Hochschule Offenburg
University of Applied Sciences

Generisches Mehrbenutzer-Framework auf Basis von Node.js und Webtechnologien am Beispiel eines Stundenplan-Gestalters

Bachelor Thesis an der Hochschule Offenburg

Fakultät Medien und Informationswesen

Studiengang Medien und Informationswesen

vorgelegt von

Daniel Betz

Sommersemester 2014

Betreuer:

Prof. Dr. Tom Rüdebusch, Hochschule Offenburg

M.Sc. Dipl.-Inform. (FH) Dennis Schober-Wenger, SinusQuadrat GmbH

Medien und Informationswesen

SinusQuadrat
Das machen wir gerne.

Erklärung

Hiermit erkläre ich, Daniel Betz, dass ich die vorliegende Bachelorarbeit bzw. Masterarbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

_____ _____

(Ort, Datum) (Unterschrift)

Inhaltsverzeichnis

Abbildungsverzeichnis

Abkürzungsverzeichnis

API	Application Programming Interface
BSON	Binary JSON
CAP	Consistency, Availability, Partition tolerance
CRUD	Create, Read, Update, Delete
CSCW	Computer Supported Cooperative Work
CSS	Cascading Style Sheets
DOM	Document Object Model
FTP	File Transfer Protocol
GUI	Graphical User Interface
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
IE	Internet Explorer
IETF	Internet Engineering Task Force
JS	JavaScript
JSON	JavaScript Object Notation
LAN	Local Area Network
MS	Microsoft
NoSQL	Not only SQL
PC	Personal Computer
PHP	PHP Hypertext Preprocessor
RDBMS	Relational Database Management System
REST	Representational State Transfer
RFB	Remote Framebuffer Protocol
SSE	Server-Sent Events
VNC	Virtual Network Computing
W3C	World Wide Web Consortium
WHATWG	Web Hypertext Application Technology Working Group

1 Einleitung

1.1 Aufgabenbeschreibung

Web-Applikationen sind ein immer beliebter werdendes Mittel, um dem Menschen Aufgaben abzunehmen und Lösungsprozesse zu vereinfachen. Wo früher noch das Erlernen nativer Systemsprachen und das Durchführen komplexer lokaler Installationen nötig war, um Programme zu schreiben und anzuwenden, so lässt sich das heute mit weltweiten Standards plattformübergreifend mit wenigen, einheitlichen Sprachen im Browser erledigen. Browser liefern sich einen Wettkampf um die Implementierung von Neuerungen und der Anwender profitiert nicht nur von einer Plattformunabhängigkeit, sondern sogar weitestgehend auch von einer Browserunabhängigkeit. In einer Zeit, in der Endgeräte zur Betrachtung von Webseiten immer mobiler, performanter und vielfältiger werden und sich die Netzbetreiber Internet „everywhere and everytime“ zum Ziel machen, finden weborientierte Applikationen mehr Anwendung denn je.

Die Gründe, beispielsweise einen Taschenrechner für Windows, Mac OS X, Linux, iOS und Android in der jeweils nativen Programmiersprache redundant zu programmieren, während alle Systeme auf eine responsive Seite zugreifen könnten, auf der ein einzelner JavaScript-Taschenrechner liegt, werden verschwindend gering. Durch den Internet-Netzausbau, immer großzügigere Internet-Datentarife und das Bestreben, Web-Applikationen auch offline nutzen zu können, wird die Online-Abhängigkeit relativiert. Durch das immer tiefere Einbinden des Browsers in das Betriebssystem werden Performance-Schwächen reduziert.

Die Idee von Anwendungen, die im Browser laufen, ist nicht neu. Da gibt es zum einen die serverseitigen Applikationen. Sie präsentieren ihre Oberfläche dem Client über dynamische Webseiten. Eine User-Interaktion wird an den Server weitergeleitet, dieser verarbeitet sie und sendet das Ergebnis zurück an den Client.

Zum anderen werden Web-Anwendungen clientseitig ausgeführt. Die Programmlogik wird hier beim Seitenaufruf lokal heruntergeladen. Mittels JavaScript und CSS lassen sich komplexe User-Interfaces erstellen und Algorithmen implementieren.
Auch üblich war bis vor ein paar Jahren der Gebrauch von Adobe Flash, welches sehr komfortable Möglichkeiten bietet, mit dem Anwender multimedial zu kommunizieren und komplexe Datenstrukturen zu verarbeiten. Es benötigt allerdings ein proprietäres Plug-In, das teilweise Sicherheits- und Performanceschwächen aufwies. Recht früh wurde es von Apple und bald auch von Google auf deren mobilen Endgeräten verbannt. Dies führte zu einem langsamen, aber stetigen Rückgang von Adobe Flash-Websites.

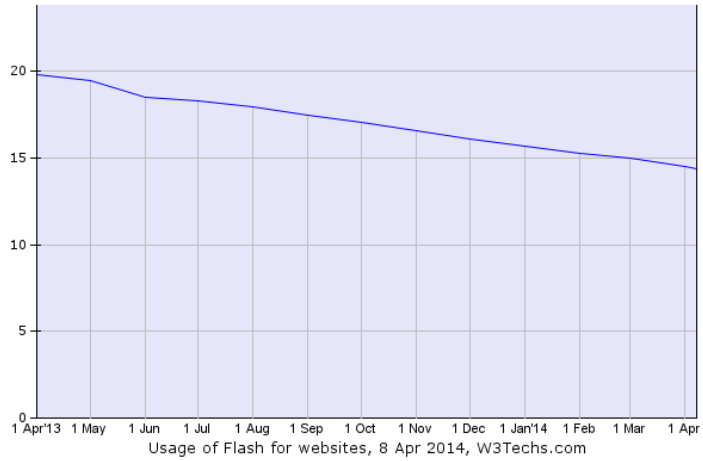

Abbildung 1: Prozentualer Anteil an Flash-Websites im Internet [1]

Dafür musste Ersatz her. Multimedialität ist keine Stärke des aktuellen HTML-Standards. Browserhersteller und Entwickler beschäftigen sich seit vielen Jahren mit einem Nachfolger für HTML 4.01, welches seit 1999 nicht mehr aktualisiert wurde. Mitte 2004 veröffentlichte die WHATWG einen ersten Vorschlag für HTML5 und erweiterte die bis dahin reine Auszeichnungssprache um multimediale Features, wie native Unterstützung von Audio und Video mit teils quelloffenen Codecs, sowie dem „Canvas“, einer Zeichenfläche, die direkt vom Browser gerendert wird. Damit wurde die Lücke geschlossen, die Adobe Flash bis dahin gefüllt hatte, und Applikationen ließen sich auf eine neue Art entwickeln.

Seit 2004 implementieren alle gängigen Browserhersteller die Technologien von HTML5. Auch im mobilen Bereich, allen voran Android und iOS mit zusammen knapp 95% Marktanteil, findet HTML5 eine sehr breite Unterstützung. [2]

An diesem Punkt setzt die Thesis an. Ziel dieser Abschlussarbeit ist ein Framework, das Entwickler benutzen können, um kollaborative Echtzeit-Applikationen für den Browser zu programmieren.

Das User Interface wird durch einfache Definitionen von GUI-Komponenten realisiert und vollständig in einem HTML5-Canvas dargestellt. Interaktion durch Anwender besteht hauptsächlich darin, einer freien Gestaltungsfläche Hintergründe, grafische Elemente und Textfelder hinzuzufügen und diese entsprechend zu positionieren und zu transformieren.
Das Framework soll generisch sein, d.h. es ist dem Entwickler überlassen, welche Problematik oder Aufgabe die Applikation vereinfachen soll; das Framework benötigt die Daten dazu in einem bestimmten Format und definierten Regeln folgend und erstellt daraus die Anwendung.

Im Rahmen dieser Arbeit wird das Framework für die Erstellung eines Stundenplan-Gestalters genutzt. In diesem Gestalter, gezielt für Schüler entwickelt, lassen sich Fächer eintragen. Kombiniert wird dies mit zahlreichen optischen Gestaltungsmöglichkeiten.
Im Sinne der Mehrbenutzerfähigkeit können mehrere Anwender einen Stundenplan gleichzeitig bearbeiten. Alle gestalteten Stundenpläne werden in einer Datenbank gespeichert und lassen sich jederzeit wieder weiterbearbeiten.

1.2 Firmenumfeld

Die SinusQuadrat GmbH ist eine in Offenburg ansässige Webagentur. Sie wurde 2009 gegründet. Ihre thematische und geografische Nähe zur Medienfakultät der Hochschule Offenburg bietet für Absolventen ein optimales Umfeld zur Bearbeitung ihrer Abschlussarbeit. Einer der Inhaber und einige der Mitarbeiter sind Abgänger der Offenburger Hochschule und stehen zusammen mit einem freundlichen und kompetenten Team an Designern und Entwicklern jederzeit mit Rat und Tat zur Seite.

Mit zehn bis zwölf Mitarbeitern zählt SinusQuadrat zu den kleineren bis mittelgroßen Webagenturen. Im Portfolio sind neben Websites auch komplexere Web-Applikationen, Onlineshops, Online-Marketing, Suchmaschinenoptimierung, Apps für mobile und stationäre Endgeräte, Newsletter und Design.

Der Slogan „Das machen wir gerne!“ [3] steht für die Philosophie des Unternehmens, mit Freude und Überzeugung Lösungen für unterschiedlichste Problemstellungen zu finden. Auch der Einsatz und die Erprobung neuer Technologien sind dabei wichtige Themen.

Ein früheres Projekt der SinusQuadrat GmbH war die Programmierung eines Stundenplan-Gestalters für den Kunden „Mildenberger Verlag GmbH“. Dieser bietet Lehrinhalte an und wollte diesen Service für Schüler anbieten. Realisiert wurde das Projekt seitens SinusQuadrat mittels Flash. [4] Da dieser Gestalter ein sehr verständliches und auch beliebtes Produkt ist, bietet er sich als Vorlage für eine Beispielapplikation an.

Freundlicherweise stellt der Mildenberger Verlag für diese Thesis sämtliche Ressourcen seines Flash-Stundenplan-Gestalters zur Verfügung, so dass sich das GUI der Beispielapplikation recht detailliert auf die bereits gestalteten Elemente des aktuellen Flash-Produkts stützen kann.

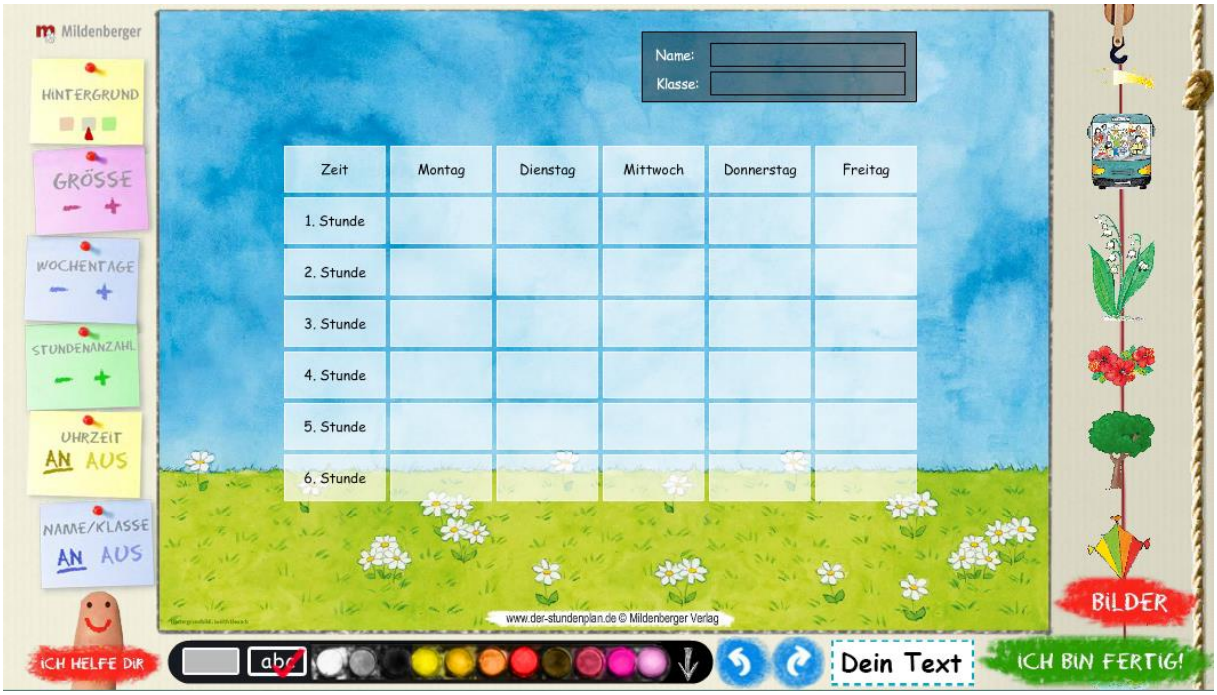

Abbildung 2: Stundenplan-Gestalter in Flash [4]

2 Theoretische Grundlagen und Stand der Technik

In diesem Kapitel wird die Theorie behandelt, die dem fertigen System zu Grunde liegt. Es geht darum, essentielle Strukturen von generischen Frameworks und kooperativen Web-Anwendungen zu erläutern und diese anhand von Beispielen aus der Realität aufzuzeigen.

2.1 Generische Frameworks

Bruce Powel Douglass, ein Amerikaner, der seit langem für IBM arbeitet, beschreibt den Charakter eines Frameworks 1999 wie folgt:

„Much of the infrastructure among applications is the same – the means for starting and ending tasks, task and resource scheduling, methods for implementing and executing state machines, and ways of implementing associations are not very application-specific. This is even more true among applications drawn from the same domain. Usually, anywhere from 60 percent to 90 percent of an application is common "housekeeping" code that can be reused if properly structured. The "traditional" way of building systems involves reconstructing the same implementation strategies over and over.
Another approach is to capture the common facilities required by all the applications of a domain in such a way that they can be reused as is, specialized as appropriate, or easily replaced if necessary. These facilities are generally a set of cooperating patterns that make up the backbone of the application. This infrastructure is called a framework.” [5]

Kategorisiert nach technischen Einsatzbereichen stellen Frameworks also eine Infrastruktur dar, in der die immer wiederkehrenden Anforderungen an ein Softwaresystem implementiert werden. Im Bereich des Internet sind das zum Beispiel Datenbankanbindung oder Lokalisierung. Der Aufbau wird möglichst modular gehalten, um Wiederverwendung oder Spezialisierung möglichst einfach zu machen.

„Generisch“ kommt aus dem Lateinischen und wird von „generalis“ abgeleitet, was mit „allgemein“ oder „die ganze Gattung betreffend“ übersetzt wird. Zusammengesetzt geht es also um eine Programmstruktur, die nicht nur auf einen Anwendungsfall zugeschnitten ist, sondern auf verschiedene Aufgabenstellungen anwendbar ist.

Ralph E. Johnson und Brian Foote formulierten ihre Definition in einem Artikel des "Journal of Object-Oriented Programming“ von 1988 wie folgt: „A framework is a semi-complete application. A framework provides a reusable, common structure to share among applications. Developers incorporate the framework into their own application and extend it to meet their specific needs.“ [6]

Noch kompakter hat es Gregory Rogers 1997 mit folgendem Satz ausgedrückt: “A framework is a partially completed application, pieces of which are customized by the user to complete the application.” [7]

Diese Definitionen sind bereits mehrere Jahrzehnte alt, dennoch treffen sie auch auf moderne Frameworks zu.

2.1.1 Vor- und Nachteile eines Frameworks

Frameworks bieten diverse Vorteile [5], die hier noch einmal im Einzelnen herausgestellt werden:

Senkung der Komplexität

Übliche Prozesse der Anwendungsentwicklung werden ausgelagert abgehandelt. So lassen sich Anwendungsstruktur und -verhalten leichter nachvollziehen.

Erledigung üblicher Aufgaben

Vom Framework bereitgestellte „Service-Klassen“ erledigen Aufgaben, die in praktisch jeder Applikation implementiert werden müssen. So kann der Programmierer sich auf spezifischere Aspekte konzentrieren.

Bereitstellung einer Architektur

Frameworks stellen eine Applikations-Architektur zur Strukturierung bereit, wie etwa das weit verbreitete „Model View Controller“-Modell.

Ersparnis von Zeit und Ressourcen

Die Tatsache, dass bereits fertige, allgemein gebräuchliche Applikations-Funktionalitäten implementiert sind, verkürzt die Zeitspanne und damit den Ressourcenbedarf bis zur Marktreife einer Applikation.

Spezialisierung in Zielrichtung

Frameworks lassen sich teil-spezialisieren, etwa für bestimmte Anwendungsszenarien oder Themengebiete, und bieten damit ähnlichen Projekten einen verstärkten Vorteil der bisher genannten Punkte.

Community

Populäre Frameworks werden durch ihre Community ständig verbessert und umfangreich getestet. Dies ist dem Sicherheits- und dem Supportaspekt dienlich und wird dadurch begünstigt, dass viele Frameworks „open source“, also Quellcode-offen angeboten werden.

Allerdings sind Frameworks auch nicht immer das Mittel der Wahl, um neue Applikationen zu entwerfen. So muss auf folgende Punkte geachtet werden:

Lerneffekt

Bei Verwendung eines Frameworks lernt man nicht den besseren Umgang mit einer Programmiersprache oder einem Endsystem, sondern die Interaktion mit dem Framework selbst.

Funktionelle Einschränkung

Legt man ein Framework als Grundgerüst seiner Applikation fest, so lebt man mit dessen Infrastruktur. Dies impliziert dessen Architektur und Funktionalität. Ist das Ziel der eigenen Anwendung weit entfernt von der Art des verwendeten Frameworks, so muss dieses nicht unbedingt von Vorteil sein oder kann gar nachteilig sein. Bevor man Funktionalitäten komplett umschreibt, um die eigenen Vorstellungen umzusetzen, ohne dabei die Schnittstelle zum Rest des Frameworks aufzugeben, sollte man sich überlegen, ob nicht ein direkter Weg

zum Ziel unter Verzicht auf das Framework weniger Aufwand an Arbeitsschritten und Zeit erfordern würde.

Funktionelle Überlastung

Existieren auf der einen Seite Einschränkungen durch die gegebene Funktionalität eines Frameworks, so existiert auf der anderen Seite ein Überangebot an ungenutzten Funktionen. Diese kann man entweder im Code entfernen, was umfangreiche Kompetenz auf der Codeebene erfordert, oder man nutzt das Framework komplett und vergrößert damit die physikalische Größe der Applikation.

2.1.2 Anforderungen und Abgrenzungskriterien

Anforderungen an ein Framework kann man in drei Kategorien unterteilen: Funktionale Anforderungen, nicht-funktionale Anforderungen und Einschränkungen beziehungsweise Auflagen.

Abzugrenzen ist ein Framework von einer Bibliothek. Während Bibliotheken vom Entwickler aufgerufen werden, um darin implementierte Funktionen zu verwenden, so rufen Frameworks den Code des Entwicklers auf, um ihn in ihren Algorithmen zu verwenden.

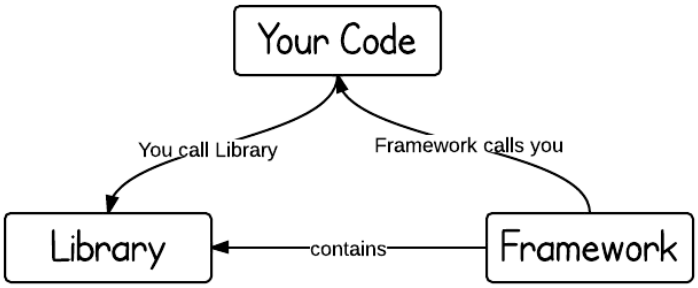

Abbildung 3: Framework vs. Library [8]

Funktionale Anforderungen an ein Framework

Diese Kategorie bestimmt schlicht, welche Funktionalität das Framework erfüllen soll. Das Autoren-Team Robertson & Robertson beschreibt diese Kategorie mit dem Satz „Functional requirements are things the product must **do**.“. [9] Funktionale Anforderungen werden in der Konzeptionsphase entworfen, in der Implementierungsphase entwickelt und können in der Testphase eindeutig validiert werden. [10]

Nicht-funktionale Anforderungen an ein Framework

Den funktionalen Anforderungen stehen die nicht funktionalen Anforderungen gegenüber. Sie betreffen die „Quality of Service“, oder auch „Dienstgüte“ des Frameworks.
Robertsons definieren diese Anforderungen mit folgendem Satz: „Nonfunctional requirements are qualities the product must **have**.“. [9]
Nicht-funktionale Anforderungen können beispielsweise Verlässlichkeit, Flexibilität, Wartbarkeit, Bedienbarkeit, das „Look and Feel“, Sicherheit oder Performance sein. Sie werden oft subjektiv und formlos bewertet, was die Validierung am Ende erschwert. Nicht-funktionale Anforderungen können meist nicht als Programmpunkt in der Konzeptions- oder Implementierungsphase abgehakt werden – vielmehr begleiten sie das Projekt durchgehend oder werden in der Endphase nachimplementiert. [10]

Auflagen/Einschränkungen eines Frameworks

Die dritte Kategorie sind die „Constraints“. Nach Robertson: „Constraints are global issues that shape the requirements.“. [9]

In diesem Sinne sind Auflagen etwa zeitlicher Natur, Stichwort: Deadline, oder bestimmen einen Wunsch, der dann in den funktionalen Anforderungen konkret umgesetzt wird. Beispiele für diesen Fall wären etwa der Wunsch, das Produkt müsse auch ohne Maus bedienbar sein oder es müsse auf jedem Betriebssystem laufen.

2.1.3 Konzepte

Man kann Frameworks zwischen White-Box- und Black-Box-Frameworks differenzieren.

Bei der Black-Box-/White-Box-Kategorisierung geht es darum, wieviel der Anwender eines Frameworks mit dessen Code zu tun bekommt beziehungsweise wieviel des Innenlebens er zu sehen bekommt.

White-Box Frameworks

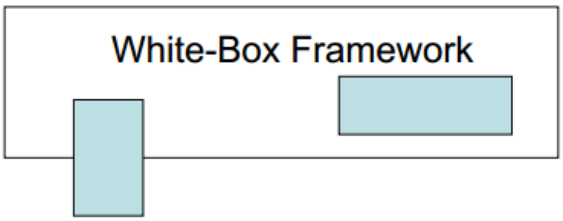

Abbildung 4: White-Box Framework [11]

Im obigen Bild sieht man das Konzept einer White-Box. Man kann und muss in den Code schauen und bildet in der Regel Subklassen, die von den Framework-Klassen erben, um das Framework für seine spezifischen Bedürfnisse zu verwenden. Je nach Anwendungsfall ändert man Code **im** Framework und modifiziert die Schnittstellen nach außen.

Der Lern- und Programmieraufwand ist im Vergleich zu Black-Box Frameworks generell höher. Weiterhin hat der Anwender für die Wartung seiner Subklassen zu sorgen und muss darauf achten, dass er diese sauber implementiert.

Der Ansatz der Vererbung stellt einige Aspekte heraus:

- Das Zusammenspiel zwischen Sub- und Superklasse muss verstanden werden
- Man hat Zugriff sowohl auf public- als auch auf protected-Teile der Superklasse
- Man hat Zugriff auf die Methoden der Superklasse und kann diese je nach speziellem Bedarf überschreiben

Allgemein gilt der Satz: „Programmieren statt Konfigurieren“ [12] oder „Override to customize“ [13]

Black-Box Frameworks

Bei einem Black-Box Konzept geht es darum, dem Anwender möglichst wenige Berührungspunkte mit dem Inneren des Frameworks zu liefern. Generell gilt „Wiederverwendung durch Komposition“ [12] oder „Configure to customize“ [13].

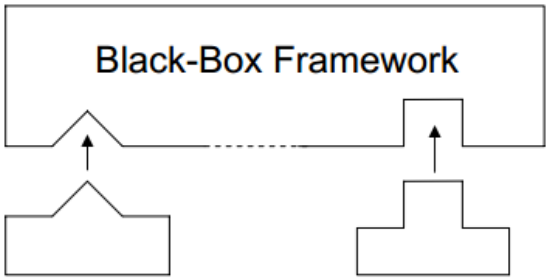

Abbildung 5: Black-Box Framework [11]

Dem Konzept der Vererbung steht hier das Konzept der Komposition entgegen. Es werden keine Subklassen erstellt, stattdessen kreiert man Objekte, deren genaue, durch die Klasse vorgegebenen Methoden und Attribute man nicht kennt. Stattdessen wird zum Beispiel durch eine Dokumentation erläutert, wie das Interface funktioniert und welche Möglichkeiten man damit hat.

Wartung und Updates obliegen dem Entwickler des Frameworks, der Anwender hat keine Möglichkeit, dem Framework durch Einfügen unsauberen Codes oder ähnlichem die Performance zu verschlechtern oder bewährte Strukturen zu verändern, was an anderer Stelle nachteilige Folgen haben könnte.

2.1.4 Praxisbeispiele

Inzwischen gibt es für alle denkbaren Betriebssysteme und Programmiersprachen Frameworks. Um den thematischen Bezug zu dieser Thesis zu wahren, sollen hier nur Web-Frameworks betrachtet werden.

Das Portal „HotFrameworks“ [14] vergleicht die Beliebtheit von Web-Frameworks anhand ihrer GitHub- und Stack Overflow -„Scores“. Die GitHub-Punktzahl berechnet sich aus den Sternen, die zugeneigte Entwickler dem Framework auf dem Git-Portal [15] vergeben haben. Die Stack Overflow-Punktzahl generiert sich an der Anzahl der Fragen, die zu dem Framework auf stackoverflow.com [16] gestellt wurden. Erachtet man diese Methode der Eruierung als repräsentativ, so zeigen sich als beliebteste Frameworks „Ruby on Rails“, „AngularJS“ und „Django“.

In der Rubrik der JavaScript-Frameworks sind laut HotFrameworks.com AngularJS [17], Express [18] und Ember.js [19] am beliebtesten.

Alle drei Frameworks dienen grundsätzlich zur Web-Applikations-Entwicklung.
AngularJS und Ember.js werden clientseitig im HTML-Code eingebunden und erledigen über referenzierte JavaScript-Dateien die Modellierung, die Präsentation und die Steuerung der Applikation.
Express hingegen wird serverseitig eingesetzt und arbeitet auf Basis von Node.js. Mit ihm lassen sich neben üblichen Model, View und Control-Funktionalitäten ganze Webseiten-Infrastrukturen aufbauen und steuern. Es stellt unter anderem mehrere View-Engines zur Auswahl und bietet komplexe Routing-Optionen.

2.2 Kollaborative Echtzeit-Web-Anwendungen

2.2.1 Begriffserklärungen

2.2.1.1 Kollaboration

„Kollaboration“ kommt aus dem Lateinischen (con- „mit-“, laborare „arbeiten“) und kann frei als Zusammenarbeit mehrerer Personen oder Gruppen übersetzt werden.

Während des zweiten Weltkriegs bekam der Begriff ein negatives Image: man galt als „Kollaborateur“, wenn man mit seinem Feind, meist Nazi-Deutschland, gemeinsame Sache machte. Zur eindeutigen Abgrenzung wird in der deutschen Literatur häufig der englische Ausdruck „Collaboration“ verwendet, der diesbezüglich eindeutig ist. In dieser Thesis wird aber die deutsche Bezeichnung „Kollaboration“ verwendet.

Inzwischen ist der Begriff rehabilitiert und bezeichnet die Zusammenarbeit zeitlich, räumlich oder organisatorisch getrennter Gruppen bzw. Einzelpersonen.

Abzugrenzen ist der Begriff vom Begriff der Kooperation. Nach Leimeister [20] sind sowohl Kooperation als auch Kollaboration Tätigkeiten, die durch mehrere Beteiligte ausgeführt werden. Allerdings steht bei ersterem die Zielerreichung eines jeden Beteiligten im Vordergrund, während bei zweitem die Erreichung eines Gruppenziels verfolgt wird. Leimeister definiert Kollaboration als ein Zusammenspiel aus Koordination, Kooperation und Kommunikation. Dies lässt sich auch in folgendem Schema aufzeigen:

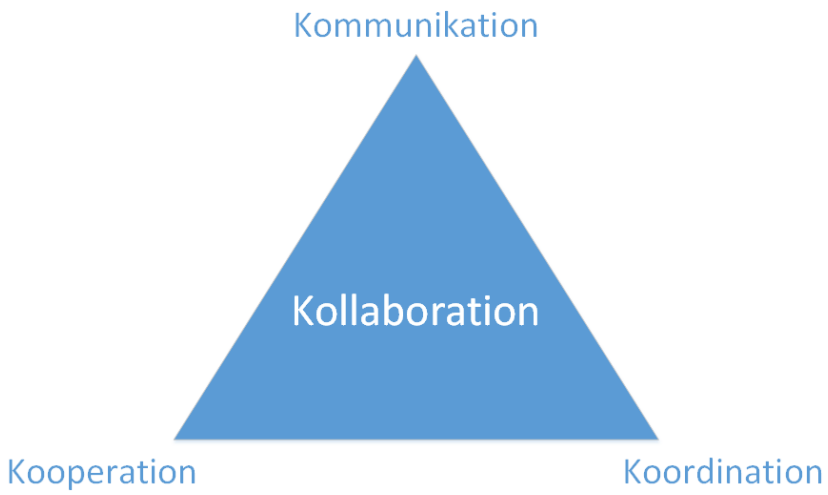

Abbildung 6: Das Zusammenspiel der vier K's [20]

Die Website itwissen.info, die ein Lexikon für Informationstechnologie betreibt, beschreibt Kollaboration in einem pragmatischeren Ansatz als „kooperative Zusammenarbeit von Projektgruppen, Mitarbeitern, Unternehmen und deren Zulieferern und Partnern zur Optimierung der Wertschöpfungskette.“ [21]
Dabei gehört zur Kollaboration „die Verteilung von Informationen und die gemeinsame Nutzung von Ressourcen.“ [21]

Dies veranschaulicht das Online-Lexikon in nachstehender Grafik.

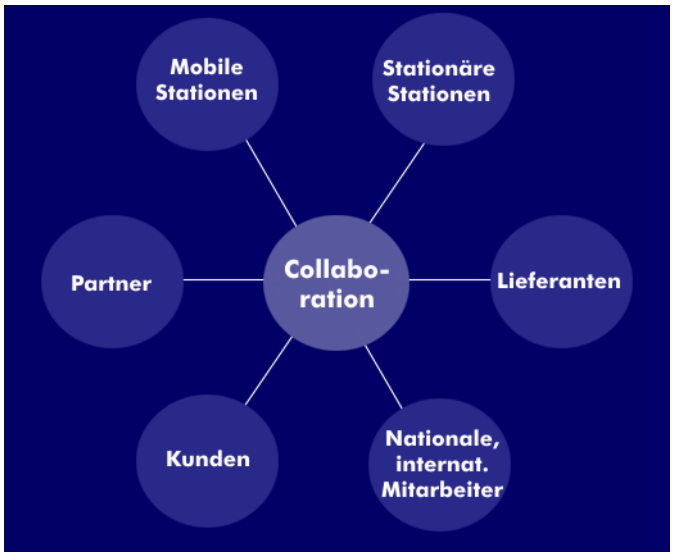

Abbildung 7: Beteiligte an der Kollaboration [21]

Der kollaborative Aspekt des Framework, das im Rahmen dieser Thesis entstehen soll, besteht darin, dass Applikationen eine grafische Oberfläche besitzen, die zwischen allen registrierten Clients stets synchron gehalten und von allen Parteien live editiert werden kann.

2.2.1.2 Echtzeit

Das Deutsches Institut für Normung (DIN) definiert Echtzeit in DIN 44300 folgendermaßen: „Unter Echtzeit versteht man den Betrieb eines Rechensystems, bei dem Programme zur Verarbeitung anfallender Daten ständig betriebsbereit sind, derart, dass die Verarbeitungsergebnisse innerhalb einer vorgegebenen Zeitspanne verfügbar sind. Die Daten können je nach Anwendungsfall nach einer zeitlich zufälligen Verteilung oder zu vorherbestimmten Zeitpunkten anfallen.“ [22] (Verweis hier auf Scholz, 2005, da DIN nur kostenpflichtig abrufbar).

Diese Definition räumt mit dem weit verbreiteten Irrglauben auf, Echtzeit wäre eine Übertragung ohne Zeitverlust. Erstens kann eine Übertragung ohne Zeitaufwand physikalisch nie stattfinden – selbst Licht benötigt dafür ein paar Sekundenbruchteile – zweitens wird Echtzeit laut Definition eingehalten, wenn eine vorgegebene Zeitspanne nicht übertroffen wird.

Übertragen in die Informationstechnik sprechen wir bei der Verwendung von Echtzeit von Realzeitsystemen. Realzeitsysteme sind Systeme, die neben den funktionalen Anforderungen auch zeitlichen Anforderungen genügen. [23]

Ein System ist nach Sebastian Keck definiert als eine „Menge untereinander verbundener Komponenten zur Erfüllung eines Zwecks. Ein System ist von der Umwelt abgegrenzt. Der Austausch von Informationen mit der Umwelt erfolgt durch Ein- und Ausgänge.“ [24]

In diesem Framework gilt als Realzeit der kleinste mögliche zeitliche Abstand, der durch optimale Implementierung der Algorithmen unter Berücksichtigung der physikalischen Gegebenheiten, wie Leitungslänge und Funkübertragungen, erzielt werden kann. Das Realzeitsystem stellt die fertige Applikation dar, die durch Schnittstellen zu den Clients mit ihrer Umwelt interagiert.

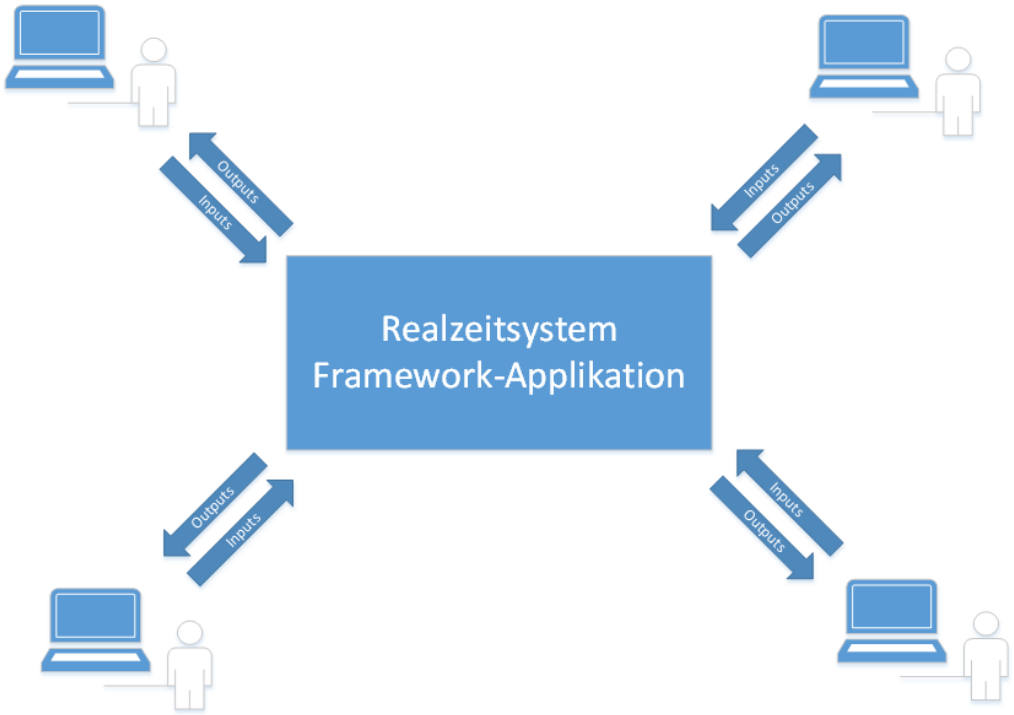

Abbildung 8: Realzeitsystem mit Schnittstellen nach Außen

2.2.1.3 Web-Anwendung

Nach Kappel, Pröll, Reich ist eine Web-Anwendung „ein Softwaresystem, das auf Spezifikationen des World Wide Web Consortium (W3C) beruht und Web-spezifische Ressourcen wie Inhalte und Dienste bereitstellt, die über eine Benutzerschnittstelle, den Webbrowser, verwendet werden.“ [25]

Somit sind Web-Applikationen also Programme, geschrieben in einer der zahlreichen Sprachen des Web wie etwa PHP, die auf einem Webserver liegen. Ein Client verbindet sich über seinen Browser mit dem Server, lädt sich Dateien herunter, die die Web-Anwendung für ihre Funktionalität lokal benötigt und stellt den geforderten Inhalt dar. Je nach Interaktion des Benutzers werden Events oder Reaktionen ausgelöst, die client- oder serverseitig verarbeitet werden und je nach Kontext den gezeigten Inhalt im Browser dynamisch verändern.

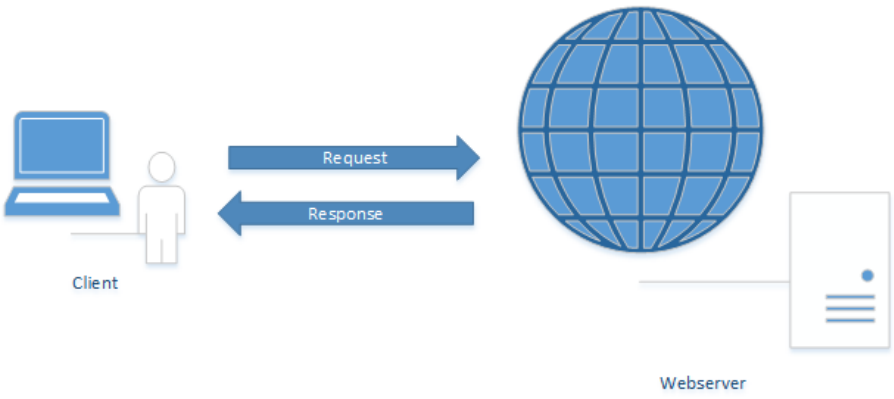

Abbildung 9: Die klassische Client-Server-Architektur

2.2.2 Zusammenarbeit in der Theorie

2.2.2.1 Allgemein

Konzepte der Kollaboration gibt es fast so lange wie den Menschen selbst. Einst wurde im Team gejagt – heutzutage werden über große Distanzen hinweg gemeinsam und produktiv komplexe Arbeitsprozesse absolviert.

Das vorliegende Kapitel orientiert sich an Jan Marco Leimeisters Buch „Collaboration Engineering“, das 2014 im Springer-Verlag erschien und sehr ausführlich und zeitaktuell auf Psychologie, Konzepte und Techniken der Kollaboration eingeht. Außer bei wörtlichen Zitaten wird daher auf sich wiederholende Referenzen verzichtet.

Konzepte der Kollaboration finden ihre Anwendung in der Praxis in sogenannten „Werkzeugen“. Sie lassen sich auf verschiedene Arten klassifizieren. Sehr populär sind zwei Methoden. So lässt sich zum einen nach Grad der IT-Unterstützung und Vernetzung folgende Aufteilung vornehmen:

- **nicht-IT-gestützte Werkzeuge**
  Hiermit sind Systeme gemeint, die nicht von der IT abhängig sind, aber durchaus auch elektronisch sein dürfen. Somit sind Werkzeuge wie Tafel oder Telefon abgedeckt.
- **IT-gestützte Einzelrechner-Werkzeuge**
  Hier sind Werkzeuge gemeint, die auf einem Rechner laufen, aber gleichzeitig von mehreren Personen benutzt werden können. Ein Beispiel wäre hier ein Smartboard, auf dem mehrere Anwender mit „elektronischer Kreide“ gleichzeitig arbeiten können.
- **IT-gestützte Werkzeuge für private Netzwerke**
  In dieser Kategorie geht es um Kollaborations-Tools, die innerhalb geschlossener Netzwerke, wie etwa Firmen-LANs, verwendet werden. Denkbar sind zum Beispiel Text-Editoren.
- **webgestützte Werkzeuge**
  In die vierte Kategorie fallen ähnliche Werkzeuge wie in die dritte, allerdings geht die Vernetzung hier über das LAN hinaus ins Internet. Beispiele sind E-Mails oder Weblogs.

Werkzeuge, die IT-gestützt sind, fallen auch unter das Akronym CSCW, das für „Computer Supported Cooperative Work“ steht.

In dieser Klassifizierung wird nur eingeschränkt auf Zeit- und Ortsfaktoren der verschiedenen Werkzeuge eingegangen. Dies versucht das folgend erläuterte Verfahren besser zu lösen.

Die zweite Möglichkeit ist eine Raum-Zeit-Matrix. Hier werden vier Fälle unterschieden und „Dimensionen“ genannt:

	Gleiche Zeit (synchron)	Unterschiedliche Zeit (asynchron)
Gleicher Ort	**1. Dimension**	**3. Dimension**
Unterschiedlicher Ort	**2. Dimension**	**4. Dimension**

Bei dieser Aufteilung ist nicht relevant, ob ein Konzept IT-gestützt funktioniert oder ob eine Vernetzung nur im LAN besteht oder über das LAN hinaus geht – es zählt schlicht die Frage, ob sich die Partizipanten der Kollaboration während der Arbeit am selben Platz oder an unterschiedlichen Orten aufhalten und ob die Kollaboration gleichzeitig oder zu unterschiedlichen Zeiten stattfindet.

2.2.2.2 Die vier Dimensionen

Im Folgenden wird auf die wichtigsten Vertreter der Raum-Zeit-Matrix eingegangen.

2.2.2.2.1 Erste Dimension

In der ersten Dimension sind Kollaborations-Werkzeuge aufgeführt, die von allen Teilnehmern zur selben Zeit am selben Ort verwendet werden. Da auch ein „gleicher Ort" impliziert ist, entfallen hier vernetzte Werkzeuge.

Diese erste Dimension lässt sich in zwei große Bereiche gliedern. Ein Bereich sind die *Präsentationswerkzeuge*, ein weiterer die *Mehrbenutzer-Editoren*.

Ein wichtiges Element der Kollaboration ist die Präsentation der eigenen Arbeit im Team. Das dient dem Fortschritt, der Kreativität und der Entscheidungsfindung. Gibt es einen zentralen Akteur, so dient es auch der Moderation und in deren Kontext auch der Visualisierung von Daten. So notiert beispielsweise eine zentrale Person hereingerufene Begriffe für eine Mindmap oder ein Brainstorming.

Klassische Werkzeuge zum Präsentieren sind:

- Flipchart – Eine Leinwand mit mehreren Seiten Papier
- Overheadprojektor – Eine Linse, die lichtdurchflutete Folien an die Wand projiziert
- Beamer – Eine Linse, die Daten über analoge und digitale Schnittstellen direkt vom Endgerät (PC, Tablet, Smartphone) an die Wand projiziert

Mit dem Flipchart und dem Overheadprojektor werden hier Tools benutzt, die nicht IT-gestützt sind, während der Beamer ein IT-gestütztes Einzelrechner-Werkzeug darstellt.

Ein weiterer Bereich der Werkzeuge in der ersten Dimension sind die Mehrbenutzer-Editoren. Der Überbegriff klingt sehr technisch, doch ist damit einfach eine lokale Installation gemeint, an der mindestens zwei Personen zugleich arbeiten können.

Nicht IT-gestützte Werkzeuge wären hier:

- Pinnwand – Eine Kork-Platte, die sich mittels Pinnnadeln fixierter Papierzuschnitte wieder und wieder gestalten lässt
- Tafel/Whiteboard – Bemalbare Flächen, an der mehrere Teilnehmer mit Kreide oder Filzstift gleichzeitig arbeiten können

Vertreter der IT-gestützte Einzelrechner-Werkzeuge wären beispielsweise:

- Interaktives Whiteboard oder auch Smartboard (ugs.) – Eine elektronische Tafel, die sich über mehrere Eingabegeräte gestalten lässt
- Tabletop – Komplett berührungsempfindlicher Tisch, dessen Tischfläche mit Handgesten bedient wird

2.2.2.2.2 Zweite Dimension

Die zweite Dimension beinhaltet Kollaborations-Werkzeuge, die von Teilnehmern zur selben Zeit, jedoch an unterschiedlichen Orten genutzt werden. Es geht somit um synchrone, vernetzte Systeme und betrifft damit IT-gestützte Tools für private Netzwerke und das Internet.

Die Werkzeuge der zweiten Dimension lassen sich grob in Kommunikation (Text, Audio und Video), Dateiübertragung und Desktop/Application Sharing einteilen.

Kommunikationswerkzeuge

Bei der synchronen Kommunikation über eine räumliche Distanz fällt das Stichwort „Instant Messaging“. Es bedeutet frei übersetzt „Nachrichten in Echtzeit“ und meint den Austausch von meist kurzen Text-Nachrichten, die ähnlich einem persönlichen Gespräch sehr schnell auf der Gegenseite eintreffen.

Die meisten dieser Systeme unterstützen ebenfalls einen asynchronen Ansatz, da Nachrichten auch zugestellt werden, wenn die Gegenseite nicht in der entsprechenden Applikation online ist. Sie erhalten diese dann, wenn sie die Applikation öffnen.

Populäre Clients finden sich für alle Desktop-Systeme und von allen namhaften Softwareherstellern wie Microsoft, Apple, Google, Yahoo und Facebook. Viele dieser Firmen stellen Clients für alle Plattformen her, einige haben sich aber auf das Smartphone spezialisiert und koppeln den Dienst mit der Mobilfunknummer.

Soll die Kommunikation über Text hinausgehen, so werden Systeme interessant, die Audio- und Videodaten in Echtzeit aufnehmen, von einem Ort zum nächsten übertragen und dort wieder darstellen. Folgende Lösungen erfreuen sich großer Beliebtheit:

- (IP-) Telefonie – Der Klassiker zur reinen Audio-Übertragung seit über 100 Jahren, inzwischen schon oft nicht mehr analog über eine separate Leitung, sondern digital über die Internet-Infrastruktur
- Audio-/Videochat-Software – Ebenfalls angeboten von allen großen Softwarefirmen, hervorstechend Google mit Skype [26]

Dateiübertragung

Bei der Dateiübertragung muss der Begriff der Synchronität relativiert werden, da hier oftmals eine Datei von einem Sender erst auf einen Server hochgeladen wird, bevor der Empfänger sie dann aktiv wieder herunterlädt. Wir gehen hier aber von einem Kontext aus, in dem man zeitlich synchron arbeitet und daher aktiv auf den fertigen Prozess der physikalischen Übertragung wartet und dessen Beendigung direkt zur Weiterarbeit nutzt.

Auch hier gibt es verschiedene Möglichkeiten:

- FTP-Server – Ein FTP-Server dient als temporärer Speicher, auf den der Sender die Dateien hochlädt, die der gewünschte Empfänger dann mittels der URL wieder herunterladen kann.
- Austausch per E-Mail – Auch hier werden Dateien als E-Mail-Anhänge auf dem Mail-Server des Empfängers gespeichert, wo er sich diese dann herunterladen kann.
- Direkte Übertragung – Diverse im Betriebssystem inkludierte Protokolle bieten den Dateiaustausch mittels IP direkt von Computer zu Computer an. Installiert man sich lokal einen Web- oder FTP-Server und macht diesen nach außen hin zugänglich, so kann auch dieser genutzt werden.

Desktop/Application Sharing

Briggs beschreibt diese Unterkategorie wie folgt: „Desktop/Application Sharing Werkzeuge eignen sich dazu, Benutzern den Inhalt von einzelnen Anwendungen oder den Bildschirm für weitere Anwender freizugeben. Dabei können einzelne Bereiche oder der gesamte Bildschirm angezeigt werden, um so den Fokus des angezeigten Inhalts zu steuern. Einige der Anwendungen erlauben zudem die Steuerung der Maus und der Tastatur des ferngesteuerten Computers. Desktop/Application-Sharing-Werkzeuge eignen sich gut für die Durchführung von virtuellen Besprechungen mit interaktiven Präsentationen.“ [27] (Übersetzung nach Leimeister [20])

In dieser Klassifizierung geht es also um Werkzeuge, mit denen man sich auf entfernten Computersystemen anmelden kann, um deren lokale Hard- und Software-Infrastruktur nutzen zu können.

Es gibt drei Protokolle beziehungsweise Lösungen, die in diesem Marktsegment eine breite Unterstützung erfahren. Die ersten beiden basieren auf dem Prinzip, dass auch am ferngesteuerten Computer ein Anwender sitzt und aktiv am Geschehen beteiligt ist. Oft wird hierzu ein Kommunikationskanal wie das Telefon hinzugezogen:

- TeamViewer [28] – Eine proprietäre Lösung, die durch ihre einfache Handhabung besticht. Der Client der TeamViewer GmbH muss heruntergeladen werden, danach kann man sich über eine eindeutige ID mit einem fremden Desktop verbinden und diesen steuern.
- VNC – Über „Virtual Network Computing“ lässt sich ähnlich wie bei TeamViewer der Desktop eines entfernten PCs bedienen. Proprietäre VNC-Clients gibt es für fast jedes Betriebssystem, sie basieren auf dem offenen RFB-Protokoll.

Die dritte Lösung verfolgt einen anderen Ansatz: Es geht nicht mehr darum, gemeinsam an einem Schreibtisch zu arbeiten, sondern entfernt installierte Applikationen zu nutzen:

- Remote Login – Mit diesem Tool kann man sich aus der Entfernung mit einem Endgerät bei einem anderen anmelden, etwa vom Privat-PC zuhause auf dem geschäftlichen PC am Arbeitsplatz. Das Zielgerät muss zu diesem Zweck natürlich immer eingeschaltet bleiben. Es lassen sich fortan alle Netzverbindungen und Applikationen nutzen, als wäre man direkt vor Ort.

2.2.2.2.3 Dritte Dimension

In diesem Viertel der Raum-Zeit-Matrix geht es um Werkzeuge, die sich zwar am gleichen Ort befinden, nicht jedoch von Kollaborateuren zur gleichen Zeit bedient werden. Auch in dieser Kategorie entfallen somit wieder vernetzte Werkzeuge der Kollaboration.

Unter diesen Punkt fallen sehr wenige Werkzeuge. Hauptsächlich geht es hier um Tools der ersten Dimension (synchron, gleicher Ort), welche aber nicht gleichzeitig, sondern asynchron verwendet werden. Dies schränkt den Nutzen der Werkzeuge in der Regel stark ein und wird nicht angewendet.

Im Bereich der Mehrbenutzer-Editoren sind Tafeln, Whiteboards oder ein klassisches „Schwarzes Brett“ vorstellbar, die stationär installiert sind und mit Post-Its, Filzstiften und Kreide zur Arbeit „on the fly“ einladen, etwa Ideensammlungen oder ähnliches. Danach muss eine zeitliche Distanz zum nächsten Kollaborateur geschaffen werden, zum Beispiel durch sofortiges Verlassen des Raumes.

Leimeister erwähnt außerdem den Anrufbeantworter als IT-gestütztes Werkzeug, das an einem Ort steht, und zu unterschiedlichen Zeiten besprochen und abgehört wird – allerdings wird ein Anrufbeantworter mit einem Werkzeug der zweiten Dimension, nämlich einem entfernten Telefon besprochen, was nicht ganz der Definition „gleicher Ort“ entspricht.

2.2.2.2.4 Vierte Dimension

Die vierte Dimension beschäftigt sich mit den Kollaborations-Werkzeugen, die ortsunabhängig und nicht gleichzeitig genutzt werden. Durch die Ortsunabhängigkeit betrifft diese Dimension die Kategorie der IT-gestützten Werkzeuge in privaten Netzwerken sowie die webgestützten Werkzeuge, was relativ viele Tools mit einschließt.

In dieser Kategorie wird auf Werkzeuge der Kommunikation, der Präsentation sowie Mehrbenutzereditoren, Versionsverwaltung, Social Tagging und Abstimmungstools eingegangen.

Kommunikationswerkzeuge

Zum asynchronen Austausch von Nachrichten in LANs oder dem Internet haben sich drei Werkzeuge etabliert:

- E-Mail – Der Klassiker der elektronischen Kommunikation. 1971 das erste Mal benutzt [29], ist die digitale Postkarte heute in der Privat- und Geschäftswelt gang und gäbe.
- Forum – Foren stellen Plattformen dar, über die unzählige Mitglieder in sogenannten „Threads“ Beiträge zu Themen verfassen können. Die Beiträge erscheinen chronologisch und bilden meist schriftlich geführte Gruppengespräche ab.
- Weblog – Blogs stellten in ihrer ursprünglichen Form digitale Tagebücher dar, inzwischen ist der Kreativität keine Grenze mehr gesetzt. Blogs dienen zur Verbreitung von Wissen, Meinungen und Ansichten, Tutorials und ähnlichem. Obligatorisch ist die Kommentarfunktion, mittels der berechtigte Leser den Beitrag diskutieren können.

Präsentationswerkzeuge

Inzwischen gibt es zahlreiche Möglichkeiten, seine Arbeitsergebnisse online zu präsentieren. Diese können dann über ihre URL von geneigten Kollegen aufgerufen werden. Auch hier gibt es verschiedene Ansätze:

- Präsentation als Video – In diesem Fall werden die Folien der Präsentation einfach in ein Video konvertiert und jeweils ein paar Sekunden stehen gelassen. Dieses Video lässt sich zum Beispiel auf die Webvideo-Plattform YouTube hochladen.
- Präsentation mittels proprietärer Plug-Ins – Diverse Anbieter haben Präsentationssoftware implementiert, die es erlaubt, Folien hochzuladen oder sogar direkt im Browser zu gestalten Die zugrunde liegende Technologie sind oft externe Plug-Ins wie Adobe Flash.
- Präsentation mit offenen Browser-Standards – auch im Präsentations-Genre gibt es bereits Lösungen, die auf HTML5-Techniken wie etwa CSS3-Transformationen basieren.

Mehrbenutzer-Editoren

In der Kategorie der zeitlich und räumlich unabhängigen Mehrbenutzer-Editoren geht Leimeister speziell auf einen ein: das Wiki.

Das Wiki ist in seiner populärsten Form auf wikipedia.de [30] zu betrachten. Es stellt eine Wissensdatenbank dar, die gleichwohl intern in Unternehmensnetzwerken als auch global im Internet benutzt werden kann. Es läuft im Browser. In ihm lassen sich Artikel zu bestimmten Überschriften schreiben und abspeichern. Je nach Rollenvergabe können andere Besucher den Artikel nicht nur lesen, sondern auch ergänzen und korrigieren.

Typisch für Wikis ist eine Zusammenfassung im ersten Abschnitt, darauf folgend eine anklickbare Inhaltsübersicht und schließlich die Information im Fließtext. Charakteristisch ist ebenfalls, dass bestimmte Schlüsselwörter im Fließtext mit einem Hyperlink hinterlegt sind, der auf eine weitere Artikel-Seite verweist. Dort wird ebenjener Begriff vertiefend behandelt. So stellt ein Wiki eine Vielzahl statischer Seiten dar, die untereinander vernetzt sind.

Versionsverwaltung

Die asynchrone und ortsunabhängige Zusammenarbeit birgt die Gefahr, dass mehr als ein Teilnehmer gleichzeitig auf ein Objekt zugreift, also beispielsweise zwei Mitarbeiter gleichzeitig einen Wiki-Artikel überarbeiten möchten. Sie laden sich beide die aktuelle Version in den lokalen Cache ihres Browsers herunter, fügen ihre Änderungen ein und laden ihre Version hoch. Leider wissen die beiden neuen Versionen nichts voneinander und so wird die zuletzt hochgeladene die vorhergehenden überschreiben.

Eine einfache Lösung, dieses Problem zu beheben, ist der Ansatz, das Laden/Öffnen des Dokuments zu blockieren, solange es woanders geöffnet ist. Dieses Konzept ist aber wenig komfortabel, da man unter Umständen sehr lange warten muss, bis man etwas editieren kann.

Ein moderner Ansatz ist die zentrale oder verteilte Versionsverwaltung, bei der mehrere Kollaborateure gleichzeitig arbeiten können. Ihre Dokumente werden versioniert und anhand dieser Angabe in der richtigen Reihenfolge zusammengefügt und konsistent gehalten.

Social Tagging

Der Begriff des Social Tagging steht für die Verschlagwortung gemeinsamer Ressourcen. „Dabei geht es nicht in erster Linie um die Generierung neuer Inhalte, sondern um die Anreicherung vorhandener Inhalte wie Bookmarks mit Informationen“ [20]. So sind etwa gemeinsam genutzte, online gehostete Bookmark-Sammlungen gemeint, die kollaborativ erweitert, gepflegt und verschlagwortet werden.

Abstimmungstools

Zum Schluss seien noch die Abstimmungstools erwähnt, die es möglich machen, in einer zentralen, funktionellen Umgebung mit geringem Zeitaufwand Entscheidungen zu treffen. Inzwischen sind viele Lösungen zentral im Internet gehostet. Ob sich die Abstimmung mehr im Bereich der Umfragen bewegt oder zur Ermittlung eines gemeinsamen Termins dient, ist dabei ganz dem Ersteller überlassen. Zu definierten Zeitpunkten sind die Abstimmungen nicht mehr erreichbar und präsentieren dem Ersteller das Ergebnis.

2.2.3 Stand der Technik

Nun sind die elementaren Begrifflichkeiten geklärt und einige Konzepte der Kollaboration aufgezeigt – doch mit welchen Techniken werden kollaborative Echtzeit-Web-Anwendungen heutzutage realisiert? Wie wird Echtzeit verwirklicht und wie werden Daten zwischen Kollaborateuren ausgetauscht? Mit dieser Fragestellung beschäftigt sich dieses Kapitel.

Was bedeutet Echtzeit bei Web-Applikationen im technischen Sinne? Es geht darum, dass der Server dem Client neue Daten in dem Moment schickt, in dem sie generiert wurden. Demnach dürfen neue Informationen nicht nur auf der Seite erscheinen, wenn der Benutzer in eigenem Rhythmus den „Reload“-Button drückt. Stattdessen sollte der Client die Informationen erhalten, sobald sie auf dem Server zur Verfügung stehen.

Zurzeit finden in der Praxis vier Wege Anwendung, um Echtzeit-Kommunikation zwischen Client und Server zu realisieren. Hierbei wird zwischen Pull- und Push-Verfahren unterschieden. Bei den Pull-Verfahren muss der Client beim Server erfragen, ob neue Informationen verfügbar sind, beim Push-Verfahren kann der Server Informationen aktiv an den Client versenden, sobald diese verfügbar sind:

- Pull-Verfahren
  - AJAX Polling
  - AJAX Long-Polling
- Push-Verfahren
  - WebSockets
  - Server-Sent Events

AJAX steht für „Asynchronous JavaScript and XML“ und macht es möglich, Teile einer Website dynamisch nachzuladen, anstatt immer die komplette Webseite neu laden zu müssen. AJAX folgt dabei dem Konzept der asynchronen Datenübertragung. HTTP-Requests werden dabei nicht wie üblich direkt von der Benutzeroberfläche an den Server geschickt und eins zu eins mit Responses beantwortet, sondern stattdessen über eine in JavaScript geschriebene AJAX-Engine abgefangen und wunschgemäß weiter verarbeitet.

Ein normaler HTTP -Request folgt diesem Schema und lädt immer die komplette Seite:

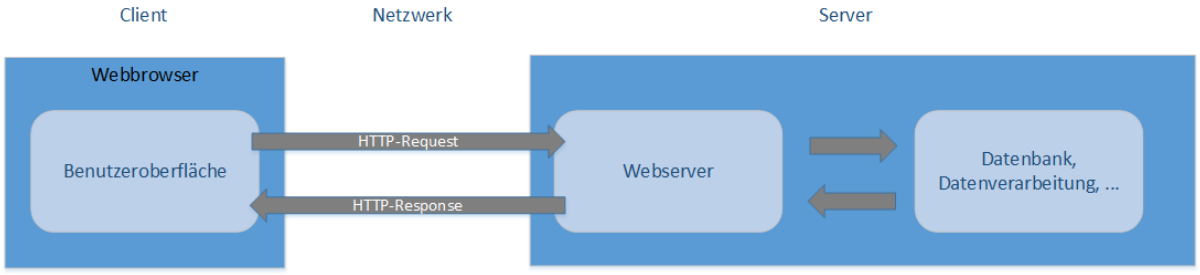

Abbildung 10: HTTP Client-Server Architektur

Eine AJAX-Architektur lädt Teile der Website über JavaScript nach:

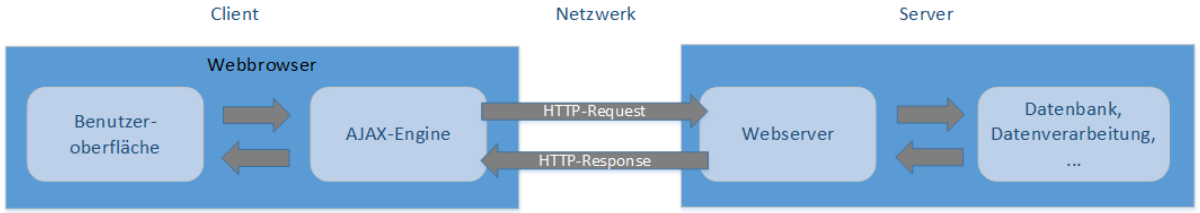

Abbildung 11: AJAX Client-Server Architektur

AJAX verfolgt zwei Konzepte, um Daten dynamisch nachzuladen. Man unterscheidet zwischen Polling und Long Polling. Beides sind aber Pull-Konzepte, das heißt, der Client muss aktiv Requests an den Server schicken, um Änderungen in der Response mitgeteilt zu bekommen. Die Requests sind Mittel zum Zweck und enthalten keine „Nutzdaten“.

Polling

Beim Polling wird dem Client mitgegeben, in welchem Rhythmus er einen HTTP -Request an den Server schicken soll, um nachzusehen, ob Änderungen im Datenbestand erfolgt sind.

So schaut er beispielsweise alle 0,5 Sekunden, ob eine neue Wetterinformation des vom Client betrachteten Gebiets vorliegt. Im positiven Fall wird diese über die Response nachgeladen und dem User in „Echtzeit“ angezeigt, im negativen Fall informiert die Response darüber, dass nichts Neues eingetroffen ist.

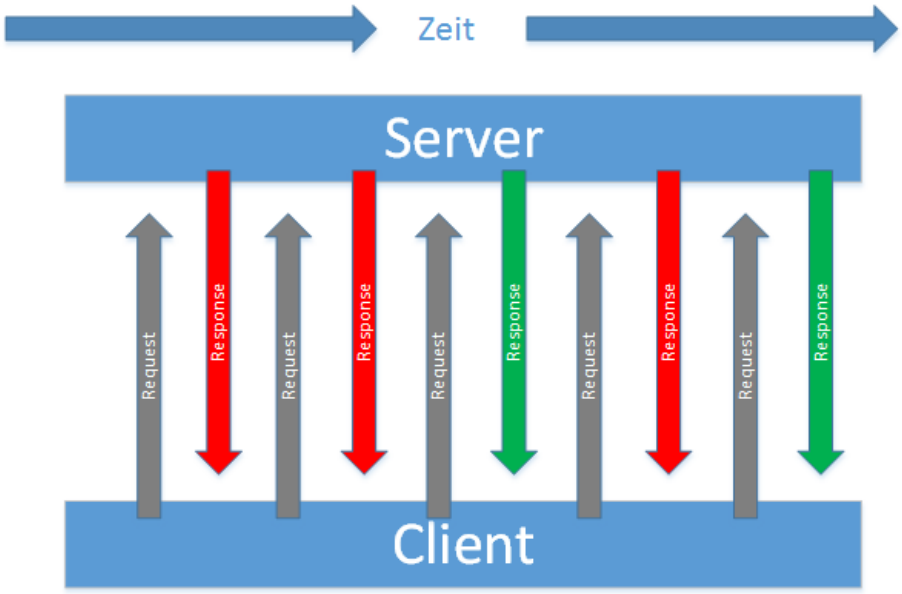

Abbildung 12: Polling

Diese Architektur erfordert eine sensible Taktung der Requests. Ist die Zeitspanne zu kurz gewählt, so wird unnötig viel Datenverkehr produziert, ist die Zeitspanne zu lange gewählt, so können Änderungen mit zu großer Latenz auf dem Client ankommen. Aufgrund dieser Gratwanderung und der generell nicht vorhandenen Eleganz des Algorithmus wird von diesem Konzept heutzutage weitgehend abgesehen.

Long-Polling

Anders gelöst wird dies mittels Long Polling. Hier wird parallel zum eigentlichen Seitenaufruf ein zweiter HTTP -Request gestartet, der vom Server erst dann beantwortet wird, wenn Änderungen erfolgt sind. Wurde die Response geschickt, startet der Client einen neuen HTTP -Request für die nächste Änderung.

Auch diese Requests dienen nur dem Zweck, dem Server eine Response-Möglichkeit zu bieten. Allerdings wird gegenüber dem Polling das Datenaufkommen drastisch reduziert und Änderungen treffen am Client ein, sobald sie auf dem Server zur Verfügung stehen.

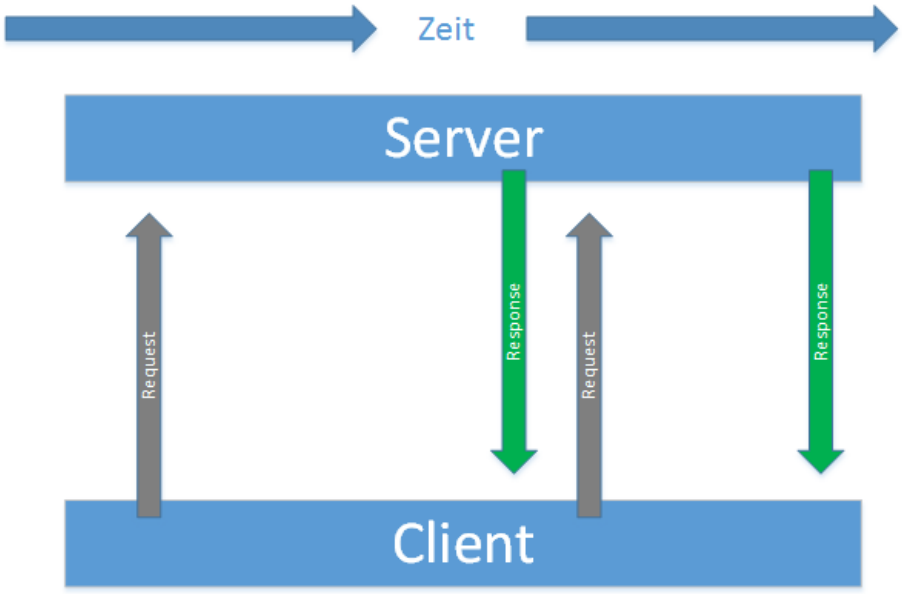

Abbildung 13: Long-Polling

Aber auch dieses Verfahren hat Nachteile: So öffnet der eingehende Request einen Thread, der Ressourcen belegt, die erst wieder freigegeben werden, wenn die Response gesendet wurde. Long-Polling-Anwendungen bekommen so Probleme mit der Skalierung, wenn sehr viele Clients die Anwendung gleichzeitig benutzen.

Weiterhin wird die Idealvorstellung Request → Event → Response vom HTTP -Protokoll zunichte gemacht, da dieses eine Timeout-Funktion besitzt, die nach einer bestimmten Wartezeit als Response einen HTTP -Statuscode 408 „Request Timeout“ sendet. So müssen doch mehr Requests und Responses gesendet werden, als dies rein konzeptuell nötig wäre.

AJAX ist „State of the Art“ für dynamisches Nachladen von Inhalten, Nun wurden im Zuge von HTML5 WebSockets und Server-Sent-Events annonciert, die dessen Schwächen auszugleichen versuchen.

WebSockets

Die Technologie der WebSockets, mit der auch dieses Framework arbeitet, ist erst fünf Jahre alt. Sie sorgt für einen Voll-Duplex-Kanal zwischen Client und Server. Sie wurde 2009 das erste Mal vorgestellt und ist noch dabei, sich zu etablieren. [31] Das neue WebSocket-Protokoll wurde inzwischen standardisiert, die Standardisierung der WebSockets selbst ist noch in Arbeit. Dennoch unterstützen derzeit fast alle Server-Systeme und alle Browser die Technik. Im Detail werden Funktionalität und Kompatibilität in Kapitel 3.1.2 erläutert.

Durch den Voll-Duplex-Kanal ist es dem Server möglich, zu jeder von ihm gewählten Zeit Daten an den Client zu schicken, ohne dass dieser nach ihnen fragt. Umgekehrt gilt das gleiche. Damit übertragen bis auf einen initiierenden HTTP -Request alle ein- und ausgehenden Nachrichten echte Nutzdaten.

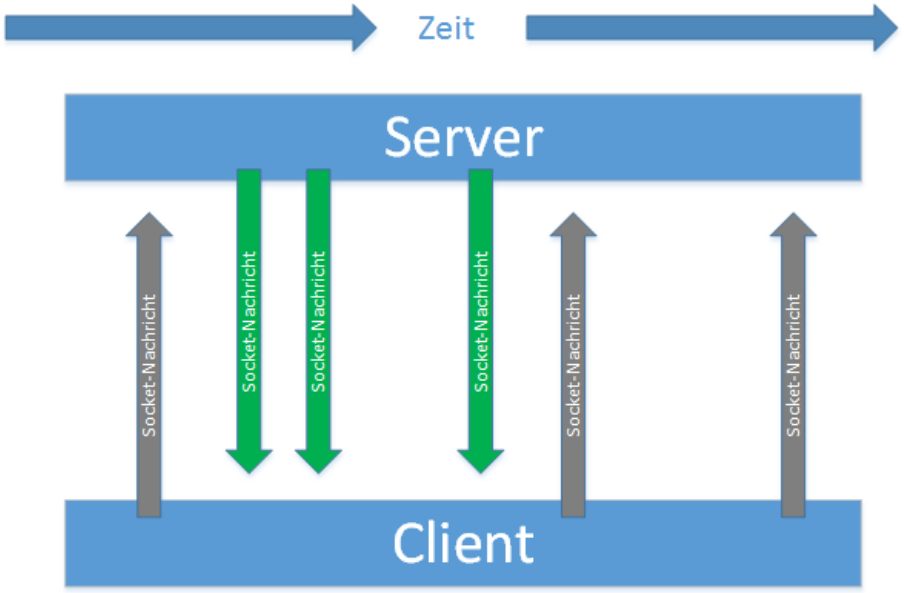

Abbildung 14: WebSockets

WebSockets bieten sich für Web-Applikationen an, die einerseits auf Echtzeit-Informierung des Clients angewiesen sind, die aber andererseits auch sehr schnelle und kleine Datenpakete vom Client an den Server schicken müssen. Damit sind WebSockets das Mittel der Wahl, um viele interaktive Clients in Echtzeit miteinander zu synchronisieren.

Abgesehen davon, dass der Server autark agieren kann, ist eine der größten Stärken der WebSockets ihr eigenes, auf sie optimiertes Protokoll. So müssen sie nicht über das HTTP - Protokoll kommunizieren, welches eine große Menge an Header-Informationen mit sich führt. In Kapitel 3.1.2 wird der Aufbau einer WebSocket-Nachricht aufgezeigt. Hier werden Metadaten in Höhe von maximal 14 Byte benötigt. Bei HTTP liegen diese durchschnittlich zwischen 700-800 Byte. [32] Den Vorteil, der hieraus gezogen wird, zeigt folgende Grafik auf:

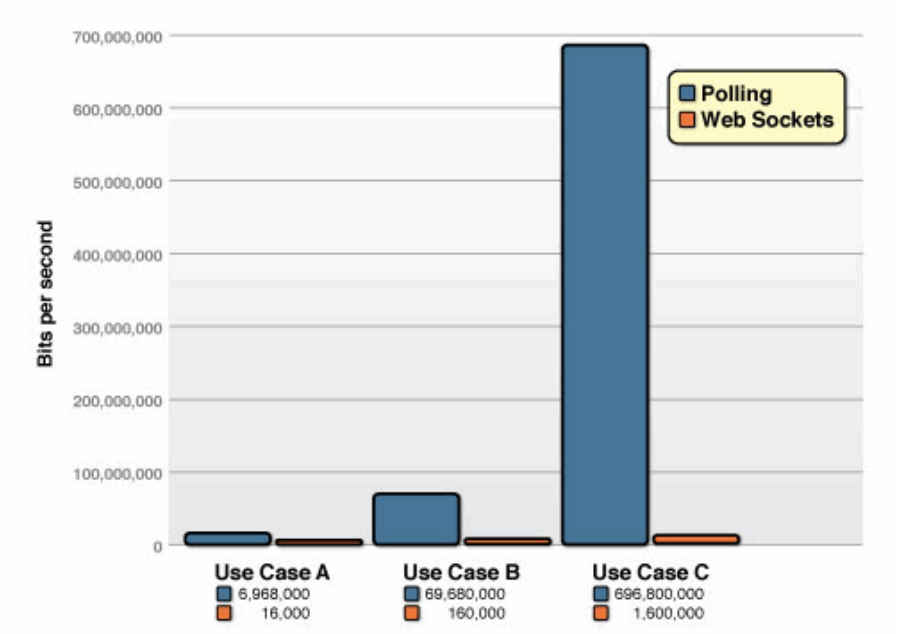

Abbildung 15: Datenrate WebSockets vs. Polling bei 1k, 10k und 100k Clients [33]

Im dargestellten Szenario greifen 1.000 (A), 10.000 (B) und 100.000 (C) Clients auf eine Web-Anwendung zu.
Der blaue Balken zeigt den Datendurchsatz bei Kommunikation über Polling, der rote den über WebSockets. Beim Polling entstehen extrem hohe Datenraten durch zahlreiche Requests und große HTTP-Header, während WebSockets auch bei hohen Client-Anfragen sehr gut skalieren.

Ähnlich wie die WebSockets funktionieren die proprietären Adobe Flash Sockets [34].

Server-Sent Events

Beim vierten und letzten Konzept geht es um Server-Sent Events, die ebenfalls mit HTML5 eingeführt wurden. Auch hier hat der Server die Möglichkeit, dem Client jederzeit Daten zuzusenden. Die Option, SSEs zu versenden, ist aber – das besagt schon der Ausdruck – dem Server vorbehalten, weswegen das Verfahren als simplex bezeichnet wird.

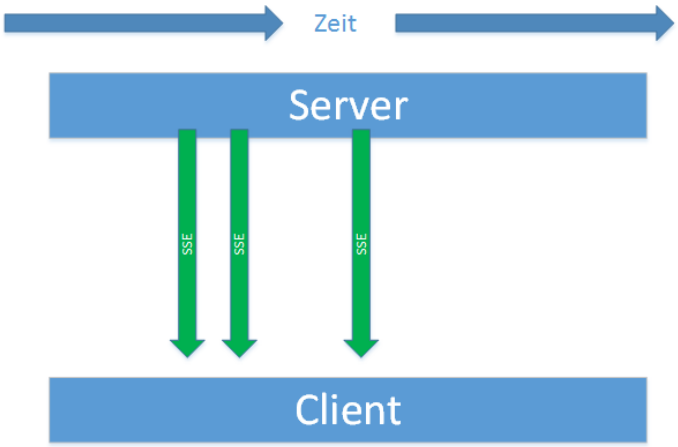

Abbildung 16: Server-Sent Events

Aufgrund dieser Tatsache haben Server-Sent Events großes Potenzial bei Realzeit-Informationsportalen, etwa bei Aktienkursen oder News-Tickern, sind aber nicht für Applikationen geeignet, in denen es darauf ankommt, dass auch User-Aktivitäten sehr schnell und mit geringer Latenz an den Server gesendet werden.

2.2.4 Praxisbeispiel

Bis auf das klassische Polling werden alle drei Technologien gegenwärtig in der Praxis eingesetzt. Folgend ein Beispiel für die in dieser Thesis verwendeten WebSockets:

Etherpad Lite

Ursprünglich war EtherPad ein kollaborativer Echtzeit-Text-Editor, der über AJAX feststellte, ob andere Kollaborationspartner etwas am Text geändert hatten und aktualisierte die Ansicht des betrachtenden Clients. Er wurde 2008 veröffentlicht und im Dezember 2009 von Google aufgekauft. Zu diesem Zeitpunkt wandelte Google das Projekt in ein Open-Source-Projekt. Als Google die Entwicklung des originalen EtherPad ein halbes Jahr später zu Gunsten eines eigenen Services einstellte, entstanden über den offenen Quellcode zahlreiche Eigenentwicklungen.

2009 wurde die Idee von EtherPad von einem Open-Source-Team neu aufgegriffen und *Etherpad Lite* entstand. [35] Die neue Version basiert auf Node.js und kommuniziert mit dem serverseitigen Framework über WebSockets.

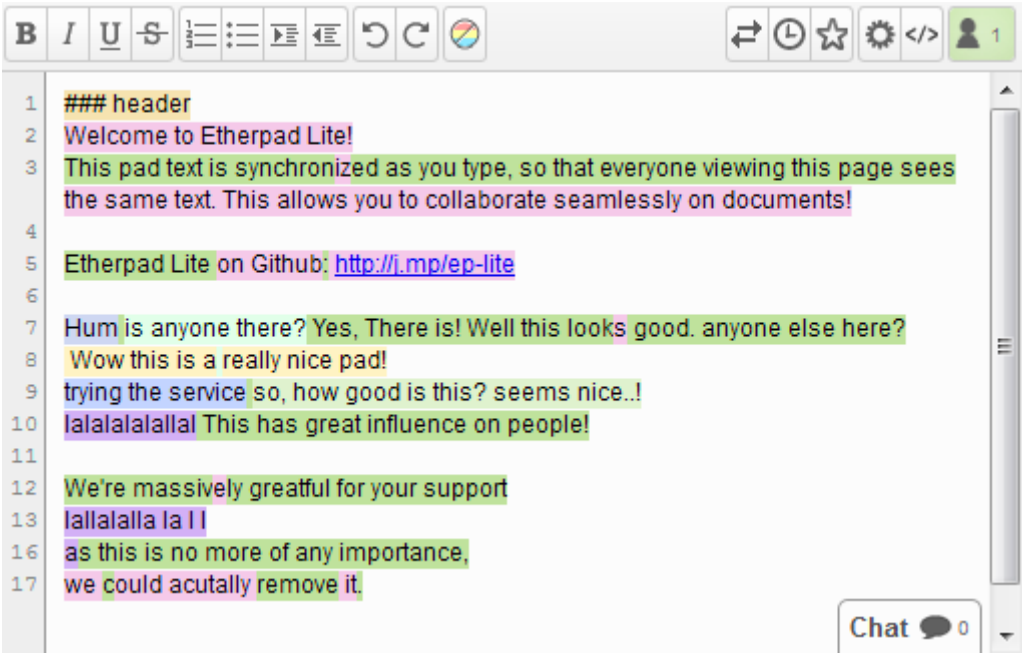

Abbildung 17: Etherpad Lite [35]

Die verschiedenen Farben im Text dienen zur Identifizierung des jeweiligen Autors.

3 Relevante Webtechnologien

Im den folgenden Unterkapiteln werden die Technologien und Frameworks vorgestellt, die in dieser Thesis verwendet werden. Sie sind unterteilt in die Basis-Technologien und Frameworks, die auf diesen Technologien basieren.

3.1 Basis-Technologien

3.1.1 HTML5

HTML5 gilt als der Nachfolger von HTML 4.01. Letzteres ist der aktuelle Standard für Auszeichnungssprachen im Internet und wurde 1999 verabschiedet und seitdem nicht mehr aktualisiert. HTML5 ist noch nicht standardisiert, wie ein rotes „Work-in-Progress"-Banner auf der Seite des W3C, des World Wide Web Consortiums, verrät [36], dennoch sind erste Entwürfe der WHATWG seit 2004 im Umlauf und werden seitdem kontinuierlich weiterentwickelt. [37]

Führende Browser, namentlich Google Chrome, Mozilla Firefox, Microsoft Internet Explorer, Apple Safari und Opera [38] haben es sich zur Aufgabe gemacht, die Features, die dieses HTML-Upgrade mit sich bringt, zu implementieren und Entwicklern wie auch den Konsumenten zur Verfügung zu stellen.

In den vergangenen zehn Jahren hat sich hier einiges getan, Konzepte wie das „Canvas", das bereits 2004 spezifiziert wurde, finden breite Unterstützung in allen Browsern [39], neuere Konzepte, wie etwa das `<track>`-Element, das zum ersten Mal Ende 2010 eingeführt wurde, werden von Firefox und Safari noch nicht unterstützt [40].

Im Folgenden gibt es einen kurzen Überblick über die Neuerungen von HTML5 gegenüber HTML 4, außerdem wird das Canvas als zentrales Element dieser Thesis genauer diskutiert. Auf einen genaueren Überblick über die Browserkompatibilität von HTML5 folgen Praxisbeispiele und ein Ausblick auf künftige Versionen von HTML.

3.1.1.1 Kurzer Überblick über Neuerungen

HTML 4.01 wurde seit seiner Verabschiedung 1999 nicht mehr aktualisiert. Die W3C arbeitete einige Zeit an XHTML 1.1 und 2.0 als Nachfolger, erkannte dann aber, dass HTML5, das bis dahin unter dem Namen „Web Applications 1.0" von der WHATWG parallel entwickelt wurde, die besseren Konzepte beinhaltete. So übernahm die W3C 2006 offiziell die Version der WHATWG als Grundlage für HTML5 und seit 2007 arbeiten beide Institutionen gemeinsam an dessen Weiterentwicklung. [41]

Doch in was bestanden die Änderungen der WHATWG gegenüber der Vorversion 4.01 und welche neuen Konzepte wurden bereits von den Browsern implementiert?

Abbildung 18: Die Neuerungen in HTML5 optisch dargestellt [42]

Eine große Unterstützung hält HTML5 für das „Semantische Web“ parat, den Ansatz, das Internet inhaltlich sortieren, gliedern und bewerten zu können.
Da Suchmaschinen menschliche Sprachen weder verstehen, noch interpretieren können, muss der Entwickler hier nachhelfen. Elemente wie `<header>`, `<footer>`, `<article>`, `<section>`, `<aside>` und `<nav>` helfen technischen Systemen, Webseiten nicht nur im Volltext zu durchsuchen und wiederzugeben, sondern diese auch strukturell nachzuvollziehen und gewissen Bereichen eine höhere Bedeutung zu geben als anderen. [43]
Eine Folge dieses Ansatzes ist auch das Support-Ende von Gestaltungs-Elementen wie `<font>`, `<big>` oder `<center>`. [44] Derlei Eigenschaften sollen nur noch per CSS definiert werden. Das Ziel ist, in der HTML-Datei nur noch Inhalt zu strukturieren und die Optik separat zu definieren.

Auch bei den Formularen hat sich einiges getan. Es gibt zahlreiche neue Input-Types, etwa „email“ oder „number“. [45] Außerdem wurden in HTML5 einfache Validatoren nativ implementiert. Wo vorher mühsam einzelne Zeichen ausgelesen und überprüft wurden, lässt sich mit Attributen wie etwa „required“ überprüfen, ob ein Feld ausgefüllt wurde oder nicht [46]. Ein Input-Feld wie `<input type="text" required>` würde ein „Invalid“-Event auslösen, wenn es beim Absenden des Formulars noch leer ist. Der `<input type="email">`, überprüft beim Senden automatisch, ob die Eingabe nach dem Schema xxx@xxx.xx erfolgt ist.

Im grafischen und multimedialen Bereich wartet HTML5 unter anderem mit dem Element `<canvas>` auf. Das Canvas ist ein Kernbestandteil dieser Thesis und wird im nächsten Kapitel ausführlich behandelt. Neben dem Canvas liefern die Elemente `<audio>` und `<video>` die Grundlage, um multimediale Inhalte darstellen zu können. Mit ihnen werden Bewegtbilder und Tondateien nativ in die Webseite eingebunden. [47, 48]
Auch der Support für SVG-Grafiken wurde verbessert. So lassen sich diese jetzt als Inline-Elemente einbinden und werden mit einfachen Attributwerten im HTML-Dokument gestaltet.

HTML5 hat eine enge Bindung zu JavaScript. Das Canvas wird mit JavaScript gestaltet, außerdem bietet HTML5 zahlreiche neue APIs, die über JavaScript angesprochen werden. Die Programmiersprache ist Thema des Kapitels 3.1.3.

Das Geolocation-API ermöglicht die clientseitige Ermittlung des Standorts des Endgeräts, das auf die Webseite zugreift. Unabhängig vom verwendeten Gerät für den Zugriff wird über installierte GPS-Systeme, die IP Adresse, RFID, WiFi, Bluetooth oder GSM/CDMA-Zellen-Informationen versucht auf den Standort zu schließen. [49]

Mit dem Drag-and-drop-API lassen sich Elemente sogar über Webseiten hinweg „draggen und droppen“. Dies ermöglicht es, Abhängigkeiten zu externen Frameworks wie jQuery zu vermeiden. [50]

Der „Web Storage“ stellt ein API dar, mit der sich Name-Wert-Paare lokal auf dem Client speichern lassen, ähnlich wie bei Cookies. Allerdings werden diese Name-Value-Paare nie auf den Server übertragen und können einen wesentlich höheren Speicherplatz beanspruchen als Cookies. Man unterscheidet zwischen localStorage-Objekten, die kein Ablaufdatum besitzen und sessionStorage-Objekten, die beim Schließen des Browserfensters gelöscht werden. [51]

Ebenfalls um lokale Speicherung geht es beim „Application Cache“. Hier werden Dateien auf Client-Seite gespeichert und zur Verfügung gestellt, wenn der Client offline ist. Damit sind etwa Bilder, CSS- oder JavaScript-Dateien gemeint. [52] Somit lassen sich Web-Applikationen entwickeln, die eine komplette Offline-Funktionalität aufweisen.

Abschließend seien noch die „Web Workers“ und die „Server-Sent Events“ erwähnt.
„Web Workers“ ist eine Bezeichnung für JavaScript-Funktionen, die unabhängig von der Interaktion auf der Website selbst im Hintergrund laufen und die Performance der Website nicht beeinflussen. [53]
„Server-Sent Events“ geben dem Server die Möglichkeit, dem Client Daten zu schicken, ohne dass dieser per Request vorher darum gebeten hat. [54]
Server-Sent Events haben auf den ersten Blick eine ähnliche Funktionalität wie WebSockets. Der exakte Unterschied wird im Kapitel 2.2.3 herausgestellt.

3.1.1.2 Das Canvas im Speziellen

Eine der wesentlichsten Weiterentwicklungen in HTML5 ist die Einführung des „Canvas“. Laut Kröner kann man es „sich als programmierbares `<img>` vorstellen“. [44] Es stellt eine Leinwand im Browser dar, die mittels JavaScript gestaltet werden kann. Die entsprechenden JavaScript-Events können dabei hard-coded, dynamisch oder durch User-Eingaben mit Maus-, Tastatur- und Touch-Eingaben ausgelöst werden.

Ein einfaches Canvas lässt sich wie folgt aufziehen:

```
<canvas width="480" height="280>
      Fallback-Content
</canvas>
```

JavaScript zeichnet nicht auf dem `<canvas>` Element direkt, sondern auf einem „Rendering Context“ davon. Diese Schnittstelle stellt die JavaScript-Methoden und -Attribute bereit, um auf dem Canvas zu malen. Bis jetzt steht nur ein 2D-Kontext bereit, ein 3D-Kontext ist aber in der Entwicklung. Alternativ lassen sich mit dem WebGL-API 3-dimensionale Elemente auf dem Canvas zeichnen. [44, 55]

Erstellen wir nun also eine HTML-Datei mit folgendem Inhalt:

```
<canvas id="canvas" width="480" height="280>
      Ihr Browser ist leider zu alt
</canvas>
```

Und dazu eine JavaScript-Datei mit folgendem Inhalt:

```
var canvas = document.getElementById('canvas');
var context = canvas.getContext('2d');
context.fillStyle='#FF0000';
context.fillRect(30,40,50,50);
```

Das resultierende Bild kann auf der nächsten Seite betrachtet werden (zur Veranschaulichung wurde dem Canvas mittels CSS noch einen schwarzen Rahmen gegeben):

```
canvas {
      border:1px solid #000000;
}
```

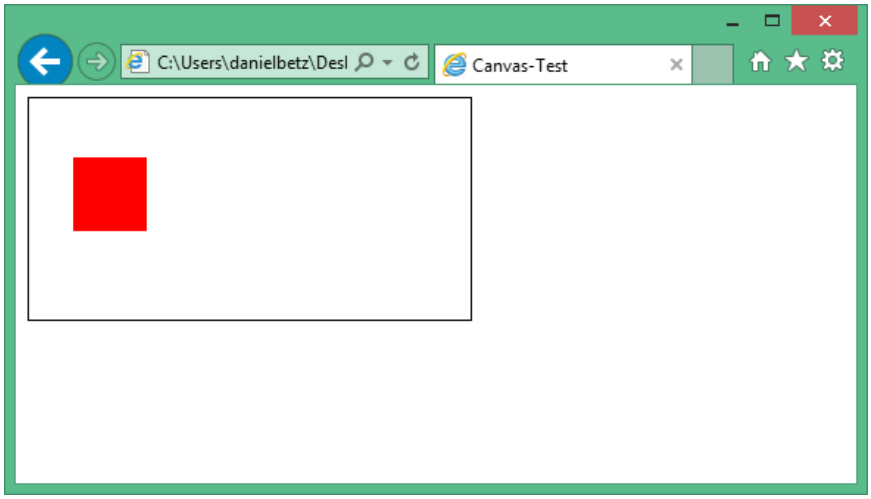

Abbildung 19: Canvas 480x280 Pixel mit rotem Rechteck

Damit erhalten wir ein Canvas der Größe 480x280 Pixel. Sollte der Browser noch nicht neu genug sein, um das Canvas darstellen zu können (siehe nächstes Kapitel „Browserkompatibilität“), so greift der Browser automatisch auf den Fallback-Content zu und stellt diesen dar. In diesem Beispiel würde ein Benutzer des IE9 oder älter den Text „Fallback-Content“ in seinem Fenster sehen, da das Canvas vom IE erst ab Version 10 unterstützt wird.

Die Argumente, die hier an fillRect() übergeben werden, sind
`fillRect(X-Position, Y-Position, Breite, Höhe).`

Davor wurde mittels `.fillStyle` noch eine Farbe festgelegt.

Basierend auf diesem Beispiel lässt sich ein Canvas mit einer Vielzahl an Formen und Parametern gestalten. Neben Rechtecken gibt es Funktionen für Farben, Stile und Schatten, Pfade, Transformationen, Text, Bilder, Pixel-Manipulation und Compositing. [39]

Die Werte für Breite und Höhe im Tag definieren die absolute Größe des Canvas und lassen sich wie beim `<img>` Tag per CSS skalieren. Eine Skalierung nach oben sorgt für Qualitätsverluste beim gerenderten Bild. Eine Skalierung nach unten hat einen positiven Effekt auf hochauflösende Endgeräte. So wird das Canvas im HTML mit einem Vielfachen der Pixel definiert und gezeichnet und per CSS auf die tatsächliche Größe verkleinert. Den Effekt sieht man bei Geräten wie etwa Apples Retina Displays bei iPad und MacBook Pro, außerdem äußert er sich beim Drucken. Nachstehendes Beispiel zeigt diesen Fall auf.

In der ersten Abbildung wurde das Canvas mit 500x500 Pixeln definiert und per CSS nicht beeinflusst.

```
<canvas id="myCanvas" width="500" height="500">
      Fallback
</canvas>
```

Darauf platzieren wir diverse Formen und Textbausteine:

```
<script>
      ctx.font = '15px Arial';
      ...
</script>
```

Werden diese Grafiken auf dem iPad Retina im Browser betrachtet, verschwimmen die Kanten:

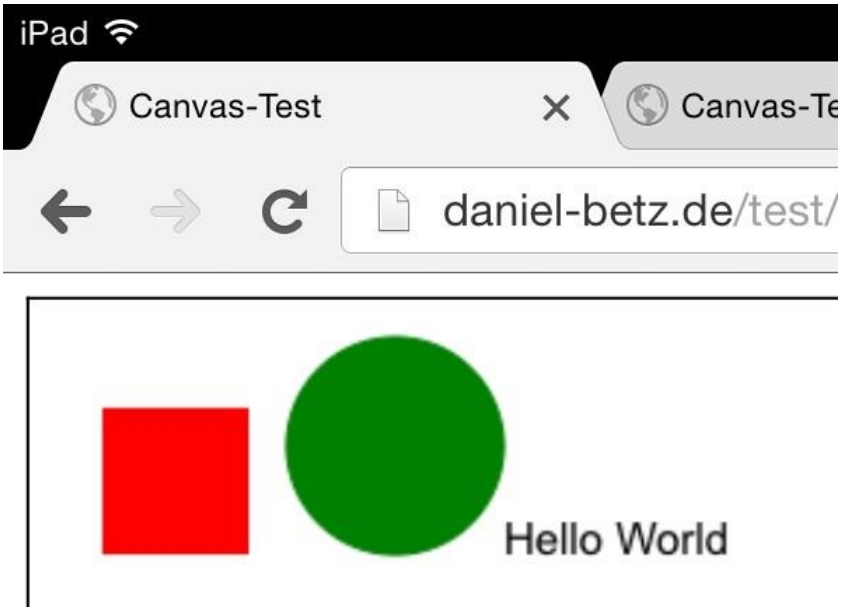

Abbildung 20: iPad Retina – Canvas nicht skaliert

Nun definieren wir das Canvas in der vierfachen Größe:

```
<canvas id="myCanvas" width="2000" height="2000">
```

Ebenso die dargestellten Elemente:

```
ctx.font = '60px Arial'; ...
```

Per CSS skalieren wir es auf unsere gewünschten 500x500 Pixel:

```
canvas {
      width: 500px;
      height: 500px;
}
```

Auf dem iPad Retina sind die Formen nun wesentlich schärfer:

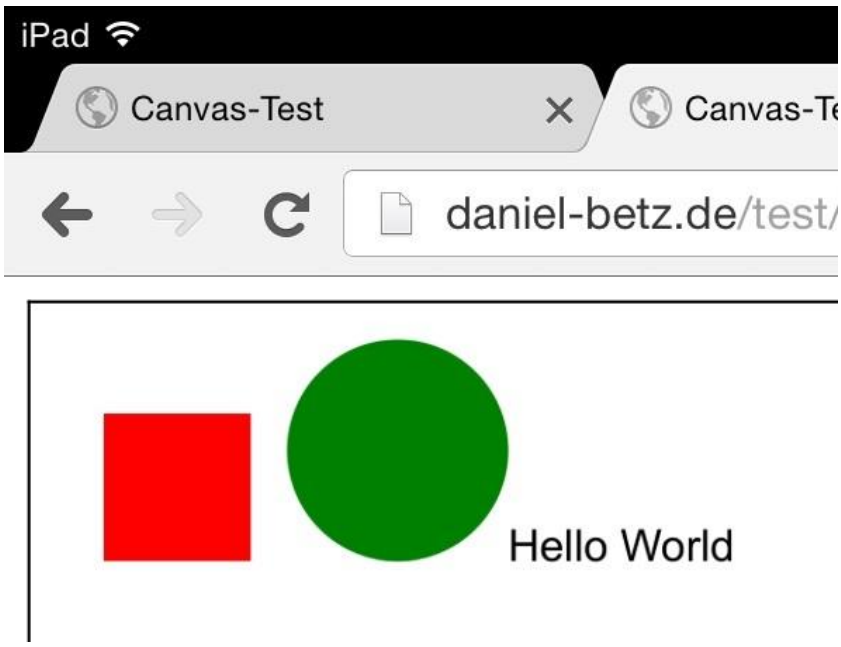

Abbildung 21: iPad Retina – Großes Canvas herunterskaliert

3.1.1.3 Browserkompatibilität

Alle gängigen Browserhersteller entwickeln ihre Produkte ständig weiter, um kompatibel zu den Entwicklungen der WHATWG/W3C zu bleiben und ihren Kunden einen möglichst großen Funktionsumfang zu bieten.

Die Seite www.caniuse.com [56] analysiert seit langem die Change-Logs der Browserupdates und dokumentiert deren Unterstützung für die jeweiligen Komponenten von HTML5.

Die Betrachtung neuer Funktionen soll hier auf solche beschränkt werden, die entweder exemplarisch oder für die Realisierung des Projekts notwendig sind. Eine komplette Liste ist unter der referenzierten URL zu finden. Der Stand dieser Analysen ist April 2014. Natürlich wird sich die Unterstützung im Laufe der Monate weiter verbessern. Als Beispiele wurden jeweils ein Repräsentant der Kategorie „Gute Unterstützung“, „Mittelmäßige Unterstützung“ und „Schlechte Unterstützung“ gewählt.

Canvas

Das HTML5-Canvas wurde 2004 das erste Mal eingeführt und wird von allen aktuellen Browsern unterstützt.

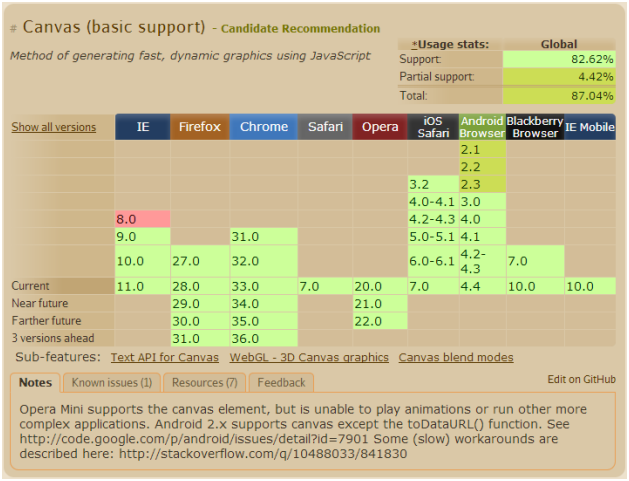

Abbildung 22: Browsersupport Canvas [57]

Form Validation

Das JavaScript-API wird von den mobilen Browsern größtenteils nicht oder nur teilweise unterstützt. Eine Ausnahme bildet hier der Blackberry Browser. Die Desktop-Browser unterstützen die Form Validation bis auf Apple Safari durchgängig.

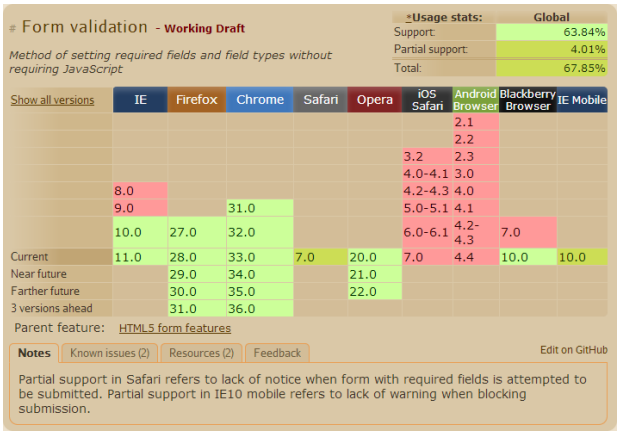

Abbildung 23: Browsersupport Form Validation [58]

Date & Time Input Types

Zu den neu entwickelten Formular-Feldern gehören auch die Typen „Datum“ und „Zeit“, die Eingaben nur in einem entsprechenden Datums- oder Zeit-Format entgegennehmen. Der Browsersupport ist hier noch sehr schlecht:

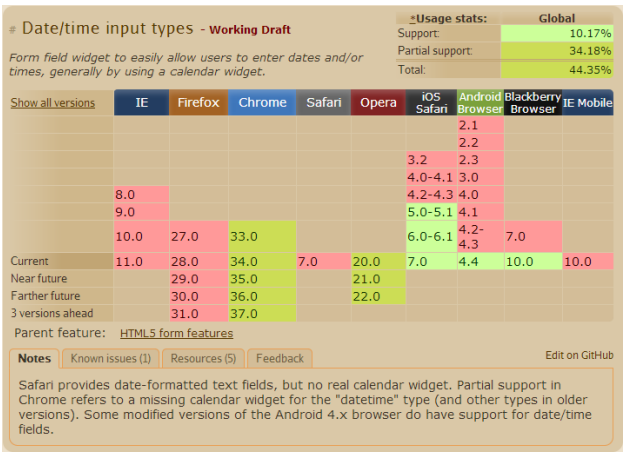

Abbildung 24: Browsersupport der Date/Time Input Types [59]

3.1.1.4 Praxisbeispiele

Die größte Online-Video-Plattform weltweit, youtube.com, bietet große Teile ihres Content-Angebots neben einer Flash-basierten Version inzwischen auch in HTML5 mittels `<video>`-Element an.

Wenn man sich in den Entwickler-Tools des Browsers lange genug durch die verschachtelten `<div>`s klickt, entdeckt man das zentrale `<video>`-Element.

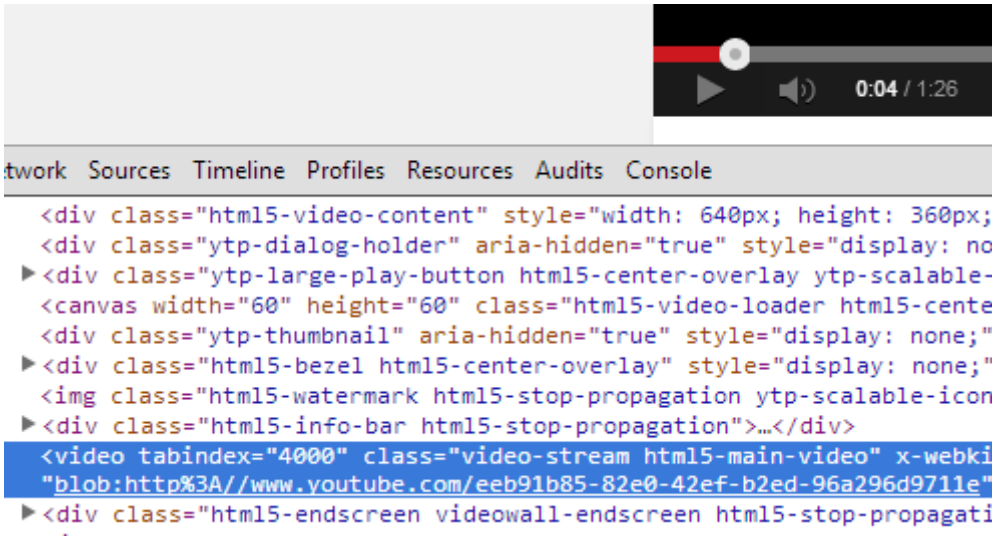

Abbildung 25: Das <video>-Element bei YouTube [60]

Die Web-Anwendung „Sketchpad“ ist ein Zeichenprogramm auf Canvas-Basis. Dort lassen sich durch gängige Mouse- und Keyboard-Events komplexe Zeichnungen anfertigen. Sogar der Shortcut STRG-Z funktioniert und sorgt dafür, dass die letzte Aktion rückgängig gemacht wird. Wenn man fertig ist, kann man sich das Bild als PNG abspeichern.

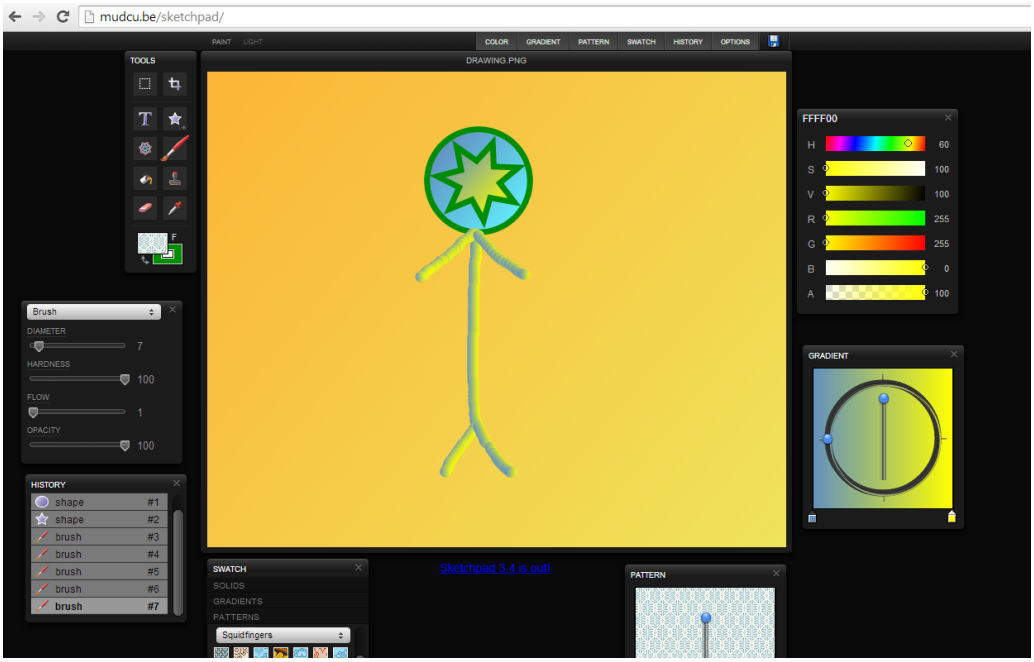

Abbildung 26: UI-Elemente, mit denen das Canvas beeinflusst wird [61]

3.1.1.5 Ausblick - Künftige Versionen von HTML

Wie bei Produktreihen namhafter Hardware-Hersteller wird auch bei HTML schon vor dem „offiziellem“ Erscheinen der nächsten Version über die übernächste diskutiert.

Zahlreiche Autoren und Blogger formulieren ihre Wünsche [62], aber auch von der W3C selbst gibt es bereits eine Wiki-Seite, die als Ideensammlung für die nächste Version von HTML dient. [63]

Auch die WHATWG, eine Institution, die HTML kontinuierlich weiterentwickelt und ihre Vorschläge der W3C präsentiert, hat hierzu einiges an Input. [64, 65]

Laut WHATWG wird die nächste Version von HTML nicht HTML6 heißen. Man wird stattdessen Abstand von der Versionierung nehmen und „HTML“ als Namen etablieren. Dieser stehe für eine Auszeichnungssprache, die nicht als Momentaufnahme standardisiert und an die Entwickler und Konsumenten verteilt wird, sondern einem kontinuierlichen Entwicklungsfluss unterliegt. Sie spricht von HTML als einem „Living Standard“.
Die WHATWG, die parallel zur W3C ihre eigene Spezifikation von HTML pflegt, praktiziert dies bereits offiziell. [66]

Sehr viel Aufsehen hat ein Element namens `<device>` erregt. Es wird in einem Dokument der WHATWG namens „r4439“ eingeführt. [67] `<device>` soll als Selektor für an den Rechner angeschlossene Peripherie gelten. Mit einem type-Attribut lässt sich die Hardware näher spezifizieren, etwa eine Video-Kamera. So lassen sich zum Beispiel live Bewegtbilder auf einer Website anzeigen, ohne proprietäre Plug-Ins wie Adobe Flash zu benutzen.
Außerdem gehen Überlegungen hin zu einem `sandbox`-Attribut für iFrames oder sogar ganze Dokumentbereiche, die mit `<sandbox></sandbox>` umschlossen werden. [68] Dieses Element sorgt dafür, dass der entsprechende Bereich strengeren Rechten unterliegt, beispielweise darf er das DOM nicht manipulieren.
Beide Elemente sind noch konzeptuell, teilweise wurden sie auch schon wieder verworfen oder ersetzt.

Die W3C bewegt sich weiter in die Richtung, in die sie sich bisher bewegt hat. Es werden viele neue semantische Tags vorgeschlagen, etwa `<location>`Ortsangabe`</location>`. Im medialen Bereich soll es weitere native Steuerelemente für Video und Audio-Elemente geben, etwa „Zurückspulen“ oder „Frame vorwärts/Frame rückwärts“, außerdem Balancing der Audioinhalte nach links und rechts. Auch das Verhalten von Formularen soll weiter entwickelt werden, zum Beispiel dahin gehend, dass eine intelligente Großschreibung den ersten Buchstaben des Vor- und Nachnamens automatisch groß schreibt.

Viele dieser Ideen und Vorschläge sind noch Gedankenexperimente, bei einigen kann man sich sicher auch über den Sinn streiten. Man darf auf jeden Fall gespannt sein, welche Konzepte wie in Zukunft von den großen Browserherstellern umgesetzt werden. Ersichtlich ist auf jeden Fall, dass HTML aktuell gepflegt und weiterentwickelt wird. Webentwickler orientieren sich nicht an dem zurückgebliebenen Standard von vor 15 Jahren, sondern nutzen aktiv die „inoffiziellen“ Neuerungen, die seit 2004 in den gängigen Browsern implementiert werden. Da weder von Browserherstellerseite noch von Entwicklerseite der Zwang existiert, sich an den Standard von 1999 zu halten, entsteht eine Dynamik, deren Resultat ein permanenter Fortschritt ist.

3.1.2 WebSockets

Im Kapitel 2.2.3 wurde die WebSocket-Technologie bereits grob umrissen, hier wird sie näher betrachtet.

WebSockets waren zu Beginn ein Teil der Spezifikation von HTML5, wurden aber inzwischen von der W3C in ein eigenes Standardisierungsverfahren übertragen. Das WebSocket-Protokoll ist bereits seit 2011 von der IETF standardisiert [69], das WebSocket API der W3C befindet sich aber noch im Entwurfsstatus. [70]

Server-Sent Events, ebenfalls eine Neuerung von HTML5, besitzen ein paar funktionelle Ähnlichkeiten zu WebSockets. Gemeinsamkeiten und Unterschiede wurden bereits in Kapitel 2.2.3 herausgearbeitet. Anwendungen für SSEs sind reine Informationsdienste, wie ein Aktienkurs-Ticker oder ein Twitter-Feed-Update, während WebSockets prädestiniert für Anwendungen sind, die Informationen von vielen Usern bekommen und diese in Echtzeit an alle weiterleiten, etwa bei einem Chat oder einem Multiplayer-Spiel.

3.1.2.1 Konzept

Verbindungsaufbau

Besucht ein Client eine Website, die eine Socket-Verbindung erwünscht, so bekommt er per HTTP JavaScript-Code geliefert, der ihn clientseitig für die persistente Verbindung vorbereitet.

Es wird folgender HTTP -Request an den Server gesendet: [71]

```
GET /chat HTTP/1.1
Host: 141.79.33.200:3000
Sec-WebSocket-Version: 13
Sec-WebSocket-Key: lFlADxzOzN5gDR2nj2lyog==
Connection: Upgrade
Upgrade: websocket
```

Dies ist ein normaler, standardkonformer HTTP 1.1 Request, allerdings mit dem Spezial-Feld „Upgrade“ im Header. Damit signalisiert der Client, dass er gerne auf ein anderes Protokoll wechseln möchte. [72] Um diesen Wechsel später zu authentifizieren, generiert der Client bereits einen Schlüssel, den er mitschickt.

Der Server antwortet daraufhin mit dem Statuscode 101, was bedeutet, dass die Umleitung akzeptiert wurde. [73] Außerdem sendet er einen `Accept`-Schlüssel, der mittels dem `WebSocket-Key` vom Client berechnet wurde.

```
HTTP/1.1 101 Switching Protocols
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Accept: yuYsDJmoAHyqstpiLiAl8mgAuq8=
```

Der Client berechnet den `Accept`-Schlüssel parallel. Sind der selbst berechnete und der vom Server berechnete Schlüssel gleich, weiß der Client, dass der Server das Protokoll wirklich versteht und die persistente Verbindung wird geöffnet.

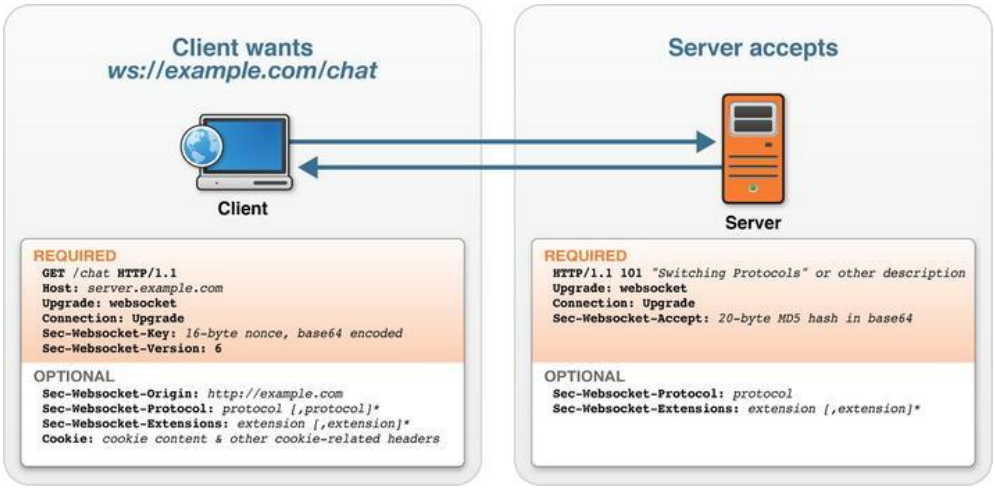

Abbildung 27: Header-Felder bei einem WebSocket-Verbindungsaufbau [71]

Austausch von Nachrichten

Wenn die Verbindung aufgebaut wurde, lauschen Client und Server auf sogenannte onmessage-Events. Diese werden ausgelöst, wenn der Kommunikationspartner ein send-Event abschickt.

Wird auf Client-Seite beispielsweise eine Socket-Verbindung auf der Variable `ws` initiiert, so lässt sich mit `ws.send` auf dem Server ein `onmessage`-Event auslösen.

```
var ws = new WebSocket('ws://localhost/chat')
ws.onopen = function(e) {
      ws.send('Hallo');
}
```

In diesem Beispiel wird ein Objekt abgeschickt, das den String „Hallo“ enthält.

Serverseitig gibt es ebenfalls eine Variable `ws`, die dem Socket-Client zugewiesen ist:

```
ws.onmessage = function(e) {
      console.log('Nachricht erhalten: ' + e.data);
};
```

In diesem Codeschnipsel speichert der Server den Event in der Variable `e` und greift mittels `e.data` direkt auf den gesendeten String zu.

Im folgenden Bild sehen wir, wie eine WebSocket-Nachricht zusammengesetzt ist und welcher Teil des Events mit `e.data` angesprochen wird (Feld zur Rechten).

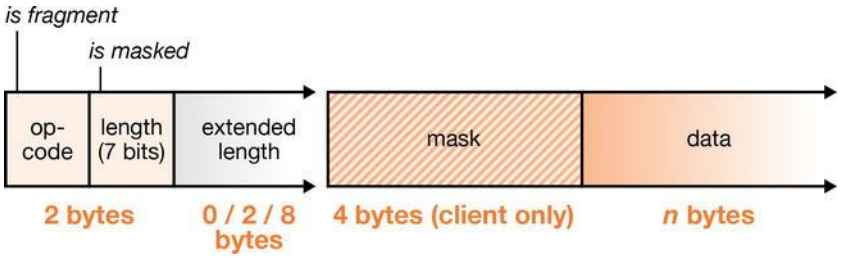

Abbildung 28: Zusammensetzung einer WebSocket-Nachricht [71]

Außer dem `data`-Feld enthält eine Nachricht, bzw. in der WebSocket-Welt ein „Frame“, noch Felder für „opcode“, „length“ und „mask“.

- Opcode: Numerischer Bezeichner für den Typ der Nachricht (beispielsweise „1“ für Text)
- Length bzw. Extended Length: Länge des kompletten Frames in Bits
- Mask: Sicherheitsmechanismus zum Verschleiern von `data`.

Verbindungsabbau

Für den Verbindungsabbau sendet der jeweilige Initiator einfach einen speziellen Opcode in einer Nachricht und die Gegenseite ist informiert. Tatsächlich gibt es 13 Arten, eine Verbindung zu beenden, beispielsweise „1000“ – Normal Close oder „1007“ – Invalid Data.

3.1.2.2 Browserkompatibilität

WebSockets sind nach Polling und Longpolling die erste „wirkliche“ Echtzeit-Option, um Informationen zwischen Client und Server auszutauschen. Die Web-Gemeinde zeigt großes Interesse, was auch Google Trends bestätigt.

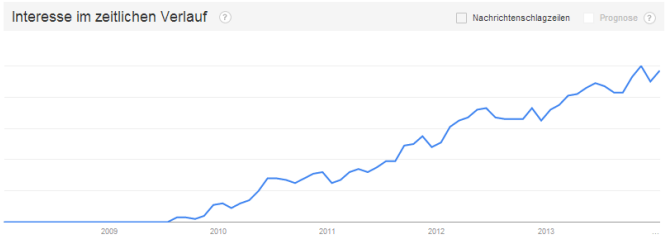

Abbildung 29: "WebSocket" bei Google Trends [74]

Diesem Trend folgend implementierten die Browserhersteller die Neuerung sehr früh in ihre Produkte, und heute wird sie in jedem Browser unterstützt.

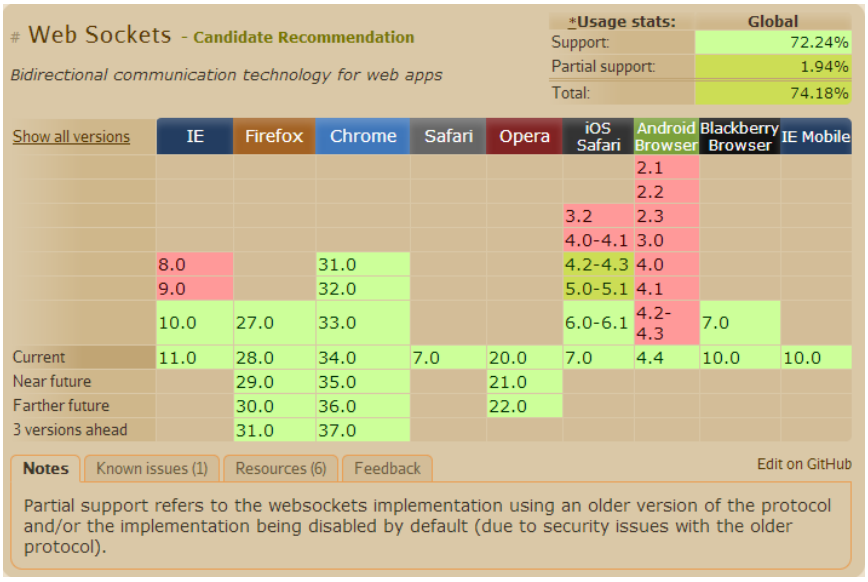

Abbildung 30: Browser-Unterstützung für WebSockets [75]

3.1.2.3 Praxisbeispiele

Viele WebSocket-Applikationen, wie auch die, die durch dieses Framework entstehen sollen, dienen dazu, Oberflächen synchron und in Echtzeit an alle angemeldeten Sockets zu

übertragen. Da für diese Anwendung in Kapitel 2.2 bereits ein Beispiel genannt wurde, werden hier zwei Applikationen vorgestellt, die auch für einzelne Sockets interessant sind.

Ein Anbieter eines Frameworks für Echtzeitkommunikation führt zwei Demos an. So werden die Aktienkurse in dieser Applikation in Echtzeit aktualisiert. Da der User gewisse Interaktionsmöglichkeiten hat, die auf dem Server registriert werden, kann dieses Beispiel nicht mit SSEs realisiert werden.

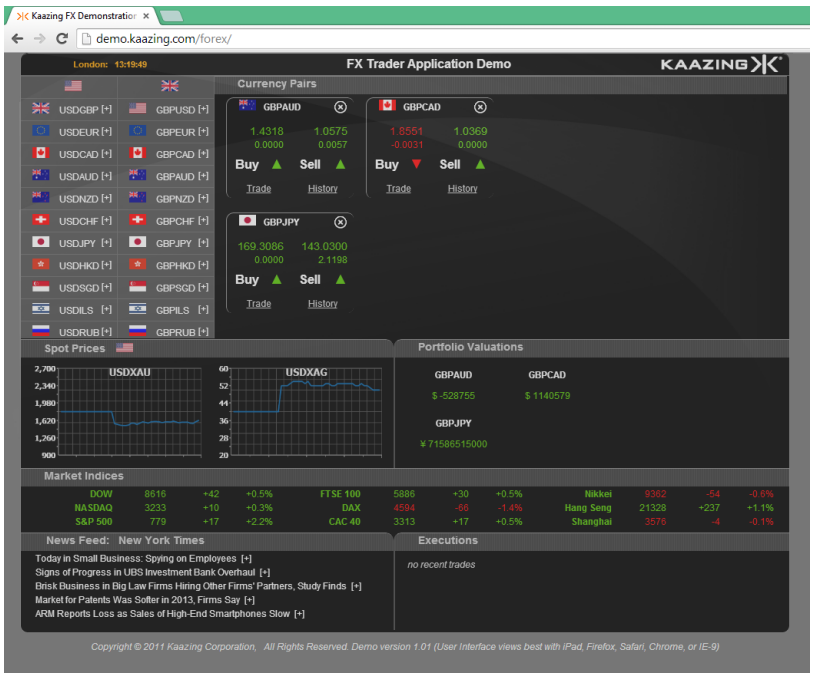

Abbildung 31: Aktienkurse in Echtzeit per WebSockets [76]

In einem weiteren Beispiel werden Server-Systemstatus, wie etwa CPU-Auslastung, von einem WebSocket-Server ausgelesen und „live“ an die verbundenen Sockets gestreamt. Dies wäre auch ein klassischer Anwendungsfall für SSEs.

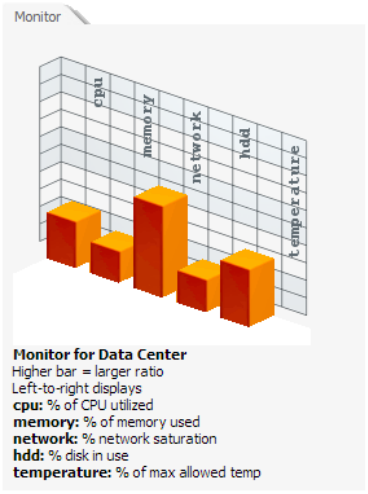

Abbildung 32: Hardware-Monitor per WebSockets [77]

3.1.3 JavaScript

JavaScript ist die aktuelle Standard-Programmiersprache für die clientseitige Entwicklung im Browser. JavaScript arbeitet hauptsächlich prozedural, bietet aber auch einige objektorientierte Ansätze.

„Alles in JavaScript ist ein Objekt“. [78]

Tatsächlich sind alle Methoden, Attribute und Variablen Objekte, mit Ausnahme der drei primären Datentypen Booleans, Zahlen und Zeichenfolgen. Objektorientierte Konzepte wie Vererbung und Kapselung lassen sich manuell implementieren, es gibt aber auch Bibliotheken, die das übernehmen.

3.1.3.1 Entwicklung

Mitte der Neunziger war es das Ziel des Browserherstellers Netscape, eine Skriptsprache zu entwickeln, die es jedem möglich machte, auf seinen Webseiten Skripte einzusetzen. So entstand „LiveScript“. Da zeitgleich die Programmiersprache „Java“ von Sun veröffentlicht wurde und einen großen Hype erlebte, entschieden sich die Entwickler von LiveScript um Branden Eich, diesen Hype zu nutzen und die Sprache in „JavaScript“ umzubenennen. [79]

Dies gilt auch als Geburtsstunde der „Browserkriege“, da Netscape mit seinem Netscape Navigator 2.0 und der Implementierung von JavaScript zum ersten Mal eine ernste Bedrohung für Microsofts Quasi-Monopol für Software darstellte. So veröffentlichte Microsoft 1995 den ersten Internet Explorer zusammen mit der Skriptsprache JScript, das einen Dialekt von JavaScript darstellt. Microsoft konnte aufgrund seiner finanziellen Möglichkeiten sowie der hohen Verbreitung seines Betriebssystems mit ab da vorinstalliertem Browser den Marktanteil des Netscape Navigators von über 80% auf unter 4% senken und gleichzeitig den des Internet Explorer auf 95% steigern. [80]

Im Juni 1997 wurde JavaScript unter dem Namen „ECMAScript“ standardisiert, was an sich einen Fortschritt bedeutete; dennoch bereiteten den Entwicklern die verschiedenen Implementierungen der Browser sowie der parallel gepflegte Dialekt JScript weiter Schwierigkeiten.

Microsoft JScript Version 5 und JavaScript Version 1.5 waren dann äquivalent zu ECMAScript 3 [81] und ein paar ECMAScript-Versionen später verhalfen Unternehmen wie Google der Sprache mit ihren Gratisdiensten auf JS-Basis zu zusätzlicher Beliebtheit. [79]

Mit ECMAScript 5 aus dem Jahre 2011 haben wir schließlich einen aktuellen Standard, der als Basis von JavaScript und seinen Dialekten gilt. JavaScript ist bei der Entwicklung dynamischer Webseiten nicht mehr wegzudenken und wird kontinuierlich weiterentwickelt. Es wird aktuell an ECMAScript 6 gearbeitet [82], welches zum Ende 2014 veröffentlicht werden soll. [83]

3.1.3.2 Bedeutung, Benutzung

JavaScript wird von allen gängigen Browsern unterstützt und lässt sich somit auf fast jedem Client nutzen. Einige wenige User schalten JavaScript aktiv aus, sodass eine Statistik ca. 98% Unterstützung von JS clientseitig bescheinigt. [84]

JavaScript lässt sich einfach mit `<script type="text/javascript"></script>` direkt in einer HTML-Datei programmieren, alternativ lässt sich im gleichen Tag mittels dem `src`-Attribut eine externe JavaScript-Datei mit Endung .js einbinden.

Design/Oberfläche

JavaScript lässt sich nutzen, um die Oberfläche, die dem User im Browser angezeigt wird, auf dessen Reaktionen hin zu verändern. Einfache Manipulationen verändern CSS-Werte, komplexe Veränderungen sind etwa Slideshows, die stehen bleiben, wenn der User mit der Maus darüber fährt, außerdem lassen sich mit JavaScript Untermenüs beim Öffnen animieren oder Bilder exponieren, auf denen der Fokus der Maus liegt.

Funktionalität clientseitig

Als vollwertige Programmiersprache kann JavaScript eine ganze Programm-Logik clientseitig abbilden. Das Ausrechnen von Warenkorb-Summen ist kein Problem, auch einfachere Prozesse wie Formular-Validierung wurden hier programmiert, bevor HTML5 das automatisierte. Anstatt die Eingaben auf den Server zu senden, dort validieren zu lassen und dem User ein Feedback an den Client zurückzuschicken, werden sie bereits clientseitig als richtig oder falsch erkannt.

Eine weitere große Aufgabe von JavaScript ist die dynamische Bereitstellung des Inhalts einer Website. So werden etwa Ziele von Links dynamisch in Abhängigkeit von User-Aktivitäten (etwa Sprachwahl) gesetzt oder manipulierende Eingriffe ins DOM vorgenommen.

Funktionalität serverseitig

Seit Erfindung von JavaScript Mitte der Neunziger wird versucht, die Sprache auch serverseitig einzusetzen. Kurz nach der browserseitigen Implementierung stellte Netscape die Software „Netscape Enterprise Server“ vor. [85] Es folgten zahlreiche weitere Implementierungen, die aber nur geringen Erfolg hatten. Erst 2009 wurde ein Framework unter dem Namen „Node.js“ von Ryan Dahl entwickelt, bei dem sich das ändern sollte. Es fand schnell Anklang und wird inzwischen von zahlreichen großen Webportalen eingesetzt. Node.js wird ausführlich im nächsten Kapitel behandelt.

3.2 Frameworks

3.2.1 Node.js

3.2.1.1 Einführung

Node.js ist ein Framework, mit dem sich eine serverseitige Architektur per JavaScript umzusetzen lässt. Es wurde erstmalig 2009 von Ryan Dahl veröffentlicht und seitdem kontinuierlich weiterentwickelt.

Node.js differenziert sich insofern von herkömmlichen Serverarchitekturen wie PHP + Apache, dass nicht für jede eintreffende Anfrage ein neuer Thread im Prozessor gestartet wird, sondern dass ein einziger Thread sich um alle Anfragen kümmert. Dabei wird jedoch nicht abgeblockt, wenn gerade eine Anfrage eintrifft, während eine andere noch abgearbeitet wird, da Node.js hier nach dem Non-Blocking I/O-Prinzip agiert.

Dieses Konzept erklärt Golo Roden wie folgt: "Das [Non-Blocking I/O, Anm. d. Verfassers] bedeutet, dass der Thread von Node.js eine eingehende Anfrage bis zur ersten Interaktion mit einer externen Ressource verarbeitet, dann diese Interaktion startet und die Anfrage so lange beiseitelegt, bis eine Antwort von der Ressource vorliegt.

In der Zwischenzeit kümmert er sich bereits um die nächste Anfrage und verarbeitet diese bis zu deren erster Interaktion mit einer externen Ressource und so weiter.

Sobald eine Antwort für eine beiseitegelegte Anfrage vorliegt, wird ein Callback ausgeführt und die beiseitegelegte Anfrage fortgesetzt – entweder bis zur nächsten Interaktion mit einer externen Ressource oder bis zu deren Abschluss." [86]

In Schemata sieht das wie folgt aus:

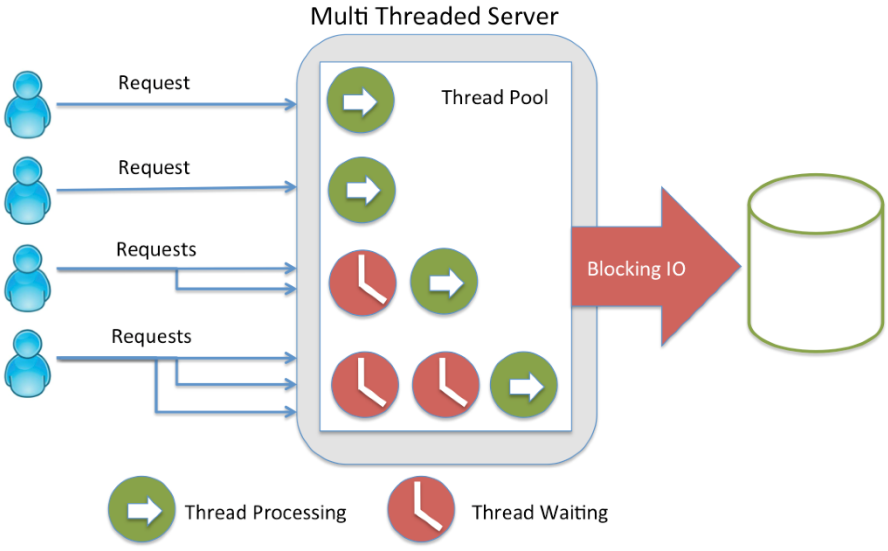

Abbildung 33: Herkömmlicher Server mit Blocking IO [87]

Hier sieht man einen herkömmlichen Webserver. Weitere Requests müssen warten, bis der aktuelle Datenbank-Zugriff absolviert wurde.

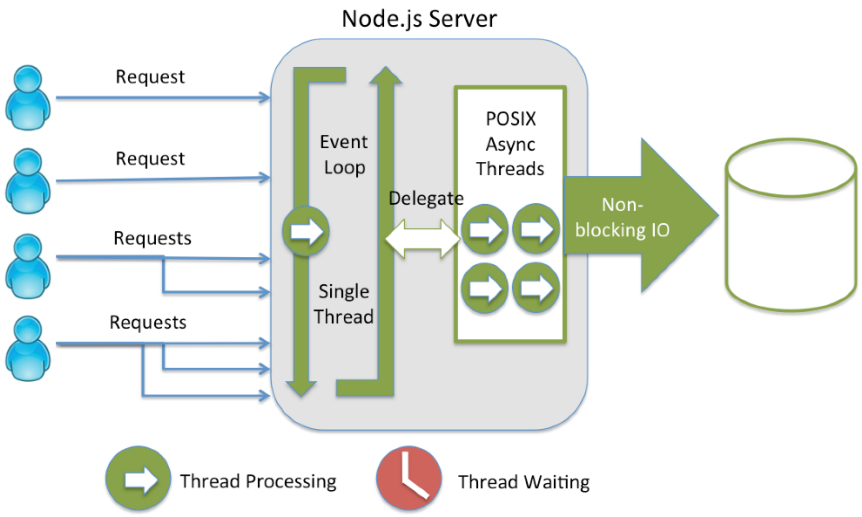

Abbildung 34: Node.js-Server mit Non-blocking IO [87]

Hier sieht man einen Node.js-Server. Weitere Requests werden direkt abgearbeitet. Der fertige Datenbank-Zugriff löst ein Callback aus und wird dann weiterbearbeitet. Dieses Prinzip ermöglicht es einem Node.js-Server, sehr viele Anfragen in sehr kurzer Zeit abarbeiten zu können.

Node.js ist derzeit für die drei gängigsten Betriebssysteme Windows, Mac OS X und Linux verfügbar und lässt sich komfortabel per Installer einrichten. Zusammen mit Node.js wird ein Paketmanager namens „npm“, ein Akronym für „Node Packet Manager“, installiert. Mittels `npm install` lassen sich über die Konsole zahlreiche zusätzliche Module für Node.js laden und installieren, die Webserver etwa um Internationalisierung oder Datenbank-Funktionalitäten erweitern. [88]

3.2.1.2 Beispiel eines Webservers

Ein einfacher Webserver, der auf Port 3000 lauscht und seinen Besuchern ein „Hallo Welt“ anzeigt, ist in fünf Zeilen geschrieben.

```
var http = require('http');
http.createServer(function (req, res) {
      res.writeHead(200, {'Content-Type': 'text/plain'});
      res.end('Hello World\n');
}).listen(3000);
```

Erst wird ein Modul namens „http“ geladen, welches Methoden zur Webserver-Funktionalität enthält, dann wird mit `createServer()` ein Server auf Port 3000 gestartet. Bei eingehenden Requests (`req`) wird eine Response (`res`) mit HTTP -Headerinformationen und einem Body abgeschickt.

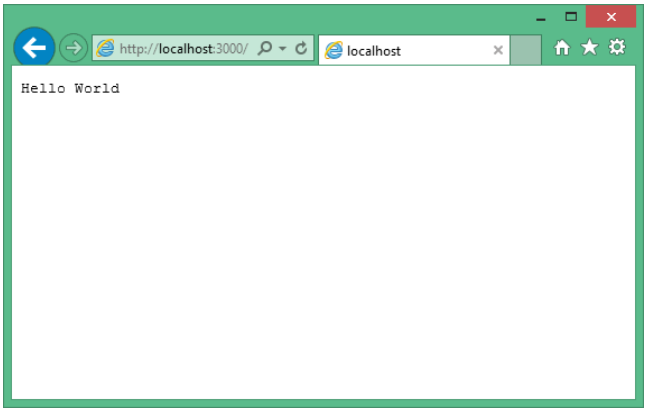

Abbildung 35: Response eines fünfzeiligen Webservers

Um die asynchrone Bearbeitung von Anfragen zu verdeutlichen werden hier folgende Code-Beispiele gegenüber gestellt:

Beispiel 1: Synchroner Code

```
var result = database.query('SELECT * FROM table');
console.log('query finished');
console.log('Next line');
```

Ausgabe:

```
query finished
Next line
```

Beispiel 2: Asynchroner Code

```
database.query('SELECT * FROM table', function() {
    console.log('query finished');
});
console.log('Next line');
```

Ausgabe:

```
Next line
query finished
```

Man sieht, dass der asynchrone Code die Konsolenausgabe aus Zeile 4 direkt abarbeitet und die Konsolenausgabe aus Zeile 2 erst weiterbearbeitet, wenn das Callback durch den Datenbank-Zugriff ausgelöst wurde.

So lassen sich auch komplexere Webserver implementieren, die mühelos und *non-blocking* mit sehr vielen HTTP -Requests umgehen können.

3.2.1.3 Vor- und Nachteile von Node.js

Node.js ist natürlich nicht die Universal-Lösung für alle künftigen Web-Applikationen. Es kommt auf die Anwendung an. Wenn Faktoren wie Skalierbarkeit, Performance oder Echtzeitfähigkeit gefragt sind, bietet sich Node.js an.

So eignet sich Node.js hervorragend für Web-Anwendungen, die oft auf externe Ressourcen zugreifen und diese weiterverarbeiten, um Requests zu beantworten.

Der folgende Benchmark zeigt, wie sich PHP und Apache gegen Node.js bei massiven Zugriffen auf eine per CMS generierte Webseite schlagen. [89] Die Inhalte der Website werden bei Aufruf aus einer Datenbank geladen. Für die PHP-Version wurde Wordpress [90] gewählt, für die Node-Version Ghost [91].

```
siege -v -c 100 -i -t 10M -f urls.txt -d 1
Transactions:                  27789 hits
Availability:                 100.00 %
Elapsed time:                 599.83 secs
Data transferred:             318.06 MB
Response time:                  1.65 secs
Transaction rate:              46.33 trans/sec
Throughput:                     0.53 MB/sec
Concurrency:                   76.65
Successful transactions:       27790
Failed transactions:               0
Longest transaction:            2.62
Shortest transaction:           0.51
```

Abbildung 36: Benchmark – Node.js mit Ghost [89]

```
siege -v -c 100 -i -t 10M -f urls.wp.txt -d 1
Transactions:                   4095 hits
Availability:                  99.95 %
Elapsed time:                 599.43 secs
Data transferred:              76.32 MB
Response time:                 13.92 secs
Transaction rate:               6.83 trans/sec
Throughput:                     0.13 MB/sec
Concurrency:                   95.07
Successful transactions:        4095
Failed transactions:               2
Longest transaction:           33.41
Shortest transaction:           0.90
```

Abbildung 37: Benchmark – PHP mit WordPress [89]

Es zeigt sich, dass über einen Zeitraum von knapp 10 Minuten eine Transaktions-Rate von 46,33 Transaktionen pro Sekunde bei Node.js und eine Rate von 6,83 Transaktionen pro Sekunde bei PHP erreicht wurde. Dies bescheinigt Node.js einen Geschwindigkeitsvorteil von über 678% bei hohem Kommunikationsaufkommen mit externen Ressourcen.

Neben diesem Vorteil eignet sich Node.js auch für Streaming und andere Echtzeit-Anwendungen, da der Entwickler über Node.js direkten Zugriff auf den Datenfluss hat, auch während der Übertragung. Mit der Verwendung von WebSockets hat der Server außerdem die Möglichkeit, auch nach dem Laden der Seite Daten an den Client zu pushen. [86]

Da das Datenformat JSON in Node.js große Verwendung und Unterstützung erfährt, eignet sich Node.js auch besonders für APIs, die JSON verarbeiten und zurückgeben, wie zum Beispiel einer JSON-basierten REST-Schnittstelle.

Nicht gut geeignet ist Node.js für Web-Anwendungen, die sehr rechenintensiv sind und gar nicht darauf ausgelegt sind, eine große Anzahl an In- und Outputs zu bearbeiten. Da Node.js nur in einem Prozess läuft, wird dieser so schnell blockiert. [92, 86] Seine Stärken in

Skalierbarkeit, Performance und Echtzeitfähigkeit werden hier nicht benötigt und auch nicht ausgespielt.

Auch einfache CRUD-Anwendungen, die Daten ohne weitere Verarbeitung einfach nur zwischen Datenbank und Server und Client hin- und herschieben, sowie einfache HTML-Webseiten lassen sich in Node.js wie in konventionellen Architekturen programmieren, bieten aber weder in dem einen noch in dem anderen Fall besondere Performance-Vorteile. Die serverseitigen Architekturen sind hier austauschbar.

3.2.1.4 socket.io

socket.io ist eines der vielen Module, die über den bereits erwähnten Paketmanager npm für Node.js-Applikationen installiert werden können. [93]

Socket.io soll den Umgang mit WebSockets in einer Node.js Umgebung vereinfachen. Hierzu gibt es diverse Module, etwa „websocket-node“ oder „engine.io“. Die Entwickler von socket.io haben sich allerdings darauf spezialisiert, automatische Fallbacks für Clients zu implementieren, die die WebSocket-Technologie nicht unterstützen. Werden die WebSockets nicht unterstützt, so wird versucht, die Verbindung über Adobe Flash Sockets aufzubauen. Klappt das auch nicht, so kommen AJAX Long Polling, AJAX Multipart Streaming, Forever Iframe oder JSONP Polling zum Einsatz. [93]

Während das Modul serverseitig mit `npm install socket.io` installiert wird, muss clientseitig eine .js-Datei mittels `<script src="…"></script>` eingebunden werden. Nach der serverseitigen Installation wird eine solche .js-Datei automatisch in ein virtuelles Verzeichnis kopiert. Der Pfad zur Datei muss dann einfach nur noch mittels

```
<script type='text/javascript' src='/socket.io/socket.io.js'></script>
```

referenziert werden. Zu diesem Zeitpunkt können Client und Server auf alle Modul-spezifischen Methoden zugreifen.

Mit `var io = require(socket.io)` wird das Modul serverseitig in einer Node.js-Applikation eingebunden, nun lässt sich mit `io.listen(server)` ein WebSocket-Server auf Basis eines HTTP -Servers starten. Sich neu verbindende Sockets werden mit `io.sockets.on('connection'){...}` registriert.

Clientseitig wird mit `var socket = io.connect();` die Verbindung zum WebSocket-Server aufgebaut. In diesem Moment wird das „connection“-Event ausgelöst, auf das der Server dann entsprechend reagieren kann.

Weitere manuelle Events werden einfach mit `emit()` und `on()` sowohl server- als auch clientseitig ausgelöst und registriert. So wird beispielsweise ein Ereignis serverseitig über folgenden Code ausgelöst:

```
io.socket.emit('newStockValue', {AAPL : 531.70, MSFT: 35.89});
```

und mit folgendem Code clientseitig weiterverarbeitet:

```
socket.on('newStockValue', function (data){
      console.log('New Apple stock value: ' + data.AAPL);
      console.log('New Microsoft stock value: ' + data.MSFT);
});
```

An diesem Beispiel sieht man auch, dass socket.io eine Verarbeitung von JSON-Objekten implementiert hat. Der Entwickler muss sich nur noch um Datenstruktur und -inhalt

kümmern, das Modul übernimmt das de- und encoden von JSON-Strings auf Client- und Server-Seite.

Weitere Module, wie etwa das inzwischen eingestellte „NowJS“, basieren auf socket.io, nutzen dessen Fallback-Funktionalitäten und erweitern es gleichzeitig um weitere Socket-Features. So ließen sich in NowJS etwa serverseitig Funktionen schreiben, die dem Client automatisch zum Aufrufen zur Verfügung gestellt wurden.

"Die NowJS zugrunde liegende Idee ist, einen Namensraum zur Verfügung zu stellen, der gleichermaßen von Client und Server verwendet und automatisch zwischen beiden synchronisiert wird: Funktionsaufrufe und Zuweisungen, die am Client ausgeführt werden, werden automatisch auch am Server ausgeführt und umgekehrt." [86]

3.2.1.5 Praxisbeispiele

Es gibt inzwischen zahlreiche Webauftritte, die Node.js im Backend verwenden. Hier seien drei bekannte Beispiele genannt:

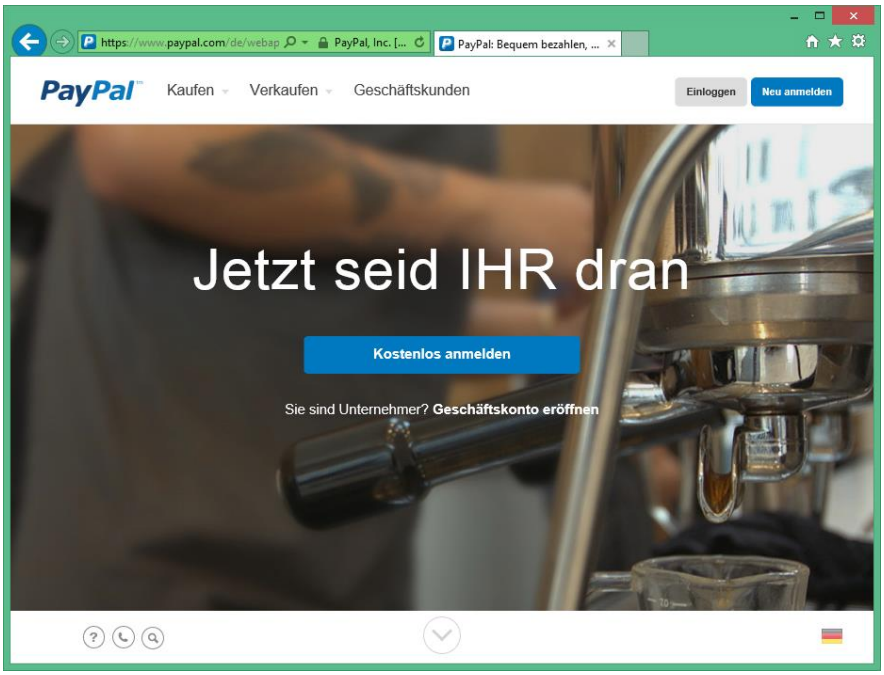

Abbildung 38: Beispiel für Node.js: paypal.com [94]

PayPal annoncierte auf seinem Engineering-Blog am 22.11.2013, dass seine Web-Applikationen im Backend von Java auf JavaScript mit Node.js umgestellt wurden. Begründet wurde dies unter anderem mit einer höheren Performance und 33% weniger Zeilen Code. [95]

Das Team von „Express“, dem bereits in Kapitel 2.1.4 erwähnten Node.js-Framework für Web-Applikationen, wirbt auf seiner Seite unter anderem mit einem weiteren bekannten Anwender:

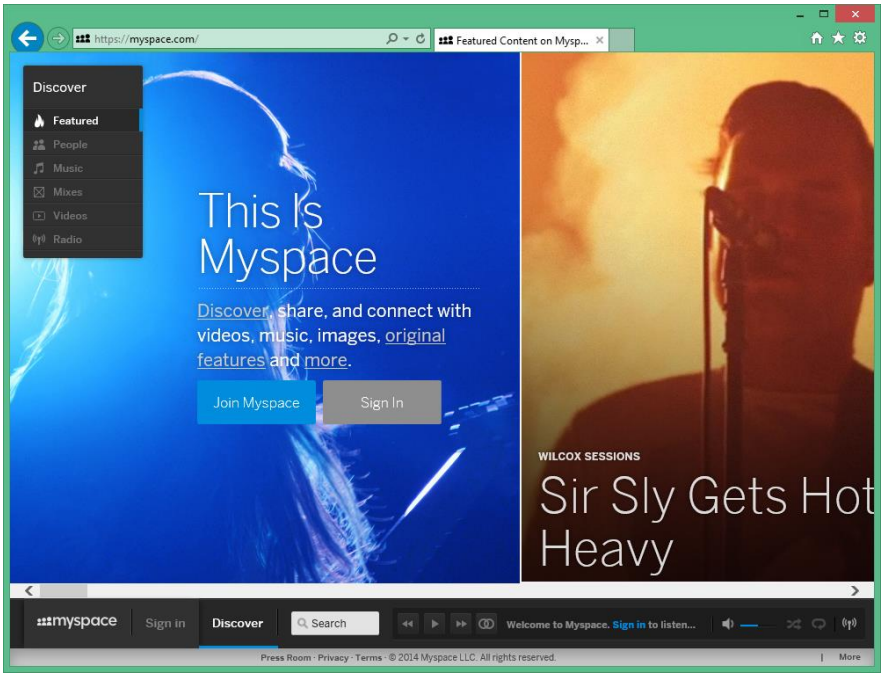

Abbildung 39: Beispiel für Node.js: myspace.com [96]

Neben „MySpace“ sind dort populäre Vertreter wie „LearnBoost“ und „Klout“ aufgelistet.

Einen etwas anderen Ansatz verfolgen LinkedIn und Walmart. Sie nutzen Node.js, um die mobile Version ihrer Webseite als Web-App zur Verfügung zu stellen. [97, 98] Hier folgend exemplarisch ein Screenshot von linkedin.com mit dem Smartphone:

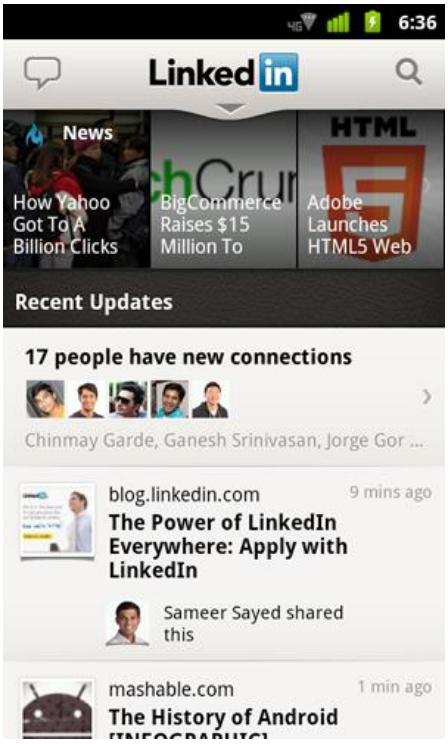

Abbildung 40: Beispiel für Node.js: linkedin.com (mobile Version) [99]

3.2.1.6 Ausblick

Node.js ist derzeit in der Version 0.10.26 (Stand 23.04.14) verfügbar. Das Interesse, wie folgende Google-Trends-Analyse zeigt, steigt seit Erstveröffentlichung stetig.

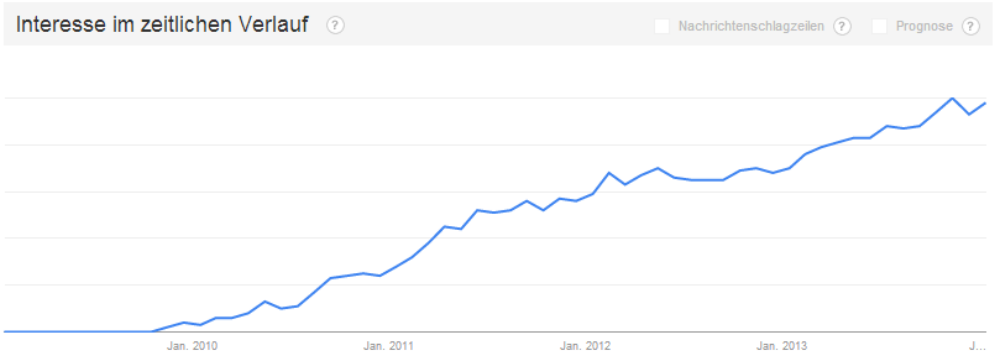

Abbildung 41: Google Trends mit dem Stichwort "node.js" [100]

Der offizielle Blog des Node.js-Teams enthält neben den Changelogs der Node.js-Updates auch hin und wieder Beiträge zur Zukunft des Frameworks. [101]

So schreibt Isaac Schlueter im Januar dieses Jahres, dass er sich mehr auf die Entwicklung des Paketmanagers npm konzentrieren wird und annonciert einen neuen Projektleiter. [102]

In einem weiteren Blogeintrag am Tag darauf berichtet der neue Chef, Timothy J. Fontaine, über die kommenden Neuerungen in Node.js. [103] So wird das Kernziel sein, einen kleinen Node-Kern zu entwickeln, auf dessen Basis Programmierer unkompliziert arbeiten und neue Funktionalitäten implementieren können. Es wird ein kleines, einfaches und stabiles Repertoire an APIs geben, das es der Community leicht macht, ihre Ideen umzusetzen. Außerdem betont er die Abwärtskompatibilität von künftigen Node-Versionen. So soll es auch künftig möglich sein, mit bereits programmierten APIs in Node arbeiten zu können.

Performance-Verbesserungen und Bugfixing seien immer Teil der Entwicklung, daneben muss auf Änderungen der JavaScript Engine V8 von Google Rücksicht genommen werden.

Abschließend betont er sehr stark den Community-Gedanken und dass künftige Releases davon abhängig sein werden, wie gut diese Community das Projekt durch Tests, Eigenentwicklungen, Doku-Beiträge und Tutorials unterstützt.

Da Node.js breite Unterstützung erfährt und durch den mächtigen Paketmanager npm ergänzt wird, sollte es weiteren Anklang finden. Große Firmen wie PayPal haben ihre Systeme bereits umgestellt, was vielen Mut zusprechen wird, außerdem zeigen gerade auch diese Firmen, dass Node.js kein unsicheres Start-Up mit Kinderkrankheiten mehr ist.

3.2.2 Serverseitige Datenbank

3.2.2.1 NoSQL-Datenbanken

Es gibt etliche Konzepte und Produkte, um strukturierte Daten persistent auf einem Datenträger zu speichern. Darunter sind relationale, objektorientierte und nicht-relationale Datenbanken. Am weitesten verbreitet sind relationale Datenbanken. [104]

Natürlich kann Node.js mit klassischen relationalen Datenbanken wie MySQL arbeiten. Aber es lohnt sich, die Produkte der NoSQL-Bewegung in Betracht zu ziehen. NoSQL-Datenbanken verfolgen einen nicht-relationalen Ansatz und sind häufig mit JavaScript-

fähigen Schnittstellen ausgestattet, die über JSON kommunizieren. Damit eignen sie sich optimal für einen Einsatz mit Node.js. Zudem sind NoSQL-Datenbanken oft einfacher zu verwalten und zu verwenden als relationale Datenbanken. [86]

Bei NoSQL-Datenbanken wird im Wesentlichen zwischen vier Typen unterschieden:

Key-Value Datenbanken

Key-Value-Datenbanken bestehen aus zweispaltigen Tabellen. Zu einem eindeutigen Schlüsselattribut wird jeweils ein Wert gespeichert. Der Wert ist in der Regel eine Zeichenkette oder ein binäres Datenfeld, es können beliebige, unstrukturierte Daten gespeichert werden.

Key	Value
P1_Name	John Doe
P1_Alter	32
P1_Geschlecht	männlich

Dokumentenorientierte Datenbanken

Dokumentenorientierte Datenbanken speichern strukturierte Daten in schemalosen einspaltigen Tabellen, also praktisch Listen. Ein Datensatz nennt sich Dokument. Dokumente sind üblicherweise Objekte, oft in JSON-Notation verfasst. Diese standardisierte Struktur ermöglicht komplexe Abfragen. Da die Auflistung der Dokumente schemalos ist, können Dokumente mit unterschiedlichen Gliederungen vermischt werden.

```
{
	"id" : 1,
	"typ" : "Person",
	"name" : "Johne Doe",
	"alter" : 32,
	"geschlecht" : "männlich",
	"kleidung" : {
		"oberkoerper" : "t-shirt",
		"unterkörper" : "hose"
	},
	"hobbies": ["tennis", "geige"]
}
```

```
{
	"id" : 2,
	"typ" : "Auto",
	"marke" : "Volkswagen",
	"modell" : "Polo",
	"farbe" : "schwarz"
}
```

Dokumentenorientierte NoSQL-Datenbanken bieten den weiteren Vorteil, dass ebenjene Dokumente viel feiner adressiert werden können als SQL-Einträge. Während dort nur komplette Felder angesprochen werden können, kann hier jedes Attribut jedes Subdokuments gezielt adressiert werden.

Spaltenorientierte Datenbanken

Spaltenorientierte Datenbanken, oder auch „Wide-Column“-Datenbanken, speichern die Daten ähnlich wie relationale Datenbanken in Tabellen. Allerdings wird ein Datensatz auf den Datenträger nicht horizontal in Reihen gespeichert, sondern vertikal in Spalten.

Gegeben seien folgende beiden Datensätze in der Tabelle auf der nächsten Seite:

ID	Vorname	Nachname	Alter
1	Max	Muster	32
2	Liesl	Müller	28

Klassisch relational würde das wie folgend physisch auf einem Datenträger abgespeichert:

```
1,Max,Muster,32;2,Liesl,Müller,28;
```

Spaltenorientiert werden die Datensätze folgend gespeichert:

```
1,2;Max,Liesl;Muster,Müller;32,28;
```

Diese physische Sortierung sorgt für bessere Performance bei Analyse und Aggregation umfangreicher Datenmengen. [86]

Graphen-Datenbanken

Graphen-Datenbanken sind im Web verhältnismäßig selten vertreten. Das liegt daran, dass sie für spezielle Anwendungsfälle entwickelt wurden: Das Ziel von Graphen-Datenbanken ist nicht primär das Speichern der eigentlichen Daten sondern das Abspeichern der Beziehungen zwischen diesen. [86]

Graphen-Datenbanken lassen sich theoretisch in klassischen relationalen Datenbanken abbilden, bieten aber für komplexe Beziehungen einfachere Abfragemöglichkeiten und haben Algorithmen implementiert, die etwa die kürzeste Verbindung zwischen zwei Knoten suchen. So eignen sich Graphen-Datenbanken besonders für soziale Netze. Ein Anwendungsbeispiel wären hier die Benutzer als Knoten. Besucht man das Profil eines Nutzers, den man noch nicht kennt, so sieht man über die „Kanten“, über welche Freundeskette man sich kennen könnte. [105]

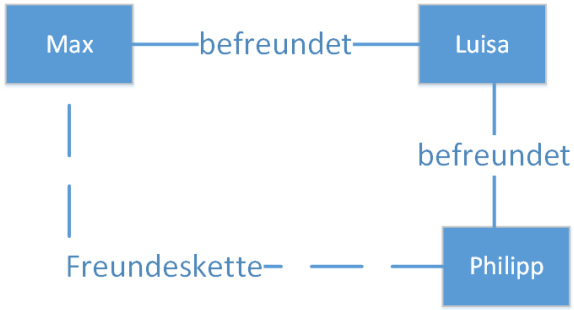

Abbildung 42: Beziehungen in einer Graphen-Datenbank – Max kennt Philipp über Luisa

3.2.2.2 Produkte und Beschreibung

Das Framework, das im Rahmen dieser Thesis entsteht, soll den Zustand einer Anwendung serverseitig als JSON-Objekt in einer Datenbank speichern. Darum wird eine dokumentenorientierte NoSQL-Datenbank genutzt.

Die populärsten Vertreter dieser Kategorie sind MongoDB [106] und CouchDB [107]. [108]

MongoDB

MongoDB wurde 2009 von der amerikanischen Entwicklerschmiede 10gen veröffentlicht. Heute nennt sich die Firma MongoDB, Inc. MongoDB ist aktuell die verbreitetste NoSQL-

Datenbank im Internet. Das Produkt wird von namhaften Internet-Auftritten verwendet, wie etwa GitHub, Foursquare und Bit.ly. [109]

Serverseitig lässt sich MongoDB unter allen gängigen Betriebssystemen installieren. Die Datenbank wurde in C++ geschrieben und verwaltet ihre Dokumente in BSON, Binary JSON. Abfragen werden über proprietäre Sprach-Konstrukte in JavaScript auf JSON-Basis getätigt.

Das Äquivalent zur Tabelle in relationalen DBMS sind hier „Collections“. Eine Collection enthält eines oder mehrere Dokumente, die mit JSON strukturiert, aber schemalos sind.

Eine Stärke von MongoDB ist das simple API, das teilweise Ähnlichkeiten zu SQL hat.

So lassen sich in eine Collection „Personen“ mittels folgender Befehle ein paar Beispiel-Dokumente einfügen, die dem SQL Befehl `INSERT INTO` entsprechen:

```
db.Personen.save({name: 'Andreas', alter: 25});
db.Personen.save({name: 'Daniel', alter: 32});
```

Hier werden zwei JSON-Objekte erstellt und als Dokumente in der Collection `Personen` gespeichert. Neben `name` und `alter` erhalten die Objekte jeweils noch ein drittes, automatisch generiertes Attribut namens `_id`. Es ist ein eindeutiger Bezeichner.

Eine Abfrage, die in SQL dem `SELECT FROM WHERE`-Konstrukt entspricht, sieht in MongoDB folgendermaßen aus:

```
db.Personen.find({alter: 25});
```

In diesem Beispiel werden alle Personen-Dokumente ausgegeben, die ein Attribut `alter` besitzen, welches den Wert `25` hat. Die Ausgabe könnte so aussehen:

```
{ '_id' : ObjectId('535e17b41cdca'), 'name' : 'Andreas', 'alter' : 25 }
```

Auch `UPDATE, DELETE, …` finden in ähnlicher Manier wie in SQL statt. Weitere Vorteile von MongoDB sind integriertes Sharding (gleichmäßiges Verteilen von Last und Daten auf mehrere Server), sehr gute Skalierung, diverse Index-Formate und neben einer großen Community auch kommerzieller Support.

Dem Konzept des CAP-Theorems folgend steht MongoDB für Konsistenz und Partitionstoleranz, dafür wird auf stetige Verfügbarkeit verzichtet. [110]

CouchDB

CouchDB ist der größte Rivale von MongoDB auf dem Feld der dokumentorientierten NoSQL-Datenbanken, kommt aber nicht ganz an dessen Popularität heran. CouchDB gibt es bereits seit 2005, Entwickler ist die Apache Software Foundation. CouchDB läuft ebenfalls auf allen gängigen Server-Betriebssystemen und wurde in der Programmiersprache Erlang implementiert.

Dokumente bestehen dort inzwischen aus JSON-Objekten, während sie früher noch in XML-Notation geschrieben waren. Abfragen werden nicht mit JavaScript-Funktionen realisiert, sondern erfolgen mittels HTTP-REST-Requests. Dokumente werden nicht in Collections abgelegt, wie in MongoDB. Die Hierarchie über einer Gruppe von Dokumenten nennt sich stattdessen schlicht „Database“, zu Deutsch Datenbank. [111]

Äquivalent zu den Beispielen oben würde man mit dem Kommandozeilentool `curl`, mit dem sich individuell konfigurierte HTTP -Requests gestalten lassen, folgende Befehle abschicken, um einer Database `Personen` ein Dokument hinzuzufügen.

```
curl -X PUT http://127.0.0.1:5984/Personen/6e1295ed6c -d
'{"name":"Andreas","alter":25}'
```

Die zentrale Navigation erfolgt hier REST-typisch über die URL. Es wird mit `curl` ein PUT-Request an die IP und den Standard-Port der CouchDB-Anwendung gesendet. Hinter IP und Port folgt der Name der Datenbank, danach wird die eindeutige ID angegeben, die jedes Dokument benötigt. Als Inhalt des PUT-Requests wird ein JSON-Objekt mit den Dokument-Inhalten gesendet.

Die eindeutige ID ist eine UUID, die sich vorher mittels `curl -X GET http://127.0.0.1:5984/_uuids` generieren lässt.

Abfragen nach dem SQL-Schema `SELECT FROM WHERE` gestalten sich etwas aufwändiger als bei MongoDB. Sie orientieren sich an dem MapReduce-Konzept, dass Google für große Datenbanken entwickelt hat. So könnte eine Map-Funktion aussehen, der man Personen-Dokumente übergibt.

```
function(doc) {
  if(doc.alter) {
    emit(doc.alter, null);
  }
}
```

Sie überprüft, ob das Dokument ein Attribut namens `alter` hat und erstellt eine View mittels `emit()`.

Danach lässt sich mit dem HTTP -Request

```
curl -X GET http://127.0.0.1:5984/Personen/_view/alter?key=25
```

filtern, in welchen Dokumenten das Attribut `alter` den Wert 25 hat. Die Ausgabe würde in diesem Fall etwa so aussehen:

```
{'_id':'6e1295ed6c','_rev':'1-2902191555','name':'Andreas','alter':25}
```

Sharding ist in CouchDB nicht integriert und muss über Erweiterungen installiert werden. In der CouchDB-Installation ist eine grafische JavaScript-Applikation integriert, über die sich die Datenbank mit einem übersichtlichen UI administrieren lässt. Da CouchDB einen eigenen Webserver mitbringt, der Daten an den Browser senden kann, kann auf serverseitige Programmierung mit Sprachen wie PHP verzichtet werden.

Im Sinne des CAP-Theorems legt CouchDB Wert auf Partitionstoleranz und Verfügbarkeit, weswegen die Konsistenz der Daten zu jedem Zeitpunkt nicht zwingend gegeben ist. [110]

3.2.2.3 Entscheidungsfindung

Die Datenbanken ähneln sich in einigen Punkten. So ist die Speicherung von Daten als JSON-Objekte wünschenswert für dieses Framework. Dies erfüllen beide. Außerdem arbeiten beide Datenbanken sehr schnell und skalieren performant.

Allerdings fallen im Vergleich auch gewichtige Unterschiede auf. Das REST-API von CouchDB wirkt nicht so komfortabel wie die einfachen `save(), find()` oder `remove()`-Funktionen von MongoDB, außerdem müssen dort nicht zusätzlich MapReduce-Funktionen bedient werden. Generell lehnt sich das Bedienkonzept von MongoDB an SQL an, was vielen erfahrenen Webentwicklern zu Gute kommen mag.

Im Sinne des CAP-Theorems ist es viel wichtiger für ein Framework für kollaborative Web-Anwendungen, dass permanent Konsistenz gewährleistet ist und alle Teilnehmer auf demselben Stand sind. Längere Reaktionszeiten im sind dagegen tolerierbar.

Konzepte wie Sharding sind in MongoDB bereits nativ implementiert, was das Entwickeln vereinfacht. In CouchDB müssen diese nachinstalliert werden. Außerdem sind zu viele Abhängigkeiten zu externen Frameworks zu vermeiden, um bei Updates oder Supportende den Wartungs- und Anpassungsaufwand nicht zu erhöhen.

Die große Beliebtheit und die weltweite Community sowie der Einsatz bei großen Internet-Portalen wie GitHub, die sicher selbst ausführlich evaluiert haben, sind außerdem ein Punkt für MongoDB.

Aus den genannten Gründen fällt die Entscheidung für dieses Framework auf MongoDB.

3.2.3 Canvas-Bibliotheken

3.2.3.1 Allgemein

Wie man in Kapitel 3.1.1.2 feststellen kann, ist die Programmierung des Canvas recht umständlich. Für jedes Canvas muss erst ein Kontext initiiert werden, außerdem muss erst eine globale Farbe für kommende Zeichnungen festgelegt werden, die auch so lange aktiv bleiben wird, bis man sie ändert. An Standard-Formen gibt es im Repertoire nur Rechtecke. Dreiecke, Kreise und andere müssen aufwändig mit Pfaden gemalt werden. Es wird komplett prozedural gearbeitet.

```
var canvas = document.getElementById('canvas');
var context = canvas.getContext('2d');
context.fillStyle='#FF0000';
context.fillRect(30,40,50,50);
```

Um dem Canvas-Entwickler hier etwas Arbeit abzunehmen, gibt es eine Vielzahl an JavaScript-Bibliotheken, die diese Schritte zusammenfassen und eine einfache Schnittstelle nach außen hin anbieten. Je nach Bibliothek wird diese Schnittstelle mit objektorientierter Programmierung angesprochen, was ebenfalls Vereinfachung bedeuten kann. So werden die obigen vier Zeilen mit einer Bibliothek wie etwa fabric.js [112] zu folgenden drei zusammengefasst:

```
var canvas = new fabric.Canvas('canvas');
var rect = new fabric.Rect({left:30,top:40,width:50,height:50,fill:'red'});
canvas.add(rect);
```

Das Rechteck wird hier nicht über eine Funktion gemalt, wie in dem nativen Canvas-API, stattdessen wird mit `new Rect()` ein Konstruktor aufgerufen und ein eigenes Objekt erstellt. Mit dem einfachen Aufruf von `add()` werden im Hintergrund die bekannten Funktionen des nativen Canvas-API angesprochen.
Der objektorientierte Ansatz ermöglicht es uns, unzählige Form-Objekte mit völlig verschiedenen, unabhängigen Eigenschaften zu erstellen. Diese Eigenschaften haben eindeutige Bezeichner, was viele Blicke in die Dokumentation erspart.

Zusätzlich liefern uns Bibliotheken viele weitere Formen. Sie werden mit den jeweiligen Konstruktoren erstellt und im Hintergrund erledigt die Bibliothek die Pfad-Zeichnung. So wartet fabric.js etwa mit `fabric.Circle`, `fabric.Ellipse` oder `fabric.Triangle` auf.

Alle Form-Objekte können zur Laufzeit ihren Wert ändern, womit auf sehr einfache Weise Animationen realisiert werden können. Auch Transformationen, Text-Gestaltung und Ebenen-Aufbau werden durch Bibliotheken vereinfacht. Während mit dem nativen API vor jeder Änderung das Canvas „weiß übermalt“ werden muss, da sonst Original und Änderung gleichzeitig erscheinen, erledigen das die Bibliotheken automatisch. Komplexe Event-Handlings, etwa Tastendrucke, Mausklicke oder Wischgesten werden registriert und je nach Wunsch des Programmierers weiterverarbeitet.
Einige Bibliotheken haben sogar den Export der Canvas-Zeichenfläche in SVG-, JPEG- oder PNG-Grafiken implementiert.

3.2.3.2 Auswahl der Canvas-Bibliothek

Im Rahmen dieser Thesis geht es natürlich nicht darum, das Rad neu zu erfinden. So war es eine Aufgaben, herauszufinden, welche Canvas-JS-Bibliotheken aktuell entwickelt und gepflegt werden und wie diese den Anforderungen des Frameworks entsprechen.

Die Anforderungen an das Framework, das im Rahmen dieser Thesis erstellt wird, wurden bereits in groben Zügen in der Einleitung genannt. Daraus resultieren als wünschenswerte Kriterien für eine Canvas-Bibliothek:

Ebenen-Unterstützung im Canvas

Eine Ebenen-Unterstützung ist sinnvoll, da so Objekte besser übereinander platziert werden können, außerdem lassen sich so lokal fixierte und dynamische Elemente besser trennen.

Die Möglichkeit eines Text-Inputs im Canvas

Text-Input wird bei der konkreten Anwendung, dem Stundenplan-Gestalter, benötigt, außerdem ist davon auszugehen, dass dieses Feature generell bei kollaborativen Web-Applikationen benötigt wird.

Support für Touch-Geräte

Der Support für Touch-Geräte ist dem Faktum geschuldet, dass ein Canvas, anders als Flash, auf allen aktuellen Browsern, eben auch jenen aus dem Hause Apple, unterstützt wird. Diese Tatsache soll genutzt und implementiert werden.

Eine serverseitige Unterstützung durch Node.js

Da serverseitig Node.js verwendet wird, ist es wünschenswert, dass die Datenmodelle, die vom Client kommen, auch virtuell verarbeitet werden können, etwa zum Export eines Screenshots. Hier bietet sich eine Node.js-Implementierung der Bibliothek an.

Community

Als subjektiv bewertetes Kriterium gibt es außerdem die „Community“. Da eine Bibliothek ab einem gewissen Bekanntheits- und Verwendungsgrad eine Vielzahl von tatkräftigen ehrenamtlichen Helfern mit sich bringt, die sich viel mit der Bibliothek beschäftigt haben und Support in Foren anbieten, spart man sich gelegentlich stundenlanges Debugging und Frust.

Dokumentation

Ebenfalls subjektiv wird der Status der Dokumentation bewertet. Kriterien sind Vollständigkeit und Aktualität. Außerdem benötigen derart große Dokumente eine nachvollziehbare Struktur.

Die Anforderungen an das Framework werden in aller Ausführlichkeit in Kapitel 4 behandelt. Die Analyse des gegenwärtigen Angebots, versehen mit den Angaben zu den einzelnen Kriterien, wird in der folgenden Tabelle dargestellt.

Nr.	Name	Layer	Text-Input	Touch-Support	Node.js	Comm.	Doku
1	FabricJS[112]	√	√	√	√	+	+
2	KineticJS[113]	√		√		++	+
3	EaselJS[114]	√		√	√	++	++
4	PaperJS[115]	√			√	0	+
5	Zebkit[116]	√	√			-	++
6	oCanvas[117]			√	√	--	++
7	HTML Canvas Lib[118]	√		√		--	0
8	JcanvaScript[119]					+	+
9	Bhive[120]					--	--

Der Favorit ist hier klar ersichtlich: Fabric.js. Er bringt alle benötigen Funktionen mit sich.

Neben den gezeigten Vorzügen bietet er zahlreiche weitere Features, etwa, dass alle Elemente im Canvas mit Handlern versehen werden können, mit denen man das Element bewegen, skalieren und drehen kann, was das Verarbeiten von User Events auf der Zeichenfläche deutlich vereinfacht. Damit erfüllt die Bibliothek eine Kernfunktionalität des Frameworks.

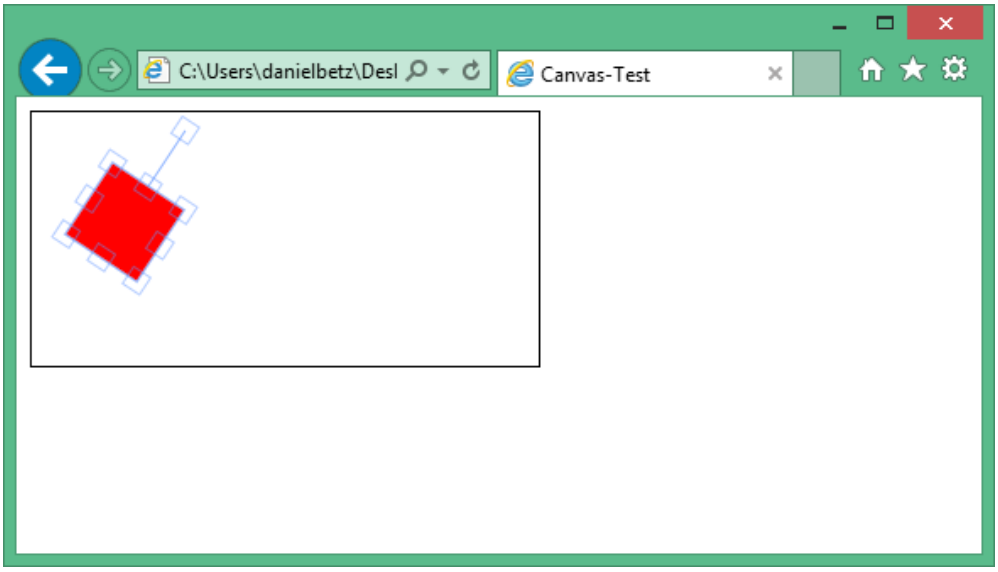

Abbildung 43: Fabric.js-Canvas mit Transformations-Optionen

Außerdem ist ein Export in JPEG-, PNG- und SVG-Dateien implementiert. Verläufe, Filter, eine umfangreiche Textbearbeitung, Gruppierung von Elementen, JSON-Umwandlung, „Frei-Zeichnen“-Funktionen sowie ein umfangreicher Node.js-Support für die serverseitige Erstellung von Fabric-Canvas runden das Bild ab.

Neben der Bibliothek selbst bieten die Entwickler auf ihrer Website zahlreiche Benchmarks und Tests an, teilweise sogar im Live-Vergleich mit anderen Bibliotheken. Auch Tutorials werden von den Entwicklern bereitgestellt.
Viele Demos inspirieren den Anwender, alle werden mit Quellcode zur Verfügung gestellt.

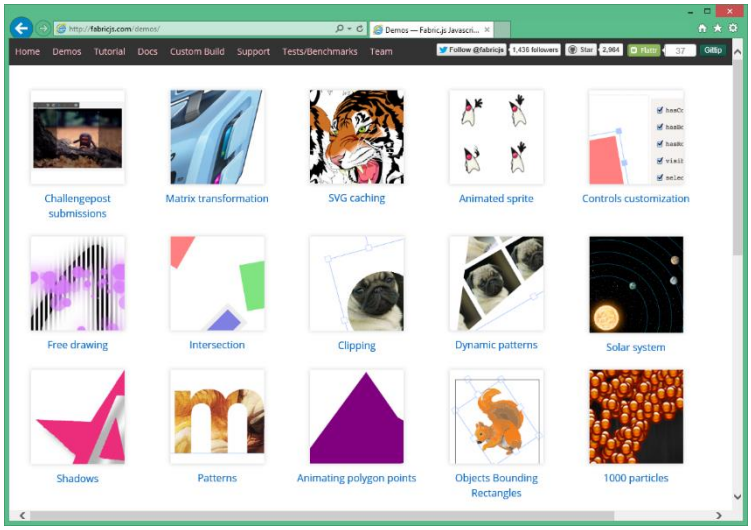

Abbildung 44: Demo-Sammlung von fabric.js [121]

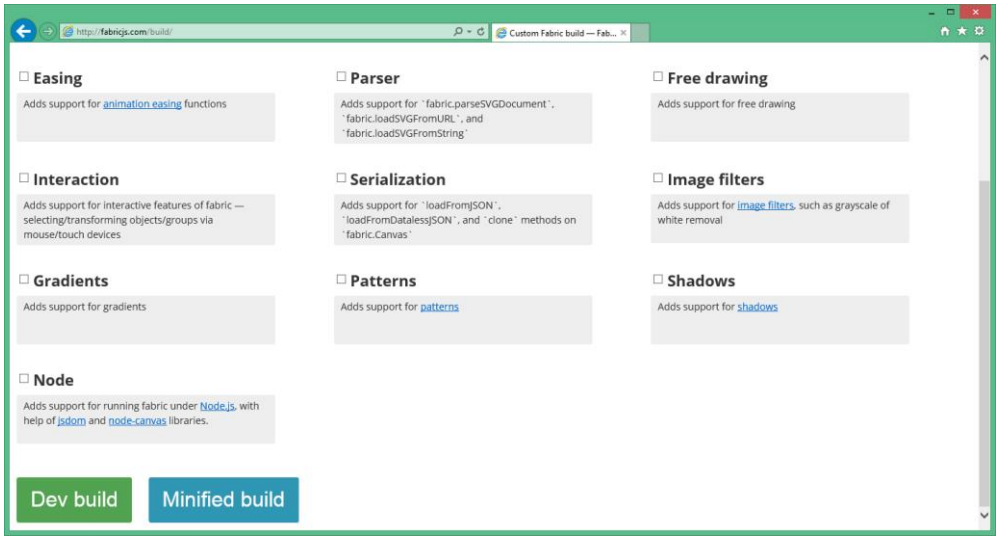

Abbildung 45: Build-Former für Fabric.js Bibliothek [122]

Mit dem interaktiven Build-Former lässt sich eine individuell kombinierte Fabric.js-JavaScript-Bibliothek in die eigene Website einbinden, die nur die Features beinhaltet, die man selbst benötigt. Damit werden die .js-Dateien entsprechend klein, was dem Ladeprozess der Webseite zu Gute kommt.

Der integrierte Touch-Support deckt den Bedarf des Frameworks komplett ab. Formen wie das Rechteck können mit einem Finger bewegt werden. Das Vergrößern von aktivierten Formen lässt sich mit dem Auseinanderziehen zweier Finger realisieren, genauso wie das Verkleinern durch Zusammenziehen. Drehung lassen sich ebenfalls mit zwei Fingern durchführen.

Tippt man auf ein editierbares Textfeld, so fährt automatisch das mobile Keyboard hoch und man kann darin schreiben.

All diese Argumente begünstigen die Entscheidung hin zu Fabric.js.

4 Anforderungen und Abgrenzungskriterien des Frameworks

In diesem Kapitel wird beschrieben, welche Funktionalitäten des Frameworks seitens der Firma ausdrücklich gewünscht wurden und welche nicht gefordert waren.

In einer Kurzübersicht gelten folgende Anforderungen:

- Das Framework dient für Anwendungen im Browser und soll eine breite Palette an Betriebssystemen unterstützen. Dies impliziert auch Support von mobilen Endgeräten.
- Die Oberfläche basiert auf einzelnen Komponenten, etwa Texteingabefeldern oder Grafiken, die sich per Drag-and-drop positionieren lassen und interaktiv transformiert werden können.
- Das Framework zielt auf Anwendungen ab, die im Team genutzt werden. So wird eine Oberflächensynchronisation in Echtzeit über alle verbundenen Clients realisiert.
- Serverseitig soll Node.js eingesetzt werden, da sich diese Architektur sehr gut zum schnellen Austausch von Zustandsänderungen eignet.
- Der Zustand einer Anwendung wird während der Benutzung serverseitig gespeichert und lässt sich jederzeit wieder zur Weiterarbeit laden.
- Die Gestaltung der Oberfläche findet im HTML5-Canvas statt. Für das Rendern des GUI soll eine JavaScript-Bibliothek zum Einsatz kommen, mit der auf dem Canvas objektorientiert gearbeitet werden kann.
- Das Arbeitsergebnis soll jederzeit als Pixelgrafik exportiert werden können.

In den folgenden Unterkapiteln werden die Anforderungen an das Framework detailliert beschrieben. Sie sind thematisch nach Technik, Oberfläche, Anwendung und Implementierung sortiert.

4.1 Oberfläche

Komponentenbasiert

Um den Entwicklern beim Einsatz des Frameworks die Möglichkeit zu geben, ihre Applikationen möglichst individuell zu konzipieren und zu gestalten, wird die Oberfläche in Komponenten organisiert, die ihrerseits wieder Unterkomponenten, wie UI-Elemente, besitzen. Die Daten sollen in strukturierten Objekten nach JSON-Syntax beschrieben werden. Unterschieden wird zwischen statischen, unbeweglichen Komponenten, etwa Reglern und Buttons, deren Position und Funktion einmalig festgelegt wird, und dynamischen Komponenten, die zur Laufzeit generiert und verändert werden können, wie etwa Grafiken und Textfelder.

Drag-and-drop

Die dynamischen Elemente sollen per Drag-and-drop durch den User ihre Position verändern können.

Texteingabe

Explizit gefordert ist bei den dynamischen Komponenten neben grafischen Elementen auch die Möglichkeit, Texteingabefelder zu benutzen.

Interaktives Transformieren

Die grafischen Elemente und Textfelder sollen neben ihrer Position auch ihre Größe und Drehung durch den User zur Laufzeit ändern können.

4.2 Anwendung

Mehrbenutzerfähigkeit

Die Anwendungen, die mit diesem Framework entstehen, sollen von mehreren Clients gleichzeitig genutzt werden können. So kann gemeinsam in einer Anwendung gearbeitet werden. Realisiert wird dies über WebSockets.

Screenshots

Es soll möglich sein, das Arbeitsergebnis jederzeit als Pixelgrafik zu exportieren.

4.3 Technik

Browser-Anwendung

Das Framework soll zur Erstellung von Applikationen im Browser dienen. So erfahren sie unter den Betriebssystemen eine breite Unterstützung. Außerdem können sie zentral gewartet und weiterentwickelt werden und müssen nicht für jedes System in einer eigenen Sprache implementiert werden.

Keine proprietären Plug-Ins / iOS-Support

Das Framework soll Technologien nutzen, die aktuelle Browser auf möglichst allen Endgeräten unterstützen. Das schließt zum Beispiel die Verwendung von Adobe Flash aus, welches auf mobilen Endgeräten der Firma Apple mit ihrem Betriebssystem iOS nicht mehr unterstützt wird.

Alle Frameworks, Bibliotheken und Techniken, die im Framework verwendet werden, sollen kostenlos und quelloffen vorliegen. Die Lizenz soll sowohl eine private als auch eine kommerzielle Nutzung erlauben.

User-Interface im Canvas

Das mit HTML5 eingeführte Canvas soll Grundlage der Benutzer-Interaktion mit einer Applikation sein. Nachteilig ist, dass im Canvas keine HTML-Standardkomponenten zur Verfügung stehen, dem gegenüber steht, dass Positionierung und Gestaltung der Oberflächen-Komponenten sehr flexibel definierbar sind. Das Canvas ist optisch wie funktional frei gestaltbar. Weiterhin fügt sich das Canvas mit seinem JavaScript-API gut in den JavaScript-Stack ein.

Objektorientierte Canvas-Bibliothek

Eine Canvas-Bibliothek ersetzt viel manuelle Code-Arbeit an dem nativen Canvas-API. Objektorientierung bietet sich für die Komponenten an, da sich so Eigenschaften, etwa Inhalt, Farbe, Position, kompakt strukturieren lassen. Schachtelungen sind gut realisierbar und Zustandsänderungen können relativ einfach bemerkt und synchronisiert werden. Die Objekte können gut über die Full-Stack JavaScript Architektur ausgetauscht und aktualisiert werden.

Node.js-Basis

Das Framework generiert eine hohe Anzahl an In- und Outputs durch die Echtzeit-Unterstützung. Diese Zahl steigt mit der Anzahl der beteiligten Clients, die sich wegen des kollaborativen Aspekts mit einer Applikation verbinden können. Node.js wartet mit einer guten Skalierbarkeit und einem nicht-blockierenden Input/Output-Algorithmus auf und soll für dieses Framework als serverseitige Infrastruktur eingesetzt werden.

NoSQL-Datenbank

Im Sinne des Full-Stack-JavaScript bietet es sich an, auch für die Kommunikation mit der Datenbank und Speicherung der darin enthaltenen Daten JavaScript zu verwenden. Viele NoSQL-Datenbanken speichern ihre Daten im JSON-Format und werden mit JavaScript angesprochen. Im Framework soll eine dokumentenbasierte NoSQL-Datenbank eingesetzt werden. Da die Daten zwar strukturiert, aber schemalos als JSON-Objekte abgelegt werden, begünstigt dies die Verwendung in einem generischen Framework.

Touch-Support

Applikationen, die mit diesem Framework entstehen, sollen auf mobilen Browsern aller gängigen Endgeräte funktionieren. Damit sie sich auch intuitiv bedienen lassen, soll die Unterstützung von Touch-Gesten gewährleistet sein.

Keine Offline-Unterstützung nötig

Ein Framework wie dieses ist prädestiniert dafür, mit neuen HTML5-Techniken wie „Application Cache“ [52] und „Web Storage“ [51] Offline-Funktionalität zu gewährleisten. Allerdings wird dies hier nicht ausdrücklich gefordert, um den Rahmen der Bachelor-Thesis nicht zu überschreiten.

4.4 Implementierung

Generisch

Das Framework soll generisch implementiert werden. Konkret bedeutet dies, dass der Programmcode keinen konkreten Anwendungsfall abdeckt, sondern sich für Anwendungen anpassen lässt, die die Technologien dieses Frameworks in dieser Kombination nutzen wollen. Die Aufgaben, die die resultierende Anwendung erledigen soll, sind den Anforderungen und der Kreativität des verwendenden Entwicklers überlassen.

5 Konzeption und Implementierung des Frameworks

In den folgenden Kapiteln geht es um die konkrete Konzeption und Umsetzung des Frameworks. Gegliedert wird das Kapitel in serverseitige und clientseitige Entwicklung. Jeder Abschnitt wird zur Veranschaulichung mit Codebeispielen aus dem Framework und dessen Anpassung für die Verwendung als Stundenplan-Gestalter versehen. Die Codebeispiele haben keinen Anspruch auf Vollständigkeit. Teilweise wurden Algorithmen auch vereinfacht. Dies ist der Lesbarkeit und Nachvollziehbarkeit geschuldet. Im Framework wurde darauf geachtet, bei Datei- und Ordnernamen sowie Variablennamen und Kommentaren die englische Sprache zu verwenden, damit keine Sprachbarrieren bei Verwendung des Frameworks in anderen Nationen entstehen. Der komplette Code liegt dieser Arbeit als CD bei.

5.1 Allgemeines Konzept

Das Framework soll eine Infrastruktur bieten, die es Entwicklern möglich macht, ihre eigenen Ideen mit bereits implementierten Algorithmen zu realisieren.

Aufgabe des Frameworks ist es also, eine client- sowie serverseitige Funktionalität herzustellen, ohne nur einen konkreten Anwendungsfall zu bedienen. Die konkrete Anwendung entwickelt der Programmierer dann durch Implementierung seiner eigenen Klassen für spezielle Komponenten sowie der Anpassung einiger Konfigurations-Dateien.

In der folgenden Grafik sieht man einen groben Überblick über die Elemente des Frameworks. Diese müssen durch den Entwickler angelegt beziehungsweise modifiziert werden.

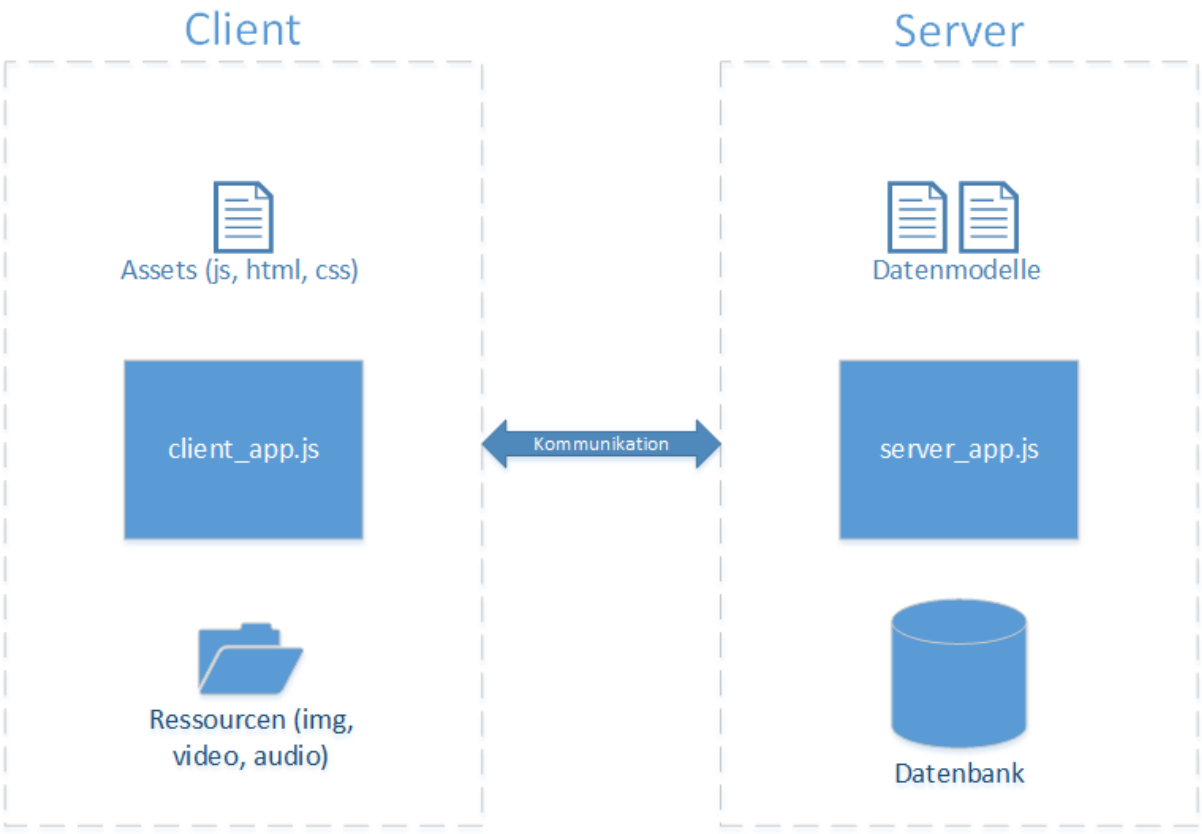

Abbildung 46: Allgemeiner Aufbau des Frameworks

Dieses Framework ist einzuordnen in die Kategorie der „White-Box“-Frameworks. Der Quellcode ist somit offen einsehbar und modifizierbar. Er besteht ausschließlich aus Dateien in lesbaren und bearbeitbaren Text-Formaten, wie etwa `.js`, `.css`, `.html` und `.json`.

Das Framework ist für Applikationen gedacht, die auf einer einzigen Seite im Browser ihre komplette Funktionalität zur Verfügung stellen (Single-Page Applikationen). Kernaspekt ist die Darstellung einer grafischen Benutzeroberfläche im `<canvas>`-Element des HTML-Dokuments sowie die Synchronisierung bestimmter Komponenten über mehrere Clients hinweg.

Das Framework unterscheidet zwischen einem statischen GUI, das nicht synchronisiert wird und Elemente wie Buttons und Schalter enthält und einem dynamischen GUI, dessen Inhalte zwischen allen kollaborierenden Clients synchronisiert werden. Darum bieten sich ein zweigeteiltes Datenmodell an, dessen Teile im Folgenden Application-Model (statisch) und User-Model (dynamisch) genannt werden.

Zur Verdeutlichung ein bildlicher Vergleich: In einer Textverarbeitungs-Software gibt es Elemente wie Buttons für Schriftstil und Schriftart sowie die weiße Schreibfläche selbst, die jedem Kollaborateur einmalig geliefert werden.
Dies entspricht auf Betriebssystem-Ebene einer Installation – im Browser wird dem Client das Application-Model übertragen. Weiterhin gibt es das Textdokument selbst. Es wird auf Betriebssystem-Ebene in eine separate Datei gespeichert. Im Framework gibt es dafür das User-Model, das die individuellen Änderungen speichert und – der Kollaboration wegen – mit allen autorisierten Clients synchronisiert. Jedes neu geöffnete „Dokument“ entspricht einem weiteren User-Model. Diese „Dokumente“ können jederzeit geladen und weiterbearbeitet werden.

In folgendem Bild wird das Framework in einer MVC-Struktur zwischen Server und Client dargestellt:

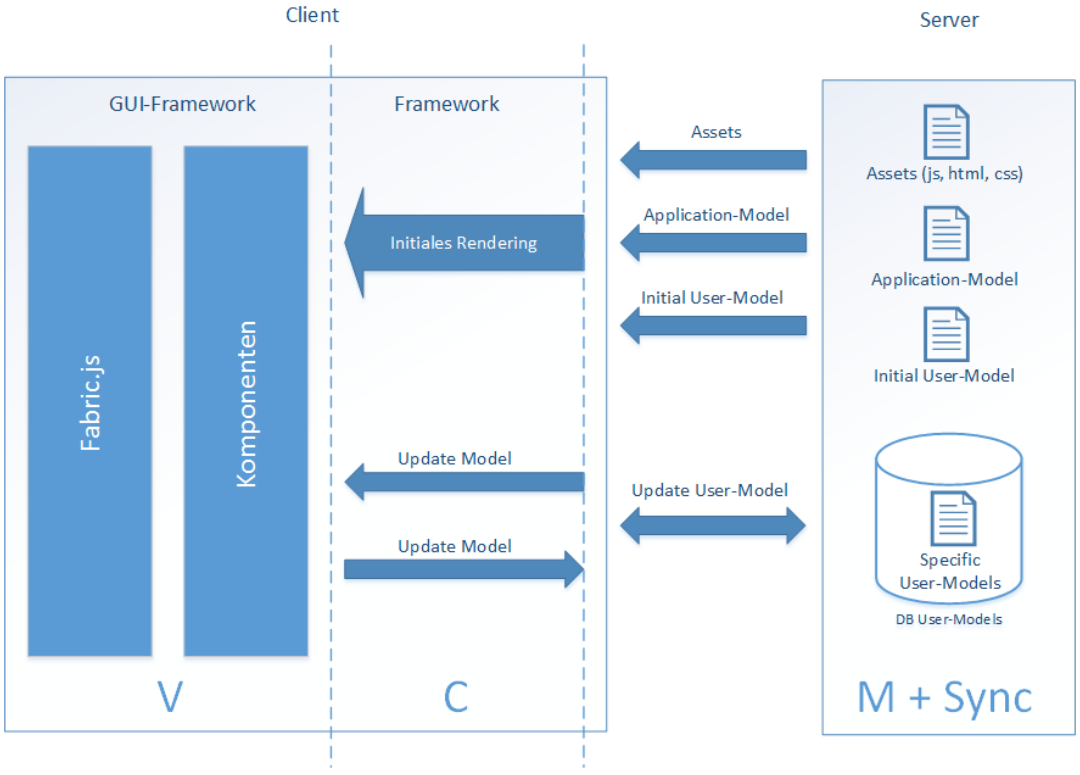

Abbildung 47: MVC-Modell des Frameworks

„M“, „V“, und „C“ in der vorangegangenen Abbildung stehen für „Model, View, Controller“, was zu Deutsch mit „Modell, Präsentation, Steuerung“ übersetzt wird. Die MVC-Architektur ist eine klassische Möglichkeit, um Applikationen zu gliedern und aufzubauen.

Von rechts nach links gelesen: Serverseitig sind die Datenmodelle gespeichert. Weiterhin kümmert sich der Server um die Synchronisation. Clientseitig findet das komplette Controlling statt. Außerdem wird hier die View erzeugt. Die View teilt sich in „Komponenten“, die der Entwickler als Subklassen definiert und die Fabric.js-Bibliothek, mit deren Hilfe aus den Komponenten sichtbare Objekte gerendert werden.

Die Adressierung der einzelnen User-Modelle erfolgt über ein Feld „url“, in dem ein eindeutiger Bezeichner gespeichert ist. Dieser Bezeichner erfüllt mehrere Zwecke:

Zum einen dient er als Parameter in der URL, die an den Server gerichtet ist.
Erreicht den Server ein GET-Request mit Zusatz wie `http://www.server.de/urlParam` wird serverseitig überprüft, ob dieser Parameter `urlParam` als Feld „url“ im Pool der User-Models vorhanden ist. Ist er das nicht, so wird dem User ein neues, leeres User-Model übermittelt. Ist er bereits vorhanden, so wird dem anfragenden Client das entsprechende, bereits modifizierte User-Model zurückgegeben.

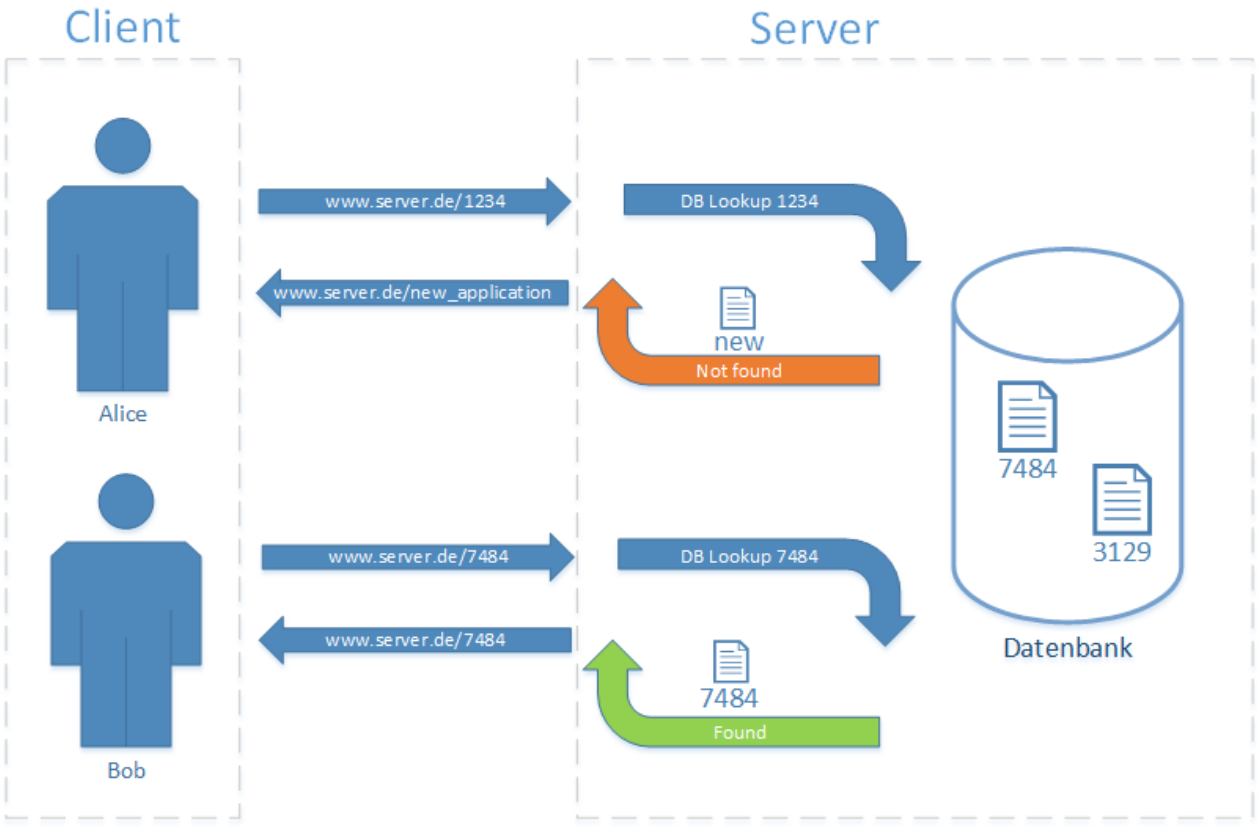

Abbildung 48: Der URL-Parameter als ID für User-Datenmodelle

Alice wird in diesem Beispiel natürlich nicht auf gut Glück einen Parameter eingeben. Auf einer Begrüßungs-Seite erhält sie einen Parameter vom Server, der sicher noch nicht vorhanden ist. Ruft sie die Seite dann damit auf, erhält sie wie gewünscht ein leeres User-Datenmodell.

Ein weiterer Zweck des eindeutigen Bezeichners innerhalb eines User-Models dient der Adressierung in der Datenbank. Änderungen werden über WebSockets an den Server übermittelt. Diesen Daten liegt weiterhin der `urlParam` des Senders bei. Der Server sucht das entsprechende User-Model anhand dieses Parameters aus der Datenbank heraus und speichert die Änderungen am korrekten Ort.

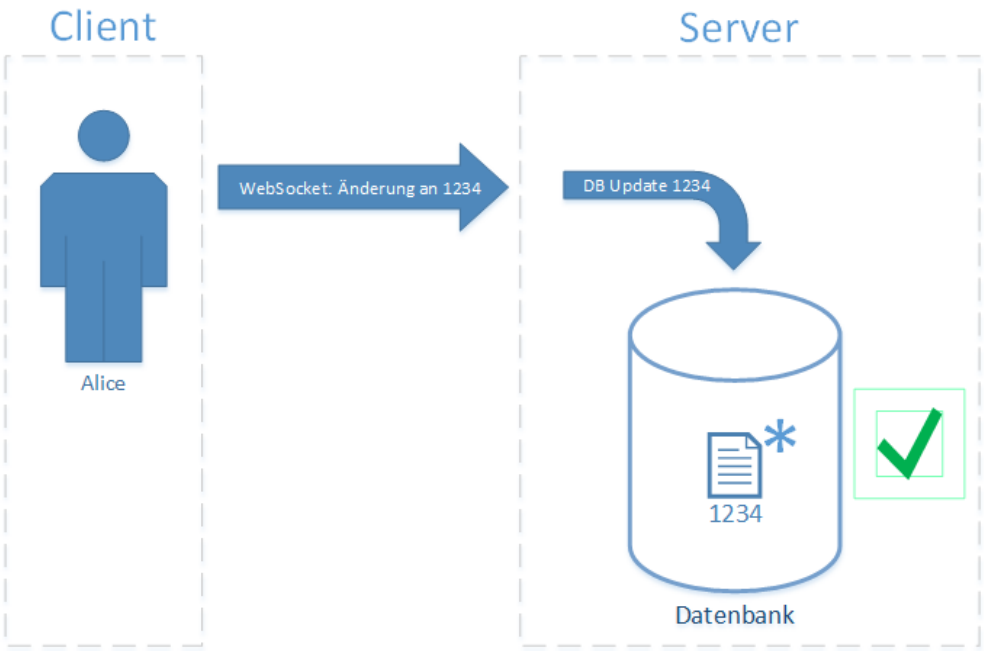

Abbildung 49: Datenbank-Adressierung anhand URL-Parameter

Ein dritter Zweck liegt in der Synchronisation der Daten. Wenn mehrere Sockets im Sinne der Kollaboration an einem User-Model arbeiten, so erkennt der Server anhand ihres URL-Parameters, an wen welche Änderungen zur Synchronisierung übermittelt werden müssen.

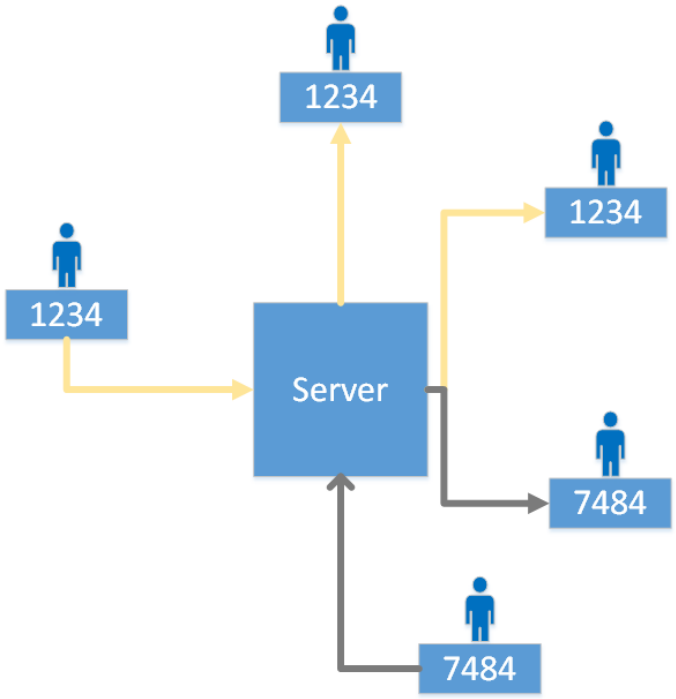

Abbildung 50: Synchronisation anhand des URL-Parameters

Die konkrete Konzeption wie auch die Umsetzung im Detail folgen in den nächsten Kapiteln.

5.2 Dateistruktur und -funktion

Die nachstehende Grafik zeigt die Dateistruktur, die der Betreiber der Applikation auf seinem Server speichert.
Auf oberster Ebene liegen die Dateien zum Betrieb des Node.js-Servers. Im Client-Server-Schema von Abbildung 46 auf Seite 61 repräsentieren sie die linke Seite.
Im Unterordner `public` liegen die Dateien, die ein Client sich beim Verbinden herunterlädt. Diese entsprechen dem rechten Teil der Abbildung 46.

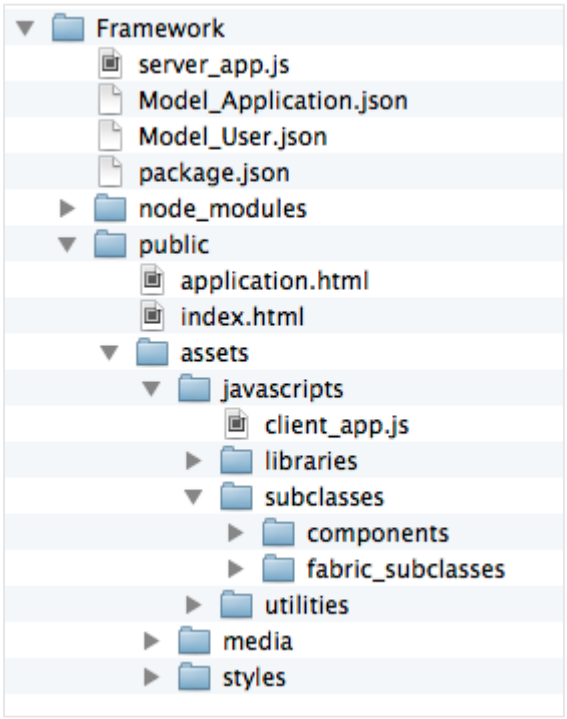

Abbildung 51: Dateistruktur des Frameworks

5.2.1 Server-Dateien – „server_app.js“, „package.json“ und „node_modules“

Die Datei `server_app.js` beinhaltet die Konfiguration des Node.js-Servers. In dem Ordner `node_modules` befinden sich die Node.js-Module, die mit dem Paketmanager NPM heruntergeladen und dem Projekt hinzugefügt wurden. Inhalte dieser Dateien werden ausführlich in Kapitel 5.5 vorgestellt.
In der `package.json` befinden sich in JSON-Notation geschriebene, allgemeine Informationen über das Projekt. Dazu gehören etwa verwendete Module und deren Versionsnummern, um automatisierte Update-Algorithmen anstoßen zu können. Ein Auszug sieht zum Beispiel so aus:

```
"name": "Express_SocketIO",
"version": "0.0.1",
"description": "Great Framework",
"main": "server_app.js",
"author": "Daniel Betz",
"dependencies": {
      "express": "4.1.1",
      "mongoskin": "1.4.4",
      "socket.io": "1.0.4",
      "clone" : "0.1.16"
}
```

5.2.2 Datenmodelle – „Model_Application.json“ und „Model_User.json“

In den nachstehenden Bildern sehen wir jeweils den Aufbau eines exemplarischen Application- und User-Models. Um die Struktur nachvollziehbar zu machen, stellt das User-Model Auszüge des für diese Thesis beispielhaft programmierten Stundenplan-Gestalters dar.

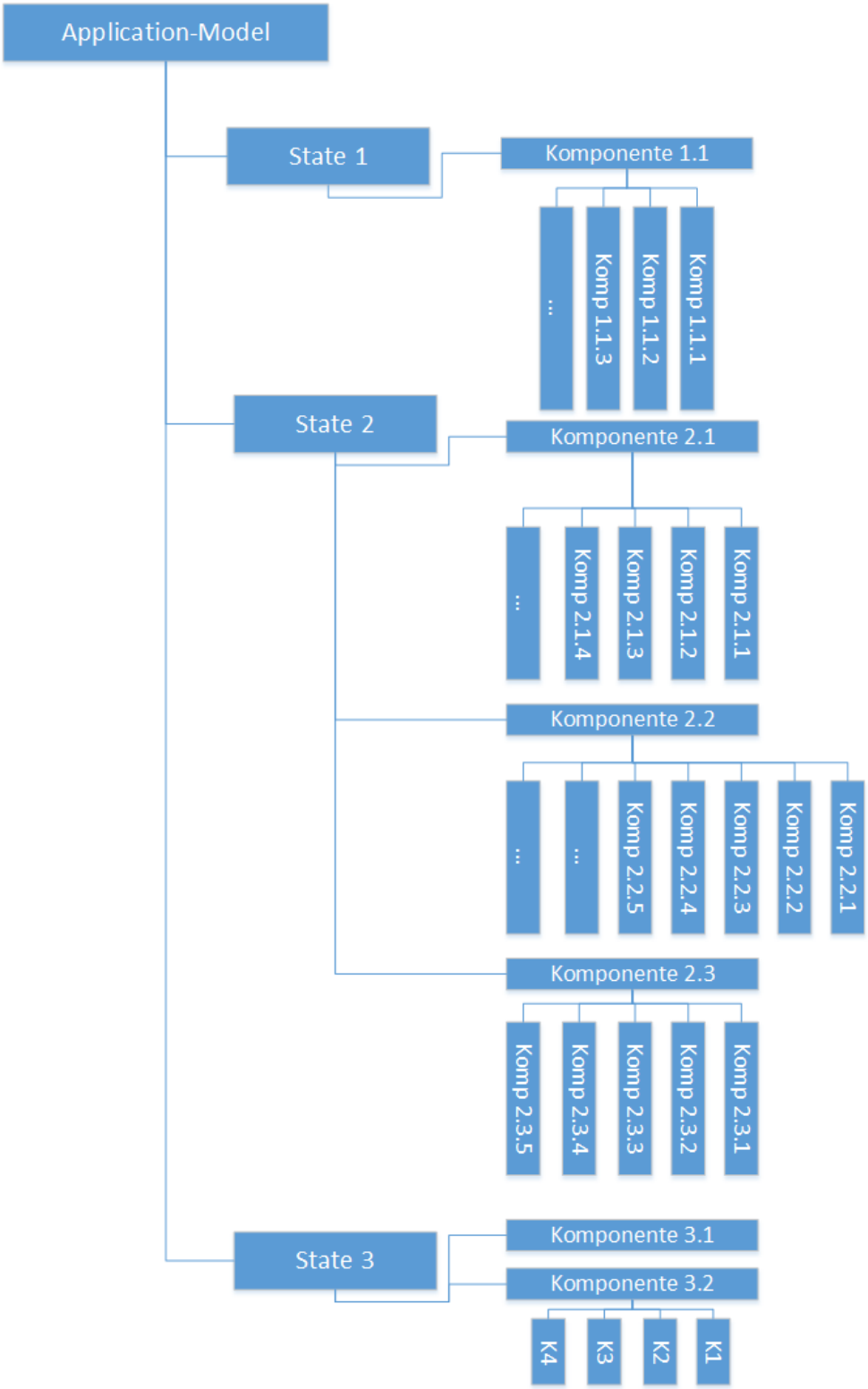

Abbildung 52: Application-Model

Die „States“ im Application-Model stellen die Zustände der Anwendung dar. So möchte man beim Öffnen der Applikation beispielsweise erst einen begrüßenden Text, bevor es per Klick

in den nächsten State geht, auf dem dann die eigentliche Applikation geladen wird. Von der State-Nummerierung abwärts wird hierarchisch verschachtelt, welche Komponente in welchem Kontext gerendert wird.

Die Datenmodelle sind als verschachtelte JSON-Objekte beschrieben. Ein Auszug des Stundenplan-Gestalters sieht so aus:

```
"state1" : {
      "canvas_bg" : {
            "type" : "Area",
            "debug": "S1 - Area - canvas_bg",
            "width": 1060,
            "height": 600,
            "left": 0,
            "top": 0,
            "fill": "rgb(244,236,220)"
      },
      "navi_left" : {
            "type" : "VirtualGroup",
            "debug": "S1 - VirtualGroup - navi_left",
            "width": 140,
            "height": 600,
            "left": 0,
            "top": 0,
            "fill": "rgb(244,236,220)",
            "config" : {
                  "align" : "vertical"
            },
            "fillings" : {
                  "el01" : {
                        "type" : "OneClickButton",
                        "name" : "goToBackgrounds",
                        "width" : 100,
                        "height" : 70,
                        "fill" : "rgb(255,250,177)",
                        "heading" : "Hintergrund"
                  },
                  "el02" : {
                        "type" : "PlusMinusButton",
                        "name" : "changeSize",
                        "width" : 100,
                        "height" : 70,
                        "fill" : "rgb(243,188,245)",
                        "heading" : "Größe",
                        "option_left" : "-",
                        "option_right" : "+"
                  },
            }
      }
}
```

Im User-Model finden sich Einträge zu den entsprechenden State-Komponenten, die modifizierbar sind. Diese werden dann als einfache Strings im User-Model gespeichert und können dort auch jederzeit wieder geladen werden.

So ist es im Beispiel Stundenplan-Gestalter Aufgabe des Application-Models, dem User verschiedene Hintergründe für eine Zeichenfläche anzubieten, aber Aufgabe des User-Models ist es, diese Auswahl für diesen speziellen Plan zu speichern.

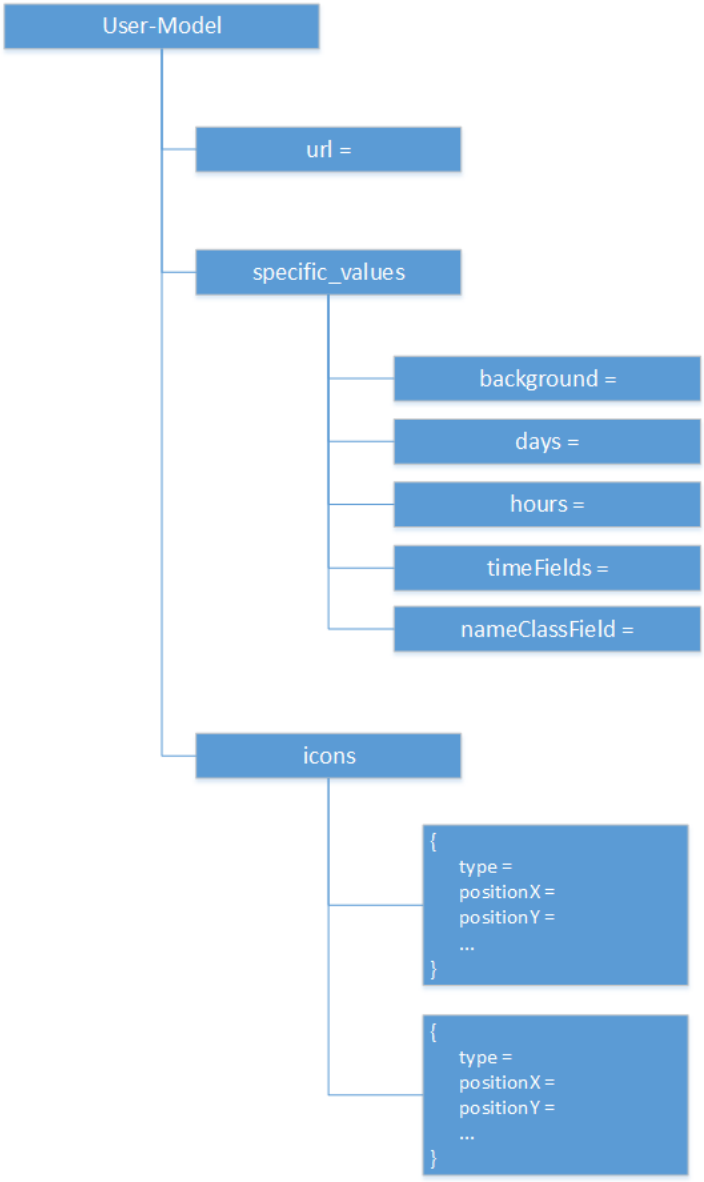

Abbildung 53: Beispielhaftes User-Model mit Schlüssel-Wert-Paaren (ohne Werte)

Ein Auszug hier könnte so aussehen:

```
{
	"url" : "1234",
	"specific_values" : {
		"background" : "media/images/background01.jpg",
		"days" : 5,
		"hours" : 6,
		"timeFields" : false,
		"nameClassField": true
	},
	"icons" : [
		"media/images/image01",
		"media/images/image04"
	],
}
```

5.2.3 Client-Dateien – Dateien in „public“

Im Ordner `public` sind die Dateien enthalten, die der Client sich beim Verbinden über die HTTP -Response herunterlädt.
Für das Beispiel des Stundenplan-Gestalters finden sich zwei HTML-Dateien. Ein User wird mit der `index.html` empfangen, in der er wählen kann, dass er einen neuen Stundenplan gestalten möchte.

Abbildung 54: Einfache index.html, die zur Erstellung eines neuen Stundenplans einlädt

Daraufhin wird er zur `application.html` weitergeleitet, in der der eigentliche HTML-Code mit `<canvas>`-Element und den zugehörigen JavaScript-Dateien geladen wird. An dieser Stelle wird die WebSocket-Verbindung initiiert und die Datenmodelle werden an den Client übertragen.

In den `assets` befinden sich Ressourcen zum Betrieb der Applikation. Dazu gehören Multimedia-Inhalte, zu finden im Ordner `media`. Weiterhin die Applikations-Logik, verteilt auf mehrere JavaScript-Dateien im Ordner `javascripts` sowie gestaltende CSS-Dateien im Ordner `styles`. Konkrete Abläufe auf Client-Seite werden in Kapitel 5.6 aufgezeigt.

5.3 Kommunikation zwischen Client und Server

Die Haupt-Kommunikation zwischen Client und Server findet über die in Kapitel 3.1.2 vorgestellten WebSockets statt. Mit Ausnahme des Verbindungsaufbaus wird ausschließlich das WebSocket-Protokoll genutzt, um Events und Daten auszutauschen.

5.3.1 Verbindungsaufbau

Der Verbindungsaufbau erfolgt klassisch über HTTP. Der Wechsel auf das WebSocket-Protokoll wurde ausführlich in Kapitel 3.1.2 erläutert und wird hier nur noch skizziert.

Unterschieden werden muss beim Verbindungsaufbau ab einer gewissen Stelle, ob der Client mit einem neuen User-Model beginnen oder ein vorhandenes weiterbearbeiten möchte. Der erste Fall wird hier dargestellt:

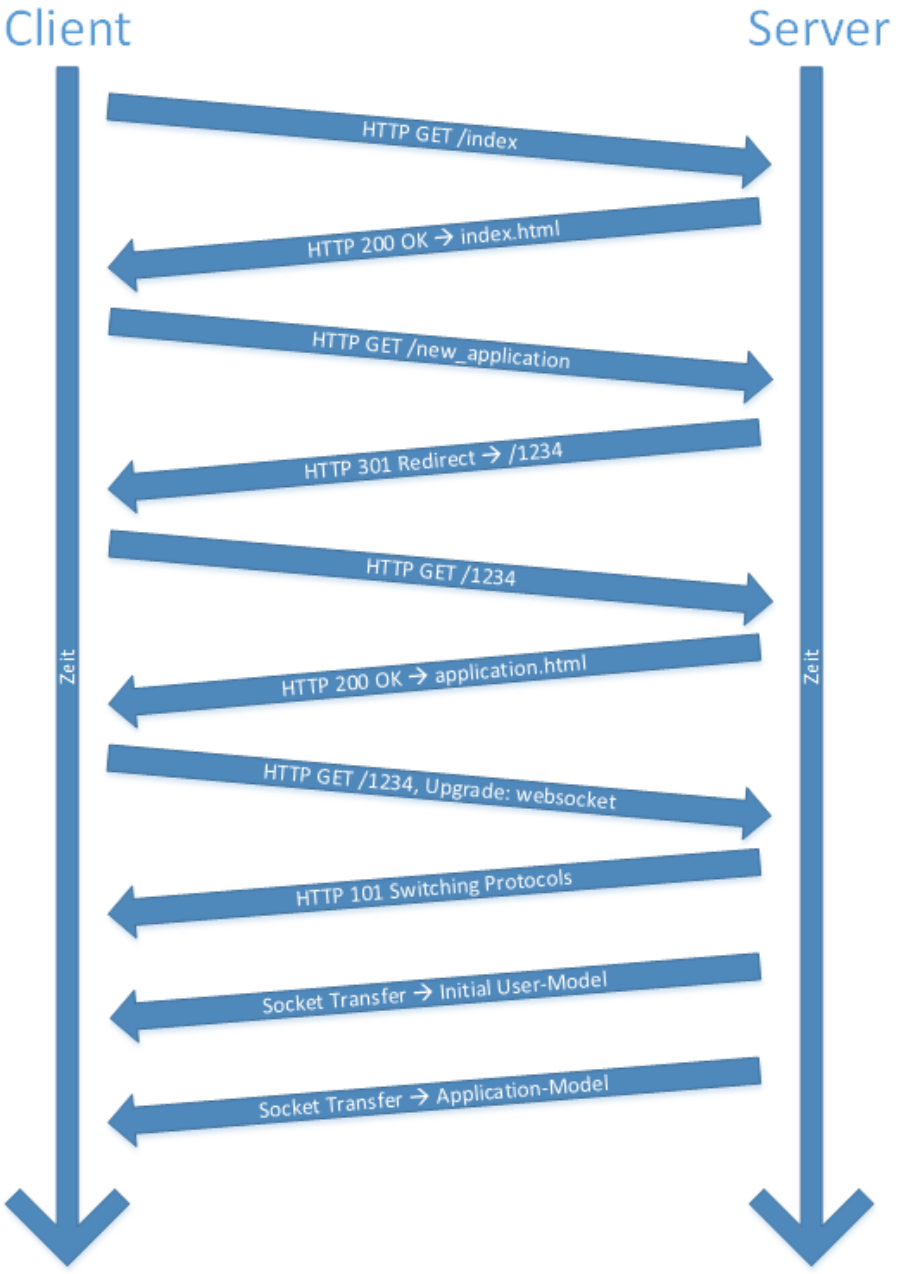

Abbildung 55: Verbindungsaufbau bei Wunsch nach neuem User-Datenmodell

Der Client möchte seine Arbeit mit einem neuen User-Model beginnen. Er wählt die entsprechende Option auf der `index.html` aus und wird auf die `application.html` umgeleitet, die die Anweisungen zur Socket-Verbindung enthält. Die Verbindung ist jetzt aufgebaut. Zur Demonstration werden im Schema direkt im Anschluss erste Daten mittels Sockets übertragen.

Der zweite Fall tritt ein, wenn der User bereits ein personalisiertes User-Model kreiert hat. In dieser Variante kennt der der Anwender den Link zu seinem personalisierten User-Model und ruft dieses mit dessen ID als Parameter hinter der URL auf. Als ID wurde hier vereinfacht „1234“ gewählt.

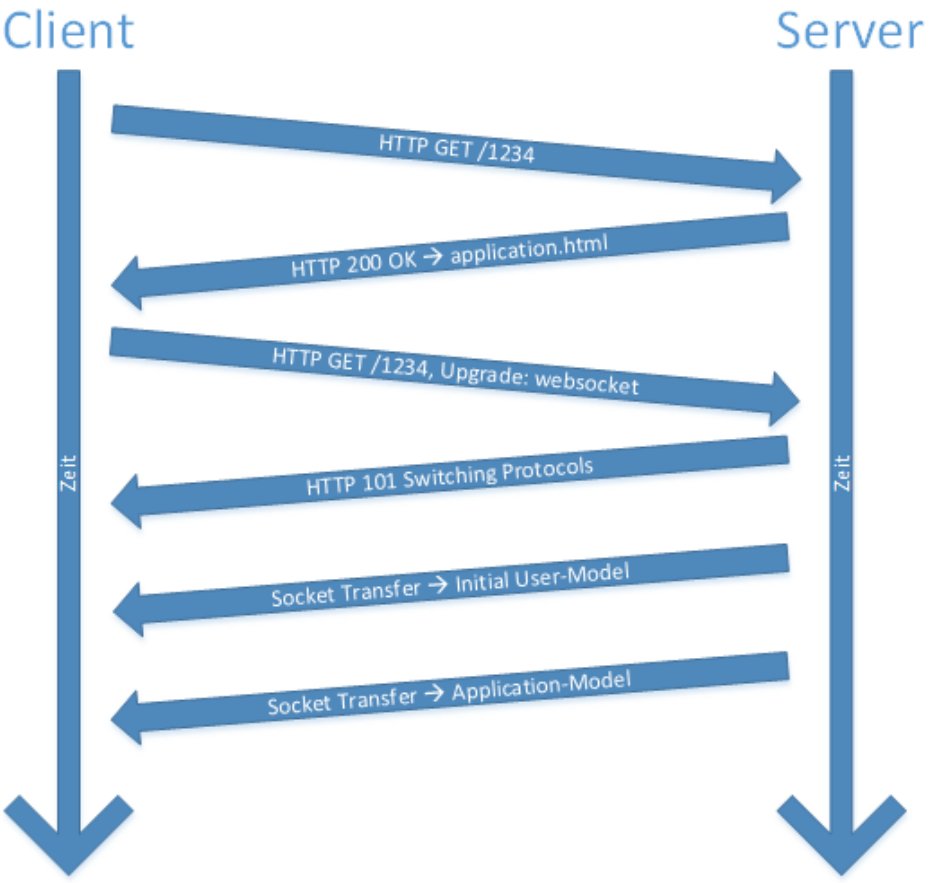

Abbildung 56: Verbindungsaufbau bei Wunsch nach existierendem User-Datenmodell

Der Server erkennt den Parameter und liefert dementsprechend direkt die `application.html` aus. In den ersten beiden Socket-Transfers wird neben dem üblichen Application-Model das konkrete User-Model gesendet.

5.3.2 Nachrichtenaustausch

Nach dem Verbindungsaufbau, dem erfolgreichen Wechsel des HTTP -Protokolls auf das WebSocket-Protokoll und dem Versand der Datenmodelle vom Server an den Client beginnt der kontinuierliche Austausch von Socket-Nachrichten, sobald clientseitig Änderungen am User-Model durchgeführt wurden.

Trifft ein solches Event am Server ein, so sucht er den entsprechenden Eintrag in der Datenbank, aktualisiert ihn und sendet daraufhin die Änderung an alle anderen beteiligten Clients. Folgende Grafik zeigt dies schematisch:

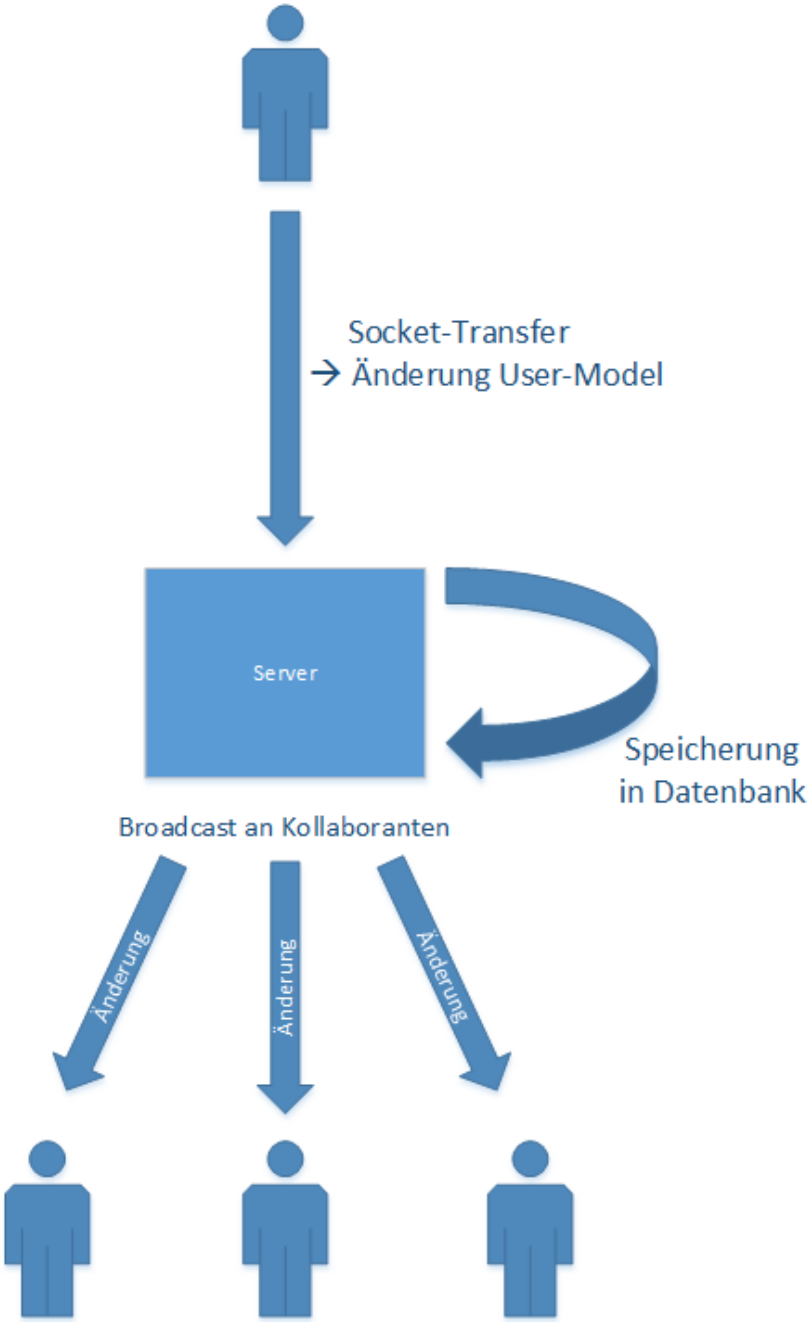

Abbildung 57: Nachrichtenaustausch bei laufender Applikation

5.4 Beispielapplikation Stundenplan-Gestalter

In diesem Kapitel soll auf die Beispielapplikation "Stundenplan-Gestalter" konkret eingegangen werden. Anwendungsspezifische Überlegungen werden in den beiden folgenden Kapiteln angestellt.

Vieles orientiert sich im Prototyp des Stundenplan-Gestalters an den Konzepten des bereits existierenden Flash-Stundenplan-Gestalters. So auch das grobe Design und erste Funktionen.

5.4.1 Flash-Version

5.4.1.1 Design

Für das Design des Flash-Stundenplan-Gestalters wurde eine Fläche von 1060x600 Pixeln für die Applikation gewählt. Der Rest der Seite ist mit einem Hintergrundbild gefüllt.

Geladen wird die Applikation für einen neuen Stundenplan in dem Status, in dem man einen Hintergrund auswählen kann.

Abbildung 58: Flash-Stundenplan-Gestalter: nach dem Starten [4]

Klickt man auf einen der Thumbnails, erscheint eine Hilfsseite, auf der die Oberfläche erklärt wird. Wird diese weggeklickt, wird die Hauptseite der Applikation geladen:

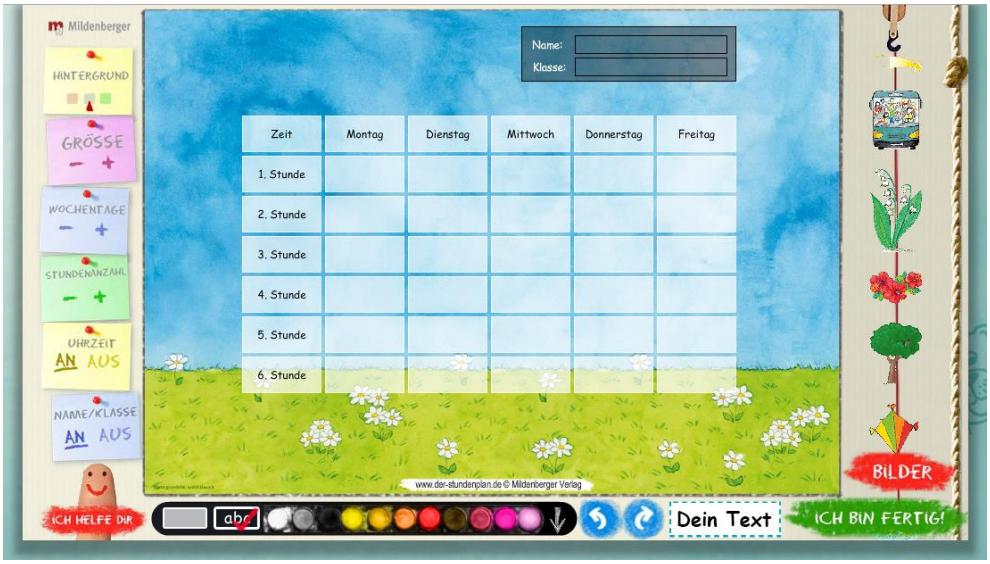

Abbildung 59: Flash-Stundenplan-Gestalter: Hauptseite der Applikation [4]

Die bisherigen beiden Seiten entsprechen im Framework den „States“ im Application-Model. Die Auswahl des Hintergrunds wird als `State0` bezeichnet, die eigentliche Gestaltung findet in `State1` statt.

Von State 1 aus erreicht man in der Flash-Version einige „Overlays“: Komponenten, die mit leichter Transparenz über die Zeichenfläche gelegt werden. Etwa eine Bilderbibliothek, wenn man den roten Bilder-Button klickt. Weiterhin lässt sich jederzeit die Hilfe über den Finger unten links aufrufen. Ein weiterer State wird geladen, wenn man auf „Ich bin fertig“ klickt:

Abbildung 60: Flash-Stundenplan-Gestalter: Ansicht nach Klick auf "Ich bin fertig" [4]

In diesem State sieht man eine Ansicht seines personalisierten Stundenplans und kann einen Druckdialog öffnen oder den Link zum Plan in die Zwischenablage kopieren beziehungsweise per Mail verschicken.

5.4.1.2 Funktionen

Der Flash-Stundenplan-Gestalter bietet zahlreiche Funktionen, die man über das umfangreiche GUI erreichen kann.

Nach dem Laden landet man in State 0, der Hintergrundauswahl. Klickt man auf einen der Hintergrund-Thumbnails, wird man in State 1 transferiert und bekommt den angeklickten Hintergrund als Bühnenwand der Zeichenfläche präsentiert.

In der Hauptansicht der Applikation befinden sich vier Bereiche zur Interaktion mit dem Benutzer. Eine Zeichenfläche in der Mitte und Button-Leisten links, rechts und unten.

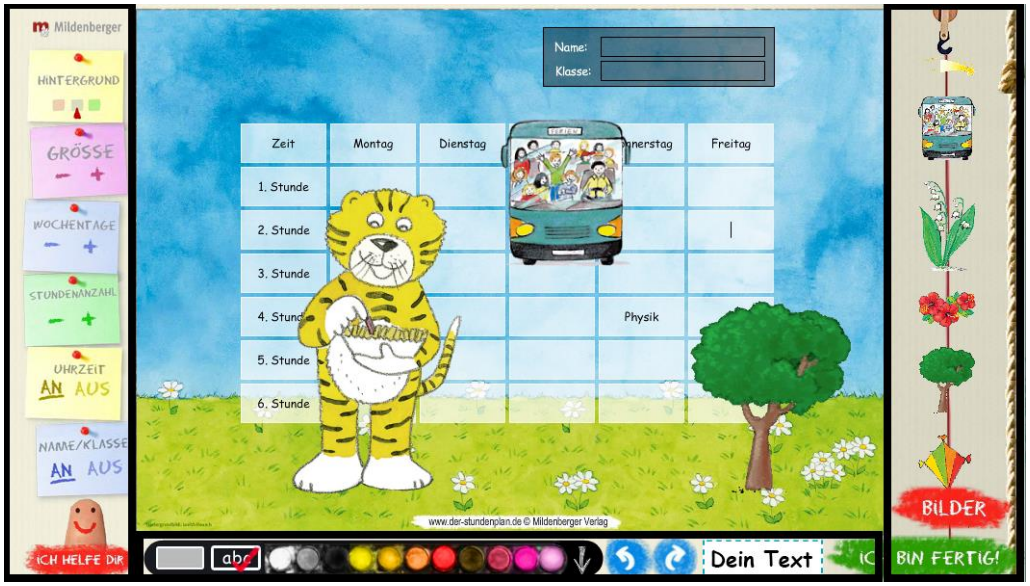

Abbildung 61: Flash-Stundenplan-Gestalter: Interaktions-Elemente in der Hauptansicht [4]

Die linke Button-Leiste enthält von oben nach unten folgende Funktionen:

1. Per Klick geht man in State 0 zurück, um die Hintergrundwahl zu ändern.
2. Plus und Minus lassen die Größe des Stundenplanrasters variieren.
3. Plus und Minus fügen Wochentage (= Spalten) hinzu oder entfernen sie
4. Plus und Minus fügen Fächer (= Zeilen) hinzu oder entfernen sie
5. An und Aus blenden die Spalte der Stundenanzeige ein oder aus
6. An und Aus blenden das Feld für Angabe von Name und Klasse ein oder aus
7. Per Klick geht man in einen Overlay, der hilfreiche Tipps präsentiert

Weiterhin gibt es eine Button-Leiste unterhalb der Zeichenfläche:

1. Links gibt es eine Farbwahl. Mit den beiden Buttons vor den Farbtöpfchen lässt sich auswählen, ob man Text oder Rasterfelder färben möchte.
2. Nun folgen zwei Pfeile mit Undo-Redo-Funktionalität.
3. Bei Klick auf „Dein Text“ lässt sich ein beschreibbares Textfeld auf die Zeichenfläche ziehen.

Schlussendlich kann der User mit der Fläche rechts der Zeichenfläche interagieren:

1. Ein Bereich, mit dem man durch die Icons scrollen kann. Bei Klick auf ein Bild lässt sich das entsprechende Bild in höherer Qualität auf die Zeichenfläche ziehen.
2. Ein Button „Bilder“, der einen Overlay öffnet, in dem man viele weitere Bilder-Sets anzeigen kann
3. Ein Button „Ich bin fertig“, der in State 2 führt

Die Icons (Bilder und Textfelder), die auf der Zeichenfläche platziert sind, lassen sich löschen, in dem man sie von der Zeichenfläche herunterzieht.

5.4.2 HTML5-Version

5.4.2.1 Design

In der Beispielapplikation für das Framework werden nicht alle States implementiert. In leicht abgewandelter Form werden die States für Hintergrund-Auswahl, Hauptapplikation und Link-

Anzeige als `State0`, `State1` und `State2` realisiert. Auch der Prototyp wird eine Größe von 1060x600 Pixeln haben. Diese definieren damit die Größe des Canvas-Elements.

Abbildung 62: Canvas-Stundenplan-Gestalter: nach dem Starten

Der Prototyp von State 0 lässt sich im folgenden Bild betrachten.

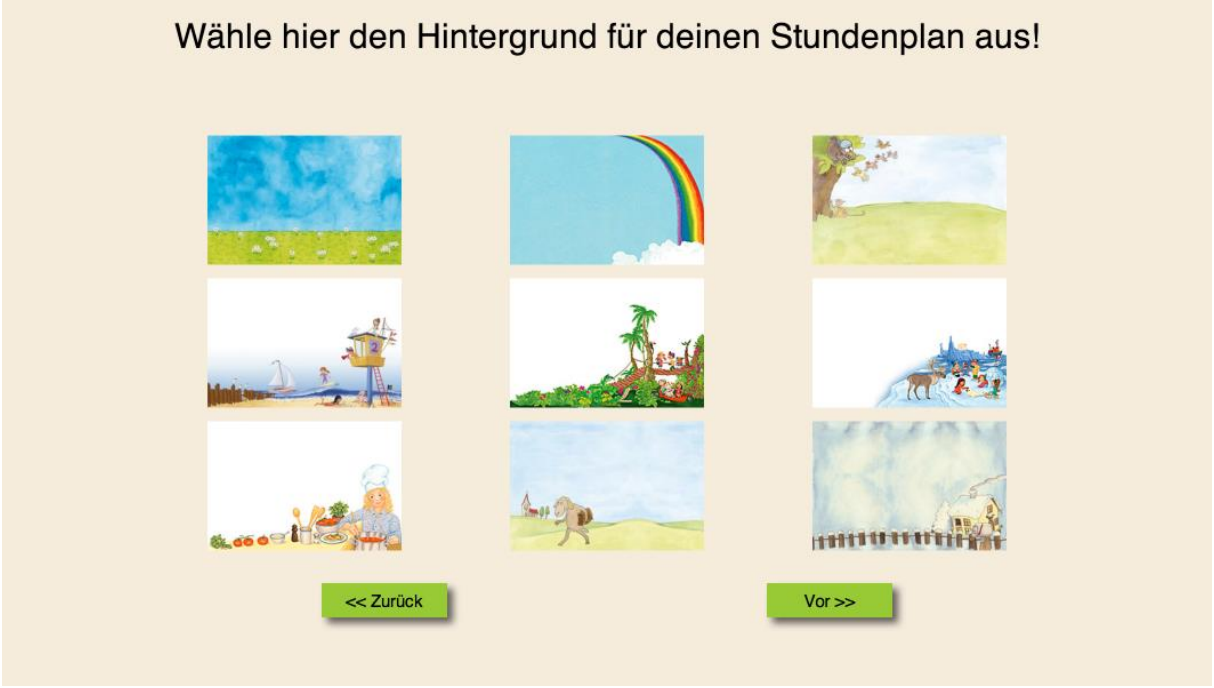

Abbildung 63: Canvas-Stundenplan-Gestalter: State 0 (Hintergrundauswahl)

Hat man einen Hintergrund gewählt, so wird man in folgenden State 1 weitergeleitet:

Abbildung 64: Canvas-Stundenplan-Gestalter: State 1 (Hauptapplikation)

Der Klick auf „Link teilen“ führt zu State 2:

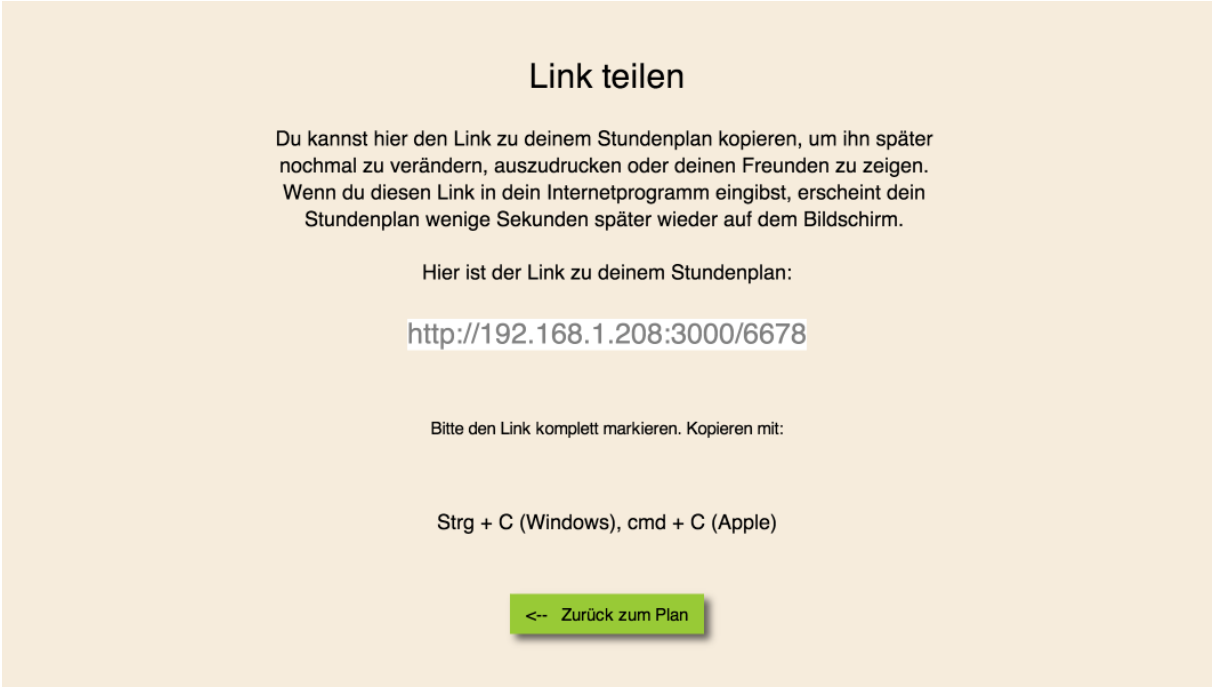

Abbildung 65: Canvas-Stundenplan-Gestalter: State 3 („Link teilen“)

5.4.2.2 Funktionen

Für den Prototypen des Stundenplan-Gestalters werden nicht alle Funktionen umgesetzt, dennoch wird es neben und unter der Zeichenfläche ebenfalls insgesamt drei Button-Leisten geben:

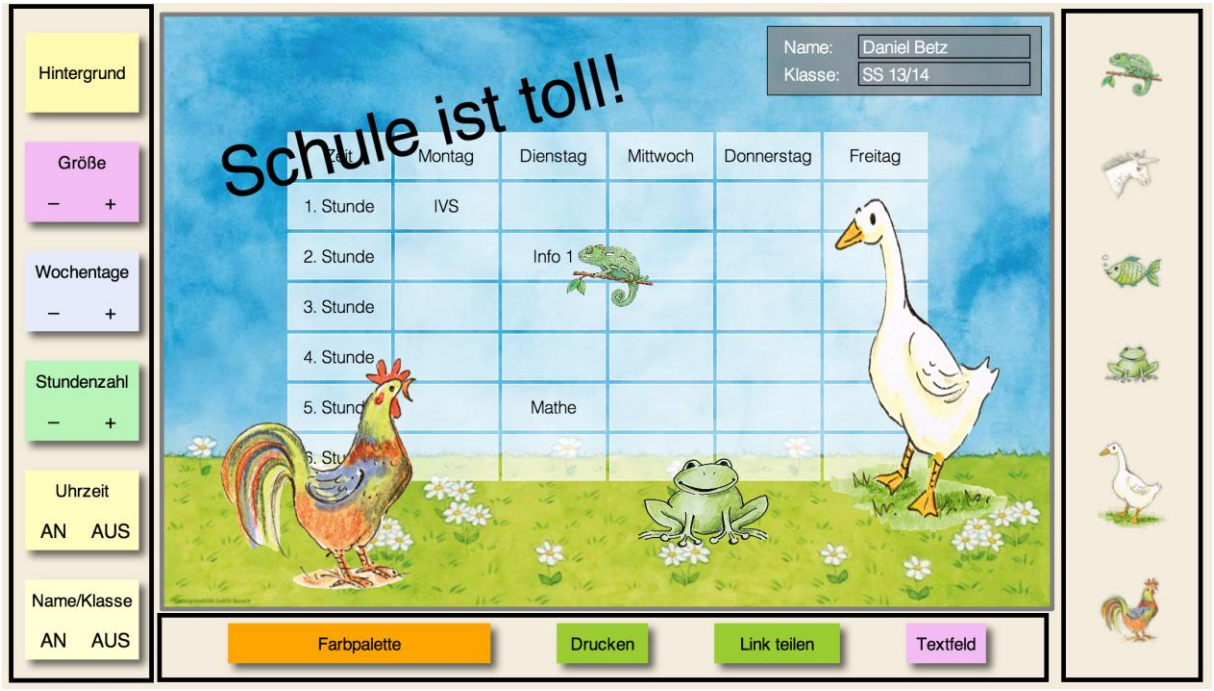

Abbildung 66: Canvas-Stundenplan-Gestalter: Interaktionselemente (nicht alle funktional)

Folgende Buttons erhalten Funktionalität:

Links:

- Der Button „Hintergrund“ zum Wechseln in State 0. Von dort aus kann man per Klick auf ein Thumbnail den Hintergrund aktualisieren und wieder in State 1 gelangen.
- Der Button „Name/Klasse“ zum Ein- und Ausblenden des entsprechenden Name/Klasse-Felds

Unten:

- Der Button „Drucken“, um ein PNG des Zeichenareals in einem Pop-Up zu öffnen.
- Der Button „Link teilen“ zum Wechseln in State 2. Dort lässt sich der Link zum Plan für kollaborative Zwecke kopieren. Über einem entsprechenden Klick landet man wieder in State 1.
- Der Button „Textfeld“ um ein vorausgefülltes, editierbares Textfeld auf dem Zeichenareal erscheinen zu lassen

Rechts:

- Buttons mit Thumbnail-Bildern, die bei Klick ein Bild in höherer Qualität auf dem Zeichenareal erscheinen lassen.

Weiterhin ist es möglich, Icons zu löschen, wenn man sie über die Zeichenfläche hinaus zieht.

5.5 Serverseitig – Node.js und MongoDB

Abbildung 47 auf Seite 62 zeigt, dass der größte Teil der Anwendungslogik auf Seiten des Clients ausgeführt wird.

Die Aufgaben des Servers lassen sich daher in vier Punkten zusammenfassen:

- Routing
- Persistente Speicherung des Anwendungszustands
- Bereitstellung des initialen User-Datenmodells beziehungsweise der bereits modifizierten User-Datenmodelle für die Weiterbearbeitung
- Synchronisation eingehender Daten mit allen autorisierten Clients

In den nächsten Unterkapiteln wird erläutert, wie diese Aufgaben konzeptuell und konkret umgesetzt werden. Die konkrete Umsetzung erfolgt in der Datei `server_app.js`.

5.5.1 Routing

Aufgabe des Routings ist es, eingehende Requests zu ihren Zielen zu führen. Im Framework treten drei Fälle von HTTP -Requests ein:

- GET-Request auf die begrüßende `index.html`
- GET-Request auf `/new_application`, wenn der entsprechende Link auf der index.html gewählt wurde – siehe Abbildung 54 auf Seite 69
- GET-Request auf `/:application_id`, wenn
  - ein neues User-Model angelegt wurde und der Client dorthin geleitet wird
  - ein bereits existierendes User-Model mit Angabe seiner ID als URL-Parameter zur Weiterarbeit aufgerufen wurde

Für das Routing wird das Framework „Express“ verwendet. Neben vielen praktischen Vorteilen, die bereits in Kapitel 2.1.4 aufgezählt wurden, bietet es Hilfe beim Umgang mit eintreffenden GET-Requests.

Zu Beginn wird das Framework in den lokalen Kontext des Projekts geladen. Dazu navigiert man in den Projekt-Ordner und führt in der Kommandozeile folgenden Befehl aus:

```
npm install express --save
```

Der Zusatz `--save` sorgt dafür, dass eine Referenz auf das Modul mitsamt Versionsnummer in der Datei `package.json` unter dem Punkt `dependencies` angelegt wird. Führt man später in der Directory ein `npm update` durch, so wird die Express-Installation automatisch auf Aktualität geprüft und gegebenenfalls aktualisiert.

Folgend lässt sich in der server_app.js das Modul mit folgender Zeile einbinden:

```
var express = require('express');
```

Nun ist der gesamte Funktionsumfang des Frameworks über die Variable `express` erreichbar. Man erstellt eine `app` mithilfe dieser Variable:

```
var app = express();
```

und definiert folgende Route:

```
app.get('/', function (req, res) {
      res.sendfile(__dirname + '/public/index.html');
});
```

In vorangegangenem Codebeispiel sieht man das Konzept hinter Express. Die Funktion `get()` bekommt als Argument den zu erwartenden Pfad eines HTTP -GET-Request – `'/'` steht hier für den Aufruf der URL ohne Parameter, vergleichbar mit `www.hs-offenburg.de/` – und arbeitet im Callback mit `req` und `res`, den Stellvertretern für den HTTP -Request und die HTTP -Response des eingehenden Aufrufs. Die Variable `req` wird hier gar nicht benötigt, dem anfragenden Client wird mit `res.sendfile()` schlicht die genannte HTML-Datei zugesendet.

Ähnlich wird mit den übrigen beiden HTTP -Requests verfahren. Klickt jemand auf das Bild zur Erstellung eines neuen User-Datenmodells, so generiert er einen neuen Request, der folgender Route folgt:

```
app.get('/new_application', function (req, res) {
      newPath = createPath();
      res.redirect(newPath);
});
```

Die selbstgeschriebene Funktion `createPath()` erstellt eine zufällige Zahl, die an die URL angehängt wird. Diese neue URL wird dem Client mit der HTTP -Antwort `301 redirect` zurückgesendet, wie in Kapitel 5.3.1 erläutert. Dieser startet nun erneut einen HTTP -Request auf die neue URL. Der Parameter wird wie folgt von Express ausgelesen:

```
app.get('/:url_param', function(req, res){
      var url_param = req.param("url_param");
      res.sendfile(__dirname + '/public/application.html');
});
```

Nun kann dem Client unter seiner individuellen URL die `application.html` mit den benötigten `assets` geliefert werden.

Der Parameter `urlParam` dient hier vorerst nur der korrekten Zuweisung des GET-Requests. Die Unterscheidung, ob der Client ein neues, leeres User-Datenmodell bekommt, oder ob unter dem Parameter `:urlParam` bereits ein User-Model in der Datenbank gespeichert ist, das weiterbearbeitet werden soll, ist nicht im Routing implementiert, sondern wird von der WebSocket-Logik übernommen. Diese Differenzierung wird im Kapitel 5.5.3 erläutert.

5.5.2 Datenbankanbindung

Aufgabe der Datenbank ist es, die einzelnen User-Datenmodelle persistent zu speichern und jederzeit verfügbar zu machen.

Als Datenbank wurde die NoSQL-Datenbank MongoDB gewählt. Diese Entscheidung wurde in Kapitel 3.2.2 erläutert.

Es gibt viele Node.js-Module, die eine Schnittstelle zu MongoDB-Datenbanken ermöglichen. In diesem Framework wurde ein Modul mit einem sehr niedrigen Abstraktionslevel gewählt, das sich nahe am nativen Treiber der Datenbank orientiert und diesen eher ergänzt als komplett eigene Funktionen zu verwenden. Das Modul mit dem Namen „Mongoskin“ [123] wird nach dem Download mit NPM wie folgt eingebunden:

```
var mongo = require('mongoskin');
```

Anschließend wird eine Verbindung zum Datenbank-Server mittels

```
var db = mongo.db("pathToDB");
```

herstellt. Im dritten Schritt wird eine Collection namens `userModels` referenziert.

```
db.bind('userModels');
```

Ab sofort lassen sich beispielsweise die in Kapitel 3.2.2.2 vorgestellten Befehle auf `db.userModels` ausführen, etwa `db.userModels.find({url: '1234'});`

5.5.3 Verteilung der Datenmodelle

Folgend wird die serverseitige Verteilung der Datenmodelle anhand eines einzelnen Application- und eines einzelnen User-Models erläutert, wobei ersteres vor zweitem verteilt werden soll. Je nach Anwendung kann die Anzahl und Reihenfolge der Datenmodelle natürlich variieren.

Die beiden Datenmodelle sind zur Übersicht ausgegliedert in separaten JSON-Dateien gespeichert, wie in Kapitel 5.2.2 aufgezeigt. Um diese zur Laufzeit auszulesen und in ein JavaScript-Objekt reinzuschreiben, wird dem Node.js-Server mittels

```
var fs = require('fs');
```

über die Variable `fs` der Zugriff auf das Dateisystem ermöglicht. Das Modul `fs` ist ein Bestandteil der Node.js-Installation und muss nicht über den Paketmanager NPM nachgeladen werden. Mit folgender Funktion wird der Inhalt der JSON-Datei geladen und in der Variable `dataModel` gespeichert:

```
var dataModel;

fs.readFile('pathToFile', function (err, data) {
  dataModel = JSON.parse(data);
});
```

Dies wird mit beiden Datenmodellen durchgeführt, um sie anschließend per WebSocket zu versenden.

Bei der Verbindung eines neuen Clients zu dem WebSocket-Server tritt ein `connection`-Ereignis auf, bei dem der Server nun überprüfen muss, mit welchem URL-Parameter diese initiiert wurde. Entsprechend wird dem Socket zu dem Application-Datenmodell ein initiales User-Datenmodell oder ein bereits modifiziertes ausgeliefert.

Für das Handling mit WebSockets wird auf ein weiteres, sehr etabliertes Framework gesetzt, das bereits in Kapitel 3.2.1.4 vorgestellt wurde: Socket.io.

Socket.io wird ebenso wie Express per NPM dem Projekt hinzugefügt und mittels folgendem bekannten Code verwendbar gemacht:

```
var io = require('socket.io');
```

Sobald ein Socket sich verbindet, registriert der Server das mit diesem Listener:

```
io.sockets.on('connection', function (socket) {
      //Do some action
}
```

Dem Callback wird der Socket, der sich verbunden hat, als Variable `socket` übergeben. So lässt sich nun innerhalb dieses Callbacks das User-Datenmodell an den `socket` versenden:

```
socket.emit('send_ApplicationModel_fStC', applicationModel);
```

Das Versenden von Nachrichten wird generell mit `emit()` ausgelöst. Als erstes Argument wird der Name des Events genannt, als zweites, optionales Argument folgt ein JavaScript-Objekt, in dem eventuell zu übermittelnde Daten gespeichert sind. Dem Eventnamen wurde ein `fStC` hinzugefügt, was beim Debugging von hohem Socket-Verkehr hilft, die Laufrichtung zu erkennen. In diesem Fall steht die Abkürzung für „**f**rom **S**erver **t**o **C**lient“.

Chronologisch folgend wird jetzt das User-Datenmodell versendet. Dafür muss zuvor geprüft werden, ob der URL-Parameter des Sockets bereits in der Datenbank einen Repräsentanten in Form eines User-Datenmodells besitzt oder ob ihm ein neues geliefert werden soll.

Der URL-Parameter des verbundenen Sockets wird von Socket.io im `socket`-Objekt gespeichert und in der Variable `socketPath` gespeichert. Jetzt muss überprüft werden, ob der Parameter bereits als Wert der Variable url in den existierenden User-Datenmodellen vorhanden ist oder nicht.

Mongoskin bietet eine hierzu nützliche Funktion: `distinct()`. Ihr wird der Name eines Schlüssels übergeben und sie durchsucht daraufhin alle Dokumente der Collection nach diesem Schlüssel und speichert dessen Wert in ein Array. `distinct()` ist wie alle Datenbank-Zugriffe asynchron, deshalb muss mit dem resultierenden Array im Callback der Funktion weitergearbeitet werden.

Variante 1: User-Model ist bereits vorhanden

```
db.userModels.distinct('url', function(err, array) {
      for(var i in array){
            if(array[i] === socketPath){
                  //Sende passendes User-Model an Socket
                  break;
            }
      }
});
```

In der Callback-Variable `array` sind die `url`-Werte gespeichert. Die `if`-Bedingung prüft in einer Schleife, ob der URL-Parameter der verbundenen Sockets im Array vorkommt. Ist dies der Fall, so wird das passende User-Model an den Client gesendet und die Schleife wird beendet.

Variante 2: Neues User-Model ist gewünscht

Wird die Schleife ohne Erfolg durchlaufen, so bemerkt der Algorithmus das anhand eines Booleans und führt folgende Zeilen aus:

```
var newModel = clone(userModel);
newModel.url = socketPath;
db.userModels.insert(newModel, function(err, result) {
            if (result) {
                  //sende neues User-Model an Socket
            }
);
```

Zuerst klont er das aus dem JSON-Dokument ausgelesene User-Model in ein neues Objekt, um nicht fälschlicherweise die originale User-Model-Vorlage zu verändern. Dies geschieht mithilfe des Node.js-Moduls „Clone“ [124], das mit dem Befehl

```
var clone = require('clone');
```

wie bekannt eingebunden wird. Anschließend wird das `url`-Attribut des neuen User-Models auf den URL-Parameter des eingehenden Sockets gesetzt und in die Datenbank eingefügt. Der `insert`-Befehl von Mongoskin erzeugt ein Callback, in dem man weiter arbeiten kann, sobald der `insert` erfolgreich abgeschlossen wurde. Erhält die Variable `result` den Wert `true`, kann auch das neu erzeugte User-Model an den Socket übertragen werden.

`clone()` kann in einer künftigen Version des Framework das Feature bieten, bestehende User-Models zu kopieren und den User an einer Kopie weiterarbeiten zu lassen, ohne das Original zu beeinflussen.

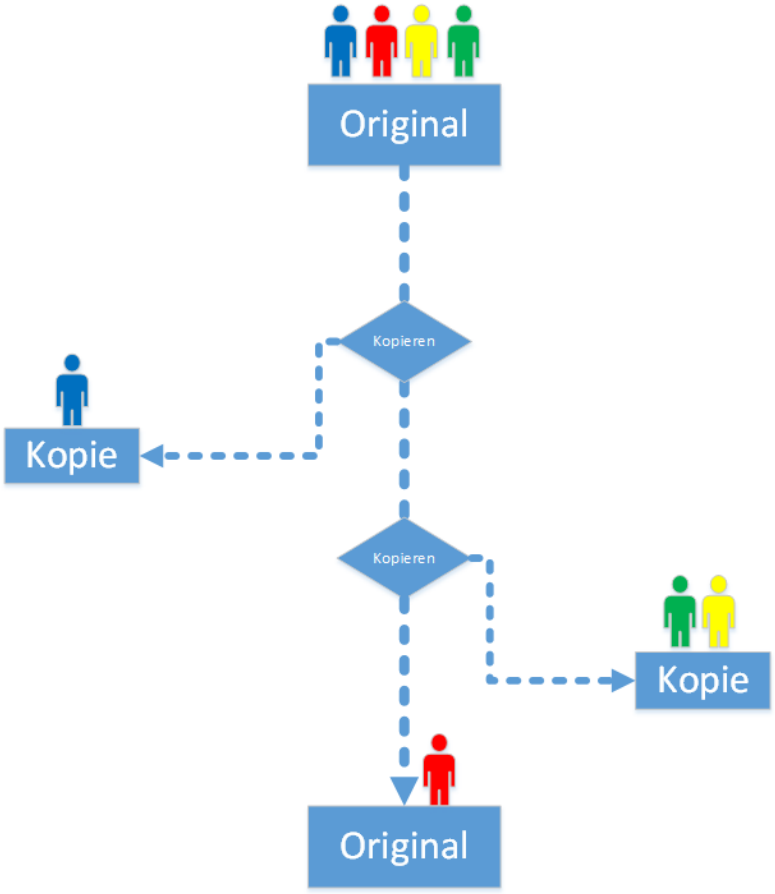

Abbildung 67: Eine Möglichkeit, Kopien von bestehenden User-Models zu erstellen

Um dem Server deutlich zu machen, dass man eine Kopie eines existierenden User-Models erstellen und bearbeiten möchte, wäre beispielsweise das Anhängen eines weiteren URL-Parameters eine Möglichkeit. So erstellt man mit folgender URL eine Kopie des User-Models 1234:

```
http://www.server.de/1234&copy=true
```

Dieses Feature existiert rein konzeptuell und ist in der ersten Version des Frameworks nicht vorgesehen.

5.5.4 Synchronisation

Bei der Synchronisation besteht die Herausforderung darin, Änderungen nur an die Sockets zu schicken, die am gleichen User-Model arbeiten wie ihre Kollaborateure.

Auch hier bietet Socket.io eine komfortable Möglichkeit: Räume. Der Begriff „Raum“ kann hier sehr bildlich verstanden werden. Auch Chat-Dienste arbeiten damit. Auf einem Chat-Server können mehrere Räume geöffnet sein. Es erhalten nur die Teilnehmer eine Nachricht, die sich im selben Raum befinden wie der Sender selbst.

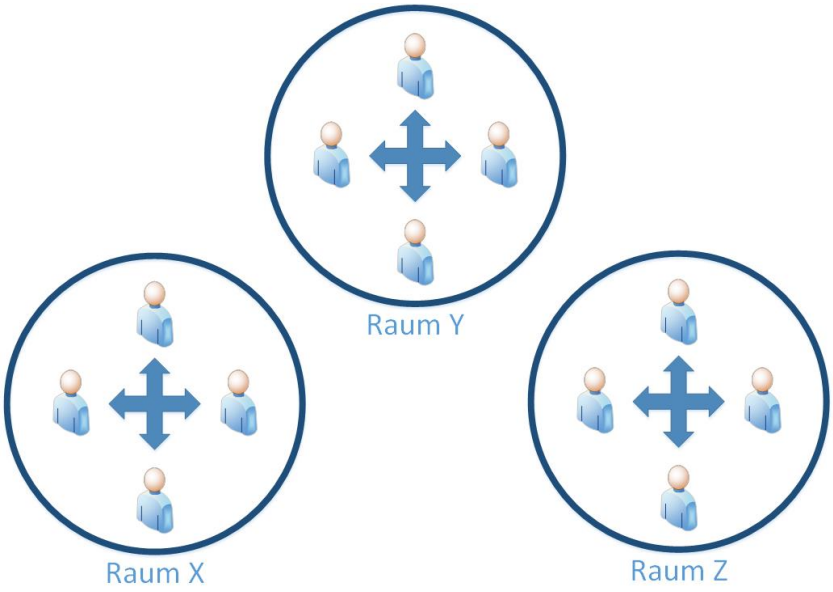

Abbildung 68: Räume in Socket.io

Sockets können Räume betreten (join) und verlassen (leave). Als eindeutige ID dient ein Raumname.

Es liegt an der Hand, die ID, die bereits als URL-Parameter übergeben wird und für die einzelnen User-Modelle in der Datenbank benutzt wird, auch für den ganzen Raum zu benutzen – schließlich wirken alle Rauminsassen auf dieses eine User-Modell ein.

So kommt es, dass ein Socket nach der Verbindung nicht nur die Prüfung durchläuft, ob er ein neues oder ein bereits existierendes User-Model bearbeiten möchte, sondern dass er auch den dazugehörigen Raum betritt. Dies geschieht mit dem Befehl `socket.join()`. Der URL-Parameter ist nach wie vor in der Variable `socketpath` gespeichert, weswegen wir einfach folgenden Befehl ausführen:

```
socket.join(socketPath);
```

In diesem Fall ist es nicht von Bedeutung, ob der Raum bereits existiert, also andere Sockets gerade das gleiche User-Model bearbeiten oder nicht. In zweitem Fall wird der Raum einfach erstellt.

Fortan lassen sich Nachrichten zwischen den Sockets eines Raums mit diesen beiden Befehlen austauschen:

```
socket.to(room).emit();        //an alle Sockets des Raums
io.sockets.to(room).emit();    //an alle Sockets des Raums OHNE den Sender
```

Mit diesen konzeptionellen Überlegungen wurde folgendes Vorgehen beim lokalen Ändern eines Werts am User-Model durchgeführt:

1. Änderung lokal im Canvas
2. Senden eines `changed`-Events an den Server
3. Speicherung der Änderung in der Datenbank

4. Broadcasten des `changed`-Events an alle anderen Teilnehmer im Raum

Der clientseitige Code für Punkt 1 und 2 wird in Kapitel 5.6 aufgegriffen.

Für das serverseitige Kapitel hier wird davon ausgegangen, dass ein EventListener serverseitig auf ein Ereignis `'userModel_changed'` hört, das ein Client ihm bei Änderungen sendet.

Der serverseitige Code bei einer Änderung sieht demnach wie folgt aus:

```
socket.on('userModel_changed_fCtS', function (data) {
      db.userModels.update(
            {url : data.URL},
            { $set: /*changed values*/},
            function(err, result) {
                  if (result){
                        console.log('Value changed');
                  }
            }
      );

      io.sockets.to(room).emit('userModel_changed_fStC', /*changed
values*/);
});
```

Erst wird serverseitig die Änderung im korrekten User-Model in der Datenbank gespeichert. Anhand der Bedingung `url: data.URL` wird gewährleistet, dass die ID `url` in der Datenbank mit der URL des Change-Events übereinstimmt, die im Objekt `data` mitgeschickt wurde. So landet die Änderung im korrekten User-Model. Die Bedingung kann im SQL-Kontext als `where`-Statement verstanden werden.
Anschließend wird die Änderung an alle anderen Sockets im Raum weitergegeben, die ihrerseits wiederum lokal mit einem EventListener auf dieses Ereignis hören und es dann lokal in ihrem Canvas weiterverarbeiten.

Hervorzuheben ist, dass bis auf die initiale Übertragung des User-Datenmodells fortan nur noch *einzelne Änderungen* synchronisiert werden, nicht mehr das ganze Modell. Dies optimiert den Datenverkehr, da es die Menge der Daten klein hält.

5.6 Clientseitig – HTML 5 und JavaScript

Aufgabe des Clients ist zum einen das Controlling, also die Verarbeitung der beiden Datenmodelle, zum anderen die View, die er daraus erstellt.

Zentrale Datei des Clients ist die `client_app.js`. In ihr wird die Socket-Verbindung initialisiert, woraufhin alle weiteren Interaktionen möglich werden. Weiterhin werden hier die Datenmodelle empfangen, in lokale JavaScript-Objekte transferiert und initial bearbeitet.

In den Unterkapiteln dieses Kapitels werden zuerst drei JavaScript-Helfer-Bibliotheken vorgestellt, die im Framework Einsatz finden, danach folgt ein detaillierter Überblick über die implementierten JavaScript-Dateien und ihre Funktionalitäten, im dritten Teil wird der Ablauf nach Start des Programms mit allen bis dahin beschriebenen Komponenten durchgespielt.

Abbildung 69: Detaillierte Darstellung der clientseitigen Ordnerstruktur

5.6.1 Bibliotheken

5.6.1.1 Darstellung – Fabric.js

Zur Darstellung von Elementen im Canvas wird die Canvas-Bibliothek Fabric.js verwendet. Die Vorteile der Verwendung von Bibliotheken sowie die Auswahl der richtigen Bibliothek wurden in Kapitel 3.2.3 erläutert.

5.6.1.2 Modularisierung – RequireJS

Die Code des Frameworks selbst sowie die Anpassung auf eigene Bedürfnisse mit eigenen Subklassen umfasst viele Zeilen JavaScript. In eine Datei geschrieben wäre der Code sehr lang und unübersichtlich. Außerdem würde beim ersten Laden der Seite erstmal ein sehr großes JavaScript-File heruntergeladen werden.

RequireJS [125] schafft hier Abhilfe. Es dient zur Modularisierung des Codes. So können einzelne Funktionen in eigene Dateien ausgegliedert werden. Alles, was RequireJS benötigt, um den Code zu verwenden, ist ein `return` am Ende des Moduls, das das Objekt beziehungsweise die Funktion zurückgibt.

Weiterhin werden die einzelnen JavaScript-Dateien erst geladen, wenn sie wirklich gebraucht werden. Ist ein Modul je nach Gebrauch einer Website oder Applikation optional, so wird es erst nach expliziter Anforderung nachgeladen.

Zudem dienen Module als lokale Gültigkeitsbereiche. So hilft RequireJS dagegen, Variablenkonflikte durch gemeinsame Gültigkeitsbereiche zu provozieren oder gar globale Variablen zu initialisieren.

Das Teilen des Codes funktioniert auch ohne Modularisierung, wenn man die einzelnen JavaScript-Dateien separat in die HTML-Datei einbindet. Hier ist allerdings darauf zu achten, dass die Reihenfolge stimmt. Beides nimmt einem RequireJS ab. Mit ihm benötigt man nur einen einzigen `<script>`-Eintrag im Kopf des HTML-Dokuments. Die zeitliche Reihenfolge wird durch die Angabe von Abhängigkeiten zwischen den Modulen automatisch bestimmt.

Mit folgender Zeile wird RequireJS in das Projekt eingebunden.

```
<script data-main="scripts/main" src="scripts/require.js"></script>
```

Das `src`-Attribut zeigt hier auf die Bibliothek selbst, unter `data-main` verbirgt sich die JavaScript-Datei, die zuerst geladen wird. Generell wird auf die Endung `.js` in RequireJS verzichtet.

Die Definition eines einfachen Moduls in einer Datei `Person.js` könnte wie folgt aussehen:

```
define(function(){
	function Person(name){
		this.name = name;

		this.getName = function(){
			return this.name;
		}
	}
	return Person;
});
```

Möchte man dieses Modul nun in einer anderen JavaScript-Datei verwenden, so genügt folgender Aufruf:

```
var Person = require('Person');
var person1 = new Person('Tom');
person1.getName(); // --> 'Tom'
```

Der Befehl `require()` benötigt hier den Pfad des zu ladenden Moduls – in diesem Fall liegt es auf der gleichen Ebene – und speichert dessen Return-Wert in der lokalen Variable `Person`. Über diese Variable ist der Code im Modul erreichbar.

5.6.1.3 Hilfsfunktionen – Underscore.js

Underscore.js [126] bietet ein sehr umfangreiches Repertoire an Funktionen, die man bei alltäglichen Aufgaben benötigt, aber jedes Mal umständlich von Hand programmieren müßte. Für Arrays gibt es beispielsweise Funktionen wie `first()`, `last()`, `min()` oder `max()`, für Objekte zahlreiche Vergleichs-Operatoren wie `isEmtpy()`, `isString()`, `isNan()` oder Funktionen, die das Objekt auslesen und bestimmte Werte in Arrays zurückgeben wie `keys()` oder `values()`.

Alle Funktionen werden über die Variable `_` erreicht. So würde man den Maximalwert eines Arrays wie folgt ermitteln:

```
var array = [1,2,3,4,5]
var maximum = _.max(array);
console.log(maximum); // --> 5
```

5.6.1.4 Lokales Event-Handling – Jvent.js

Während Sockets Events über das Netzwerk von Client zu Server und umgekehrt schicken, wird im Framework auch eine Klasse benötigt, die es möglich macht, Events lokal zu versenden und auf sie zu reagieren. So können Teile des Codes auf Aktionen anderer Abschnitte „hören“ und darauf reagieren.

Jvent.js [127] von Guille Paz macht es möglich einem Objekt mittels

```
var object = new JVent();
```

Methoden wie `on()` oder `emit()` zur Verfügung zu stellen, mit denen es auf Events hören oder solche abschicken kann.

Eine einfache Kette von Events wäre folgende:

```
var eventCenter = new Jvent();
eventCenter.on('weiterGehts', function(){
    console.log('Bar');
});
eventCenter.on('losGehts', function(){
    console.log('Foo -');
    eventCenter.emit('weiterGehts');
});
eventCenter.emit('losGehts');
```

Die letzte Zeile startet die Kette. Ausgabe: `Foo - Bar`.

5.6.2 Werkzeuge

5.6.2.1 require_config.js

Durch RequireJS wird festgelegt, dass `require_config.js` nach der RequireJS-Bibliothek selbst die erste JavaScript-Datei sein soll, die beim Seitenaufbau geladen wird.

```
<script data-main="require_config" src="libraries/require.js"></script>
```

Die `require_config.js` ist eine Konfigurationsdatei von RequireJS, in der Meta-Informationen definiert werden können, etwa den außerplanmäßigen Wunsch, Module auch

vor ihrer Verwendung zu laden oder Pfadangaben auf Module zu einem Wort zu verkürzen, wie man es in folgendem Code-Snippet sieht:

```
requirejs.config({
      paths: {
            'underscore' : 'libraries/underscore',
      }
});
```

Abgeschlossen wird die `require_config.js` im Framework mit dem Befehl:

```
requirejs(['client_app']);
```

Er legt fest, dass der Applikationsdurchlauf nun mit der `client_app.js` fortgesetzt wird. Weitere Module werden erst geladen, wenn sie mittels `require()` aufgerufen werden.

5.6.2.2 eventCenter.js

Die `eventCenter.js` hat den Hauptzweck, ein Objekt der Jvent-Klasse zu instanziieren, wie es schon in deren Vorstellung in Kapitel 5.6.1.4 gezeigt wurde. Mit

```
var eventCenter = new Jvent();
```

wird das Objekt `eventCenter` instanziiert, das im Folgenden für den Empfang und Versand lokaler Events zuständig ist. Ein `return eventCenter;` am Ende der Datei sorgt dafür, dass alle Module, die diese JavaScript-Datei per `require()` einbinden, exakt diese Instanz der Klasse geliefert bekommen.

Weiterhin wird hier die Socket-Kommunikation mit dem Server verwaltet. Alle User-Interaktionen, etwa Button-Klicks, senden in der Klasse des Objekts selbst über `eventCenter.emit()` lokale Events, die in der `eventCenter.js` abgehört werden.

Wurde eine Komponente auf der Zeichenfläche verändert, so sendet es beispielsweise ein `valueChanged`-Event, dass in der `eventCenter.js` wie folgt registriert und an den Server weitergeleitet wird:

```
eventCenter.on('valueChanged',function(data){
      socket.emit('userModel_changed_fCtS',data);
});
```

Gleiches passiert, wenn spezfische Werte geändert werden, wie im Stundenplan-Gestalter etwa eine Änderung des Hintergrunds oder das Ausblenden des Name/Klasse-Feldes.

5.6.2.3 GLOBALS.js

`GLOBALS.js` ist ein Modul, in welchem sich Objekte befinden, die den Charakter einer globalen Variable besitzen sollen.

Sie sind nicht wirklich global erreichbar, da sie in ein RequireJS-Modul eingeschlossen sind, dennoch kann man mit `require()` die `GLOBALS.js` in jedes gewünschte Modul einbinden und exakt auf die Objekte zugreifen, die man dort abgelegt hat.

Neben den States liegt dort beispielsweise auch der URL-Parameter als String, der jedem `changed`-Event an den Server mitgegeben wird.

Statt eine Klasse zu definieren, liegt in der GLOBALS.js ein einfaches Objekt mit Subobjekten, die dann die „globalen Variablen“ darstellen.

```
define(function(){
	var GLOBALS = {
		url : '1234',
	};
	return GLOBALS;
});
```

Dadurch, dass das Wrapper-Objekt `GLOBALS` per `return`-Statement zurückgegeben wird, kann jedes Subobjekt nach der Einbindung der JavaScript-Datei mittels `require()` einfach per Punktschreibweise auf die Variablen zugreifen. So wäre `GLOBALS.url` beispielsweise der Aufruf, um an den URL-Parameter heranzukommen.

Wie der URL-Parameter werden alle GLOBALS-Variablen erst zur Laufzeit hinzugefügt, da sie abhängig von den spezifischen Gegebenheiten sind, wie etwa der URL, die dem Client zugewiesen wird.

5.6.2.4 helpers.js

Die `helpers.js` funktioniert ähnlich wie die Underscore.js-Bibliothek und bietet selbstgeschriebene Hilfs-Funktionen für allgemeine Probleme.

Exakt wie die GLOBALS.js besitzt sie ein umgebendes Objekt, hier `helperFunctions` genannt, das retourniert wird. Statt Variablen beinhaltet es aber komplette Funktions-Algorithmen. Im Framework wird beispielsweise an mehreren Stellen eine Funktion benötigt, um eine UUID zu berechnen. Dazu wurde in der `helpers.js` folgende Funktion angelegt:

```
var helperFunctions = {
	generateUUID : function(){
		var d = Date.now();
		var uuid = 'xxxxxxxx-xxxx-4xxx-yxxx-xxxxxxxxxxxx'
			.replace(/[xy]/g, function(c) {
				var r = (d + Math.random()*16)%16 | 0;
				d = Math.floor(d/16);
				return (c=='x' ? r : (r&0x7|0x8)).toString(16);
			});
		return uuid;
	},
};
```

Sie wird mit `helpers.generateUUID()` aufgerufen und gibt eine Standard-konforme UUID zurück, die mithilfe der aktuellen Zeit und mathematischen Operationen einen möglichst zufälligen Wert annimmt.

5.6.3 Subklassen

Die folgenden Unterkapitel erläutern die Subklassen, die der Nutzer des Frameworks selbst implementieren muss. Sie werden auch *Komponenten* genannt.

Komponenten besitzen im Application-Model ein Attribut `type`, anhand dessen der Algorithmus erkennt, welche Klasse er für die Instanziierung laden muss.

Der Prozess der Instanziierung ist wichtig für das Verständnis der Komponenten, darum wird dieser in folgendem Schema aufgezeigt:

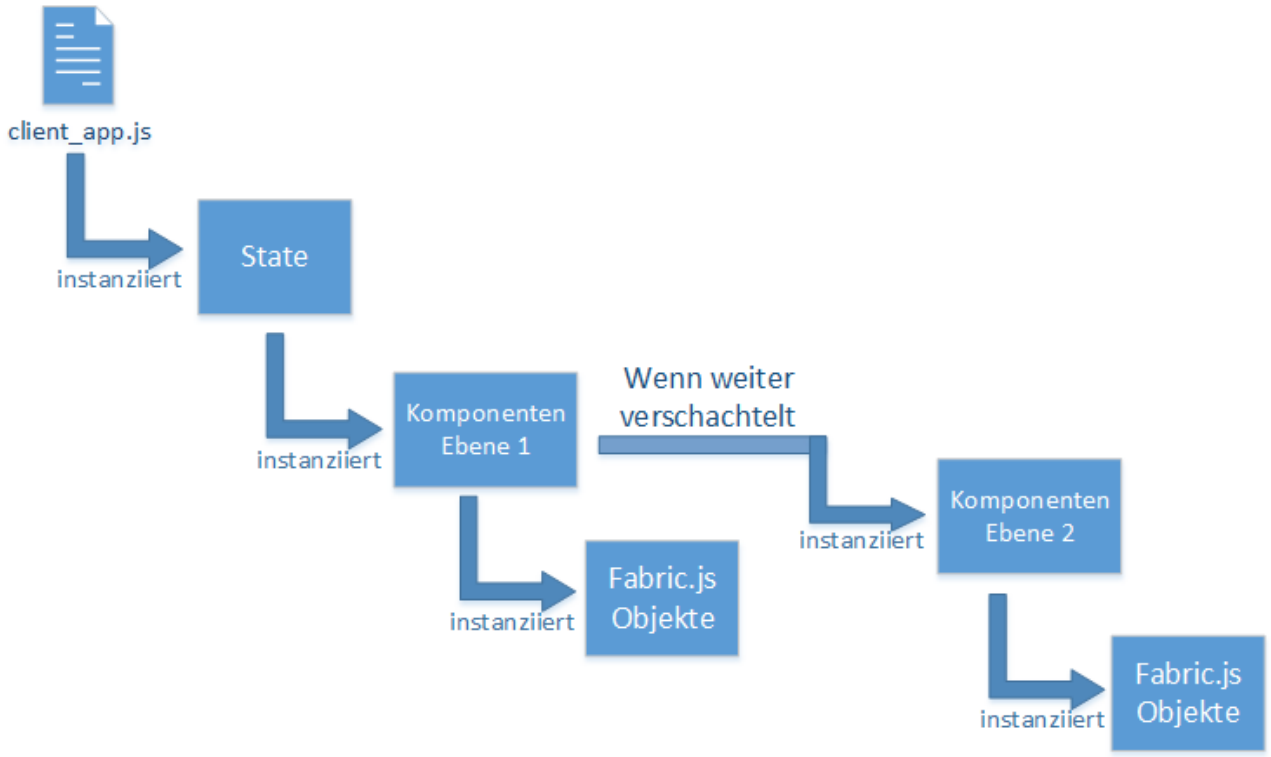

Abbildung 70: Prozesskette der Instanziierungen

In der `client_app.js` werden alle States aus dem Application-Model geladen und als Instanz der Klasse State `angelegt`. Die Klasse `State` instanziiert alle Komponenten, die in der nächsten Ebene liegen. Sollte der `type` auf eine Klasse verweisen, die weitere Unterkomponenten besitzt, wie etwa die `VirtualGroup`, so wird die Instanz weitere Unterkomponenten instanziieren. Am Ende jeder Kette werden die für die Komponente benötigen Fabric.js-Objekte instanziiert.

5.6.3.1 Komponenten

Im Ordner `components` stecken die Herzstücke der Applikation: Die Komponenten mitsamt ihrer Funktionalität.

Sie sind anwendungsspezifisch, das heißt, auch in diesem Kapitel wird konkret mit der Beispiel-Applikation des Stundenplan-Gestalters gearbeitet.

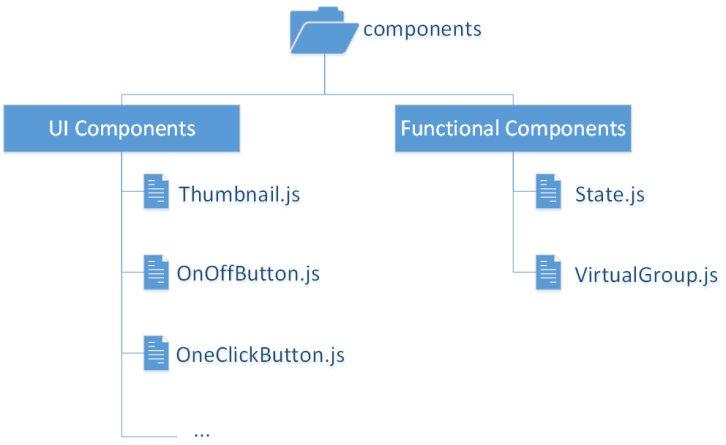

Abbildung 71: Einteilung der Komponenten beim Stundenplan-Gestalter

Der Prototyp des Stundenplan-Gestalters besteht aus dreizehn Komponenten. Elf dieser Komponenten haben gestalterischen Charakter. Sie nutzen die Fabric.js-Bibliothek, um UI-Elemente wie Buttons oder Schalter zu gestalten und versehen diese mit Logik. Dazu kommt als elfte Komponente die Zeichenfläche selbst, die bis auf die Darstellung des Wunschhintergrundes nicht direkt mit dem Anwender in Kontakt tritt.

Komponente zwölf und dreizehn sind nicht sichtbar auf dem Canvas. Sie werden funktionelle Komponenten genannt. Eine dieser beiden dient der Positionierung von anderen Komponenten und die andere dem Controlling des Datenmodells.

Analog zur Abbildung 64 auf Seite 77 wird hier gezeigt, welche Komponente für welches Element in der Hauptapplikation in State 1 eingesetzt wird.

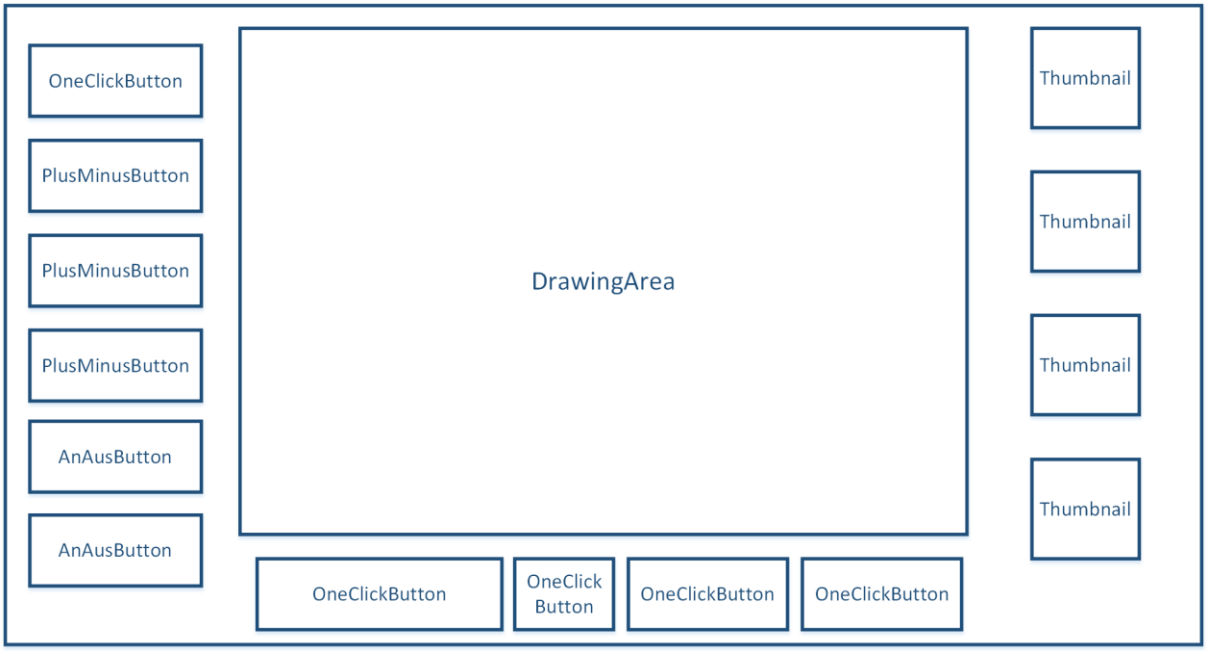

Abbildung 72: Schematische Zeichnung über Verwendung der Komponenten

Im Folgenden werden exemplarisch jeweils zwei der User-Interface-Komponenten und der funktionalen Komponenten vorgestellt.

5.6.3.1.1 UI-Komponente – OneClickButton

Ein klassischer Vertreter der UI-Komponenten ist der „OneClickButton“. Die Bezeichnung steht für ein einfaches, klickbares Feld wie das HTML-`<button>`-Element.

An dieser Stelle wird der Bogen zwischen Application-Model und Komponenten geschlagen. Die Farbe, die Größe und die Position der einzelnen Komponenten sind natürlich nicht fest im Code verankert, sondern werden durch den Entwickler im Application-Model festgelegt. Folgend sehen wir einen Auszug aus dem Application-Model, der das Erstellen eines OneClickButtons zur Folge hat:

```
"element01" : {
      "type" : "OneClickButton",
      "name" : "dummyButton",
      "top" : 10,
      "left" : 10,
      "width" : 110,
      "height" : 30,
      "fill" : "rgb(154,205,50)",
      "heading" : "Klick mich!"
}
```

Dieses JSON-Objekt wird an den Konstruktor der `OneClickButton`-Klasse in der OneClickButton.js-Datei weitergegeben, der es als `element` übernimmt und daraus Fabric.js-Objekte instanziiert.

```
function OneClickButton(element){

	this.element = element;

	var rect = new fabric.Rect({
		width: this.element.width,
		height: this.element.height,
		left: this.element.left,
		top: this.element.top,
		fill: this.element.fill
	});

	var heading = new fabric.Text(this.element.heading, {
		top: this.element.top,
		left: this.element.left
	});
}
```

Es lässt sich erkennen, wie die Daten aus dem JSON-Objekt des Application-Models ausgelesen und als Werte für ein `fabric.Rect`, also ein Rechteck, und einen `fabric.Text`, also einen einfachen Textbaustein verwendet werden.

Auch nachfolgende Codeabschnitte befindet sich im Gültigkeitsbereich des Konstruktors, werden aber aus Gründen der Übersichtlichkeit separat erläutert.

Nachdem `rect` und `heading` instanziiert wurden, werden sie in einem Array gespeichert.

```
this.componentArray = [];
this.componentArray.push(rect, heading);
```

Dieser Array `componentArray` wird verwendet, um die Objekte später auf dem Canvas sichtbar zu machen. Er ist allen Komponenten gemein und wird im Kapitel 5.6.4 näher beschrieben.

Zum Abschluss wird im Konstruktor das Klick-Event auf dem Button registriert. Anhand des Namens, den man im Application-Model definiert hat, differenziert der EventListener bei mehreren OneClickButtons, welche Aktion bei welchem Button durchgeführt wird.

```
rect.on('mousedown', _.bind(function(e){
	if(this.element.name === 'dummyButton'){
		alert(this.element.name + ' geklickt!');
	}
}, this));
```

Hier lässt sich die Verwendung der Underscore.js-Bibliothek beobachten. Mittels `_.bind()` wird der `this`-Kontext von außen (`element`) in den EventHandler mit hineingenommen, wo er normalerweise von dem Event `e` abgelöst worden wäre. So lässt sich auch innerhalb der `on()`-Funktion mit `this.element` auf das JSON-Objekt zugreifen.

5.6.3.1.2 UI-Komponente – DrawingArea

Als weiteres Beispiel für UI-Kompontenen wird die `DrawingArea` vorgestellt. Die Klasse wird in der `DrawingArea.js` definiert und beschreibt die Zeichenfläche, auf der der Stundenplan gestaltet wird.

Sie hat zwei Basis-Funktionalitäten: Bei einem Klick auf ein entsprechendes Thumbnail ändert sich ihr Hintergrund. Außerdem muss sie erkennen, wann ein Icon die Zeichenfläche verlässt und gelöscht werden muss.

Zunächst wird auch hier wie beim OneClickButton ein Rechteck in der Größe der Komponente gezeichnet, außerdem wird es dem Klassen-eigenen `componentArray` hinzugefügt:

```
function DrawingArea(element){

	this.element = element;

	var rect = new fabric.Rect({
		width: this.element.width,
		height: this.element.height,
		left: this.element.left,
		top: this.element.top
	});

	this.componentArray = [];
	this.componentArray.push(rect);
}
```

Das Besondere an dieser Komponente steckt in ihrem EventHandling. Wählt jemand einen neuen Hintergrund mittels Thumbnail aus, wird ein Event im `eventCenter` ausgelöst. Dem Event wird als Datenpaket der Pfad zum Bild in Originalgröße mitgegeben.

Die `eventCenter.js` wurde vorher mit RequireJS eingebunden, daher kann auch hier auf das Event gehört werden:

```
eventCenter.on('background_changed', function(data){
	fabric.util.loadImage(data.pathToImage, function (img) {
		rect.setPatternFill({ source: img, repeat: 'no-repeat' });
	});
});
```

Bei Eintreffen des Events `background_changed` wird der ImageLoader von Fabric.js angestoßen. Hat er das entsprechende Bild über den Pfad aus `data` geladen, so kann im Callback der Funktion mit der Variable `img` weitergearbeitet werden. Mit der Funktion `setPatternFill()` wird der Hintergrund des vorher definierten `rect` auf das gewünschte Bild gesetzt.

Eine weitere Eigenschaft der DrawingArea ist das Löschen von Icons, wenn diese außerhalb des Zeichenbereichs geschoben werden. Dazu wird im EventCenter registriert, sobald ein Icon losgelassen wurde. Im EventCenter wird daraufhin ein Event `stageObject_mouseup` ausgelöst, was lokal in der Klasse `DrawingArea` verarbeitet wird. Dem Event wird per Datenpaket das betreffende Icon mitgegeben:

```
eventCenter.on('stageObject_mouseup', function(icon){
	if(	icon.left < rect.left ||
		icon.left > rect.left + rect.width ||
		icon.top < rect.top ||
		icon.top > rect.top + rect.height){
		icon.remove();
	}
});
```

Diese if-Anweisung hat eine Bedingung, die abhängig von der Position der DrawingArea-Komponente erkennt, ob sich das Icon beim Loslassen der Maustaste außerhalb des

Zeichenbereichs befunden hat. Nachstehende Grafik verdeutlicht das. Auf der blau schraffierten Fläche soll das Icon gelöscht werden.

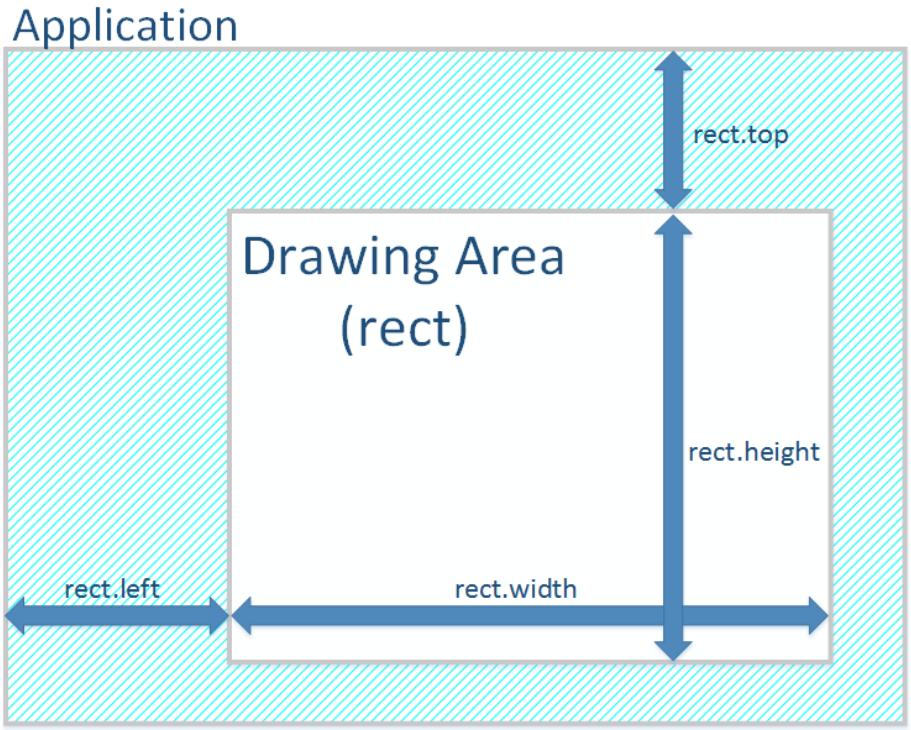

Abbildung 73: Geometrie der DrawingArea

Der erste Teil der if-Bedingung überprüft, ob das Element kleiner ist als der `left`-Wert der Zeichenfläche. Ist dies der Fall, so wird das Icon mit `remove()` vom Canvas entfernt. Gleiches gilt für den `top`-Wert. Für die Flächen rechts und unterhalb der Zeichenfläche werden zu den `top`- und `left`-Werten jeweils noch die `width`- und `height`-Werte addiert.

Mit OneClickButton und DrawingArea wurden zwei UI-Komponenten des Stundenplan-Gestalters im Detail vorgestellt. Die anderen neun UI-Komponenten sind ihnen ähnlich aufgebaut und unterscheiden sich im Wesentlichen durch die Art, Anzahl und Gestaltung der Fabric.js-Objekte, wodurch das EventHandling und die Logik, die damit umgesetzt wird, variiert werden. Sie wurden als Area, CopyField, EditText, LinkToCopy, NameClassField, OnOffButton, PlainText, PlusMinusButton und Thumbnail bezeichnet.

5.6.3.1.3 Funktionale Komponente – State

Im Bereich der funktionalen Komponenten besitzt der Stundenplan-Gestalter zwei: den `State` und die `VirtualGroup`.

Die Klasse State erhält bei Instanziierung in der `client_app.js` ein JSON-State-Objekt aus dem Application-Model.

Der Konstruktor speichert die Schlüssel und Werte des JSON-Objekts in der Instanz der State-Klasse. Außerdem wird die Nummer des States gespeichert.

```
function State(statenum, stateObj){

      this.statenum = statenum;
      this.elements = stateObj;
```

```
        this.componentArray = [];

        for (var prop in this.elements){
              var propObj = this.elements[prop];
              if(propObj.type){
                    var Subclass = require('subclasses/components/'
                                                              + propObj.type);
                    propObj.state = this;
                    var elem = new Subclass(propObj);
                    this.componentArray.push(elem);
              }
        }
}
```

Die for-Schleife des Codes sorgt für die Instanziierung der Komponenten auf erster Ebene. Die Struktur eines States lässt sich in Abbildung 52 auf Seite 66 nachsehen.

UI-Komponenten besitzen einen Array `componentArray`, in den sie die erzeugten Fabric.js Objekte legen. Auch der `State` hat einen `componentArray`. Allerdings werden hier keine Fabric.js-Objekte instanziiert, stattdessen werden dort die Komponenten hineingespeichert, nachdem sie dynamisch mittels `require()` instanziiert wurden.

Die Wahl der richtigen Klasse für die UI-Komponente erkennt RequireJS an deren Attribut `type`, das genauso benannt ist wie die Datei, die RequireJS einbinden muss. Dieser Code wird in einer Schleife für jedes Subobjekt durchgeführt, das ein `type`-Attribut besitzt.

Die Bedeutung des `componentArray` wird wie auch bei den UI-Komponenten im kommenden Kapitel 5.6.4 erläutert, wenn der Programmablauf exemplarisch erklärt wird.

5.6.3.1.4 Funktionale Komponente – VirtualGroup

Als zweite funktionale Komponente wird die VirtualGroup-Komponente vorgestellt.

Die `VirtualGroup` dient der Positionierung von Elementen. Zwar bietet auch Fabric.js selbst eine Klasse zur Gruppierung, die `fabric.Group`, allerdings ist dort das EventHandling noch nicht ausgereift beziehungsweise komplett unterbunden. Würde man mit ihr mehrere Buttons gruppieren, so würde nur noch ein einziger ClickHandler, nämlich der der Gruppe selbst, aktiviert werden können. Zwischen einzelnen Buttons wird nicht mehr differenziert.

Weiterhin bietet eine Fabric.js-Gruppe keine echte Verschachtelung. Es ist eine „Verschmelzung“ der gruppierten Elemente zu einem Eltern-Element, eine Hierarchie ist nicht gegeben.

Verzichten muss man bei der VirtualGroup allerdings vorerst auf das gemeinsame Verschieben aller gruppierten Elemente, was bei der `fabric.Group` möglich ist. Dies wird in einer späteren Version des Frameworks implementiert und ist nicht Teil des Prototyps.

Vermeidet man die Verwendung der `fabric.Group`, müsste man theoretisch jeder UI-Komponente eine absolute Positionsangabe mitgeben, die ausgehend vom Canvas-Ursprung oben links angibt, wie weit sich das Element mit `top`- und `left`-Werten davon entfernt.

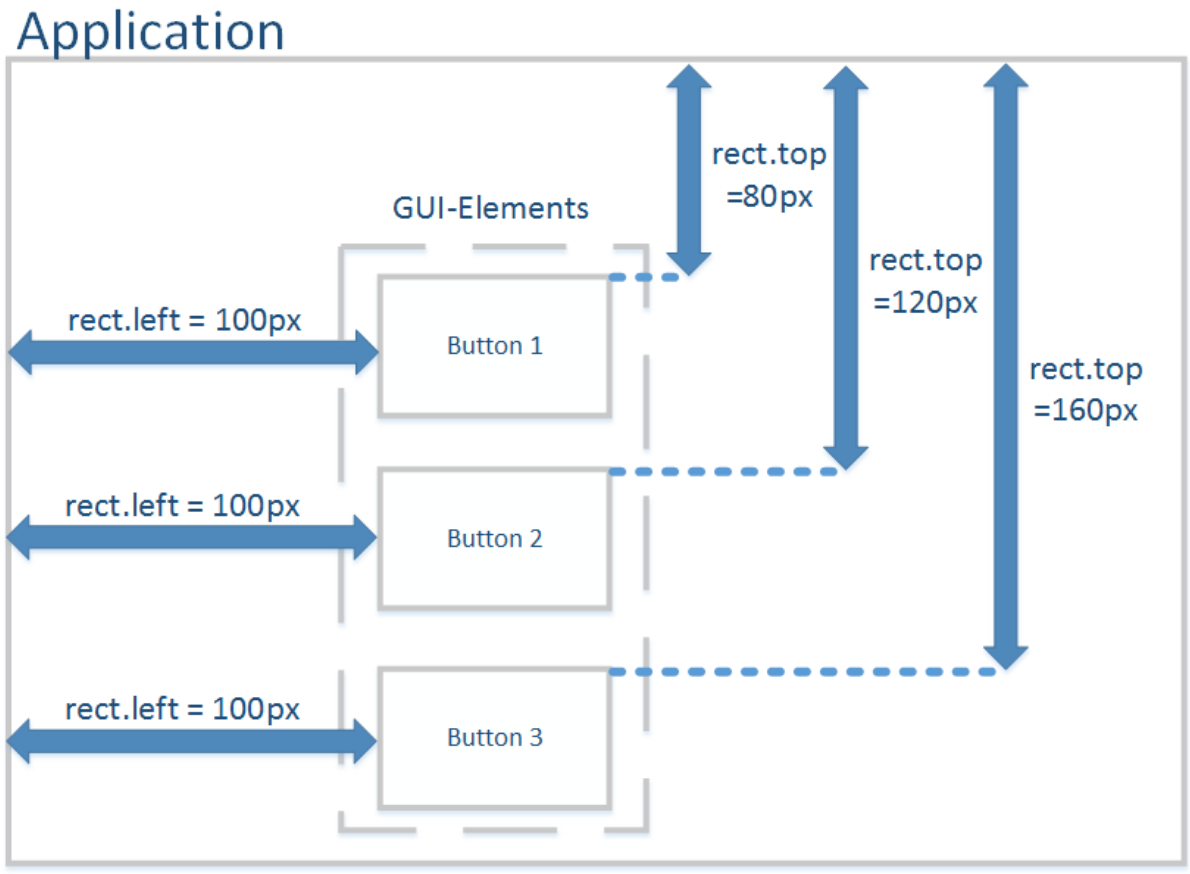

Abbildung 74: Absolute Positionierung von GUI-Elementen trotz Gruppierungswunsch

Da das nicht praktikabel ist, wurde eine eigene Gruppierungs-Klasse geschrieben, die als Abgrenzung zur `fabric.Group` den Namen `VirtualGroup` bekam. Sie tut nichts weiter, als Komponenten, wie beispielsweise dem OneClickButton, vor deren Instanziierung anhand ihrer eigenen Position berechnete `left`- und `top`-Werte mitzugeben. Damit beeinflusst sie beispielsweise das EventHandling nicht.

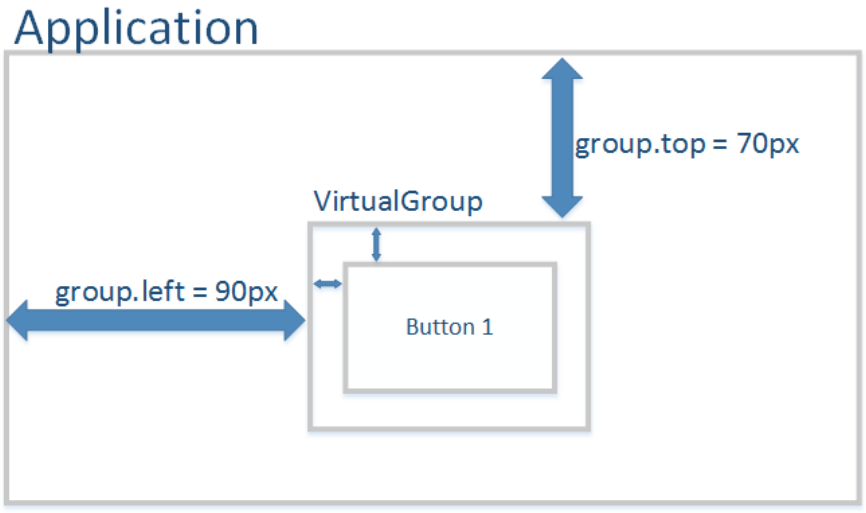

Abbildung 75: Relative Positionierung von GUI-Elementen innerhalb einer VirtualGroup

Im Prototyp des Stundenplan-Gestalters kann man mit der Angabe `manually`, `grid`, `horizontal` oder `vertical` die Ausrichtung der Gruppierung bestimmen. Bei `manually`

kann man die Position der Komponenten weiterhin mit eigenen `top`- und `left`-Werten bestimmen, diese sind aber nun relativ zur Position der Gruppe. Mit `grid` ist eine 2-dimensionale Matrix aus Komponenten gemeint. Es muss zusätzlich die Anzahl der Spalten und Reihen angegeben werden. Bei den Modi `horizontal` beziehungsweise `vertical` werden die Komponenten in immer gleichem Abstand hintereinander beziehungsweise untereinander angereiht.

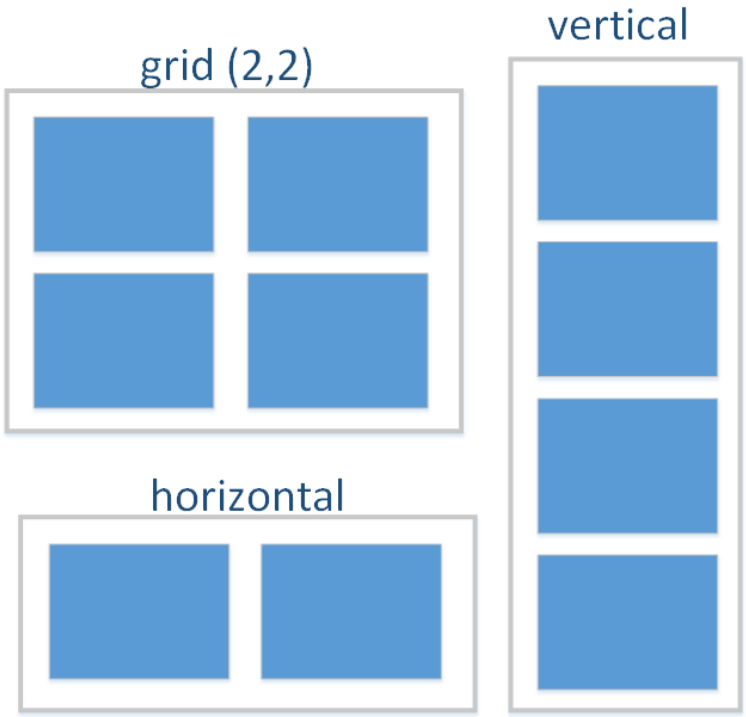

Abbildung 76: Die drei verschiedenen Modi einer VirtualGroup

Ein Auszug aus dem Application-Model zeigt folgend die JSON-Struktur einer VirtualGroup mit zwei Komponenten:

```
"navi_left" : {
      "type" : "VirtualGroup",
      "width": 200,
      "height": 200,
      "left": 20,
      "top": 20,
      "fill": "rgb(244,236,220)",
      "config" : {
            "align" : "vertical"
      },
      "fillings" : {
            "el01" : {
                  "type" : "OneClickButton",
                  "name" : "goToBackgrounds",
                  "width" : 100,
                  "height" : 70,
                  "fill" : "rgb(255,250,177)",
                  "heading" : "Hintergrund"
            },
            "el02" : {
                  "type" : "PlusMinusButton",
                  "name" : "changeSize",
                  "width" : 100,
```

```
                "height" : 70,
                "fill" : "rgb(243,188,245)",
                "heading" : "Größe",
                "option_left" : "-",
                "option_right" : "+"
            }
        }
}
```

Dieses Objekt wird der Klasse `VirtualGroup` übergeben. Als Ausrichtung wurde `vertical` gewählt.

Zunächst wird jede Komponente, die in dem Subobjekt `fillings` steht, in ein Array namens `naviElements` geschoben:

```
function VirtualGroup(element){

      var groupElement = element;
      var naviElements = [];

      for (var prop in groupElement.fillings) {
            naviElements.push(groupElement.fillings[prop]);
      }
      ...
}
```

Danach wird festgestellt, welche Gesamthöhe die Komponenten gemeinsam haben:

```
var sumHeightOfElements = 0;

for(var i = 0; i < naviElements.length; i++){
      sumHeightOfElements += naviElements[i].height;
}
```

Damit kann die Lücke zwischen den einzelnen Elementen berechnet werden. Sie soll immer gleich bleiben.

```
var gapVertical = 0;
gapVertical =     (groupElement.height - sumHeightOfElements)
                  / (naviElements.length +1);
```

Hier wird es nun schon komplexer. Was man mit diesen mathematischen Operationen erreicht, zeigt folgende Grafik:

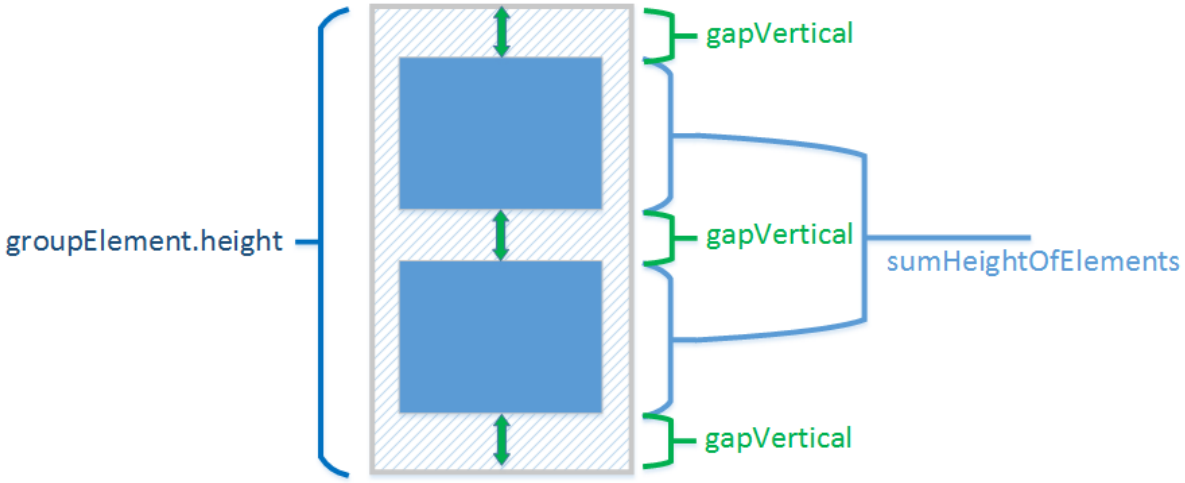

Abbildung 77: Positionierung innerhalb einer VirtualGroup

Von der Gesamthöhe der Gruppe wird die Gesamthöhe der Komponenten abgezogen. Als Differenz bleibt die Gesamthöhe des schraffierten Bereichs. Nun muss dieser Bereich durch die Anzahl der Komponenten plus eins geteilt werden (bei zwei Komponenten gibt es drei „gaps“) um die Höhe einer einzelnen Lücke zu ermitteln.

Zum Abschluss werden die `top`-Werte der einzelnen Komponenten mithilfe der vorher berechneten `gapVertical` ermittelt. Die `left`-Werte werden in einer vertikalen Gruppierung immer gleich bleiben.

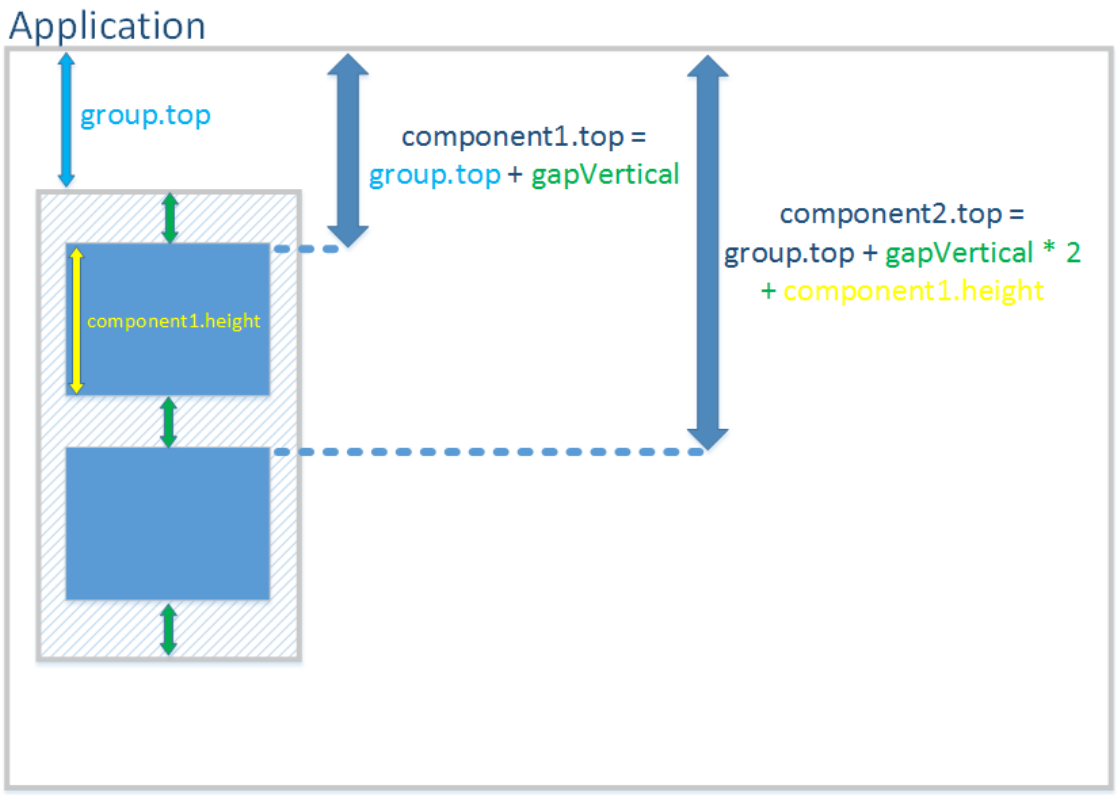

Abbildung 78: Berechnung der einzelnen top-Werte

Der `top`-Wert der ersten Komponente wird durch die Addition des `top`-Werts der Gruppe und der `gapVertical` ermittelt.

Beim top-Wert der zweiten Komponente wird zum top-Wert der Gruppe die `gapVertical` zwei Mal hinzugefügt, außerdem wird die Höhe des vorher platzierten Elements berücksichtigt.

Der Algorithmus dazu würde den Rahmen dieses Kapitels überschreiten, da er diese Additionen in Schleifen relativ zur Anzahl der Komponenten pro Gruppe berechnet. Er kann der CD entnommen werden.

Auch die `VirtualGroup` hat einen `componentArray`. Wie im `State` werden hier die Komponenten hineingespeichert, nachdem sie instanziiert wurden. Damit ist eine VirtualGroup eine Komponente erster Ebene, die weitere Unterkomponenten besitzt und diese eigenständig instanziiert, wie auf Abbildung 70, S.91 rechts zu sehen.

```
this.componentArray = [];

var Subclass = require('subclasses/components/' + naviElement.type);

var elem = new Subclass(naviElement);
this.componentArray.push(elem);
```

Dieser Code wird in einer Schleife für jede Komponente durchgeführt, die Teil der Gruppe ist.

Zusammenfassend gibt es drei Schritte, die jede gruppierte UI-Komponente durchmacht:

1. Die `top`- und `left`-Werte der Komponente werden nach Wahl der Ausrichtung festgelegt.
2. Die Komponente wird instanziiert.
3. Die Instanz wird im `componentArray` gespeichert.

5.6.3.2 Fabric.js Subklassen

Fabric.js bietet zwar einen großen Funktionsumfang, allerdings ist es nicht darauf ausgelegt, einzelne Objekte auf dem Canvas programmatisch zu adressieren. Instanziierte Objekte werden zwar in einem Array `canvas._objects` abgelegt, sobald sie dem Canvas mit `canvas.add()` hinzugefügt wurden, aber dort erhalten sie nicht mehr als einen Array-Index, der in der Reihenfolge ihrer Hinzufügung hochzählt.

Um Änderungen an einzelnen Objekten zielgerichtet im User-Model speichern zu können und an andere Clients zu übertragen, bedarf es der Erweiterung einiger Fabric.js-Klassen.

Die Erweiterung der Fabric.js-Klassen richtet sich nach dem Anwendungsfall des Frameworks, daher muss hier ein konkreter Fall demonstriert werden. Im Stundenplan-Gestalter wird mit herumschiebbaren Icons und Freitext gearbeitet, daher müssen die Klassen `fabric.Image` und `fabric.IText` erweitert werden.

Beide Klassen benötigen eine Version, in der jede Instanz einen eindeutigen Bezeichner erhält. Die beiden Subklassen nennen sich `fabric.ImageWithID` und `fabric.ITextWithID`.

Von der Klasse `fabric.Image` erbt außerdem eine weitere Klasse `fabric.ThumbnailFromImage`, die spezielle Attribute zur Funktionalität eines Thumbnails besitzt.

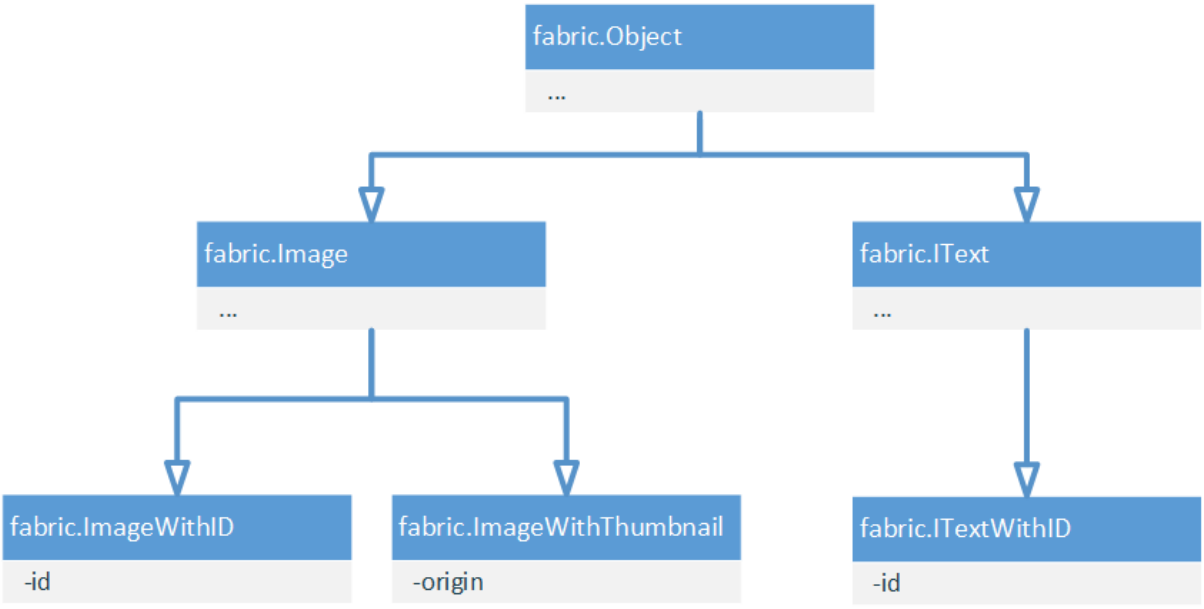

Abbildung 79: Vererbung der selbsterstellten Fabric.js-Klassen im UML-Diagramm

Fabric.js liefert eine interne Funktion zur Erstellung von Klassen:

```
fabric.util.createClass(superClass, customizeObj);
```

Eine Klasse `ImageWithID` lässt sich somit mit folgendem Code realisieren:

```
fabric.ImageWithID = fabric.util.createClass(fabric.Image, {
      initialize: function(element, options) {
            this.callSuper('initialize', element, options);
            options && this.set('id', options.id);
      }
});
```

Im Aufruf von `createClass()` übergibt man als erstes Argument die Superklasse `fabric.Image`. Als zweites Argument folgt ein Objekt, in dem die Subklasse definiert wird.

Die `initialize()`-Funktion darin ist zuständig für Instanziierungen neuer Objekte mit dem `new()`-Operator. Hier wird zunächst die Funktionalität der Superklasse mit `callSuper()` geerbt. Danach wird ein Attribut `id` gesetzt, das aus dem übergebenen `options.id` ausgelesen wird.

Ein Bild mit ID kann demnach zum Beispiel wie folgt aus einem HTML-`<img>` instanziiert werden:

```
var imgElement = document.getElementById('my-img');
var imgInstance = new fabric.ImageWithID(imgElement, {
      left: 100,
      top: 100,
      id: '1s7xr3'
});
canvas.add(imgInstance);
```

Nach diesem Schema wurden alle drei Fabric.js-Subklassen erstellt.

Weiterhin werden auch hier noch ein paar spezielle EventHandler registriert. In Kapitel 5.6.2.2 sieht man ein Beispiel für einen EventListener auf geänderte Werte auf der Zeichenfläche. Die Klassen `ImageWithID` und `ITextWithID` sind die Icons, mit denen man die Zeichenfläche gestalten kann. Sie müssen Änderungen sofort an das `eventCenter` weiterschicken, damit dieses die Änderungen per Socket an den Server übertragen kann.

Derlei Ereignisse werden wie folgt abgehandelt:

```
this.on('rotating', function(e){
      eventCenter.emit('valueChanged', {
            planURL: GLOBALS.url,
            id: this.id,
            values:{angle: this.getAngle()}});
});
```

Gleiches passiert mit den Events `moving` und `scaling`. Neben dem geänderten Wert werden der URL-Parameter und die ID des Elements mitgeschickt, damit bei Speicherung und Synchronisation alles richtig zugeordnet werden kann.

Auf ein weiteres Event sollte noch aufmerksam gemacht werden. In Kapitel 5.6.3.1.2 erkennt man, wie die Icons auf der DrawingArea gelöscht werden. Sie übermitteln bei jedem `mouseup` ihre Position dorthin, indem sie sich selbst mitschicken. Auch dies wird hier implementiert:

```
this.on('mouseup', _.bind(function(e){
      eventCenter.emit('stageObject_mouseup', this);
}, this));
```

5.6.4 Programmablauf und client_app.js

In der Konfigurationsdatei von RequireJS ist festgelegt, dass nach dem Download der clientseitigen Dateien als erstes JavaScript die Datei `client_app.js` ausgeführt wird.

Die `client_app.js` beginnt damit, eine Verbindung zum Socket-Server aufzubauen:

```
var socket = io.connect();
```

Ist die Verbindung aufgebaut, wartet der Socket mit folgender Listener-Funktion auf ein Ereignis, wie es in Kapitel 5.5.3 zur Übermittlung des Application-Models definiert wurde:

```
socket.on('send_ApplicationModel_fStC', function(data){
      var states = data;

      var i = 0;
      for(var prop in states){
            var state = new State(i,states[prop]);
            GLOBALS['state'+i] = state;
            i++;
      }
      eventCenter.emit('renderState', 0);
});
```

Hier wird das Application-Model in eine Variable `states` geschrieben. Daraufhin wird mit einer `for in`-Schleife jede Objekteigenschaft auf erster Ebene durchlaufen. Die Objekteigenschaften auf erster Ebene repräsentieren die einzelnen States, wie man auch in Abbildung 52 auf Seite 66 feststellen kann.
Im ersten Teil der Schleife wird jeder State als Instanz der Klasse `State` instanziiert. Im zweiten Schritt wird diese Instanz als Objekteigenschaft mit Nummerierung in das GLOBALS-Modul hineingeschrieben.
Die Instanziierung der State-Objekte löst eine Kettenreaktion aus, wie sie in Abbildung 70 auf Seite 91 zu sehen ist. Im Konstruktor der State-Klasse werden alle tiefer verschachtelten Komponenten bis zum granularsten Punkt, dem Fabric.js-Objekt, instanziiert und in das jeweilige `componentArray` hineingeschrieben.

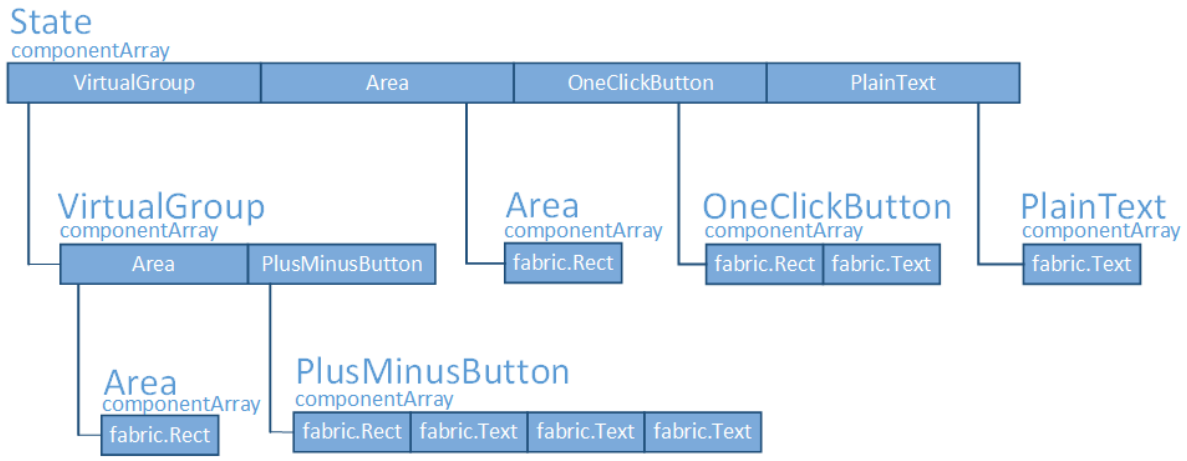

Abbildung 80: Beispielhafte Befüllung der componentArrays

Sind alle States und deren Komponenten instanziiert, so kann fortan an jeder Stelle jeder State mit der Methode `renderState()` auf dem Canvas dargestellt werden.

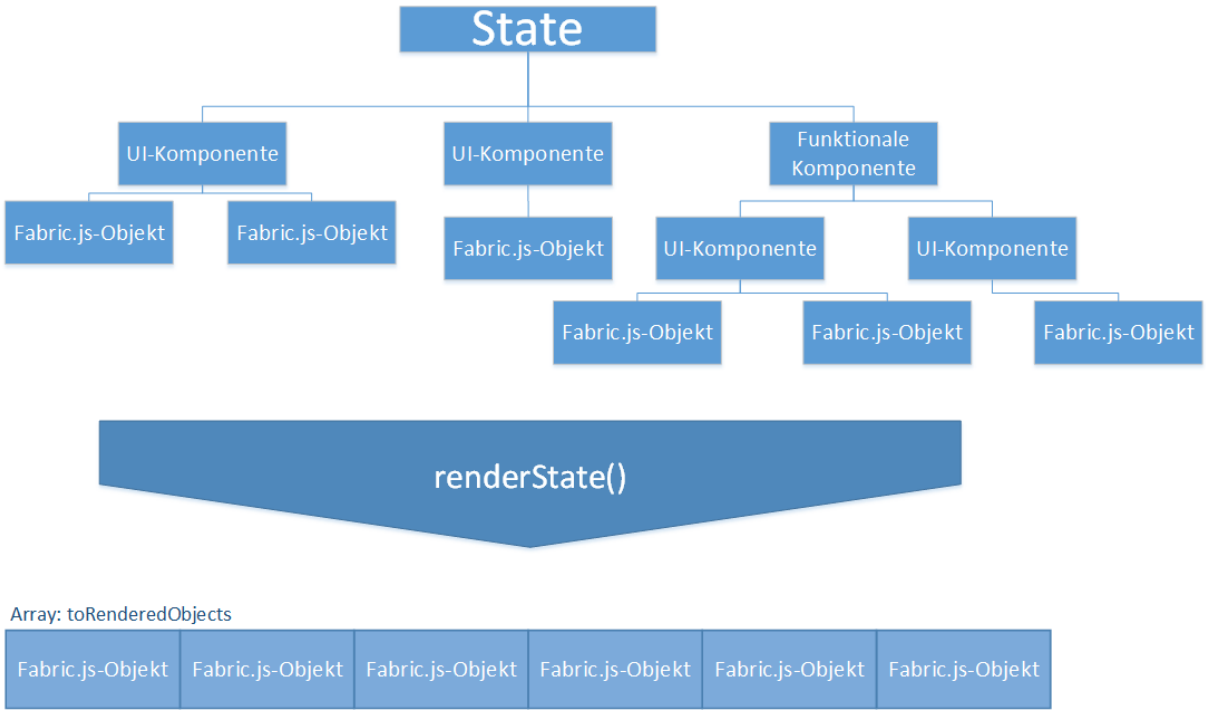

Abbildung 81: Funktionsweise der renderState()-Funktion

Die Funktion `renderState()` speichert alle Fabric.js-Objekte der Komponenten in einem Array `toRenderedObjects`, welches dann mithilfe einer Schleife auf dem Canvas dargestellt wird. Zuvor wird das Canvas noch geleert:

```
canvas.clear();
for(var fabricObject in toRenderedObjects){
      canvas.add(toRenderedObjects[fabricObject]);
}
```

Bei der Entstehung des Arrays `toRenderedObjects` kommt der `componentArray` einer jeden Komponente zum Zug. `renderState()` löst ebenfalls eine Kettenreaktion aus. Jede Komponente verfügt über eine Methode `getRenderObjects()`, die das `componentArray` zurückgibt. Dadurch, dass man sie in den `prototype` der einzelnen Komponenten schreibt, kann die Funktion auf jeder Instanz aufgerufen werden.

```
Komponente.prototype.getRenderObjects = function(){
      return this.componentArray;
};
```

`renderState()` ruft diese Methode rekursiv bis in die tiefste Ebene der State-Hierarchie für jede Komponente auf und erhält so als Rückgabewert alle Fabric.js-Objekt-Arrays.

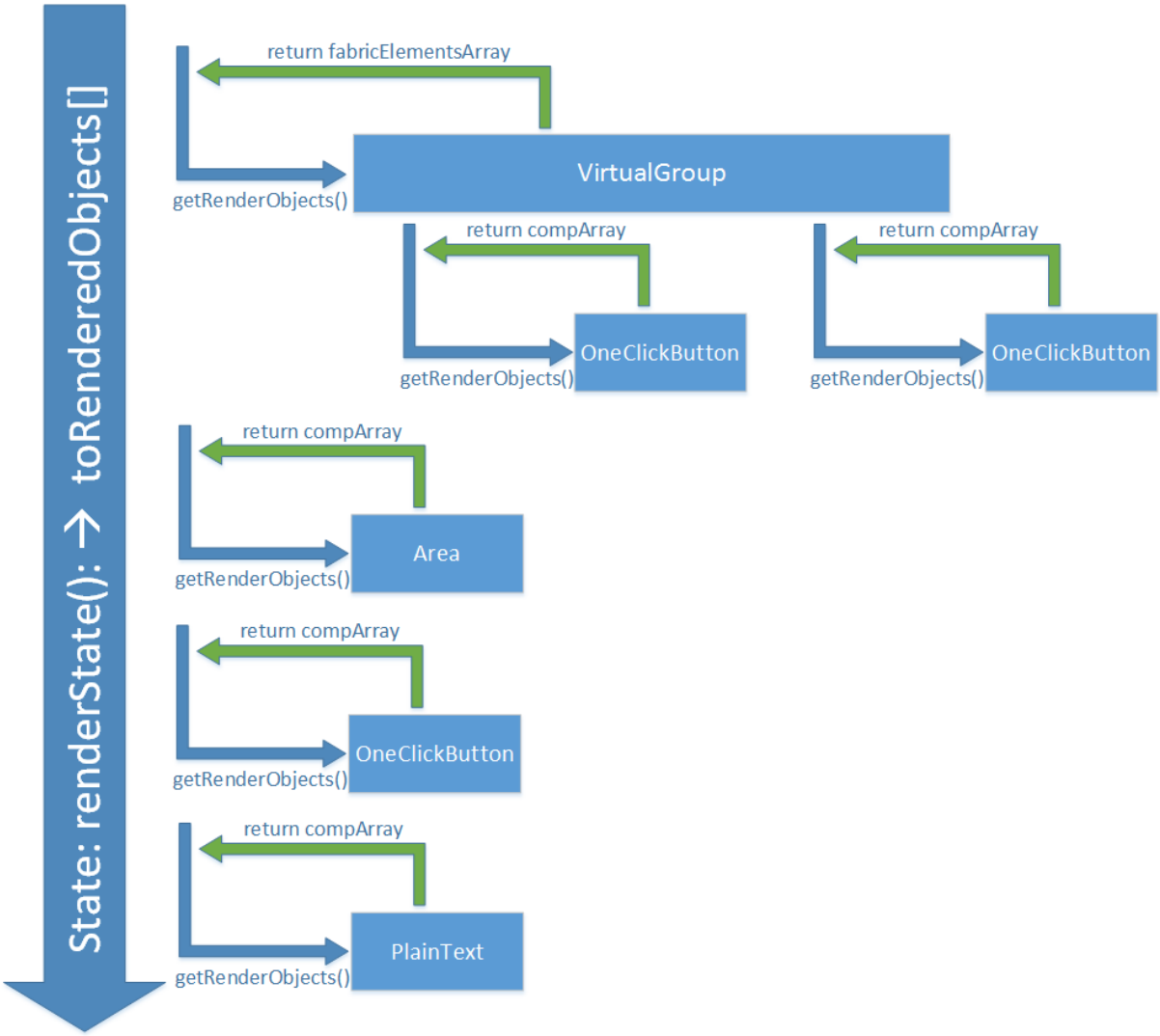

Abbildung 82: Die Kettenreaktion der renderState()-Methode

UI-Komponenten in der ersten Ebene können ihr `componentArray` direkt zurückgeben, da es ihre fabric.js-Objekte enthält (siehe Abbildung 80, Seite 103, `Area`, `OneClickButton` und `PlainText`).

Funktionale Komponenten, die ihrerseits UI-Komponenten besitzen (siehe Abbildung 80, `VirtualGroup`), erwarten die `componentArray`s ihrer Subkomponenten und speichern deren Inhalte in einem Array `fabricElementsArray` welcher an den State zurückgegeben wird.

Am Ende dieses Prozesses besitzt die `renderState()`-Funktion einen Array `toRenderedObjects`, der alle Fabric.js-Objekte enthält, die in diesem State instanziiert wurden.

Um Elemente ein- und auszublenden, erhält jede Komponente zudem einen Boolean `hidden`. Ist dieser bei einer Instanz, manuell oder programmatisch, auf `true` gesetzt, so wird diese beim Einsammeln der `componentArrays` nicht berücksichtigt.

Zurück in der `client_app.js`: Nachdem das Application-Model an den Client gesendet wurde, daraufhin die States instanziiert wurden und diese ihrerseits die Komponenten instanziiert haben, wird mit dem Befehl

```
GLOBALS.state0.renderState();
```

zur Begrüßung die Hintergrund-Auswahl präsentiert.

Sobald der User ein Thumbnail angeklickt hat, wird zuerst in der `DrawingArea`-Instanz der Hintergrund neu gesetzt, was in der Stundenplanlogik dazu führt, dass mit dem Befehl

```
GLOBALS.state1.renderState();
```

der State mit der Hauptapplikation geladen wird. Dann wird die Änderung zum Server geschickt.

Wird eine URL aufgerufen, die einen Parameter enthält, zu dem es bereits einen Plan gibt, so wird dieser per Socket direkt hinterhergesendet. Auf dieses Event hin werden die entsprechenden Instanzen der UI-Elemente modifiziert und ebenso wird mit dem Befehl

```
GLOBALS.state1.renderState();
```

sofort der State mit der Hauptapplikation geladen.

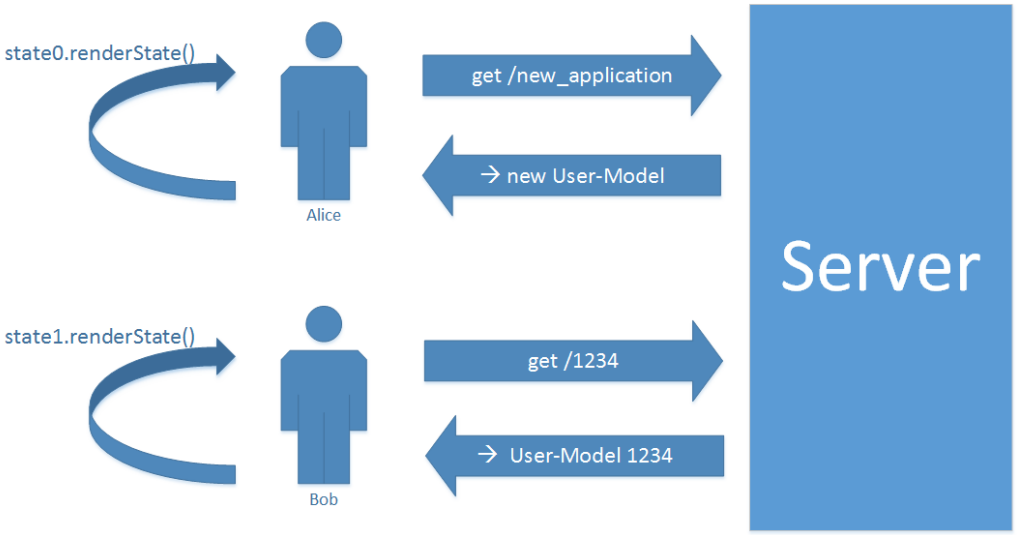

Abbildung 83: Zusammenhang zwischen URL-Aufruf und renderState()

Werden in Zukunft Änderungen am User-Model vorgenommen, so wird immer folgender Prozess durchlaufen:

1. Änderung an der Instanz des Zielobjekts (Bspw. Hintergrund der Zeichenfläche)
2. Rendering des betroffenen States mit `state.renderState()`
3. Parallel dazu: Senden der Änderung per Socket an den Server

Gehen Änderungen von anderen Kollaborateuren bei einem selbst ein, so werden ebenfalls Punkt 1 und 2 durchlaufen.

Beim State-Wechsel wird generell ein `renderState()` aufgerufen. Hat ein anderer Kollaborateur beispielsweise Änderungen vorgenommen, während man sich in State 3 mit dem Link-Sharing beschäftigt hat, so werden im Hintergrund bereits die Instanzen angepasst. Beim Zurückkehren in den State der Hauptapplikation werden alle zugehörigen Instanzen dem Canvas neu hinzugefügt und man ist automatisch auf dem aktuellen Stand.

5.7 Fazit zur Implementierung

Die Implementierung des Frameworks bot zahlreiche Herausforderungen, brachte aber auch mindestens ebenso viele Erkenntnisse.

Auch die Anwendung auf einen konkreten Fall, den Stundenplan-Gestalter, half, die Konstruktion auch von Seite des Benutzers des Frameworks und schließlich auch des Anwenders einer fertigen Applikation zu betrachten, zu bewerten und zu verbessern.

Für jemanden mit wenig Erfahrung in der Entwicklung von Single-Page-Applikationen im Webbrowser beindruckte, dass man über einfache, in JSON notierte Datenmodelle und selbstgeschriebene Komponenten Benutzeroberflächen mit differenzierter Logik und Optik erstellen kann und dass sogar Applikations-„Seiten“ oder -„Zustände“, in diesem Framework „States“ genannt, gut zu realisieren sind.

Bezüglich der Performance lässt sich noch nicht viel sagen, da keine Tests auf großen Anzahlen von Clients realisiert wurden. Aufgefallen ist jedoch, dass einige Browser Bewegungen im Canvas wesentlich flüssiger darstellen als andere. Auf mobilen Geräten fiel auf, dass die Latenz der Echtzeit-Synchronisierung auf neueren Geräten wesentlich geringer ist als auf älteren.

Es wurden zwar einige Funktionen des Flash-Stundenplan-Gestalters im Prototyp schon umgesetzt, doch vieles fehlt noch. Dies ist größtenteils keine Technologie-, sondern eine Zeitfrage. Manche Funktionen würden nicht nur den Wert der Beispielapplikation steigern, sondern auch den Wert des ganzen Frameworks.

Viele Funktionen des Flash-Stundenplan-Gestalters lassen sich mit Wiederverwendung von Code bereits implementierter Features realisieren. Komplett neu implementieren muss man neben der Druckfunktion eine Audio-Ausgabe, eine Undo-Redo-Funktionalität und scrollbare Bereiche. Auch Drag-and-drop-Prozesse von der Menüführung auf die Zeichenfläche erwiesen sich als schwierig, weshalb sie im Prototyp durch einfache Klicks ersetzt wurden. Weiterhin unterstützt der Flash-Stundenplan-Gestalter einige Shortcuts, um Elemente per Tastatur zu modifizieren.

Manche Teile des Frameworks lassen sich noch generischer gestalten. Das EventHandling beispielsweise reagiert teilweise sehr spezifisch auf Erfordernisse des Stundenplan-Gestalters. Auch manche Werte wie die Serveradresse zur Datenbank, der Port der Anwendung oder die Größe des <canvas>-Elements im HTML-Code stehen fest im Code und können noch in eine Konfigurations-Datei ausgelagert werden.

Wurden in der Einleitung Nachteile bei der Verwendung von Flash hervorgehoben, so sind während der Implementierung auch Nachteile von JavaScript im Zusammenhang mit dem Canvas aufgefallen. Umgehbar sind Probleme wie der Zugriff auf die Zwischenablage des Systems, was mit JavaScript nicht möglich ist. Hier kann mit einem Hinweis auf jeweilige Tastenkombinationen abgeholfen werden. Weit gewichtiger ist die Schwierigkeit, in einem Canvas scrollbare Bereiche zu definieren. Letztere wurden im Prototyp des HTML-Stundenplan-Gestalters mithilfe blätterbarer Komponenten angedeutet.

Auch die Tatsache, dass es nur offene Standards verwendet, bringt für das Framework außer den dargestellten Vorteilen gegenüber einer proprietären Lösung wie Flash auch Nachteile mit sich. Eine Flash-Version der Applikation muss einmal entwickelt werden und funktioniert

dann auf jedem Endgerät, das mit dem Plug-In von Adobe arbeiten kann. Eine JavaScript-Lösung muss auf alle Eigenheiten browserseitiger Interpreter von verschiedenen Firmen abgestimmt sein, um überall zu funktionieren.

Weiterhin Schwierigkeiten bereitete die asynchrone Programmierung in JavaScript. Dass Prozesse völlig aus der Aufruf-Reihenfolge des Codes herausgenommen werden, um dann irgendwann später ins Spiel zu kommen, war nicht immer leicht zu realisieren Serverseitige Asynchronität war hauptsächlich beim Zugriff auf die Datenbank gegeben. Da hier nur relative simple `Save`- und `Find`-Prozesse ausgeführt wurden, konnte der anschließende Code relativ einfach in die Callbacks der Funktionen verlagert werden. Clientseitig allerdings war vor allem das Laden von Bildern problematisch. Im komplexen Gefüge der State-Architektur bedurfte es spezieller Lösungen, um beispielsweise States erst zu rendern, wenn auch alle dazugehörigen Bilder wirklich geladen waren.

Mit Fabric.js gab es auch ein paar Probleme. Noch während der Entwicklung des Frameworks wurden Versionen veröffentlicht, die Mouseover-Events und programmatische Rotation um den Mittelpunkt eines Fabric.js-Objekts möglich machten. Davor konnten Objekte zwar aktiv um ihren Mittelpunkt gedreht werden. Kam aber eine passive, also von einem anderen Client übertragene Änderung, so wurde um die linke obere Ecke gedreht, was in ungewollten Änderungen in der Position resultierte.

Ein ähnliches Problem besteht weiterhin beim Skalieren von Elementen. So erkennt Fabric.js nicht, an welchem Greifer das Element vergrößert wurde und vergrößert programmatisch immer nach rechts unten, obwohl man per Maus in alle vier Ecken vergrößern kann. Darum wurde hier, genau wie bei Drehungen, generell festgelegt, dass Objekte von ihrem Mittelpunkt aus vergrößert werden.

Auch die bereits erwähnte `fabric.Group` ist nicht das, was man sich unter einer echten Gruppierung vorstellt. Sie sammelt nicht hierarchisch die gruppierten Elemente unter sich, sondern schmilzt diese zu einem zusammen, damit man sie gemeinsam transformieren kann. Das zerstört nicht nur das EventHandling, sondern macht auch eine hierarchische Arbeit im Sinne des Datenmodells unmöglich. Dies mag in vielen Fällen argumentierbar sein – und wird so auch in diversen Internetforen bekräftigt –, ist aber spätestens dann, wenn ursprünglich editierbarer `fabric.IText` nicht mehr editiert werden kann, als Bug anzusehen.

Im Laufe des Implementierungs-Prozesses wurde immer deutlicher, dass eine komplett lokale Kopie des User-Datenmodells redundant gepflegt zur Datenbankversion auf dem Server in Kombination mit einem gänzlich zentralisierten EventHandling sinnvoll wäre.
Treten aktuell Änderungen ein, so wird die komplette Prozedur des in Kapitel 5.6.4 beschriebenen Prozesses der renderState()-Funktion durchgeführt.
Existiert eine lokale Version des User-Datenmodells, so kann auf Änderungen lokal granular eingegangen werden, ohne den ganzen State rendern zu müssen. Auch dies ist ein Feature für eine nächste Version des Frameworks.

6 Schlussbetrachtung

6.1 Zusammenfassung

Einleitend wird der Kontext der Aufgabenbeschreibung vorgestellt. In einer Umgebung, in der Internet jederzeit und überall abrufbar ist, macht die Entwicklung von Web-Applikationen gegenüber systemeigenen Applikationen immer mehr Sinn. Hier gilt es, die im Multimedia-Bereich etablierte Flash-Programmierung wegen des immer geringer werdenden Supports durch offene Lösungen des HTML5-Standards zu ersetzen. Konkret wird dies hier mit dem Canvas-Element gelöst. Außerdem dienen WebSockets der Synchronität in Echtzeit zu kollaborativen Zwecken.

In Kapitel zwei geht es um die Theorie, die dem Framework zugrunde liegt. Es ist in zwei Abschnitte eingeteilt. Im ersten Abschnitt werden Vor- und Nachteile sowie Konzepte von generischen Frameworks aufgezeigt. Weiterhin wird auf funktionale und nicht-funktionale Anforderungen an derartige Frameworks eingegangen. Beendet wird der Abschnitt mit der Vorstellung einiger Frameworks, die in der Praxis häufig verwendet werden. Der zweite Abschnitt befasst sich mit Kollaboration im Zusammenhang mit Echtzeit-Webanwendungen. Zu Beginn wird erläutert, wie die Begriffe „Kollaboration“, „Echtzeit“ und „Web-Anwendung“ definiert sind. Im weiteren Verlauf wird auf Techniken der Kollaboration in der Raum-Zeit-Klassifizierung eingegangen. Abgeschlossen wird das Kapitel mit der Vorstellung diverser technischer Umsetzungsmöglichkeiten sowie einem Beispiel aus der Praxis.

Kapitel drei befasst sich mit den Webtechnologien, die im Rahmen dieser Thesis verwendet werden. Der erste Teil des Kapitels beinhaltet neben einer Kurzzusammenfassung über die Neuerungen in HTML5 eine detaillierte Vorstellung des HTML5-Canvas und der WebSockets. Es folgt ein kurzer Abriss über die Programmiersprache JavaScript. Im zweiten Teil des Kapitels wird Node.js als serverseitige Infrastruktur präsentiert. Weiterhin werden Produkte für die serverseitige Speicherung der Daten vorgestellt und verglichen. Zum Abschluss werden einige Canvas-Bibliotheken gegenübergestellt.

Kapitel vier beschäftigt sich mit den konkreten Anforderungen an das Framework. Sie sind gegliedert in Oberfläche, Anwendung, Technik und Implementierung.

Als letztes Kapitel des Hauptteils beinhaltet Kapitel fünf die Konzeption und Implementierung des Frameworks. Nach einer kurzen Übersicht über die Dateistruktur folgen konkrete Konzepte zur Client-Server-Kommunikation. Es wird verglichen, wie Design und Funktionalität des aktuellen Flash-Stundenplan-Gestalters Verwendung im Prototyp der zu entwickelnden HTML5-Version finden. Schließlich werden Entwurf und Umsetzung des Frameworks mit Codebeispielen präsentiert. Abschließend wird ein Zwischenfazit zur Implementierung gezogen, welches auf Schwierigkeiten und Erkenntnisse im Entwicklungsprozess sowie mögliche Funktionen in künftigen Versionen eingeht.

6.2 Fazit

Diese Thesis beinhaltet eine Auseinandersetzung mit der Theorie von Frameworks und kollaborativen Webanwendungen. Als praktischer Teil wurde ein Framework für kollaborative Echtzeit-Webanwendungen implementiert. Diese Abschlussarbeit wurde von der Firma SinusQuadrat GmbH in Offenburg mitbetreut und soll dort in Zukunft weiterentwickelt werden.

6.3 Ausblick

Features, die kommende Versionen des Frameworks erhalten sollen, sind an einigen Stellen in Kapitel fünf aufgezeigt worden. Hier soll ein Überblick gegeben werden, welche Applikationen man neben dem Stundenplan-Gestalter noch mit dem Framework entwickeln könnte. Zusammengetragen wurden viele Ideen von Kommilitonen, Verwandten, Bekannten und Mitarbeitern, dennoch ist es letztlich der Kreativität des Benutzers selbst überlassen, was er mit dem Framework anstellt.

Zum einen wurden Vorschläge gemacht, die Konzeptionsprozesse vereinfachen sollen:

- Mindmapping-Tool
- Schaltpläne
- Grundriss-Zeichnungen für Architekten
- Allgemein: Strukturpläne

Zum anderen könnte das Framework bei organisatorischen Problemen helfen:

- Sitzplan eines Saals (Hochzeit, etc.)
- WG-Putzplan
- Allgemein: Einteilungen im zeitlichen und/oder örtlichen Kontext

Drittens wurden Ablaufdiagramme vorgeschlagen:

- Im Kontext der aktuell stattfindenden WM: Taktik-Planung für Spielzüge
- Handlungsabläufe bei der Bedienung komplexer Geräte

6.4 Danksagung

Hiermit möchte mich bei meinen beiden Betreuern, Herrn Prof. Dr. Rüdebusch und Herrn Schober-Wenger, für die gute Zusammenarbeit und zahlreiche hilfreiche Ratschläge bedanken. Außerdem gilt mein Dank meinem Vater für seine ausführlichen grammatikalischen Lektionen.

Weiterhin bedanke ich mich Juriy Zaytsev, dem Entwickler von Fabric.js, der mir bei einigen meiner Fragen im privaten Gespräch zur Seite stand, sowie der ganzen Open-Source-Community für ihre großartigen Produkte.

Zuletzt geht mein Dank an die Mildenberger Verlag GmbH, die sich bereit erklärt hat, mir die Grafiken des Flash-Stundenplan-Gestalters für den Prototyp der HTML-Version zur Verfügung zu stellen. Das Copyright der verwendeten Grafiken (Hintergründe, Icons) liegt bei der Mildenberger Verlag GmbH.

Literaturverzeichnis

[1] W3Techs, *Usage Statistics of Flash for Websites, May 2014.* Available: http://w3techs.com/technologies/details/cp-flash/all/all (2014, May. 14).

[2] de.statista.com, *Mobile Betriebssysteme - Marktanteile Internetnutzung Deutschland bis 2014 | Statistik.* Available: http://de.statista.com/statistik/daten/studie/184332/umfrage/marktanteil-der-mobilen-betriebssysteme-in-deutschland-seit-2009/ (2014, Apr. 16).

[3] SinusQuadrat GmbH, *Internetagentur Offenburg.* Available: http://www.sinusquadrat.com/ (2014, Apr. 02).

[4] SinusQuadrat ~ Dennis Wenger, *der-stundenplan.de: Gestalte deinen Stundenplan.* Available: http://www.der-stundenplan.de/stundenplan/der-stundenplan/ (2014, Apr. 02).

[5] B. P. Douglass, *Doing hard time: Developing real-time systems with UML, objects, frameworks, and patterns*. Reading, Mass: Addison-Wesley, 1999.

[6] B. Foote and R. E. Johnson, "Designing Reusable Classes," *Journal of Object-Oriented Programming*, no. 2, pp. 22–35, 1988.

[7] G. F. Rogers, *Framework-based software development in C++*. Upper Saddle River, N.J: Prentice Hall PTR, 1997.

[8] X Wang, *Library vs. Framework?* Available: http://www.programcreek.com/2011/09/what-is-the-difference-between-a-java-library-and-a-framework/ (2014, May. 16).

[9] S. Robertson and J. Robertson, *Mastering the requirements process: Getting requirements right*, 3rd ed. Upper Saddle River, NJ: Addison-Wesley, 2013.

[10] A. Rusnjak, "Entrepreneurial Business Modeling: Definitionen - Vorgehensmodell - Framework - Werkzeuge - Perspektiven," *Entrepreneurial Business Modeling*, 2014.

[11] T. Lüecke, *Software-Engineering.* Available: http://www.se.uni-hannover.de/priv/lehre_2004winter_seminar_software_entwurf/15_frameworks.pdf (2014, May. 20).

[12] R. Bergmann, *Frameworks.ppt.* Available: http://wwwiti.cs.uni-magdeburg.de/iti_ti/SS04/vortraegeSeminar/Frameworks.pdf (2014, May. 20).

[13] K. Stirewalt, *Brief Introduction to Object-Oriented Frameworks.* Available: http://www.cse.msu.edu/SENS/local_only/Presentations/Slides/frameworks-6-14-01-sens.ppt.

[14] HotFrameworks, *Web framework rankings | HotFrameworks.* Available: http://hotframeworks.com/ (2014, May. 21).

[15] github.com, *Build software better, together.* Available: https://github.com/ (2014, May. 21).

[16] Stack Exchange, *Stack Overflow.* Available: http://stackoverflow.com/ (2014, May. 21).

[17] angularjs.org, *AngularJS — Superheroic JavaScript MVW Framework.* Available: https://angularjs.org/ (2014, Jun. 02).

[18] expressjs.com, *Express - node.js web application framework.* Available: http://expressjs.com/ (2014, Jun. 02).

[19] emberjs.com, *Ember.js - A framework for creating ambitious web applications.* Available: http://emberjs.com/ (2014, Jun. 02).

[20] J. M. Leimeister, *Collaboration Engineering: IT-gestützte Zusammenarbeitsprozesse systematisch entwickeln und durchführen*, 2013rd ed. Berlin: Springer Berlin, 2013.

[21] itwissen.info, *Kollaboration :: collaboration :: ITWissen.info.* Available: http://www.itwissen.info/definition/lexikon/Collaboration-collaboration.html (2014, May. 22).

[22] P. Scholz, *Softwareentwicklung eingebetteter Systeme: Grundlagen, Modellierung, Qualitätssicherung,* 1st ed. Berlin: Springer, 2005.
[23] J. Quade and M. Mächtel, *Moderne Realzeitsysteme kompakt: Eine Einführung mit Embedded Linux,* 1st ed. Heidelberg [Germany]: Dpunkt.verlag, 2012.
[24] S. Keck, "Vorlesung System- und Informationstheorie," Offenburg, Wintersemester 2013.
[25] G. Kappel, *Web engineering: Systematische Entwicklung von Web-Anwendungen,* 1st ed. Heidelberg: Dpunkt-Verl, 2004.
[26] skype.com, *Kostenlos anrufen mit Skype, Webkonferenzen.* Available: http://www.skype.com/de/ (2014, Jun. 02).
[27] Briggs, R. O, Kolfschoten, G. L, de Vreede, G.-J, Albrecht, C, Lukosch, S, & Dean, D. L, "A six-layer model of collaboration for designers of collaboration systems," in *Advances in Collaboration Systems,* Nunamaker Jr, Briggs, Romano Jr, Ed, New York: Armonk, 2009.
[28] TeamViewer GmbH, *TeamViewer – Access your computer remotely and share your desktop with friends – it's free!* Available: http://www.teamviewer.com/de/index.aspx (2014, Jul. 07).
[29] Daily Mail Reporter, *The fridge-sized computer that sent the very first email 40 years ago.* Available: http://www.dailymail.co.uk/sciencetech/article-1224100/Internets-40th-birthday-First-email-crashes-just-letters.html (2014, Jun. 02).
[30] Wikimedia Foundation Inc, *Wikipedia – Die freie Enzyklopädie.* Available: http://de.wikipedia.org/w/index.php?oldid=130433952 (2014, Jul. 07).
[31] W3C Consortium, *The WebSocket API Publication History - W3C.* Available: http://www.w3.org/standards/history/websockets (2014, Jun. 02).
[32] The Chromium Projects, *SPDY: An experimental protocol for a faster web - The Chromium Projects.* Available: http://dev.chromium.org/spdy/spdy-whitepaper (2014, Jun. 02).
[33] websocket.org, *WebSocket.org | The Benefits of WebSocket.* Available: http://www.websocket.org/quantum.html (2014, Apr. 10).
[34] Adobe, *Socket - Adobe ActionScript® 3 (AS3) API Reference.* Available: http://help.adobe.com/en_US/FlashPlatform/reference/actionscript/3/flash/net/Socket.html (2014, Jun. 03).
[35] Etherpad Foundation, *Etherpad.* Available: http://etherpad.org/ (2014, Jun. 03).
[36] W3C Consortium, *HTML5.* Available: http://www.w3.org/TR/html5/ (2014, Apr. 08).
[37] WHATWG, *Web Hypertext Application Technology Working Group.* Available: http://www.whatwg.org/ (2014, Apr. 16).
[38] w3schools.com, *Browser Statistics.* Available: http://www.w3schools.com/browsers/browsers_stats.asp (2014, Apr. 08).
[39] w3schools.com, *HTML Canvas Reference.* Available: http://www.w3schools.com/tags/ref_canvas.asp (2014, Apr. 08).
[40] w3schools.com, *HTML track Tag.* Available: http://www.w3schools.com/tags/tag_track.asp (2014, Apr. 16).
[41] Dive Into HTML5, *How Did We Get Here? - Dive Into HTML5.* Available: http://diveintohtml5.info/past.html#timeline (2014, Apr. 16).
[42] IrusSoft, *Web Design Development, Mobile Apps, in Pakistan,| Irus Soft.* Available: http://www.irussoft.com/ (2014, May. 14).
[43] w3schools.com, *HTML5 Semantic Elements.* Available: http://www.w3schools.com/html/html5_semantic_elements.asp (2014, Apr. 16).
[44] P. Kröner, *HTML5: Webseiten innovativ und zukunftssicher*. München: Open Source Press, 2010.

[45] w3schools.com, *HTML5 Input Types.* Available: http://www.w3schools.com/html/html5_form_input_types.asp (2014, Apr. 16).
[46] W3C Consortium, *4.10 Forms — HTML5.* Available: http://www.w3.org/TR/html5/forms.html#client-side-form-validation (2014, Apr. 08).
[47] w3schools.com, *HTML5 Video.* Available: http://www.w3schools.com/html/html5_video.asp (2014, Apr. 16).
[48] w3schools.com, *HTML5 Audio.* Available: http://www.w3schools.com/html/html5_audio.asp (2014, Apr. 16).
[49] W3C Consortium, *Geolocation API Specification.* Available: http://www.w3.org/TR/geolocation-API/ (2014, Apr. 16).
[50] w3schools.com, *HTML5 Drag and Drop.* Available: http://www.w3schools.com/html/html5_draganddrop.asp (2014, Apr. 16).
[51] w3schools.com, *HTML5 Web Storage.* Available: http://www.w3schools.com/html/html5_webstorage.asp (2014, Apr. 16).
[52] w3schools.com, *HTML5 Application Cache.* Available: http://www.w3schools.com/html/html5_app_cache.asp (2014, Apr. 08).
[53] w3schools.com, *HTML5 Web Workers.* Available: http://www.w3schools.com/html/html5_webworkers.asp (2014, Apr. 16).
[54] w3schools.com, *HTML5 Server-Sent Events.* Available: http://www.w3schools.com/html/html5_serversentevents.asp (2014, Apr. 16).
[55] Khronos, *WebGL Specification.* Available: https://www.khronos.org/registry/webgl/specs/latest/1.0/ (2014, Apr. 16).
[56] caniuse.com, *Can I use… Support tables for HTML5, CSS3, etc.* Available: http://caniuse.com/ (2014, Apr. 09).
[57] caniuse.com, *Can I use the HTML5 canvas element.* Available: http://caniuse.com/canvas (2014, May. 14).
[58] caniuse.com, *Can I use Form validation.* Available: http://caniuse.com/form-validation (2014, May. 14).
[59] caniuse.com, *Can I use the Input-Datetime.* Available: http://caniuse.com/#feat=input-datetime (2014, May. 14).
[60] youtube.com, *Charging Smartphone in 30S: StoreDot Flash-Battery Demo.* Available: http://www.youtube.com/watch?v=9DhJZAhjbcI (2014, May. 14).
[61] mudcu.be, *Sketchpad Online.* Available: http://mudcu.be/sketchpad/ (2014, May. 14).
[62] CodeGraphics, *HTML6 and CSS4: My Proposals - CSS-Tricks.* Available: http://css-tricks.com/forums/topic/html6-and-css4-my-proposals/ (2014, Apr. 09).
[63] W3C Consortium, *HTML/next - W3C Wiki.* Available: http://www.w3.org/wiki/HTML/next (2014, Apr. 09).
[64] Ian Hickson, *The WHATWG Blog — HTML is the new HTML5.* Available: http://blog.whatwg.org/html-is-the-new-html5 (2014, Apr. 10).
[65] Mark Pilgrim, *The WHATWG Blog — What's Next in HTML, episode 1.* Available: http://blog.whatwg.org/whats-next-in-html-episode-1 (2014, Apr. 10).
[66] WHATWG, *HTML Standard.* Available: http://www.whatwg.org/specs/web-apps/current-work/multipage/ (2014, Apr. 10).
[67] WHATWG, *HTML Standard Tracker.* Available: http://html5.org/tools/web-apps-tracker?from=4438&to=4439 (2014, Apr. 10).
[68] Mark Pilgrim, *The WHATWG Blog — What's next in HTML, episode 2: who's been peeing in my sandbox?* Available: http://blog.whatwg.org/whats-next-in-html-episode-2-sandbox (2014, Apr. 10).
[69] IETF, *The WebSocket Protocol.* Available: http://tools.ietf.org/html/rfc6455 (2014, Apr. 16).

[70] W3C Consortium, *The Web Sockets API.* Available: http://www.w3.org/TR/2009/WD-websockets-20091222/ (2014, Apr. 16).
[71] V. Wang, *The definitive guide to HTML5 WebSocket*. [New York]: Apress, 2013.
[72] W3C Consortium, *HTTP/1.1: Header Field Definitions.* Available: http://www.w3.org/Protocols/rfc2616/rfc2616-sec14.html#sec14.42 (2014, Apr. 11).
[73] IETF, *Hypertext Transfer Protocol -- HTTP/1.1.* Available: http://tools.ietf.org/html/rfc2616#section-10.1.2 (2014, Apr. 11).
[74] google.de, *Google Trends - Websuche-Interesse: websocket - Weltweit, 2004 - heute.* Available: http://www.google.de/trends/explore#q=websocket (2014, May. 14).
[75] caniuse.com, *Can I use… Support tables for HTML5, CSS3, etc.* Available: http://caniuse.com/#feat=websockets (2014, May. 14).
[76] kaazing.com, *Kaazing FX Demonstration.* Available: http://demo.kaazing.com/forex/ (2014, May. 14).
[77] kaazing.com, *Kaazing Real-Time Web Communication.* Available: http://demo.kaazing.com/livefeed/ (2014, May. 14).
[78] R. Harmes and D. Diaz, *JavaScript, Objektorientierung und Entwurfsmuster: [stabilen und sicheren Code mit JavaScript entwickeln]*. Poing: Franzis, 2008.
[79] E. T. Freeman and E. Robson, *Head First JavaScript Programming,* 1st ed. Sebastopol, CA: O'Reilly & Associates, 2014.
[80] Ingo Pakalski, *15 Jahre WWW: Die Browserkriege - Golem.de.* Available: http://www.golem.de/0805/59377.html (2014, Apr. 14).
[81] John Resig, *John Resig - Versions of JavaScript.* Available: http://ejohn.org/blog/versions-of-javascript/ (2014, Apr. 24).
[82] ECMAScript, *harmony:specification_drafts [ES Wiki]*. Available: http://wiki.ecmascript.org/doku.php?id=harmony:specification_drafts (2014, Apr. 14).
[83] A. Rauschmayer, *Speaking JavaScript*. Sebastopol, CA: O'Reilly Media, 2014.
[84] Team23 Webdesign, *Webanalyse: Verbreitung von Webtechnologien und Plugins - Tutorials, Tipps und Tricks für Webmaster auf Webmasterpro.de.* Available: http://www.webmasterpro.de/portal/webanalyse-technologien.html (2014, Apr. 14).
[85] readwrite.com, *Server-Side Javascript: Back With a Vengeance.* Available: http://readwrite.com/2009/12/17/server-side_javascript_back_with_a_vengeance#awesm=~oG1iiHY3YfLbL3 (2014, Jun. 02).
[86] G. Roden, *Node.js & Co: Skalierbare hochperformante und echtzeitfähige Webanwendungen professionell in JavaScript entwickeln,* 1st ed. Heidelberg: dpunkt-Verl, 2012.
[87] StrongLoop, *StrongLoop | What Makes Node.js Faster Than Java?* Available: http://strongloop.com/strongblog/node-js-is-faster-than-java/ (2014, May. 14).
[88] NPM, *npm.* Available: https://www.npmjs.org/ (2014, Apr. 17).
[89] AppDynamics, *An example of how Node.js is faster than PHP - Application Performance Monitoring Blog from AppDynamics.* Available: http://www.appdynamics.com/blog/nodejs/an-example-of-how-node-js-is-faster-than-php/ (2014, Apr. 23).
[90] WordPress, *WordPress › Blog Tool, Publishing Platform, and CMS.* Available: http://wordpress.org/ (2014, Apr. 23).
[91] Ghost Foundation, *Ghost - Just a blogging platform.* Available: https://ghost.org/ (2014, Apr. 23).
[92] F. Geisendörfer, *Felix's Node.js Convincing the boss guide.* Available: http://nodeguide.com/convincing_the_boss.html#building-a-prototype (2014, Apr. 23).
[93] Socket.io, *Socket.IO: the cross-browser WebSocket for realtime apps.* Available: http://socket.io/#browser-support (2014, Apr. 17).

[94] PayPal, *PayPal: Bequem bezahlen, Zahlungen empfangen & Geld senden.* Available: https://www.paypal.com/de/webapps/mpp/home (2014, May. 14).
[95] PayPal, *Node.js at PayPal | PayPal Engineering Blog.* Available: https://www.paypal-engineering.com/2013/11/22/node-js-at-paypal/ (2014, Apr. 23).
[96] MySpace, *Featured Content on Myspace.* Available: https://myspace.com/ (2014, May. 14).
[97] venturebeat.com, *Exclusive: How LinkedIn used Node.js and HTML5 to build a better, faster app.* Available: http://venturebeat.com/2011/08/16/linkedin-node/ (2014, Apr. 23).
[98] venturebeat.com, *Why Walmart is using Node.js.* Available: http://venturebeat.com/2012/01/24/why-walmart-is-using-node-js/ (2014, Apr. 23).
[99] venturebeat.com, *LinkedIn's new mobile app is so gorgeous, you'll actually want to use it.* Available: http://venturebeat.com/2011/08/16/linkedin-mobile-app/ (2014, May. 14).
[100] google.de, *Google Trends - Websuche-Interesse - Weltweit, 2004 - heute.* Available: http://www.google.de/trends/explore#q=node.js (2014, May. 14).
[101] Node.js-Team, *Node.js Blog.* Available: http://blog.nodejs.org/ (2014, Apr. 23).
[102] I. Z. Schlueter, *The Next Phase of Node.js.* Available: http://blog.nodejs.org/2014/01/15/the-next-phase-of-node-js/ (2014, Apr. 23).
[103] T. J. Fontaine, *Node.js and the Road Ahead.* Available: http://blog.nodejs.org/2014/01/16/nodejs-road-ahead/ (2014, Apr. 23).
[104] db-engines.com/, *DB-Engines Ranking - die Rangliste der populärsten Datenbankmanagementsysteme.* Available: http://db-engines.com/de/ranking (2014, Apr. 24).
[105] A. Hofstetter, "Einsatz von NoSQL-Datenbanken in modernen Webapplikationen," Hochschule Offenburg, 2012.
[106] mongodb.org, *MongoDB.* Available: https://www.mongodb.org/ (2014, Apr. 28).
[107] Apache Software Foundation, *Apache CouchDB.* Available: http://couchdb.apache.org/ (2014, Apr. 28).
[108] db-engines.com, *DB-Engines Ranking - die Rangliste der populärsten Document Stores.* Available: http://db-engines.com/de/ranking/document+store (2014, Apr. 28).
[109] M. Boeker, *MongoDB: Sag ja zu NoSQL,* 1st ed. Frankfurt: Entwickler Press, 2010, 2010.
[110] N. Hurst, *Visual Guide to NoSQL Systems.* Available: http://blog.nahurst.com/visual-guide-to-nosql-systems (2014, Apr. 28).
[111] Apache Software Foundation, *Apache CouchDB™ 1.6.0-dev Documentation — Apache CouchDB 1.6 Documentation.* Available: http://docs.couchdb.org/en/latest/contents.html (2014, Apr. 28).
[112] Fabric.js, *Fabric.js Javascript Canvas Library.* Available: http://fabricjs.com/ (2014, Apr. 09).
[113] kinetic.js, *KineticJS HTML5 Canvas Framework.* Available: http://kineticjs.com/ (2014, Apr. 09).
[114] CreateJS, *EaselJS | A Javascript library that makes working with the HTML5 Canvas element easy.* Available: http://www.createjs.com/#!/EaselJS (2014, Apr. 09).
[115] Paper.js, *Paper.js.* Available: http://paperjs.org/ (2014, Apr. 09).
[116] zebkit, *Zebkit HTML5 Canvas Rich UI JavaScript Library.* Available: http://www.zebkit.com/ (2014, Apr. 09).
[117] J. Koggdal, *oCanvas - Object-based canvas drawing.* Available: http://ocanvas.org/ (2014, Apr. 09).
[118] kallaspriit, *HTML Canvas Library.* Available: http://html-canvas-lib.sourceforge.net/ (2014, Apr. 09).
[119] http://jcscript.com/, *jCanvaScript.* Available: http://jcscript.com/ (2014, Apr. 09).

[120] bhivecanvas.com, *bHive - The JavaScript Canvas Library, Framework, API.* Available: http://www.bhivecanvas.com/ (2014, Apr. 09).
[121] Fabric.js, *Demos — Fabric.js Javascript Canvas Library.* Available: http://fabricjs.com/demos/ (2014, May. 14).
[122] Fabric.js, *Custom Fabric build — Fabric.js Javascript Canvas Library.* Available: http://fabricjs.com/build/ (2014, May. 14).
[123] NPM guileen, *mongoskin.* Available: https://www.npmjs.org/package/mongoskin (2014, Jul. 03).
[124] NPM - pvorb, *clone.* Available: https://www.npmjs.org/package/clone (2014, Jul. 03).
[125] James Burke, *RequireJS.* Available: http://requirejs.org/ (2014, Jul. 07).
[126] Jeremy Ashkenas, *Underscore.js.* Available: http://underscorejs.org/ (2014, Jul. 07).
[127] Guille Paz, *pazguille/jvent.* Available: https://github.com/pazguille/jvent (2014, Jul. 07).

Anhang

Der komplette Quellcode der praktischen Implementierung liegt auf einer CD-ROM bei.