationswissenschaft versteht man unter Kontext, laut Bußmann und Lauffer (2002), alle Elemente einer Kommunikationssituation, die das Verständnis einer Äußerung bestimmen. Die Benutzer treten mit CAMPUSonline über eine Seite in den Dialog. Damit diese Seite für die

Benutzer sinnvoll angezeigt werden kann oder damit die Benutzer Daten an CAMPUSonline schicken können, sind Kontextinformationen notwendig.

Die Berechtigung eine Seite zu öffnen bzw. betrachten zu dürfen könnte zum Beispiel an eine Kontextinformation geknüpft sein. Es gibt Benutzer, die nur im Kontext des „Institute for Interactive Systems and Data Science “ eine Seite öffnen dürfen. Ein anderes Beispiel wäre, dass eine Professorin Müller eine Seite nur im Kontext des Studiengangs Telematik öffnen darf. Zusätzlich zu Berechtigungskontexten gibt es noch viele weitere Kontexte, wie der Zugang über den eine Seite geöffnet wurde, welcher sich auf das anzuzeigende Menü auswirkt. Ein anderes Beispiel wäre, dass man beim Zugang hinterlegen will, in welchem Design die Seite geöffnet wird. (CAMPUSonline PL/SQL Design oder das neue CAMPUSonline EE Design).

Ein Zugang öffnet eine Seite in einem speziellen Kontext. Sollte der Kontext beim Zugang nicht statisch definiert worden sein, muss der Benutzer die Möglichkeit bekommen, den Kontext für die Seite mittels eines Kontextauswahl-Dialoges (Context Chooser) zu wählen. Die wichtigste Randbedingung, die man an den Kontext stellt ist, dass dieser über eine Navigation über mehrere Seiten oder über mehrere Aktionen hinweg erhalten bleiben kann. Eine andere wichtige Bedingung ist, dass der Kontext, in dem man eine Seite geöffnet hat, auch in einem Lesezeichen hinterlegt werden kann. Für die Benutzer ist es wichtig den aktuell gewählten Kontext jederzeit transparent einsehen, aber auch ändern bzw. verlassen zu können. Dazu wird im Seitenkopf die Kontextinformation präsent eingeblendet und zusätzliche Steuerungselemente zum Ändern bzw. Verlassen angezeigt.

2.5. Ableitung eines Datenmodells

Um die bisher beschriebenen Strukturen zu speichern kann ein Datenmodell für die IA abgeleitet werden. Wie man dem Datenmodell (Abbildung 2.6) entnehmen kann, sind die zwei zentralen Tabellen die Programmmodule (`mod_module`) und die Zugänge (`mod_accesses`). Ein Zugang wird auf Basis der Attribute eines Programmmoduls angelegt und enthält alle Metadaten, die für die Anzeige eines Zugangs benötigt werden. Das zugrundeliegende Programmmodul wird im Zugang gespeichert, um jederzeit auf die

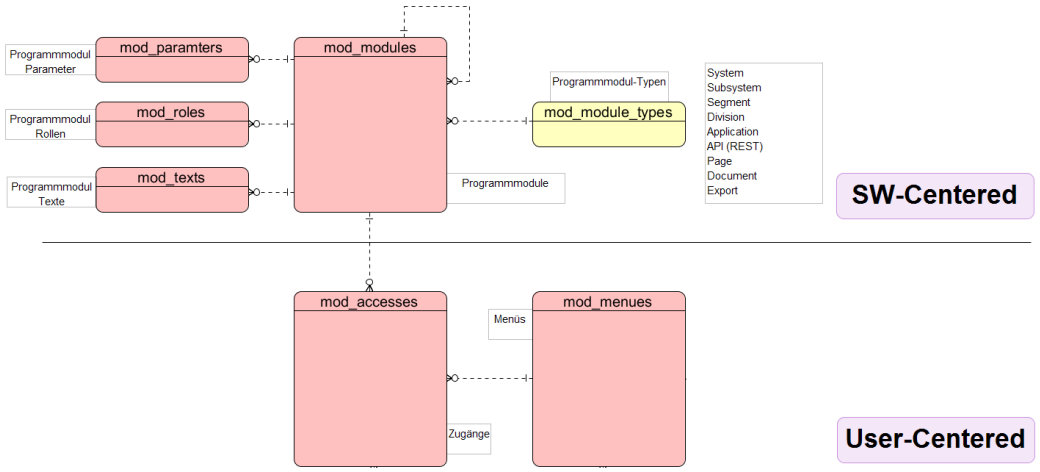

Abbildung 2.6.: Das Datenmodell mit dem die flexible Informationsarchitektur von CAMPUSonline EE abgebildet wird, umfasst als zentrale Elemente Module und Zugänge. Während die Hierarchie von Modulen die Sicht der Entwicklung und der Systemadministratoren darstellt, repräsentieren die Zugänge hingegen die Sicht der Benutzer auf das System.

originalen Attribute rückschließen zu können. Die Programmmodule sind hierarchisch (mittels Selfjoin) angeordnet und werden mittels der Tabelle (`mod_module_types`) typisiert. Sie stellen die entwicklerzentrierte Sicht auf das System dar. Die Programmmodule sind Aufhänger von diversen Konfigurationsobjekten wie zum Beispiel Systemparameter (`mod_parameters`), Benutzer Rollen (`mod_roles`) und Systemtexten (`mod_texts`). Die Zugänge (`mod_accesses`) werden dynamisch zur Laufzeit durch Systemadministratoren angelegt und stellen die benutzerzentrierte Sicht auf das System dar. Zusätzlich werden die Programmmodule in Menüs mithilfe der Tabelle Menüs (`mod_menus`) gruppiert. Durch die Tabelle Zugangsgruppen (`mod_groups`) kann eine Gruppierungsebene von Zugängen in Menüs eingezogen werden kann.

3. Ressourcenorientierter Single-Page-Architekturstil (ROSPA)

Nachdem die IA das System aus der Benutzersicht beschreibt, wird in diesem Kapitel der Architekturstil des Systems beschrieben. Ein Architekturstil definiert Grundsätze und beschreibt die grundlegenden Eigenschaften (Möglichkeiten und Einschränkungen) eines Systems, welches konform dem Architekturstil entwickelt wurde (Fielding und Taylor, 2000, S. 13). Die Wahl eines Architekturstils bringt Vor- und Nachteile mit sich. Die Konsequenzen müssen aus diesem Grunde immer im aktuellen Kontext beleuchtet bzw. alternativen Ansätzen gegenüber gestellt werden. Im Speziellen dann wenn eine Migration bzw. Integration eines bestehenden Systems durchgeführt werden soll. Ein tiefgehendes Verständnis des Altsystems ist Voraussetzung, um dieses erfolgreich und iterativ in das neue System integrieren bzw. überführen zu können (Bisbal u. a., 1997, S. 4–5).

Laut Fielding und Taylor (2000, S. 13) gibt es zwei Möglichkeiten einen Architekturstil zu entwickeln. Fielding nennt diese beiden Stile „Null Style“ und „System Needs Style“. Während der „Null Style“ von einer leeren Menge an Grundsätzen aus geht, also von Null beginnt, werden beim „System Needs Style“ die konkreten Bedürfnisse des Systems erhoben und anschließend Grundsätze und Einschränkungen für das System definiert. Der „Null Style“ eignet sich sehr gut, wenn ein allgemeiner, von einem konkreten System unabhängiger Mechanismus gesucht wird, um Grundsätze zu formulieren und die Auswirkungen dieser untereinander zu bewerten. Fielding selbst hat den „Null Style“ als Ausgangspunkt gewählt, um die Grundprinzipien des WWW zu definieren. Dadurch wird es auf einer abstrakten Ebene möglich, Weiterentwicklungen und neue Standards auf die Einhaltung der

Grundsätze zu überprüfen. Durch Fieldings Arbeit wurden zum Beispiel der Cookie-Mechanismus und die Verwendung von HTML-Framesets als nicht zum WWW passend enttarnt. Der „System Needs Style“ wird im Gegensatz dann angewandt, wenn es sich um einen Architekturstil für ein konkretes System handelt. Im Fall von CAMPUSonline wird aus diesem Grund dieses Vorgehensmodell gewählt. Ausgehend von der Produktvision (Abschnitt 1.3) und der IA (Kapitel 2) in dem die Bedürfnisse Technologie und Architekturstil neutral erhoben wurden, werden nun Schritt für Schritt Designentscheidungen für das System definiert und analysiert. Um das Rad nicht komplett neu erfinden zu müssen, empfiehlt es sich bei dieser Vorgehensweise auf bestehende, allgemein definierte Architekturstile zurückzugreifen und diese miteinander und mit den konkreten Anforderungen in Einklang zu bringen. Neben der Dokumentation und dem Aufbau einer Wissensgrundlage für neue Entwickler und Architekten, können die Grundsätze auch bei zukünftigen Designentscheidungen als Basis herangezogen werden. Des Weiteren ist es wichtig die prinzipiell gewählten Architekturstile entsprechend durch die Produktvision zu begründen, zu motivieren und die wichtigsten Stakeholder bei jeder Entscheidung an Bord zu holen. Dadurch wird verhindert, dass der definierte Architekturstil an der Realität vorbei entwickelt wird und verworfen werden muss.

Im Folgenden wird der für CAMPUSonline EE gewählte neue Architekturstil und der bestehende Architekturstil von CAMPUSonline PL/SQL kurz vorgestellt. Anschließend werden ausgewählte alternative Architekturstile diskutiert. Der folgende Hauptteil dieses Kapitels definiert und argumentiert die einzelnen Grundsätze des für CAMPUSonline EE gewählten Architekturstils.

3.1. Architekturstil von CAMPUSonline EE

Wie in der Einleitung dieses Kapitels erwähnt empfiehlt es sich auf bewährte Architekturstile aufzusetzen und diese zu verfeinern bzw. an die Bedürfnisse des Systems anzupassen. Für CAMPUSonline EE wurde ein Architekturstil namens Ressourcenorientierter Single-Page-Architekturstil (ROSPA) kreiert. Diese Wortschöpfung setzt sich zusammen aus den Begriffen:

- Ressourcenorientierter Architekturstil (ROA)
- Single-Page-Application (SPA)

Obwohl sich beide Architekturstile ideal ergänzen, werden sie in der Literatur oftmals getrennt voneinander beleuchtet. Im Kontext eines Campus-Management-Systems (CMS) müssen aber beide Stile besprochen werden. Zusätzlich müssen die Auswirkungen der beiden Stile bezüglich der Integration geprüft werden.

3.1.1. Ressourcenorientierter Architekturstil (ROA)

Der Begriff ROA wurde von Richardson und Ruby (2007, S. xiv) geprägt. ROA ist ein auf Webservices basierender Architekturstil für verteilte Systeme, welcher auf den Grundsätzen und Stärken des WWW aufbaut. Diese Grundsätze werden von Fielding und Taylor (2000) in einer Dissertation beschrieben und unter dem Begriff „Representational State Transfer“ (REST) zusammengefasst. Richardson und Ruby, 2007, S. xiv leiten in ihren Buch Gestaltungsgrundlagen für Webservices ab, welche den Architekturstil REST befolgen und in weiterer Folge als RESTful-Webservices bezeichnet werden. Die Kommunikation der verteilten Systeme erfolgt dabei durch den Austausch von Ressourcen-Repräsentationen über Webservices, die den Prinzipien von REST entsprechen. RESTful-Webservices haben aufgrund ihrer Leichtgewichtigkeit in den letzten Jahren eine sehr weite Verbreitung erfahren. Da es allerdings keinen definierten Standard gibt, ist die Einhaltung der REST Prinzipien sehr unterschiedlich ausgeprägt und oft ein Thema das heiß diskutiert wird. Zur Klassifikation der Konformität wurde ein Reifegradmodell von Richardson und Ruby (2007) eingeführt. Fielding selbst stellt in einem Interview klar (Severance, 2015), dass sich seine Arbeit auf den prinzipiellen Architekturstil von Hypertext Transfer Protocol (HTTP) bezog und es ihm aus zeitlichen Gründen dabei nicht um Richtlinien zum Entwurf von RESTful-Webservices oder sogar eines neuen Mime-Types ging. Zu einem späteren Zeitpunkt veröffentlichte Fielding in seinem Blog „REST APIs must be hypertext-driven“ (Fielding, 2008b) sechs unterstützende Richtlinien zur Gestaltung von RESTful-Webservices. Dennoch wurden diese Richtlinien und gerade die Sinnhaftigkeit des *Hypermedia*

as the Engine of Application State Prinzips (HATEOAS) bis heute heftig diskutiert. Obwohl inzwischen viele Zusammenstellungen von Richtlinien (Tilkov u. a., 2015, Giessler u. a., 2015) zur Entwicklung von RESTful-Webservices existieren, sind diese zum Teil widersprüchlich. Aus diesem Grund entstehen sowohl Alternativen zu REST wie GraphQL (Abschnitt 3.4.2) oder Weiterentwicklungen wie „Introspected REST“ (Vasilakis, 2017).

Für CAMPUSonline EE wurde definiert, dass die lose gekoppelte Kommunikation zwischen den einzelnen Modulen über RESTful-Webservices erfolgt, welche dem höchsten Level (Level 3) des Reifegradmodells von Richardson entsprechen. Das bedeutet, dass alle von Fieldings aufgestellten Grundsätzen inklusive Hypermedia as the Engine of Application State (HATEOAS) - im Folgenden in aller Kürze gelistet - eingehalten werden. Eine genauere Betrachtung erfolgt in den konkreten ROSPA Grundsätzen.

- **Client-Server ($R_{\text{CLIENT-SERVER}}$)**
  Server stellen Dienste bereit, die von einem oder mehreren Clients genutzt werden können.
- **Zustandslosigkeit ($R_{\text{ZUSTANDSLOS}}$)**
  Der Server ist zustandslos ausgelegt und bearbeitetet jede Anfrage unabhängig von anderen Anfragen.
- **Caching (R_{CACHING})**
  Um die Geschwindigkeit des Systems zu steigern werden einmal abgeholte Daten zwischengespeichert, um nicht jedes Mal erneut eine Anfrage an den Server schicken zu müssen.
- **Einheitliche Schnittstelle (R_{UNIFORM})**
  Die Schnittstellen sind einheitlich und entsprechend der folgenden vier Punkten gestaltet.
  - **Adressierbarkeit von Ressourcen (R_{RESOURCE})**
    Jede Ressource des Systems kann über ein einheitliches Schema angesprochen werden.
  - **Repräsentationen zur Veränderung von Ressourcen ($R_{\text{REPRESENT}}$)**
    Ressourcen können über einheitliche Methoden, die auf deren Repräsentation ausgeübt werden, angelegt, geändert und gelöscht werden.
  - **Selbstbeschreibende Nachrichten ($R_{\text{BESCHREIBEND}}$)**
    Das für die Übertragung der Repräsentation verwendete Format

ist einheitlich gestaltet, so dass eine automatisierte Interpretation auf verschiedenen Ebenen möglich ist.
  - **Hypermedia as the Engine of Application State ($R_{HATEOAS}$)**
    Der Applikationszustand wird durch Entscheidungsmöglichkeiten in den Repräsentation von Ressourcen gesteuert. Ein Beispiel für Hypermedia wäre die Verwendung von Links in den Repräsentationen.
- **Mehrschichtige Systeme ($R_{SCHICHTEN}$)**
  Das System besteht aus mehreren Schichten, bei dem jede Schicht definierte Aufgaben übernimmt.
- **Code on Demand ($R_{ON\text{-}DEMAND}$)**
  Der Client kann durch dynamisches beziehen von Quellcode erweitert werden.

3.1.2. Single-Page-Architekturstil

Eine SPA zeichnet sich vor allem dadurch aus, dass es nur eine einzige HTML-Seite gibt, die initial genau ein einziges Mal vom Server geladen wird. Sie verhält sich dadurch während des Gebrauchs wie ein *Fat Client* und muss nicht neu aufgebaut werden. (Mikowski und Powell, 2013) Sämtliche Änderungen der Oberfläche erfolgen ausschließlich dynamisch mittels JavaScript. Werden weitere Informationen vom Server benötigt, werden diese dynamisch beispielsweise mittels Asynchronous JavaScript and XML (Ajax) angefordert.

3.1.3. Weitere Architekturstile und Muster

Ein wichtiges Merkmal von nachhaltigen Systemen ist laut Parnas (1972) ein modularer Aufbau. Diese Eigenschaft wird im Folgenden als $SYS_{MODULAR}$ bezeichnet. Im Bereich der Zugriffskontrolle von Informationsmanagementsystemen konnten sich die zwei Architekturstile Role-Based Access Control (SEC_{RBAC}) (R. S. Sandhu u. a., 1996) und Attribute-Based Access Control (SEC_{ABAC}) (Hu u. a., 2013) durchsetzen, deren Anwendung in CAMPUSonline EE unter Kapitel 5 genauer beschrieben wird.

3.1.4. Technologieauswahl

Zur Umsetzung von CAMPUSonline EE mussten die passenden Programmiersprachen und Frameworks gefunden werden, welche die Umsetzung der zwei zugrundeliegenden Architekturstile optimal unterstützen würden.

Programmiersprachen

CAMPUSonline EE verwendet im Backend zur Abbildung der Geschäftslogik Java 8.0, während das als SPA ausgelegte Frontend auf Typescript 2.5 als Programmiersprache setzt. Durch den Einsatz von Java, als die am weitesten verbreitete Programmiersprache (TIOBE-Index, 2017), wird die Nachhaltigkeit der Codebasis von CAMPUSonline EE für die nächsten 10 bis 20 Jahre sichergestellt. Trotz der langjährigen Beständigkeit der seit 1997 verfügbaren Programmiersprache Java, gilt diese Sprache als sehr lebendig und als vielfältig einsetzbar.

Typescript ist eine quelloffene von Microsoft gepflegte Programmiersprache, die als ein typisiertes Superset von JavaScript angesehen wird. Im Endeffekt wird der transpilierte JavaScript Code im Browser ausgeführt. Während der Entwicklung hat man allerdings den großen Vorteil einer typisierten Sprache, die weniger fehleranfällig ist als natives JavaScript und Umbauarbeiten und Wartung essentiell erleichtert.

Frameworks

Serverseitig benutzt CAMPUSonline EE intensiv die namensgebende Java Platform Enterprise Edition (Java EE). Java EE ist eines der führenden Frameworks zur Abbildung von hochverfügbaren, sicheren, zuverlässigen und skalierbaren Anwendungen im Geschäftsumfeld. Durch die zahlreichen APIs der Java EE Plattform und mit dem propagierten Programmierparadigma *Convention over Configuration* ist der Entwicklungsprozess auf die zu implementierende Geschäftslogik konzentriert, statt darauf abzuzielen, technische Herausforderungen zu meistern. Vor allem die Java API for RESTful Web Services (JAX-RS) und Java Architecture for XML Binding (JAXB) zur

Serialisierung und Deserialisierung von Objekten in Extensible Markup Language (XML) bzw. JavaScript Object Notation (JSON), werden zur Umsetzung der ROA in CAMPUSonline EE intensiv benutzt. Zur Entkopplung der Interfaces von ihren Implementierungen wird das Framework Context and Dependency Injection (CDI) verwendet. Außerdem wird die Java Persistence API (JPA) für den objektrelationalen Zugriff auf die Datenbank genutzt.

Die dominierenden Frameworks auf der Clientseite sind Angular 4 und folgende Versionen zur Umsetzung der SPA und Bootstrap 3.3 zur Umsetzung der *Responsive Webdesign* folgenden Benutzeroberfläche. Angular ist ein vollwertiges Framework zur Abbildung von einfachen bis komplexen Webanwendungen. Es basiert auf Typescript ist komponentenorientiert und unterstützt viele moderne Programmierparadigmen, wie *Web-Components*, *Deklarative Templates*, *Dependency Injection* und unterstützt mit seinem Modulsystem in Kombination mit *Lazy Loading* auch die Umsetzung von sehr großen Systemen.

3.2. Architekturstil von CAMPUSonline PL/SQL

Die bestehende Architektur von CAMPUSonline setzt auf eine serverseitige Generierung von HTML-Seiten mithilfe der von Oracle stammenden proprietären Programmiersprache PL/SQL. Dabei wird der PL/SQL Programmcode direkt auf der Datenbank ausgeführt. Das System ist zur Modularisierung in mehrere Datenbankschemas aufgeteilt. Dabei kann ein Schema durch eine feingranulare Zugriffskontrolle einem anderen Schema Zugriff auf Packages und Datenbank Views gewähren, die als Schnittstelle zwischen den zwei Schemas dienen. Das Framework für den Aufbau der Benutzeroberfläche ist ebenfalls in PL/SQL verfasst. Es kapselt die Generierung von HTML und JavaScript und ist komponentenorientiert aufgebaut. Die Generierung einer HTML-Seite erfolgt beinahe zustandslos. Das heißt der aufgebaute Komponentenbaum wird NICHT im Speicher gehalten, um den Zustand des Clients zu verwalten. Stattdessen wird der Komponentenbaum bei jeder HTTP-Anfrage neu aufgebaut, abschließend gerendert und geleert. Der Applikationszustand wird dabei ausschließlich über die

URL bzw. die URL-Parameter abgebildet. Nur die angemeldete Person, deren Profil und die aktuelle Sprache werden in der Applikationssitzung gehalten.

3.2.1. Laufzeitsicht

Mit dem Browser wird eine URL aufgerufen und eine HTTP-Anfrage an den Web-Server gesendet. Im Fall von CAMPUSonline PL/SQL ist der Web-Server eine Instanz von Apache, die ein spezielles Plug-In lädt. Das „mod-plsql“-Plug-In - mit der entsprechenden Konfiguration und Installation auf der Datenbank - ermöglicht es dem Web-Server eine Anfrage direkt an die Oracle-Datenbank weiterzuleiten und ein Mapping auf bestimmte PL/SQL-Prozeduren durchzuführen. Die PL/SQL-Prozeduren füllen während des Programmablaufs einen Zwischenspeicher mit Zeichen. Dieser wird als HTTP-Antwort über den Web-Server wieder an den Client zurückgegeben. In welcher Form innerhalb der aufgerufenen PL/SQL-Prozeduren Daten manipuliert oder ermittelt werden, wird nicht weiter festgelegt.

3.3. Migrationsstrategie

Das Ziel von CAMPUSonline ist eine schrittweise aber vollständige Migration des bestehenden Systems - CAMPUSonline PL/SQL - auf den neuen CAMPUSonline EE Technologie-Stack. Um den Kooperationspartnern immer ein vollwertiges und lauffähiges Produkt anbieten zu können, müssen beide Systeme auf verschiedenen technologischen Ebenen flexibel miteinander verbunden werden. Der Endbenutzer soll dank eines einheitlichen Nutzererlebnis möglichst nichts von der Integration der beiden Systeme bemerken.

Bisbal u. a. (1997) führen in ihrer Arbeit sechs Methodologien an, die für System-Migrationen verwendet werden können. Aufgrund der gewünschten iterativen Vorgehensweise und dem Wunsch trotzdem immer den vollen Funktionsumfang ausliefern zu können, kann kein *Big Bang Ansatz* gewählt werden. Aus der für CAMPUSonline PL/SQL beschriebenen Architektur

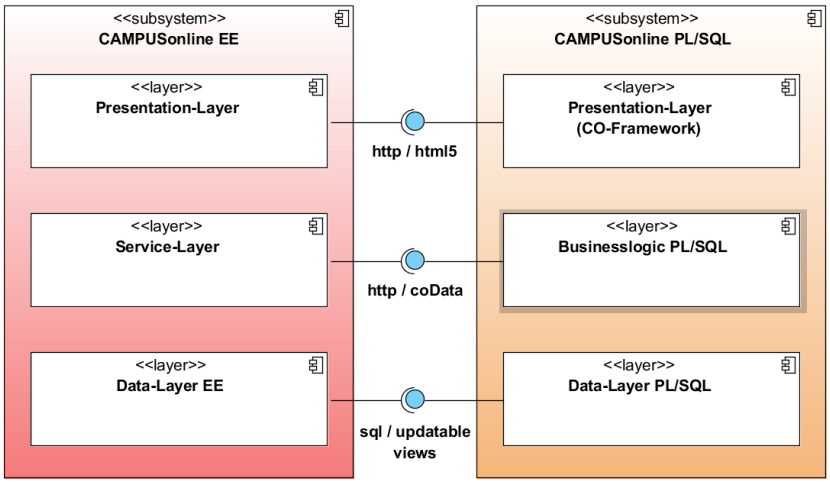

Abbildung 3.1.: Die Integration von CAMPUSonline PL/SQL erfolgt mittels drei Arten von Gateways auf unterschiedlichen Schichten der beiden Technologie-Stacks.

erfahren wir, dass das System direkt in der Datenbank implementiert wurde und es somit als schwer zerlegbar bzw. monolithisch gilt. Für schwer zerlegbare Systeme wird die Strategie *Chicken Little* von Bisbal u. a. (1997) empfohlen. Dabei werden drei Arten von Gateways eingeführt, die auf unterschiedlichen Ebenen als Verbundelemente zwischen den beiden Systemen dienen.

- **Datenbank-Gateway (CL_{DB})**
  Können als Vorwärts- oder Rückwärts-Gateways betrieben werden und adaptieren Structured Query Language (SQL) Anfragen passend für das jeweilige Zielsystem.
- **Applikations-Gateway (CL_{APP})**
  Kapseln Applikationslogik und stellen diese dem Zielsystem zur Verfügung.
- **Informationssystem-Gateway (CL_{IS})**
  Die Gesamtfunktionalität des Altsystem wird gekapselt in das neue System integriert.

Um dieses Ziel zu erreichen wurden in CAMPUSonline erfolgreich **genau** diese drei Arten von Gateways für die verschiedenen Ebenen der Schichtenarchitektur von CAMPUSonline EE implementiert, die je nach Bedarf und

auch in einer Mischform eingesetzt werden können. (Abbildung 3.1) Wie diese drei Arten von Gateways in CAMPUSonline umgesetzt wurden, kann den Grundsätzen in den Abschnitten 3.6.9, 3.6.10 und 3.7.10 entnommen werden.

3.4. Alternative Architekturstile

Bei der Ableitung eines eigenen Architekturstils und um die eigenen Entscheidungen zu festigen kann das Analysieren alternativer Ansätze eine wichtige Rolle spielen. Zusätzlich können Elemente aus konkurrierenden Ansätzen in den eigenen Architekturstil einfließen.

3.4.1. ROCA-Style

Einer der wenigen Architekturstile, der sowohl die Client- als auch die Serverseite beleuchtet ist der ROCA-Style. ROCA steht für Ressourcenorientierte Client Architektur (ROCA) und baut auf den Grundprinzipien von REST auf. ROCA selbst positioniert sich als Alternative zu aktuellen Trends in Richtung SPAs und statusbehafteten, serverseitigen Komponenten-Frameworks (Hoppe, Schulte-Coerne und Tilkov, 2012). Bei ROCA besinnt man sich vielmehr auf ein klassisches Umsetzungsmodell bei Webseiten das aber strikt den Prinzipien von REST folgt. Schulte-Coerne u. a. (2012) formulierten einige Grundsätze, die den Architekturstil ROCA begründen. Anstatt nur Daten in Form von JSON/XML mit dem Server auszutauschen, werden HTML-Seiten durch den Server zur Verfügung gestellt. Die Seiten beinhalten bereits die Daten in semantisch richtigem HTML, welches unabhängig vom Layout und Verhalten ist. Mittels des Musters *Unobtrusive JavaScript* wird der übertragene HTML-Quelltext interpretiert und in eine benutzerfreundlicheren Form dargestellt. Mit Cascading Style Sheets (CSS) wird das Design der Seite entsprechend aufgewertet. Sowohl das ausgelieferte JavaScript als auch die CSS-Dateien müssen dabei in separaten statischen Dateien ausgeliefert werden.

Tilkov (2016) kritisiert an SPAs vor allem, dass viele Standard-Browserfunktionen nicht oder nur eingeschränkt funktionieren. Außerdem wird häufig die lange initiale Ladezeit einer SPA sowie die zwingend Verwendung von JavaScript zur Darstellung der SPA kritisiert. Die fehlende Unterstützung von Standard-Browserfunktionen, wie Zurück-Button und Lesezeichen, werden in ROSPA hingegen zum Grundsatz erhoben und können in einer SPA mit einmaligen Aufwand entsprechend umgesetzt werden. (siehe §BROWSER). Des Weiteren stellt die Unterstützung von Browsern ohne aktiviertem JavaScript für CAMPUSonline EE kein verfolgenswertes Ziel dar, da CAMPUSonline PL/SQL JavaScript bereits zwingend voraussetzt. Die Kritik der initialen Ladezeit lässt sich anhand der Strategie hinsichtlich Einzelfenster-Navigation und der schnelleren Reaktionszeit der Benutzeroberfläche, sobald die SPA geladen wurde, stark abschwächen.

Aufgrund der klaren Trennung zwischen Daten und HTML wurde für CAMPUSonline entschieden den Weg in Richtung SPA einzuschlagen. Aus der Erfahrung mit CAMPUSonline PL/SQL wurde ersichtlich, dass sobald in der serverseitigen Entwicklung HTML-Quelltext verwendet werden kann, dieser sich gerne über alle Schichten bis in die tiefste Geschäftslogik verteilt. Um diese Durchmischung von Beginn an zu unterbinden, wird ein klar definiertes Schnittstellen-Protokoll, das unabhängig von HTML bevorzugt. Außerdem können durch die Verwendung von SPA Programmiertechniken Daten, die einmal geladen bzw. durch die Benutzer geändert wurden einfach im Speicher des Browsers verwaltet werden (Baschuk, 2016). Dadurch werden wiederholte Anfragen an den Server verhindert, was zu einer deutlichen Entlastung der Server-Ressourcen und zu einer wesentlich schnelleren Benutzeroberfläche führt. Aufgrund der immerwährenden Steigerung der Rechenleistung der Clients profitiert die Skalierbarkeit der Gesamtarchitektur durch die Entlastung der Serverseite. Gerade im Hinblick darauf, das viele Sichten von Sachbearbeitern, welche zentrale und komplexe Prozesse einer Bildungseinrichtung verwalten, eher Anforderung in Richtung komplexer und interaktiver Desktop Applikationen mit sich bringen, wurde für eine Umsetzung des Frontends mittels SPA Technologien entschieden.

Dennoch sind viele der Eigenschaften von ROCA, mit Ausnahme der Auslieferung von semantischen HTML und dem Einsatz von *Unobtrusive JavaScript*, in ROSPA berücksichtigt worden.

3.4.2. GraphQL

GraphQL ist ein von Facebook im Jahr 2012 entwickeltes Framework für Web-APIs. Die erste Referenzimplementierung wurde von Facebook im Jahr 2016 veröffentlicht und hat seit diesem Zeitpunkt eine relativ hohe Verbreitung gefunden (Hartig und Pérez, 2017). Im Kern von GraphQL steht ein Schema (ein typisiertes Datenmodell und Relationen zwischen Entitäten), welches mit einer sehr flexiblen Abfragesprache - ähnlich SQL - abgefragt werden kann. Durch Relationen im Datenmodell ergibt sich ein Graph an abfragbaren Daten. Die Antwort ist eine verschachtelte JSON-Struktur, welche exakt jene Attribute des Graphs beinhaltet, die der Client auch tatsächlich angefragt hat (Eizinger, 2017). GraphQL ist wie SOAP Transport unabhängig und tunnelt typischer Weise jede Anfrage des Server durch die HTTP-POST-Methode. Eizinger (2017) vergleicht in seiner Arbeit REST und GraphQL hinsichtlich mehrerer Kriterien. Dabei stellte er fest, dass GraphQL eine Lösung für eine ganz konkrete Problemklasse darstellt, wohingegen REST viel allgemeiner in seiner Definition als Architekturstil ist. Eine Architektur, die auf GraphQL aufbaut, neigt dazu Geschäftslogik und Prozesslogik eher auf die Clientseite zu verlagern und den Server als generischen Datenlieferanten zu nutzen. Während es bei einer ROA darum geht, dass der Client wie in einer Choreographie durch den Server gesteuert wird. Der Client Entwickler muss die Struktur des Servers gut kennen, bekommt dafür aber auch eine auf seine Bedürfnisse zugeschnittene Antwort. Im Falle von CAMPUSonline wird die Steuerung eines generischen Clients, die Fokussierung auf die Grundarchitektur des WWW und vor allem die Implementierung der Logik auf der Serverseite zugunsten einer zukunftssicheren Architektur bevorzugt.

Dennoch können einige Stärken von GraphQL ebenfalls in ROSPA genutzt werden. Zum Beispiel die Verwendung einer generischen Abfragesprache, die Abfragen von verschachtelten Strukturen mittels dynamisch einbettbaren Links (siehe §PROTOCOL) und vor allem eine typisierte Schnittstelle bietet (siehe §CLIENT-TYPES), wenn eine benötigt wird.

3.5. Allgemeine Grundsätze

Die Grundsätze des Architekturstils ROSPA werden in allgemeine, serverseitige und clientseitige Grundsätze unterteilt. Bei jedem Grundsatz hilft ein kurzer Steckbrief in tabellarischer Form, um schnell einen Überblick über dessen Eigenschaften zu erhalten. Im Steckbrief wird angegeben wie der vorgestellte Grundsatz abgeleitet wurde, welche anderen Grundsätze vorausgesetzt werden und welche Ziele oder Grundsätze dadurch gefördert oder gefährdet werden. In der Tabelle werden Abkürzungen verwendet, welche im Zuge der Arbeit eingeführt und im Anhang A übersichtlich zusammengefasst werden.

3.5.1. Modularisierung ($\S_{\text{MODUL}}$)

abgeleitet von	benötigt	fördert	gefährdet
$\text{SYS}_{\text{MODULAR}}$	-	$\text{Z}_{\text{ARCHITEKTUR}}$	-
		$\text{Z}_{\text{QUALITÄT}}$	

Tabelle 3.1.: Steckbrief von Modularisierung ($\S_{\text{MODUL}}$)

Das System ist aus austauschbaren und hierarchisch angeordneten Modulen aufgebaut, die sich gegenseitig möglichst nicht beeinflussen.

Das primäre Ziel fordert eine zukunftssichere Architektur die mindestens 15 Jahre Bestand hat. Dass dies im Bereich von Informationsmanagementsystemen Herausforderungen mit sich bringt, zeigten die Entwicklungen der letzten 15 Jahre. Aus diesem Grund ist ein modularer Aufbau von CAMPUSonline EE wichtig. Bei einem modularen System können einzelne Teile des Systems aufgrund der hierarchischen Organisation der Module einfach hinzugefügt, entfernt oder ausgetauscht werden. Ein anderer wichtiger qualitativer Aspekt der Modularisierung, ist eine erhöhte Systemstabilität, da bei einer losen Kopplung der einzelnen Module untereinander, mit geringeren Seiteneffekten zu rechnen ist, wenn ein Modul fehlerhafte Implementierungen aufweist.

Aufgrund der zentralen Bedeutung der Modularisierung wird im Kapitel 4 dies Thema dezidiert aufgearbeitet.

3.5.2. Mehrschichtiges System (§LAYER)

abgeleitet von	benötigt	fördert	gefährdet
RSCHICHTEN	§MODUL	ZARCHITEKTUR	ZUSABILITY
		ZINTEGRATION	
		ZQUALITÄT	
		ZSICHERHEIT	

Tabelle 3.2.: Steckbrief von Mehrschichtiges System (§LAYER)

Das System ist in Schichten strukturiert, wobei jede Schicht nur die direkt darunter liegende Schicht kennt.

Die Strukturierung einer Architektur in Schichten ist ein Grundprinzip von REST. Dabei bezieht sich die Strukturierung sowohl auf die Serverlandschaft (Datenbank, Applikationsserver, Web-Server, Proxies), als auch auf die Strukturierung des Quellcodes.

Technologien und Frameworks ändern sich im Bereich von Informationssystemen rasant. Während Datenmodelle und Datenbanktechnologien relativ langlebig sind, wird im Bereich von Benutzeroberflächen-Technologien fast jedes Jahr ein neuer Hype losgetreten. Auch die Abhängigkeit gegenüber speziellen Anbietern von Frameworks oder Infrastrukturen, welche tief verwurzelt im Kernprodukt sind, ist bezüglich der Ziele der Langlebigkeit und Portabilität von Softwaresystemen sehr kontraproduktiv. In Zeiten in denen sich Benutzeroberflächen-Technologien radikal verändern, ist es wichtig, dass diese einfach abgelöst bzw. ausgetauscht werden können. Im Sinne einer Schichtenarchitektur, welche ein Kernprinzip von REST darstellt, wurde eine saubere Trennung zwischen Client, Server und Datenbank vorgenommen. Der Client wurde als SPA mit Angular umgesetzt, der Server baut auf den weit verbreiteten Standard Java EE auf. Die Kommunikation

zwischen Client und Server erfolgt über RESTful-Webservices. Auch der Quellcode am Server ist in drei technische Schichten geteilt: REST-Schicht, Service-Schicht, Persistenz-Schicht. Neben der vertikalen fachlichen Modularisierung von CAMPUSonline, wird auch die horizontale technische Skalierung im Kapitel 5 näher beschrieben.

Durch die Strukturierung in Schichten können Altsysteme, dank definierter Schnittstellen zwischen den Schichten, einfach und sukzessive gegen eine Neuimplementierung ausgetauscht werden. Gleichzeitig kann beim Einklinken eines Altsystems - wie im Abschnitt 3.6.9 beschrieben - verhindert werden, dass der Client direkt auf das Altsystem durchgreift und neue Abhängigkeiten zum Altsystem entstehen.

Des Weiteren kann durch Schichten die maximale Komplexität der Implementierung kontrolliert werden. Das heißt die Wartbarkeit wird durch eine Trennung der Zuständigkeiten entsprechend gesteigert. Auch aus der Perspektive der Sicherheit bietet eine Strukturierung in Schichten Vorteile. So können exakt definierte Schichten im System etabliert werden, die sich um verschiedene Sicherheitsaspekte kümmern. Ein Beispiel hierfür ist der Secure Reverse Proxy, welcher sich um SSL-Terminierung und Sicherheitsmechanismen auf HTTPS-Protokollebene kümmert.

Der Nachteil bei einer Strukturierung in Schichten ist die potentiell negative Auswirkung hinsichtlich der Usability, da die Verarbeitungsgeschwindigkeit durch jede eingeführte Schicht reduziert wird. Durch entsprechende Caching Mechanismen in den verschiedenen Ebenen, kann diese negative Auswirkung allerdings gemindert werden.

3.6. Serverseitige Grundsätze

3.6.1. Ressourcen (§RESOURCES)

Jegliche Funktionalität des Systems wird mittels einheitlich adressierbaren Ressourcen angeboten.

abgeleitet von	benötigt	fördert	gefährdet
$R_{RESOURCE}$	$\S_{MODUL}$	$Z_{ARCHITEKTUR}$	-
	$\S_{LAYER}$	$Z_{INTERFACE}$	
		$Z_{INTEGRATION}$	

Tabelle 3.3.: Steckbrief von Ressourcen ($\S_{RESOURCES}$)

Ressourcen sind das zentrale Element einer ROA und stellen das Grundprinzip des WWW dar. Während in der Vergangenheit viele Architekturstile versucht haben das WWW aus der Web-Applikationsentwicklung zu verbergen (SOAP, JavaServer Faces (JSF), ...), besinnt man sich seit einigen Jahren wieder auf die Kernprinzipien von REST. Auch aktuelle Ansätze in der Webentwicklung, wie WebSockets oder GraphQL, schlagen hier einen alternativen Weg ein und setzen nicht auf dem Grundprinzip von Ressourcen auf. Fielding (2008b) der in seiner Dissertation diesen Architekturstil begründete, sagt selbst „REST is software engineering on the scale of decades“. Um den Grundstein für eine zukunftssichere Architektur ($Z_{ARCHITEKTUR}$) zu legen wurde aus diesem Grund REST gewählt.

Im Kontext der zunehmenden Vernetzung verschiedenster Services im Bereich der Campus-Management-Systeme erhält das Thema Schnittstellen eine zentrale Bedeutung. Die Strategie des neuen Systems fordert eine klare Fokussierung auf einen definierten Funktionsumfang im Bereich des SLC. Aufgrund dieser Strategie werden einige Funktionalitäten, die in CAMPUSonline PL/SQL integriert wurden durch externe Systeme abgelöst, die über Schnittstellen mit CAMPUSonline gekoppelt werden müssen. Des Weiteren gibt es in Österreich und Deutschland die Bestrebung interuniversitäre Studien anzubieten. Das heißt Studierenden soll es ermöglicht werden einen Studienplan über mehrere Universitäten hinweg studieren zu können. Der primäre Weg, der hier von CAMPUSonline und auch anderen Campus-Management-Systemen eingeschlagen wird, ist der Austausch von Ressourcen auf Basis von RESTful-Webservices.

Das bestehende CAMPUSonline PL/SQL System wurde zum Großteil auf Basis von serverseitig generierten HTML-Ressourcen aufgebaut. Es verletzt teilweise das Konzept der eindeutigen Adressierbarkeit, an Stellen wo mit-

tels Ajax Teile der Seite aktualisiert werden. Aufgrund der REST Basis kann das bestehende Altsystem trotzdem gut in die neue Architektur integriert werden.

Eine Ressource kann theoretisch alles sein, muss aber zumindest über eine URL und eine Repräsentation verfügen. Meist sind Ressourcen Objekte der jeweiligen Fachdomäne (Lehrveranstaltung, Prüfung, ...). Ein und dieselbe Information kann dabei unter verschiedenen URLs zugänglich gemacht werden. Allein die Semantik entscheidet, ob eine neue Ressource benötigt wird oder eine bestehende wieder verwendet werden soll. Aufbauend auf der Schichtenarchitektur (§LAYER), wurde für CAMPUSonline EE entschieden, die Präsentationsschicht und die Geschäftslogik sowohl des Altsystems als auch des neuen Systems unter folgenden Basis URLs zugänglich zu machen:

`<base>/ee/ui`	Benutzeroberfläche von CAMPUSonline EE
`<base>/ee/rest`	Geschäftslogik von CAMPUSonline EE
`<base>/pl/ui`	Benutzeroberfläche von CAMPUSonline PL/SQL
`<base>/pl/rest`	Geschäftslogik von CAMPUSonline PL/SQL

Tabelle 3.4.: Basis URLs von CAMPUSonline EE und CAMPUSonline PL/SQL

Ein sehr wichtiger Aspekt der Integration ist die Verwendung der gleichen Basis-URL. (z.B. `https://online.tugraz.at/tugonline`). Dadurch wird es möglich, dass die beiden Subsysteme uneingeschränkt der Sicherheitsvorkehrungen des Browsers (Same Origin Policy) miteinander auf der Ebene des Clients kommunizieren können. Die konkrete Integration auf Browser Ebene wird unter §INTEGRATION aber auch unter §LEGACY beschrieben. Zusätzlich werden Ressourcen ausgehend von der Modularisierung (§MODUL) in einzelne APIs inhaltlich zusammengefasst. Zum Beispiel werden Informationen und Operationen auf Lehrveranstaltungen einer Bildungseinrichtung unter der API `<base>/ee/rest/slc.tm.cs` gebündelt zugänglich gemacht. Einerseits kann durch diese Konventionen die Semantik einer URL durch einen Entwickler schnell kategorisiert werden, andererseits kann mittels OAuth2 die Basis-URL der API als OAuth2-Scope genutzt werden. Obwohl auf Basis des URL-Musters die Semantik einer

URL transparent nachvollzogen werden kann, soll der Client aufgrund dieses Wissens keine Entscheidungen treffen. Der Server muss immer dazu in der Lage sein, URLs nach Belieben zu ändern, ohne dass der Client angepasst werden muss. Im Abschnitt §HATEOAS wird näher auf dieses Thema eingegangen.

3.6.2. Einheitliche Zugriffsmethoden (§ZUGRIFF)

abgeleitet von	benötigt	fördert	gefährdet
RREPRESENT	-	ZINTERFACE	-

Tabelle 3.5.: Steckbrief von Einheitliche Zugriffsmethoden (§ZUGRIFF)

Ressourcen können durch die Anwendung eines kleinen Sets an Methoden auf ihre Repräsentation abgefragt, manipuliert bzw. gelöscht werden.

Die verwendeten Methoden besitzen eine exakt definierte Semantik. Das HTTP-Protokoll definiert zum Beispiel GET, POST, PUT, DELETE, HEAD, OPTIONS als die wichtigsten Methoden für den Zugriff auf Ressourcen. Jegliche Funktionalität, welche ein RESTful-Webservice anbietet muss mittels dieser sechs Methoden, auf eine Repräsentation einer Ressource angewandt, realisiert werden. Oft wird REST auch mit dem Nominalstil verglichen, da es nur wenige Methoden (Verben) dafür aber viele Ressourcen (Nomen) verwendet.

Methoden können aufgrund ihrer Wirkung auf Ressourcen laut Richardson und Ruby (2007, S. 102) wie folgt klassifiziert werden. Wenn sie den Zustand einer Ressource nicht verändern werden sie als *sicher* bezeichnet. Zum Beispiel kann eine Suchmaschine gefahrlos die Web-API indexieren, da sie nur über die sichere HTTP-Methode GET auf Ressourcen zugreift. Idempotente Methoden (alle Methoden außer POST) können ohne Seiteneffekte wiederholt aufgerufen werden. Diese Wiederanlauffähigkeit ist in verteilten Systemen besonders erstrebenswert.

3.6.3. Zusätzliche Formate (§FORMATS)

abgeleitet von	benötigt	fördert	gefährdet
#SHOULD-FORMATS	§PROTOCOL	ZARCHITEKTUR ZINTERFACE	-

Tabelle 3.6.: Steckbrief von Zusätzliche Formate (§FORMATS)

Eine Ressource soll unter ihrer URL in verschiedenen Formaten abgerufen bzw. manipuliert werden können, da Clients unterschiedliche Bedürfnisse und Möglichkeiten haben Formate zu parsen bzw. darzustellen.

Der Client der Web-API soll mittels *Content-Negotiation* entscheiden können, mit welcher Repräsentation er arbeiten möchte. Repräsentation bezieht sich nicht nur auf das Datenformat, sondern genauso auf die Sprache und den verwendeten Zeichensatz. *Content-Negotiation* ist der Mechanismus des HTTP-Protokolls, der es einem Client erlaubt über den *Accept-Header* dem Server mitzuteilen, welche Repräsentation einer Ressource er bevorzugt. Umgekehrt teilt der Client dem Server mittels des *Content-Type-Header* mit, welches Format die Repräsentation besitzt, die der Client an den Server schickt. Es ist zu empfehlen, dass Repräsentationen, die in einem bestimmten Format vom Server abgeholt wurden, im selben Format an den Server wieder zurück geschickt werden können. Es ist davon auszugehen ist, dass der Client die abgeholte Struktur sehr einfach manipulieren kann. Für das Altsystem kann dies allerdings nicht mehr bewerkstelligt werden, da nur klassische HTML-Formulardaten unterstützt werden.

Bei RESTful-Webservices ist das JSON Format sehr weit verbreitet , da sich dieses optimal in JavaScript Applikationen integrieren lässt. Aber auch XML wird standardmäßig angeboten, weil dieses Format ausdrucksstärker sowie validierbar ist. Zusätzlich wird die XML-Verarbeitung durch eine Vielzahl an mächtigen Werkzeugen unterstützt. Auch für das Altsystem ist XML das bessere Format, da PL/SQL in einer Oracle Datenbank Umgebung XML besser verarbeiten kann. JSON hat wiederum den Vorteil, dass es sehr effizient und einfach in JavaScript interpretiert werden kann und

eine geringere Bandbreite benötigt. Während sich die meisten RESTful-Webservices auf JSON und XML beschränken, ist es für ein CMS besonders wichtig, alternative Formate für Exporte zu unterstützen. Klassischerweise sind das Formate wie CSV, Excel, einfaches semantisches HTML aber auch PDF.

Es gibt mehrere gängige spezialisierte Repräsentationsformate zur Umsetzung von RESTful-Webservices. Es konnte sich allerdings noch kein internationaler Standard etablieren. Als Beispiel seien hier Hypertext Application Language (HAL) (Kelly, 2016), Atom (M. Nottingham, 2005) und Siren (Swiber, 2017) erwähnt. Jeder dieser Media-Types hat Vorteile und Nachteile und wurde für unterschiedliche Client-Anforderungen konstruiert. Der tatsächlich verwendete primäre Media-Type rückt jedoch in den Hintergrund, da die entsprechenden unterschiedlichen Formate bei Bedarf mittels *Content Type Negotiation* zur Verfügung gestellt werden können. Wichtig ist hingegen, dass alle Metainformationen die benötigt werden, das heißt Daten sowie Links zu anderen Ressourcen, verfügbar sind, um alternative Formate angeboten werden können. Dank JAX-RS 2.0 kann dies mit einem *ContainerFilter* sehr einfach umgesetzt werden. Gerade mit der vor kurzem eingeführten Java API for JSON Processing (JSON-P) ist eine Umsetzung unterschiedlichster JSON-Formate im Java EE 8 Standard ohne Probleme möglich.

3.6.4. Einheitliche Schnittstellen (§PROTOCOL)

abgeleitet von	benötigt	fördert	gefährdet
RBESCHREIBEND	§RESOURCES	ZARCHITEKTUR	-
#HTTP	§REPRESENT	ZINTERFACE	

Tabelle 3.7.: Steckbrief von Einheitliche Schnittstellen (§PROTOCOL)

Über das ganze System hinweg wird standardmäßig ein einheitlicher Basis Media-Type verwendet.

Einer der zentralen Punkte von Fieldings Grundsatz einer einheitlichen Schnittstelle zum Austausch von Ressourcen ist die Verwendung von selbstbeschreibenden Nachrichten. Nachrichten sollen durch einen Client automatisiert interpretiert werden können, genauso wie ein Browser den Inhalt eines HTML-Dokumentes präsentieren kann, ohne dessen Inhalt zu verstehen. Fielding (2015, S. 37) behauptet sogar „A RESTful API is just a website with limited vocabulary." Der meiste Aufwand bei der Definition einer RESTful API soll laut Fielding (2008b) in die Definition des Media-Types und der Definition von Link Relationen investiert werden, sodass ein Client diese automatisiert interpretieren kann. Der Kern der Aussage ist, dass bei einer Änderung der API der Client-Code nicht durch einen Entwickler angepasst werden muss, sondern der Client flexibel auf die Änderungen reagieren kann. Nur dadurch kann gewährleistet werden, dass sowohl die Geschäftslogik als auch die Prozesslogik serverseitig umgesetzt werden und nicht beim Austausch des Clients verloren gehen.

Das Format soll selbstbeschreibend und flexibel erweiterbar sein.

Vasilakis (2017) definiert in seinem Architekturstil *Introspected REST*, dass moderne RESTful APIs durch das Einklinken von Mikroformaten erweitert werden können, um sich selbst zu beschreiben. Dabei soll ein möglichst einfacher erweiterbarer Basis-Media-Type geschaffen werden, der von jedem Client interpretiert werden kann. Zusätzlich soll es aber intelligenten Clients möglich sein zu einer Ressource zusätzliche Informationen zu beziehen, welche die Semantik der Ressource beschreiben. Die eingebetteten Informationen können vielfältig sein, sollten jedoch möglichst einem bereits etablierten Standard, wie JSON-Schema (Pezoa u. a., 2016) zur Definition der Datentypen oder JSON-LD (Lanthaler und Gütl, 2012) zur Klärung der Semantik der Ressource folgen. Wichtig ist auf alle Fälle, dass es sich um ein allgemeines und wohl definiertes Mikroformat handelt, welches vom Client interpretiert werden kann, wenn er die Ressource verarbeiten will. Vasilakis (2017) setzt auf die Verwendung der HTTP-OPTIONS-Methode um dem Client die für eine Ressource zur Verfügung stehenden Mikroformate mitzuteilen. In CAMPUSonline wird hingegen der traditionelle Hypermedia Weg von verlinkten Mikroformaten in der Ressourcen-Repräsentation gewählt. Inwiefern Mikroformate in CAMPUSonline verwendet werden, kann den Grundsätzen §DATA-VS-DESIGN entnommen werden.

coData Protokoll

Das Collection Data Protocol (coData) ist ein sehr einfach aber zugleich mächtiges Protokoll, welches von bekannten bestehenden Protokollen abgeleitet wurde. Es stellt den Basis-Media-Type dar, welcher mittels Mikroformaten flexibel erweitert werden kann. Als Basis-Media-Type wird sowohl XML (application/codata+xml) als auch JSON (application/codata+json) unterstützt. Einerseits wurde coData an das Protokoll „Collection+JSON" (Amundsen, 2013) angelehnt, dessen Ressourcen immer eine strukturierte Liste von Einträgen mit Links zurück liefert. Andererseits wurde die Normierung der Abfragesprache des oData-Protokolls (Kirchhoff und Geihs, 2013) übernommen. Daraus resultiert auch der Name des coData-Protokolls, welcher von der Langform *Collection Data Protocol* abgeleitet wurde. Durch die Normierung der Abfragesprache kann über ganz CAMPUSonline hinweg einheitlich sortiert, gefiltert und paginiert werden. Zusätzlich können auch Links eingebettet werden, indem sie mit dem *expand Parameter* ähnlich wie bei oData eingebettet werden. Entscheidend für das gewählte coData-Format war eine möglichst einfache und generische Umsetzung mit JAXB, welches in Java EE als Standard zur Serialisierung und Deserialisierung von XML und JSON dient. Die Struktur von coData ist wie beschrieben immer listenartig, da in der Benutzeroberfläche auch sehr oft Listen oder Tabellen zur Darstellung von CAMPUSonline-Daten verwendet werden. Der Zugriff auf eine coData-Struktur kann immer in derselben Art und Weise erfolgen, ohne zu prüfen, ob es sich um eine einzelne Ressource oder um eine Listenressource handelt. Zusätzlich kann eine Ressource die initial als einzelne Ressource gedacht war ohne den Client zu brechen zu einer Listenressource erweitert werden. Außerdem erscheint die Umsetzung zweckmäßig, wenn man bedenkt dass relationale Datenbanken auch immer zweidimensionale Ergebnismengen liefern. Ressourcen beinhalten neben den eigentlichen Daten zusätzlich auch Links. Die Grundstruktur des in Java mittels eines *Generics* abgebildeten coData-Container-Formats sieht wie folgt aus:

```
1 <resources>
2   <link rel="" href="">
3   ...
4   <resource>
5     <link rel="" href="" .../>
6     ...
```

```
 7       <content>
 8         ...
 9       </content>
10     </resource>
11     <resource>
12       <link rel="" href=""  .../>
13       ...
14       <content>
15       ...
16       </content>
17     </resource>
18     ...
19   </resources>
```

Die semantische Abbildung in JSON ist bei coData exakt ident mit der des XML Media-Types, was nicht bei allen Protokollen durchgehend der Fall ist. Zum Beispiel weicht die Darstellung von Links in HAL+JSON von der in HAL+XML erheblich ab.

Eine weitere wichtige Entscheidung beim Entwurf des coData-Protokolls war, dass eine mittels HTTP-GET abgeholte Ressource genau in der gleichen Form per HTTP-PUT oder HTTP-POST an den Server zurück geschickt werden kann. Dies erleichtert den Umgang beim Zugriff mittels eines REST-Clients, da nur jene Daten geändert werden müssen, die man auch zu ändern beabsichtigt. Viele der gängigen Protokolle arbeiten hingegen mit *Templates*, die ausgefüllt werden müssen (z.B. *Collection+JSON*) oder definieren gar nicht wie sich der Media-Type im schreibenden Fall verhält (z.B. HAL). Die Information, welche Daten nun im Falle von einem PUT oder POST tatsächlich angegeben werden sollen, werden mit *Stylesheets* (siehe Abschnitt §STYLE) umgesetzt, die flexibel je nach Anwendungsfall gestaltet werden können. Wichtig ist allerdings dabei, dass die *Stylesheets* für den jeweiligen Anwendungsfall genormt sind, damit der Client die *Stylesheets* auch entsprechend automatisiert interpretieren kann.

3.6.5. Zustandsloser Server (§NO-STATE)

Der Server ist zustandslos und kennt den Applikationszustand des Clients nicht. Jede HTTP Anfrage ist atomar und isoliert ausführbar.

abgeleitet von	benötigt	fördert	gefährdet
$R_{\text{ZUSTANDSLOS}}$	-	$Z_{\text{ARCHITEKTUR}}$	-
$\#_{\text{SESSION}}$		$Z_{\text{QUALITÄT}}$	

Tabelle 3.8.: Steckbrief von Zustandsloser Server ($\S_{\text{NO-STATE}}$)

Die Zustandslosigkeit des Servers wird von Fielding und Taylor (2000) als eine der wichtigsten Eigenschaft einer ressourcenorientierte Architektur genannt, da eine Anwendung dadurch sehr einfach horizontal skaliert werden kann. Gerade für sehr große Universitäten mit mehr als 30.000 Studierenden oder auch in interuniversitären Verbünden sind so keine Limitierungen gesetzt. Des soll dabei der gesamte Applikationszustand im Client und niemals am Server werden. Jegliche Information, die der Sever braucht um einen Anfrage abzuarbeiten, muss in der Anfrage selbst enthalten sein. Auf eine Web-Sitzung muss verzichtet werden, um das Prinzip der Zustandslosigkeit zu erfüllen.

Ein Server ist dann zustandslos, wenn keine Web-Sitzung zwischen Client und Server aufgebaut wird. Für eine genauere Betrachtung wird zuerst die Definition einer Sitzung und deren Eigenschaften erläutert. „Eine Sitzung ist eine logische Verbindung zwischen zwei adressierbaren Einheiten im Netz, um Daten auszutauschen“ (Lipinski, 2017). Eine Web-Sitzung wird durch die Generierung einer Sitzungs-ID mit der zusammengehörende Anfragen an den Server identifiziert werden gestartet. Oft passiert dies bei der Anmeldung einer Identität am Server. Aber auch anonyme Web-Sitzungen werden gerne verwendet, um die Aktivitäten von anonymen Nutzern zu unterstützen oder auch zu verfolgen. Klassischerweise wird eine Sitzung mit einem Cookie abgebildet, das vom Server ausgestellt wird und bei jeder HTTP-Anfrage erneut mitgesendet wird. In diesem Cookie befindet sich die Sitzungs-ID, die serverseitig dazu verwendet wird den Applikationszustand zu speichern. Dieses Vorgehen verletzt die REST-Prinzipien laut (Richardson und Ruby, 2007) gleich doppelt. Einerseits werden Daten zur Sitzung auf den Server übertragen und dort gespeichert. Andererseits hat der Client keine Kontrolle über seinen eigenen Applikationszustand. Auch die Benutzererfahrung leidet unter einer serverseitigen Verwaltung des Applikationszustand. Im schlimmsten Fall kann der Benutzer weder mit

mehreren Tabs arbeiten, noch Lesezeichen benutzen oder Links per E-Mail verschicken. Zustandslosigkeit des Servers bedeutet, dass weder in der Datenbank noch am Applikations-Server zu einer Sitzungs-ID Informationen gespeichert werden. Sobald Informationen gespeichert würden, werden serverseitig Ressourcen pro Sitzung verbraucht, was die Skalierungsfähigkeit der Architektur negativ beeinflusst.

In ROCA wird definiert, dass es außer zum Zwecke der Authentifizierung keine Web-Sitzung gegeben darf. Unter §AUTHN wird beschrieben, wie die Authentifizierung in CAMPUSonline bezüglich Zustandslosigkeit implementiert ist. Eine Gefahr für die Zustandslosigkeit ist definitiv der bei der Authentifizierung ausgestellte OAuth2-Zugriffstoken. Dieser könnte als Sitzungs-ID benutzen werden, und somit das REST-Prinzip der Zustandslosigkeit brechen. In der Systemlandschaft von CAMPUSonline EE hätte dies gravierende Folgen. Zur horizontalen Skalierung werden mehrere parallele Applikations-Server eingesetzt, die allerdings nicht miteinander verbunden sind. Jeder Applikations-Server arbeitet eigenständig für sich und weiß nichts von anderen Applikations-Servern. Benutzeranfragen werden je nach Auslastung auf die Applikations-Server verteilt. Ein Applikations-Server A darf also keine Information temporär im Speicher halten, da die nächste HTTP-Anfrage am Applikations-Server B keinen Zugriff auf diese Session Information hätte. Die einzigen Information die zwischen den Applikations-Servern geteilt werden dürfen, sind die Ressourcen, die in der Datenbank gespeichert sind.

Im Zusammenspiel mit CAMPUSonline PL/SQL ist zu bedenken, dass dieser Technologie-Stack nicht zu hundert Prozent zustandslos ist. Zentrale Informationen, wie die angemeldete Person, das ausgewählte Profil und die aktuelle Sprache, wurden an die Session gehängt. Bei der Integration (§INTEGRATION) von CAMPUSonline PL/SQL Seiten, wird dieser Umstand speziell berücksichtigt.

3.6.6. Applikationslogik am Server (§SERVER-LOGIC)

Applikationslogik wird immer zentral am Server implementiert, um unterschiedliche Clients mit dieser versorgen zu können.

abgeleitet von	benötigt	fördert	gefährdet
#APPLICATION-LOGIC	§LAYER	ZARCHITEKTUR ZSICHERHEIT	-

Tabelle 3.9.: Steckbrief von Applikationslogik am Server (§SERVER-LOGIC)

Wie bereits in §LAYER erwähnt, ist die Präsentationsschicht jene, die am schnellsten veraltet. Aus diesem Grund ist es wichtig sie dünn zu halten und möglichst viel Logik auf der Serverseite zu implementieren. Bei einem Wechsel des Client-Frameworks geht die Implementierung der Geschäftslogik nicht verloren, was die Zukunftssicherheit der Architektur stärkt. Ein weiterer Vorteil ist, dass beim Einsatz eines alternativen Clients, die Dateninteграität nicht gefährdet wird, da dieselbe Geschäftslogik durchlaufen werden muss.

Laut Fielding sollte der Client die ausgelieferten Ressourcen möglichst generisch interpretieren bzw. darstellen können. Das HTML-Format in Kombination mit einem Web-Browser stellt dieses REST-Grundprinzip in seiner Perfektion dar. Das ist auch wenig verwunderlich, da Fielding selbst in seiner Dissertation die damals aktuelle Architektur des WWW in einer abstrahierten Form dargestellt hat (Severance, 2015). Die Entwicklung von RESTful APIs auf der Basis von XML bzw. JSON kam erst Jahre später auf. Aus diesem Grunde ist die Entwicklung eines Protokolls (§PROTOCOL) notwendig, welches auf die Anwendungsfälle von CAMPUSonline optimiert ist. Die Entscheidung XML bzw. JSON im Gegensatz zu dynamisch generierten HTML auszuliefern, wird in Abschnitt 3.4.1 argumentiert. Aufgrund des Ziels möglichst viel Applikationslogik am Server zu belassen, wurde auch gegen eine Architektur auf Basis von GraphQL entschieden, wie im Abschnitt 3.4.2 dargestellt wird.

3.6.7. Authentifizierung (§AUTHN)

Es werden aktuelle Standards zur Authentifizierung verwendet. Zusätzliche Authentifizierungsmodule können flexibel integriert werden.

abgeleitet von	benötigt	fördert	gefährdet
#AUTH	-	ZARCHITEKTUR	-
		ZINTERFACE	
		ZSICHERHEIT	

Tabelle 3.10.: Steckbrief von Authentifizierung (§AUTHN)

Die Authentifizierung in CAMPUSonline EE erfolgt auf Basis des OAuth2-Standards. Der OAuth2 Standard wird in RFC6749 (Hardt, 2012) formal definiert und beschrieben, wie Applikationen einen eingeschränkten Zugang für über HTTP angebotene Dienste erhalten. Der Zugriff auf Daten kann dabei durch die Ressourcen Eigentümer selbst oder durch explizite Freigabe von Ressourcen an einen Client erfolgen. OAuth2 trennt strikt zwischen dem sogenannten Autorisierungs-Server und dem Ressourcen-Server. Der Autorisierungs-Server führt die Authentifizierung der Benutzer und die Autorisierung des Clients durch die Benutzer durch. Das Ergebnis des Prozesses der Autorisierung ist ein OAuth2-Zugriffstoken, den der Client erhält. OAuth2 wurde zur Anbindung beliebiger externer Clients zur flexiblen Erweiterung des Funktionsumfangs des bestehenden Systems konzipiert. Benutzer können durch Bestätigung der Autorisierungsanfrage selektiv und transparent entscheiden, ob sie Daten für einen externen Client freigeben wollen, um dessen dessen Service zu nutzen. Dadurch, dass OAuth2 für beliebige externe Clients konzipiert wurde, können die Benutzer durch Bestätigung der Autorisierungsanfrage selektiv entscheiden, ob sie Daten für einen externen Client freigeben. Die Methode zur Authentifizierung wird im OAuth2-Standard selbst nicht definiert. Auf Basis von OAuth2 können aber externe *Identity-Provider* (z.B. Shibboleth, Google, Facebook, ...) eingebunden werden. Externe Accountverwaltungssysteme und zusätzliche instanzspezifische Authentifizierungs-Mechanismen können durch den standardisierten auf Java Authentication and Authorization Service (JAAS) basierenden Loginmodul-Mechanismus eingebunden werden (Microsystem, o.D.). Zusätzlich zu den Login-Modulen kann auch die Abbildung der authentifizierten Identität hin zu einer gültigen CAMPUSonline-Identität instanzspezifisch konfiguriert werden. Somit können auch externe Identity-Management-Systeme einfach eingeklinkt werden. Standardmäßig wird

CAMPUSonline EE mit einer lokalen Anmeldung mittels Benutzername und Passwort und der Möglichkeit einer Anmeldung über Shibboleth ausgeliefert. Shibboleth weist im Bereich von Bildungseinrichtungen eine sehr weite Verbreitung auf.

Die Authentifizierung und Autorisierung erfolgt zustandslos über den HTTP Authorization Header.

Der Ressourcen-Server beherbergt die eigentlichen Ressourcen des Systems. Der OAuth2-Zugriffstoken wird bei der Anmeldung an CAMPUSonline EE an den Browser des Benutzers übertragen. Dort wird er im sogenannten lokalen Speicher des Browsers (local storage) gespeichert, welcher über mehrere Tabs hinweg per JavaScript zugänglich ist. Der OAuth2-Zugriffstoken wird von nun an bei jeder Ajax Anfrage an den Ressourcen-Server im HTTP-Authorization-Header als *Bearer Token* mitgeschickt. Serverseitig gibt es zwei Möglichkeiten aus diesem OAuth2-Zugriffstoken die Identitäts-Information abzuleiten.

1. Bei jeder HTTP-Anfrage an den Applikations-Server, wird der Autorisierungs-Server befragt, welche Identität sich hinter dem konkreten OAuth2-Zugriffstoken verbirgt.
2. Der OAuth2-Zugriffstoken selbst beinhaltet die Information zur Identität in verschlüsselter Form. Eine Umsetzung wird im RFC 7519 JSON Web Token (Bradley, Sakimura und Jones, 2015) beschrieben.

Während die erste Variante als sehr sicher gilt, da der OAuth2-Zugriffstoken nur eine zufällige Sequenz von Zahlen ist, hat sie den Nachteil dass sie weniger performant ist. Die zweite Variante hat den Vorteil, dass weder der Applikations-Server noch der Autorisierungs-Server den OAuth2-Zugriffstoken speichern müssen, da jegliche Information in den OAuth2-Zugriffstoken kodiert wurde, kann aber wenn die Verschlüsselung geknackt wurde, dazu missbraucht werden, sich als beliebige Identität auszugeben. Das heißt Variante zwei ist tatsächlich zustandslos, während bei Variante eins der Autorisierungs-Server sehr wohl die Verbindung zwischen OAuth2-Zugriffstoken und Identität speichern muss. Allerdings ist ja dies genau die Aufgabe des Autorisierungs-Servers. Dieser soll Informationen zum Anmeldevorgang speichern. Oft sind solche Daten sogar notwendig, da man Transparenz über die aktuell angemeldete Anzahl an Benutzer haben will.

Allerdings wird kein Zustand der Applikation im Authorization-Server hinterlegt. Der Anmeldevorgang, in Form eines ausgestellten OAuth2-Zugriffstokens, ist eine Ressource wie jede andere und beherbergt rein nur jene Informationen, die der Benutzer durch den Anmeldevorgang bzw. Autorisierungsvorgang erzeugt hat. In CAMPUSonline wurde aus Sicherheitsgründen die erste Variante gewählt. Zudem kann der Zugriff auf den Authorization-Server im Falle von CAMPUSonline über einen direkten Datenbankzugriff erfolgen was den Nachteil bezüglich Geschwindigkeit reduziert.

3.6.8. Cookies (§COOKIE)

abgeleitet von	benötigt	fördert	gefährdet
#COOKIES	§AUTHN	ZINTEGRATION ZSICHERHEIT	-

Tabelle 3.11.: Steckbrief von Cookies (§COOKIE)

Mittels eines CAMPUSonline systemweiten Cookies wird die Web-Sitzung des bestehenden Altsystem integriert ohne dabei die Zustandslosigkeit des Servers zu gefährden.

Wie unter §NO-STATE beschrieben verletzt die Verwendung von Cookies den Grundsatz der Zustandslosigkeit. CAMPUSonline EE kann grundsätzlich auch ohne Cookies betrieben werden. Zur Integration von CAMPUSonline PL/SQL, welches selbst ein Cookie verwendet, um den Benutzer zu authentifizieren, wurde die OAuth2-Anmeldung an CAMPUSonline EE so gestaltet, dass zusätzlich zur Ausstellung des OAuth2-Zugriffstokens auch ein klassisches CAMPUSonline PL/SQL Cookie ausgestellt wird, wenn der zugreifende Client über die Berechtigung verfügt ein solches Cookie auszustellen.

Ein Security Cookie erhöht die Sicherheit des Systems.

Die Verwendung eines Cookies bietet in puncto Sicherheit diverse Vorzüge, die man optional durch Konfiguration in CAMPUSonline EE nutzen kann.

Dabei wird das Cookie in CAMPUSonline EE allerdings niemals dazu verwendet, Informationen zur aktuellen Web-Sitzung serverseitig zu speichern. Während die Verwendung eines OAuth2-Zugriffstokens die Gefahr von Cross-Site-Request-Forgery-Attacken (XSRF) beträchtlich mindert, kann mittels eines „HttpOnly Cookies“ (Barth, 2017) verhindert werden, dass ein potentieller Angreifer per JavaScript das Cookie auslesen bzw. manipulieren kann. Der OAuth2-Zugriffstokens kann im Gegensatz dazu per Definition ohne weiteres mittels JavaScript ausgelesen werden. Serverseitig wird - bei entsprechender Konfiguration - immer die Identität des OAuth2-Zugriffstokens mit der Identität, die sich aus der Sitzungs-ID des Cookie ergibt, verglichen. Dadurch ergibt sich ein zusätzlicher Schutz gegen Cross-Site-Scripting (XSS). Ein weiterer positiver Effekt bei der Verwendung eines Cookies ist, dass dieses gelöscht wird sobald der Browser geschlossen wird. Wohingegen der OAuth2 Zugriffs Token im lokalen Speicher des Browsers bestehen bleibt. Dadurch werden Benutzer automatisch vom System abgemeldet wenn sie den Browser schließen, auch wenn sie dies nicht explizit mittels des Abmeldebuttons getan haben. Gerade bei geteilten Arbeitsstationen, wie sie oft in Lernzentren vorkommen, ist dieses Verhalten besonders wichtig.

3.6.9. Kommunikation mit dem Altsystem (§LEGACY)

abgeleitet von	benötigt	fördert	gefährdet
RSCHICHTEN	§PROTOCOL	ZINTEGRATION	-
CLAPP	§LAYER		

Tabelle 3.12.: Steckbrief von Kommunikation mit dem Altsystem (§LEGACY)

Integrierte PL/SQL Geschäftslogik kann zu einem späteren Zeitpunkt einfach ausgetauscht werden.

Fielding beschreibt wie dank einer Schichtenarchitektur Altsysteme mittels REST einfach eingebunden werden können. Geschäftslogik von CAMPUSonline PL/SQL kann in CAMPUSonline EE dank der im Projekt umgesetzten Dataservice-Technologie erfolgreich aufgerufen bzw. ausgeführt werden.

Dataservices sind eine auf PL/SQL basierende Technologie, die es auf einfache Art ermöglicht bestehende Geschäftslogik über RESTful-Webservices lose gekoppelt über das HTTP-Protokoll zugänglich zu machen. RESTful-Webservices auf Basis von PL/SQL sind aufgrund des systemweit einheitlichen Schnittstellenprotokolls coData nicht von jenen RESTful-Webservices die mittel CAMPUSonline EE Technologie entwickelt wurden, zu unterscheiden. Aus diesem Grund kann eine auf PL/SQL basierende Schnittstelle einfach, durch Austausch der Implementierung, zu einem späteren Zeitpunkt auf den neuen Technologie-Stack gehoben werden.

3.6.10. Einheitliches Datenmodell (§DATA-MODEL)

abgeleitet von	benötigt	fördert	gefährdet
CL_{DB}	§MODUL	$Z_{ARCHITEKTUR}$ $Z_{INTEGRATION}$	-

Tabelle 3.13.: Steckbrief von Einheitliches Datenmodell (§DATA-MODEL)

CAMPUSonline PL/SQL basiert auf einem einzigen, in sich konsistenten, der dritten Normalform entsprechenden und mittels Datenbank-Constraints abgesicherten relationalen Datenbankmodell.

Diese Eigenschaft wird oft als integriertes CMS bezeichnet, da es redundante Daten in unterschiedlichen Systemen, die untereinander abgeglichen werden müssen verhindert. Der aktuelle Microservice Trend, der darauf abzielt viele atomare Services anzubieten, die eine getrennte Datenhaltung aufweisen, wird zugunsten der Konsistenz und der Einfachheit im Bereich des SLCs nicht verfolgt.

Das bestehende Datenmodell von CAMPUSonline PL/SQL wird gekapselt und in bereinigter Form auch in der neuen Softwarearchitektur genutzt.

Im Zuge der Migration auf CAMPUSonline EE soll das bestehende Datenmodell von CAMPUSonline PL/SQL grundlegend bereinigt und modularisiert werden. Die verwendeten Terminologie wird vereinheitlicht und in die englische Sprache übersetzt. Dazu wird schrittweise ein idealisiertes

Datenmodell entwickelt, welches entsprechend der Bereiche von CAMPUSonline EE modularisiert in den neuen Bereichs-Datenbankschemen abgelegt wird. Noch nicht migrierte Tabellen, werden mittels Datenbank-Views gekapselt und dabei bereits die neue einheitliche, englische Terminologie verwendet. Die Datenbank-Views werden dabei im Sinne von CL_{DB} als Datenbank-Rückwärts-Gateways ausgeführt. Unzulänglichkeiten im bestehenden Datenmodell - wie zum Beispiel eine fehlende Internationalisierung - werden beseitigt. In den Kapsel-Views wird keine PL/SQL-Geschäftslogik aufgerufen. Nur einfache standardisierte SQL-Funktionalität wird genutzt, um das bestehende Datenmodell auf das neue Datenmodell zu adaptieren. Das Ziel ist es, dass Kapsel-Views schlussendlich durch echte Tabellen ersetzt werden und die Quelldaten in diese Tabelle überführt werden. Der bestehende Quelltext in PL/SQL soll zu diesem Zeitpunkt entweder keine Referenzen auf die alten Tabellen mehr besitzen, oder über entsprechende Schnittstellen bereits im Sinne von Datenbank-Vorwärts-Gateways auf das neue Datenmodell umgestellt werden. Ein automatisierter Abgleich mittels Triggern zwischen altem und neuem Datenmodell hat sich aufgrund der Komplexität und einer reduzierten Zuverlässigkeit nicht als zielführend erwiesen.

3.6.11. Zugriffskontrolle ($\S_{\text{AUTHZ}}$)

abgeleitet von	benötigt	fördert	gefährdet
SEC_{RBAC}	$\S_{MODUL}$	$Z_{ARCHITEKTUR}$	-
SEC_{ABAC}	$\S_{AUTHN}$	$Z_{SICHERHEIT}$	

Tabelle 3.14.: Steckbrief von Zugriffskontrolle ($\S_{AUTHZ}$)

CAMPUSonline EE setzt auf die Zugriffskontroll-Paradigmen Role-Based Access Control (SEC_{RBAC}) und Attribute-Based Access Control (SEC_{ABAC}) und verbindet somit die Vorteile beider Paradigmen.

Die Zugriffskontrolle ist eines der zentralsten Elemente eines CMS. Deshalb wird dieses Thema im Kapitel 5 vertiefend abgehandelt.

3.6.12. Caching (§CACHE)

abgeleitet von	benötigt	fördert	gefährdet
RCACHING	§RESOURCES	ZARCHITEKTUR	-
	§NO-STATE	ZUSABILITY	

Tabelle 3.15.: Steckbrief von Caching (§CACHE)

*Alle statischen Ressourcen werden mittels **URL-Fingerprinting** gecacht. Alle anonym zugänglichen Daten werden mittels **E-Tag Caching** ausgeliefert.*

Mittels Caching kann die Performance aus der Sicht des Anwenders extrem gesteigert werden. Durch einen Cache kann verhindert werden, dass unnötige Anfragen an den Server gestellt werden, wenn sich die zu erwartenden Daten sich nicht geändert haben. Der Nachteil ist allerdings, dass Daten die sich doch geändert haben veraltet sind. Außerdem können Caches ein Sicherheitsproblem oder Datenschutzproblem darstellen, wenn sicherheitsrelevante oder personenbezogene Daten durch eine andere Partei ausgelesen werden können.

Roy Fielding bezieht sich in seiner Arbeit hauptsächlich auf den HTTP-Caching-Mechanismus. Dabei werden harte Caches (HTTP-Header-Attribute *Expires* und *Cache-Control*) von weichen Caches (HTTP Header Attribute *Last-Modified* und *ETag*) differenziert (Grigorik, 2017). Bei weichen Caches behält der Server die Kontrolle darüber, wann eine Ressource neu ausgeliefert wird. Mit der negativen Konsequenz, dass bei jeder Ressource trotz Cache eine Anfrage an den Server geschickt wird, die mit dem HTTP-Status „304 not modified“ beantwortet werden kann. Erhält der Browser diese Antwort so bezieht er die Ressource aus dem Cache. Eine weitere Variante, die allerdings komplexer in der Umsetzung, vor allem in einer SPA Umgebung ist, ist die Methode des URL-Fingerprinting. Dabei behält der Server trotz der Verwendung eines harten Caches, die Kontrolle wann eine Ressource tatsächlich ausgeliefert wird. Dazu wird hinter jeder, in einem HTML-Dokument eingebundenen URL, ein Hash-Wert mit einem **?** getrennt angehängt. Dieser Hash-Wert ändert sich jedes mal, wenn die Ressource

geändert wurde. Dadurch wird der Browser gezwungen die Ressource trotz hartem Cache abzuholen, wenn sie sich geändert hat.

Im Fall von CAMPUSonline kann nicht vorhergesagt werden, wann sich eine Ressource ändert. Das gilt sowohl für die statischen Ressourcen, da das Deployment durch den Kooperationspartner selbstbestimmt umgesetzt wird, als auch für die Daten, da diese durch die Benutzer jederzeit geändert werden können. Zusätzlich sind fast alle Daten so konfigurierbar, dass sie nur authentifiziert zugänglich sind. Das heißt Caching kann aus Sicherheitsgründen für diese Daten nicht angewandt werden.

3.6.13. Logik ohne Browser nutzbar (§NON-BROWSER)

abgeleitet von	benötigt	fördert	gefährdet
#NON-BROWSER	§LAYER	ZARCHITEKTUR	-
#LINK	§PROTOCOL	ZINTERFACE	
	§AUTHN	ZQUALITÄT	
	§AUTHZ		

Tabelle 3.16.: Steckbrief von Logik ohne Browser nutzbar (§NON-BROWSER)

Jegliche serverseitige Geschäftslogik soll auch ohne Browser aufrufbar und nutzbar sein. Dabei soll transparent sein, welches System zugreift und welche Berechtigungen, dieses externes System besitzt.

Durch die Öffnung jeglicher Funktionalität in CAMPUSonline für andere Systeme, über eine homogene Schnittstellentechnologie ist die Integration in eine bestehende Systemlandschaft einfacher zu realisieren. Außerdem kann die Qualität durch automatisierte REST-Tests, die nicht über den Umweg eines Browsers erfolgen müssen, wesentlich erhöht werden.

Dafür muss die Architektur allerdings wesentliche Grundvoraussetzungen im Bereich des Schichtenmodells, des Schnittstellenprotokolls, der Authentifizierung und der Autorisierung mit sich bringen. In CAMPUSonline werden aus diesem Grund sowohl reale Personen als auch Systeme als Identitäten

unterstützt. Dazu ist es notwendig einen entsprechenden Authentifizierungsmechanismus für Systeme ohne ein Benutzerkonto, welches mit einer realen Person verbunden ist, zu unterstützen. Da System-Identitäten in CAMPUSonline EE als OAuth2-Clients umgesetzt werden, kann dies durch den OAuth2 *Client Credentials Grant* sehr gut abgebildet werden. Durch die Zuordnung von *OAuth2-Scopes* kann dem Client nun die Erlaubnis erteilt werden auf bestimmte RESTful APIs von CAMPUSonline EE zuzugreifen. Die Absicherung auf OAuth2-Scope-Ebene ist aber nur eine grobkörnige Zugriffserlaubnis. Für die meisten Funktionalitäten werden Rollen bzw. Rollen in einem speziellen Kontext benötigt. Daher ist es notwendig den OAuth2-Clients, ebenso wie reale Identitäten-Rollen, in einem Kontext zuzuordnen.

3.7. Clientseitige Grundsätze

3.7.1. Single-Page-Application (§SPA)

abgeleitet von	benötigt	fördert	gefährdet
RCLIENT-SERVER	§PROTOCOL	ZINTERFACE	§RESOURCES
IAFENSTER		ZUSABILITY	ZUSABILITY
#STATIC-ASSETS		ZSICHERHEIT	§BROWSER
			§HATEOAS

Tabelle 3.17.: Steckbrief von Single-Page-Application (§SPA)

Es gibt nur eine einzige HTML-Seite die vom Benutzer aufgerufen wird. Jegliche Änderung in der Anzeige wird durch die Manipulation des Document-Object-Models (DOM) durchgeführt.

Eine SPA arbeitet mit rein statischen HTML, JavaScript und CSS. Die restliche Kommunikation beschränkt sich auf das Austauschen von Daten mittels des coData-Protokolls mit dem Server, welche durch die SPA interpretiert

und angezeigt werden. Beim initialen Seitenaufruf der einzigen HTML-Ressource wird der gesamte JavaScript-Code des aktuellen Moduls auf einmal heruntergeladen. Durch die starke Trennung der Geschäftslogik von der Benutzeroberfläche und dem Einsatz des coData-Protokolls wird der Aufbau von präsentationsneutralen und einheitlichen Schnittstellen forciert.

Die Verwendung einer SPA bringt den großen Vorteil, dass sich das System in puncto Geschwindigkeit für die Benutzer wie eine lokale Applikation anfühlt. Nach dem initialen Laden der SPA, muss nur mehr ein sehr limitierter Datenaustausch mit dem Server erfolgen, da viele Daten im Speicher des Browsers gehalten werden können. Es müssen keine dynamisch erzeugten HTML-Strukturen bzw. JavaScript-Codes mehr vom Server geladen werden. Im Altsystem wurden die erzeugten Datenmengen für komplexe Seiten mehrere Megabyte groß und mussten immer wieder neu vom Server generiert werden. Dadurch wird die Geschwindigkeit der Benutzeroberfläche deutlich verbessert. Der Server, und im Falle von CAMPUSonline PL/SQL die Datenbank, werden durch diesen Umstand zusätzlich entsprechend entlastet.

Nachdem der Server in einer ROA komplett zustandslos ist, muss der gesamte Applikationszustand im Client verwaltet werden. Eine SPA unterstützt dieses Vorhaben immens. Bereits einmal eingegebene Daten eines Benutzers, gehen durch die Verwaltung des Applikationszustand in der SPA nicht mehr verloren. In CAMPUSonline PL/SQL wurden um dieses Verhalten zu gewährleisten neue Fenster geöffnet, damit der aktuelle Zustand der Seite erhalten bleibt. Aus der Informationsarchitektur ($IA_{FENSTER}$) geht allerdings hervor, dass nur mehr ein einziges Fenster zur Anzeige der Benutzeroberfläche verwendet werden soll. Wird nun zu einer Sicht navigiert, die der Benutzer clientseitig bereits verändert hat, kann das anzuzeigende Modell aus dem Applikationszustand der SPA geladen werden und der Benutzer findet, die Sicht so vor, wie er sie zuletzt verlassen hat. Gerade die Abbildung von „Master Detail Beziehungen“ kann so in einer SPA sehr elegant umgesetzt werden, ohne dass Daten serverseitig gespeichert werden müssen.

Ein weiterer positiver Punkt bei der Verwendung einer aktuellen SPA-Technologie - wie Angular im Falle von CAMPUSonline EE - ist das breite

Entwickler-Ökosystem welches rund um diese Technologien entsteht. Abgesehen von einem sehr ausgereiftem Framework selbst, über die gute Dokumentation, Erweiterungen, hervorragende Unterstützung durch die Entwicklungsumgebung gibt es auch eine sehr breite und hilfreiche Open-Source-Community, die bei vielen Problemstellungen unterstützen kann.

Allerdings müssen einige wichtige Entscheidungen getroffen und Spielregeln eingehalten werden, um ein benutzerfreundliches und intuitives Erlebnis zu bieten. Als erstes muss entschieden werden, ob sich die Single-Page-Application, wie eine Desktop Anwendung oder wie eine klassische ressourcenorientierte Web-Anwendung für den Benutzer anfühlen soll. Für CAMPUSonline wurde entschieden, dass sich die Benutzeroberfläche weiterhin seitenorientiert wie eine klassische Webseite verhalten soll. Dadurch können bestehende Seiten von CAMPUSonline PL/SQL auch leichter in das neue CAMPUSonline EE integriert werden. Hier wird auch der große Nachteil einer SPA ersichtlich. Viele Browserfunktionen, die bei klassischen Webseiten *out of the box* funktionieren, müssen in einer SPA erst implementiert oder durch den Architekturstil vorgegeben werden.

3.7.2. Standard Browser Verhalten (§BROWSER)

abgeleitet von	benötigt	fördert	gefährdet
#BROWSER-CONTROLS #LINK	-	ZUSABILITY	-

Tabelle 3.18.: Steckbrief von Standard Browser Verhalten (§BROWSER)

Die klassischen Browserfunktionen verhalten sich erwartungsgemäß.

Soll sich die Anwendung wie eine klassische Webanwendung verhalten muss die SPA auch diverse Browser Steuerungsmöglichkeiten unterstützen.

Die Wichtigsten davon sind:

- **Zurück-Button**: Navigation mit dem Zurück-Button bzw. auch Vorwärts-Button ist besonders wichtig bei der Navigation am Smartphone.

- **Neuladen**: Eine dargestellte Seite kann jederzeit neu geladen werden, ohne den Kontext der aktuell angezeigten Ressource zu verlieren.
- **Lesezeichen**: Jeder wichtige Applikationszustand kann gespeichert werden.
- **Neue Tabs**: Jede Navigationsmöglichkeit kann durch die Benutzer bestimmbar in einem neuen Tab oder in einem neuem Fenster geöffnet werden.
- **Transportierbare Links**: Ein Link soll einfach kopiert und verschickt bzw. mit anderen geteilt werden können.
- **Scrollposition**: Die aktuelle Scrollposition wird bei der Navigation zu einer anderen Seite gespeichert. Navigiert der Benutzer zur Seite mittels Zurück-Button, springt der Browser exakt an die Position, die beim Verlassen der Seite gespeichert wurde.
- **Ladebalken**: Der Browser zeigt, ob die Seite gerade Inhalte vom Server bezieht.

Das impliziert, dass die SPA auch entsprechend der ROA-Grundsätze konzeptioniert und umgesetzt wird. In der SPA sollen weiterhin untereinander verlinkte Seitenkomponenten angeboten werden, obwohl nur eine statische HTML-Seiten-Ressource existiert, die geladen wird. Im *Single Page Interface Manifesto* hält Santamaria (2015) fest, dass es zwei unterschiedliche Zustandsarten in einer SPA gibt:

1. **Grundzustände**: Sind Zustände, die sich in der aktuellen URL manifestieren.
2. **Sekundärzustände**: Sind Zustände, die nicht wichtig genug sind, dass sie in der URL widergespiegelt werden.

Jegliche Seitenkomponente muss ein solcher Grundzustand sein und eine eigene URL erhalten. Auch jegliche Parametrisierung die für eine angezeigte Seitenkomponente wertvoll genug ist, muss in der URL als Abfrage-Parameter abgebildet werden.

In einer SPA muss die aktuelle URL des Browsers geändert werden, ohne tatsächlich ein Neuladen einer Seite auszulösen. Neben der Variante nur das URL-Fragment zu manipulieren, also die Zeichenkette hinter dem Raute-Symbol, gibt es mittlerweile die Möglichkeit die HTML5 History API zur Manipulation der URL zu verwenden (Jenkov, 2015). Ein mit der

HTML5 History API erstellter URL-Eintrag ist von einer klassischen URL nicht mehr unterscheidbar. Angular unterstützt mit seinem Router beide oben genannten Modi. In CAMPUSonline EE wurde die Variante mit URL-Fragment gewählt, um klar zwischen klassischen URL-Aufrufen und JavaScript-Routing unterscheiden zu können. Während durch diese Abbildung des Grundzustandes der Applikation in der URL die meisten Browser Steuerungsmöglichkeiten wie gewohnt genutzt werden können, muss das Speichern der Scrollposition per Hand entsprechend implementiert werden. Da die Inhalte der Seite asynchron geladen und angezeigt werden, und die Seitengröße initial nicht bestimmt werden kann, ist diese Aufgabe nicht gerade trivial. Dahingegen ist die Anzeige, ob Inhalte gerade vom Server geladen werden, sehr leicht mit Bordmittel von Angular umzusetzen.

3.7.3. Links steuern das Client-Verhalten (§HATEOAS)

abgeleitet von	benötigt	fördert	gefährdet
R BESCHREIBEND	§ PROTOCOL	Z ARCHITEKTUR	
R HATEOAS	§ SERVER-LOGIC		

Tabelle 3.19.: Steckbrief von Links steuern das Client-Verhalten (§HATEOAS)

Das coData-Protokoll liefert neben den eigentlichen Ressourcen-Repräsentationen in XML bzw. JSON zusätzlich noch Links mit aus, welche den Prozess beschreiben.

Fielding (2008b) schrieb in seinem die REST-API-Welt prägenden Blogeintrag: „REST APIs must be hypertext-driven“. Zusätzlich ist laut dem Reifegradmodell von Leonard Richardson (Webber, Parastatidis und Robinson, 2010) Hypermedia as the Engine of Application State (HATEOAS) die äußerst wichtige aber auch gleichzeitig höchste Ausbaustufe einer ROA. Fielding hat in seiner Dissertation den Begriff HATEOAS zwar eingeführt, ihn aber nicht erklärt. Um zu verstehen, was HATEOAS tatsächlich bedeutet, wird der Begriff in die einzelnen Bestandteile zerlegt.

- **Hypermedia**: Fielding (2008a) bezieht sich dabei auf die Definition von Nelson (1974) und meint damit nicht linearen Text, welcher Steue-

rungselemente beinhaltet, durch die Leser selbstbestimmt Aktionen auslösen können.
- **as the engine**: Mit der Maschine meint Fielding (2013) einen endlichen Automaten. Die einzelnen Zustände des endlichen Automaten spiegeln die möglichen Zustände der Applikation wider.
- **of application state**: Die Zustände sind mittels Links miteinander verbunden. Ausgehend von einem konkreten Zustand stellen sie alle möglichen nächsten Zustände der Applikation dar.

Clients lösen, mithilfe der Steuerungselemente von Hypermedia, Aktionen aus, bewegen sich dabei von Ressource zu Ressource und ändern damit den Zustand der Client-Applikation. Der große Vorteil dabei ist, dass die Prozesslogik serverseitig implementiert wird und daher zukunftssicher und zentral änderbar ist. Die Links sind direkt und kontextabhängig in die Repräsentation der Ressource integriert. Der aktuelle Kontext definiert, ob ein Link ausgeliefert wird und damit für den Client verfügbar ist. Zum Beispiel hängt es vom aktuellen Status der Ressource, der Systemeinstellung und von der angemeldeten Person ab, ob ein Link eingebettet wird oder nicht. Obwohl man anstrebt, diese über die Lebensdauer einer API nicht zu verändern, ist dies aufgrund der Dynamik des Webs nicht immer gewährleistet. Eine URL kann sich beispielsweise beim Umzug der REST-API auf eine Domain ändern, oder einfach nur aus dem Grund, dass sich die API weiterentwickelt hat. Deshalb propagiert Hypermedia unter anderem die Verwendung von Links in Ressourcen. Clients sollen URLs nicht selbst konstruieren, sondern diese auf Basis eines Links dynamisch auslesen.

Ein Link definiert exakt welche Aktion auf die Ressource ausgeübt werden soll, wenn der Link aktiviert wird.

In HTML sind Steuerungselemente in unterschiedlichster Form ausgeprägt, wie etwa mit Links, Formularen oder auch eingebetteten Ressourcen. In HTML sind Aktionen auf Ressourcen durch die Steuerungselemente eindeutig definiert. Zum Beispiel ist klar festgelegt, dass ein klassischer Link immer mit einer HTTP-GET-Anfrage oder ein Formular mit einer HTTP-POST-Anfrage verbunden ist. Der Zusammenhang zwischen Steuerungselementen und Aktionen ist Hingegen in einigen verbreiteten Media-Types für RESTful-Webservices nur wage oder eingeschränkt definiert. Darüber hinaus werden oft nicht alle über HTTP möglichen Aktionen unterstützt.

Um die Hypermediafähigkeit eines Media-Types zu bewerten hat Amundsen (2011) den sogenannten H-Faktor eingeführt. Der coData-Link wurde mithilfe der Kriterien des H-Faktors entwickelt und erfüllt alle neun H-Faktoren, die von Amundsen (2011) beschrieben wurden. Um den Client über die Semantik eines Links zu informieren, wird das coData-Attribut rel verwendet. Aufgrund der bekannten Bedeutung kann der Client automatisiert darüber entscheiden, wie er den Link verwenden bzw. darstellen will. In coData wird ein genormtes Subset an IANA Relationen (Mark Nottingham, Reschke und Algermissen (2017)) verwendet. Wie in HAL ist rel das primäre Mittel einen Link in einer Liste von Links zu identifizieren. Im Gegensatz zum HAL-Format wird die Semantik einer Relation gleichzeitig mit einem definierten Verhalten des Clients verbunden. Die Mächtigkeit der Verbindung von Semantik mit einem vorgeschriebenen Verhalten für den Client kann auch im HTML-Format (rel=stylesheet) oder im Atom-Protokoll (rel=edit) beobachtet werden.

Die SPA verwendet klassische HTML Links zur Änderung des Applikationszustandes, welche von den über REST ausgelieferten Links abgeleitet werden.

Die Links werden zuerst serverseitig generiert (§SERVER-LOGIC) und in Angular mittels generischer Komponenten interpretiert. Diese extrahieren übertragene Links aus der Datenressource, um sie als klassisches HTML für den Benutzer zugänglich zu machen. Dabei selektieren sie den Link mittels dessen Relation und gegebenenfalls Namen und ID. Aufgrund der Adressierbarkeit (§RESOURCES) von Ressourcen ist es essentiell, dass beim Auslösen eines Links dieser vom Client als aktueller Grundzustand der Applikation interpretiert wird. Durch diese Regel kann sichergestellt werden, dass Benutzer Links jederzeit in einem eigenen Tab oder Fenster öffnen können. Durch die Abbildung der URL wie unter §BROWSER beschrieben, kann dies erfolgreich mit einer SPA umgesetzt werden. Viele Entwickler lassen sich dazu verleiten, Event-Binding direkt auf klickbare Elemente zu vergeben. Durch diese semantisch falsche Verwendung von HTML leidet zusätzlich die Accessibility. Links werden von diversen Screenreadern automatisch erkannt und können jederzeit aufgerufen werden. Werden allerdings Event-Binding Mechanismen verwendet, um eine Navigation auszulösen, so erkennt dies der Screenreader nicht.

*Die Seitenkomponenten der **SPA!** sind alternative Ressourcen Repräsentationen*

der auslieferbaren Datenressourcen.

Die vom Server ausgelieferten Links beziehen sich im Regelfall auf die Ressourcen der REST-Schnittstelle. Nur Links für das Altsystem oder für externe Systeme sind von dieser Regel ausgenommen. Die Seitenkomponenten von Angular können als eine weitere alternative Repräsentation von serverseitigen Ressourcen gesehen werden. Die URLs von Server und Client sind bis auf die Basis-URL identisch und können so generisch und einfach transformiert werden.

SPA URL	REST URL
`<base>/ui/#/courses`	`<base>/rest/courses`
`<base>/ui/#/courses/1234`	`<base>/rest/course/1234`
`<base>/ui/#/courses?top=20`	`<base>/rest/courses?top=20`

Tabelle 3.20.: URLs der SPA können einfach durch Austauschen der Basis URL in REST URLs transformiert werden. Somit müssen über die REST Schnittstelle nur REST bezogene URLs ausgeliefert werden.

Wie in der Tabelle 3.7.3 beschrieben, besteht die Transformation darin die Basis URL `<base>/ui/#/` gegen die Basis URL `<base>/rest/` auszutauschen bzw. umgekehrt. Sowohl der Name der Ressource (courses) als auch Pfadparameter (/1234), als auch Abfrageparameter (?top=20) werden in die Route der SPA als Grundzustand kodiert. Hält man sich an die Regel, dass jegliche Information, die per GET-Anfrage zum Server geschickt werden soll, als Grundzustand in die Route kodiert wird, so ist die Übertragung des aktuellen Zustandes des Clients an den Server trivial, durch das Austauschen der Basis-URL, durchführbar. Dieses Vorgehen hat mehrere Vorteile. Einerseits zwingt man Entwickler so dazu, den wichtigsten Applikationszustand in die SPA Route zu kodieren, sodass die klassischen Browser Steuerungsmöglichkeiten unterstützt werden. Andererseits hilft es den Entwicklern den Überblick zu behalten, wie Front-End und Back-End zusammen hängen und fördert ein gesamtheitliches Verständnis. Nicht von CAMPUSonline ausgelieferte Clients haben so den Vorteil, dass sie sich unabhängig von den in CAMPUSonline verwendeten URLs, durch den kompletten Prozess von CAMPUSonline auf Basis eines wohldefinierten

Protokolls bewegen können, ohne dabei HTML parsen und interpretieren zu müssen.

3.7.4. Wahl des Kontextes (§CONTEXT)

abgeleitet von	benötigt	fördert	gefährdet
IA$_{\text{KONTEXT}}$	§HATEOAS §PROTOCOL	Z$_{\text{USABILITY}}$	-

Tabelle 3.21.: Steckbrief von Wahl des Kontextes (§CONTEXT)

Benutzer können den aktuellen Kontext jederzeit einsehen, ändern, verlassen, speichern und kommunizieren.

In der IA von CAMPUSonline EE wird beschrieben, dass der Kontext jene Informationen beinhalten, welche vom System benötigt werden um mit Benutzern sinnvoll zu interagieren. Der Kontext wird in CAMPUSonline EE im Sinne eines Grundzustandes der SPA als eine Reihe von URL-Abfrage-Parametern umgesetzt. Die genormten Abfrage-Parameter für den Kontext werden von der SPA ausgelesen und in eine bearbeitbare Kontextinformation im Seitenkopf umgeformt. Benutzer können den Kontext über den Seitenkopf einsehen, ändern und löschen. Dadurch, dass die Information in die URL kodiert wird, kann der aktuelle Kontext auch in einem Lesezeichen oder in einem Favoriten-Menü am Desktop gespeichert werden. Eine Beispiel URL für den Semester Kontext wäre `<ui-base>/#/courses?objTermId=WS18`.

Benötigt eine Seite einen speziellen Kontext, muss diese Kontextinformation von den Benutzern abgefragt werden.

Sowohl Entwickler als auch Administratoren müssen dazu in der Lage sein, die Eingabe einer Kontextinformation durch Benutzer vor dem Betreten einer Seite zu erzwingen. Entwickler können dies durch Metadaten in der Programmierung der Seite bewerkstelligen. Administratoren können dies beim Anlegen eines Zugangs durch sogenannte Template-Parameter ebenso erreichen. Dabei muss die SPA, bevor die Seite dargestellt wird, überprüfen,

ob Template-Parameter in der URL oder in den Metadaten der Seite existieren, die zuvor durch die Anzeige von Kontext-Auswahldialogen aufgelöst werden müssen. Dies kann durch die Verwendung eines zentralen *Angular Guards* realisiert werden. (Precht, 2016) Ein Beispiel für eine URL mit Template-Parametern wäre `<ui-base>/#/courses?objTermId={termId}` Aufgrund des Wertes des Template-Parameters *termId* kann die SPA einen für diesen Wert registrierten Kontextauswahldialog anzeigen.

Ist ein Kontext bereits bekannt so müssen Benutzer diesen nicht erneut bekannt geben.

Haben Benutzer bereits eine Kontextauswahl getroffen, dann darf innerhalb desselben Menüs, nicht nochmals die gleiche Kontextinformation abgefragt werden. Dieses Verhalten kann durch eine Normierung der Namen der Kontext-Parameter bewerkstelligt werden. Dadurch kann bei der Ausgabe des Menüs erkannt werden, dass der Wert für *objTermId* bereits bekannt ist und dieser für die Links im Menü bereits vorausgefüllt werden. Befindet man sich zum Beispiel aktuell auf der Seite `<ui-base>/#/courses?objTermId=WS18` und wird nun über das Menü der Zugang mit der URL `<ui-base>/#/exams?objTermId={termId}` gewählt, so soll kein Kontextauswahldialog erscheinen. Stattdessen soll die Seite `<ui-base>/#/exams?objTermId=WS18` direkt geöffnet werden.

Benutzern dürfen nur sinnvolle Kontext-Auswahlmöglichkeiten angeboten bekommen.

Wird eine Kontext-Auswahlmöglichkeit angeboten, so dürfen aus Gründen der Usability nur Werte angeboten werden, welche für die aktuelle Seite und den agierenden Benutzer sinnvoll sind. Wird zum Beispiel ein Organisationsauswahldialog angezeigt, so sollen nur jene Organisationen gelistet sein, an denen der Benutzer auch tatsächlich eine Rolle besitzt, welche sie berechtigt die Seite aufzurufen. Seiten werden immer über Zugänge betreten und der gewählte Zugang selbst wird als Kontextinformation im Parameter *accId* in der URL hinterlegt. `<ui-base>/#/exams?$ctx=accId=123&$objTermId=WS18` Dadurch ist es möglich den Zugang bzw. die dahinterliegende Applikation zu befragen, welche Organisationen im aktuellen Kontext angezeigt werden sollen.

3.7.5. Trennung von Menüs und Seiten (§MENU)

abgeleitet von	benötigt	fördert	gefährdet
IA$_{MENU}$	§HATEOAS	Z$_{ARCHITEKTUR}$	Z$_{USABILITY}$
IA$_{AKTIONEN}$	§CONTEXT	Z$_{INTERFACE}$	-
		Z$_{INTEGRATION}$	-

Tabelle 3.22.: Steckbrief von Trennung von Menüs und Seiten (§$_{MENU}$)

Jede Seite kann durch Konfiguration in jedes beliebige Menü eingehängt werden.

Dieser Grundsatz wird direkt aus den Forderungen der geplanten Vorgangsweise für Menüs aus der Informationsarchitektur abgeleitet. Um die Forderung der losen Kopplung zwischen Menüs und Seiten umzusetzen, muss es möglich sein das anzuzeigende Menü aus dem aktuellen Kontext zu ermitteln. In CAMPUSonline EE kann man diese Information indirekt aus dem aktuellen Zugangskontext ableiten. Dazu ist es notwendig, dass beim Aufruf eines Zugangs dieser in den Systemkontext übernommen wird, damit der aktuell gewählte Zugang über jede weitere Navigation hinweg erhalten bleibt. Das zu einem Zugang gehörende Menü mit seinen Einträgen kann mittels eines RESTful-Webservice vom Server bezogen werden. Die so abgeholten Zugänge müssen beim Aufbau des Menüs noch mit der aktuellen Kontextinformation angereichert werden. Das heißt alle Template-Werte, die in den Zugängen vorkommen, müssen gegen potentiell bekannte Werte ausgetauscht werden (siehe §CONTEXT).

Durch die flexible Konfiguration eines Menüs mittels Zugängen können Kooperationspartner bestehende Menüs entsprechend ihrer Bedürfnisse anpassen und auch lokal implementierte Seiten in Menüs einklinken. Sie können auch komplett neue Menüs flexibel anlegen, und somit ähnlich eines Baukastensystems neue Funktionalitäten bzw. Geschäftsprozesse abbilden, ohne eine Implementierung durchführen zu müssen. Dabei muss allerdings bedacht werden, dass solche lokale Ausprägungen nicht durch CAMPUSonline dokumentiert oder unterstützt werden können. Des Weiteren wird die Möglichkeit geschaffen, Seiten von anderen Frameworks nahtlos in die Menüführung einzuhängen. Zum Beispiel wird in §INTEGRATION beschrieben,

wie CAMPUSonline PL/SQL Seiten in das neue Produktdesign eingeklinkt werden können. Ein negativer Punkt bei der Integration von anderen Frameworks ist, dass der gesamte JavaScript-Code der SPA erneut evaluiert und ausgeführt werden muss, wenn zwischen unterschiedlichen Frameworks gewechselt wird. Dies schlägt sich in einer erhöhten initialen Ladezeit nieder. Je nach Modularisierung der Implementierung und der eingebundenen Frameworks kann dies zu einer unterschiedlich langen Dauer führen.

3.7.6. Trennung von Daten und Darstellung (§STYLE)

abgeleitet von	benötigt	fördert	gefährdet
RBESCHREIBEND	§PROTOCOL	ZARCHITEKTUR	-
#KNOW-STRUCTURE		ZUSABILITY	
		ZQUALITÄT	

Tabelle 3.23.: Steckbrief von Trennung von Daten und Darstellung (§STYLE)

Daten sind von ihrer Darstellung strikt getrennt. Eine generische Visualisierung mittels Formatvorlagen fördert die Usability der Oberfläche und die Qualität der Implementierung.

Dies gilt einerseits für die erzeugten semantischen HTML-Strukturen, welche mittels CSS ansprechend gestaltet werden. Andererseits gilt dies auch für über RESTful-Webservices abrufbare Datenressourcen. Diese können mittels Formatvorlagen und einer generischen Visualisierung dargestellt werden, ohne die einzelnen Attribute einer Ressource zu kennen. Ein Browser interpretiert auch nur den Mime Type HTML, kennt aber nicht den semantischen Inhalt des Dokumentes. Zur visuellen Darstellung von HTML wird CSS verwendet. Das gleiche Konzept einer Formatvorlage wird in CAMPUSonline EE auch für RESTful-Webservices eingesetzt. Datenressourcen, die keine Information bezüglich Darstellung beinhalten, können mittels der Einbettung eines oder auch mehrerer Stylesheet-Links, um Darstellungsressourcen ergänzt werden. Das entspricht auch dem Gedanken der Einbettung von Mikroformaten, welcher von Vasilakis (2017) in seinem

Architekturstil *Introspected REST* vorgestellt wurde. Die Darstellungsressource beinhaltet dabei Metadaten, sowohl für die gesamte Datenressource als auch mittels Attributselektoren für die einzelnen Attribute der Datenressource. Die Darstellungsressource wird in Angular als Konfiguration für eine generische Komponente verwendet. Allerdings kann die Darstellungsressource genauso gut zum Aufbau einer Extensible Stylesheet Language (XSL) Transformation genutzt werden. Dadurch wäre es möglich beliebige andere Ausgabeformate zu erzeugen. Zum Beispiel könnte auf dieser Basis ein HTML-Dokument serverseitig erzeugt und auf den Einsatz einer SPA bei Bedarf verzichtet werden. Immer wiederkehrende Anwendungsfälle werden so mit minimalen Quelltext in der Präsentationsschicht schnell und konsistent umgesetzt. Klassische Anwendungsfälle sind in CAMPUSonline zum Beispiel durchsuchbare und filterbare Tabellen, Exportmöglichkeiten, Eingabemasken und Auswahllisten. Durch den Aufbau der Oberfläche auf Basis von exakt definierten Darstellungsressourcen, wird die Erwartungskonformität (Norm, o.D., S. 10) und somit die Usability für Endbenutzer gesteigert. Außerdem werden durch die Zentralisierung der Implementierung die Qualität und die Dokumentierbarkeit gefördert. Da die Darstellungsressourcen serverseitig generiert werden, kann auch auf die Berechtigungen der Benutzer Rücksicht genommen werden. Zusätzlich können individuelle Anpassungen (Norm, o.D., S. 15) der Komponenten, sowohl für den Einzelne als auch systemweit, zur Laufzeit mittels Konfiguration erfolgen. Dazu können die Darstellungsressourcen in einem Metadaten-Repository serverseitig verwaltet werden, wenn der Bedarf der Individualisierbarkeit besteht.

Die Benutzeroberfläche wird automatisiert oder teilautomatisiert auf Basis bereits vorhandener Metadaten aufgebaut.

Kennard und Leaney (2010) beschreiben, wie eine Benutzeroberfläche auf Basis von bereits im Backend vorhandenen Information, teilautomatisiert und wesentlich schneller erzeugt werden kann, um redundanten Quelltext im Frontend zu verhindern. Sie zeigen wie aus unterschiedlichen Modellen im Backend, zum Beispiel das Ressourcen-Modell oder das Datenmodell, Informationen zur Generierung der Benutzeroberfläche gewonnen werden können. Durch die Architektur von CAMPUSonline EE und die strikte Trennung von Daten- und Darstellungsressourcen, kann diese Vorgehensweise gut in die Architektur integriert werden.

3.7.7. Zugänglichkeit (§ACCESSIBILITY)

abgeleitet von	benötigt	fördert	gefährdet
#ACCESSIBILITY	-	ZUSABILITY	-

Tabelle 3.24.: Steckbrief von Zugänglichkeit (§ACCESSIBILITY)

Menschen mit Beeinträchtigungen sollen das System barrierefrei nutzen können.

Gerade für Campus-Management-Systeme ist diese Forderung unumgänglich, da neben den moralischen Grundgedanken der Offenheit von Bildungseinrichtungen sich Österreich sowie Deutschland dazu auf Ebene der Europäischen Union verpflichtet haben, die Leitlinien der Web Accessibility Initiative (WAI) umzusetzen. Diese wurde auch im § 1 Abs. 3 E-Government-Gesetz (E-GovG) verankert und ist seit 1. Jänner 2008 bindend. In Anbetracht der Einführung einer neuen Benutzeroberflächen-Technologie, muss die Barrierefreiheit einen entsprechend großen Stellenwert in der Evaluierung und in der Einführung erhalten. Während früher noch die Verwendung von JavaScript und Ajax für Screenreader eine große Herausforderung darstellten, haben sich die Screenreader selbst und die zur Verfügung stehende Standards mittlerweile weiterentwickelt. Für SPAs können die gleichen Praktiken wie für klassische Webseiten verwendet werden. Eine wichtige Grundvoraussetzung für diese Forderung ist die Verwendung von semantisch richtigem Markup. Mit HTML5 sind einige semantische Elemente, wie *menu* oder *header*, ergänzt worden, welche es erlauben zusätzliche semantische Auszeichnungen zu verwenden. Zusätzlich können noch WAI-ARIA-Attribute verwendet werden, um weitere semantische Auszeichnungen für Screenreader anzubieten. Zusätzlich soll entsprechend geprüft werden, ob sämtliche Aktionen auch ohne Maus rein mit einer Tastatur erfolgen können. Visuelle Informationen sollen stets auch in textueller Form zur Verfügung stehen. Ein entsprechendes Kontrastverhältnis ist wesentlich für Menschen mit einer eingeschränkten Farbwahrnehmung.

abgeleitet von	benötigt	fördert	gefährdet
R_ON-DEMAND	§MODUL	Z_USABILITY	-
	§SPA	Z_INTERFACE	-

Tabelle 3.25.: Steckbrief von Lazy-Loading (§LAZY-LOAD)

3.7.8. Lazy-Loading (§LAZY-LOAD)

Es werden nur jene Module der SPA vom Server geladen, die tatsächlich benötigt werden.

Dadurch kann die initiale Ladezeit stark reduziert werden. Während *Code on Demand* sich bei Fielding auf die Einbettung von nachladbaren Quelltext (z.B. JavaScript Datei) in einem Mime Type bezog, wird das dynamische Nachladen von JavaScript in einer SPA über den *Router* abgewickelt. Angular unterstützt die Verwaltung von Modulen und das dynamische Nachladen dieser, wenn eine bestimmte Basis Route aufgerufen wird (Motto, 2017).

3.7.9. Typisierter Client (§CLIENT-TYPES)

abgeleitet von	benötigt	fördert	gefährdet
-	-	Z_QUALITÄT	§STYLE

Tabelle 3.26.: Steckbrief von Typisierter Client (§CLIENT-TYPES)

Der Client wird mit einer typisierten und objektorientierten Programmiersprache umgesetzt, sodass Fehler die aufgrund einer falschen Typisierung auftreten, bereits zum Zeitpunkt des Kompilierens entdeckt werden.

Der Client von CAMPUSonline wurde mit der typisierten Programmiersprache Typescript umgesetzt. Typescript ist eine Obermenge von JavaScript, und erzeugt durch Transpilieren des Quelltextes ebenfalls JavaScript, welches schlussendlich im Browser ausgeführt wird (Bierman, Abadi und Torgersen, 2014). Durch die Typisierung können Fehler frühzeitig erkannt

werden und Überarbeitung des Quelltexts auch im großen Stil ohne Gefahr durchgeführt werden. Diese Möglichkeit ist gerade bei großen Projekten sehr vorteilhaft, bei denen viele Entwickler zusammenarbeiten.

Auch die Verbindung zwischen Client und Server wurde typisiert. Passt die Implementierung des internen Clients nicht zur Schnittstelle des Servers so tritt bereits zum Zeitpunkt des Transpilierens ein Fehler auf. Diese Regel kann deshalb eingeführt werden, da CAMPUSonline die gemeinsame Auslieferung des Clients und des Servers selbst kontrolliert. Wie in §STYLE ausgeführt, soll der Client bevorzugt mit allgemeinen Visualisierungen arbeiten. Gerade in der initialen Phase eine Projektes existieren die Komponenten zur generischen Visualisierung jedoch noch nicht. Außerdem ist die Entwicklung von generischen Komponenten aufwendig, und wird bei Anwendungsfällen die nur selten auftreten aus wirtschaftlichen Gründen auch nicht durchgeführt. Ein weiterer Grund warum CAMPUSonline EE auf eine interne typisierte Schnittstelle setzt, ist die zunehmende Verbreitung von GraphQL. Gerade die starke Typisierung von GraphQL ist einer der Hauptgründe warum GraphQL sich als würdige Alternative zu REST etabliert. Die Entscheidung, ob mit einer typisierten Schnittstelle gearbeitet wird oder nicht, wird somit in Richtung Client-Entwicklung verschoben.

3.7.10. Integration des Altsystems (§INTEGRATION)

abgeleitet von	benötigt	fördert	gefährdet
CL_{IS}	§SPA	$Z_{INTEGRATION}$	$Z_{USABILITY}$
$IA_{AKTIONEN}$	§MENU	$Z_{USABILITY}$	$IA_{FENSTER}$

Tabelle 3.27.: Steckbrief von Integration des Altsystems (§INTEGRATION)

Bestehende Seiten von CAMPUSonline PL/SQL können ins neue Produktdesign eingeklinkt werden, um so sämtliche bestehende Funktionalitäten weiterhin und in einer homogenen Benutzeroberfläche entsprechend dem neuen Navigationskonzeptes nutzen zu können. Zusätzlich können bestehende Seiten vom CAMPUSonline-Desktop aus verlinkt werden und im neuen sowie im alten Design geöffnet werden.

Nach mehreren Versuchen der Integration auf der Ebene eines Informationssystem-Gateways (CL_{IS}) stellte sich die Verwendung einer einzigen generischen PL/SQL-Route in Angular als praxistaugliche Lösung heraus. Die Integration auf der Ebene der Benutzeroberfläche erfolgt mittels eines Iframes, welcher als eigene generische Seite in Angular ausgeprägt wurde. Dabei wird jede PL/SQL-URL durch eine Angular Route getunnelt. (`<base>/ee/ui/ca2/app/desktop/#/pl/ui/$ctx/**`). Der flexible Teil der Route (**) wird als PL/SQL URL interpretiert und an den Iframe weitergeleitet. Bei der Navigation im Iframe wird die aktuelle Iframe-URL entsprechend in die Browser-URL reflektiert. Dadurch wird eine klassische Steuerung mittels Browser ermöglicht, welcher häufig als Grund genannt wird keine Iframes zu verwenden. Sowohl der Zurück-Button, der Vorwärts-Button als auch Lesezeichen können so verwendet werden. ($\S_{BROWSER}$) Außerdem können klassische Links zur Steuerung des Applikationszustandes benutzt werden, um direkt auf eine integrierte PL/SQL-Seite zu verlinken ($\S_{HATEOAS}$). Auch die Auswahl des Kontextes inklusive des vorgeschalteten Login-Dialogs bei Seiten, die eine Benutzeranmeldung erfordern, funktioniert, da es sich aus der Sicht von Angular um eine klassische Angular-Seitenkomponente handelt. Bestehender Titel, Untertitel und Menü können automatisch in die neue Seitenkomponente übernommen werden. Um den Grundsatz der Trennung von Aktionen und navigatorischen Links gerecht zu werden, wurde eine einfache Konfigurationsmöglichkeit geschaffen, mit welcher der Entwickler Links die Aktionen ausführen auszeichnen kann, welche nicht ins Menü übernommen werden sollen. Stattdessen werden sie direkt in der Seite als Aktionsmöglichkeiten integriert, so wie es im Navigationskonzept vorgesehen wurde.

Die Größe des Iframes erfolgt nicht mittels JavaScript sondern durch native Bordmittel des Browsers.

Die erste Lösung setzte auf einen Iframe, der relativ positioniert wurde, und sich mittels JavaScript dynamisch an die Größe des Inhalts der integrierten Seite anpasste. Diese Lösung hat sich als zu instabil erwiesen, da Sonderfälle berücksichtigt werden mussten und es zu Seiteneffekten kam. Die tatsächliche Lösung setzt auf die Möglichkeit von CSS3 eine *Flexbox* zu verwenden, anstatt manuell die Größe des Iframes selbst zu berechnen. Dadurch ist es möglich einen Iframe im Seitenfluss zwischen Kopf- und Fußzeile einzubinden und diesem immer automatisch die maximal mögliche

Höhe zuzuordnen. Ohne JavaScript, ausschließlich über CSS. Die Konsequenz ist, dass der Scrollbalken nicht über die ganze Seite geht, sondern auf den Iframe Inhalt begrenzt ist.

Für integrierte Sichten werden statt Popup-Fenster neue Browser-Tabs geöffnet, um den Applikationszustand der integrierten Sicht zu erhalten.

In der ersten Lösung wurde versucht, das Öffnen und Schließen von Fenstern und modalen Fenstern (Overlays) zu überschreiben, so dass sie im aktuellen Iframe geöffnet werden anstatt ein neues Fenster zu öffnen. Während dies dem Konzept eines einzigen Fensters ($IA_{FENSTER}$) entgegen kam, wurde in der Praxis festgestellt, dass dies als Standardverhalten nicht optimal sei, da viele PL/SQL-Sichten ihren Applikationszustand nicht in die URL reflektierten. Als Beispiel blieben diverse ausgeklappte Elemente einer Seite bei Neuladen des Browsers nicht erhalten. Dieses Problem wurde in CAMPUSonline PL/SQL typischerweise mit dem Öffnen eines neuen Fensters bewerkstelligt. Wird aber nun in Angular die aktuelle URL überschrieben, geht dieser Applikationszustand verloren. Aus diesem Grund wurde das Öffnen einen Popup-Fensters (JavaScript Methode window.open) mit dem Öffnen eines neuen Tabs überschrieben. Analog zum Überschreiben der `window.open` Methode wird auch die `window.close` Methode überschrieben.

Der aktuelle Applikationszustand von CAMPUSonline EE übersteuert den Applikationszustand von CAMPUSonline PL/SQL.

CAMPUSonline PL/SQL verwaltet die aktuelle Benutzersprache und das gewählte Benutzerprofil in einer Web-Session, die in der Datenbank persistent gespeichert wird. Außerdem muss der Kontext, welcher in Angular gewählt wurde, an CAMPUSonline PL/SQL übertragen werden. Dadurch, dass CAMPUSonline PL/SQL nur relative URLs zur Navigation verwendet, kann ein Matrix-Parameter (Berners-Lee, 1996) verwendet werden, um Informationen an CAMPUSonline PL/SQL zu übergeben, welche dank des Matrix Parameters über die Navigation hinweg erhalten bleiben. Dadurch kann der Kontext von CAMPUSonline PL/SQL mit dem von CAMPUSonline EE synchron gehalten werden.

Der DOM-Baum der integrierten Seite wird überwacht, um auf dynamische Änderungen per JavaScript in der integrierten Seite reagieren zu können.

Um auf dynamische Änderungen mittels JavaScript einer PL/SQL-Seite in Angular reagieren zu können, kann der DOM-Baum der PL/SQL-Seite mit der HTML5 API überwacht werden, und diese Ereignisse entsprechend an das Angular-Framework weiter zu delegieren. (Scholz und Bode, 2015) Dadurch kann mit minimalem Berechnungsaufwand die Anzeige des Menüs und Seitentitels aktuell gehalten werden.

4. Modulare Systemarchitektur

Starke (2014, S. 81) stellt in seinem Buch einen Leitfaden vor, wie eine Systemarchitektur entwickelt und beschrieben werden kann. Er verwendet dabei vier architektonische Sichten um das System zu abstrahieren:

- Die **Kontextabgrenzung** beschreibt, wie das System in seinem Kontext eingebettet ist, welche Stakeholder das System nutzen und welche externen Systeme mit dem eigenen System kommunizieren.
- Die **Bausteinsicht** zerlegt das System in Module und beschreibt die Abhängigkeiten der Module untereinander.
- Die **Laufzeitsicht** stellt das Laufzeitverhalten des Systems dar und beschreibt die Kommunikation der einzelnen Module untereinander.
- Die **Verteilungssicht** beschreibt die Infrastruktur, die notwendig ist um das System zu betreiben.

Während alle vier Sichten für die unterschiedlichen Stakeholder bei der Planung und beim Aufbau des Systems wichtig sind, kann aufgrund des Umfangs in diesem Kapitel nur auf die Bausteinsicht des Systems eingegangen werden. Die Kontextabgrenzung entspricht der in der Einleitung beschriebenen Abbildung 1.1. Die zwei anderen Sichten wurden entsprechend CAMPUSonline intern beschrieben.

Bausteinsichten zeigen die statischen Strukturen der Architekturbausteine des Systems, Subsysteme, Komponenten und deren Schnittstellen. (Starke, 2014, S. 81) Die Bausteine (Einheiten) des Systems werden identifiziert und in unterschiedlichen Vergrößerungsstufen übersichtlich dargestellt. Es werden Abhängigkeiten der Bausteine untereinander aber auch zu externen Systemen definiert. Ziel ist es, die Kohäsion in den einzelnen Bausteinen zu maximieren, gleichzeitig aber auch die Kopplung der Bausteine untereinander zu minimieren. Aus einer einfachen Erhebung der Quantität und

Handhabbarkeit der entstehenden Softwarekomponenten wurden sinnvolle Strukturen, Gruppierungen und Bausteine abgeleitet.

CAMPUSonline EE besteht aus einer typisierten Hierarchie von Software-Bausteinen, wobei jeder Baustein-Typ unterschiedliche architektonische Eigenschaften besitzt (Abbildung 4.1). Ein Software-Baustein kann je nach Typ wiederum untergeordnete Software-Bausteine eines speziellen Typs besitzen. Somit sind nicht nur die Software-Bausteine, sondern auch die Baustein-Typen hierarchisch organisiert. Durch diese Gliederung entsteht ein azyklischer hierarchischer Graph von Abhängigkeiten. Ein großer Vorteil der hierarchischen Struktur ist die Möglichkeit, einen Bereich oder eine Applikation unabhängig weiterentwickeln oder diese komplett durch eine andere Implementierung zu ersetzen. Das Gesamtsystem wird dabei nicht beeinflusst. Die praktische Umsetzung der Modularisierung erfolgt mittels Apache Maven (Lalou, 2013).

Der Begriff des Bausteins weist in CAMPUSonline EE folgende Eigenschaften auf:

- Bausteine haben immer einen bestimmten Typ und können untergeordnete Bausteine beinhalten.
- Für jeden Baustein-Typ ist festgelegt, welche Typen als untergeordnete Bausteine eingehängt werden können.
- Jeder Baustein-Typ entspricht einer eindeutigen Vergrößerungsstufe bzw. Betrachtungsebene in der Bausteinsicht.
- Der oberste Baustein ist CAMPUSonline vom Typ *System*.

.

4.1. Systemebene

Der oberste Baustein-Typ der Hierarchie heißt System. Ein System im Sinne von Geschäftsprozessen einer Bildungseinrichtung umfasst voneinander abhängige Geschäftsprozesse, um Gesamtaufgaben im Kontext einer Bildungseinrichtung zu erfüllen. Der Umfang eines Systems wird durch die

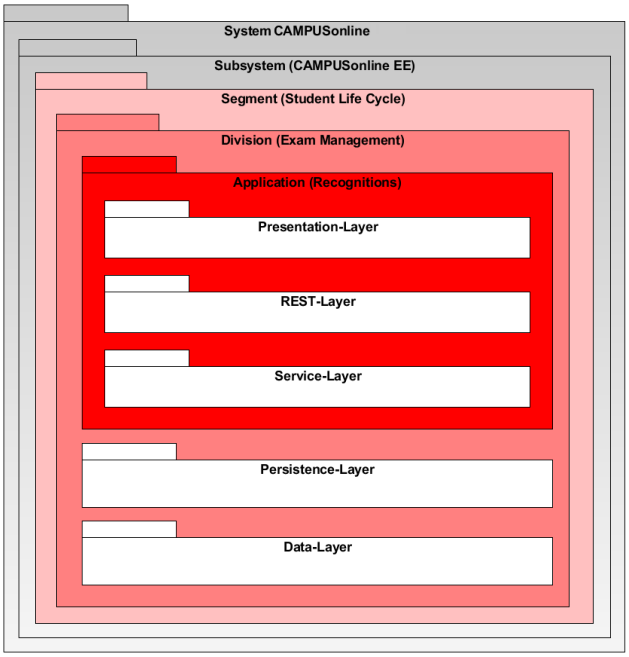

Abbildung 4.1.: Die unterschiedlichen Modultypen von CAMPUSonline EE mit entsprechenden beispielhaften Modulnamen.

zentralen Segmente bestimmt, die das System abbilden sollen. In CAMPUSonline EE ist das zentrale Segment die Verwaltung der Studien der Studierenden. Die Verwaltung wird im sogenannten Student Lifecycle (SLC) abgebildet. Neben den zentralen Geschäftsprozessen sind die sogenannten unterstützenden Geschäftsprozesse, entweder direkt im System implementiert, oder sie sind über System-Schnittstellen durch die Kopplung mit anderen Systemen realisiert. Wichtig ist, dass die Gesamtaufgabe zur Abwicklung konkreter Geschäftsfälle erfolgreich und vollständig erfüllt werden kann. Die Systemgrenzen von CAMPUSonline EE erstrecken sich ausgehend von der Bewerbung und Einschreibung über die Organisation der Lehre und der Prüfungen bis hin zum Abschluss des Studiums von Studierenden.

CAMPUSonline integriert zwei Subsysteme mit komplett unterschiedlichen Technologie-Stacks. Das Subsystem CAMPUSonline PL/SQL basiert auf der Programmiersprache PL/SQL und auf Oracle-Datenbank-Technologien (siehe Abschnitt 3.2). Dieses Subsystem soll im Zuge einer sukzessiven Komplettmigration, von dem auf Java EE und Angular basierenden Subsystem CAMPUSonline EE wie in Abschnitt 3.3 beschrieben, abgelöst werden. Beide Subsysteme sind transparent miteinander verbunden und erscheinen für den Benutzer als ein Gesamtsystem.

Auf Systemebene (Abbildung 4.2) erkennt man, welche benachbarten Systeme mit CAMPUSonline kommunizieren. Aufgrund der großen Anzahl von angeschlossenen Systemen, die noch dazu je nach Instanzkonfiguration optional eingesetzt werden können, sind in der folgenden Grafik nur ausgewählte exemplarische Systeme abgebildet. Prinzipiell wird unterschieden zwischen den in CAMPUSonline EE integrierten Systemen und externen Systemen, die mittels System-Adapter angeschlossen sind. Der Unterschied zwischen diesen zwei Systemklassen ist, dass integrierte Systeme für den Regelbetrieb von CAMPUSonline verpflichtend benötigt werden, während externe Systeme rein optional eingesetzt werden können. Bei der Anbindung von externen Systemen wird die Schnittstelle durch CAMPUSonline vorgegeben und muss mittels System-Adapter für die jeweilige Ausprägung des externen Systems angepasst werden.

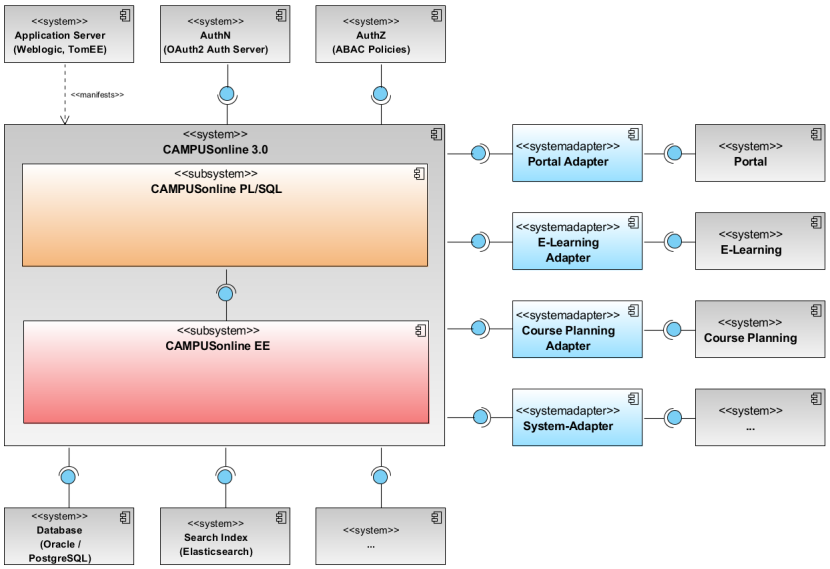

Abbildung 4.2.: Auf Systemebene erkennt man die zwei Subsysteme von CAMPUSonline und mittels System-Adapter angeschlossene Systeme.

4.1.1. Integrierte Systeme

Integrierte Systeme sind essenziell für den Betrieb von CAMPUSonline. Daher werden Richtlinien und Empfehlungen zur Integration in die Systemlandschaft der Kooperationspartner und zur Konfiguration dieser Systeme mitgeliefert. Fallweise können auch alternative Systeme durch den Kooperationspartner betrieben werden, sofern diese die definierten Schnittstellen erfüllen. Das wichtigste in CAMPUSonline EE integrierte System ist die Datenbank. Aber auch Systeme zur Dokumentenerstellung, zum Versand von E-Mails, zur Erstellung von Drucksorten und ein Suchindex sind zum sinnvollen Betrieb von CAMPUSonline EE notwendig.

4.1.2. Externe Systeme

Beliebige optionale externe Systeme können an CAMPUSonline angeschlossen werden. Externe Systeme sind mannigfaltig und verfügen über diverse

und meist voneinander abweichende Schnittstellendefinitionen. Beispiele für externe Systeme sind Lernplattformen (z.B. Moodle) oder das Portal einer Universität als öffentliche Internetpräsenz. Aber auch Systeme zur Evaluierung von Veranstaltungen oder zur Optimierung von Stunden- und Belegungsplänen sollen an CAMPUSonline EE angeschlossen werden können. Um den unterschiedlichsten Schnittstellen externer Systeme dennoch gerecht zu werden, kommen System-Adapter zum Einsatz, welche zwischen den beiden Schnittstellenwelten übersetzen.

4.2. Segmentebene

Wie beschrieben ist das zentrale Segment (engl. segment) von CAMPUSonline der Student Life Cycle (SLC). Zusätzlich gibt es noch das Segment Basis und Ressourcen Management (BRM) in CAMPUSonline, in der die Grundlagen, die für den SLC notwendigerweise gemanagt werden müssen, zur Verfügung gestellt werden. In diesem Segment werden unter anderem Organisationen, Identitäten, Rechte und Räume verwaltet, Objekte die allesamt zwingend im SLC benötigt werden. Das Segment BRM bietet die grundlegenden Möglichkeiten um beliebige weitere Segmente unabhängig vom SLC darauf aufbauen zu können. Zum Beispiel kann ein Segment für Forschung eingeklinkt werden, das bis jetzt in CAMPUSonline abgebildet wurde, allerdings in Zukunft nicht mehr Teil des Portfolios von CAMPUSonline EE sein wird. Stattdessen wird die Verwaltung der Forschung in einem externen System abgebildet und mittels eines Systemadapters angeschlossen.

Segmente kommunizieren über definierte Schnittstellen mittels RESTful-Webservices entsprechend dem coData-Protokoll mit externen Systemen. Eines dieser an CAMPUSonline EE angeschlossenen Systeme ist CAMPUSonline PL/SQL. Auch die Kommunikation über diese Schnittstelle erfolgt lose gekoppelt über das coData-Protokoll. Auf Ebene der Segmente können allgemeine Funktionalitäten und Entitäten segmentübergreifend angeboten werden und somit bei der Applikationsentwicklung genutzt werden. Jedes Bereichs-Modul ist eindeutig einem Segment zugeordnet und darf auch nur die Bibliothek dieses Segments inkludieren. Allerdings können die Bibliotheken eines Segments auf die Bibliotheken eines untergeordneten Segments

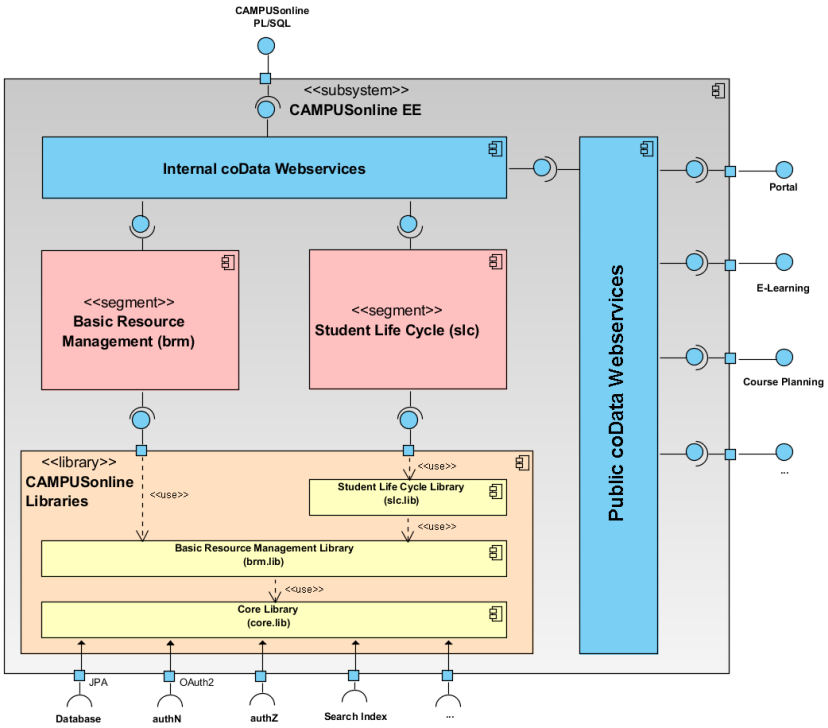
CAMPUSonline
PL/SQL
<<subsystem>>
CAMPUSonline EE
Internal coData Webservices
<>
Basic Resource
Management (brm)
<>
Student Life Cycle (slc)
Public coData Webservices
Portal
E-Learning
Course Planning
...
<<library>>
CAMPUSonline
Libraries
<<use>>
Student Life Cycle Library
(slc.lib)
Basic Resource Management Library
(brm.lib)
Core Library
(core.lib)
JPA
OAuth2
Database
authN
authZ
Search Index
...

Abbildung 4.3.: Die zwei Hauptsegmente von CAMPUSonline sind das Basis Ressourcen Management und der Student Life Cycle.

zugreifen. Somit ist es im SLC möglich, die Bibliotheken des BRM und des Kernsegment (CORE) zu nutzen. Dadurch können gemeinsame Logiken, entsprechend dem DRY- und KISS-Prinzip redundanzfrei, wiederverwendet werden. Gerade die Möglichkeit, zentrale Entitäten auf Segment-Ebene lesend zu teilen, erlaubt es weiterhin sehr performante Datenbankabfragen zu formulieren, ohne auf die langsamere lose Kopplung über RESTful-Webservices zurückgreifen zu müssen. Die fachlichen Segmente SLC und BRM sollen allerdings möglichst wenig an Funktionalität und Entitäten beinhalten, um einem monolithischen Softwaredesign entgegenzuwirken.

Zusätzlich zu den zwei fachlichen Segmenten gibt es noch ein Kernsegment (CORE), das die rein technischen Frameworks von CAMPUSonline beherbergt. Eine wichtige Aufgabe des CORE ist der Anschluss der in CAMPUSonline EE integrierten Systeme. Dadurch können diese von CAMPUSonline EE aus abstrahiert angesprochen werden. Die Datenbank wird zum Beispiel mit der JPA und mit Apache DeltaSpike abstrahiert, sodass in Zukunft auch PostgreSQL als Datenbank angeschlossen werden kann. Aber auch für die Authentifizierung und die Autorisierung sind entsprechende Abstraktionen im CORE vorhanden, um diese gegen andere Systeme ersetzen zu können. Der CORE kann im Gegensatz zu den schlanken fachlichen Segmentbibliotheken sehr viele Funktionalitäten und Frameworks beinhalten, um die technologischen Grundlagen für alle Bausteine zur Verfügung zu stellen.

4.3. Bereichsebene

Die Ebene direkt unter der Segmentebene beinhaltet die Bereiche (engl. divisions) von CAMPUSonline EE und wird in Abbildung 4.4 schematisch dargestellt. Dazu gehören unter anderem *Lehre*, *Prüfungen*, *Personen* und *Organisationen*. Bei der Migration eines Altsystems können Bereich aufgrund von bestehenden Strukturen, Datenmodellen und auch durch Experteninterviews abgeleitet werden. Die Ebene eines Bereichs entspricht der CAMPUSonline-internen Teamstruktur, die sich wiederum mit der klassischen Struktur der Fachabteilungen von Universitäten deckt. So natürlich

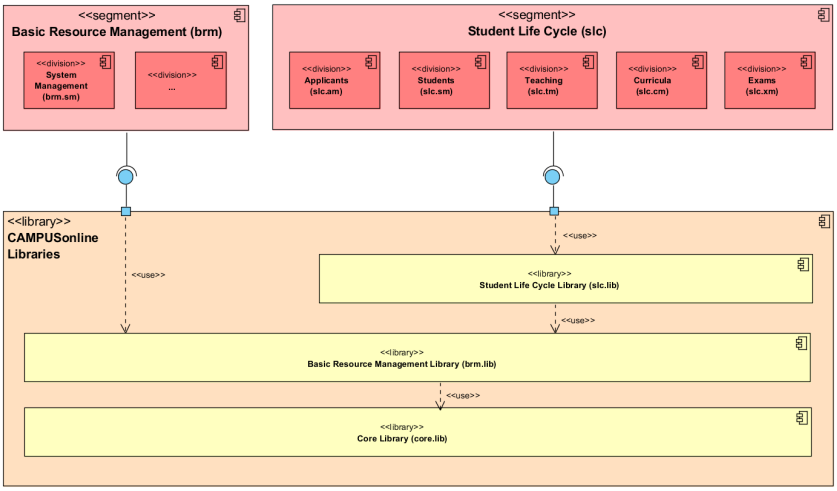
<>
Basic Resource Management (brm)
<<division>>
System Management (brm.sm)
<<division>>
...
<>
Student Life Cycle (slc)
<<division>>
Applicants (slc.am)
<<division>>
Students (slc.sm)
<<division>>
Teaching (slc.tm)
<<division>>
Curricula (slc.cm)
<<division>>
Exams (slc.xm)
<<library>>
CAMPUSonline Libraries
<<use>>
<<use>>
<<library>>
Student Life Cycle Library (slc.lib)
<<use>>
<<library>>
Basic Resource Management Library (brm.lib)
<<use>>
<<library>>
Core Library (core.lib)

Abbildung 4.4.: Bereiche sind in einer für ein Entwicklungsteam verwaltbaren Größenordnung. Sie können unabhängig voneinander ausgeliefert und betrieben werden.

wie die Gliederung in Fachbereiche ergeben sich entsprechend *Conways Law* auch die Grenzen des Baustein-Typs *Bereich*.

„Organisationen, die Systeme modellieren, [...] sind auf Modelle festgelegt, welche die Kommunikationsstrukturen dieser Organisationen abbilden." (Conway, 1968) Aufgrund der bereits etablierten Teamstrukturen können mithilfe von Conways Law sinnvolle Bereichsgrenzen aus der bestehenden Modularisierung in CAMPUSonline PL/SQL abgeleitet werden. Für das Segment des SLC wären dies folgende Bereiche:

- **am**: Bewerbungs- und Zulassungs-Management (Application and admission management)
- **sm**: Studierenden- und Studien-Management (Student and study management)
- **tm**: Lehre (Teaching management)
- **cm**: Curricula Management
- **xm**: Prüfungen (Exams)

Auf Basis der Kommunikationskultur können auch fehlende Bereiche identifiziert werden. Beispielsweise konnte durch Interviewtechnik bei beste-

henden Teamstrukturen festgestellt werden, dass das Thema der Curricula bisher in den zwei Bereichen Lehre und Prüfungen abgewickelt wurde. Aufgrund der Aufteilung dieses Themas auf zwei Teams gab es immer wieder erhöhten Kommunikationsbedarf und Missverständnisse bei konkreten Umsetzungen. Auch die strategische Betrachtung ist auf Bereichsebene am sinnvollsten, weil auf dieser Ebene am effektivsten die strategische Ausrichtung des Systems definiert werden kann. Zum Beispiel kann entschieden werden, welche Bereiche besser in einem externen System abgebildet werden, um sich auf die übrigen Bereiche konzentrieren zu können. Zusätzlich ist eine Priorisierung der Bereiche im Rahmen einer Migration sehr sinnvoll, damit Kräfte gebündelt werden können.

Für die Entwicklung, für die Datensicherheit und für den Betrieb ist die Einteilung in Bereiche von großer Bedeutung. Sie sind eigenständig lauffähig und auslieferbar. Bereiche sollen unabhängig voneinander entwickelt werden können, da sie von unterschiedlichen Teams betreut werden. Feedback bezüglich Fehler und Quelltext-Komplexität sollen im Rahmen von *Continuous Integration* gezielt dem Team rückgemeldet werden. Die Bereiche sollen lose gekoppelt über schmale und wohldefinierte Schnittstellen miteinander kommunizieren. Zur übergreifenden Kommunikation greifen sie entweder auf gekapselte und gezielt freigegebene Funktionalitäten anderer Bereiche über Bibliotheken zu, oder sie kommunizieren nur lose gekoppelt über RESTful-Webservices entsprechend dem coData-Protokoll. Um den obigen Forderungen gerecht zu werden, muss die technische Modularisierung - im Falle von CAMPUSonline wird diese mittels Apache Maven umgesetzt - auch auf dieser Ebene etabliert werden. Es gibt für jeden Bereich ein eigenes Maven-Projekt. Durch Evaluierungen aus der Praxis und durch Hochrechnungen des Umfang der entstehenden Entwicklungs-Artefakte hat sich gezeigt, dass die Einteilung in Bereiche als Gruppierungselement notwendig ist. Hier wäre zum Beispiel die Anzahl an Ordner auf dem Dateisystem erwähnt, die mit dieser Bereichseinteilung überschaubar bleibt.

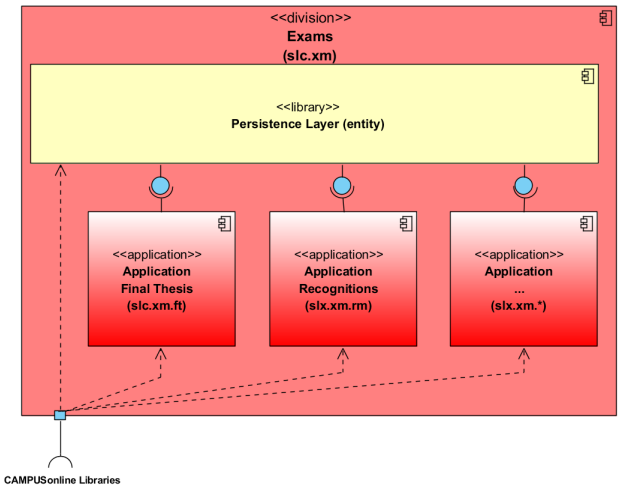

Abbildung 4.5.: Ein Bereich beinhaltet eine oder mehrere Applikationen. Alle Applikationen in einem Bereich nutzen dieselbe Persistenzschicht.

4.4. Applikationsebene

Durch eine Applikation kann eine abgeschlossene Funktionalität innerhalb eines Bereiches definiert werden. Durch das Aufkommen von mobilen Anwendungen ist das Denken in Applikationen mittlerweile bei den Endbenutzern weit verbreitet. Auch die Konfigurationsobjekte von CAMPUSonline sind zugunsten der Übersichtlichkeit meist auf dieser Ebene angesiedelt. Applikationen können allerdings mehrere Zugänge für den Endbenutzer anbieten. Je nach Benutzergruppe bzw. Anwendungsfall sind Zugänge Einsprungpunkte, die den Benutzer direkt an jene Stelle in einer Applikation leiten, die gerade benötigt wird.

Auf Applikationsebene wird bereits mittels Java-Packaging gearbeitet. Applikationen sollen im Sinne der Modularisierung auf keine Geschäftslogik anderer referenzieren. Stattdessen kann die Geschäftslogik in einer sogenannten Bereichs-Bibliothek allen zur Verfügung gestellt werden. Durch diese Strukturierung können Applikationen einfach gewartet und weiterentwickelt werden, ohne gravierende Seiteneffekte zu bewirken. Bei großen Applikationen kann dasselbe Muster durch Einführung von Applikations-Bibliotheken zur weiteren Modularisierung genutzt werden. Im Gegensatz

zu Segmentbibliotheken können Bereichs- bzw. Applikations-Bibliotheken durchaus viel an Geschäftslogik beinhalten.

4.5. Schichtenebene

Zusätzlich zur fachlichen Modularisierung wird eine technische Strukturierung von Applikationen in Schichten eingesetzt. Eine Standardapplikation baut aus diesem Grund auf ein striktes Schichtenmodell auf. Diese sind:

- Präsentationsschicht (Presentation Layer)
- REST-Schicht (REST Layer)
- Serviceschicht (Service Layer)
- Persistenzschicht (Persistence Layer)
- Datenhaltungsschicht (Data Layer)

Die Schichten erstrecken sich über eine typische 3-Tier-Architektur: Der Browser des Benutzers (Client-Tier), der Application-Server mit den Java EE Applikationen (Middle-Tier) und die Datenbank (Data-Tier) (Abbildung 4.6).

Die Präsentationsschicht beinhaltet sämtliche Implementierungen der anzuzeigenden Benutzeroberfläche. Die REST-Schicht übersetzt die Daten aus dem Service-Layer in XML bzw. JSON, um die Übertragung mittels HTTP-Protokoll hin zum Browser des Benutzers zu ermöglichen. Zusätzlich kümmert sich die REST-Schicht um das Verlinken von Ressourcen im Sinne von §HATEOAS. Die von der Präsentation unabhängige fachliche Logik wird dabei in der Serviceschicht implementiert, die auf den fachlichen Domänenobjekten (Entitäten) der Persistenzschicht operiert. In der Datenhaltungsschicht werden die Daten durch die Persistenzschicht langfristig gespeichert bzw. ausgelesen.

Prinzipiell gestaltet sich die Strukturierung der Geschäftslogik in CAMPUSonline EE entsprechend dem ECB-Muster. Bien (2009) beschreibt wie Geschäftslogik in Java EE anhand von ECB nachhaltig, verständlich und mit wenigen Redundanzen im Quelltext umgesetzt werden kann. Er definiert

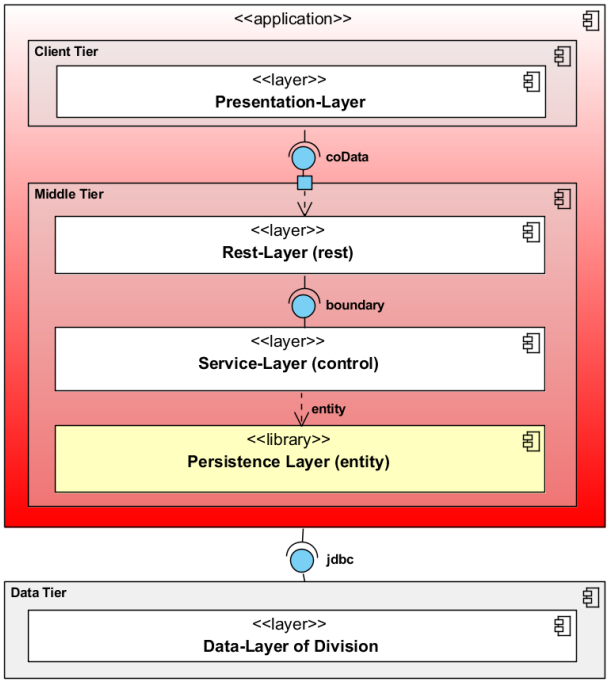

Abbildung 4.6.: Die technische Modularisierung erfolgt in einem klassischen Schichtenmodell. Die Präsentationsschicht greift lose gekoppelt über das coData-Protokoll auf den REST-Layer zu, welcher die Anfragen an die Geschäftslogik an die Serviceschicht weiterleitet. Die Serviceschicht gestaltet sich entsprechend dem Muster von Entity Control Boundary (ECB).

entsprechend dem ECB-Musters drei Strukturierungseinheiten, die durch separate Java-Packages abgebildet werden. Dabei bildet das Boundary die definierte und schmale Schnittstelle zur übergeordneten Präsentationsschicht. Wird die Logik im Boundary zu komplex bzw. müssen Teile der internen Geschäftslogik vor der Präsentationsschicht versteckt werden, so soll diese Geschäftslogik im Control implementiert werden. Für Adam Bien ist es aber essentiell, dass im Boundary begonnen wird die Implementierung umzusetzen und erst bei Bedarf die Geschäftslogik ins Control herausgezogen wird. Bei den Entitäten plädiert Bien dafür, dass Geschäftslogik, die den fachlichen Objekt inhärent ist, direkt in den Entitäten implementiert werden soll. Nur übergreifende Logik soll in Boundary bzw. Control umgesetzt werden.

Während die Präsentationsschicht auf die REST-Schicht über das coData-Protokoll zugreift, kommuniziert die REST-Schicht mit der Serviceschicht über eine dedizierte Schnittstelle, dem sogenannten Boundary. Die Kommunikation erfolgt dabei über *data transfer objects* (DTO), die bereits für spezielle Anwendungsfälle aufbereitet sind. Die DTOs beinhalten ausschließlich Daten und Metadaten, aber keine Geschäftslogik. Sie dienen dazu, das darunter liegende Datenmodell der Persistenzschicht vollständig zu kapseln. Somit ist die Präsentationsschicht unabhängig von der implementierten Geschäftslogik änderbar bzw. austauschbar. Gerade wenn sich Präsentationstechnologien sehr schnell ändern und alternative Oberflächen für unterschiedliche Endgeräte gefordert werden, ist es nachhaltig die Geschäftslogik in einer eigenen gekapselten Serviceschicht verwahrt zu wissen.

Die Serviceschicht greift auf die Entitäten der Persistenzschicht mittels dem Repository-Muster zu. Fowler (2002) beschreibt in seinem Buch das Repository-Muster, als eine zusätzliche Schicht, die sich rein um die Formulierung von Datenbankabfragen kümmert. Dadurch kann gerade bei großen Entitätsmodellen - wie bei CAMPUSonline EE - Quelltext Duplizierung vermieden und die Testbarkeit des Quelltextes erhöht werden. Der Zugriff auf die Datenbank erfolgt über Java Database Connectivity (JDBC). Zur Unterstützung der Datenbankunabhängigkeit wird auf eine objektrelationale Abbildung mittels JPA zurückgegriffen.

5. Kontextuelle rollenbasierte Zugriffskontrolle

Role-Based Access Control (RBAC) ist ein weit verbreitetes Modell um Informationssysteme abzusichern und wurde 2004 als ANSI-Norm 359-2004 veröffentlicht. Benutzern werden dabei entsprechend ihren Verantwortungen 1 bis n Rollen zugeordnet. Dadurch kann die Berechtigung der angemeldeten Person abstrahiert anhand von Rollen abgefragt werden. Die Administration mittels Rollen gestaltet sich wesentlich einfacher und effizienter als die Zuweisung von Berechtigungen direkt an die Benutzer mittels einer Zugriffssteuerungsliste (R. S. Sandhu u. a., 1996).

Wie in Abbildung 5.1 illustriert werden in RBAC Rollen der angemeldeten Person (Subjekt) in die aktuelle Sitzung des Benutzers geladen. Aufgrund der Zuordnung von Rollen zu Berechtigungen erhält der Benutzer die Erlaubnis Aktionen auf bestimmte Objekte auszuführen. Es werden zwei Ausbaustufen bei RBAC unterschieden. Die erste Ausbaustufe führt hierarchische Rollen ein. Die Vererbung von Rollen vereinfacht die Administration der Benutzer-Rollen-Zuordnung insofern, dass ein Benutzer durch die Zuordnung einer übergeordneten Rolle gleichzeitig alle untergeordneten Rollen erhält. Bei der zweiten Ausbaustufe handelt es sich um die Funktionstrennung auch *Separation of Duties* genannt. Bei der Funktionstrennung geht es darum unrechtmäßige Handlungen zu unterbinden. Zum Beispiel soll sich ein Studierender nicht selbst benoten können. Diese Funktionstrennung kann bereits statisch bei der Rollenvergabe erfolgen, indem verhindert wird, dass einem Benutzer sich gegenseitig ausschließende Rollen zugeordnet werden. Zusätzlich kann die Funktionstrennung auch zur Laufzeit erfolgen, wenn die Rollen in die Sitzung geladen werden.

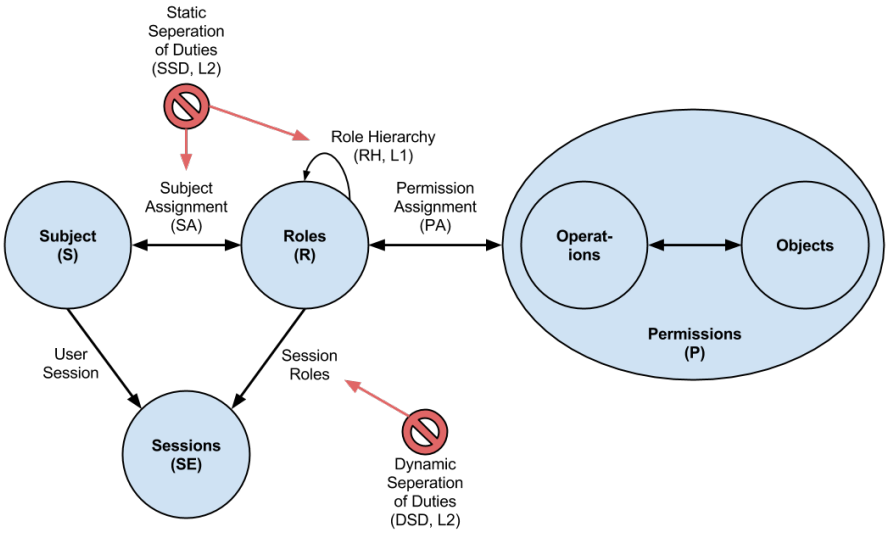

Abbildung 5.1.: In RBAC werden einem Subjekt Rollen innerhalb einer Session zugeordnet, wodurch der angemeldete Benutzer Berechtigungen erhält.

5.1. RBAC im Einsatz an Campus-Management-Systemen

Der Einsatz von RBAC an Campus-Management-Systemen ist weit verbreitet. Jedoch stößt man mit der reinen RBAC Umsetzung an Grenzen, welche durch individuelle und kontextspezifische Erweiterungen umschifft werden.

Im Paper von Luo und S. Xu (2010) wird beschrieben wie das RBAC Modell erfolgreich in einem CMS implementiert werden konnte. Sie beschreiben den Zusammenhang zwischen Benutzer, Rollen und Berechtigungen und weisen ausdrücklich darauf hin, wie zentral die Bedeutung eines einheitlichen und zentralen Zugriffskontrollsystems für eine heterogene universitäre Systemlandschaft ist. Zusätzlich beschreiben die Autoren, wie sich aufgrund der Berechtigungen die Benutzeroberfläche entsprechend an die Geschäftsregeln anpasst. Um redundante Implementierungen zu vermeiden wird typischerweise eine Seite für verschiedene Benutzergruppen konzi-

piert. Aktionen und Navigationsmöglichkeiten zu denen der Benutzer in der aktuellen Sitzung nicht berechtigt ist, werden dynamisch ausgeblendet.

Zheng, Jiang und Liu, 2009 stellten fest, dass das RBAC Modell trotz seiner weiten Verbreitung als Zugriffskontrolle einige Limitierung hat. Die Anforderungen an ein universitäres Identitäts- und Zugriffskontrollsystem übersteigen die Möglichkeiten des RBAC Modells sowohl in der Administrierbarkeit als auch in der Abfragegeschwindigkeit. Viele Berechtigungsabfragen sind sehr dynamisch und hängen stark von der aktuellen Kontextinformation ab. Außerdem stellten die Autoren fest, dass die Vergabe von Rollen statisch ist. Hingegen bestimmt die Beziehung zwischen der handelnden Identität und dem Objekt auf das eine Aktion ausgeübt werden soll, ob eine Rolle wahrgenommen werden darf oder nicht.

5.2. RBAC kombiniert mit ABAC (RBAC-A)

Franqueira und Wieringa (2012) analysieren in ihrer Arbeit die Vor- und Nachteile von RBAC und erkannten, dass es in Systemen mit vielen unterschiedlichen Benutzern und Objekten zu einer schwer administrierbaren Explosion der Rollen kommen kann. Diese Rollen-Vielfalt wird hauptsächlich dadurch begründet, dass eine Berechtigung eine Aktion auf ein Objekt auszuführen oft von einer Vielfalt anderer Parameter abhängt als nur von den Rollen der angemeldeten Person. Zum Beispiel kann die Berechtigung aus dem Umstand resultieren, dass der Benutzer zum Objekt einen Bezug hat (Prüfer dürfen nur ihre eigenen Prüfungen benoten) oder der aktuelle Kontext des Benutzers verleiht ihm die Berechtigung (Administratoren dürfen die Lehrveranstaltungsplanung nur an einer ihnen zugeordneten Organisation planen). Aber auch ein dynamischer Systemkontext kann dazu führen, dass ein Benutzer Berechtigungen erhält (Studierende können sich erst um Mitternacht zur Prüfung anmelden).

Eine weitere Herausforderung ist der Aufbau und die Wartung der Hierarchie der Rollen. Diese anspruchsvolle Aufgabe benötigt ein tiefes Verständnis der abzubildenden Domäne und führt laut Franqueira und Wieringa, 2012 trotzdem immer wieder zu unerwarteten Seiteneffekten. Bei einer Vielzahl an zu verwalteten Rollen kann es schnell zu einer Fehlkonfiguration

kommen, die einzelne Benutzer zu viele oder zu wenige Berechtigungen zuteilt.

Aus diesen Gründen wurde von D. R. Kuhn, Coyne und Weil, 2010 eine Durchmischung der Konzepte von RBAC und Attribute-Based Access Control (ABAC) vorgeschlagen. ABAC arbeitet mit sogenannten Policies in welchen die Berechtigungslogik anhand der Attribute des Benutzers (Subjekt), des Objektes und der Systemumgebung formuliert wird (Hu u. a., 2013). ABAC erscheint wesentlich flexibler als RBAC, da keine Definition einer Rollen Hierarchie notwendig ist, sondern direkt die für die Berechtigung notwendige Policy definiert werden kann. Dadurch ergibt sich allerdings eine Komplexität bei der Auswertung, welche Berechtigungen einem einzelnen Benutzer nun tatsächlich zugesprochen sind (D. R. Kuhn, Coyne und Weil, 2010). Im schlimmsten Fall müssen für einen Benutzer sämtliche Policies mit allen möglichen Objekten durchlaufen werden, um eine Liste all ihrer Berechtigungen zu erhalten. Im Gegensatz zu RBAC, wo jederzeit einfach ermittelt werden kann, welche Berechtigungen ein Benutzer besitzt.

In der Arbeit von D. R. Kuhn, Coyne und Weil (2010) werden drei Möglichkeiten vorgestellt wie RBAC und ABAC kombiniert werden können, die wiederum in einer Mischform zum Einsatz kommen können.

1. Im **attributzentrierten Ansatz** wird die Rolle als ein weiteres abfragbares Attribut in einer ABAC-Policy interpretiert. Durch dieses Vorgehen wird die Flexibilität erhöht und die Explosion der benötigten Rollen kann verhindert werden. Die Intransparenz zwischen Benutzer und Berechtigungen bleibt allerdings bestehen.
2. Im **rollenzentrierten Ansatz** werden die groben und primären Berechtigungsabfragen stets gegen die Rollen des Subjektes durchgeführt. Mittels einer auf ABAC basierenden Policy kann die Berechtigung noch weiter eingeschränkt werden. Dadurch bleiben die Vorteil beider Paradigmen zum großen Teil erhalten.
3. Beim Einsatz von **dynamische Rollen** wird das Set der maximal zur Verfügung stehenden Rollen pro Benutzer dynamisch zur Laufzeit aufgrund einer Policy gefiltert. Sprich an den aktuellen Kontext angepasst. Dieses Vorgehensmodell kann auch als Vorstufe mit dem rollenzentrierten Ansatz kombiniert werden.

Für CAMPUSonline EE wurde der rollenzentrierte Ansatz in Kombination mit dynamischen Rollen gewählt. Zusätzlich wird noch der Aspekt des Kontextes, wie im Grundsatz §CONTEXT beschrieben, berücksichtigt.

5.3. Entitäten

Das im folgenden vorgestellte Konzept der kontextuellen rollenbasierten Zugriffskontrolle (CORBAC) basiert auf vier Entitäten, die dem Datenmodell 5.2 entnommen werden können. Die vier Entitäten sind Identität (`identity`), Rolle (`role`), Context (`context`) und Rollenzuordnung (`assignment`). Die tatsächlichen Datenstrukturen zur Vergabe von Rollen in einem System sind im Hintergrund oft wesentlich komplexer. Zum Beispiel wird in CAMPUSonline ein Organisations-Metamodell verwendet, auf Basis dessen die Rollenzuordnung an den einzelnen Einrichtungen einer Universität erfolgt. Schlussendlich soll eine möglichst einfache aber gleichzeit mächtige Datenstruktur befüllt werden, die als Schnittstelle für unterschiedlichste Rollenzuordnungs-Mechanismen dienen kann. Diese Schnittstelle kann folglich einheitlich abgefragt und referenziert werden, und auch konzeptionell für andere Systeme übernommen werden. Durch die einfache Ausprägung der Schnittstelle ist es möglich, dass externe Systeme CAMPUSonline als zentrales Identitäts- und Zugriffsmanagement-System nutzen. Zum Beispiel können so die Nutzungsberechtigung von Ressourcen (Schließanlagen, Gebäude, IT-Infrastruktur, ...) an einer Bildungseinrichtung zentral gesteuert werden. Externe Systeme können über RESTful-Webservices die aktuellen Rollen eines Benutzers einheitlich und lose gekoppelt abfragen. Die Verwaltung der Zuordnungen kann zentral im CAMPUSonline System erfolgen. Da unterschiedlichste Anforderungen und Datentypen in Rollen-verwaltenden-Systemen verwendet werden, wurde die Datenstruktur sehr offen und flexibel definiert. Jeder verwendete Schlüssel wird typisiert und verwendet `VARCHAR` als Datentyp. In den folgenden Abschnitten werden die einzelnen Entitäten kurz vorgestellt.

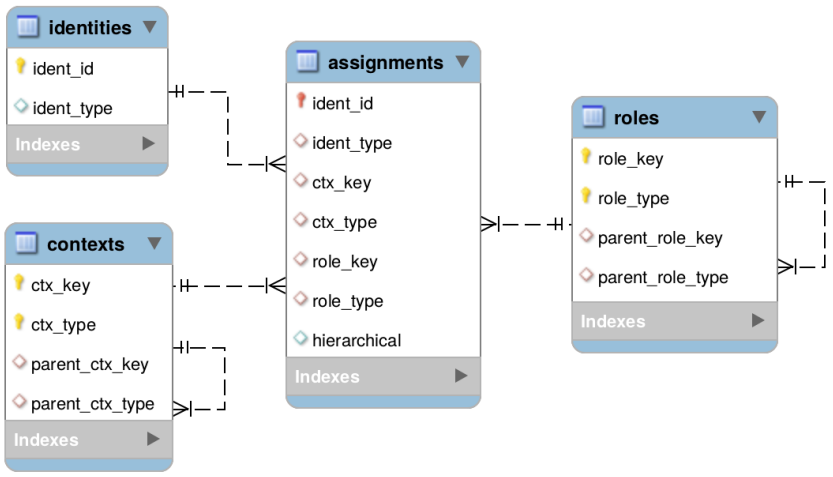

Abbildung 5.2.: Das Datenmodell für die Zuweisung einer Rolle in einem Kontext zur Abbildung von CORBAC ist aus vier Entitäten aufgebaut. Es wird automatisiert durch unterschiedliche Mechanismen und Systeme im Sinne einer neutralen Schnittstelle befüllt.

5.3.1. Rollen

Eine Rolle definiert die Rechte, die Benutzer erhalten, wenn ihnen diese Rolle zugeteilt wird. Um die Administration der Rollenzuordnung zu erleichtern, sind Rollen hierarchisch strukturiert. Aus der Sicht der Entwicklung werden Rollen im Quelltext dazu verwendet, um die Berechtigungen der Benutzer zu bestimmen. Das bedeutet, es gibt Rollen, die statisch im Quelltext referenziert werden und solche die dynamisch durch die Administratoren angelegt werden und baumartig strukturiert sind. Die dynamischen Rollen werden durch ein Mapping in statisch abfragbare Rollen transformiert.

In CAMPUSonline EE werden die statischen Rollen relational erfasst und als Applikationsrollen bezeichnet. Applikationsrollen sind dabei die granularste Stufe an Rollen welche das System unterstützt. Die Modularisierung in Applikationsrollen, also die Zuordnung einer statischen Rolle zu einem Applikations-Programmmodul, hilft der Administration die Auswirkung einer Applikationsrolle schnell zu klassifizieren. Dadurch dass eine Applikationsrolle auch genau ein spezifisches Applikationsrecht innerhalb einer Applikation verleiht, kann jederzeit die Berechtigung eines Benutzers auf

Applikationsebene bestimmt werden. Die Regeln zur Transformation der dynamisch anlegbaren Rollen in Applikationsrollen sind in einem relationalen Datenmodell gespeichert. Dynamische Rollen können aus einer Vielzahl unterschiedlicher Quellen stammen. In CAMPUSonline werden sie meist auf der Ebene von Organisationstypen der Bildungseinrichtungen definiert und als Funktionen von Personen bezeichnet. Die Zuordnung erfolgt hierbei manuell. Rollen können aber auch aus einer dynamisch gebildeten Gruppe oder aus anderen Systemen übernommen werden. Die Rollen von Studierenden werden zum Beispiel dynamisch aufgrund einer Klassifizierung von Studierenden nach Studium oder Semester ermittelt und automatisiert periodisch in die Rollenzuordnung eingetragen.

Wichtig dabei ist eine einfache hierarchische Datenstruktur zur Verfügung zu stellen, welche aus unterschiedlichen Quellen bespielt werden kann, um das Mapping auf Applikationsrollen schlussendlich durchführen zu können. Die Tabelle `roles` in Abbildung 5.2 ist eine typisierte, hierarchische Struktur, die als Schnittstelle für die Entwicklung verwendet wird. Nur die Blätter des Baumes, die vom Typ Applikationsrolle sind, werden tatsächlich in der Entwicklung referenziert.

5.3.2. Identitäten

Identitäten sind allgemein beschrieben Personen, Gruppen oder auch Ressourcen, die innerhalb des Systems aufgrund ihrer Merkmale und Attribute eindeutig identifiziert werden können. Im Kontext von RBAC sind Identitäten jene Entitäten denen Rollen zugeordnet werden können. Sie werden in Identitätsmanagement-Systemen verwaltet und sind typischerweise aber nicht zwingend, Benutzer des Systems. Greift ein Benutzer auf das System zu, wird die Identität als handelndes Individuum oder als Subjekt bezeichnet. In CAMPUSonline werden drei grundlegende Arten von Identitäten unterschieden:

1. **Personenbezogene Identitäten** sind reale Personen, die eindeutig in CAMPUSonline identifizierbar sind.
2. **System Identitäten** sind Systeme, die auf CAMPUSonline automatisiert - ohne menschliche Interaktion - zugreifen.

3. Die **anonyme Identität** ist die Gruppe aller nicht authentifizierten Zugriffe auf CAMPUSonline, die im Sinne der Identität gleichgestellt behandelt werden.

Der Zugriff auf CAMPUSonline erfolgt immer mit einem eindeutigen OAuth2-Zugriffstoken. Auf Basis dieses Zugriffstokens kann sowohl die personenbezogene Identität als auch das zugreifende System, wie unter §AUTHN beschrieben, ermittelt werden. Durch die Aufteilung in die drei grundlegenden Identitätsarten können nicht nur reale Personen Berechtigungen im System erhalten sondern auch Fremdsysteme, um die Maschine zu Maschine Interaktion zu unterstützen. Fremdsysteme können sich mittels des in OAuth2 beschrieben *Client Credential Flows* ohne menschliche Interaktion an CAMPUSonline authentifizieren und erhalten ebenfalls die Rollen, welche ihnen zugeordnet wurden. Durch die Möglichkeit auch der Gruppe der anonymen Identitäten Rolle zuzuordnen, kann jede Bildungseinrichtung für sich selbst entscheiden, welche Informationen öffentlich zugänglich sind. In der Entwicklung von Berechtigungsabfragen werden nur Rollen abgefragt und es muss nicht zwischen den drei Identitätsarten unterschieden werden, was große Vorteile in Bezug auf die Automatisierung bringt.

5.3.3. Kontexte

Die Erweiterung des klassischen RBAC-Modells, um organisationsbezogene Berechtigungen wurde bereits in mehreren wissenschaftlichen Arbeiten beschrieben und wird von Kalam u. a. (2003) als *Organization based access control* (ORBAC) oder von Z. Zhang, X. Zhang und R. Sandhu (2006) als *Role and Organization Based Access Control* (ROBAC) bezeichnet. Bei der organisationsbasierten Zugriffskontrolle wird einer Identität eine Rolle innerhalb einer Organisation verliehen. Bewegt sich die Person nun innerhalb des Kontextes dieser Organisation kann die Rolle wahrgenommen werden und der Benutzer erhält die der Rolle zugeordneten Berechtigungen. Umgekehrt kann ein Benutzer, der sich außerhalb des Kontextes dieser Organisation bewegt, nicht über die ihm zugeordneten Rollen verfügen. Zum Beispiel darf ein Prüfer an der Fakultät für Architektur keine Prüfungen im Rahmen der Fakultät für Mathematik abnehmen. Die Vergabe der Rollen erfolgt oft auch hierarchisch. Das heißt, hat man die Rolle an einer Fakultät, hat man

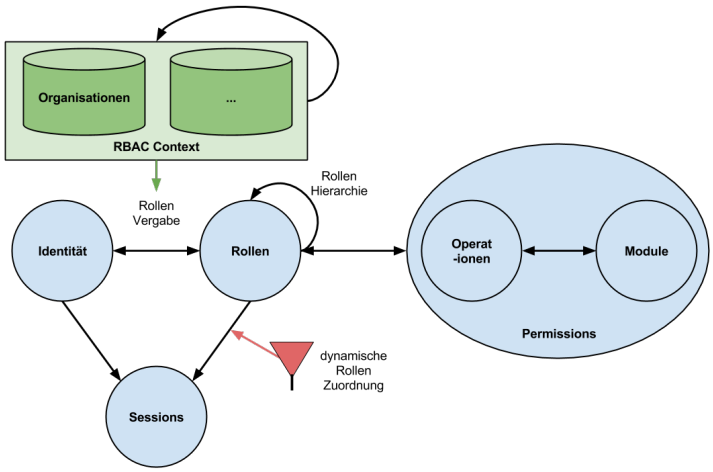

Abbildung 5.3.: Die Zuordnung von Rollen erfolgt in CORBAC immer innerhalb eines Kontextes. Bei einem Zugriff auf das System werden nur jene Rollen dynamisch in die Web-Sitzung geladen, welche dem aktuellen Kontext des Benutzers entsprechen.

auch die Rolle an allen Instituten der Fakultät. Während viele Bildungseinrichtungen ihre Rollen tatsächlich auf der Ebene von Organisationen verwalten - zum Beispiel die TU Graz oder aber auch die Technische Universität München - gibt es Hochschulen die es bevorzugen ihre Rollen im Rahmen von Studienrichtungen zu vergeben. Eine andere Anforderung ist die Vergabe von Rollen im Kontext einer Benutzergruppe, um beispielsweise der Benutzergruppe Alumni die Nutzung des E-Mail-Systems der Bildungseinrichtung weiterhin zu gestatten, oder eben nicht.

Vom Konzept der organisationsbezogenen Berechtigungen und den beschriebenen Anforderungen wird das generalisierte Konzept der kontextuellen Berechtigungen abgeleitet, welches in Abbildung 5.3 dargestellt wird. Der Kontext ist das entscheidende Element von CORBAC. Identitäten erhalten eine Rolle immer innerhalb eines Kontextes und können diese Rolle auch nur in diesem Kontext wahrnehmen. Ein Kontext kann zum Beispiel eine Organisation, eine Studienrichtung, eine Benutzergruppe, ein Gebäude oder ähnliche situationsbedingte Eigenschaften eines Benutzers sein. Benutzer können, wie im echten Leben, über ihren Kontext selbst bestimmen. Sie

können ihren Kontext wechseln oder ihn auch verlassen. Die Auswahl des Kontextes erfolgt aus Usability Sicht mittels eines Kontextauswahl-Dialoges (Context Chooser) und wurde bereits in 2.4 dargestellt. Kontexte sind hierarchisch organisiert. Das heißt ein Kontext kann untergeordnete Kontexte enthalten. Ähnlich den Organisationen können so Rollen auf einer übergeordneten Ebene vergeben werden, die sich auf die untergeordneten Kontexte vererben. Daraus ergibt sich, die in Abbildung 5.2 dargestellte hierarchische Datenstruktur für Kontexte (Tabelle `contexts`).

Es handelt sich bei diesem Konzept um eine Spezialisierung des Konzeptes der dynamischen Rollen, das von D. R. Kuhn, Coyne und Weil (2010) vorgeschlagen wurde. Der Unterschied liegt darin, dass ein hartes referentielles Modell anstelle von flexiblen ABAC-Policies verwendet wird, um zu definieren wann eine Rolle für die angemeldete Person nutzbar ist. Dadurch wird die Transparenz in der Verwaltung erhöht und es lässt schnellere Rückschlüsse in der Analyse von Berechtigungen einer Identität zu.

5.3.4. Rollenzuordnung

Die Zuordnung von Rollen erfolgt, wie in Abbildung 5.2 dargestellt, mittels der Entität `assignments`. Einer Identität wird eine Rolle innerhalb eines Kontextes zugeordnet. Die Zuordnung der Rollen kann mittels des Flags `hierarchical` sowohl hierarchisch auf untergeordnete Kontexte vererbt, oder genau auf den zugeordneten Kontext eingeschränkt werden. Wird der Kontext hierarchisch vererbt, so verfügt der Benutzer auch in untergeordneten Kontexten über die zugeordneten Rollen.

Um den aktuellen Kontext vom Client an den Server zu übertragen werden zwei Werte übermittelt. Der *rbacType* identifiziert den Typ des Kontextes (z.B. Typ Organisation) und die *rbacId* definiert das konkrete Kontextelement innerhalb des Typs (z.B. Institut für Mathematik). Auf Basis dieser zwei übertragenen Werte und der über den OAuth2-Zugriffstoken (siehe §AUTHN) ermittelbaren Identität können die Rollen des Subjektes im aktuell gewählten Kontext bestimmt, und die aktiven dynamischen Rollen in die Web-Sitzung geladen werden. Auf Basis dieses eingeschränkten Sets an Rollen, kann nun serverseitig geprüft werden, ob der Benutzer über die

notwendigen Rollen verfügt. Die Überprüfung kann durch eine deklarative Angabe von benötigten Rollen durch das Framework automatisiert werden. Damit wird eine grobkörnige Zugriffskontrolle sichergestellt, bevor jegliche andere Server-Funktionalität aufgerufen wird. Nun kann im Sinne des Rollenbasierten Ansatz von D. R. Kuhn, Coyne und Weil (2010) die Berechtigung mittels ABAC noch weiter eingeschränkt werden.

5.4. Abfrage von Rollen in Java EE

Die Absicherung einer Ressource erfolgt in Java aus der Sicht des Anwendungsentwicklers sehr einfach mittels der Annotation *@RolesAllowed* auf der mittels JAX-RS exponierten Methode. Durch diese deklarative Vorgehensweise können Logikfehler verhindert und eine Absicherung an der Basis sicher gestellt werden, da ein Zugriff ohne Rolle vom System automatisch verhindert wird. Anonyme Ressourcen müssen explizit mit der Annotation *@PermitAll* freigegeben werden. In 5.1 wird eine Liste von Lehrveranstaltungen als Ressource angeboten. Mittels *@RolesAllowed* wird definiert, dass nur Identitäten mit den Rollen Studierende oder Lehrende auf diese zugreifen dürfen. Außerdem wird mit *@CoRequireCtxType(CoRbacCtxType.ORG)* definiert, dass ein Organisationskontext erforderlich ist. Beim Zugriff auf die Geschäftslogik über das CoursesService wird die aktuelle rbacId - in diesem Fall der Schlüssel der Organisation - übergeben. Dadurch kann in der Geschäftslogik auf jene Lehrveranstaltungen eingeschränkt werden, die mit der übergebenen Organisation relational verbunden sind.

Listing 5.1: Entwickler können deklarativ über Annotationen die notwendigen Rollen und den notwendigen Kontext bekannt geben.

```
1  @Path("courses")
2  public class CoursesRestAdapter {
3
4    @Inject
5    private CoSysCtx sysCtx;
6
7    @Inject
8    private CoursesService service;
9
10   @GET
11   @CoRequireCtxType(CoRbacCtxType.ORG)
12   @RolesAllowed({"STUDENT", "LECTURER"})
13   public List<Course> getAllCourses() {
```

```
14      // return all courses of organization
15      return service.getAllCoursesByOrganization(sysCtx.getRbacId());
16    }
17
18  }
```

Auswahl des RBAC Kontextes durch den Benutzer

Die einzelnen Seitenkomponenten einer SPA erfordern einen RBAC-Kontext in welchem sie geöffnet werden. Wie unter Wahl des Kontextes (§CONTEXT) beschrieben, kann diese Auswahl sowohl durch Entwickler mittels Metadaten, als auch durch Administratoren durch die Verwendung von Template-Variablen bei der Konfiguration eines Zugangs gefordert werden. Aber auch direkt auf einer geöffneten Seite können Kontextauswahlmöglichkeiten angeboten werden.

Übertragen des RBAC-Kontextes an den Server

Der RBAC-Kontext wird immer in der Form zweier Werte als HTTP-Abfrageparameter übertragen. Die zwei Werte sind der RBAC-Kontext-Typ (rbacType) und die RBAC-Kontext-Id (rbacId) und entsprechen den Feldern der Tabelle `contexts`.

Benötigte Rollen und Kontexte identifizieren

Mittels eines JAX-RS *ContainerRequestFilters* können Queraspekte wie zum Beispiel Authentifizierung und Autorisierung abgebildet werden. Bevor die Methode der Ressource aufgerufen wird, können im *ContainerRequestFilter* die Annotationen *@RolesAllowed*, *@CoRequireCtxType*, *@PermitAll* und *@DenyAll* ausgelesen werden und die notwendigen Rollenzuordnungen für den Ressourcen-Zugriff bestimmt werden. Die Annotationen können sowohl auf Methoden-Ebene als auch auf Klassen-Ebene verwendet werden. Annotationen auf Methoden-Ebene überschreiben dabei Annotationen der Klassen-Ebene. Die einzelnen Annotationen haben folgende Bedeutungen:

1. **@RolesAllowed**: Eine der angegeben Rollen muss dem Subjekt im aktuellen Benutzerkontext zugeordnet sein.
2. **@PermitAll**: Es ist egal über welche Rollen das Subjekt verfügt. Trotzdem werden alle Applikationsrollen in das Subjekt geladen, sodass der Entwickler selbst entscheidet, wie die Berechtigungsabfrage seiner Ressource gestaltet ist.
3. **@DenyAll**: Verhindert den Zugriff auf die Ressource und führt automatisch zu einer HTTP 403 *Forbidden* Antwort.

Rollenhierarchie und Kontexthierarchie laden

Die Rollenhierarchie und Kontexthierarchie können von den Tabellen `roles` und `contexts` aus der Datenbank geladen werden und in einer aufbereiteten Form am Applikations-Server für alle Benutzer gecached werden. Die aufbereitete Struktur (5.2) entspricht einer Matrix, in welcher jede Rolle bzw. Kontext als Schlüssel vorhanden ist. Hinter dem Schlüssel wird die aufgelöste Hierarchie in Form einer linearisierten Liste von Schlüsseln gespeichert. Diese beiden Strukturen werden als *RoleMap* und als *CtxMap* bezeichnet. Es verbergen sich beispielsweise hinter dem Schlüssel der Universität eine Liste alle Fakultäten, Institute usw. Dadurch kann die Performance gesteigert werden, da alle Benutzer die gleiche indizierte Datenstruktur verwenden, die nicht bei jedem Zugriff aus der Datenbank geladen werden muss. Die Rollenhierarchie und Kontexthierarchie ändern sich nur selten. Daher müssen die zwischengespeicherten Werte nur periodisch (z.B. jede Minute) aktualisiert werden. Durch das Caching wird der Grundsatz der Zustandslosigkeit (§NO-STATE) nicht verletzt, da jeder Applikation-Server unabhängig vom Benutzer und den anderen Applikation-Servern das Caching autonom durchführen kann.

Listing 5.2: In der *RoleMap* und *CtxMap* werden die Rollenhierarchie und Kontexthierarchie für alle Benutzer am Applikationsserver zwischengespeichert.

```
1  Map<CoRbacCtx, Set<CoRbacCtx>> ctxMap;
2  Map<CoRole, Set<CoRole>> roleMap;
```

Identität des Benutzers ermitteln

Auf Basis des OAuth2-Zugriffstoken kann, wie unter Authentifizierung (§AUTHN) beschrieben, die Identität des Benutzers ermittelt werden und eindeutig zwischen personenbezogenen Identitäten, System Identitäten und einem anonymen Zugriff unterschieden werden.

Rollenzuordnung des Benutzers laden

Es werden alle zugeordneten Rollen des Benutzers standardmäßig bei jedem Zugriff auf die Datenbank neu geladen, um die Aktualität der Rollenzuordnung zu garantieren und das Prinzip der Zustandslosigkeit (§NO-STATE) nicht zu verletzten. Dadurch wird die Vergabe aber auch der Entzug von Rollen unverzüglich wirksam.

Aktive Rollenzuordnungen filtern

Die tatsächliche Filterung der Rollen erfolgt wie unter 5.3 dargestellt. Um denn Quelltext zu verstehen werden zuerst der *AllCtx* und der *AnyCtx* vorgestellt. Durch den *AllCtx* können alle zugeordneten Rollen eines Benutzers abgefragt werden, ungeachtet dessen in welchen Kontext dieser eine Rolle erhalten hat. Zum Beispiel kann auf diese Weise entschieden werden, ob ein Zugang angezeigt werden soll, auch wenn der Benutzer noch keinen Kontext gewählt hat. Somit kann die konkrete Kontextauswahl nach der Aktivierung des Zuganges angeboten werden. Der *AnyCtx* gibt Administratoren die Möglichkeit den Benutzer einer Rolle unabhängig eines Kontextes zuzuordnen. Egal welchen Kontext der Benutzer aktuell gewählt hat, er erhält die mit *AnyCtx* zugeordneten Rollen immer.

Daraus ergeben sich folgende Bedingungen für aktive Rollen. Eine Rollenzuordnung gilt dann als aktiv,

- wenn der Kontext des Benutzers nicht relevant ist (*AllCtx*).
- wenn die Rollenzuordnung unabhängig vom Kontext auf jeden Fall erfolgen soll (*AnyCtx*).
- wenn der aktuelle Benutzerkontext dem der Zuordnung entspricht.

- wenn die Zuordnung hierarchisch ist und der aktuelle Benutzerkontext in der Liste der untergeordneten Kontexte vorkommt, welche über die zwischengespeicherte *CtxMap* bezogen werden kann.

Wird keine dieser Bedingungen erfüllt gilt eine Rollenzuordnung im aktuellen Kontext als inaktiv und der Benutzer kann diese Rolle nicht ausüben.

Listing 5.3: Abfrage nach aktiven Rollen im aktuellen Benutzerkontext.

```
1  public boolean isActive(
2    final CoCtxMap ctxMap,
3    final CoRbacCtx currentRbacCtx) {
4
5    if (currentRbacCtx instanceof CoAllCtx) {
6      return true;
7    }
8
9    if (getRbacCtx()  instanceof CoAnyCtx)) {
10     return true;
11   }
12
13   if (getRbacCtx().equals(currentRbacCtx)) {
14     return true;
15   }
16
17   if (isHierarchicalInCtx()) {
18     final Set<CoRbacCtx> assignedRbacCtxElements =
           ctxMap.getAllRbacCtxElements(getRbacCtx());
19     return assignedRbacCtxElements.contains(currentRbacCtx);
20   }
21
22   return false;
23 }
```

Applikationsrollen laut Rollen Baum bestimmen

In diesem Schritt werden auf Basis der aktiven Rollenzuordnung alle Applikationsrollen bestimmt, welche dem Benutzer aufgrund der Rollenhierarchie verliehen wurden. Dazu wird die *RoleMap* verwendet, die zuvor für alle Benutzer im Applikations-Server gecached wurde. Die Basis zur Ermittlung der zugeordneten Applikations-Rollen sind die Identität des Benutzers und der aktuelle Benutzerkontext. Die Implementierung des Algorithmus erfolgt mit Hilfe von funktionaler Programmierung, da diese eine besonders prägnante und Fehler unanfällige schreibweise zulässt.

Zuerst werden alle Rollenzuordnungen des Benutzers auf Basis der Identität geladen. Anschließend wird mit der bereits beschrieben Methode *isActive* gefiltert, sodass nur mehr aktive Rollenzuordnungen im Set vorhanden sind. Aus der Rollenzuordnung wird als nächstes die Rolle extrahiert und mittels der *RoleMap* für jede Rolle eine Liste aller resultierenden Applikationsrollen bestimmt. Diese Listen werden mittels dem *flatMap*-Operator wieder zu einem einzigen Set an Applikationsrollen aggregiert.

Listing 5.4: Ermittlung aller aktiven Applikationsrollen eines Benutzers.

```
1  public CoRoleSet getEnabledRoleSetInCtx(
2    final CoIdentity identity,
3    final CoRbacCtx currentRbacCtx) {
4
5    final Set<CoRoleAssignment> assignments = getRoleAssignments(identity);
6
7    final Set<CoRole> roles = assignments.stream()
8      .filter(assignment -> assignment.isActive(getCtxMap(), currentRbacCtx))
9      .map(CoRoleAssignment::getRole)
10     .map(getRoleMap()::getAllRoleElements)
11     .flatMap(Collection::stream)
12     .collect(Collectors.toSet());
13
14     return CoRoleSet.create(roles);
15 }
```

Rollen im Subjekt speichern

Das Subjekt repräsentiert die aktuell zugreifende Person und beinhaltet eine Liste aller dem Benutzer zugeordneten Rollen und andere sicherheitsrelevante Eigenschaften der Person. Im konkreten Fall wird die Identität, die Applikationsrollen der Person im aktuellen Kontext und alle potentiellen Applikationsrollen, in das typisierte Subjekt gespeichert. Die Typisierung des Subjektes mit dem Kontext beugt Missverständnissen bei der Entwicklung vor. Aufgrund des häufigen Anwendungsfalls einer kontextlosen Abfrage nach potentiellen Applikationsrollen, werden auch diese standardmäßig im Subjekt verfügbar gemacht.

Listing 5.5: Das Subjekt wird auf Basis des aktuellen Benutzerkontext gebildet.

```
1  public <C extends CoRbacCtx> CoSubject<C> getSubjectInCtx(final CoIdentity
       identity, final C rbacCtx) {
2
```

```
3     final Set<CoRoleAssignment> assignments =
          roleService.getRoleAssignments(identity);
4
5     return CoSubject.Builder
6       .withIdentity(identity)
7       .enabledRoles(roleService.getEnabledRoleSetInCtx(assignments, rbacCtx))
8       .allEnabledRoles(roleService.getAllEnabledRoleSet(assignments))
9       .buildInCtx(rbacCtx);
10  }
```

Prüfen, ob Subjekt benötigte Rollen besitzt

In diesem Schritt werden die aktiven Rollen des Benutzers mit den per Annotation hinterlegten Bedingungen verglichen. Wenn alle Regeln erfüllt wurden, wird die Anfrage ausgeführt, wenn nicht wird mit HTTP 403 *Forbidden* die Anfrage abgelehnt.

Ressourcen auf aktuellen Kontext einschränken und zurückgeben

Wurde diese grobkörnige Zugriffskontrolle durchgeführt, kann der Entwickler noch weitere Sicherheitsabfragen implementieren. Ganz wesentlich ist die Einschränkung der zurückgegebenen Ressourcen auf den übergebenen Kontext. Die einzelnen Ressourcen, die auf der Seite dargestellt werden, sind ebenso mit einem RBAC-Kontext verbunden. Hat ein Benutzer einen Kontext gewählt, dürfen nur jene Ressourcen gelistet oder manipuliert werden, die dem aktuellen Kontext des Benutzers - oder einem untergeordneten Kontext - entsprechen. Hat ein Benutzer beispielsweise die Fakultät für Architektur gewählt, dürfen nur jene Prüfungen gelistet werden, die entweder der Fakultät für Architektur direkt oder deren Instituten zugeordnet sind. Ohne diese Regel würden zwar die einzelnen HTTP-Zugriffe abgesichert sein, aber nicht die zurück gelieferte Ergebnismenge oder die dahinter liegende Aktion. Aus diesem Grund ist diese Regel besonders wichtig bei der Umsetzung eines nach CORBAC gestalteten Systems.

6. Fazit und Ausblick

In dieser Arbeit wurde gezeigt, wie die Migration eines webbasierten und in PL/SQL entwickelten etablierten Campus-Management-Systems (CMS), namens CAMPUSonline, auf einen neuen Technologie-Stack erfolgen kann. Durch die Analyse und den Einsatz von bewährten Vorgehensmodellen und Architekturstilen konnte die besondere Herausforderung, ein seit über 20 Jahren wachsendes System mit enormer Funktionsvielfalt zu migrieren, gemeistert werden. Es wurden ein technisches Modell entwickelt, welches die Grundlage dafür schafft, dass während der mehrjährigen Migrationszeit die Gesamtfunktionalität von CAMPUSonline erhalten bleibt. Die zwei koexistierenden technischen Subsysteme, CAMPUSonline EE und CAMPUSonline PL/SQL, erscheinen dabei dem Benutzer, dank flexibler einsetzbarer Integrationstechnologien, als ein einziges Gesamtsystem, sodass die Benutzererfahrung während der Migration nicht gefährdet wird.

Ausgehend von einer abstrakten Produktvision wurde eine flexible Informationsarchitektur erarbeitet, um anschließend auf Basis dieser einen passenden Architekturstil namens Ressourcenorientierter Single-Page-Architekturstil (ROSPA) basierend auf 25 Grundsätzen abzuleiten. Bei der Entwicklung des Architekturstils ROSPA wurde der von Fielding und Taylor (2000, S. 13) vorgestellte „System Needs Style“ gewählt. Dabei werden aufgrund der Anforderungen eines konkreten Systems Schritt für Schritt Grundsätze formuliert, welche den Architekturstil und somit die grundlegenden Möglichkeiten und Eigenschaften, aber auch die Einschränkungen des Systems nachhaltig prägen. Die Abhängigkeit der Grundsätze untereinander und die Ziele, die durch einen Grundsatz gefördert oder potentiell auch gefährdet werden, wurden analysiert und jeweils in einem tabellarischen Steckbrief je Grundsatz prägnant dokumentiert. Durch die Formulierung möglichst technologie neutraler Grundsätze können die Ideen des Systems leicht vermittelt werden und bei der Einführung neuer Grundsätze die Konformität dieser einfach

geprüft werden. Der Architekturstil ROSPA kombiniert Eigenschaften einer ressourcenorientierten Architektur (ROA) und mit den Merkmalen einer Single-Page-Webanwendung (SPA). Ebenso wurden alternative Ansätze wie GraphQL und ROCA beleuchtet und dem gewählten Architekturstil gegenüber gestellt. Einige Ansätze konnten bei der Analyse der konkurrierenden Stile übernommen und erfolgreich in ROSPA integriert werden.

Im folgenden Text werden die Ziele der Produktvision dem gewählten Architekturstil ROSPA zusammenfassend gegenüber gestellt.

Dem Ziel der *zukunftssicheren Architektur* die mindesten 15 Jahre besteht wurde vor allem durch eine saubere Trennung in der Schichtenarchitektur, durch die Modularisierung des Gesamtsystems und durch den Architekturstil Representational State Transfer (REST) nachgekommen. Fielding (2008b) der in seiner Dissertation diesen Architekturstil begründete, sagt selbst „REST is software engineering on the scale of decades“. Gerade in Zeiten in denen sich Technologien für Benutzeroberflächen radikal verändern, ist es wichtig, dass diese einfach abgelöst bzw. ausgetauscht werden können. Im Sinne einer Schichtenarchitektur, welche ein Kernprinzip von REST darstellt, wurde eine saubere Trennung zwischen Client, Server und Datenbank vorgenommen. Der Client wurde als SPA mit Angular umgesetzt, wohingegen der Server auf den weit verbreiteten Standard Java EE aufbaut. Die Kommunikation zwischen Client und Server erfolgt über RESTful-Webservices. Des Weiteren sieht REST eine transparente und einheitliche Schnittstellentechnologie vor, welche eine automatisierte Interpretation der ausgelieferten Ressourcen durch einen intelligenten Client zulässt. Nur dadurch kann gewährleistet werden, dass sowohl die Geschäftslogik als auch die Prozesslogik, serverseitig umgesetzt werden und nicht beim Austausch des Clients verloren gehen. Aus diesem Grund wurde das coData-Protokoll, als das grundlegende über alle Bereiche von CAMPUSonline hinweg einheitliche Austauschformat entwickelt und etabliert. Das coData-Protokoll liefert neben den eigentlichen Ressourcen-Repräsentationen in XML bzw. JSON zusätzlich noch Links mit aus, welche die möglichen nächsten Prozessschritte der aktuellen Ressource beschreiben. Im Sinne von Hypermedia as the Engine of Application State (HATEOAS), werden die ausgelieferten Links über Angular dem Benutzer zugänglich gemacht, und steuern somit den Prozess am Client. Die grundlegende Eigenschaft eines zustandslosen Servers, ermöglicht eine horizontale Skalierung der Applikationsserver, so-

dass auch bei wachsenden Benutzerzahlen die Architektur nahtlos skaliert. Gerade für sehr große Universitäten mit mehr als 30.000 Studierenden oder auch interuniversitäre Verbünde sind so keine Limitierungen gesetzt.

In Bezug auf die *ausgereifte Schnittstellentechnologie* baut CAMPUSonline EE auf eine ressourcenorientierte Architektur (ROA) auf, welche streng zwischen den Datenressourcen und der Darstellung trennt. Alle Datenressourcen von CAMPUSonline EE werden über RESTful-Webservices entsprechend dem systemweit einheitlichen coData-Protokoll zugänglich gemacht. RESTful-Webservices haben sich aufgrund ihrer Leichtgewichtigkeit, ihrer Transparenz und ihrer Skalierungsfähigkeit zur Standardtechnologie bei der Anbindung von externen Systemen entwickelt. Geschützt werden diese RESTful-Webservices mittels des OAuth2-Standards, welche eine sehr feingranulare Zugriffssteuerung für extern zugreifende Clients gewährleistet. Neben dem neuen Technologie-Stack, der die Erstellung von RESTful-Webservices auf der Basis von Java EE ermöglicht, wurde mit Dataservices eine Möglichkeit geschaffen, SQL- und PL/SQL-RESTful-Webservices auf sehr einfache Art und Weise zu implementieren.

Zur *Integration* des Altsystems wurde die von Bisbal u. a. (1997) für schwer zerlegbare Systeme empfohlen Chicken-Little-Migrationsstrategie gewählt. Diese sieht drei Arten von Gateways vor, welche dazu dienen das neue System und das Altsystem auf unterschiedlichen technologischen Ebenen zu verbinden. Ausgehend von diesen Gateways wurden drei grundlegende Integrationstechnologien für die verschiedenen Ebenen der Schichtenarchitektur von CAMPUSonline EE implementiert, die je nach Bedarf und auch in einer Mischform eingesetzt werden können.

- Im Zuge der Migration auf CAMPUSonline EE soll das bestehende Datenmodell von CAMPUSonline PL/SQL grundlegend bereinigt, modularisiert, die verwendeten Terminologie vereinheitlicht und in die englische Sprache übersetzt werden. Dazu wird schrittweise ein idealisiertes Datenmodell entwickelt, welches entsprechend der Bereiche von CAMPUSonline modularisiert und in den neuen Bereichs-Datenbankschemen abgelegt wird. Noch nicht migrierte Tabellen, werden mittels Datenbankviews gekapselt und dabei bereits die neue einheitliche, englische Terminologie verwendet.

- Geschäftslogik von CAMPUSonline PL/SQL kann in CAMPUSonline EE dank der im Projekt umgesetzten Dataservice-Technologie erfolgreich aufgerufen bzw. ausgeführt werden. Dataservices sind eine auf PL/SQL basierende Technologie die es auf einfache Art ermöglicht bestehende Geschäftslogik über RESTful-Webservices lose gekoppelt über das HTTP-Protokoll zugänglich zu machen. RESTful-Webservices auf Basis von PL/SQL sind aufgrund des systemweit einheitlichen Schnittstellenprotokolls coData nicht von jenen RESTful-Webservices, die mittels CAMPUSonline EE Technologie entwickelt wurden, zu unterscheiden.
- Auf der Ebene der Benutzeroberflächen können bestehende Applikationen von CAMPUSonline PL/SQL direkt vom neuen CAMPUSonline Desktop-Menü aus gestartet werden. Die Applikationen werden dabei mittels eines generische Adapters, der in Form eines Iframe implementiert wurde, in das CAMPUSonline EE Desktop-Menü eingeklinkt. Sie wurden von der Darstellung in Richtung eines neuen Produktdesigns angepasst. Die Navigation der Applikation und die Seitentitel, werden entsprechend dem neuen Navigationskonzept dargestellt.

Das Ziel ein *hochqualitatives Produkt* zu erzeugen, wurde in dieser Arbeit nur gestreift und vor allem als unterstützendes Argument bei der Bewertung einzelner Grundsätze herangezogen. Die Sicherstellung einer geringen Fehlerrate durch automatisierte Tests auf den unterschiedlichen Ebenen, durch Unittests, Integrationstests und Systemtests, wurde parallel zu dieser Arbeit entwickelt. Auch die Aufzeichnung und Auswertung von Qualitätsmetriken und die Automatisierung der Entwicklungsstraße im Sinne von Continuous Integration bis hin zu Continuous Delivery wurden im Rahmen des Aufbaus von CAMPUSonline EE umgesetzt, in dieser Arbeit aber nicht näher erwähnt. Die sich aus den neu eingeführten Praktiken ergebenden Unterschiede in Bezug auf die zuvor praktizierte Vorgehensweise bei CAMPUSonline PL/SQL, wären es Wert in einer eigenen wissenschaftlichen Arbeit beleuchtet zu werden.

Im Bereich der *Usability* wurde die Informationsarchitektur ausgearbeitet und das Navigationskonzept, welches sechs Menüarten umfasst, ausführlich dargestellt. Die Informationsarchitektur lässt sich aufgrund des Konzepts der Zugänge und der losen Kopplung von Seiten und deren Menüs beson-

ders gut an die Anforderungen unterschiedlicher Bildungseinrichtungen aber auch an individuelle Benutzerbedürfnisse, anpassen. Die Benutzeroberfläche passt sich an die Auflösung der entsprechenden Endgeräte flexibel an und fühlt sich, dank der Möglichkeiten einer SPA Technologie, den aktuellen Applikationszustand im Speicher des Browsers zu halten, auch deutlich schneller an. Der oft am Altsystem kritisierte *Wald aus Fenstern* wird durch den Einsatz einer SPA schlussendlich Einhalt geboten. In dieser Arbeit wurden die Auswirkungen und Überschneidungen des gewählten Architekturstils mit dem Bereich der Usability eingehend beleuchtet. Weiterführende Usability-Themen, wie die Ausarbeitung des Usability-Pattern-Katalogs als Leitfaden für Entwickler, die ständige Betreuung der Entwickler durch Usability-Experten und die etablierte Praxis der Durchführung von Usability-Studien wurden allerdings in einem parallelen Projekt abgehandelt.

Die für CAMPUSonline EE gewählte Architektur im Punkt der *Informationssicherheit* der in CAMPUSonline PL/SQL gewählten Architektur überlegen. Durch die Verwendung einer auf ausschließlich statischen Ressourcen basierenden SPA können Cross-Site-Scripting-Attacken, dank der in Angular eingebauten Mechanismen, sehr effektiv unterbunden werden. Die Verwendung von OAuth2 und RESTful-Webservices verhindern ihrerseits wiederum Cross-Site-Request-Forgery-Attacken, da die aktuelle Benutzersitzung nun neben Cookies zusätzlich auf OAuth2-Zugriffstokens setzt, die über den HTTP-Authorization-Header übermittelt werden. Auch die Gefahr von SQL-Injection wird durch die konsequente Verwendung der Java Persistence API sehr erfolgreich minimiert. Abschließend wurde das Konzept der kontextuellen rollenbasierten Zugriffskontrolle (CORBAC) im Detail vorgestellt. CORBAC erlaubt aufgrund einer kontextabhängigen Rollenzuordnung eine dynamische Einschränkung der aktuellen Rolle des Benutzers, abhängig von dessen Kontext. Durch den Einsatz von CORBAC in Kombination mit ABAC konnte eine Explosion der Rollen verhindert werden und die Transparenz bei der Verwaltung von Benutzerrechten wesentlich erhöht werden. Bezüglich Verfügbarkeit, kann durch die gewählte Architektur mit einer sehr guten horizontalen Skalierbarkeit gepunktet werden. Die Datenbank, die oft ein Nadelöhr in Hochlastsituationen darstellte, wird in der neuen Architektur zunehmend entlastet, da sie nur mehr als Datenlieferant und nicht mehr zur Ausführung der Geschäftslogik genutzt wird. Die Entlastung

der Datenbank wird allerdings erst mit der zunehmenden Verschiebung der Geschäftslogik in Richtung Java EE tatsächlich spürbar werden.

Abschließend sei noch erwähnt, dass neben den technischen Herausforderungen, die ein Projekt dieser Größenordnung mit sich bringen, auch sehr viele begleitende organisatorische Aufgaben hinzu kommen. Sei es die interne Schulung und Weiterentwicklung der Mitarbeiter oder auch der Abgleich der fachlichen und technischen Entscheidungen unter Einbeziehung der Kooperationspartner. Auch das Change Management der einzelnen Benutzergruppen beim Ausrollen der neuen Benutzeroberflächen stellt eine große Herausforderung dar, die CAMPUSonline unter anderem dank der Flexibilität und der Nachhaltigkeit des gewählten Architekturstils mit Sicherheit hervorragend meistern wird.

Appendix

Anhang A.

Abkürzungen von technischen Zielen und Grundsätzen

A.1. Technische Ziele

Z$_{\text{ARCHITEKTUR}}$ Zukunftssichere Architektur

Z$_{\text{INTERFACE}}$ Ausgereifte Schnittstellentechnologie

Z$_{\text{INTEGRATION}}$ Anbindung CAMPUSonline PL/SQL

Z$_{\text{USABILITY}}$ Ausgezeichnete Usability

Z$_{\text{QUALITÄT}}$ Hohe Qualität

Z$_{\text{SICHERHEIT}}$ Informationssicherheit

A.2. REST

R$_{\text{CLIENT-SERVER}}$ Client-Server

R$_{\text{ZUSTANDSLOS}}$ Zustandslosigkeit

R$_{\text{CACHING}}$ Caching

R$_{\text{UNIFORM}}$ Einheitliche Schnittstelle

R$_{\text{RESOURCE}}$ Adressierbarkeit von Ressourcen

R$_{\text{REPRESENT}}$ Repräsentationen zur Veränderung von Ressourcen

R$_{\text{BESCHREIBEND}}$ Selbstbeschreibende Nachrichten

R$_{\text{HATEOAS}}$ Hypermedia as the Engine of Application State

R$_{\text{SCHICHTEN}}$ Mehrschichtige Systeme

R$_{\text{ON-DEMAND}}$ Code on Demand

A.3. Chicken Little

CL$_{\text{DB}}$ Datenbank-Gateway

CL$_{\text{APP}}$ Applikations-Gateway

CL$_{\text{IS}}$ Informationssystem-Gateway

A.4. ROCA

#$_{\text{REST}}$ REST-Prinzipien

#$_{\text{APPLICATION-LOGIC}}$ Applikationslogik am Server

#$_{\text{HTTP}}$ Kommunikation über HTTP

#$_{\text{LINK}}$ Teilbare URL

#$_{\text{NON-BROWSER}}$ Logik ohne Browser nutzbar

#$_{\text{SHOULD-FORMATS}}$ Zusätzliche Formate

#$_{\text{AUTH}}$ Authentifizierung

#$_{\text{COOKIES}}$ Cookies

#$_{\text{SESSION}}$ Session

#$_{\text{BROWSER-CONTROLS}}$ Browser Steuerung

#$_{\text{POSH}}$ Semantisches HTML

#ACCESSIBILITY Accessibility

#IDIOMATIC-CSS Idiomatisches CSS

#UNOBTRUSIVE-JAVASCRIPT Unaufdringliches JavaScript

#NO-DUPLICATION Keine Duplizierungen

#KNOW-STRUCTURE Unbekannte Struktur

#STATIC-ASSETS Statische Ressourcen

#HISTORYAPI History API

A.5. Informationsarchitektur

IA$_{\text{FENSTER}}$ Einzelfenster

IA$_{\text{KONTEXT}}$ Kontext

IA$_{\text{MENU}}$ Entkopplung von Menüs und Seiten

IA$_{\text{AKTIONEN}}$ Navigation vs. Aktion

A.6. Weitere

SYS$_{\text{MODULAR}}$ Modularer Aufbau

SEC$_{\text{RBAC}}$ Role-Based Access Control

SEC$_{\text{ABAC}}$ Attribute-Based Access Control

A.7. ROSPA

§MODUL Modularisierung

§LAYER Mehrschichtiges System

§RESOURCES Ressourcen

§REPRESENT Repräsentationen

§ZUGRIFF Einheitliche Zugriffsmethoden

§NO-STATE Zustandsloser Server

§SERVER-LOGIC Applikationslogik am Server

§PROTOCOL Einheitliche Schnittstellen

§FORMATS Zusätzliche Formate

§AUTHN Authentifizierung

§COOKIE Cookies

§LEGACY Kommunikation mit dem Altsystem

§DATA-MODEL Einheitliches Datenmodell

§AUTHZ Zugriffskontrolle

§CACHE Caching

§NON-BROWSER Logik ohne Browser nutzbar

§SPA Single-Page-Application

§BROWSER Standard Browser Verhalten

§HATEOAS Links steuern das Client-Verhalten

§CONTEXT Wahl des Kontextes

§MENU Trennung von Menüs und Seiten

§STYLE Trennung von Daten und Darstellung

§ACCESSIBILITY Zugänglichkeit

§**LAZY-LOAD** Lazy-Loading

§**CLIENT-TYPES** Typisierter Client

§**INTEGRATION** Integration des Altsystems

Anhang B.

Abkürzungen der Module in CAMPUSonline

CORE Kernsegment

SLC Student Life Cycle

BRM Basis und Ressourcen Management

AM Bewerbungs- und Zulassungs-Management

SM Studierenden- und Studien-Management

TM Lehre

CM Curricula Management

XM Prüfungen

Literatur

Alt, Rainer und Gunnar Auth (2010). „Campus Management System“. In: *Business & Information Systems Engineering* 2.3, S. 187–190. ISSN: 1867-0202. DOI: 10.1007/s12599-010-0105-9. URL: https://doi.org/10.1007/s12599-010-0105-9 (siehe S. 1).

Amundsen, Mike (2011). *REST: from research to practice*. Springer Science & Business Media, S. 93–116 (siehe S. 69).

Amundsen, Mike (2013). *Collection+JSON - Document Format*. URL: http://amundsen.com/media-types/collection/format/ (besucht am 07. 10. 2017) (siehe S. 50).

Andrews, Keith (2009). *Information Architecture and Web Usability* (siehe S. 13).

Barth, A. (2017). *HTTP State Management Mechanism*. URL: https://tools.ietf.org/html/rfc6265 (besucht am 09. 10. 2017) (siehe S. 58).

Baschuk, Alex (2016). *Redux · An Introduction*. URL: https://www.smashingmagazine.com/2016/06/an-introduction-to-redux/ (besucht am 11. 10. 2017) (siehe S. 39).

Berners-Lee, Tim (1996). *Roget's Interactive Thesaurus*. URL: https://www.w3.org/DesignIssues/MatrixURIs.html (besucht am 07. 10. 2017) (siehe S. 80).

Bien, Adam (2009). *Real World Java EE Patterns Rethinking Best Practices*. First Edition. press.adam-bien.com, S. 55–111. ISBN: 9781300149316 (siehe S. 93).

Bierman, Gavin, Martin Abadi und Mads Torgersen (2014). „Understanding typescript“. In: *European Conference on Object-Oriented Programming*. Springer, S. 257–281 (siehe S. 77).

Bisbal, Jesús u. a. (1997). „A survey of research into legacy system migration“. In: *Technique report* (siehe S. 29, 36, 37, 115).

Bradley, John, Nat Sakimura und Michael Jones (2015). *JSON Web Token (JWT)*. URL: https://tools.ietf.org/html/rfc7519 (besucht am 09. 10. 2017) (siehe S. 56).

Bußmann, Hadumod und Hartmut Lauffer (2002). *Lexikon der Sprachwissenschaft*. 3 Auflage. Kröner Stuttgart: ISBN: 3-520-45203-0 (siehe S. 26).

Conway, Melvin E. (1968). „How Do Committees Invent?“ In: *Datamation*, S. 28–31 (siehe S. 18, 90).

Eizinger, Thomas (2017). „API Design in Distributed Systems: A Comparison between GraphQL and REST“. Magisterarb. (siehe S. 40).

Fielding, Roy T. (2008a). „A little REST and Relaxation“. In: *ApacheCon*. URL: https://de.slideshare.net/royfielding/a-little-rest-and-relaxation (besucht am 25. 09. 2017) (siehe S. 67).

Fielding, Roy T. (2008b). *REST APIs must be hypertext-driven*. URL: http://roy.gbiv.com/untangled/2008/rest-apis-must-be-hypertext-driven (besucht am 25. 09. 2017) (siehe S. 31, 44, 49, 67, 114).

Fielding, Roy T. (2013). „Keynote: Scrambled Eggs“. In: *Evolve*. URL: https://de.slideshare.net/royfielding/evolve13-keynote-scrambled-eggs (besucht am 25. 09. 2017) (siehe S. 68).

Fielding, Roy T. (2015). *REST in AEM* (siehe S. 49).

Fielding, Roy T. und Richard N. Taylor (2000). *Architectural styles and the design of network-based software architectures*. University of California, Irvine Doctoral dissertation (siehe S. 29, 31, 52, 113).

Fowler, Martin (2002). *Patterns of Enterprise Application Architecture*. Addison-Wesley-Longman. ISBN: 0321127420 (siehe S. 95).

Fowler, Martin und Jim Highsmith (2001). „The agile manifesto“. In: *Software Development* 9.8, S. 28–35 (siehe S. 7).

Franqueira, Virginia und Roel Wieringa (2012). „Role-based access control in retrospect“. In: *Computer* 45.6, S. 81–88 (siehe S. 98).

Friedman, Vitaly (2008). *Should Links Open In New Windows?* URL: https://www.smashingmagazine.com/2008/07/should-links-open-in-new-windows/ (besucht am 22. 09. 2017) (siehe S. 14).

Garrett, Jesse James (2010). *Elements of user experience, the: user-centered design for the web and beyond*. Pearson Education (siehe S. 11, 13).

Geene, Mark, Ross Garrett und Kin Lane (2017). *The State of API Integration Report* (siehe S. 3).

Geoffrey, James (1987). *The tao of programming*. InfoBooks (siehe S. 19, 24).

Giessler, Pascal u. a. (2015). „Best Practices for the Design of RESTful Web Services“. In: *International Conferences of Software Advances (ICSEA)*, S. 392–397 (siehe S. 32).

Grigorik, Ilya (2017). *HTTP-Caching*. URL: https://developers.google.com/web/fundamentals/performance/optimizing-content-efficiency/http-caching?hl=de (besucht am 09. 10. 2017) (siehe S. 61).

Hardt, Dick (2012). *The OAuth 2.0 Authorization Framework*. RFC 6749. Fremont, CA, USA: RFC Editor. URL: http://www.rfc-editor.org/rfc/rfc6749.txt (siehe S. 55).

Hartig, Olaf und Jorge Pérez (2017). „An Initial Analysis of Facebook's GraphQL Language". In: *AMW 2017 11th Alberto Mendelzon International Workshop on Foundations of Data Management and the Web, Montevideo, Uruguay, June 7-9, 2017*. Bd. 1912. Juan Reutter, Divesh Srivastava (siehe S. 40).

Hoppe, Falk, Till Schulte-Coerne und Stefan Tilkov (2012). „ROCA: Resource-oriented Client Architecture". In: *OBJEKTspektrum*. URL: https://www.sigs-datacom.de/uploads/tx_dmjournals/hoppe_schulte-coerner_tilkov_OS_Architekturen_12_hgz8.pdf (siehe S. 38).

Hu, Vincent C. u. a. (2013). „Guide to attribute based access control (ABAC) definition and considerations (draft)". In: *NIST special publication* 800.162 (siehe S. 33, 99).

Instone, Keith (2002). „Location, path and attribute breadcrumbs". In: *3rd Annual Information Architecture Summit*, S. 15–17 (siehe S. 25).

Jenkov, Jakob (2015). *HTML5 History API*. URL: http://tutorials.jenkov.com/html5/history-api.html (besucht am 09. 10. 2017) (siehe S. 66).

Kalam, A. A. E. u. a. (2003). „Organization based access control". In: *Proceedings POLICY 2003. IEEE 4th International Workshop on Policies for Distributed Systems and Networks*, S. 120–131. DOI: 10.1109/POLICY.2003.1206966 (siehe S. 103).

Kelly, Mike (2016). *HAL - Hypertext Application Language*. URL: https://tools.ietf.org/html/draft-kelly-json-hal-08 (besucht am 07. 10. 2017) (siehe S. 48).

Kennard, Richard und John Leaney (2010). „Towards a general purpose architecture for UI generation". In: *Journal of Systems and Software* 83.10, S. 1896–1906. ISSN: 0164-1212. DOI: https://doi.org/10.1016/j.jss.2010.05.079. URL: http://www.sciencedirect.com/science/article/pii/S0164121210001597 (siehe S. 75).

Kirchhoff, Marc und Kurt Geihs (2013). „Semantic Description of OData Services“. In: *Proceedings of the Fifth Workshop on Semantic Web Information Management*. SWIM '13. New York, New York: ACM, 2:1–2:8. ISBN: 978-1-4503-2194-5. DOI: 10.1145/2484712.2484714. URL: http://doi.acm.org/10.1145/2484712.2484714 (siehe S. 50).

Kuhn, D. R., E. J. Coyne und T. R. Weil (2010). „Adding Attributes to Role-Based Access Control“. In: *Computer* 43.6, S. 79–81. ISSN: 0018-9162. DOI: 10.1109/MC.2010.155 (siehe S. 99, 105, 106).

Lalou, Jonathan (2013). *Apache Maven Dependency Management*. Packt Publishing Ltd (siehe S. 83).

Lämmerhirt, Marcel, Marguerite Franssen und Christoph Becker (2013). „Reorganization and IT Implementation in Campus Management: The Project “PuL” at RWTH Aachen University“. In: *Next Generation of Information Technology in Educational Management*, S. 142–147. DOI: 10.1007/978-3-642-38411-0_13 (siehe S. 1).

Lanthaler, Markus und Christian Gütl (2012). „On Using JSON-LD to Create Evolvable RESTful Services“. In: *Proceedings of the Third International Workshop on RESTful Design*. WS-REST '12. Lyon, France: ACM, S. 25–32. ISBN: 978-1-4503-1190-8. DOI: 10.1145/2307819.2307827. URL: http://doi.acm.org/10.1145/2307819.2307827 (siehe S. 49).

Lipinski, K. (2017). *ITWissen.info*. URL: http://www.itwissen.info/Sitzung-session.html (siehe S. 52).

Luo, J. und S. Xu (2010). „Based on RBAC Security Design of University Management System“. In: *2010 International Conference on E-Business and E-Government*, S. 1779–1782. DOI: 10.1109/ICEE.2010.450 (siehe S. 97).

Microsystem, SUN (o.D.). *Java™ Authentication and Authorization Service* (siehe S. 55).

Mikowski, Michael S. und Josh C. Powell (2013). *Single Page Web Applications*. 1. Aufl. Manning Publications. ISBN: 978-1617290756 (siehe S. 33).

Motto, Todd (2017). *Lazy loading: code splitting NgModules with Webpack*. URL: https://toddmotto.com/lazy-loading-angular-code-splitting-webpack (besucht am 11. 10. 2017) (siehe S. 77).

Nelson, Theodor H (1974). *Dream Machines: New Freedoms through Computer Screens. A Minority Report*. Nelson : [available] from Hugo's Book Service (siehe S. 67).

Nielsen, Jakob (1999a). *Designing web usability: The practice of simplicity*. New Riders Publishing (siehe S. 11).

Nielsen, Jakob (1999b). „Top 10 mistakes in web design“. In: *Nielsen Norman Group: Evidence-Based User Experience Research, Training, and Consulting* (siehe S. 14).

Norm, DIN (o.D.). „EN ISO 9241-110 2006“. In: *Ergonomie der Mensch-System-Interaktion-Teil* 110, S. 9241–110 (siehe S. 14, 75).

Norman, Donald A und Stephen W Draper (1986). „User centered system design“. In: *Hillsdale, NJ*, S. 1–2 (siehe S. 11).

Nottingham, M. (2005). *The Atom Syndication Format*. URL: https://tools.ietf.org/html/rfc4287 (besucht am 07. 10. 2017) (siehe S. 48).

Nottingham, Mark, Julian Reschke und Jan Algermissen (2017). *Link Relations*. URL: https://www.iana.org/assignments/link-relations/link-relations.xhtml (besucht am 14. 10. 2017) (siehe S. 69).

Parnas, David Lorge (1972). „On the criteria to be used in decomposing systems into modules“. In: *Communications of the ACM* 15.12, S. 1053–1058 (siehe S. 33).

Pezoa, Felipe u. a. (2016). „Foundations of JSON Schema“. In: *Proceedings of the 25th International Conference on World Wide Web*. WWW '16. Montreal, Quebec, Canada: International World Wide Web Conferences Steering Committee, S. 263–273. ISBN: 978-1-4503-4143-1. DOI: 10.1145/2872427.2883029. URL: https://doi.org/10.1145/2872427.2883029 (siehe S. 49).

Precht, Pascal (2016). *PROTECTING ROUTES USING GUARDS IN ANGULAR*. URL: https://blog.thoughtram.io/angular/2016/07/18/guards-in-angular-2.html (besucht am 10. 10. 2017) (siehe S. 72).

Richardson, Leonard und Sam Ruby (2007). *Web-services mit REST*. O'Reilly Germany (siehe S. 31, 46, 52).

Rosenfeld, Louis und Peter Morville (2006). *Information architecture for the world wide web*. Third Edition. O'Reilly Media, Inc. ISBN: 978-0-596-52734-1 (siehe S. 13).

Sandhu, R. S. u. a. (1996). „Role-based access control models“. In: *Computer* 29.2, S. 38–47. ISSN: 0018-9162. DOI: 10.1109/2.485845 (siehe S. 33, 96).

Santamaria, Jose Maria Arranz (2015). *The Single Page Interface Manifesto*. URL: http://itsnat.sourceforge.net/php/spim/spi_manifesto_en.php (besucht am 09. 10. 2017) (siehe S. 66).

Scholz, Florian und Achmin Bode (2015). *Lazy loading: code splitting NgModules with Webpack*. URL: https://developer.mozilla.org/de/docs/Web/API/MutationObserver (besucht am 11. 10. 2017) (siehe S. 81).

Schulte-Coerne, Till u. a. (2012). *A collection of simple recommendations for decent Web application frontends*. URL: http://roca-style.org/index.html (besucht am 11. 10. 2017) (siehe S. 38).

Severance, C. (2015). „Roy T. Fielding: Understanding the REST Style“. In: *Computer* 48.6, S. 7–9. ISSN: 0018-9162. DOI: 10.1109/MC.2015.170 (siehe S. 31, 54).

Starke, Gernot (2014). *Effektive Softwarearchitekturen*. 6., überarbeitete Auflage. Carl Hanser Verlag GmbH Co KG. ISBN: 978-3-446-43614-5 (siehe S. 82).

Sthamer, Doro (2016). *Webwissen: User Journeys*. URL: http://www.netzstrategen.com/sagen/webwissen-user-journeys/ (besucht am 14. 10. 2017) (siehe S. 17).

Swiber, Kevin (2017). „Siren: a hypermedia specification for representing entities“. In: DOI: 10.5281/zenodo.556783 (siehe S. 48).

Tilkov, Stefan (2016). *Why I hate your Single Page App*. URL: https://medium.freecodecamp.org/why-i-hate-your-single-page-app-f08bb4ff9134 (besucht am 11. 10. 2017) (siehe S. 39).

Tilkov, Stefan u. a. (2015). *REST und HTTP: Entwicklung und Integration nach dem Architekturstil des Web*. 3. Aufl. dpunkt.verlag. ISBN: 3864901200 (siehe S. 32).

Vasilakis, Filippos (2017). *Introspected REST: An alternative to REST and GraphQL*. URL: https://github.com/vasilakisfil/Introspected-REST/commits/master (besucht am 27. 08. 2017) (siehe S. 32, 49, 74).

Webber, Jim, Savas Parastatidis und Ian Robinson (2010). *REST in practice: Hypermedia and systems architecture*. O'Reilly Media, Inc. (siehe S. 67).

Wertheimer, Max (1923). „Untersuchungen zur Lehre von der Gestalt. II“. In: *Psychologische forschung* 4.1, S. 301–350 (siehe S. 17, 20).

Wolf, Frank (2010). *Das lautlose Geheimnis guter Websites: Was ist eigentlich Informationsarchitektur?* URL: http://besser20.de/das-lautlose-geheimnis-guter-websites-was-ist-eigentlich-informationsarchitektu 804/ (besucht am 30. 09. 2017) (siehe S. 11).

Xu, X. u. a. (2008). „Resource-Oriented Architecture for Business Processes“. In: *2008 15th Asia-Pacific Software Engineering Conference*, S. 395–402. DOI: 10.1109/APSEC.2008.52 (siehe S. 3).

Zhang, Z., X. Zhang und R. Sandhu (2006). „ROBAC: Scalable Role and Organization Based Access Control Models“. In: *2006 International Conference on Collaborative Computing: Networking, Applications and Worksharing*, S. 1–9. DOI: 10.1109/COLCOM.2006.361879 (siehe S. 103).

Zheng, S., D. Jiang und Q. Liu (2009). „A Role and Activity Based Access Control Model for University Identity and Access Management System“. In: *2009 Fifth International Conference on Information Assurance and Security*. Bd. 2, S. 487–490. DOI: 10.1109/IAS.2009.43 (siehe S. 98).