Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Bachelorarbeit

Micha Severin

Anwendungsgestützte Analyse des JavaScript-Sprachkerns in Webbrowsern

Fakultät Technik und Informatik
Studiendepartment Informatik

Faculty of Engineering and Computer Science
Department of Computer Science

Micha Severin

Anwendungsgestützte Analyse des JavaScript-Sprachkerns in Webbrowsern

Bachelorarbeit eingereicht im Rahmen der Bachelorprüfung

im Studiengang Bachelor of Science Angewandte Informatik
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

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

Eingereicht am: 27. März 2015

Micha Severin

Thema der Arbeit
Anwendungsgestützte Analyse des
JavaScript-Sprachkerns in Webbrowsern

Stichworte
JavaScript, Sprachkern-Analyse, Anwendungsgestützte Tests, Objekt-Verfügbarkeit

Kurzzusammenfassung
Der JavaScript-Sprachkern in Webbrowsern ist durch die Version und Implementierung definiert. Eine Übersicht der Verfügbarkeit von Objekten, Ausdrücken, Operatoren, Statements, Deklarationen und Funktionen ist schwer zu erlangen. Eine benutzergesteuerte Anwendung zur Erstellung von Tests ist der zentrale Punkt dieser Arbeit. Tests werden in verschiedenen Umgebungen ausgeführt und liefern ihr Ergebnis zu der Anwendung zurück. Eine Auswertung dieser Ergebnisse führt zu einer Übersicht der Verfügbarkeit.
Micha Severin

Title of the paper
JavaScript Core Analysis in Web Browsers

Keywords
JavaScript, core analysis, application driven tests, object existence

Abstract
JavaScript core syntax in web browsers is defined by its version and implementation. It is difficult to gain an overall view of the existence of objects, expressions, operators, statements, declarations and functions. An application for creating tests driven by users is the main part of the thesis. These tests will be executed in different environments and the results are returned to the application. By evaluating the results an overview of the availability will be generated.

Inhaltsverzeichnis

1 Einleitung

Eine Programmiersprache wird geprägt von Objekten, Ausdrücken, Operatoren, Statements, Deklarationen sowie Funktionen und wächst in der Weiterentwicklung stetig an. Ist der Sprachkern in frühen Versionen meist noch überschaubar, so vergrößert er sich in späteren Versionen kontinuierlich. Bei der Entwicklung neuer Software kann der Entwickler nur Bezug auf den aktuellen Stand bzw. die aktuelle Version der Programmiersprache nehmen. Weiterhin ist für ihn eine detaillierte Dokumentation der Programmiersprache und somit ihrer Bestandteile wichtig. Diese Faktoren sind unabdingbar für den erfolgreichen Einsatz der Software.
Die konkrete Entwicklungsumgebung entspricht im Idealfall genau der späteren Einsatzumgebung. Trifft dies nicht zu oder es gibt mehr als eine Einsatzumgebung, dann steigt der spätere Testaufwand. Aus dieser Konstellation heraus können weitere Faktoren entstehen, die auf die Software Einfluss nehmen. Am Beispiel von Webbrowsern lässt sich dies veranschaulichen. Hinter jedem Webbrowser steht ein kommerzielles Unternehmen, eine Community oder auch einzelne Personen bzw. Entwickler. Deren Produkte sind häufig nicht ausschließlich auf nur ein Betriebssystem (z.B. Windows, Mac, Linux) ausgelegt.
In ihrer Gesamtheit haben alle Webbrowser ein Ziel: Informationen (Daten), die auf Servern hinterlegt werden, abzurufen und diese dem Benutzer zu präsentieren. Teile dieser Daten für Webseiten sind häufig in der Skriptsprache JavaScript geschrieben. Zusammenfassend spielen somit folgende Faktoren eine Rolle für den Einsatz von Skripten in JavaScript:

- Die Version der Programmiersprache
- Die Einsatzumgebungen der Software
- Die Dokumentation der Programmiersprache
- Der verwendete Webbrowser
- Das Betriebssystem

Diese Punkte sind ein zentraler Bestandteil für die Entwicklung von Scripten für Webseiten. Diese Skripte können zum Beispiel marginale dynamische Änderungen am Seitenbild bewirken, aber auch Teil komplexer Webanwendungen sein.

1.1 Motivation

Der Einsatz der Skriptsprache JavaScript in einem Webbrowser begann im März 1996. Der Webbrowser Netscape in seiner zweiten Version war der erste seiner Art. Fortan wurde die Sprache weiterentwickelt und fand weitere Implementierungen in weiteren Webbrowsern (z.B. JScript - Microsoft). Die entstandenen Versionen führten dazu, dass ein Objekt der Sprache von manchen Webbrowsern unterstützt wurde, von anderen hingegen nicht. Ferner wurden Objekte mit spezifischer Funktionalität der Hersteller ausgestattet.
Von 1996 bis heute hat sich die Anzahl der Geräte vervielfacht, die eine Darstellung von Webseiten mit einhergehender Unterstützung von JavaScript bieten. Auf der Grundlage dieser Tatsache ist die Übersicht über den Umfang des JavaScript-Sprachkerns in verschiedenen Webbrowsern schwer geworden.

1.2 Ziel der Arbeit

Diese Arbeit befasst sich mit der Erarbeitung und Entwicklung einer Anwendung, die in verschiedenen Webbrowser-Umgebungen automatisiert eine Übersicht über den JavaScript-Sprachkern generiert. Untersucht wird die Sprache JavaScript mit ihren verschiedenen Implementierungen. Die Analyse geschieht auf Basis von Tests, die in Webbrowsern zur Ausführung gebracht werden. Auf diese Weise werden Informationen gewonnenen, die die Basis für den Vergleich der Webbrowser-Umgebungen zueinander bilden. Die benutzergenerierten Tests resultieren schlussendlich nach Ausführung in einer Übersicht, die Aufschluss über den Erfolgs- oder Fehlerfall des ausgeführten Tests geben.

1.3 Abgrenzung

Die Skriptsprache JavaScript findet Verwendung in verschiedenen Bereichen der Programmierung. Sie wird im Layoutbereich der Printproduktion für Adobe InDesign, in der Back-End Programmierung bis hin zu Webbrowsern eingesetzt. Diesem Fakt geschuldet lässt sich das Einsatzgebiet nicht mehr eindeutig bestimmen. In Adobe InDesign existiert zum Beispiel kein klassisches Document Object Model (DOM), was wiederum ein wesentliches Charakteristikum der Sprache in Webbrowsern darstellt. Der Fokus dieser Arbeit liegt jedoch auf der Untersuchung von JavaScript in Webbrowsern. Die entwickelte Anwendung wird ausschließlich diesem Zweck dienen.

2 Grundlagen

Dieses Kapitel gibt einen Überblick darüber, welche Programmiersprachen und Technologien für die Entwicklung der Anwendung wichtig sind. Es wird auf die primär verwendete Skriptsprache JavaScript eingegangen. Die erste Betrachtung findet im historischen Kontext statt. Der Fokus liegt hierbei auf der Webentwicklung. Weiterhin wird die Plattform Node.js, die auf JavaScript basiert, analysiert. Das Konzept der Entwicklung ist hierbei primärer Bestandteil der Analyse. Zuletzt werden die zum Einsatz kommenden Frameworks vorgestellt. Das Ziel der zu entwickelnden Anwendung steht dafür im Vordergrund der Betrachtung.

2.1 JavaScript

JavaScript gehört zu der Klasse der Skriptsprachen. Diese Sprachen zeichnen sich unter anderem dadurch aus, dass die Skripte zur Laufzeit durch einen Interpreter ausgeführt werden und keiner Kompilation bedürfen. JavaScript ist dynamisch typisiert, objektorientiert, aber klassenlos. Es existiert zwar ein reserviertes Wort „class“ aber es liegt bislang keine Implementation dafür vor (Odell, 2014, S.9). Der Einsatz von Skriptsprachen ist unter anderem für Bereiche gedacht, in denen leichtgewichtige Aufgaben zu erledigen sind. Dieses Merkmal ist bei JavaScript in Webbrowsern beispielhaft. Zudem lässt sich eine Skriptsprache im Vergleich zu kompilierten Sprachen einfacher und schneller entwickeln (Scott, 2009, S.655). Das hat sich auch bei JavaScript gezeigt, das von Brendan Eich in nur zehn Tagen entwickelt worden sein soll (Rauschmayer, 2014, S.44). Der wohl verbreitetste Einsatz von JavaScript liegt clientseitig in Webbrowsern.

2.1.1 Entstehung

1995 wurde JavaScript von Brendan Eich entwickelt. Innerhalb dieses Jahres wurde der Name für die Sprache mehrfach geändert bis zu der endgültigen Benennung in JavaScript. Der Anstoß für die Entwicklung war die damalige, sich vor der Fertigstellung befindliche zweite Version des Webbrowsers Netscape. Kurz vor der Veröffentlichung empfanden die Entwickler, dass

dem Webbrowser noch eine Möglichkeit der dynamischen Manipulation von Seiteninhalten fehlte. Aus diesem Anlass wurde Brendan Eich mit der Entwicklung betraut.

2.1.2 ECMAScript

Ab der dritten JavaScript Version (1.3), die im Jahr 1998 veröffentlicht wurde, begann die private Organisation European Computer Manufacturers Association (ECMA)-International den Sprachkern von JavaScript zu standardisieren. Die Organisation wurde von führenden Browserherstellern mit dieser Aufgabe betraut. Der erste Entwurf wurde unter der Bezeichnung „ECMA 262“ ECMA (1997) veröffentlicht. Ziel der Standardisierung war es, weiteren Herstellern eine Möglichkeit zu bieten JavaScript in deren Produkten zu implementieren. Die Tabelle 2.1 gibt einen Überblick über die einzelnen ECMA-Entwürfe mit Veröffentlichungsdaten.

Edition	Veröffentlicht
1	1997, Juni
2	1998, August
3	1999, Dezember
4	abgebrochen
5	2009, Dezember
5.1	2011, Juni
6	in Arbeit
7	in Arbeit

Tabelle 2.1: Übersicht der ECMA-Editionen. Nach Zakas (2009, S.4)

2.1.3 Versionsgeschichte

Der Aufteilung der Versionsnummern folgt einem klaren Schema. Die Versionen werden in Haupt-, Neben-, Revisions- sowie Buildnummern untergliedert. Hauptversionen (Major) entstehen durch signifikante Änderungen, die nicht mehr einem Konzept der vorhergehenden Version folgen. Nebenversionen (Minor) sind geringfügigere Änderungen, die nur erweiternd oder auch ergänzend den Umfang ändern. Revisionsnummern (Patch) entsprechen einer Überarbeitung oder Korrektion. Die Buildnummer (Build) dient der Identifikation einzelner Kompilationen. Die Unterscheidung der Versionsnummern ist in der Abbildung 2.1 visualisiert.

Die JavaScript-Versionen sind in der Tabelle 2.2 chronologisch aufgeführt. Die verbreitetsten Webbrowser mit ihren ersten oder entscheidenden Veröffentlichungen sind außerdem ersichtlich. Zu beachten ist, dass die Version 1.4.0 eine Fokussierung von Netscape war JavaScript serverseitig einzusetzen. Dieser Ansatz scheiterte. Erst durch die Entwicklung von Node.js im

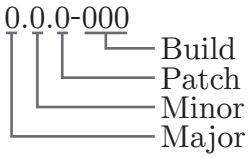

Abbildung 2.1: Schema der Versionsnummer. Angelehnt an A.A.Puntambekar (2010)

Jahr 2009 durch Ryan Dahl wurde dieser Ansatz wieder aufgegriffen.
Die vierte Edition von ECMA-262 wurde nicht fertiggestellt. Grund hierfür waren Uneinigkeiten der Hersteller in Bezug auf die Komplexität des Entwurfs. Teile des Entwurfs finden aber Verwendung in der sechsten Edition, die sich in der Erstellung befindet.

Version	Veröffentlicht	ECMA-262	Browser Beispiele
1.0.0	1996, März		Netscape Navigator 2.0
1.1.0	1996, August		
1.2.0	1997, Juni		
1.3.0	1998, Oktober	1, 2 Edition	Mozilla Firefox 1.0
1.4.0	1999, Januar		Netscape Server 3.6
1.5.0	2000, November	3 Edition	Microsoft Internet Explorer 6.0
1.6.0	2005, November		Apple Safari 3.0
1.7.0	2006, Oktober		Google Chrome 1.0
1.8.0	2008, Juni		
1.8.1	2009, Juni		
1.8.5	2010, Dezember	5 Edition	Yandex Webbrowser 1.0

Tabelle 2.2: Übersicht der JavaScript-Versionen (Resig, 2014)

2.1.4 Architektur

Dieser Abschnitt gibt einen kleinen Überblick über die Architektur von JavaScript. Hierbei werden die Bereiche der Objekte, Ausdrücke, Operatoren, Statements, Deklarationen und Funktionen beleuchtet. Es wird aber keine allumfassende Sammlung aller Aspekte der Sprache angestrebt.

2.1.4.1 Objektorientierung

Den Paradigmen der Objektorientierung (Objektorientierte Programmierung (OOP)) wird die Sprache JavaScript gerecht. Dem Verzicht auf Klassen steht das Konzept des Prototypings ge-

genüber. Diese Art der Objektorientierung wird auch als prototypenbasierte Objektorientierung bezeichnet (Stefanov und Sharma, 2013).

2.1.4.2 Prototyping

Die auf Prototypen basierende Programmierung ist zentraler Bestandteil von JavaScript. Jedes neue Objekt (B) basiert auf einem zweiten Objekt (A), dem Prototypen. Von dem Objekt A werden alle Eigenschaften an das Objekt B vererbt. Der Zugriff auf den Prototyp von B ist über die Eigenschaft „.prototype“ möglich (Flanagan, 2011, S.118).

2.1.4.3 Typisierung

Die Skriptsprache JavaScript ist schwach typisiert (weak typing). Diese Eigenschaft macht sich bei der Behandlung zweier Datentypen, die durch einen Operator verbunden sind, bemerkbar. Wenn ein String mit dem Plus-Operator mit einer Nummer verbunden wird, dann wird die Nummer als String an den vorherigen String konkateniert. Die dynamische Typisierung ist ein weiteres Charakteristikum der Sprache. Hierbei findet die Variablenprüfung erst zur Laufzeit des Programms statt. Weiterhin wird die Sprache dem Duck-Konzept zugeordnet, in dem nicht die Klasse, sondern die Methoden und Attribute eines Objekts dessen Typ bestimmen.

2.1.4.4 Reservierte Worte

In JavaScript gibt es eine begrenzte Anzahl an reservierten Worten, die beispielsweise nicht für die Verwendung von Variablennamen verwendet werden dürfen. In der folgenden Gliederung sind einige aktuell in Verwendung befindliche Wörter aufgeführt.

Operatoren delete, new, instanceof, this, typeof, void

Statements break, case, catch, continue, default, do, else, export, finally, for, function, if, import, in, return, switch, throw, try, var, while

2.1.4.5 Datentypen

Es gibt fünf verschiedene Typen, die in JavaScript unterschieden werden. Alle anderen Elemente werden zu den Objekten gezählt und bilden keinen eigenen Datentyp. Diese Objekte sind veränderbar (Crockford, 2008, S.20). Die Datentypen sind:

- Numbers

- Strings
- Booleans
- Null
- Undefined

Nummern, Strings, und Booleans sind unveränderbar, bieten nach außen aber Methoden an. Als vollwertige Objekte werden folgende Elemente gezählt:

- Array
- Funktionen
- Reguläre Ausdrücke
- Objekte

2.1.4.6 Das Globale Objekt

Das Globale Objekt ist das Fundament in jeder clientseitigen JavaScript Programmumgebung. Es entspricht einem regulären Objekt, erweitert aber zugleich den globalen Geltungsbereich um neue Objekte. Diese neu erschaffenen Objekte entsprechen zugleich den Eigenschaften des Globalen Objekts. Die Eigenschaften, Funktionen und Objekte des Globalen Objekts werden wie folgt gegliedert:

Eigenschaften	undefined, Infinity, NaN
Funktionen	isNaN(), parseInt(), eval()
Konstruktor Funktionen	Date(), RegExp(), String(), Object(), Array()
Objekte	Math, JSON

Tabelle 2.3: Erzeugt für das Globale Objekt (Flanagan, 2011, S.42).

2.2 Node.js Plattform

Die Plattform Node.js ist eine serverseitige Implementation der Skriptsprache JavaScript. Kern der Plattform ist die „Google V8 JavaScript Engine“. Ein perfektes Einsatzgebiet sind Echtzeit-Anwendungen, die datenintensive Operationen ausführen (Rai, 2013, S.28). Weiterhin sind Aspekte wie „event-driven“ und „non-blocking I/O model“ grundlegende Bestandteile der Plattform (Node.js, 2014).

2.2.1 Entstehung

Im Abschnitt 2.1.3 wurden die ersten Versuche einer serverseitigen Implementation der Sprache JavaScript beschrieben. Es dauerte gut zehn Jahre, bis ein neuer Versuch unternommen wurde. Im Jahr 2009 stellte Ryan Dahl erstmals seine neue Entwicklung Node.js auf der JavaScript Conference Europe (JSConfEU) vor. Kurze Zeit später bekam Dahl Unterstützung von der Softwarefirma Joyent[1], die die Entwicklung bis heute federführend vorantreibt. Publiziert wird die Plattform unter der Massachusetts Institute of Technology (MIT)-Lizenz.

2.2.2 Google V8 JavaScript Engine

Die Entwicklung einer Engine, die die Ausführung von JavaScript beschleunigt, begann bei Google im Jahr 2008. Ursprünglich wurde die Engine für den Einsatz im Webbrowser Chrome konzipiert. Der Kern des Erfolges war, dass die Geschwindigkeit von JavaScript gegenüber anderen Ausführungsarten enorm zunahm. Die V8 Engine verwendet einige der neuesten Techniken der Compiler Technologie (Hughes-Croucher und Wilson, 2012), sodass Node.js stark davon profitiert. Bekannte Einsatzbereiche der V8 Engine sind neben Chrome auch TeaJS und V8js.

2.2.3 Bibliotheken

Die erste Veröffentlichung von Node.js kam bei ihrer Installation automatisch mit einem Paket-Manager: dem Node Package Manager (NPM). Der Paket-Manager wird über die Kommandozeile bedient. Benötigte Bibliotheken können somit schnell einer Anwendung hinzugefügt werden. Bezeichnend und zugleich ein Beispiel für Node.js-Anwendungen ist, dass der NPM selbst in Node.js geschrieben ist. Die Akzeptanz und starke Verwendung des Paket-Managers lässt sich auf der offiziellen Webseite[2] belegen. Im November 2014 sind über 100.000 Bibliotheken beziehbar mit über 25.000.000 Downloads pro Tag (npm, 2014). Neben der Vielzahl an öffentlichen Bibliotheken bietet Node.js auch eine begrenzte Anzahl interner Bibliotheken. Zu den gebräuchlichsten gehören zum Beispiel „http“ (Für die Servererstellung) oder auch „fs“ (Zugriff auf das Dateisystem).

2.2.4 Skalierung

Die Skalierbarkeit von Anwendungen ist ein wichtiger Aspekt für die zukünftige Entwicklung von Software (Kuan-Ching, 2009, S.27). Bei der Entwicklung von Node.js wurden die Möglich-

[1] https://www.joyent.com/
[2] https://www.npmjs.com/

keiten der vertikalen sowie horizontalen Skalierung bedacht.
Vertikal lassen sich Anwendungen über die interne Bibliothek „cluster“ skalieren. Das Konzept dieser Skalierung ist das Hinzufügen von Ressourcen zu einem vorhandenen Rechner. Hierzu zählen die Central Processing Unit (CPU) oder auch der Random Access Memory (RAM). Die Skalierung der Horizontalen ist eine Bezeichnung für das Hinzufügen ganzer Rechner (Knoten) zu einem vorhandenen Verbund. Diese Art der Skalierung ist realisierbar durch einen „Reverse Proxie Server“ im Vordergrund (z.B. nginix) und entsprechenden Node.js Servern auf den einzelnen Knoten im Hintergrund.

2.2.5 Prozess-Modell

Eine einleitende Betrachtung der Node.js-Prozesse wird Thema dieses Abschnitts sein. Die Unterscheidung der Plattform zu „traditionellen“ Konzepten der Prozessverwaltung ist wichtig im Hinblick auf ihren Einsatz. Im folgenden Abschnitt wird auf die Begriffe „Event-Driven“, „Non-Blocking I/O“, „Multi-Threading“ sowie „Event-Loop“ in ihrem Zusammenhang mit dem Prozess-Modell näher eingegangen.

2.2.5.1 Event-Driven

In der Betrachtung von JavaScript ist ereignisgesteuerte Programmierung ein zentraler Bestandteil. Node.js macht sich diesen Aspekt zunutze um so hoch skalierbare Server realisieren zu können (Hughes-Croucher und Wilson, 2012, S.19). Ein Ereignis kann die Texteingabe eines Benutzers, ein Mausklick oder eine Nachricht eines anderen Programms sein. In Node.js werden diese Ereignisse in der sogenannten „Event-Loop“ verwaltet. Diese Schleife ist ein einzelner Thread (Single-Thread), der alle auftretenden Ereignisse organisiert. Bei „Multi-Thread“ Anwendungen muss in der Entwicklung auf mögliche Konflikte mehrerer Threads geachtet werden. Dies erfordert bei komplexen Anwendungen ein hohes Maß an Abstraktion. Bietet ein System zusätzlich die Möglichkeit, Threads parallel auf verschiedenen Kernen der CPU auszuführen, kann es auch zu Konflikten beim Zugriff auf den gemeinsamen Speicher kommen (Teixeira, 2014, S.49).

2.2.5.2 Non-Blocking I/O

Das Nicht-Blockieren einer Anwendung durch Benutzerein- und ausgaben wird auch als „Asynchronous I/O“ bezeichnet. Diese Art der Verarbeitung ermöglicht die Ausführung weiterer Prozesse, ohne dass der Ursprungsprozess das System blockiert. Bei blockierenden Anwendungen hingegen muss erst auf die Eingabe eines Benutzers gewartet werden. Das kann

Performanceverluste zur Folge haben. Die nicht blockierende Ein- und Ausgabeeigenschaft ist Bestandteil der Event-Loop sowie anderer Teile von Node.js (Wilson, 2013, S.18). Bekommt die Schleife Ereignisse für rechenintensive Operationen, dann werden diese in sogenannte „Worker-Processes“ verschoben. Diese Prozesse tätigen auch Operationen wie Dateisystem- und Datenbankzugriffe.

2.2.5.3 Übersicht

Am Beispiel eines Hypertext Transfer Protocol (HTTP)-Servers lassen sich die beschriebenen Eigenschaften von Node.js veranschaulichen. Die initiale Anfrage eines Clients (Webbrowsers) geht an die Event-Loop. Diese Schleife wird durch das Betriebssystem aufgeweckt und bekommt ein Request- und ein Response-Objekt übergeben. Sofern die Abarbeitung der angeforderten Resource keinen Worker-Prozess benötigt, wird das Ergebnis an den Client zurückgegeben. Andernfalls wird ein neuer Prozess erzeugt. Bei der Erzeugung wird eine Callback-Funktion übergeben, die bei erfolgreicher Bearbeitung an die Event-Loop zurück geht. Diese Schleife sendet das Ergebnis dann zurück an den Client.

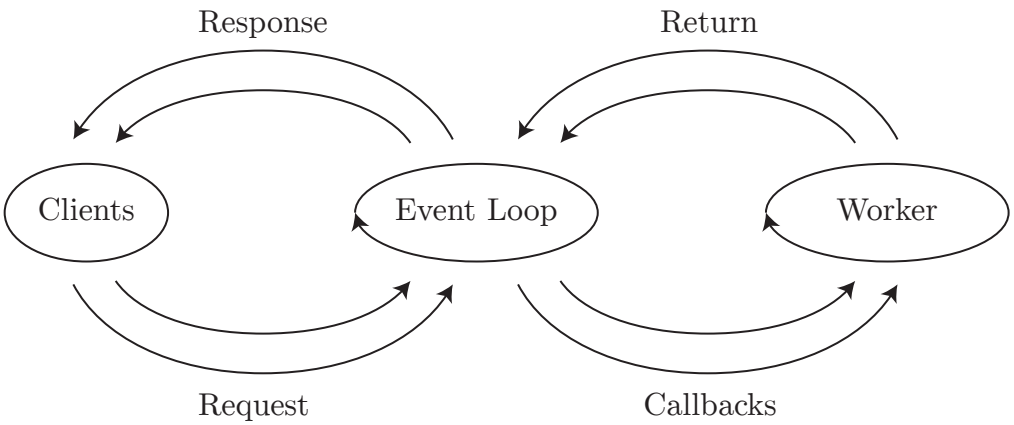

Abbildung 2.2: Node.js Prozess-Modell. Angelehnt an Wilson (2013, S.17)

2.2.5.4 Echtzeitanwendungen

Ein Einsatzgebiet für Node.js sind Anwendungen, die unmittelbar auf Vorkommnisse reagieren müssen. Die herkömmliche HTTP-Kommunikation bietet nur ein unidirektionales Verhalten.

Ein Client sendet eine Anfrage an einen Server und dieser sendet auf die Anfrage eine Antwort. Es ist nur über Umwege möglich ein unechtes bidirektionales Verhalten zu erhalten (z.B. Long-Polling). Mit dem Hinzukommen eines clientseitigen Webbrowser-WebSockets wurde die Protokolllandschaft erweitert. Über Node.js lässt sich mit nativen Mitteln ein serverseitiges WebSocket implementieren. Bedingt durch diese Einsatzmöglichkeit von WebSockets können zum Beispiel Webbrowser-Spiele den Nutzen aus dieser Technologie ziehen (Ihrig, 2013, S.207).

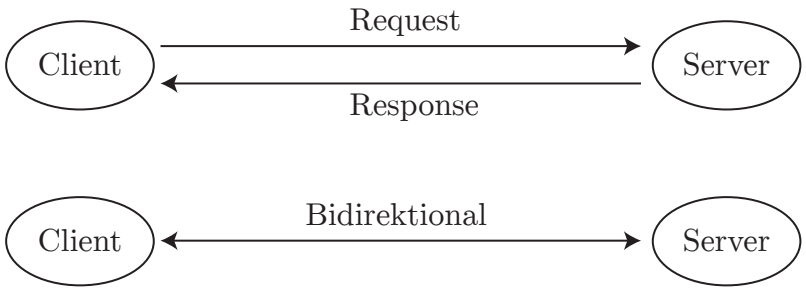

Abbildung 2.3: Veranschaulichung unidirektionale und bidirektionale Kommunikation. Angelehnt an Firtman (2013, S.535)

2.3 Webbrowser

Die Ausführung der Tests des JavaScript-Sprachkerns findet im Webbrowser statt. Die verschiedenen Umgebungen sind durch die Engines der Webbrowser definiert. Im folgenden Abschnitt werden die Ausführungsumgebungen genauer beleuchtet im Hinblick auf Implementation, Standardkonformität und Aktualität.

2.3.1 Komponenten

Webbrowser basieren auf sieben grundlegenden Komponenten. Das Benutzer-Interface beinhaltet alle sichtbaren Bereiche des Webbrowsers. Es dient der Interaktion des Benutzers mit dem Programm (Webbrowser) sowie der Webseite. Die Webbrowser-Engine dient der Kommunikation mit der Rendering-Engine und ist die Schnittstelle, die für die Speicherung von Daten zuständig ist. Die Darstellung der Seiteninhalte bzw. des Seitencodes findet durch die Rendering-Engine statt. Sie ist die Komponente, die für die Verarbeitung und Darstellung der Informationen verantwortlich ist. Die Datenpersistenz oder auch die Persistenz-Ebene wird für die Speicherung verschiedener Daten benötigt. Das können Cookies sein sowie auch Datenbanken in HTML5. Die Ausführung von HTTP-Requests wird über die Netzwerkkomponente

abgewickelt. Sie ist die Schnittstelle für die Kommunikation nach außen - also ins World-Wide-Web. Der JavaScript-Interpreter dient dem Parsen und Ausführen von JavaScript-Code. Diese Komponenten können auch als Engine betrachtet werden. Die letzte UI-Back-End Komponente ist Teil des Host-Systems und dient der Visualisierung des Webbrowsers im System an sich. Die Abbildung 2.4 verdeutlicht das Zusammenwirken der einzelnen Komponenten.

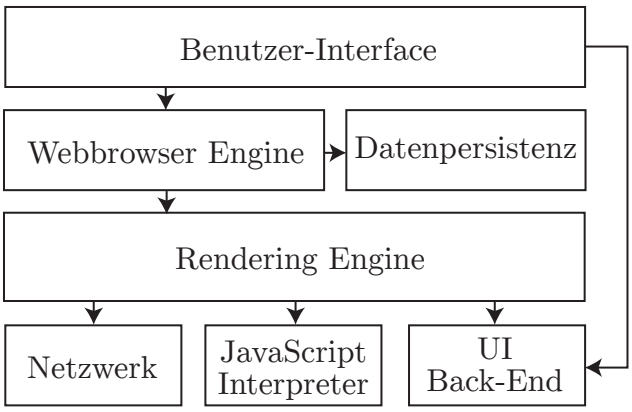

Abbildung 2.4: Webbrowser-Komponenten (Taligarsiel, 2015)

2.3.2 Engine

Die Engine ist Bestandteil eines jeden Webbrowsers und für die Implementation von ECMA-Script zuständig. Somit gibt es neben der Layout-Engine auch eine eigene Engine für JavaScript. Die folgende Tabelle gibt einen Überblick über die größten und wichtigsten Engines, die auf dem Webbrowser-Markt eingesetzt werden.

Layout-Engine	**JS-Engine**	**Browser-Beispiele**
Trident	JScript, Chakra	Microsoft Internet-Explorer
Gecko	Spidermonkey, TraceMonkey	Mozilla Firefox
WebKit	JavaScriptCore, SquirrelFish Extreme	Apple Safari
Blink	V8	Google Chrome

Tabelle 2.4: Webbrowser Layout- und JavaScript-Engines. Angelehnt an Kopec (2014, S.285)

2.3.3 ECMAScript

Die konforme Implementation des ECMA-Standards wird nicht bei allen JavaScript-Engines einheitlich umgesetzt. So findet zum Beispiel die dritte Edition eine einheitliche und komplette Umsetzung bei allen großen Engines (inklusive Presto Engine). Ein anderes Verhalten ist bei der fünften Edition zu beobachten. Hierbei existiert nur eine partielle Deckung in WebKit-basierten Systemen.

3 Analyse

Dieses Kapitel umfasst die Analyse der Probleme und Aufgaben, die durch die zu entwickelnde Anwendung entstehen. Der Kontext der Anwendung ist anhand eines Anwendungsfalldiagramms ersichtlich. Die Perspektiven der Akteure als Administratoren, Tester und Nutzer liegen im Fokus der Betrachtung.
Weiterhin sind drei verschiedene Ebenen der Anwendung zu analysieren. Die Testausführung bildet die erste Ebene. Hierbei wird die Umgebung eines Webbrowsers untersucht, in der Tests ausgeführt werden. An zweiter Stelle steht die clientseitige Analyse. Diese betrifft die Anforderungen im Webbrowser des Benutzers. Am Schluss steht die Untersuchung der Anwendung auf Serverseite. Die Unterscheidung der Anforderungen je nach ihrer Funktionalität sind Bestandteil der Untersuchung jeder Ebene.

3.1 Terminologie

Die Analyse setzt die Kenntnis einiger wichtiger Begriffe voraus. Ebenso müssen Ideen sowie auch Definitionen eindeutig sein. Eine klare Beschreibung wichtiger Begriffe ist Ziel dieses Abschnitts. Dieses Kapitel beinhaltet mehrere Stellen, an denen Definitionen passend zum behandelten Themenbereich auftreten. Vorab sind zwei zentrale Begriffe erklärungsbedürftig. Zu unterscheiden sind zum einen der „Test“ und zum anderen die „Suite“:

Definition 1 (Test)**.** *Ein Test ist ein Stück Quellcode, der in einer Webbrowserumgebung ausgeführt wird und Information über den getesteten Sachverhalt liefert. Der Test wird durch den zu testenden JavaScript Sprachteil definiert.*

Definition 2 (Suite)**.** *Eine Suite ist die Menge mehrerer Tests. Sie bildet eine logische oder inhaltliche Klammer um Tests. Weiterhin existiert keine innere Ordnung, anhand derer die Tests gewichtet werden.*

Definition 3 (Suiterelation). *Die Beziehung von Suiten zueinander ist durch eine inhaltliche Verbundenheit definiert. Eine Suiterelation ist eine inhaltliche Verbundenheit ohne Gewichtung der einzelnen Ebenen.*

Definition 4 (Ergebnis). *Die Ausführung eines Tests (Definition: 1) mit anschließender Auswertung des Resultats führt zu einem Ergebnis. Ein ausgeführter und verarbeiteter Test ist ein Ergebnis.*

3.2 Kontext

Die Abbildung 3.1 zeigt alle Akteure der Anwendung mit ihren unterschiedlichen Zugriffsmöglichkeiten auf das System. Die Anwendungsfälle sind explizit für einen Akteur formuliert. Eine Ausnahme bildet hierbei der Administrator. Er hat, sofern authentifiziert, einen kompletten Zugriff auf die Anwendung. Beim Prozess der Erstellung von Tests kann er aber auch die Rolle eines Testers oder Nutzers einnehmen. Weiterhin lassen sich die Akteure, Tester und Nutzer folgendermaßen charakterisieren: Ein Tester bietet durch eine Anfrage an die Anwendung seine Webbrowserumgebung zum Testen an. Er erwartet keine „Gegenleistung“, sondern erweitert die Informationsbasis. Der Nutzer hingegen ist auf den „Konsum“ von Informationen ausgerichtet, da er ausschließlich die Informationsbasis einsehen und seine Fragen beantwortet wissen will.

3.3 Webbrowser

Das Ziel der Anwendung ist die Untersuchung der JavaScript-Umgebung in Webbrowsern. Für das Erreichen des Ziels sind einige Faktoren genauer zu analysieren.
Zum einen betrifft dies die Ausführungsumgebung. Alle Tests sollen Informationen in der Ausführungsumgebung generieren. Dafür wird der fundamentale Sprachkern untersucht und das Ergebnis an die Anwendung zurückgeliefert. Die Basis bildet hierbei der Hypertext Markup Language (HTML)-Script-Tag. Dieser Tag ist mit entsprechenden Attributen für die Ausführung von JavaScript gedacht. Weitere Faktoren sind die Document Type Declaration (Doctype) sowie einige HTML-Form-Elemente mit entsprechendem Application Programming Interface (API) auf der JavaScript-Seite.
Zum anderen muss das Ausführungsergebnis zu der Anwendung zurückgeführt werden. Das

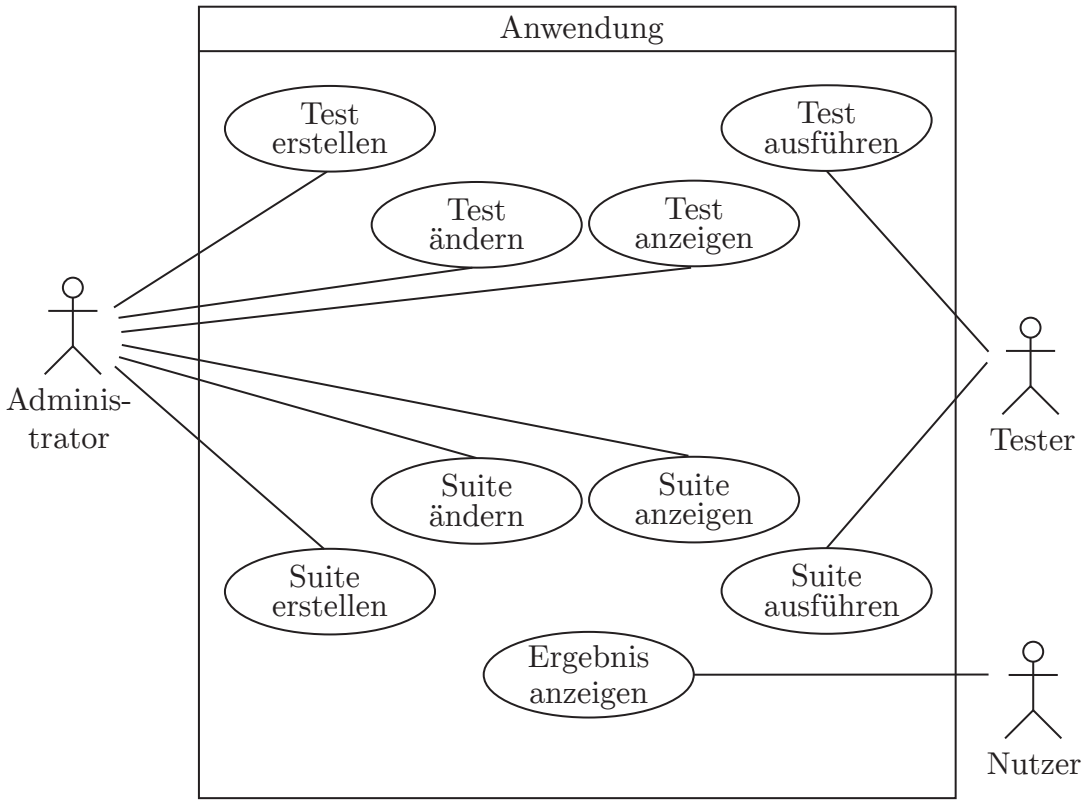

Abbildung 3.1: Anwendungsfalldiagramm nach UML

HTML-Form-Element ist wichtig für die temporäre Speicherung und Rückführung des Ergebnisses. Hierfür ist die Analyse des Interfaces notwendig, das eine teilautomatisierte Rückführung ermöglicht. Die Verarbeitung eines Ergebnisses ist aber Teil des Back-Ends.

3.3.1 Dokumententyp

Die Angabe des Dokumententyps ist für die Vermeidung des Quirk-Modus wichtig. Hierbei wird dem Webbrowser die verwendete HTML-Version mitgeteilt. Anhand der Angabe wird der Inhalt verarbeitet und dargestellt. Bei der initialen Einführung von JavaScript wurde HTML in seiner vierten Version verwendet. Der Script-Tag wurde für das Einbetten von JavaScript-Code in HTML-Dokumenten eingeführt (W3C, 2014).

```
1 <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN">
```

Listing 3.1: HTTP 4.0 Dokumententyp

Die vierte HTML-Version brachte einige Neuerungen mit sich, die wesentlich für die Entwicklung der Anwendung sind. Erstmalig wurde in dieser Version die Möglichkeit der Ausführung von Scripten geschaffen. Einhergehend damit konnten Formulare per Script manipuliert werden.

3.3.2 Form-Tag und Input-Tag

Form Elemente dienen der Eingabe von Benutzerinformationen mit anschließendem Versand. Der Form-Tag ummantelt alle Control-Tags (Text, Checkbox, Radio, etc.) und definiert über das Action-Attribut die Zieladresse, an welche die Informationen verschickt werden. Weiterhin wird auch die Übertragungsmethode angegeben.

```
1 <form name="myForm" action="http/response/path" method="post">
2     <input type="text" name="myText" value="Eins">
3 </form>
```

Listing 3.2: HTTP 4.0 Form-Tag und Input-Tag

3.3.3 Script-Tag

Die Ausführung von JavaScript-Code geschieht über die Definition von Script-Tags. Der gesamte Code zwischen dem öffnenden und schließenden Tag wird als Code interpretiert. Die Ausführung erfolgt, sobald der Tag durch den Webbrowser gerendert bzw. interpretiert wird.

```
1 <script type="text/javascript">
2     // JavaScript-Code
3 </script>
```

Listing 3.3: HTTP 4.0 Script-Tag

3.3.3.1 Fehlerfälle

Wichtig bei der Untersuchung des Script-Tags sind mögliche Fehlerfälle. HTML-Content wird immer zeilen- bzw. tagweise verarbeitet. Dieser Abarbeitungsreihenfolge unterliegt somit auch der Script-Tag. Sobald dieser Tag gerendert bzw. durch den JavaScript-Interpreter verarbeitet wird, sind folgende Punkte in der Analyse wichtig.

Syntaxfehler Bei Syntaxfehlern, die zur Laufzeit auftreten, wird je nach Webbrowser-Konfiguration eine visuelle Meldung ausgegeben (alert) oder die Webbrowser-Konsole wirft

eine Fehlermeldung aus. Das Script wurde dann bis zur Zeile des Syntaxfehlers abgearbeitet, stoppt dann aber an dieser Stelle. Alle nachfolgenden Statements werden durch den Interpreter ignoriert.

Multiple Script-Tags Werden mehrere Script-Tags nacheinander in einem Dokument definiert, ist das Auftreten von Fehlern interessant. Gegeben seien drei Script-Tags hintereinander. Beim ersten Tag seien keine Fehler im Quellcode. Dieser wird komplett verarbeitet. Der zweite Tag hingegen beinhaltet einen Syntaxfehler. Dieser unterliegt dann der Beschreibung aus Abschnitt 3.3.3.1. Trotz des geworfenen Fehlers beim zweiten Tag wird der dritte Tag beim Prozess des Renderns wieder interpretiert. Der aufgetretene Fehler des zweiten Tags hat keinen Abbruch des Renderns des restlichen Inhalts zu Folge. Somit kann im dritten Tag auf die generierten Informationen von erstem und zweitem Tag zugegriffen werden. Dieses Verhalten lässt sich im Listing 3.4 nachvollziehen.

```
1  <!-- Tag 1 -->
2  <script type="text/javascript">
3      var a = 1;
4  </script>
5  <!-- Tag 2 -->
6  <script type="text/javascript">
7      var b = 2;
8      !!!!!!!!!! // Provozierter Syntaxfehler
9      var c = 3;
10 </script>
11 <!-- Tag 3 -->
12 <script type="text/javascript">
13     var d = 4;
14     // a = 1, b = 2, c = undefined, d = 4
15 </script>
```

Listing 3.4: Beispiel JavaScript-Fehlerverhalten

3.3.4 Ansprechbarkeit

Die Verbindung der beschriebenen Tags mit JavaScript-Code ist auf folgende Weise möglich: Je nach Dokumententyp wird ein Formular definiert, das eine feste Anzahl an Feldern mit Schlüsseln als Namen bietet. Dieses Formular kann aus den Script-Tags bzw. dem JavaScript-Code heraus angesprochen werden. Das globale Window-Objekt beinhaltet eine Referenz auf das aktuelle Dokument. Innerhalb des Dokuments können alle Form-Tags angesprochen werden. Unterhalb der Formulare können dann selektiv auch alle Input-Elemente manipuliert

werden. Eine exemplarische Manipulation des Input-Tags „myText“ geschieht in JavaScript folgendermaßen:

```
1 window.document.myForm.myText = "Zwei";
```

Listing 3.5: Manipulation Input-Tag

3.3.5 Automatisierung

Zwei entscheidende Faktoren für eine Automatisierung von Tests bzw. einer sequenziellen Ausführung sind wichtig. Zunächst besteht seit der ersten JavaScript Version die Möglichkeit Formulare per Script zu versenden. Hierfür steht die Submit-Methode zur Verfügung. Diese kann auf jedem Formularobjekt aufgerufen werden. Sie serialisiert die Formulardaten und sendet diese an die entsprechende Adresse aus dem HTML-Form-Action-Attribut.
Durch die Methode und das Action-Attribut besteht die Möglichkeit eine teilautomatisierte Sequenz zu schaffen. Jedem clientseitigen Request folgt eine serverseitige Response. Dieses Schema gilt sowohl für Aufrufe über die POST- als auch über die GET-Methode. Eine Vollautomatisierung ist nicht realisierbar, da der initiale Aufruf immer vom Client erfolgen muss. Die Submit-Methode wird folgendermaßen aufgerufen:

```
1 window.document.myForm.submit();
```

Listing 3.6: Versenden eines Formulars

3.3.6 Abhängigkeiten

Das JSON-Objekt, das von Douglas Crockford entwickelt wurde (Crockford, 2008, 136), ist kein Standard der ersten Version von JavaScript. Dieses Objekt bietet i.d.R. zwei Methoden (parse, stringify). Die Überprüfung dieser Methoden setzt die Existenz des Objekts voraus. Insofern müsste der Test für eine Methode zunächst das Objekt prüfen. Das widerspricht aber der Definition eines Tests, da diese nur einen Aspekt untersuchen soll. Die Analyse führt nun zu dem Sachverhalt der Existenz zweier Tests, wobei der eine die Voraussetzung für den anderen Test bildet. Die so entstehende Relation ist wie folgt definiert:

Definition 5 (Testrelation). *Die technische Voraussetzung für eine weitere Überprüfung eines Sprachaspekts wird durch Relationen abgebildet.*

Ein Test kann somit einen oder mehrere andere Tests für sich voraussetzen. Diese Abhängigkeit beschreibt die Testrelation.

3.3.7 Testsequenz

Die Abbildung 3.2 zeigt den sequenziellen Ablauf einer Testanfrage durch einen Client. Die initiale Anfrage mittels der GET-Methode fungiert als Einstieg in den Ablauf. Anschließend wird der Test im Mantel eines HTML-Dokuments an den Client zurückgegeben. Das clientseitige Rendern geschieht automatisch durch den Webbrowser. Auf die Ausführung sowie Speicherung des Testergebnisses hat der Client keinen direkten Einfluss. Am Ende wird das Formular automatisch an die Anwendung zurückgegeben (Siehe 3.3.5). Eine weitere Ausführung des Tests ist durch den Testkontext bzw. die Suite definiert. Aus dieser Betrachtung heraus lässt sich eine Testsequenz genau beschreiben, was zu der folgenden Definition führt:

Definition 6 (Testsequenz)**.** *Eine Testsequenz ist eine Folge von Tests. Sinn einer Sequenz ist das sequenzielle Durchlaufen mehrerer Tests in einer Folge. Hierbei bauen die Tests aufeinander auf, sodass eine Sequenz scheitert, wenn ein Teilabschnitt der Sequenz fehlschlägt.*

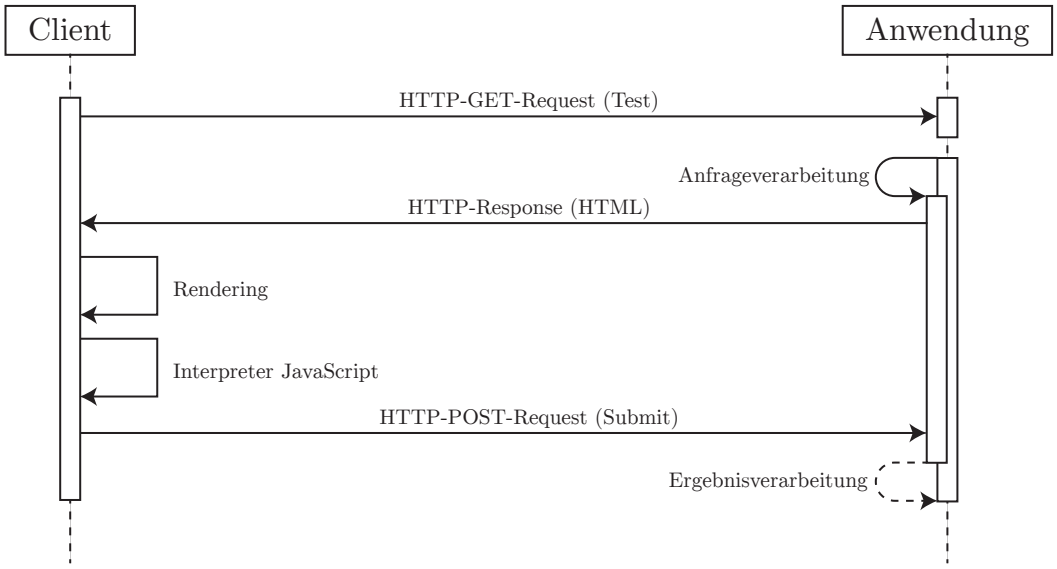

Abbildung 3.2: Ablauf einer Testsequenz

3.4 Ausführung

Das Testen der Existenz von beispielsweise einem Objekt bildet den Kern der Anwendung. Das Thema der Arbeit sieht die Untersuchung des JavaScript-Sprachkerns über die Anwendung vor. Hierbei dient die Anwendung dem Ziel, das Ergebnis zu verarbeiten und somit Informationen über die Existenz dieses Objektes zu erlangen. Der Benutzer der Anwendung schreibt den Test nicht statisch, sondern dynamisch. Er gibt auf diesem Wege den Kontext vor, der untersucht werden soll. Die Verarbeitung eines Testergebnisses in der Terminologie dieser Arbeit wird wie folgt definiert :

Definition 7 (Ergebnis)**.** *Die Ausführung eines Tests (Definition: 1) mit anschließender Auswertung führt zu einem Ergebnis. Ein analysiertes und verarbeitetes Testergebnis ist ein Ergebnis.*

3.4.1 Anwendungsfälle

Dieser Abschnitt beschreibt den Anwendungsfall des Aufrufens eines Tests an der Anwendung. Die Betrachtung geht von einem „Tester“-Akteur aus. Grundsätzlich existieren zwei verschiedene Szenarien des Aufrufs und Ablaufs. Diese beiden Anwendungsfälle leiten sich von einem Test und einer Suite ab.

Name	**Test aufrufen**
Beschreibung	Das Aufrufen eines Tests an der Anwendung durch den Akteur „Tester“. Der Test wird in der Webbrowser-Umgebung ausgeführt und das Ergebnis wird zur Anwendung zurückgegeben.
Beteiligte Akteure	Tester
Ergebnis	Das Ergebnis wurde zur Anwendung zurückgegeben und gespeichert.
Nachbedingung	Die Testsequenz endet.
Standardablauf	1. Der Tester sendet einen HTTP-Request an einem Test. 2. Ein HTML-Dokument kommt als HTTP-Response. 3. Das Dokument wird durch den Webbrowser gerendert und der Test wird ausgeführt. 4. Das Testergebnis wird automatisch per HTTP-POST-Request an die Anwendung gesendet. 5. Ein leerer HTTP-Response
Alternative Ablaufschritte	5.1 Der nächste Test der Sequenz kommt als HTTP-Response.

Tabelle 3.1: Anwendungsfall: Test aufrufen

Name	**Suite aufrufen**
Beschreibung	Das Aufrufen einer Suite an der Anwendung durch den Akteur „Tester“. Die Suite wird in der Webbrowser-Umgebung ausgeführt und zur Anwendung zurückgegeben.
Beteiligte Akteure	Tester
Verwendete Anwendungsfälle	Test aufrufen.
Ergebnis	Das Testergebnis wurde zur Anwendung zurückgegeben und gespeichert.
Nachbedingung	Ein neuer Test wurde ausgeliefert.
Standardablauf	1. Der Tester sendet einen HTTP-Request an eine Suite. 2. Ein HTML-Dokument kommt als HTTP-Response. 3. Das Dokument wird durch den Webbrowser gerendert und der Test wird verarbeitet. 4. Das Testergebnis wird automatisch per HTTP-POST-Request an die Anwendung gesendet. 5. Ein leerer HTTP-Response
Alternative Ablaufschritte	5.1 Der nächste Test der Suite kommt als HTTP-Response.

Tabelle 3.2: Anwendungsfall: Suite aufrufen

3.4.2 Funktionale Anforderungen

Die funktionalen Anforderungen an einen Test und eine Suite lassen sich wie folgt auflisten: Jede Anforderung bekommt einen eindeutigen Schlüssel zugeordnet. Der erste Teil des Schlüssels ist eine Abkürzung für „Funktionale Anforderung“. An zweiter Stelle wird der Typ identifiziert wie „T“ für Test und „S“ für Suite mit einem darauf folgenden und fortlaufenden Index.

FA-T-1

Nachdem der Tester einen Test aufgerufen hat, soll das Dokument, das den Test beinhaltet, an den Tester ausgegeben werden.

FA-T-2

Das Durchlaufen einer Testsequenz bricht nur beim Fehlschlagen eines Tests ab.

FA-T-3

Ergebnisse werden als „abgebrochen“ eingestuft und gespeichert, wenn ein vorheriger Test der Sequenz fehlschlägt.

FA-T-4

Das Rendern und Interpretieren des Dokuments (Tests) durch den Webbrowser hat zur Folge, dass am Ende das Ergebnis automatisch gesendet wird.

FA-S-1

Nachdem der Tester eine Suite aufgerufen hat, soll das Dokument des ersten Tests, das den Test beinhaltet, an den Tester ausgegeben werden.

TA-S-2

Das Durchlaufen einer Suite bricht nicht beim Fehlschlagen eines Tests ab.

3.5 Darstellung

Die clientseitige Analyse der Anwendung ist Bestandteil dieses Abschnitts. Diese bezieht sich auf den Akteur „Nutzer“, der hierbei im Mittelpunkt der Betrachtung steht. In erster Linie zeichnet sich der Akteur durch ein rein konsumierendes Verhalten aus. Er ist nicht Teil der Informationsgenerierung, aber auch nicht ausschließlich auf diese Rolle festgelegt. Er kann genau wie der Akteur „Tester“ seine Rolle wechseln.

3.5.1 Anwendungsfälle

Die Anwendungsfälle für einen Tester lassen sich aus Abschnitt 3.5 ableiten. Der Nutzer tritt bei der Anwendung als Konsument auf und erlangt Informationen.

Name	**Ergebnis abrufen**
Beschreibung	Das Aufrufen eines Testergebnisses bei der Anwendung durch den Akteur „Nutzer“
Ergebnis	Das Testergebnis einer konkreten Umgebung wird ausgeliefert.
Standardablauf	1. Der Tester sendet einen HTTP-Request an einen Test. 2. Ein HTML-Dokument kommt als HTTP-Response. 3. Die konkreten Informationen über das Testergebnis werden dargestellt.

Tabelle 3.3: Anwendungsfall: Testergebnis abrufen

3.5.2 Funktionale Anforderungen

FA-E-1

Nachdem der Nutzer ein Testergebnis aufgerufen hat, soll das Dokument, das den Test beinhaltet, an den Nutzer ausgegeben werden.

FA-E-2

Ein Testergebnis gibt visuellen Aufschluss über den Erfolg oder das Fehlschlagen eines Tests.

3.6 Verwaltung

Dieser Abschnitt beschreibt die Möglichkeiten der Einflussnahme eines Administrators. Es wird eine Übersicht erstellt, die Aufschluss über den Einfluss eines Administrators an der Anwendung gibt. Diese Zugriffsstelle am System bildet den verwaltungs- und erstellungstechnischen Kern der Anwendung.

3.6.1 Anwendungsfälle

Name	**Test erstellen**
Beschreibung	Das Erstellen eines Tests über die Anwendung
Beteiligte Akteure	Administrator
Ergebnis	Ein Test wurde erstellt und gespeichert.
Vorbedingungen	Benutzerkonto für Authentifizierung am System
Standardablauf	1. Der Benutzer authentifiziert sich am System. 2. Das Formular für die Erstellung eines Tests wird aufgerufen. 3. Es wird ein eindeutiger Name angegeben (Optional mit Beschreibung). 4. Dem Test werden ggf. Vortests zugewiesen. 5. Der Test wird ggf. Suites zugewiesen. 6. Der Test wird in JavaScript formuliert. 7. Der Test wird abgesandt und gespeichert.

Tabelle 3.4: Anwendungsfall: Test erstellen

Name	**Suite erstellen**
Beschreibung	Das Erstellen einer Suite über die Anwendung
Beteiligte Akteure	Administrator
Ergebnis	Eine Suite wurde erstellt und gespeichert.
Vorbedingungen	Benutzerkonto für Authentifizierung am System
Standardablauf	1. Der Benutzer authentifiziert sich am System. 2. Das Formular für die Erstellung einer Suite wird aufgerufen. 3. Es wird ein eindeutiger Name angegeben (Optional mit Beschreibung). 4. Die Suite werden ggf. Vorsuites zugewiesen. 5. Die Suite wird abgesandt/gespeichert.

Tabelle 3.5: Anwendungsfall: Suite erstellen

3.6.2 Funktionale Anforderungen

FA-A-1

Nach der Erstellung eines Tests gibt es eine Rückmeldung über den Erfolgs- oder Fehlerfall des Vorgangs.

FA-A-2

Nach der Änderung eines Tests gibt es eine Rückmeldung über den Erfolgs- oder Fehlerfall des Vorgangs.

FA-A-3

Nach der Löschung eines Tests gibt es eine Rückmeldung über den Erfolgs- oder Fehlerfall des Vorgangs.

FA-A-1

Nach der Erstellung einer Suite gibt es eine Rückmeldung über den Erfolgs- oder Fehlerfall des Vorgangs.

FA-A-2

Nach der Änderung einer Suite gibt es eine Rückmeldung über den Erfolgs- oder Fehlerfall des Vorgangs.

FA-A-3

Nach der Löschung einer Suite gibt es eine Rückmeldung über den Erfolgs- oder Fehlerfall des Vorgangs

4 Konzeption

Eingangs wird in diesem Kapitel die zugrundeliegende Software vorgestellt mit ihren spezifischen Eigenschaften, die für die Anwendung wichtig sind. Weiterhin wird das Konzept der Anwendung erstellt und die verwendete Architektur mit ihrer besonderen Eignung für die Anwendung definiert. Das Kapitel widmet sich der Konzeption bezogen auf die Bereiche des Front- und Back-Ends. Ausgangspunkt der Untersuchung ist das Kapitel der Analyse unter Berücksichtigung der gewonnenen Erkenntnisse.
Zuerst wird die Architektur des Systems näher betrachtet, indem die einzelnen Teilbereiche eingehend beschrieben werden.

4.1 Verwendete Software

Für die Realisierung der Konzeption dieser Arbeit ist eine Vielzahl an Bibliotheken notwendig. Diese Bibliotheken können Teil einer Middleware oder aber notwendige Werkzeuge sein, ohne deren Existenz enorme Vorarbeit notwendig wäre. In diesem Abschnitt möchte ich auf die wichtigsten Bibliotheken eingehen. Es soll dabei deutlich werden, welche Funktionalität sie bieten und was die entscheidenden Faktoren für den Einsatz sind. Eine Gliederung besteht in den Einsatzbereichen des Front-Ends und des Back-Ends. Da dieses Projekt in beiden Bereichen auf JavaScript basiert, ist eine Abgrenzung bzw. Zuordnung nicht immer eindeutig.

4.1.1 Front-End

Der Bereich des Front-Ends ist in der Regel nah am Benutzer angesiedelt. Er umschreibt Akteure, die direkt an einem System Eingaben tätigen - also mit dem System interagieren. Drei Bibliotheken sind hier entscheidend: Für eine sauber formatierte Darstellung benötigt man das Cascading Style Sheets (CSS)-Framework Bootstrap. Die Benutzerinteraktion im Graphical User Interface (GUI) soll in der Kommunikation einen geringen Overhead haben. Aus diesem Grund wird AngularJS als Framework für eine Single Site Page (SSP) eingesetzt. Außerdem findet „„D3“ Verwendung für die Visualisierung der Ergebnisse von Tests.

4.1.1.1 Bootstrap

Das Open-Source Framework Bootstrap wurde von Mark Otto und Jacob Thornton entwickelt. Zu der damaligen Zeit arbeiteten sie für das Unternehmen Twitter (Spurlock, 2013, S.17). Bootstrap zählt sich selbst zu der Klasse von HTML, CSS und JavaScript (JS) Frameworks (Otto und Thornton, 2014).

Raster Bootstrap bietet ein Rastersystem, mit dem eine Positionierung von Elementen erreicht werden kann. In der horizontalen Positionierung können bis zu zwölf Spalten definiert werden. Die Spalten orientieren sich an der gesamten Breite, die einer Zeile zugeteilt werden. Die vertikale Aufteilung erfolgt nach einer Gruppierung zu Zeilen. Spalten können durch einen Offset versetzt wie auch verschachtelt werden. Richtet sich die Positionierung nicht an dem Standard des Rasters aus, können auch fließende Bereiche definiert werden. Diese Bereiche orientieren sich an einer prozentualen und nicht pixelbezogenen Ausrichtung.

Responsive Design Für eine passende Darstellung der Inhalte richtet sich Bootstrap nach dem Paradigma des Responsive Designs. Hierfür gibt es zum Beispiel verschiedene Klassen, die einem Raster zugeordnet werden können, sodass eine optimierte Darstellung auf Endgeräten möglich ist. Hierfür gibt es CSS-Klassen, die für folgende Ausgabegeräte konzipiert wurden:

- Mobiltelefone
- Tablets
- Desktops
- Große Desktops

Typographie Das Framework verwendet nur Standard-Schriftarten, die auf allen großen Betriebssystemen verfügbar sind (Spurlock, 2013, S.25). Die Formatierungen aller gängigen HTML-Tags sind aufeinander abgestimmt. Dies umfasst die Überschriften bis zu den Aufzählungspunkten oder Zitierungsanweisungen.

Symbole (Icons) Eine bessere optische Orientierung kann durch den Einsatz von Symbolen erreicht werden. Das Framework bietet über 100 Symbole, die in Überschriften, Absätzen oder Buttons verwendet werden können.

Layout-Komponenten Die Benutzereingabe auf Webseiten ist bei der Interaktion zentral. Die klassischen Formularelemente wie Text, Select oder Checkbox werden durch Bootstrap zusätzlich um nicht native HTML-Elemente erweitert. Diese setzen sich meist aus einer Vielzahl an HTML-Tags zusammen. Die Anordnung dieser Elemente ist ein Verbund aus Raster, Typographie sowie Symbolen. Einige Beispiele sind Header, Button-Groups und Paginierung.

4.1.1.2 AngularJS

AngularJS ist ein Framework von Google für die Erstellung von SSP-Anwendungen. Es ist eine quelloffene Software, die unter der MIT-Lizenz veröffentlicht wird. Die Entwicklung von AngularJS ist auf einen Aspekt zurückzuführen. Die Darstellungssprache „...HTML wurde nicht für dynamische Ansichten konzipiert“ (Google, 2014). Diese Erkenntnis ist der Grundgedanke hinter der Entwicklung des Frameworks.

Template Engine Das Füllen von inhaltsleeren Templates mit Informationen von Servern ist eine Funktionalität von AngularJS. Templates werden in AngularJS mit spezifischen Elementen und Attributen versehen. Bei dem Prozess des Renderns werden Daten aus dem Model und dem umschließenden Controller herangezogen. Zu den Elementen gehören Direktiven, Markups, Filter und Form-Controls.

Model View Controller Das Konstrukt, Software in verschiedene Bereiche zu unterteilen, ist ein weiterer Bestandteil von AngularJS. Das Model View Controller (MVC)-Muster ist bei „nahezu jeder Desktop-Entwicklungsumgebung populär, bei der Benutzerinterfaces involviert sind “ , (Green und Seshadri, 2013, S.15). Dieses Prinzip wird wie folgt zur Anwendung gebracht: Das DOM als View, mit dem der Benutzer interagiert. Das Model sind JavaScript-Objekte mit definierten Eigenschaften, den Daten. Controller sind durch AngularJS erzeugbar und mit Logik vom Entwickler anreicherbar.

Data Bindings Das einmalige Rendern von Daten in einen View mit der anschließenden Ausgabe beim Benutzer ist trivial. Kompliziert wird es meist erst, wenn die bereits gerenderten Daten aktualisiert werden sollen. Data Bindings sind die persistente Verbindung von Daten aus dem Model mit dem Platzhalter eines Views. Bei jeder Änderung der Daten des Models ändert sich auch der View.

Dependency Injection Die Implementation der Einbindungen von Abhängigkeiten folgt dem Law of Demeter (LoD)-Muster. Beispielsweise wird bei der Erstellung eines Controllers

nach dem Kontext ($scope) gefragt und dieser muss nicht vorab erzeugt werden. Dieses Prinzip gilt in weiteren Bereichen von AngularJS (Controller, Services, Factories und Directiven).

Direktive HTML besitzt eine definierte Anzahl an Tags. Diese Tags folgen einer Formatierung, die sich ausschließlich per CSS manipulieren lässt. AngularJS bietet die Definition von eigenen Tags, die einem selbstbestimmten Verhalten folgen. Diese Möglichkeit besteht, da AngularJS-Anwendungen beim Bootstrapping-Prozess das komplette HTML-Dokument (oder definierte Teile davon) einlesen und auswerten.

4.1.1.3 D3

Die Bibliothek kann HTML, Scalable Vector Graphics (SVG) und Canvas Elemente des DOM interaktiv visualisieren. Der Name ist eine Ableitung aus „Data Driven Documents“. Sie wurde von Mike Bostock für die clientseitige Visualisierung von Daten entwickelt.

Draw Funktion Der zentrale Punkt, an dem eine Visualisierung von Daten beginnt, ist die Draw Funktion. Zuerst wird ein Element des DOM als Wurzel gewählt. Dieser Knoten bekommt alle Elemente zugewiesen, die für die Darstellung notwendig sind. Der Programmierstil auf dem D3-Objekt ist das Method-Chaining. Hierbei liefert jede aufgerufene Methode das Root-Objekt zurück, sodass gleich wieder die nächste Methode darauf aufgerufen werden kann.

Formatierung Durch die Bibliothek bzw. deren Technik ist es möglich, dass jegliche Formatierung von Elementen durch CSS definiert werden kann. Anhand eines SVG-Elements als Wurzel kann der gesamte Code in einem externen Stylesheet ausgelagert werden. Diese Trennung von Logik und Formatierung dient dem Paradigma des „Separation of Concerns“.

Datenselektion Nach der Initialisierung eines D3-Objekts können selektiv bestimmte Datensätze angesprochen werden. Zu diesem Zweck können allen Formatierungsanweisungen auch Funktionen als Callback übergeben werden. Die über die Daten iterierende Schleife ruft den Callback mit dem entsprechenden Datensatz auf.

4.1.2 Back-End

Das Back-End kann metaphorisch als das Rückgrat eines Systems gesehen werden. Die wichtigsten Technologien für die Anwendung sind Express, Jade und Mongoose. Zuerst muss die Anwendung eine API nach außen anbieten. Für diesen Zweck wird das Framework Express

verwendet. Für das serverseitige Rendern von Templates kommt die Engine Jade zum Einsatz. Über ein Plug-In kann die Engine nahtlos in Express verwendet werden. Ein Teil der Anwendung wird Daten in MongoDB speichern. Für die Datenbank-Kommunikation wird die Bibliothek Mongoose verwendet.

4.1.2.1 Express

Das Framework Express für Web-Anwendungen in Node.js ist eines der verbreitetsten. Die Codebasis der dritten Version umfasst weniger als 1000 Zeilen Quellcode. Bei der Entwicklung legte man großen Wert auf die Modularität und Erweiterbarkeit.

Objekte Eine Express-Anwendung besteht aus drei fundamentalen Objekten: Zum einen handelt es sich um das **Anwendungsobjekt**. Auf diesem werden alle Konfigurationen vorgenommen. Weiterhin wird, sofern eine spezielle Middleware die Anwendung erweitern soll, diese auch über das Objekt hinzugefügt. Außerdem werden etwaige Routes genauso wie Template Engines über das Anwendungsobjekt definiert.
Das zweite wichtige Objekt ist das **Requestobjekt**. Jede Anfrage an eine Express-Anwendung hat das Objekt zur Folge. Das Objekt hält Informationen wie die angefragte Seite samt Anfrageparameter vor. Weiterhin sind Informationen zu den gesendeten Cookies sowie die Internet Protocol (IP)-Adresse des Anfragestellers enthalten.
Das Pendant zu dem Requestobjekt ist das **Responseobjekt**. Die Angabe der Kodierung einer Antwort ist Teil seiner Funktionalität. Außerdem werden die HTTP-Status-Codes genauso wie mögliche Cookies hierüber gesetzt. Die beiden letzten Objekte bilden zusammen den zentralen Zugriff auf die Verarbeitungskette durch Middleware.

Middleware Das modulare Konzept von Express ist für das Einbinden von Middleware notwendig. Funktionalitäten werden hierbei nicht dem Kern des Frameworks hinzugefügt, sondern gekapselt. So können Anwendungen nur die Teile hinzugefügt werden, die für die Realisierung der Aufgabe wirklich notwendig sind. Middleware für Express ist für die Manipulation von Requests und Responses gedacht (Yaapa, 2013, S.19). Der Verarbeitungsfluss einer Anfrage kann über diesen Weg direkt durch eine Middleware beendet bzw. auch beantwortet werden.

4.1.2.2 Jade

Bei Jade handelt es sich um eine Template-Sprache. Syntaktisch bildet Jade eine eigene Sprache, muss aber zu HTML kompiliert werden. Aus diesem Grund spricht man von einem Preprocessor. Die Sprache ist für mehrere Plattformen verfügbar und wurde von Hypertext Abstraction

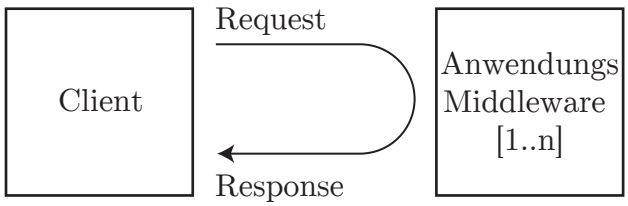

Abbildung 4.1: Anfrageverlauf über Middleware. Angelehnt an Yaapa (2013, S.34)

Markup Language (HAML) beeinflusst. Neben der Abbildung aller HTML-Ausdrücke ist die Sprache aber noch mit weiterer Funktionalität angereichert.

Formatierung Bei dem Vergleich inhaltlich gleicher Daten ist die Zeichenanzahl in Jade kürzer als in HTML. In HTML werden Bereiche durch öffnende und schließende Tags definiert. Die Einrückung des Codes spielt für die Ausgabe keine Rolle. Jade hingegen richtet die Bereiche nach der Tiefe der Einrückung aus. Somit werden keine schließenden Klammern benötigt. Das erhöht die Übersicht, macht sich aber auch beim Platzbedarf der Templates bemerkbar.

Templating Ähnlich wie bereits im Abschnitt über AngularJS gibt es in Jade die Möglichkeit Templates mit Daten zu versehen. Für diesen Zweck werden Platzhalter definiert, die beim Kompilierungsprozess ersetzt werden. Neben der trivialen Ersetzung mit Text können die Platzhalter auch ausgewertet werden. Ein Beispiel dafür sind mathematische Berechnungen.

Logik Die strikte Trennung von darstellendem Code und Programmlogik verschwimmt in Jade in Teilen. Der Jade Code wird in JavaScript kompiliert und erlaubt somit auch die Verwendung von JavaScript. Daher können die logischen Operatoren aus JavaScript auch direkt in Jade verwendet werden (Lang, 2014, S.29).

Filter Auf Grundlage der Bibliothek „Transformers“von Forbes Lindesay besteht in Jade die Möglichkeit weitere vorverarbeitete Sprachen mit einzubinden. So ist der Einsatz von CoffeeScript oder auch Markdown in Jade möglich. Der Verwendung geht ausschließlich die Installation genannter Sprachen voraus (Lang, 2014, S.35).

Überblick Ein weiterer Bestandteil von Jade ist die Vererbung von Code. Hierbei können Dateien in entsprechende Bereiche eingebunden und außerdem Blöcke definiert werden, die

bei einer Vererbung jeweils anders ersetzt werden. Eine komplette Übersicht über die Sprache ist auf der offiziellen Webseite zugänglich[1].

4.1.2.3 Mongoose

Die Bibliothek Mongoose ist ein Object-Document Modeler (ODM) für das Abbilden von Daten in MongoDB. Sie dient als Schnittstelle für die Kommunikation der Anwendung mit der Datenbank. Hierbei wird das Model auf der Anwendungsseite definiert und nicht auf der Datenbankseite (Holmes, 2013, S.12).

Schema Die Erstellung eines Schemas für die Dokumente auf Anwendungsebene ist eine Voraussetzung für den Einsatz von Mongoose. Das Schema dient der Definition der Struktur eines Dokuments. Neben ein paar bibliothekseigenen Schematypen wie „ObjectId“ und „Mixed“ sind alle Basisdatentypen aus JavaScript anwendbar. Hierzu gehören:

- String
- Number
- Date
- Buffer
- Boolean
- Array

Validierung Das Überprüfen oder Beschränken von Benutzereingaben ist ein wichtiger Bestandteil einer jeden Anwendung. Mongoose bietet diese Möglichkeit bei der Definition eines Schemas. Der Basisvalidator „required“ kann in jedem Schema verwendet werden. Er gibt an, ob ein Teil eines Dokuments unabdingbar ist. Die Überprüfung des Number-Basisdatentyps ist über „min“ oder „max“ möglich. Weitere Validatoren für Strings sind „match“ oder „enum“.

Model Bei einem Model handelt es sich um die kompilierte Version eines Schemas (Holmes, 2013, S.14). Dieses Model repräsentiert auf der Anwendungsebene ein konkretes Dokument aus der Datenbankebene. Alle klassischen Create Read Update Delete (CRUD)-Operationen sind auf dem Model verfügbar und sorgen für die Persistenz der Daten.

[1] http://jade-lang.com/

4.2 Systemarchitektur

Die Abbildung 4.2 ist eine rudimentär-schematische Darstellung, die Aufschluss über die Kommunikationswege der Anwendung im Kontext des Systems gibt. Die einzelnen Pfade der Kommunikation sind in der nachfolgenden Auflistung zusammengefasst und spezifiziert.

Viele Pfade der Kommunikation basieren auf dem datenorientierten Austausch von Informationen. Hierbei ergibt sich der Vorteil, dass eine Schnittstelle einer Domäne verwendet werden kann, ohne die Logik dieser Domäne implementieren zu müssen (Elliott, 2014, S.174).

Zugrunde liegt dem System die Architektur der „Drei-Schichten“. Hierbei dient die Schicht der Präsentation als Schnittstelle zum Benutzer. Alle Daten werden hier visuell dargestellt und können durch Interaktion über das GUI manipuliert werden. Inhaltlich kann an dieser Stelle auch von dem Front-End gesprochen werden. Die Verarbeitung und eigentliche Logik der Anwendung ist der Logik-Schicht zugeordnet. Diese ist die Zentrale des Systems und zugleich Schnittstelle zur Persistenzschicht (Datenhaltung). Letztere ist für die Speicherung und das Laden von Daten zuständig. Alle Tests, Suiten und Ergebnisse sowie Benutzerdaten werden hier gespeichert.

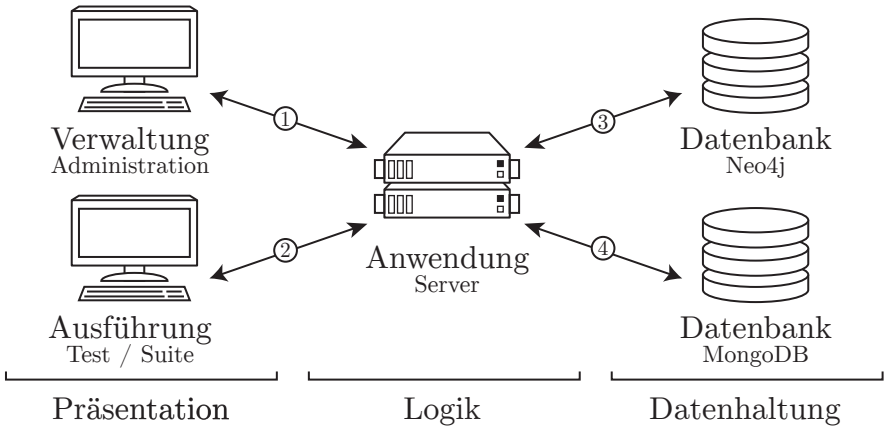

Abbildung 4.2: Systemarchitektur

Die folgende Auflistung dient der vertiefenden Erläuterung der Abbildung 4.2 und beschreibt die Pfade der Kommunikation.

1. Verwaltung und Anwendung
    a) Test-CRUD-Operationen

   b) Suite-CRUD-Operationen
   c) Ergebnis-RD-Operationen
2. Ausführung und Anwendung
   a) Test-Anfragen u. Antworten
   b) Suite-Anfragen u. Antworten
3. Anwendung und Datenbank (DB)-Neo4j
   a) Test-Anfragen
   b) Suite-Anfragen
4. Anwendung und DB-Mongo
   a) Ergebnis-Anfragen

4.3 Anwendungsarchitektur

Das Konzept der Architektur der Anwendung basiert auf vier primären Komponenten. Die Anwendung läuft auf einem zentralen Server und nutzt für die Speicherung zwei unterschiedliche Datenbanken auf verschiedenen Servern. Die Abbildung 4.3 stellt dies in einem Überblick dar.

4.3.1 Web-Server

Die Anwendung basiert auf dem Einsatz von Frameworks, die im Kapitel der Grundlagen (2) aufgeführt wurden. Express[2] wird hierbei unter anderem für das Ausliefern von statischen Inhalten eingesetzt. Hierzu zählen Inhalte wie Cascading Stylesheets (CSS) oder auch JavaScript-Dateien, die als externe Ressource eingebunden werden.
Diese Art von Inhalten bilden aber nicht den ausschließlichen Teil des Web-Servers. Vielmehr obliegt ihm auch die Aufgabe des Renderns von authentifizierungspflichtigen Inhalten. Das können unter anderem Views der Anwendung sein. Auf der Clientseite wird, wie im Abschnitt 4.1 der Grundlagen beschrieben, das Framework AngularJS[3] eingesetzt. Die Architektur sieht vor, dass auf diesem Weg viele Views beim Client gerendert werden. Die Daten der Tests, Suites und Ergebnisse werden ausschließlich über die Representational State Transfer (REST)-Schnittstelle abgerufen.
Teilbereiche der Anwendung unterliegen einer vorhergehenden Authentifizierung. Diese ist zwingende Voraussetzung für das Abrufen der zuvor beschriebenen Ressourcen. Auf Grundlage

[2] http://expressjs.com/
[3] https://angularjs.org/

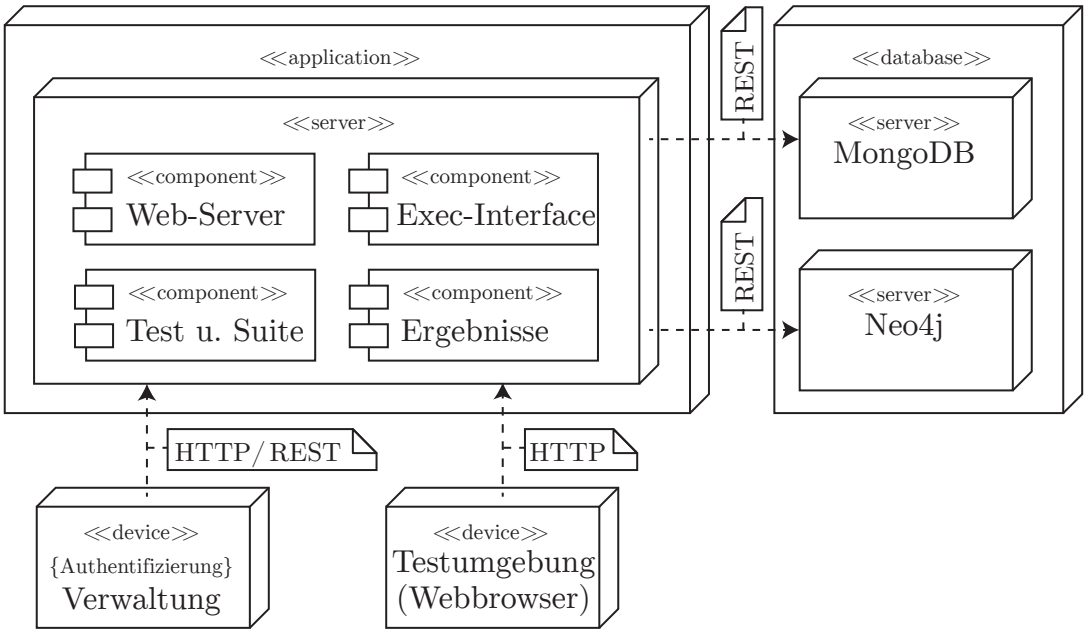

Abbildung 4.3: Anwendungsarchitektur

der Middleware-Architektur von Express (Siehe 4.1) ist die Überprüfung der Zugriffsberechtigung problemlos möglich. Die Administration der Anwendung setzt auf das Konzept des clientseitigen Renderns von Templates mit entsprechender Schnittstelle für die Datenbeschaffung. Somit können die Daten als auch die inhaltsleeren Templates geprüft und ausgeliefert werden. Die Aufgaben des Servers lassen sich in die folgenden Bereiche gliedern:

1. Ausliefern statischer Inhalte (z.B. Bildformate, CSS- u. JS-Dateien)
2. Organisation der Routes (private- u. public-Bereiche)
3. Rendern von Views (z.B. für Exec-Interface)
4. Authentifizierung

4.3.2 Test

Diese Komponente ist für die Verwaltung von Tests zuständig. Sie bietet eine REST-Schnittstelle nach außen, über die nach erfolgreicher Authentifizierung CRUD-Operationen ausgeführt werden können. Die Persistenz wird durch die Anbindung an die Datenbank realisiert.

4.3.2.1 Struktur

Die Analyse hat gezeigt, dass ein Test aus der Abhängigkeit zu weiteren Tests bestehen kann. Konkretisiert man diesen Sachverhalt an einem Beispiel, entsteht folgendes Szenario:
Die Überprüfung der Existenz einer Methode setzt die Existenz des Objekts voraus. Dieser Sachverhalt führt unweigerlich zu einer Kausalkette. Basiert die Überprüfung der Existenz eines Objekts auf weiteren Objekten, lässt sich dies am besten durch einen Graph modellieren. Die Abbildung 4.4 zeigt einen gerichteten Graphen von Tests.

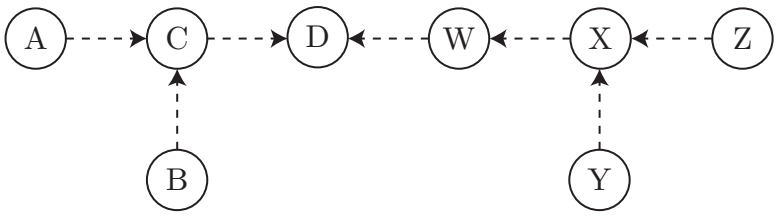

Abbildung 4.4: Gerichteter Graph von Tests mit dem Zieltest „D“

4.3.2.2 Schnittstelle

Die folgende Tabelle 4.1 beschreibt das Schema der Schnittstelle der Tests. Sie basiert auf dem REST-Protokoll.

Methode	**Schema**	**Beschreibung**
GET	/api/test	Abfragen aller Tests
GET	/api/test/:id	Abfragen des Tests mit der :id
POST	/api/test	Neuen Test erstellen
UPDATE	/api/test/:id	Test mit :id ändern
DELETE	/api/test/:id	Test mit :id löschen

Tabelle 4.1: Test REST-Schnittstelle

4.3.3 Suite

Die Struktur einer Suite ähnelt der eines Tests. Es gibt aber Unterschiede, die erst anhand ihrer Struktur erkennbar sind. Der Unterschied gegenüber der Testsequenz liegt in der Ordnung. Ein Beispiel wären eine Suite „JavaScript“mit jeweils zugeordneten Suiten „Version 1.0“, „Version 1.1“et cetera. Die jeweiligen Versionen haben zueinander keine Ordnung, sie geben lediglich Aufschluss über die direkte Zuordnung zu einer „Hauptsuite“.

4.3.3.1 Struktur

Die Abbildung 4.5 zeigt einen Beispielgraphen von Suiten ohne die Zuweisung von Tests.

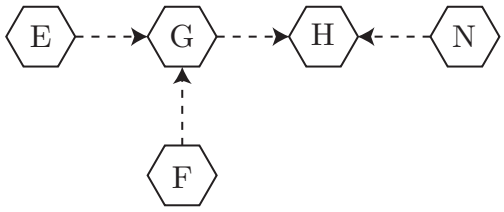

Abbildung 4.5: Gerichteter Graph von Suiten mit der Zielsuite „H“

4.3.3.2 Schnittstelle

Die folgende Tabelle 4.2 beschreibt das Schema der Schnittstelle der Suites. Sie basiert auf dem REST-Protokoll.

Methode	Schema	Beschreibung
GET	/api/suite	Abfragen aller Suites
GET	/api/suite/:id	Abfragen der Suite mit der :id
POST	/api/suite	Neue Suite erstellen
UPDATE	/api/suite/:id	Suite mit :id ändern
DELETE	/api/suite/:id	Suite mit :id löschen

Tabelle 4.2: Suite REST-Schnittstelle

4.3.4 Ergebnis

Die eindeutige Definition eines Ergebnisses ist ein noch ausstehender Bestandteil der Arbeit. Die Tests wurden bereits klar zu den Suiten hin abgegrenzt. Nun folgen die Ergebnisse, die sich wie folgt definieren:

Hauptsächlich sind Ergebnisse über die in der Tabelle 4.3 beschriebene Schnittstelle abrufbar. Sie geben Aufschluss über einen ausgeführten Test und unterscheiden fünf verschiedene Zustände.

4.3.4.1 Schnittstelle

Die folgende Tabelle 4.3 beschreibt das Konzept der Schnittstelle der Ergebnisse. Sie basiert auf dem REST-Protokoll. Zu beachten ist, dass ein Ergebnis nicht von außerhalb der Anwendung

dem Datenbestand hinzugefügt oder abgeändert werden kann. Es handelt sich um Daten, die bei dem Verarbeitungsprozess einer Testauswertung generiert werden. Das Ergebnis ist somit ein Artefakt, das die Anwendung und nicht ein Benutzer generiert.

Methode	Schema	Beschreibung
GET	/api/result	Alle Ergebnisse
GET	/api/result/:id	Das Ergebnis zu der angegebenen :id
DELETE	/api/result/:id	Das Ergebnis mit der :id löschen

Tabelle 4.3: Ergebnis REST-Schnittstelle

4.3.4.2 Zustände

Grundsätzlich kann ein Testergebnis zu verschiedenen Zuständen führen. Die Ausgangsbedingungen sind in jedem Fall dieselben, nur der Zustand variiert. Die verschiedenen Zustände geben dann direkt Aufschluss über den geprüften Sachverhalt oder lassen sich in Bezug auf die Abhängigkeiten interpretieren.

Erfolg Ein Test wird in der Verarbeitung als Erfolg gespeichert, wenn die „success-Funktion“ im Test-Code mit einem Infoparameter aufgerufen wurde. Dieser Zustand entspricht dem Idealfall bzw. Erfolgsfall.

Fehler Die „failure-Funktion“ im Test-Code bewirkt die Speicherung als „Fehlgeschlagen“. Diese Funktion sollte entgegen dem Erfolgsfall zur Anwendung kommen um keinen Konflikt mit anderen Zuständen zu provozieren.

Ergebnislos Der Zustand der Ergebnislosigkeit ergibt sich, sofern weder die „success“- noch „failure“-Funktion im Test-Code ausgeführt wurde. Trotzdem wurde aber eine Antwort zur Anwendung zurückgegeben.

Antwortlos Ein Ergebnis wird als „Antwortlos“ gespeichert, sofern ein Test abgerufen wurde, die Antwort aber ausblieb. Jeder Testaufruf wird gespeichert, sodass keine Antwort rückwirkend erkannt wird.

Abgebrochen Ein Test wird abgebrochen bzw. dieses Zustand ergibt sich, wenn vorherige Tests einer Testsequenz scheitern. Die Aussage ist hierbei ein vorzeitiges Scheitern, was in der Konsequenz einen Abbruch zur Folge hat.

4.3.5 Exec-Interface

Dieses Interface beschreibt den konkreten Aufruf eines Tests an der Anwendung. Nach außen muss hierbei die Möglichkeit bestehen einen Test eindeutig zu identifizieren. Der Aufbau der URL richtet sich nach folgendem Schema.

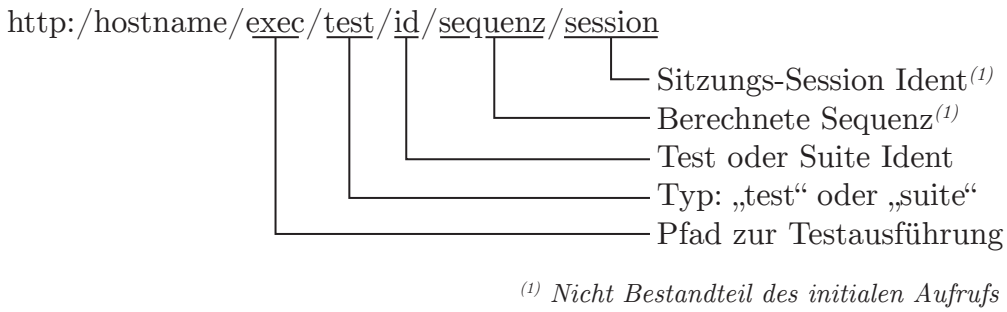

Abbildung 4.6: Test- und Suite-URL-Schema

Der initiale Aufruf beinhaltet weder den Sequenz-Pfad noch den Session-Pfad. Diese Parameter werden erst nach der Initialisierung durch Weiterleitung ergänzt. Eine Sequenz setzt sich aus Nummern zusammen, die durch Semikola getrennt werden. Jede Nummer identifiziert einen Test. Der Session-Pfad ist ein Universally Unique Identifier (UUID) und dient der Erkennung einer „Sitzungsabfrage“.

4.4 Testumgebung (Webbrowser)

Dieser Abschnitt beschreibt den konkreten Aufbau eines Test-Dokuments, das an einen Client ausgegeben wird. Hierfür werden die Erkenntnisse der Analyse aus Abschnitt 3.3.3.1 aufgegriffen und finden ihre Ausarbeitung im Konzept.

4.4.1 Dokumentenstruktur

Ein Test wird im Mantel eines HTML-Dokuments ausgeliefert. Externe Abhängigkeiten existieren nicht, sodass beim Rendering-Prozess kein Nachladen von weiteren Ressourcen stattfindet. Aus diesem Grund enthält der Dokumentenkopf keine testrelevanten Angaben - von der Enkodierung abgesehen.
Der Großteil eines Tests arbeitet ohne eine Visualisierung von Informationen. Weiterhin sind die einzigen sichtbaren Informationen für den Client von statischer Natur. Der Dokumenten-

körper enthält die im Folgenden genauer spezifizierten Bereiche der Datenerfassung sowie der dreifach aufeinanderfolgenden Script-Tags. Diese beiden Bereiche bilden den Kern eines Tests.

4.4.2 Datenerfassung

Die Speicherung von Informationen, die beim Client gesammelt werden, bilden einen wichtigen Teil eines Tests. In der späteren Verarbeitung bzw. Auswertung eines Tests fließen die gesammelten Informationen unter Umständen in die Bewertung (Ergebnis) mit ein.
Für diesen Zweck bieten sich der HTML-Form-Tag mit einhergehender Verwendung des HTML-Input-Tags an. Die Möglichkeit Input-Tags über den Attributstyp nicht sichtbar anzuzeigen dient der Beschreibung aus Abschnitt 4.4.1. Zu den Informationen, die in der Ausführungsumgebung beim Client gesammelt werden, gehören:

Test-Id Diese ID identifiziert den Test in der Datenbank. Sie fließt serverseitig beim Rendering bereits als Wert des Tags mit ein.

Navigator-Interface Das Interface enthält, je nach Implementation, verschieden viele Informationen. Für einen Test sind die folgenden Werte von besonderem Interesse:

Name	Beschreibung
appCodeName	Enthält einen Kurznamen des verwendeten Webbrowsers.
appVersion	Die Version des Webbrowsers
language	Die aktuelle Sprache in der Webbrowsereinstellung
platform	Name der Plattform der Webbrowsers
userAgent	Der User-Agent-String des Webbrowsers
oscpu	Name des Betriebssystems des Webbrowsers
onLine	Gibt den Onlinestatus als Boolean
languages	Objekt mit den unterstützten Sprachen des Webbrowsers
plugins	Ein Objekt, das die installierten Plug-Ins des Webbrowsers hält.
mime_types	Alle durch den Webbrowser unterstützten Mime-Types als Objekt

Tabelle 4.4: Navigator-Interface Informationen

Success und Failure Der Wert des Parameters der Funktion bei ihrem jeweiligen Aufruf. Der Wert wird somit vom Verfasser des Tests bestimmt.

4.4.3 Ternärer Script-Tag

Anschließend an das Formular kommen drei aufeinanderfolgende Script-Tags. Jeder dieser Tags hat eine spezielle Aufgabe. Getrennt sind die Tags auf der Grundlage der Analyse aus 3.3.

4.4.3.1 Initialisierung

Der erste Tag dient der Initialisierung des Tests - also der Vorbereitung. In dieser Phase werden die Informationen aus Abschnitt 4.4.2 gesammelt und in das Formular überführt. Dieser Vorgang geschieht per JavaScript und kann nur clientseitig ausgeführt werden.
Die Komplexität der Objekte „plugins“ und „mime_types“ bedingt die Verwendung von Hilfsfunktionen, die ein Objekt serialisieren und somit als String in einem Input-Tag speichern.

4.4.3.2 Ausführung

Bei der Generierung des Tests auf dem Server werden am Anfang des Skripts zwei Funktionen eingefügt. Hierbei handelt es sich um die „success“ und „failure“ Funktionen. Auf diese Weise sind die Funktionen im Geltungsbereich des Tests bekannt und können verwendet werden.
Als zweites wird der eigentliche Test direkt danach eingefügt. Dieser JavaScript-Code wird somit in direktem Anschluss interpretiert.

4.4.3.3 Rückführung

Der letzte Tag bildet den Abschluss bzw. das Versenden der gesammelten Informationen. Das automatische Versenden basiert auf Grundlage der Analyse aus Abschnitt 3.3.5. Anschließend kommt das Konzept der Testsequenz aus 3.3.7 zum Tragen, sofern weitere Tests in der Sequenz enthalten sind.

4.5 Verwaltung

Die Verwaltung der Applikation basiert im Wesentlichen auf dem Konzept des clientseitigen Renderns von Daten. Hierfür kommt AngularJS zum Einsatz. Der Web-Server bietet zwei Schnittstellen. Zum einen den HTTP-Zugang für das Ausliefern von Templates und die REST-Schnittstelle (auf HTTP basierend) für das Konsumieren und Produzieren von Daten. Aufgerufen wird die Verwaltung über einen Webbrowser. Nach erfolgreicher Authentifikation wird eine SSP durch den Bootstrapping-Prozess von AngularJS generiert. Alle weiteren Interaktionen des Benutzers haben somit einen verringerten Overhead, da Templates nur einmalig und Daten selektiv an dem Server angefragt werden.

Abbildung 4.7: Anwendungs-Verwaltungs-URL

4.5.1 Bootstrapping

Das clientseitige Verwalten der Daten findet über den Aufruf der Adresse aus Abbildung 4.7 statt. Die Anfrage wird serverseitig (wie im Abschnitt 4.3.1 beschrieben) auf Berechtigung geprüft.
Im Erfolgsfall wird ein HTML-Dokument als Antwort auf die Anfrage zurückgegeben. Die Einbindungsreihenfolge der statischen Inhalte sieht vor, dass im Dokumentenkopf alle statischen Inhalte angegeben und beim Renderungsprozess nachgeladen werden. Im Dokumentenkörper sind anschließend alle benötigten Bibliotheken definiert, die ebenfalls synchron nachgeladen und interpretiert werden. Als Beispiel sind hier „AngularJS“, „Bootstrap-UI“ und „D3JS“ zu nennen. Am Ende des Bootstrapping-Prozesses werden die Directiven, Filter, Factories und Controller für die Anwendung geladen. Sobald das Dokument den Vorgang des Renderns abgeschlossen hat, beginnt AngularJS die Initialisierungsphase auf Grundlage aller nun verfügbaren Ressourcen.

4.5.2 Routing

Da AngularJS dem Paradigma der SSP folgt, haben Routes kein komplettes Neuladen der Seite zur Folge. Sofern die HTML5-History-API zur Verfügung steht, wird diese dem Hashbang-Mode vorgezogen. Auf diesem Wege wird die beste Option gewählt (Panda, 2014, S.77).
Analog zu den Schnittstellen des Tests 4.1, der Suite 4.2 und den Ergebnissen 4.3 sind für die Verwaltung ähnliche Routes definiert. Die Tabelle 4.5.2 beinhaltet alle clientseitigen Routes.

4.5.3 Templating und Rendering

Das Framework AngularJS nutzt das SSP Paradigma, was im Kapitel der Grundlagen bereits erklärt wurde. Hierbei werden Templates und Daten gesondert beim Server erfragt und beim Client gerendert. Der daraus entstehende Vorteil ist eine Schonung der systemeigenen Ressourcen. Die Controller beinhalten Events für das Benutzerverhalten. Bei der Benutzeranfrage nach Ergebnissen werden die Daten asynchron im Hintergrund angefragt und anschließend beim Client gerendert. Weiterhin werden Daten aber auch für das Rendern von Graphen verwendet. Hierfür wurde die D3JS Bibliothek im Kapitel Grundlagen vorgestellt. Diese visualisiert unter

Schema	Beschreibung
/app/#/test/create	Test-Erstellung
/app/#/test/list	Abfrage von Tests
/app/#/test/view/:id	Abfrage eines Tests
/app/#/test/edit/:id	Bearbeitung eines Tests
/app/#/test/delete/:id	Löschen eines Tests
/app/#/suite/create	Suite-Erstellung
/app/#/suite/list	Abfrage von Suites
/app/#/suite/view/:id	Abfrage einer Suite
/app/#/suite/edit/:id	Bearbeitung einer Suite
/app/#/suite/delete/:id	Löschen einer Suite
/app/#/result/list	Abfrage von Ergebnissen
/app/#/result/view/:id	Abfrage eines Ergebnisses
/app/#/result/delete/:id	Löschen eines Ergebnisses

Tabelle 4.5: Routing-Pfade

Zuhilfenahme des SVG-Tags die Daten. Die komplette Berechnung und Darstellung dieser Graphen ist somit auch Aufgabe des Webbrowsers und belastet nicht die Serverleitung.

4.6 Datenbankarchitektur

Die Speicherung der Daten geschieht über zwei verschiedene Datenbanktypen auf zwei verschiedenen Servern. In diesem Abschnitt wird die Struktur und Architektur genauer beschrieben. Zum Einsatz kommt zum einen die Graphdatenbank „Neo4j“ und die Not only SQL (NOSQL)-Datenbank „MongoDB“. Beide bieten spezielle Eigenschaften, die ausschlaggebend für ihre Wahl sind.
Die beiden Abschnitte 4.3.2 und 4.3.3 haben bereits das Konzept der Tests und Suites beschrieben. Aus der Anforderung heraus einen Graphen zu modellieren und zu speichern, kommt Neo4j in Frage. Weiterhin ist die Interoperabilität von Neo4j nach Panzarino (2014) wichtig.
Die zweite Datenbank dient der Speicherung von Testergebnissen. Der Unterschied zu den Tests und Suiten definiert sich wie folgt: Testergebnisse repräsentieren einen „einfachen“ Sachverhalt ohne ein großes Abhängigkeits- und Beziehungsgeflecht. Außer der Session haben Ergebnisse keine Beziehung zueinander. Die gesammelten Informationen beim Client bzw. in seiner Umgebung entsprechen dem JavaScript Object Notation (JSON)-Format. Ein Objekt kann somit viele, aber auch wenige Informationen halten und ist leicht abfragbar. Weiterhin findet das Binary JavaScript Object Notation (BSON)-Format Verwendung, das auf dem JSON-Format

aufbaut. BSON baut auf Effizienz, Traversabilität und Performance (Chodorow, 2013, S.179). Diese Eigenschaften sind wichtig bei stark wachsenden Datenmengen.

4.6.1 MongoDB

Das Konzept der dokumentenbasierten Speicherung von Daten ist prädestiniert für Ergebnisse, die sich aus mehreren Feldern zusammensetzen.

4.6.1.1 Datenanbindung

Die Datenbank MongoDB bietet die Kommunikation über das „Wire Protocol“ an. Das zum Einsatz kommende Framework Mongoose[4] (4.1) nutzt dieses Protokoll.

4.6.1.2 Datenschema

Das Datenbankkonzept von MongoDB ist zwar schemafrei, clientseitig verwendet Mongoose aber sogenannte Schemata und Models für ein einfaches Arbeiten. Hierfür stehen die gängigen CRUD-Operationen zur Verfügung.

ID Die ID ist ein automatisch generierter Identifikator, der jedem neuen Dokument durch MongoDB zugewiesen wird.

Test ID Die Test ID ist eine Referenz auf den Identifikator des Tests, dessen Ergebnis das Dokument hält. Es beinhaltet somit eine Nummer, unter der ein Test in Neo4j auffindbar ist.

Session ID Jeder Test startet in einer Session. Hängen Tests voneinander ab, so bekommen alle Ergebnisse dieselbe Session. Weiterhin werden alle Ergebnisse einer Suite unter der ID vereinigt.

Executed Nachdem ein Test beim Client ausgeführt und das Ergebnis generiert wurde, bekommt dieses Feld das aktuelle Datum zugewiesen. Das Objekt dient der chronologischen Einordnung von Ereignissen bzw. Ergebnissen.

Navigator Dieses Objekt wird mit Informationen gefüllt, die vor der Testausführung gesammelt werden. Hierzu gehören per JavaScript ausgelesene Informationen des Navigator-Interface[5].

[4] http://mongoosejs.com/
[5] https://developer.mozilla.org/en-US/docs/Web/API/Navigator

Browser, Device, System Diese Felder bekommen beim Verarbeitungsprozess eines Tests Informationen zugewiesen. Sie stammen aus der User-Agent-Auswertung durch eine Bibliothek.

Success Die erfolgreiche Ausführung eines Tests bzw. sein Validieren führt Informationen in das Dokument zurück, die unter diesem Feld gespeichert werden.

Failure Das Pendant des Success-Objektes ist das Failure-Objekt. Hier wird der Wert gespeichert, der durch die Failure-Funktion eines Tests definiert wird.

Result Bei der Auswertung eines Tests können fünf verschiedene Zustände erreicht werden, die im Punkt 4.3.4.2 beschrieben sind. Dieser Zustand wird in diesem Feld gespeichert.

User-Agent Der User-Agent entspricht dem des Webbrowsers. Er ist Grundlage für die Bestimmung der Umgebung.

Beispieldokument Anhand des Listings lässt sich die Struktur des BSON-Dokuments verdeutlichen. Einige Werte sind mit drei Punkten abgekürzt. Hierbei ist der Inhalt an dieser Stelle nicht wichtig - also ausgeblendet.

4.6.2 Neo4j

Einleitend wurde bereits auf die Verflechtung von Tests und Suites hingewiesen. Daraus lässt sich folgendes Konzept erstellen.

4.6.2.1 Datenanbindung

Die Kommunikation zwischen der Anwendung und der Neo4j-Datenbank basiert auf der Neo4j-REST-Schnittstelle[6]. Diese ist die geeignete Wahl gegenüber der Shell.

4.6.2.2 Knoten

Es gibt zwei verschiedene Arten von Knoten. Die einen repräsentieren einen Test, der bereits in der Analyse beleuchtet wurde, die anderen eine Suite. In der Terminologie von Neo4j werden diese Knoten anhand von Labels unterschieden. Die beiden Label sind:

1. Test
2. Suite

[6] http://neo4j.com/

```
{
  "_id":         ObjectId("546c882d58ee4d930f1c492b"),
  "test_id":     25,
  "session_id":  "3fae9622-42f0-4343-a6a0-41eb9961e4ce",
  "executed":    ISODate("2014-12-10T12:10:13.541Z"),
  "navigator":   {
    "app_code_name":  ..., "app_name":  ..., "app_version": ...,
    "cookie_enabled": ..., "language":  ..., "platform":    ...,
    "user_agent":     ..., "oscpu":     ..., "product":     ...,
    "on_line":        ..., "languages": ..., "plugins":     ...,
    "mime_types":     ...
  },
  "browser":     {
    "patch": 2125, "minor": 0,  "major": 38, "family": "Chrome"
    },
  "device":      {
    "patch": 0,    "minor": 0,  "major": 0,  "family": "Other"
  },
  "system":      {
    "patch": 0,    "minor": 10, "major": 10, "family": "Mac OS X"
  },
  "success":     "",
  "failure":     "Object does not exists",
  "result":      "failure",
  "useragent":   "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_10_0)
      AppleWebKit/537.36 (KHTML, like Gecko) Chrome/38.0.2125.122
      Safari/537.36",
  "created":     ISODate("2014-12-10T12:08:13.541Z")
}
```

Listing 4.1: BSON-Beispieldokument

4.6.2.3 Attribute

Jeder Knoten hat definierte Attribute, die über einen Schlüssel weiterführende Informationen halten. Bei Testknoten sind die wichtigsten Attribute der Name (nicht mit der ID zu verwechseln) und der Code. Als Name ist ein möglichst eindeutiger Titel zu wählen. Der Code ist der eigentliche Test. Er enthält den JavaScript-Quelltext, der in der Umgebung des Clients ausgeführt wird.

4.6.2.4 Relationen

Verbunden werden diese Knoten über Relationen. Zu unterscheiden sind hierbei drei Arten von Relationen. Jede Relation bezieht sich auf die Aussage eines speziellen Sachverhalts. Zu unterscheiden sind die folgenden Relationen:

1. Related

2. PartOf
3. Required

Related Relationen werden als „Related“ gelabelt um einen Zusammenhang zwischen zwei Suiten anzugeben. Die Relation ist gerichtet und dient der Zuweisung von Suiten zueinander. Suiten derselben Ebene haben keine Ordnung. Das bedeutet bei mehreren Suiten (x), die einer Suite (y) zugeordnet sind, dass die x-Suiten untereinander keiner Ordnung folgen.

PartOf Jedem Suite-Knoten können Test-Knoten zugeordnet werden. Diese Relationen werden als „PartOf“ gelabelt. Sie geben an, dass ein Test unmittelbar Teil einer Suite ist. Diese Relation ist nur für eine direkte Beziehung zwischen einem Test und einer Suite zuständig.

Required Als „Required“ werden Relationen zwischen zwei Test-Knoten gelabelt. Hierbei können über mehrere Knoten hinweg diese Beziehungen eingesetzt werden, sodass Pfade aus Test-Knoten abfragbar sind. Dieses Konzept dient der Erstellung der Testsequenzen aus Abschnitt 3.3.7.

4.6.3 Beispielgraph

Das Verhalten der beschriebenen Konzepte lässt sich anhand der Abbildung 4.8 verdeutlichen. Dieser Graph basiert auf den Abbildungen 4.4 und 4.5. Hierbei ist zu beachten, dass die beiden Relationen zwischen „E“ und „A“ sowie „H“ und „D“ die neue Relation „PartOf“ beschreiben.

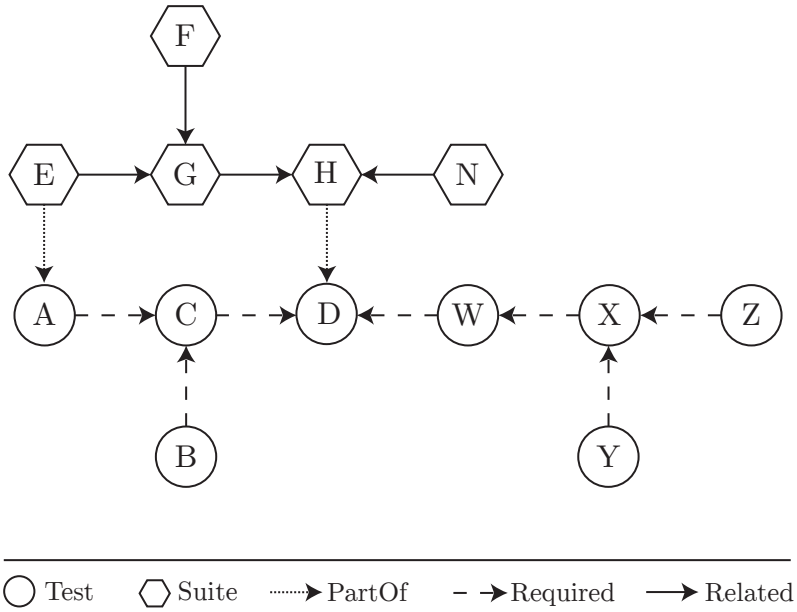

Abbildung 4.8: Beispielgraph von Tests und Suites

5 Entwicklung

5.1 Testsequenz

Die Kapitel Analyse und Konzeption haben den Aufbau von Tests dargestellt. Ein Test kann auf einem oder mehreren Tests basieren. Der daraus entstehende Graph hat somit einen „Zielknoten“, auf den unter Umständen mehrere Tests zeigen, die selbst auch dieser Möglichkeit unterliegen.

Dieser Abschnitt beschreibt die Verarbeitung eines Graphen zu einer Sequenz, die ausgeführt bzw. abgearbeitet werden kann. Am Anfang der Betrachtung steht die Abfrage an der Datenbank sowie die Modellierung der Antwort. Weiterhin wird der Algorithmus mit seinen Randbereichen beleuchtet. Betrachtet wird hierbei die Verarbeitung einer Suite-Anfrage, da diese einen höheren Grad an Komplexität aufweist als eine einzelne Test-Anfrage. Außerdem ist die Test-Abfrage Teil einer Suite-Anfrage.

5.1.1 Abfrage

Das Listing 5.1 zeigt die Abfrage einer Suite per Cypher in Neo4j. Zuerst wird der Knoten mit der Nummer 5 in der Vorbedingung einer Variablen (a) zugewiesen. Anschließend wird das Match-Pattern auf alle Knoten angewandt, die über ein Intervall von [0..n] Relationen zu „a“ in Beziehung stehen. Bis zur zweiten Zeile der Abfrage werden alle in Verbindung stehenden Suiten angefragt und der Variablen (b) zugewiesen. Die Reduktion der Suiten der dritten Zeile ist durch die Tatsache bedingt, dass keine Suiten doppelt in der Variablen „b“ vorkommen sollen. Die vierte Zeile bewirkt die Anfrage aller Tests, die direkt mit einer Suite aus „b“ in Verbindung stehen. Auch diese Ergebnismenge wird in der fünften Zeile eindeutig verarbeitet. Schlussendlich findet die Anfrage ihren Abschluss durch die Suche aller Pfade (p), die zu dem Test (c) führen. Die Rückgabe ist eine Liste aller Tests.

```
1 START a=node(5)
2 MATCH (a)-[:RELATED*0..]->(b:Suite)
3 WITH distinct b
4 MATCH (b)-[:PARTOF]->(c:Test)
5 WITH distinct c
6 MATCH p=(c)-[:REQUIRED*0..]->(d:Test)
7 RETURN nodes(p)
```

Listing 5.1: Suite-Cypher-Anfrage

5.1.2 Algorithmus

Das Ziel des Algorithmus ist es, die abgefragten Pfade in eine Ordnung zu bringen, die anschließend linear ausgeführt werden können. Hierbei sind einige Voraussetzungen zu beachten. Zuerst sollte in einer Testfolge kein Test doppelt ausgeführt werden. Wird ein Test in verschiedenen Pfaden referenziert, so würde er auch in der Sequenz doppelt vorkommen. Der Algorithmus verhindert das und führt diesen Test nur einmalig aus. Weiterhin sollte ein Zieltest nur dann ausgeführt werden, wenn alle seine Pfade gleichmäßig abgearbeitet werden.

Der Algorithmus im Listing 5.2 erfüllt diese Anforderungen. In der folgenden Erklärung werden die Begriffe „Index“ und „Tiefe“ synonym verwandt. Der Parameter „nodes“ ist ein Array, dessen Elemente jeweils die Pfade bilden. Über dieses Array wird in der dritten Zeile iteriert, indem bei der Verarbeitung jedes Elements auch die Pfadtiefe bekannt ist (depth). Sofern das neu geschaffene Array „paths“ in der Tiefe der momentanen Betrachtung noch keinen Array initialisiert hat, wird dieser Fakt nunmehr geschaffen (Zeile 5 bis 7). Die anschließende Schleife hat dann das Ziel zu untersuchen, ob die Variable „paths“ in den vorherigen Iterationen bereits den aktuellen Test beinhaltet. Diese Prüfung geschieht in den Zeilen acht bis dreizehn.

Die aktuelle Tiefe des aktuellen Pfades wird in dem neuen Array so lange auf Existenz geprüft, wie „paths“ diese Tiefe vorweisen kann (Zeile 8 bzw. 12). Sofern das der Fall ist und „paths“ an der Tiefe den Test nicht vorweisen kann, wird dieser hinzugefügt (Zeile 9 bis 11). Die neu erstellte Struktur ist ein Array, das alle Pfade auf Elemente eines Arrays aufteilt, ohne dass Tests redundant vorkommen können.

Die Nachbearbeitung ab der 16. Zeile hat das Ziel diese Neuordnung in eine lineare Folge zu übersetzen. Für diesen Zweck wird ein neues Array „sequence“ erstellt. Anschließend werden die einzelnen Elemente aus „paths“, bei denen es sich um Arrays handelt, an das Array „sequence“ angehängt. Abschließend werden die Elemente dieser neuen Struktur miteinander konkateniert. Als Trennzeichen fungiert ein Semikolon.

Auf diese Weise werden komplexe Abhängigkeiten von Tests und Suiten in eine Sequenz überführt, die anschließend nacheinander in der Umgebung eines Clients ausgeführt werden können.

```
1  function suiteToSequence(nodes) {
2      var paths = [];
3      nodes.forEach(function(path, index) {
4          path.forEach(function(node, depth) {
5              if (!paths[depth]) {
6                  paths[depth] = [];
7              }
8              while (paths[depth]) {
9                  if (paths[depth].indexOf(node.id) == -1) {
10                     paths[depth].push(node.id);
11                 }
12                 depth++;
13             }
14         });
15     });
16     var sequence = [];
17     paths.forEach(function(depthArray) {
18         sequence.concat(depthArray);
19     });
20     return sequence.join(";");
21 }
```

Listing 5.2: JavaScript-Sequenzalgorithmus

5.2 Ergebniszustand

Die Verarbeitung eines Testergebnisses beinhaltet unter anderem die Interpretation bzw. das einfache Auslesen des Testergebniszustands. Die in der Analyse und Konzeption entworfenen Zustandsmöglichkeiten haben hierbei die fünf Unterscheidungen „Erfolgreich“, „Fehlgeschlagen“, „Ergebnislos“, „Antwortlos“ und „Abgebrochen“ hervorgebracht. Diese Herleitung ist in der Anwendung wie folgt definiert:

Als „Erfolgreich“ wird ein Test gespeichert, wenn das Ergebnisfeld „Executed“ nicht leer ist - also bei der Testantwort gesetzt wurde. Weiterhin darf das Ergebnisfeld „failure“ keinen Wert enthalten. Dieses würde durch den Aufruf der Failure-Funktion im Testcode gesetzt. Zuletzt muss somit im Umkehrschluss die Success-Funktion ausgeführt worden sein, was eine Speicherung im „success“-Ergebnisfeld zur Folge hat.

Analog zur Auswertung des Erfolgsfalls verhält sich der Fehlerfall. Hierbei werden die Fakten auf umgekehrte Weise interpretiert. Zuerst muss der Test ausgeführt worden sein. Als Fehlerfall wird der Test aber nur dann gespeichert, wenn die Failure-Funktion und nicht die Success-Funktion ausgeführt wurde. Das Ergebnisfeld „failure“ muss also einen Wert enthalten.

Die als „Ergebnislos“ angesehene Ausführung und Speicherung ist folgendermaßen definiert: Ein Test muss ausgeführt worden sein, aber die beiden Ergebnisfelder „success“ und „failure“ dürfen keinen Wert enthalten. Dieser Zustand wird erreicht, wenn der Testcode beispielsweise Syntaxfehler enthält.

Der vierte Zustand „Antwortlos“ tritt ein, wenn die Ergebnisfelder „Executed“, „Success“ und „Failure“ keinen Wert enthalten. In diesem Fall wurde ein Test an der Anwendung angefragt, aber es wurde zu keinem Zeitpunkt eine Antwort zurückgesandt.

Der letzte Zustand wird durch die Anwendung geschaffen. Er tritt ein, wenn ein Test in einer Sequenz nicht erfolgreich ausgeführt wurde. Demnach müssen alle folgenden Tests der Sequenz abgebrochen werden.

5.3 Das Testdokument

Dieser Abschnitt beschreibt die konkrete Realisierung der Konzeption eines Testdokuments, das in der Umgebung eines Clients (Webbrowser) ausgeführt wird. Hierbei werden die Beziehungen zu der Analyse und Konzeption hergestellt, die ausschlaggebend für die schlussendliche Umsetzung waren.
Es sind vier Bereiche erklärungsbedürftig, die ein Testdokument ausmachen: der Bereich zur Speicherung - das Formular. Anschließend werden die drei Phasen eines Tests in Initialisierung, Ausführung und Versand untergliedert.

5.3.1 Grundgerüst

Jeder Test wird an den Client (Webbrowser) als ein komplettes HTML-Dokument ausgeliefert. Der Dokumententyp und das HTML-Skeleton (Head, Body) bilden das Gerüst eines Testdokuments. Der eigentliche Test befindet sich im „Body“ des Dokuments. Darin sind die eingangs erwähnten Bereiche Formulare, Initialisierung, Ausführung und Versand angesiedelt.

5.3.2 Formular

Das Formular ist für die Speicherung und den abschließenden Versand der gesammelten Informationen zuständig. In dem Listing 5.3 ist das Formular in Auszügen abgebildet. Die Input-Tags dienen als Speicherbereiche, die skriptgesteuert angesprochen und befüllt werden können. Beim serverseitigen Bereitstellen eines Testdokuments wird automatisch die ID des Tests eingefügt, sodass nach dem automatischen Versand der Test klar zugeordnet werden kann. Die Informationen des JavaScript-Navigator-Objekts werden in den Zeilen 3 bis 10 gespeichert. Zu erwähnen ist hierbei, dass nicht alle Input-Tags abgebildet sind, da sich diese nicht im Detail von den ersten drei Attributen App-Code-Name, App-Name und App-Version unterscheiden. Die Input-Tags der Zeilen 11 und 12 dienen der Speicherung der Informationen, die aus den beiden Testmethoden „success“ und „failure“ generiert werden. Siehe hierzu auch den Bereich der Ausführung.

```
1  <form name="result" method="post">
2      <input name="_id" value="54e630e0c34e6efe0ac73e71" type="hidden" />
3      <input name="app_code_name" type="hidden" />
4      <input name="app_name" type="hidden" />
5      <input name="app_version" type="hidden" />
6      <!--
7          Weitere Input-Tags:
8          cookieEnabled, language, platform, userAgent, oscpu,
9          product, onLine, languages, plugins, mimeTypes
10     -->
11     <input name="success" value="" type="hidden" />
12     <input name="failure" type="hidden" />
13 </form>
```

Listing 5.3: Testdokument-Formular

5.3.3 Initialisierung

In der ersten Zeile wird eine Referenz auf das Formular in einer Variablen gespeichert um dieses später adäquat ansprechen zu können. Bevor der Test, der durch den Benutzer geschrieben wurde, ausgeführt wird, werden umgebungsspezifische Informationen der Webbrowser gesammelt. Hierbei wird primär das JavaScript-Navigator-Objekt ausgelesen. Die gewonnenen Informationen dienen der späteren Bewertung und Ergänzung der Testergebnisse. Für diesen Zweck werden in den Zeilen 8 bis 22 die Daten im Navigator-Objekt abgerufen und, sofern vorhanden , im Formular gespeichert. Die Zeilen 18 bis 22 sind - wie im vorherigen Abschnitt der Beschreibung des Formulars - nicht im Detail mitaufgenommen. Der gesamte Quellcode

der Initialisierung steht im Testdokument in einem Script-Tag. Dieser folgt unmittelbar auf den Form-Tag.

```
1  var __form = window.document.result;
2
3  function setup() {
4      var __navigator = window.navigator;
5
6      // appCodeName, appName und appVersion
7      if (__navigator.appCodeName) {
8          __form.app_code_name.value = __navigator.appCodeName;
9      }
10     if (__navigator.appName) {
11         __form.app_name.value = __navigator.appName;
12     }
13     if (__navigator.appVersion) {
14         __form.app_version.value = __navigator.appVersion;
15     }
16
17     /*
18         Weitere Attribute eines Tests
19         cookieEnabled, language, platform, userAgent, oscpu, product,
20         onLine, languages, plugins, mimeTypes
21     */
22 }
23 setup();
```

Listing 5.4: Startphase

5.3.4 Ausführung

Der zweite Script-Tag umfasst den Quellcode im Listing 5.5. Hierbei werden in den Zeilen 1 bis 7 die erwähnten Methoden „success“ und „failure“ definiert. Diese Methoden sind in jedem Test eines Benutzers verfügbar und dienen der Speicherung von Informationen in den entsprechenden Form-Input-Tags. Ab Zeile 9 steht der durch den Benutzer geschriebene Test. Dieser wird beim Prozess der Generierung eines Testdokuments an der entsprechenden Stelle eingefügt. Das clientseitige Rendern des gesamten Testdokuments durch den Webbrowser ist die Stelle der Ausführung, an der Informationen gewonnen werden. Diese Informationen tragen zu der Bewertung des Ergebniszustandes in Abschnitt 5.2 bei.

5.3.5 Versand

Am Ende werden die gewonnenen Informationen zu der Anwendung versandt. Dies geschieht automatisch durch die skriptgesteuerte Methode „submit“. Diese Methode wurde im Kapitel

```
1  function success(text) {
2      __form.success.value = text || '';
3  };
4
5  function failure(text) {
6      __form.failure.value = text || '';
7  };
8
9  if (window.JSON) {
10     success(window.JSON);
11 } else {
12     failure('JSON object does not exist');
13 }
```

Listing 5.5: Testphase

Analyse in seiner Funktion beleuchtet. Zu beachten ist in der fünften Zeile, dass eine Latenzzeit vor dem Versand definiert ist. Das Verwenden der Methode „setTimeout“ hat zur Folge, dass das Formular erst nach - in diesem Fall einer Sekunde - versandt wird. Grund hierfür ist die Vermeidung von Problemen durch Überlastung eines Webbrowsers. Obwohl jedes Testdokument im Verhältnis zu gängigen HTML-Dokumenten extrem klein ist, können trotzdem Probleme bei der Performance eines Webbrowsers auftreten. Auf diesem Weg wird dem Webbrowser die Möglichkeit zum „Luftholen“ gegeben.

```
1  function setdown() {
2      __form.submit();
3  }
4
5  window.setTimeout("setdown()", 1000);
```

Listing 5.6: Versandphase

6 Bewertung

In diesem Kapitel wird ein Fallbeispiel unter Benutzung der Anwendung erstellt und im Detail erläutert. Dabei werden alle Bereiche der Anwendung und des Systems im Zusammenspiel gezeigt und erklärt. Das Beispiel beginnt bei der Authentifizierung an der Anwendung und endet mit der Bewertung der Ergebnisse, die in verschiedenen Webbrowser-Umgebungen gewonnen werden. Weiterhin werden die Erkenntnisse dieser Arbeit beschrieben.

6.1 Komplettes Fallbeispiel

Dieser Abschnitt beschreibt den Prozess der Erstellung eines Tests für dessen Ausführung in verschiedenen Webbrowser-Umgebungen. Dabei werden alle Phasen und deren Auswirkungen an der Anwendung beschrieben. Zuerst werden Vorbedingungen definiert, die für einen erfolgreichen Durchlauf vorherrschen müssen. Anschließend wird auf die Phase der Erstellung einer Suite mit anschließender Testerstellung eingegangen. Im Weiteren wird die Ausführung in verschiedenen Webbrowser-Umgebungen erläutert mit der abschließenden Bewertung des Ergebnisses und einer Zusammenfassung.

6.1.1 Vorbedingung

Zuerst muss die Anwendung auf einem Server erfolgreich gestartet werden. Hierbei ist zu beachten, dass alle externen Datenbanken erreichbar sein müssen. Zum einen betrifft dies die Instanz der Mongo-Datenbank, die Benutzerkonten verwaltet, sowie Neo4j bei den Tests und Suiten.
Vor dem Start der Anwendung müssen alle notwendigen Bibliotheken geladen werden, was durch das Kommando „npm init“gewährleistet wird. Anschließend ist beim Start der Anwendung darauf zu achten, dass die Environment-Variable „NODE_ENV“ auf „production“ gesetzt ist. Dies ist für eine performante Ausführung von „Express“ wichtig. Nach dem erfolgreichen Start kann die Anwendung über den Hostnamen des Servers aufgerufen werden. Dies betrifft zunächst die Verwaltung, die für die Test- und Suite-Erstellung zuständig ist.

6.1.2 Suite-Erstellung

Für die Erstellung einer Suite muss zuerst das entsprechende Formular aufgerufen werden. Nun besteht die Möglichkeit eine Suite mit den folgenden Angaben zu erstellen: Name, Beschreibung und Suite (Referenz bzw. Relation). Der Name sollte eindeutig sein, sodass der Inhalt klar ersichtlich ist. In unserem Beispiel erstellen wir zunächst die Suite „Object“ . Anschließend wird eine weitere Suite „Method“ erstellt, wobei diese Suite eine Referenz auf die „Object“-Suite zugewiesen bekommt und somit „Teil“ von dieser wird. Grund hierfür ist eine etwaige Ausführung aller Tests, die den jeweiligen Suiten später zugeordnet werden. Das Ergebnis dieser Phase ist in Abbildung 6.1 ersichtlich.

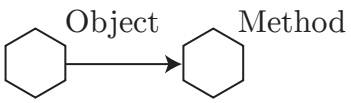

Abbildung 6.1: Suiten „Object“ und „Method“

6.1.3 Test-Erstellung

Nun folgt der eigentliche Kern der Anwendung - das Erstellen eines Tests. Dieser wird wie eine Suite über ein Formular an der Anwendung generiert. Hierfür sind folgende Angaben notwendig: Name, Beschreibung, Test (Referenz bzw. Relation), Suite (Referenz bzw. Relation) und Quellcode. Ebenso wie der Name einer Suite sollte auch der Name eines Tests eindeutig dessen Sinn wiedergeben. In diesem Beispiel wird die Verfügbarkeit des „JSON-Objekts“ getestet. Aus diesem Grund wird der Test „JSON“ genannt. Als Voraussetzung für dessen Ausführung muss kein anderer Test referenziert werden. Nun kommt eine der zuvor im Beispiel referenzierten Suiten zum Tragen. Die „Object“-Suite wird als Referenz für den JSON-Test angegeben. Nun zum Kern des Tests - dem Quellcode. Da nur die Existenz des Objektes überprüft werden soll, besteht der Test aus einer einfachen IF-Kondition. Hierbei wird überprüft, ob das JSON-Objekt im aktuellen Kontext bekannt ist. Im Erfolgsfall wird als Ergebnis eine serialisierte Information des Objekts gespeichert. Der Fehlerfall speichert eine informelle Beschreibung des Problems. Das Listing 6.1 zeigt den Quellcode.

Es existieren auf dem JSON-Objekt im Normalfall zwei Methoden (parse, stringify). Sofern nun auch die Existenz dieser Methoden auf dem Objekt überprüft werden muss, wird ein weiterer Test benötigt. Anders als beim ersten Test muss der zweite Test aber eine Referenz auf den ersten halten, da das JSON-Objekt zwangsläufig die Voraussetzung für den zweiten

```
1 if(window.JSON){
2     success(window.JSON);
3 }else{
4     failure('JSON object does not exist');
5 }
```

Listing 6.1: JSON-Test-Quellcode

Test ist. Es wird nun die Existenz der Methode „parse“ als Test definiert. Das Listing 6.2 zeigt die Überprüfung der Methode „parse“ .

```
1 if(window.JSON.parse){
2     success(window.JSON.parse);
3 }else{
4     failure('JSON.parse object does not exist');
5 }
```

Listing 6.2: JSON.parse()-Test-Quellcode

Das Ergebnis der neu erstellten Tests und Suiten ist ein Graph, der aus zwei Suiten und zwei Tests besteht. Die „Object“-Suite ist über die „Relation“-Beziehung mit der „Method“-Suite verbunden. Der JSON-Test ist mit dem Test der Parse-Methode durch eine „Required“-Beziehung in Verbindung. Weiterhin sind nun die Tests den jeweiligen Suiten zugeordnet, die ihrer Struktur bzw. ihrer Aufgabe am besten entsprechen. Verbunden sind sie durch die „Part-Of“-Beziehung. Es ergeben sich folgende Tests und Suiten in Abbildung 6.2.

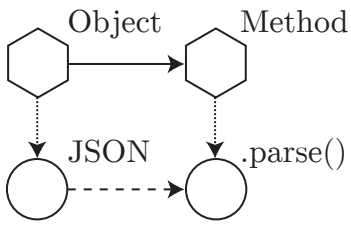

Abbildung 6.2: Tests und Suiten

6.1.4 Webbrowser-Ausführung

Die Ausführung der zuvor erstellten Tests ist Teil des nächsten Abschnitts. Hierbei kommen unterschiedliche Webbrowser zum Einsatz um unterschiedliche Ergebnisse zu erhalten. Die Abbildung 6.2 hat bereits gezeigt, wie die Tests und Suiten zusammenhängen, die getestet wer-

den sollen. Abhängig von dem Einstiegspunkt bzw. dem Ausführen eines Tests oder einer Suite entstehen unterschiedliche Testsequenzen. Diese Tatsache ist aus der Konzeption ersichtlich. Im aktuellen Beispiel ist die Suite „Object“ die zu testende Suite bzw. die Basis für die Berechnung der Testsequenz. Der eingangs gezeigte Graph führt zu einer Testsequenz bestehend aus zwei Tests. Der Test „JSON“ ist die zwingende Voraussetzung für den darauf folgenden Test (parse). Die tabellarische Aufstellung 6.1 ist das Ergebnis der entsprechenden Ausführungen in verschiedenen Webbrowser-Umgebungen:

System	Webbrowser	Version	Status
OS X	Apple Safari	8.0.3	Erfolgreich
OS X	Opera Software	27.0.1689	Erfolgreich
OS X	Opera Software	10.1.0	Fehlgeschlagen
OS X	Mozilla Firefox	3.0.13	Fehlgeschlagen
OS X	Mozilla Firefox	35.0.1	Erfolgreich
OS X	Google Chrome	40.0.2214	Erfolgreich
Windows XP	Internet Explorer	6.0.0	Fehlgeschlagen
Windows 7	Internet Explorer	7.0.0	Fehlgeschlagen

Tabelle 6.1: Ergebnisse des Webbrowser-Fallbeispiels

6.1.5 Bewertung des Ergebnisses

Die Tabelle 6.1 gibt einen Überblick über die Ergebnisse der Ausführung der Object-Suite. Die Spalte des Status beschreibt den Zustand, unter dem die Suite gespeichert wurde. Die verschiedenen Zustandsarten wurden unter 5.2 bereits erklärt. Zu beachten ist somit, dass die Ausführung bei z.B. Opera in der Version 10.1 fehlschlug. Da bereits der erste Test (JSON) fehlschlug, wurden alle darauf folgenden Tests abgebrochen. Anders hingegen verhält es sich bei der Version 27.0, die erfolgreich durchlaufen wurde. Hierbei waren sowohl der Test JSON als auch der Test der Parse-Methode erfolgreich.

6.1.6 Abschließende Betrachtung

Dieses Beispiel zeigt die Einsatzmöglichkeiten der Anwendung in Auszügen. Es wurden lediglich ein paar Webbrowser verschiedener Versionen und Plattformen getestet. Die gewonnenen Informationen basieren zu diesem Zeitpunkt somit auf verhältnismäßig wenigen Webbrowser-Umgebungen. Würde ein solches Projekt durch eine Community getragen bzw. die Basis der Webbrowser-Umgebungen durch diese erweitert, so würde auch der Gehalt der gesammelten Informationen steigen. Durch Virtualisierung von Webbrowser-Umgebungen kann, anders als

bei der Community, ein ähnlicher Informationsgewinn erzielt werden. Hierbei können Tests in einer Vielzahl von Umgebungen bei verhältnismäßig geringem Aufwand ausgeführt werden.

6.2 Erkenntnisse der Arbeit

Die gewonnenen Erkenntnisse dieser Arbeit lassen sich in drei Bereiche untergliedern. Zum einen wurden Informationen durch den Prozess der Analyse und Entwicklung gewonnen, die zu der Gesamtheit aller Erkenntnisse beitragen. Zum anderen lassen sich aber auch die beiden Bereiche der Anwendung sowie der eigentlichen Ausführungsumgebung (Webbrowser) unterscheiden.

6.2.1 Allgemein

Die Skriptsprache JavaScript hat sowohl auf der Anwendungsseite als auch in der Ausführungsumgebung die zentrale Rolle eingenommen. Sowohl als Mittel der Realisierung einer Konzeption als auch als Gegenstand der zu untersuchenden Sprache. Beide Teilbereiche lassen sich klar voneinander trennen und weisen doch die Gemeinsamkeit der verwendeten Sprache auf. Die Untersuchung der Sprache mit Hilfe selbiger ist somit scheinbar ein Paradoxon. Die Erkenntnis ist aber gegenteilig. Die clientseitige Untersuchung der Sprache weist durch die Implementation im Webbrowser andere Eigenschaften auf als die serverseitige Verwendung (Node.js).

6.2.2 Anwendung

Die Konzeption der Arbeit sieht die Teilung der Anwendung in vier Komponenten vor. Die Ausführung der Anwendung lässt diese Komponenten in einem Prozessraum laufen, sodass die Kommunikation jeweils ohne extra TCP-Sockets läuft. Dieser Eigenschaft zuträglich ist die Abhängigkeit der Komponenten bzw. die Nähe zu den einzelnen Aufgaben.

6.2.3 Webbrowser

Die Analyse des Verhaltens von Webbrowsern bezüglich der Implementation von JavaScript war Teil der Grundlagen, Analyse und Konzeption dieser Arbeit. Die Information über den Sprachumfang in einem Webbrowser ist definiert durch die Tests, die ein Benutzer an der Anwendung schreibt bzw. zur Ausführung bringt. Etwaige Besonderheiten einzelner Webbrowser (gleicher Plattform und Version) können Anomalien hervorrufen. Der Wahrheitswert

eines Tests steigt somit durch die Reproduktion eines Testergebnisses von unabhängigen Ausführungen bei gleicher Konfiguration.

6.2.3.1 NPAPI

Das Netscape Plug-In Application Programming Interface (NPAPI) - also eine Schnittstelle für Webbrowser-Plug-Ins - läuft außerhalb der Sandbox eines Webbrowsers. Somit unterliegt es den Benutzerberechtigungen (Alcorn u. a., 2014, S.326). Die Schnittstelle wurde 1995 eingeführt und fand z.B. bis 2012 beim bekannten Flash-Plug-In Verwendung. Eine Eigenschaft dieser Schnittstelle ist die Möglichkeit aus Plug-Ins auf Funktionen in JavaScript zuzugreifen. Auf diese Weise können Wechselwirkungen zwischen JavaScript und einem Plug-In auftreten.

6.2.3.2 Redundanz von Ergebnissen

Die Ausführung eines Tests in einem Webbrowser auf einem Betriebssystem führt zu einem Testergebnis. Die Wiederholung auf demselben System muss zum gleichen Ergebnis führen. In diesen Fällen untermauern die redundanten Ergebnisse dessen Aussage (Erfolgsfall oder Fehlerfall), vorausgesetzt die Webbrowser unterliegen derselben Version und Konfiguration. Eine hohe Anzahl gleicher Ergebnisse stärkt somit die Aussagekraft.

6.2.3.3 Anomalien von Ergebnissen

Der Unterschied zwischen zwei vermeintlich gleichen Webbrowsern kann durch Plug-Ins entstehen. Hierbei können Anomalien bei Testergebnissen auftreten, die auf eine veränderte JavaScript-Ausführungsumgebung zurückzuführen sind. Die in Abschnitt 6.2.3.1 beschriebene Möglichkeit der Kommunikation zwischen Plug-In und JavaScript ist hierbei der ausschlaggebende Punkt. Die JavaScript-API könnte erweitert oder schlicht verändert werden. Die Ausführung eines Tests kann somit bei gleicher Webbrowser-Version und gleichem Betriebssystem zu unterschiedlichen Ergebnissen führen.

6.2.3.4 Vermeidung und Bewertung

Die Anomalien von Testergebnissen in Abschnitt 6.2.3.3 können in der Entstehung nicht unterbunden werden. Ausschließlich in der Nachbereitung oder Wartung der Daten kann Einfluss darauf genommen werden. Hierbei kommt die Häufigkeit der Ausführung eines Tests zum Tragen. Bei einem großen Datenbestand (viele Testergebnisse) können verantwortliche Plug-Ins identifiziert werden. Hierzu würden die clientseitig ausgelesenen Informationen der Plug-Ins (5.3.3) herangezogen und analysiert werden.

Bei nur einem oder wenigen Testergebnissen kann ein System mit Webbrowser möglicherweise anfänglich falsch bewertet bzw. eingeordnet werden. Um diesen Fehler in den Daten zu erkennen wären viele Ausführungen dieses Tests in „sauberen“-Umgebungen notwendig.

7 Zusammenfassung

In diesem Kapitel wird ein Überblick über die zentralen Aspekte dieser Arbeit gegeben.

Am Anfang dieser Arbeit wird die Motivation, aus der heraus das Problem entstand bzw. sich das Thema dieser Arbeit entwickelt hat, beschrieben. Weiterhin wird das Ziel der Arbeit genauer bestimmt. Damit einhergehend wird das Thema der Arbeit abgegrenzt, um etwaige Bereiche am Rande des Themenkomplexes explizit einzubeziehen oder auszuschließen.

Die Grundlagen dieser Arbeit finden sich in dem zweiten Kapitel wieder. Hier wird die Entstehung der eingesetzten Technologien und Sprachen beleuchtet mit der einhergehenden Beschreibung ihrer Kernfunktionalitäten. Insbesondere wird auf JavaScript in der Umgebung des Clients besonders eingegangen, da diese Skriptsprache einen großen Teil der Arbeit ausmacht. Im weiteren Verlauf wird auf die Plattform Node.js eingegangen. Diese bildet das Rückgrat der entwickelten Anwendung und bedarf somit einer einführenden Beschreibung. Abschließend finden ausgewählte Frameworks und Bibliotheken Erwähnung mit der Beschreibung ihrer Kernfunktionalitäten.

Die Untersuchung der Voraussetzungen für die Entwicklung der Anwendung finden sich im dritten Kapitel der Analyse wieder. Zuerst werden die wichtigsten Begriffe dieser Arbeit definiert. Darauf folgt die Beschreibung des Kontextes, den die Anwendung schafft. Das eigentliche Ziel der Arbeit - die Untersuchung des JavaScript-Sprachkerns - findet sich in der Analyse. Dabei wird die Seite des Clients mit seinen entscheidenden Komponenten genau analysiert. Weiterhin werden auch Eigenschaften wie die Test-Automatisierung als Ziel des späteren Konzepts betrachtet. Abschließend werden in dem Kapitel dann Anwendungsfälle auf der Seite des Clients, des Servers sowie der Testumgebung beschrieben. Einhergehend mit den Testfällen finden auch Anforderungen an das System Erwähnung.

Die konkrete Ausarbeitung und Beschreibung der Anwendung ist Teil des vierten Kapitels. Zuerst wird der Systemkontext beschrieben mit der anschließenden Auseinandersetzung der Anwendungsarchitektur. Die vier Hauptbereiche Web-Server, Suiten und Tests, Ergebnisse

sowie Test-Ausführungsinterface stehen hierbei im Fokus der Betrachtung. Insbesondere deren Struktur und Schnittstellen werden ausführlich beschrieben. Der entscheidende Teil der Testausführung bildet einen großen Teil des Kapitels. Dieser Bereich wird von der Initialisierung bis hin zur Speicherung in dem Kapitel „Konzept“ beschrieben. Die Konzeption der Verwaltung der Anwendung findet anschließend Erwähnung. Abschließend wird das Datenmanagement beschrieben. Insbesondere wird auf die beiden Datenbanken „MongoDB“ und „Neo4j“ eingegangen.

Zum Schluss befasst sich diese Arbeit mit der Beschreibung der konkreten Realisierung der Anwendung. Der Algorithmus für die Berechnung einer Testsequenz wird eingehend beschrieben. Die Herleitung der Ergebniszustände folgt im Anschluss. Danach wird das konkrete Testdokument beleuchtet, das im Hinblick auf Grundgerüst, Formular, Initialisierung, Ausführung und Versand erklärt wird. Hierbei steht die Umgebung des Webbrowsers im Mittelpunkt der Betrachtung.

Die Arbeit schließt mit einem Fallbeispiel, bei dem die Phasen der Testerstellung bis hin zur Bewertung der Ergebnisse im Vordergrund stehen. Die Erkenntnisse der Arbeit bilden den Abschluss.

8 Ausblick

Dieses Kapitel gibt einen Ausblick auf weitere Möglichkeiten, die noch aus dieser Arbeit erwachsen könnten.

Der Fokus der bisherigen Arbeit lag auf der Entwicklung und Realisierung einer Anwendung, die Informationen über den Umfang von JavaScript generiert. Hierbei waren die Tests, Suiten sowie die Ergebnisse im Mittelpunkt der Entwicklung und Betrachtung. Eine weiterführende Analyse der gewonnenen Informationen war somit kein Bestandteil der Arbeit.

Unter der Annahme, dass ein Datenbestand durch die entwickelte Anwendung vorliegt, der Aufschluss über die Existenz von Objekten in verschiedenen Webbrowsern gibt, ergibt sich folgende Möglichkeit: Objekte in Webbrowsern entscheiden über die erfolgreiche Ausführung von Skripten in selbigen. Existieren Objekte nicht, schlagen somit die Skripte fehl. Auf dieser Grundlage wäre ein „Validator“ denkbar, der den eingangs beschriebenen Datenbestand als Basis für die Untersuchung von Skripten nutzte. Auf diese Weise könnte ohne konkrete Ausführung eines Skripts eine Vorhersage über den Erfolg der Ausführung in verschiedenen Umgebungen getroffen werden.

Eine weiterführende Möglichkeit besteht in einer statischen Bewertung von Skripten. Eingangs wurde die Möglichkeit beschrieben Skripte zu untersuchen. Die Quintessenz wäre eine Nachbearbeitung des Skripts. Umgekehrt würde aber eine präventive Entwicklung effektiver sein. Denkt man an eine IDE, die statisch während der Entwicklung bereits auf den Pool an Informationen zugreifen könnte, so würde die Phase der Entwicklung optimiert.

Die Gewinnung von Informationen wäre wahrscheinlich der zentrale Bestandteil einer weiterführenden Überlegung. Hierbei würde die bereits bestehende Anwendung um weitere Funktionalitäten ergänzt, sodass die Aussagekraft der Daten am Ende stiege. Dabei stünden stets die Daten im Mittelpunkt.

Anhang

8.1 Weitere Anwendungsfälle

Im Folgenden sind die weiteren Anwendungsfälle der Analyse aufgeführt.

Name	**Test anzeigen**
Beschreibung	Das Anzeigen eines Tests über die Anwendung
Beteiligte Akteure	Administrator, Nutzer
Verwendete Anwendungsfälle	Test erstellen
Ergebnis	Testinformationen werden visuell aufbereitet und dargestellt.
Vorbedingungen	Ein bereits erstellter Test durch den Administrator
Standardablauf	1. Ein Test wird anhand seiner ID aufgerufen. 2. Der Test wird visuell aufbereitet und dargestellt.

Tabelle 8.1: Anwendungsfall: Test anzeigen

8.2 Inhalt der CD-ROM

Der Inhalt der CD-ROM ist in durch folgende Verzeichnisstruktur ersichtlich:

Thesis Die komplette Arbeit als PDF-Dokument

Literatur Abbildungen der Online-Quellen

Anwendung Der Quellcode der entwickelten Anwendung

Name	**Test ändern**
Beschreibung	Das Ändern eines Tests über die Anwendung
Beteiligte Akteure	Administrator
Verwendete Anwendungsfälle	Test erstellen
Ergebnis	Ein Test wurde geändert und gespeichert.
Vorbedingungen	Benutzerkonto für Authentifizierung am System und ein bereits erstellter Test durch den Benutzer
Standardablauf	1. Der Benutzer authentifiziert sich am System. 2. Ein Test wird anhand seiner ID aufgerufen. 3. Das Formular für die Bearbeitung eines Tests wird aufgerufen. 4. Es wird ein eindeutiger Name angegeben (Optional mit Beschreibung). 5. Dem Test werden ggf. Vortests zugewiesen. 6. Dem Test werden ggf. Suites zugewiesen. 7. Der Test wird in JavaScript formuliert. 8. Der Test wird abgesandt und gespeichert.

Tabelle 8.2: Anwendungsfall: Test ändern

Name	**Test löschen**
Beschreibung	Das Löschen eines Tests über die Anwendung
Beteiligte Akteure	Administrator
Verwendete Anwendungsfälle	Test erstellen
Ergebnis	Ein Test wurde gelöscht.
Vorbedingungen	Benutzerkonto für Authentifizierung am System und ein bereits erstellter Test durch den Benutzer
Standardablauf	1. Der Benutzer authentifiziert sich am System. 2. Ein Test wird anhand seiner ID aufgerufen. 3. Eine Sicherheitsabfrage für das Löschen des Tests wird angezeigt. 4. Die Löschung des Tests wird durch Benutzeraktion gewählt. 5. Der Test wird gelöscht.
Alternative Ablaufschritte	4.1 Die Aktion wird verworfen.

Tabelle 8.3: Anwendungsfall: Test löschen

Name	**Suite anzeigen**
Beschreibung	Das Anzeigen einer Suite über die Anwendung
Beteiligte Akteure	Administrator, Nutzer
Verwendete Anwendungsfälle	Suite erstellen
Ergebnis	Suiteinformationen werden visuell aufbereitet und dargestellt.
Vorbedingungen	Eine bereits erstellte Suite durch den Administrator.
Standardablauf	1. Eine Suite wird anhand ihrer ID aufgerufen. 2. Die Suite wird visuell aufbereitet und dargestellt.

Tabelle 8.4: Anwendungsfall: Suite anzeigen

Name	**Suite ändern**
Beschreibung	Das Ändern einer Suite über die Anwendung
Beteiligte Akteure	Administrator
Verwendete Anwendungsfälle	Suite erstellen
Ergebnis	Eine Suite wurde geändert und gespeichert.
Vorbedingungen	Benutzerkonto für Authentifizierung am System und eine bereits erstellte Suite durch den Benutzer
Standardablauf	1. Der Benutzer authentifiziert sich am System. 2. Eine Suite wird anhand ihrer ID aufgerufen. 3. Das Formular für die Bearbeitung einer Suite wird aufgerufen. 4. Es wird ein eindeutiger Name angegeben (Optional mit Beschreibung). 5. Der Suite werden ggf. Vorsuites zugewiesen. 6. Die Suite wird abgesandt/gespeichert.

Tabelle 8.5: Anwendungsfall: Suite ändern

Name	**Suite löschen**
Beschreibung	Das Löschen einer Suite über die Anwendung
Beteiligte Akteure	Administrator
Verwendete Anwendungsfälle	Suite erstellen
Ergebnis	Eine Suite wurde gelöscht.
Vorbedingungen	Benutzerkonto für Authentifizierung am System und eine bereits erstellte Suite durch den Benutzer
Standardablauf	1. Der Benutzer authentifiziert sich am System. 2. Eine Suite wird anhand ihrer ID aufgerufen. 3. Eine Sicherheitsabfrage für das Löschen der Suite wird angezeigt. 4. Die Löschung der Suite wird durch Benutzeraktion gewählt. 5. Die Suite wird gelöscht.
Alternative Ablaufschritte	4.1 Die Aktion wird verworfen.

Tabelle 8.6: Anwendungsfall: Suite löschen

Name	**Ergebnis löschen**
Beschreibung	Das Löschen eines Ergebnisses über die Anwendung
Beteiligte Akteure	Administrator
Verwendete Anwendungsfälle	Test aufrufen
Ergebnis	Ein Ergebnis wurde gelöscht.
Vorbedingungen	Benutzerkonto für Authentifizierung am System und ein bereits ausgeführter Test
Standardablauf	1. Der Benutzer authentifiziert sich am System. 2. Ein Ergebnis wird anhand seiner ID aufgerufen. 3. Eine Sicherheitsabfrage für das Löschen des Ergebnisses wird angezeigt. 4. Die Löschung des Ergebnisses wird durch Benutzeraktion gewählt. 5. Das Ergebnis wird gelöscht.
Alternative Ablaufschritte	4.1 Die Aktion wird verworfen.

Tabelle 8.7: Anwendungsfall: Ergebnis löschen

Abbildungsverzeichnis

Tabellenverzeichnis

Listings

Abkürzungsverzeichnis

AJAX Asynchronous JavaScript and XML

API Application Programming Interface

BSON Binary JavaScript Object Notation

CPU Central Processing Unit

CRUD Create Read Update Delete

CSS Cascading Style Sheets

DB Datenbank

Doctype Document Type Declaration

DOM Document Object Model

ECMA European Computer Manufacturers Association

GUI Graphical User Interface

HAML Hypertext Abstraction Markup Language

HTML Hypertext Markup Language

HTTP Hypertext Transfer Protocol

IP Internet Protocol

JS JavaScript

JSConfEU JavaScript Conference Europe

JSON JavaScript Object Notation

SVG Scalable Vector Graphics

LoD Law of Demeter

MIT Massachusetts Institute of Technology

MVC Model View Controller

NOSQL Not only SQL

NPAPI Netscape Plug-In Application Programming Interface

NPM Node Package Manager

ODM Object-Document Modeler

OOP Objektorientierte Programmierung

RAM Random Access Memory

REST Representational State Transfer

RFC Request for Comments

SQL Structured Query Language

SSP Single Site Page

UML Unified Modeling Language

UUID Universally Unique Identifier

Glossar

Application Programming Interface Das API ist eine Schnittstelle, die von Programmen angeboten wird. Über diese Schnittstelle können andere Programme interagieren. 15, 73

Asynchronous JavaScript and XML AJAX bietet die Möglichkeit einer bereits gerenderten Webseite Daten nachzuladen. Hierbei tritt keine Blockierung der Webseite auf. 73, 77

Binary JavaScript Object Notation Die Notation nach den Richtlinien von JavaScript Objekten ist ein Austauschformat. Die Daten werden, anders als bei JSON, binär codiert. 43, 73

Canvas Zu deutsch: Leinwand. Hierbei handelt es sich um ein HTML-Tag, das für die Visualisierung bzw. das Zeichnen konzipiert wurde. Angesteuert wird das Element über JavaScript. 29

Cascading Style Sheets Sogenannte CSS-Datein oder auch Attribute sind Formatierungsanweisungen für HTML-Dokumente. Durch CSS wird das Aussehen einer Seite bestimmt. 26, 73

Central Processing Unit Würde man metaphorisch von dem Herz eines Computers sprechen, so wäre dies die CPU. Sie ist der eigentliche Rechner im Rechner. 9, 73

Create Read Update Delete Diese Operationen sind die Basis aller Datenmanipulationen und werden in der Softwareentwicklung als CRUD abgekürzt. 32, 73

Document Object Model Das DOM ist ein API, das ein HTML Dokument repräsentiert und zusätzlich die Möglichkeit bietet dieses zu manipulieren (Flanagan, 2011, S.361). Über diese Schnittstelle werden sämtliche Aktionen mit dem statischen Inhalt einer Seite durchgeführt. 2, 73

Document Type Declaration Die Deklaration bestimmt die Version einer Auszeichnungssprache mit Hinweis auf die Definition. 15, 73

Graphical User Interface Zu deutsch: Benutzeroberfläche. Das GUI ist der Teil eines Programms, an dem ein Benutzer Eingaben tätigt und Ausgaben präsentiert bekommt. 26, 73

Hypertext Abstraction Markup Language Eine Abstraktionsschicht auf HTML. Es wird das Schreiben von HTML erleichtert aufgrund einer einfacheren Syntax. 30, 73

Hypertext Markup Language Die Hypertext Markup Language ist eine Auszeichnungssprache. Sie dient meist der formatierten Darstellung von Inhalten des Internets. 15, 73

Hypertext Transfer Protocol Das Protokoll ist der zentrale Bestandteil des Datenaustauschs von Webseiten. Hierfür kommuniziert ein Browser mit einem Server über das Protokoll. 10, 73

Internet Protocol Das Internet Protokoll ist die Grundlage des Internets. Es dient der Adressierung von Rechnern in einem Netzwerk. 30, 73

JavaScript Ist eine Skriptsprache, die ursprünglich für Dynamisierung und Manipulation von HTML-Dokumenten entwickelt wurde. 27, 73

JavaScript Conference Europe Eine jährlich stattfindende Konferenz, die für JavaScript-Entwickler in Europa ins Leben gerufen wurde. 8, 73

JavaScript Object Notation Die Notation nach den Richtlinien von JavaScript Objekten ist ein Austauschformat. Der Vorteil liegt in seinem geringen Overhead z.B. gegenüber XML. 43, 73

Law of Demeter Allgemein betrachtet ist das LoD-Muster eine Form der losen Kopplung. Eine prägnante Formulierung seines Ziels besagt: „Sprich nur zu Freunden, nicht zu Fremden“ (Martin, 2009, S.134). 28, 73

Massachusetts Institute of Technology Ist eine nordamerikanische Universität und Hochschule für Technik in Cambridge. Die Stadt liegt im Bundesstaat Massachusetts. 8, 74

Model View Controller Die Untergliederung in verschiedene Bereiche einer Software ist der Kern dieses Musters. Hierbei ist das Model der Teil, der die reinen Daten hält. Der View ist ein datenleeres Template mit Platzhaltern. Diese werden beim Rendern mit

Daten ersetzt. Der Controller fungiert als Schnittstelle zwischen dem Model und dem View und beinhaltet die Anwendungslogik. 28, 74

Netscape Plug-In Application Programming Interface Eine plattformübergreifende Plug-In Architektur. 61, 74

Node Package Manager Ein Verwaltungswerkzeug für Bibliotheken. Über NPM kann bezogen (heruntergeladen) und publiziert (hochgeladen) werden. Ein ähnliches Werkzeug ist der populäre Paket-Manager „Aptitude“ auf Linux-Systemen. 8, 74

Not only SQL NoSQL Datenbanken unterscheiden sich meist zu Structured Query Language (SQL)-Datenbanken durch den Verzicht auf Konsistenz gegenüber der Geschwindigkeit oder Verfügbarkeit. 43, 74

Object-Document Modeler Für die Übersetzung von Objekten in der Programmierumgebung in eine NOSQL Datenbank sind ODMs Übersetzungswerkzeuge. 32, 74

Objektorientierte Programmierung Dieser Art der Programmierung liegen unter anderem folgende Konzepte zugrunde: Objekte, Methoden und Eigenschaften - Klassen - Kapselung - Aggregation - Wiederverwendung/Vererbung - Polymorphismus (Stefanov und Sharma, 2013). 5, 74

Plug-In Ein Plug-In ist ein Softwarebaustein, der die Funktionalität eines bestehenden Systems erweitert. 30

Random Access Memory Im Deutschen auch als Arbeitsspeicher bezeichnet. Dieser dient der flüchtigen Speicherung von Daten im laufenden Betrieb eines Systems. 9, 74

Representational State Transfer Hierbei handelt es sich um einen Architekturstil, der für Webservices eingesetzt wird. 34, 74

Request for Comments Bei RFC-Dokumenten handelt es sich um Spezifikationen von Technologie Standards im Internet. 74, 77

Scalable Vector Graphics Skalierbare Grafiken, die auf Vektoren basieren. Die Visualisierung erfolgt auf zweidimensionaler Basis. 29, 73

Single Site Page Der Begiff wurde durch die Asynchronous JavaScript and XML (AJAX)-Technologie geprägt. Er steht für ein teilweises Nachladen einer Webseite. 26, 74

Structured Query Language Eine Abfragesprache für relationale Datenbanken. 74, 77

Unified Modeling Language „Die Sprache dient der Visualisierung, Spezifizierung, Konstruktion und Dokumentation der Artefakte eines softwareintensiven Systems“ (Booch u. a., 2006, S.37). 74

Universally Unique Identifier Der Aufbau richtet sich nach dem Request for Comments (RFC) 4122. Dieser beschreibt die Struktur und den Aufbau eines Identifikators. 39, 74

Literaturverzeichnis

[A.A.Puntambekar 2010] A.A.Puntambekar, A.A.Puntambekar: *Software Engineering And Quality Assurance*. Technical Publications, 2010. – ISBN 978-8-184-31779-4

[Alcorn u. a. 2014] Alcorn, Wade ; Frichot, Christian ; Orru, Michele: *The Browser Hacker's Handbook*. 1. Aufl. New York : John Wiley and Sons, 2014. – ISBN 978-1-118-91435-9

[Booch u. a. 2006] Booch, Grady ; Rumbaugh, James ; Jacobson, Ivar: *Das UML-Benutzerhandbuch - aktuell zur Version 2.0*. Erste Auflage. München : Pearson Deutschland GmbH, 2006. – ISBN 978-3-827-32295-1

[Chodorow 2013] Chodorow, Kristina: *MongoDB: The Definitive Guide*. Erste Auflage. 1005 Gravenstein Highway North, Sebastopol, CA 95472 : O'Reilly Media, Inc., 2013. – ISBN 978-1-449-34482-5

[Crockford 2008] Crockford, Douglas: *JavaScript: The Good Parts - The Good Parts*. Erste Auflage. 1005 Gravenstein Highway North, Sebastopol, CA 95472 : O'Reilly Media, Inc., 2008. – ISBN 978-0-596-55487-3

[ECMA 1997] ECMA: *ECMAScript: A general purpose,cross-platform programminglanguage*. Juni 1997. – URL http://www.ecma-international.org/publications/files/ECMA-ST-ARCH/ECMA-262,%201st%20edition,%20June%201997.pdf

[Elliott 2014] Elliott, Eric: *Programming JavaScript Applications - Robust Web Architecture with Node, HTML5, and Modern JS Libraries*. Erste Auflage. 1005 Gravenstein Highway North, Sebastopol, CA 95472 : O'Reilly Media, Inc., 2014. – ISBN 978-1-491-95027-2

[Firtman 2013] Firtman, Maximiliano: *Programming the Mobile Web*. Zweite Auflage. 1005 Gravenstein Highway North, Sebastopol, CA 95472 : O'Reilly Media, Inc., 2013. – ISBN 978-1-449-33497-0

[Flanagan 2011] Flanagan, David: *JavaScript: The Definitive Guide*. Sechste Auflage. 1005 Gravenstein Highway North, Sebastopol, CA 95472 : O'Reilly Media, Inc., 2011

[Google 2014] Google: *AngularJS - Superheroic JavaScript MVW Framework*. November 2014. – URL https://angularjs.org/. – Zugriffsdatum: 22. November 2014

[Green und Seshadri 2013] Green, Brad ; Seshadri, Shyam: *AngularJS*. Erste Auflage. 1005 Gravenstein Highway North, Sebastopol, CA 95472 : O'Reilly Media, Inc., 2013. – ISBN 978-1-449-35588-3

[Holmes 2013] Holmes, Simon: *Mongoose for Application Development*. Erste Auflage. Birmingham : Packt Publishing Ltd, 2013. – ISBN 978-1-782-16820-1

[Hughes-Croucher und Wilson 2012] Hughes-Croucher, Tom ; Wilson, Mike: *Node: Up and Running*. Erste Auflage. 1005 Gravenstein Highway North, Sebastopol, CA 95472 : O'Reilly Media, Inc., 2012

[Ihrig 2013] Ihrig, Colin J.: *Pro Node.js for Developers*. Erste Auflage. New York : Apress, 2013. – ISBN 143-0-258-616

[Kopec 2014] Kopec, David: *Dart for Absolute Beginners*. Erste Auflage. New York : Apress, 2014. – ISBN 978-1-430-26482-8

[Kuan-Ching 2009] Kuan-Ching, Li: *Handbook of Research on Scalable Computing Technologies* -. IGI Global, 2009. – ISBN 978-1-605-66662-4

[Lang 2014] Lang, Sean: *Web Development with Jade*. Erste Auflage. Birmingham : Packt Publishing Ltd, 2014. – ISBN 978-1-783-28636-2

[Martin 2009] Martin, Robert C.: *Clean Code - Refactoring, Patterns, Testen und Techniken für sauberen Code*. 2009

[Node.js 2014] Node.js: *node.js*. November 2014. – URL http://nodejs.org. – Zugriffsdatum: 10. November 2014

[npm 2014] npm: *npm*. November 2014. – URL https://www.npmjs.org/. – Zugriffsdatum: 15. November 2014

[Odell 2014] Odell, Den: *Pro JavaScript Development - Coding, Capabilities, and Tooling*. Erste Auflage. New York : Apress, 2014. – ISBN 978-1-430-26269-5

[Otto und Thornton 2014] Otto, Mark ; Thornton, Jacob: *Bootstrap*. November 2014. – URL http://getbootstrap.com/. – Zugriffsdatum: 25. November 2014

[Panda 2014] Panda, Sandeep: *Angularjs - Novice to Ninja*. Erste Auflage. 48 Cambridge Street Collingwood VIC Australia 3066 : SitePoint Pty, Limited, 2014. – ISBN 978-0-992-27945-5

[Panzarino 2014] Panzarino, Onofrio: *Learning Cypher*. Erste Auflage. Birmingham : Packt Publishing Ltd, 2014. – ISBN 978-1-783-28776-5

[Rai 2013] Rai, Rohit: *Socket. IO Real-Time Web Application Development*. Erste Auflage. Birmingham : Packt Publishing Ltd, 2013. – ISBN 978-1-782-16079-3

[Rauschmayer 2014] Rauschmayer, Axel: *Speaking JavaScript*. 1005 Gravenstein Highway North, Sebastopol, CA 95472 : O'Reilly Media, Inc., 2014. – ISBN 978-1-449-36501-1

[Resig 2014] Resig, John: *John Resig - Versions of JavaScript*. Oktober 2014. – URL http://ejohn.org/blog/versions-of-javascript/. – Zugriffsdatum: 22. Oktober 2014

[Scott 2009] Scott, Michael L.: *Programming Language Pragmatics*. 3. Aufl. San Francisco, Calif : Morgan Kaufmann, 2009. – ISBN 978-0-080-92299-7

[Spurlock 2013] Spurlock, Jake: *Bootstrap - The world's most popular mobile-first and responsive front-end framework*. Erste Auflage. 1005 Gravenstein Highway North, Sebastopol, CA 95472 : O'Reilly Media, Inc., 2013. – ISBN 978-1-449-34391-0

[Stefanov und Sharma 2013] Stefanov, Stoyan ; Sharma, Kumar C.: *Object-Oriented JavaScript*. Zweite Auflage. Birmingham : Packt Publishing Ltd, 2013. – ISBN 978-1-849-69313-4

[Taligarsiel 2015] Taligarsiel: *How browsers work*. Februar 2015. – URL http://taligarsiel.com/Projects/howbrowserswork1.htm. – Zugriffsdatum: 17. Februar 2015

[Teixeira 2014] Teixeira, Pedro: *Professional Node.js*. Erste Auflage. 10475 Crosspoint Boulevard Indianapolis, IN 46256 : John Wiley and Sons, Inc., 2014

[W3C 2014] W3C: *HTML 4.0 Specification*. Dezember 2014. – URL http://www.w3.org/TR/1998/REC-html40-19980424/. – Zugriffsdatum: 15. Dezember 2014

[Wilson 2013] Wilson, Jim R.: *Node.js the Right Way - Practical, Server-Side JavaScript That Scales*. Erste Auflage. Dallas, Texas : Pragmatic Programmers, LLC, 2013. – ISBN 978-1-937-78573-4

[Yaapa 2013] Yaapa, Hage: *Express Web Application Development.* Erste Auflage. Birmingham : Packt Publishing Ltd, 2013. – ISBN 978-1-849-69655-5

[Zakas 2009] Zakas, Nicholas C.: *Professional Javascript For Web Developers.* Zweite Auflage. New Delhi : Wiley India Pvt. Limited, 2009. – ISBN 978-8-126-51970-5

Hiermit versichere ich, dass ich die vorliegende Arbeit ohne fremde Hilfe selbständig verfasst und nur die angegebenen Hilfsmittel benutzt habe.

Hamburg, 27. März 2015 Micha Severin