Jade Hochschule
Fachbereich M.I.T.
Studiengang Wirtschaftsinformatik

Bachelorarbeit

Prototypische Implementierung einer SAP UI5 Applikation im SAP Umfeld und Analyse eines effizienten Einsatz von UI-Objekten

eingereicht von: Nils Lutz

bei: Prof. Dr. Hergen Pargmann
Prof. Dr. Harald Schallner

I Kurzfassung

Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet. Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet.

Abstract

Das ganze auf Englisch.

II Inhaltsverzeichnis

III Abbildungsverzeichnis

IV Tabellenverzeichnis

V Listing-Verzeichnis

VI Abkürzungsverzeichnis

JSP	Java Server Pages
BSP	Business Server Pages
HTML	Hypertext Markup Language
CSS	Cascading Style Sheets
JS	JavaScript
WWW	World Wide Web
W3C	World Wide Web Consortium
XHTML	Extensible Hypertext Markup Language
XML	Extensible Markup Language
SGML	Standard Generalized Markup Language
DOM	Dokument-Objekt-Modell
WHATWG	Web Hypertext Application Technology Working Group
API	Application-Programming-Interface
ECMA	European Computer Manufacturers Association
SDK	Software Development Kit
ODBC	Open Database Connectivity
JDBC	Java Database Connectivity
IDE	Integrated Development Environment

1 Einleitung

Motivation // wieso weshalb warum wo
// Beschreibung abatAG
// Enstehung des Projekts

Problemstellung // aktuelle situationsbeschreibung
// was soll besser laufen

Zielsetzung // Das Produkt - Template Programmierung für SAP Frontends mit SAP UI5

Struktur // kapitel oberflächlich anreißen

2 Technologien

Zum besseren Verständnis der gesamten Thematik werden in den folgenden Kapiteln verwendete Technologien erläutert. Die Grundlagen und besonderen Merkmale der einzelnen Technologien helfen dabei die spätere Analyse nach vollziehen zu können. Zu den Kernsprachen, mit denen im Browser visuelle Informationen angezeigt und verändert werden können, zählen unter anderem die Auszeichnungssprache Hypertext Markup Language (HTML), die Gestaltungssprache Cascading Style Sheets (CSS) und die Skriptsprache JavaScript (JS). Aufbauend auf den drei genannten Sprachen setzen sich in der Regel Frameworks.Frameworks sind in sich konsistente Bibliotheken die gewisse Sprachkonstrukte, welche häufig benötigt werdem in der Entwicklung, zur Verfügung stellen. Mit dem Einsatz eines Frameworks verfolgt man das ziel oft geschriebenen Programm Code in eine Art *Bausatz-Konstruktions-Set* auszulagern. So lässt sich ein einmal durch geführter Entwicklungsprozess beliebig oft und mit weit weniger Aufwand bewerkstelligen, als wenn man jedes mal den Programm Code von neuem entwickeln müsste.

2.1 HTML5

Historie HTML5 ist die aktuell empfohlene Spezifikation des World Wide Web Consortium (W3C) und stellt eine der Kernsprachen des World Wide Web (WWW) dar. Angefangen hat es am 13. März 1989, als Tim Berners-Lee am CERN in Genf das WWW ins Leben gerufen und damit zusammen HTML festgelegt hat. So entstand ab 1990 eine Spezifikation seitens des W3C zur Festlegung und Vereinheitlichung der Kommunikation über das Internet. Im November 1995 erklärte das W3C HTML 2.0 zum offiziellen Sprachstandard. Grundlegende Unterschiede zwischen Version 1.0 und 2.0 existieren nicht. Version 3.0 der HTML Spezifikation ist gänzlich am Browser Markt vorbei definiert worden. Aus diesem Grund wurde HTML 3.2 ab Januar 1997 zum Nachfolger von Version 2.0 gemacht. Die folgende Entwicklung der Spezifikation brachte 1999 die überarbeitete Version 4.01 hervor. Im selben Zug wurde CSS, als Gestaltungssprache für HTML, immer mehr fokussiert. So Begann die Fragmentierung der HTML Spezifikation und es existierten drei Versionen zur selben Zeit. Nämlich HTML 4.01 *strict*, die dem eigentlich definiertem HTML am nächsten kam. HTML 4.01 *transitional*, in welcher auch einige übliche physische Textauszeichnungen vorgesehen waren. „Physische Textauszeichnungen haben Bedeutungen wie „fett“ oder „kursiv“, stellen also direkte Angaben zur gewünschten Schriftformatierung dar. Bei physischen Elementen sollte der Web-Browser eine Möglichkeit finden, den so ausgezeichneten Text entsprechend darzustellen.“[Self]. Sie wurde als Übergangslösung entwickelt. Die dritte Variante ist HTML 4.01 *frameset*. Der einzige Unterschied zur *transitional* Variante ist, dass sich

im Rumpf eines HTML Dokuments ein Element verändert. Neben HTML wurde ab Januar 2000 auch eine Extensible Hypertext Markup Language (XHTML) genannte Spezifikation entwickelt, die HTML mit dem Extensible Markup Language (XML) Standard vereinen sollte. XHTML ist allerdings nicht als eigenständige Sprache zu verstehen, sondern als eine Serialisierungsform für HTML unter Verwendung von XML. Mit HTML 5 wurde die Spezifikation nicht mehr durch die Standard Generalized Markup Language (SGML) - eine Metasprache zur Definition von Auszeichnungssprachen - sondern durch ein Dokument-Objekt-Modell (DOM) beschrieben. Die in dieser Version neu eingeführten Elemente sollten es erlauben HTML Dokumente semantisch klarer zu strukturieren.(vgl. [CG12, S.20ff]) Im Oktober 2014 wurde HTML5 dann vom W3C zum De-facto Standard des WWW erklärt. Heute existiert neben der Spezifikation des W3C auch noch ein sogenannter „lebender Standard“ der Web Hypertext Application Technology Working Group (WHATWG). Die WHATWG ist ein Zusammenschluss von Unternehmen wie zum Beispiel Mozilla Foundation, Opera Software und Apple. Der allgemeine Sprachgebrauch von HTML ist dadurch nicht an die W3C Spezifikation gebunden. Er erstreckt sich über den „lebenden Standard“ derWHATWG hinaus und beinhaltet zahlreiche Schnittstellen zu anderen Technologien. Abbildung 1 verdeutlicht diese Situation.

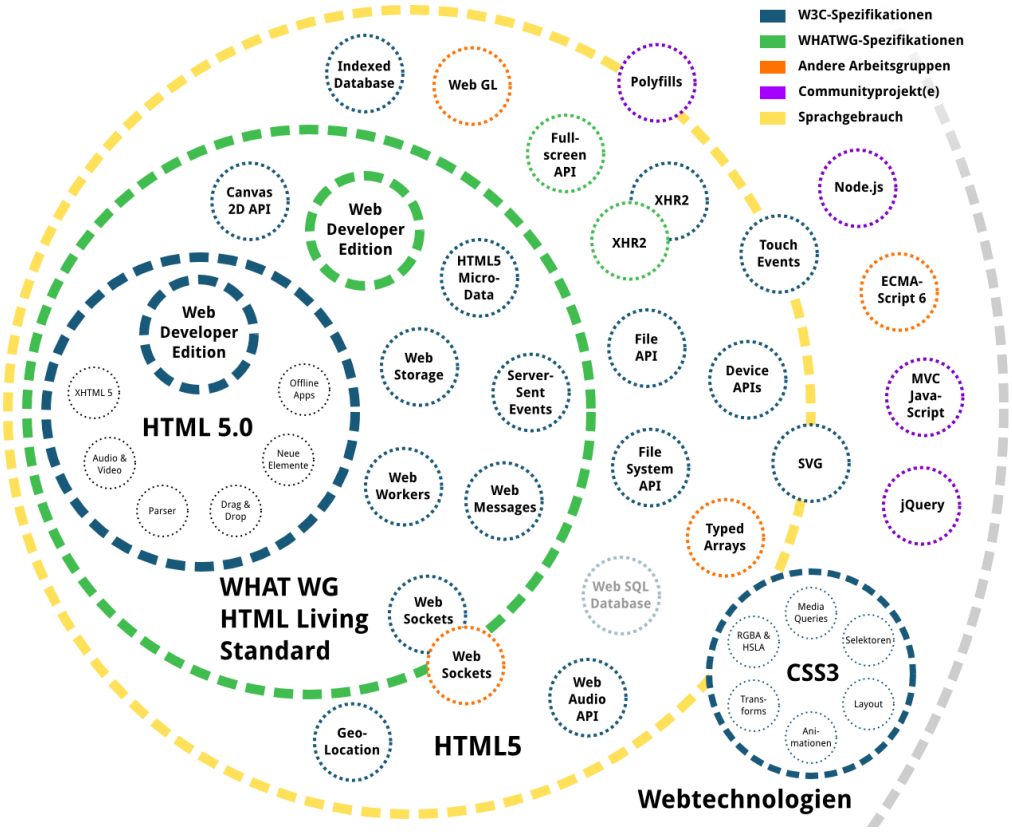

Abbildung 1: HTML5 Spezifikationen Übersicht[Kro]

Ziele HTML5 wurde mit besonderem Augenmerk auf die Kompatibilität entwickelt. Vorhandene Spezifikationen wie HTML 4.01, XHTML 1.0 und DOM 2 sollten unter einem Dach gebündelt werden. Hierdurch wird der vorangegangenen Fragmentierung entgegen gewirkt. Schon vorhandene Inhalte müssen weitestgehend unterstützt werden auch wenn sie nicht zur HTML5 Spezifikation gehören. Beispielsweise werden fehlerhaft verschachtelte Elemente trotzdem akzeptiert. *Graceful degradation* ist als ein weiteres Ziel für HTML 5 definiert worden und bedeutet soviel wie „Schrittweise Abstufung". Es stellt sicher, dass ein HTML Dokument auch dann verarbeitet wird sollte der verwendete Browser ein bestimmtes benutztes Element nicht unterstützen. Weiter galt für die Spezifikation, dass schon vorhandene Techniken, die weitläufig verbreitet sind, nicht neu entwickelt werden sollten. Stattdessen sollten sie übernommen werden. Dies beruht auf dem Umstand, dass die Browser Hersteller jeweils ihre eigenen Techniken bevorzugen und weiter entwickeln und dadurch auch für ihre Verbreitung sorgen. Evolution statt Revolution stand über den Zielen von HTML 5. (X)HTML wurde weiter entwickelt und nicht von Grund auf neu definiert. So ist in Tabelle 1 die zum aktuellen Zeitpunkt verfügbare Unterstützung von HTML 5 in den gängigsten Browsern abzulesen.

Hersteller	**Desktop/Mobile**	**Version**
Mozilla	Firefox	4.0
	Firefox Mobile	16
Google	Chrome	10
	Chrome Mobile	25
	Android	4.0
Apple	Safari	5.1
	Safari iOS	5.1
Microsoft	Internet Explorer	10
	Windows Phone	8
Opera Software	Opera	11.64
Blackberry	Browser	10

Tabelle 1: HTML5 Browserkompatibilität

Aufbau Ein jedes HTML Dokument beginnt mit dem sogenannten *Doctype*. Dieser legt fest mit welcher Syntax das Dokument aufgebaut ist und wie das Dokument vom Browser verarbeitet werden soll. Verschiedene Varianten wie *strict,transitional* und *frameset* sind in HTML5 nicht vorgesehen. In den Vorgängerversionen musste die Variante jedoch mit angegeben werden um eine eindeutige Interpretation des Dokuments zu gewährleisten. Listing 1 zeigt die beiden *Doctype* von HTML 4.01 und XHTML

1.0. Durch die Abwärtskompatibilität von HTML5 sind auch diese *Doctype* heute noch gültig und das Dokument wird korrekt vom Parser interpretiert werden.

```
1 <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
2         "http://www.w3.org/TR/html4/strict.dtd">
3 <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
4         "http://www.w3.org/TR/html4/loose.dtd">
5 <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Frameset//EN"
6         "http://www.w3.org/TR/html4/frameset.dtd">
7 <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
8         "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
```

Listing 1: (X)HTML4.01 *doctype*-Element

Listing 2 hingegen zeigt das *Doctype* von HTML5. Es wurde enorm gekürzt im Vergleich zu dem *Doctype* von HTML 4.01 und XHTML 1.0. Groß- und Kleinschreibung ist nicht von bedeutung innerhalb des *Doctype*.

```
1 <!DOCTYPE html>
```

Listing 2: HTML5 *doctype*-Element

Nach dem *Doctype* folgt der restliche Dokument Aufbau in HTML Syntax. Diese teilt sich auf in Elemente und Attribute, die diesen Elementen zugeordnet und mit Werten versehen werden können. Es existieren für die meisten Elemente Start- und Endmarkierungen. Für einige Elemente sind die Start- bzw. Endmarkierungen optional und für einige wiederum verpflichtend in der Dokumentstruktur zu setzen. In Listing 3 sieht man ein valides HTML5 Dokument mit den Mindestanforderungen.(vgl. [Krö11, S.58])

```
1  <!DOCTYPE html>
2  <html>
3   <head>
4    <title>Beispiel Seite</title>
5   </head>
6   <body>
7    <h1>Beispiel Seite</h1>
8    <p>Dies ist ein <a href="demo.html">einfaches</a> Beispiel.</p>
9    <!-- Dies ist ein Kommentar -->
10  </body>
11 </html>
```

Listing 3: HTML5 Basis Dokument

Wichtige neue Sprachelemente In HTML5 wurden die Mikrodaten mit aufgenommen. Mit Mikrodaten bietet sich eine weitere Möglichkeit ein HTML Dokument semantisch zu spezifizieren. Metadaten wie z.B. der verwendete Zeichensatz lassen sich so festlegen. Browser und Webseiten können über die Mikrodaten-Application-Programming-Interface (API) die gesetzten Werte auslesen und weiter verarbeiten. Auch Suchmaschinen können auf die Metadaten zugreifen, verwenden sie jedoch heutzutage weitestgehend nicht mehr. Aus diesem Grund tragen die Metadaten zwar zur semantischen Struktur des HTML bei, können aber aus Sicht der Suchmaschinenoptimierung getrost vernachlässigt werden.(vgl. [Selg]) Weiter kann man bei Mikrodaten davon sprechen, „[...] dass sie auf Name/Werte-Paaren basieren. Jedes Mikordatenvokabular definiert eine Menge benannter Eigenschaften."(vgl. [Sch11, S.174]) Listing 4 zeigt Beispielhaft das *head*-Element eines HTML Dokuments mit vier, darin eingeschlossenen, *meta*-Elementen. Unter anderem wird mit dem ersten *meta*-Element der Zeichensatz näher definiert. Das *meta*-Element mit Namen *viewport*, dient dazu die Skalierung auf Mobilgeräten zu unterdrücken, damit die Seite sich an den *viewport* anpasst.

```
1 <head>
2   <meta charset="utf-8">
3   <meta name="viewport" content="width=device-width;" />
4   <meta name="keywords" content="Lorem ipsum">
5   <meta name="author"   content="dolor sem it">
6 </head>
```

Listing 4: HTML5 *meta*-Element

Zwei weitere neue Elemente in HTML5 sind das *header*- und *footer*-Element. Zu Zeiten vor HTML5 wurden jene Bereiche einer Website durch *div*-Elemente mit dem *id*- oder *class*-Attribut entsprechend gekennzeichnet. Das ist jetzt nicht mehr nötig durch die beiden neuen Elemente. Üblich ist es im *header*-Element einer Website Komponenten wie das Logo, das Menü und den Titel unterzubringen. Im *footer*-Element dagegen werden Kontakt, Impressum und das Copyright aufgeführt. In Listing 5 ist die Platzierung des *header*- und *footer*-Elements innerhalb eines *body*-Elements einer Webseite zu sehen.

```
1 <body>
2   <header>
3     <img src="logo.gif" alt="logo">
4     <h1>Ueberschrift</h1>
5   </header>
6   <footer>
7     <a href="kontakt.html">Kontakt</a>
```

```
8     </footer>
9  </body>
```

Listing 5: HTML5 *header*- und *footer*-Element

Um ein HTML Dokument nach heutigen Maßstäben korrekt zu strukturieren wurden einige Elemente der Spezifikation hinzugefügt. So lässt sich die Navigation nun mit dem *nav*-Element umschließen wie in Listing 6 ab Zeile 3 zu sehen. „Das *section*-Element repräsentiert einen allgemeinen Abschnitt in einem Dokument oder einer Anwendung. Ein Abschnitt ist in diesem Kontext eine thematische Gruppierung von Inhalten, die üblicherweise unter einer Überschrift stehen. Beispiele für Abschnitte wären Kapitel, die verschiedene Tabs in einem Dialog mit Tabs oder die nummerierten Abschnitte einer wissenschaftlichen Arbeit.[...] Das *article*-Element repräsentiert eine abgeschlossene Einheit in einem Dokument, einer Anwendung oder einer Site, die unabhängig verbreitet oder wiederverwendet werden kann, z.B. in RSS-Feeds. Es könnte beispielsweise ein Forenbeitrag, ein Zeitschriften- oder Zeitungsartikel, ein Blog-Eintrag, ein Benutzerkommentar, ein interaktives Widget oder Gadget oder ein Element mit unabhängigem Inhalt enthalten. Das *aside*-Element repräsentiert einen Abschnitt einer Seite, der Inhalte enthält, die sich zwar auf den das *aside*-Element umgebenden eigentlichen Inhalt der Seite beziehen, aber als von ihm unabhängig betrachtet werden können. In Druckwerken werden derartige Abschnitte häufig als Seitenleisten dargestellt. Das Element kann für typografische Effekte wie herausgehobene Zitate oder Seitenleisten, für Werbung, für Gruppen von *nav*-Elementen und andere Inhalte verwendet werden, die als vom eigentlichen Inhalt der Seite getrennt betrachtet werden können.“[Sch11, S.43] Das neue *main*-Element ist zur Auszeichnung des Seitenhauptinhalts vorgesehen. Mit dieser Auszeichnung lässt sich z.B. mit Screenreadern direkt zum Hauptinhalt springen. Alle genannten Elemente sind in Listing 6 in ihrer vorgesehenen Reihenfolge abgebildet.

```
1  <body>
2    <header>
3      <nav>
4        <ul>
5          <li><a href="#link_1.html">Wiki</a></li>
6          ...
7        </ul>
8      </nav>
9    </header>
10   <main>
11   <article>
12     <h1>Ueberschrift</h1>
13     <p>Dies ist eine Beispiel HTML5-Seite</p>
```

```
14    </article>
15    <aside>
16      <section>
17        <h2>Kontakt</h2>
18        <ul>
19          <li><a href="link_1.html">Wiki</a></li>
20          ...
21        </ul>
22      </section>
23    </aside>
24    </main>
25    <footer>
26    </footer>
27  </body>
```

Listing 6: HTML5 Struktur Elemente

„Damit auch ältere Internet Explorer der Versionen 6-8 die neuen HTML5-Elemente darstellen können, kann ein kurzes JavaScript eingebunden werden. Am einfachsten ist es, dieses nicht auf dem eigenen Server vor zuhalten, sondern direkt von Google abzurufen. Dies hat überdies den Vorteil, dass es oft schon im Browser-Cache der Nutzer vorhanden ist. Der Aufruf erfolgt in einem *Conditional Comment*, der nur vom Internet Explorer kleiner als Version 9 (lt IE 9) verstanden wird. Alle andere Browser ignorieren dies als reinen Kommentar.“[Selc] In Listing 7 ist der beschriebene Rückfallmechanismus verdeutlicht.

```
1  <head>
2    <meta charset="utf-8">
3    <!--[if lt IE 9]>
4      <script src="//html5shim.googlecode.com/svn/trunk/html5.js"></
           script>
5    <![endif]-->
6    <title>HTML5-Seite mit Grundstruktur</title>
7  </head>
```

Listing 7: HTML5 Internet Explorer Fallback

Eingabefelder sind in den meisten Applikationen unabdingbar. Aus diesem Grund wurden in HTML5 eine vielzahl an *input*-Elementen hinzugefügt. So müssen keine komplizierten Workarounds mit JavaScript oder anderen Skriptsprachen mehr verwendet werden. „Das typische HTML5-Formular unterscheidet sich nicht fundamental von seinen in HTML 4.01 oder XHTML 1 geschriebenen Gegenstücken. Alle alten Formularelemente sind noch da und verhalten sich weitgehend wie bisher. Die Neuerungen

bestehen aus einigen neuen Funktionen, Attributen und APIs und aus einer ganzen Reihe neuer möglicher Werte für das *type*-Attribut des *input*-Elements.“[Krö11, S.176] In Listing 8 sind beispielhaft einige neue Werte des *type*-Attributs ausgeführt. Eingabefelder des Typs *tel, email* oder *url* sind vom Verständnis her simple Texteingabefelder. Als wichtige Besonderheit lässt sich bei *email*- und *url*-Feldern aber ihre eingebaute Validation nennen. Verwendet man die Validierungs-API werden nur korrekte URLs bzw. E-Mails zugelassen. Für ein *tel*-Feld gilt dies nicht. Außerdem wird auf Smartphones und Tablet-Geräten die angezeigte Bildschirmtastatur entsprechend des erwarteten Eingabetyps für eine optimale Eingabe angepasst dargestellt.(vgl. [Krö11, S.178]) So wird bei einer erwarteten E-Mail Adresse das @-Symbol direkt auf der Bildschirmtastatur als eigene Taste mit angezeigt, was normalerweise nicht der Fall ist. Einige der neuen Elementausprägungen besitzen noch zusätzliche Attribute die die Eingabemöglichkeit weiter einschränken und präzisieren können. Zu sehen in Zeile 5 von Listing 8 im *time*-Feld, bei dem die erwartete Zeit von 9:00Uhr bis 17:00Uhr nur einstellbar ist. *input*-Elemente können über das *required*-Attribut außerdem als Pflichtfeld markiert werden.

```
1 <body>
2   <input type="tel">
3   <input type="email">
4   <input type="url">
5   <input type="time"  min="09:00" max="17:00">
6   <input type="date"  required="required">
7 </body>
```

Listing 8: HTML5 *input*-Element

Für multimediale Inhalte auf Webseiten wurden der Spezifikation passende Elemente hinzugefügt. Das *canvas*-Element „[...] stellt eine Fläche zur Verfügung, auf die mittels JavaScript dynamische Bitmap-Grafiken gezeichnet werden können. So lassen sich Animationen erstellen, Diagramme zeichnen, eigene Interface-Elemente kreieren und Videos manipulieren“[Krö11, S.353] Für Audio und Video Inhalte wurden die gleichnamigen Elemente geschaffen.

2.2 CSS3

„Cascading Style Sheets (CSS) bieten mächtige Möglichkeiten, die Präsentation eines Dokuments oder einer Sammlung von Dokumenten zu beeinflussen. Offensichtlich ist CSS ohne irgendein Dokument ziemlich nutzlos, da es keine Inhalte zu präsentieren gäbe.“[ML05, S.1] CSS gehört mit zu den Kernsprachen des WWW. „Zuallererst ermög-

licht CSS eine wesentlich umfangreichere Gestaltung des Aussehens eines Dokuments, als HTML das jemals konnte, nicht einmal als die Präsentationselemente einen Großteil der Sprache ausmachten. Mit CSS kann für jedes Element eine eigene Text- und Hintergrundfarbe festgelegt werden. Jedes Element lässt sich zudem mit einem Rahmen versehen; der Platz um ein Element herum kann in der Größe verändert werden und es kann Einfluss auf die Groß- und Kleinschreibung genommen werden. Mit CSS können Sie außerdem bestimmen, wie fett bestimmte Textteile dargestellt werden sollen, welche Dekoration (z.B. Unterstreichungen) verwendet werden soll, wie groß die Abstände zwischen einzelnen Buchstaben und Wörtern sein sollen und ob der Text überhaupt angezeigt wird.“[ML05, S.4] Mit CSS ist es möglich das Aussehen der Webseite mit Bezug auf das verwendete Anzeigegerät anzupassen und entsprechend optimiert darzustellen.

Einbindung Um eine Webseite überhaupt mit CSS zu gestalten muss es innerhalb eines HTML Dokuments eingebunden sein. Dies ist auf drei verschiedene Arten möglich. Als Erstes mal das *link*-Element. „CSS benutzt dieses Tag, um Stylesheets mit dem Dokument zu verbinden.[...] Stylesheets, die nicht im HTML-Dokument selbst enthalten sind, sondern von außen eingebunden werden, bezeichnet man als externe Stylesheets[...].Um ein Stylesheet erfolgreich zu laden, muss sich ein *link* innerhalb des *head*-Elements befinden, darf aber nicht innerhalb eines anderen Elements wie etwa *title* stehen.“[ML05, S.14] In Listing 9 ist die Einbindung eines externen Stylesheets über das *link*-Element zu sehen.

```
1 <head>
2     <link rel="stylesheet" type="text/css" href="beispiel.css" />
3 </head>
```

Listing 9: Stylesheet Einbindung über *link*-Element

Eine andere Möglichkeit ist es das CSS direkt innerhalb eines *style*-Elements zu positionieren. Ein Element vom Typ „*style* sollte immer zusammen mit dem Attribut *type* verwendet werden. Im Fall eines eingebetteten CSS-Dokuments ist der korrekte Wert „*text/css*“, genau wie bei dem Element *link*.[...] Die Stildefinition zwischen den öffnenden und schließenden *style*-Tags bezeichnet man als Dokumenten-Stylesheet oder auch als eingebettetes Stylesheet[...].“[ML05, S.19] Listing 10 zeigt ein solches eingebettetes Stylesheet.

```
1 <head>
2   <title>Dokument mit Formatierungen</title>
3   <style type="text/css">
```

```
4       body { color: purple; background-color: #d8da3d; }
5     </style>
6   </head>
```

Listing 10: Stylesheet Einbindung über *style*-Element

Als letzte Möglichkeit kann man ein Stylesheet bzw Style-Informationen direkt einem HTML Element anhängen. „Durch das direkte Festlegen von Formaten, umgangsprachlich auch „Inline-Style“ genannt, gehen viele Vorteile verloren. Der Wartungsaufwand steigt während die Flexibilität sich verringert. Inline-Styles sind an ein Dokument gebunden und können nicht an zentraler Stelle bearbeitet werden.“[Selb] In Listing 11 wird dem *span*-Element ein „Inline-Style“gegeben.

```
1 <span style="font-size: small;">Text</span>
```

Listing 11: Stylesheet Einbindung in *html*-Element

Innerhalb von CSS-Dateien kann man wiederum mit der Direktive *@import* den Browser dazu zu bringen weitere Stylesheets nachzuladen und zu verwenden. Zu beachten ist, dass die *@import* Direktive vor allen anderen CSS Befehlen steht, bei den eingebetteten wie auch den externen Stylesheets.(vgl. [ML05, S.20])

Syntax Die Syntax von CSS ist denkbar einfach. Sie besteht im wesentlichen aus dem Selektor und dem Deklarationsblock. Ein Selektor entspricht in den häufigsten Fällen dem gleichnamigen HTML Element. Der Deklarationsblock wiederum besteht aus mindestens einer Deklaration.(vgl. [ML05, S.26]) „Eine Deklaration besteht dabei immer aus einer Eigenschaft, die mit einem Wert durch einen Doppelpunkt verbunden ist. Abgeschlossen wird die Deklaration durch ein nachgestelltes Semikolon. Auf den Doppelpunkt und das Semikolon kann eine beliebige Anzahl von Leerzeichen (also auch keines) folgen. Fast immer besteht der Wert aus einem einzelnen Schlüsselwort oder einer durch Leerzeichen getrennten Liste aus mehreren Schlüsselwörtern, die für die genannte Eigenschaft zulässig sind.“[ML05, S.28] Verwendet man in der Deklaration ungültige Eigenschaften oder Werte werden diese einfach ignoriert. Listing 12 zeigt die Syntax beispielhaft.

```
1 Selektor [, Selektor2, ...] {
2       Eigenschaft1: Wert1;
3       ...
4       EigenschaftN: WertN[;]
5 }
```

Listing 12: CSS3 Syntax Beispiel

Um sich wiederholende Design Regeln zusammenfassen zu können lassen sich Selektoren in CSS gruppieren. Dabei werden die zu gruppierenden Selektoren durch ein Komma von einander getrennt und dann die zu gestaltenden Eigenschaften mit den entsprechenden Werten genannt wie in Listing 13 zu sehen.

```
1 h1, h2, p {
2   color: black;
3 }
```

Listing 13: CSS3 Gruppierung

Selektoren CSS arbeitet wie HTML auch mit dem DOM. „Der erste Vorteil, der sich aus dem Verständnis dieses Modells ergibt, ist die Möglichkeit, Selektoren für Nachfahren (auch als Kontext-Selektoren bezeichnet) zu definieren. Die Definition von Selektoren für Nachfahren besteht aus dem Festlegen von Regeln, die nur für bestimmte hierarchische Strukturen gelten.“[ML05, S.48] Der erst genannte Selektor ist das Elternelement. Mittels eines Leerzeichens wird das Nachfahren Element vom Elternelement getrennt. In Listing 14 wird angegeben, dass jedes *strong*-Element innerhalb eines *h1*-Elements eine besondere Formatierung erhält. Konkret bedeutet das in diesem Fall, dass die Schriftgröße 14pt und die Schriftgewichtung, welche die Dicke und Stärke festlegt, *bold* sein soll. Zu beachten ist, dass so sämtliche Nachfahren des *h1*-Elements entsprechend formatiert werden. Es gibt auch die Möglichkeit die Formatierung nur auf einen direkt Nachfahren, ein sogenanntes Kindelement, zu beschränken. Um dies zu erreichen werden Eltern- und Kindelement nicht durch ein Leerzeichen von einander getrennt, sondern durch das Größer-als-Zeichen (>).

```
1 h1 strong {
2   font-size:  14pt;
3   font-weight: bold;
4 }
```

Listing 14: CSS3 Selektoren für Nachfahren

Das gleiche Verfahren kann auch auf Nachbarelemente angewandt werden. Nur wird hierbei ein anderes Kombinatorenzeichen benötigt, nämlich das Plus (+).
„Zusätzlich zu den rohen Dokument-Elementen gibt es noch zwei weitere Arten von

Selektoren: Klassenselektoren und ID-Selektoren, mit denen sich Stildefinitionen unabhhängig von den Elementen eines Dokuments zuweisen lassen. Diese Selektoren können eigenständig oder zusammen mit Elementselektoren verwendet werden. Allerdings ist es hierfür notwendig, die Dokumentteile entsprechend zu markieren. Die Benutzung dieser neuen Selektoren erfordert daher [...] Voraussicht und Planung.“[ML05, S.34ff] Des weiteren muss die HTML Auszeichnung angepasst werden sollten Klassen- und/oder ID-Selektoren verwendet werden. Konkret bedeutet dies, dass jedem HTML Element ein *class*- und/oder *id*-Attribut hinzugefügt werden muss sofern es von den CSS Regeln beachtet werden soll. Zu beachten ist, dass eine vergebene ID im gesamten HTML Dokument einzigartig sein muss, wohingegen eine vergebene Klasse auf mehrere HTML Elemente und auf ein HTML Element auch mehrere Klassen angewandt werden können. (vgl. [w3S]) Listing 15 zeigt einen solchen ID-Selektor, einen Klassenselektor und einen Klassenselektor der nur auf *p*-Elementen gilt die das *class*-Attribut besitzen.

```
1  <div id="ID">
2    <p class="Klasse"></p>
3  </div>
4
5  div#ID {
6    text-align: center;
7    color:      red;
8  }
9  .Klasse {
10   text-align: center;
11   color:      red;
12 }
13 p.Klasse {
14   text-align: center;
15   color:      red;
16 }
```

Listing 15: CSS3 Klassen- und ID-Selektoren

Pseudoklassen Neben den genannten gibt es noch einen weiteren Typ von Selektoren. Die sogenannten Pseudoselektoren für die Pseudoklassen. „Mit diesen Selektoren [... kann man] Stildefinitionen auf Strukturen zuweisen, die nicht unbedingt im Dokument vorkommen müssen, oder auch [... Pseudo]klassen, die vom Zustand bestimmter Elemente oder sogar vom Zustand des Dokuments selbst abhängen. Anders gesagt, werden die Stile, basierend auf etwas anderem als der Struktur des Dokuments, auf Teile des Dokuments angewendet.“[ML05, S.53ff] Um beispielsweise einen Link in ei-

nem Dokument farblich zu markieren sobald er besucht wurde wäre eigentlich für jeden Zustand des Links eine eigene Klasse nötig. Diese Klasse müsste sich ständig ändern je nach dem ob der Nutzer den Link schon besucht hat oder nicht. Um diesen Umstand zu verhindern existieren die Pseudoselektoren und Pseudoklassen. Im Fall eines *a*-Elements werden die Pseudoklassen *link* und *visited* bereitgestellt um den Effekt zu erzeugen. Diese Klassen werden dem *a*-Element durch CSS selber hinzugefügt bzw. wird ein *a*-Element, das ein *href*-Attribut besitzt und noch nicht besucht wurde mit der Pseudoklasse *link* versehen. Die Pseudoklasse *visited* bezieht sich auf *a*-Elemente mit *href*-Attribut die schon besucht wurden. Listing 16 zeigt ein solches *a*-Element und die beiden Pseudoselektoren.

```
1  <a href="http://www.example.com">Example.com</a>
2
3  /* unvisited link */
4  a:link {
5      color: #FF0000;
6  }
7
8  /* visited link */
9  a:visited {
10     color: #00FF00;
11 }
```

Listing 16: CSS3 Pseudoklassen und -selektoren

Box-Modell „CSS geht davon aus, dass jedes Element eine oder mehrere rechteckige Boxen erzeugt, die Element-Boxen genannt werden.[...] Im Kern besitzt jede Box einen Inhaltsbereich (content area). Dieser wird umgeben von optionalen Innenabständen (padding), Rahmen (borders) und Außenabständen (margins). Diese Teile werde als optional angesehen, weil sie alle eine Breite von null Einheiten haben können, also sozusagen nicht vorhanden sind.“[ML05, S.167] „Da eine Box aus mehreren Komponenten bestehen kann, werden für die einzelnen Bereiche verschiedene Kantenbezeichnungen definiert. Als Innen- oder Inhaltskante wird die Strecke bezeichnet, die den Inhaltsbereich einer Box umfasst. Das ist der Bereich, der durch den Inhalt oder die Eigenschaften *width* und *height* festgelegt wurde. Der Bereich einer Box, der von den Innenkanten umgeben ist, wird auch *content box* (Inhaltsbox) genannt. Die Kanten, die eine Box mitsamt Innenabstand umfassen, werden Polsterungskanten genannt. Der von diesen Kanten umfasste Bereich wird *padding box* (Polsterungsbox) genannt. Besitzt eine Seite keinen Innenabstand, so ist die Polsterungskante mit der Innenkante

identisch. Die Rahmenkanten umgeben eine Box mit deren Rahmen. Der durch diese Kanten abgesteckte Bereich wird *border box* (Rahmenbox) genannt. Besitzt eine Box keinen Rahmen, so ist die Rahmenkante mit der Polsterungskante identisch. Eine vollständige Box wird durch die Außenkanten definiert. Die Außenkante umfasst eine Box mitsamt Außenabständen, also die *margin box*. Sind für eine Box keine Außenabstände definiert, so ist die Außenkante mit der Rahmenkante identisch.“[Sela]
Die Gesamtbreite eines Elements definiert sich dadurch als Summe aus Breite, linkem und rechtem Innenabstand, linkem und rechtem Rahmen und dem linkem und rechtem Außenabstand. Für die Gesamthöhe verhält sich die Berechnung analog. Abbildung 2 zeigt das beschriebene CSS-Box-Modell.

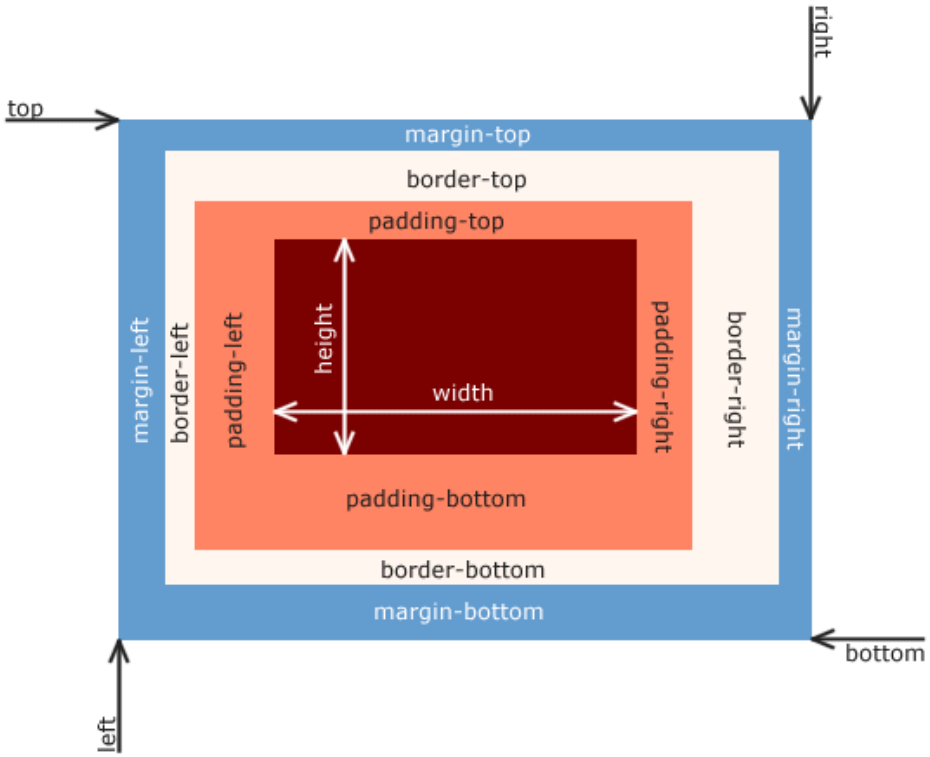

Abbildung 2: CSS-Box-Modell[Wika]

Spezifische Stylesheets „Dank der Mechanismen in CSS und HTML [... lassen sich] beliebige Stylesheets auf bestimmte Medien beschränken. In HTML-basierten Stylesheets geschieht dies mit Hilfe des Attributs *media* und gilt sowohl für *link*- wie auch *style*-Elemente. Das Attribut *media* akzeptiert entweder die Angabe eines einzelnen Mediums oder eine durch Komma getrennte Liste von Werten.“[ML05, S.434ff] Die Angaben in Listing 17 bewirken eine gesonderte Formatierung für ein *print* Medium.

```
1 <link rel="stylesheet" type="text/css" media="print" href="article-
```

```
print.css">
```

Listing 17: CSS3 medienspezifisches Stylesheet

Ein vorhandenes Stylesheet ohne Medieninformationen gilt für alle Medien. Fernen lassen sich die Medien auch innerhalb des Stylesheets näher beschreiben durch den *@media*-Block. Diesen *@media*-Blöcken können neben den Medientypen auch noch weitere Bedingungen hinzugefügt werden. In Listing 18 wird das Element mit der ID *inhalt* auf eine Breite von *800px* festgelegt. Ist das verwendete Ausgabemedium jedoch ein Bildschirm und hat nur eine Gesamtbreite von *1024px* wird das Element mit der ID *inhalt* auf eine Breite von nur noch *600px* und das *aside*-Element gar nicht mehr angezeigt.

```
1  #inhalt {
2    width: 800px;
3  }
4
5  @media screen and (max-width: 1024px) {
6    #inhalt {
7      width: 600px;
8    }
9    aside {
10     display: none;
11   }
12 }
```

Listing 18: CSS3 eigenschaftsspezifisches Stylesheet

2.3 JavaScript

Historie JavaScript wurde 1995 von Netscape entwickelt, lizenziert und eingeführt. Um der Sprache von Anfang an einen Standardcharakter zu geben wurde die Organisation European Computer Manufacturers Association (ECMA) hinzugezogen. Die ECMA veröffentlichte unter dem Namen ECMAScript einen, auf Netscapes JavaScript Spezifikation basierenden, Industriestandard der Sprache. Da JavaScript somit eine proprietäre Sprache von Netscape ist, hat Microsoft seine eigene Variante, mit Namen JScript, veröffentlicht. JScript implementiert JavaScript vollständig, besitzt allerdings auch noch Zusatzfunktionen wie z.B. Zugriff auf das Dateisystem und das Betriebssystem Windwos. Mit Version 1.5 von JavaScript erhielt das DOM Einzug in die Implementierung. Da jeder Browser seinen eigenen JavaScript Interpreter besaß, war es kaum Möglich einheitlichen Code für alle Browser zu entwickeln. Es musste auf alle Eventualitäten geachtet werden. Um diesem Missstand entgegen zu wirken wurde das

W3C hinzugezogen um einen einheitlichen Sprachstandard zu etablieren. Jedoch entwickelte das W3C keinen konkreten JavaScript-Standard, sondern eine Schnittstelle - das erwähnte DOM. Die aktuelle JavaScript Version von 2010 ist 1.8.5 und ECMAScript liegt in Version 5.1 seit Juni 2011 vor. (vgl. [Seld])

Sandbox-Prinzip JavaScript wird innerhalb eines Browser in einer sogenannten Sandbox ausgeführt. Das bedeutet es liegt in einem abgesichertem Speicherbereich, aus welchem es keinen Zugriff auf Objekte außerhalb des Browsern hat. Eine Ausnahme ist der Lesezugriff auf Dateien die mittels des *input*-Elements von einem Nutzer selbst hoch geladen werden. Des weiteren kann mit JavaScript auch nicht ohne weiteres auf bestimmte sicherheitskritische Funktionen des ausführenden Browser zugegriffen werden. Um beispielsweise das Browserfenster zu schließen, Symbolleisten ein- und auszublenden oder zugriff auf die Seitenhistorie zu erlangen sind Nutzereingaben nötig. (vgl. [Wikc])

Objektorientierung „JavaScript gehört zu den sogenannten objektorientierten Programmiersprachen (oder, um genauer zu sein, zu den objektbasierten Sprachen). Das Konzept der objektorientierten Programmierung (OOP) wird im Folgenden sehr stark vereinfacht erklärt.[...] In JavaScript ist (mit Ausnahme der Variablen) alles, worauf man zugreift, ein Objekt. Ein Objekt ist der Versuch, die reale Welt in eine Programmiersprachenumgebung abzubilden. Ein Standardbeispiel für Objekte ist etwa ein Auto. Das Auto an sich (als abstrakter Begriff) kann als Objekt angesehen werden, ein einzelnes Auto wird als Instanz des Objekts Auto bezeichnet.“[Wen08, S.93] Ein Auto bzw. ein Objekt generell lässt sich durch Parameter näher beschreiben. Diese Parameter gibt es in zwei Ausführungen. Die Eigenschaften und die Methoden. Bei einer Eigenschaft handelt es sich im Grunde um eine Variable die einen festen Bezug zum Objekt besitzt. Eigenschaften können gelesen und gesetzt werden. Ein Auto hat beispielsweise als Eigenschaft die Anzahl der Türen oder die Motorleistung. Methoden auf der anderen Seite müssen nicht immer einen Informationswert zurückgeben. Bei dem Objekt Auto könnte es eine Methode *tunen()* geben die dann Einfluss auf die Eigenschaft Motorleistung nimmt.
Im Kontext der Webentwicklung mit JavaScript existieren einige feste Objekte die zu jederzeit zur Verfügung stehen. Da wäre zum einen das *window*-Objekt, dass, wie der Name schon erahnen lässt, das aktuelle Browserfenster repräsentiert. Über die Eigenschaften und Methoden lassen sich Informationen zum Browserfenster erhalten und beispielsweise neue Fenster öffnen. Das *document*-Objekt bildet den Inhalt eines Browserfensters ab. Es stellt das Ausgangsobjekt für das DOM dar. Es steht in der Hierarchie

direkt unter dem *window*-Objekt. Neben diesen Objekten existieren noch weitere um den Browserkontext möglichst genau innerhalb einer JavaScript Anwendung verfügbar zu machen.(vgl. [Sele])

Sprachelemente JavaScript wird mit dem Unicode Zeichensatz geschrieben. Die 16-Bit-Codierung bei Unicode enthält fast sämtliche Zeichen der Schriftsprachen der Welt. Die Nutzung von Unicode trägt wesentlich zu der Internationalisierung bei. Bei der Groß- und Kleinschreibung unterscheidet JavaScript eindeutig. So müssen alle Schlüsselwörter und Bezeichner immer in der selben vorgegebenen Schreibweise geschrieben werden, damit sie vom JavaScript Interpreter korrekt behandelt werden. Whitespace, Tabulatoren und Zeilentrenner werden vom Interpreter komplett ignoriert und können deshalb zur visuellen Strukturierung des Programmcodes genutzt werden. Dadurch kann der Programmcode leicht leserlich und verständlich formatiert werden ohne das dadurch die Logik verletzt wird. Das, aus anderen Programmiersprachen bekannte, Semikolon am Ende einer Anweisung ist in JavaScript nicht zwingend erforderlich. Eine Anweisung ist im Normalfall mit dem Ende der Zeile abgeschlossen. So muss ein Semikolon lediglich gesetzt werden, wenn sich mehr als eine Anweisung in der Zeile befinden oder eine Anweisung über mehrere Zeilen hinweg formuliert wurde. Fehlende Semikola werden vom Interpreter selbst gesetzt, was in bestimmten Fällen aber zum Bruch der Logik führen kann wie in Listing 19 zu sehen.(vgl. [Fla07, S.15ff])

```
1  // kein Interpretierungsfehler
2  a = 3
3  b = 4;
4
5  // korrekte Schreibweise in einer Zeile
6  a = 3; b = 4;
7
8  // Interpreter setzt Semikolon automatisch
9  return
10 true;
11 // falsche Interpretation anschliessend
12 return;
13 true;
```

Listing 19: JavaScript Logikbruch Semikolon

Des weiteren gibt es einige Literale in JavaScript. „Ein Literal ist ein Datenwert, der direkt in einem Programm vorkommt. Literale können zum Beispiel folgendermaßen aussehen:“[Fla07, S.18]

```
1 12                  // Die Zahl zwoelf
2 1.2                 // Die Zahl eins komma zwei
3 "Hallo Welt"        // Ein Text-String
4 'Hi'                // Noch ein String
5 true                // Ein Boolescher Wert
6 false               // Der andere Boolesche Wert
7 null                // Kein Objekt vorhanden
```

Listing 20: JavaScript Literale

Im offiziellen Standard ECMAScript existieren außerdem Literale, die zur Initialisierung von Arrays und Objekten dienen. Neben den genannten Sprachelementen gibt es noch die Bezeichner. Ein Bezeichner ist ein Name in JavaScript. Er dient dazu Variablen, Funktionen und einige Schleifen-Marker zu benennen. Ein Bezeichner unterliegt gewissen Regeln. Zum einen muss das erste Zeichen ein Buchstabe, ein Unterstrich (_) oder ein Dollar-Zeichen ($) sein. Der restliche Bezeichner darf aus Buchstaben, Ziffern, Unterstrichen oder Dollar-Zeichen bestehen. Eine weitere Regel legt fest, dass ein Bezeichner nicht wie ein Schlüsselwort heißen darf. Tabelle 2 listet die Schlüsselwörter von JavaScript auf.(vgl. [Fla07, S.19])

break	do	if	switch	typeof
case	else	in	this	var
catch	false	instanceof	thorw	void
continue	finally	new	true	while
default	for	null	try	with
delete	function	return		

Tabelle 2: JavaScript Schlüsselwörter

Datentypen und Werte Werte zur Berechnung werden in Variablen mit bestimmten Datentypen gesichert. Dazu unterstützt JavaScript einige primitive Datentypen wie Zahlen, Text und boolesche Werte. Außerdem gibt es einen zusammengesetzten Datentypen, das Objekt, mit dem eine Sammlung von verschiedenen Werten dargestellt wird. So kann ein Objekt beispielsweise auch weitere Objekte beinhaltet. Sind die Werte in geordneter nummerierter Reihenfolge nennt man das Objekt Array. Die Zahlen sind der einfachste Datentyp. Mit ihm werden Zahlen dargestellt, egal ob Ganzzahlig oder Gleitkommazahlen. JavaScript behandelt jede Zahl als Gleitkommazahl und stellt diese im 64-Bit-Gleitkommaformat nach IEEE 754 dar.
Integer-Literale werden in JavaScript als eine Folge von Ziffern geschrieben. Mit diesem

Zahlenformat lassen sich alle ganzen Zahlen von einschließlich -2^{53} bis 2^{53} darstellen. Ein Hexadezimal-Literale wird mit einem *0x* oder *0X* begonnen gefolgt von einer Hexadezimalzahl. Bei Gleitkomma-Literalen wird der ganzzahlige Teil durch einen Punkt vom Bruchteil der Zahl getrennt. Des weiteren können sie in der Exponentenschreibweise geschrieben werden. Um Text darzustellen wird der Datentyp String verwendet. Ein String ist eine Folge von Unicodezeichen in einzelnen oder doppelten Anführungszeichen. Damit spezielle Zeichen innerhalb von Strings benutzt werden können, gibt es eine *Escape-Sequenz* in Form eines Backslash (\). So lassen sich Tabulatoren oder Zeilenumbrüche in einem String darstellen. Weiter gibt es die beiden booleschen Werte *true* und *false*. Sie werden zumeist bei Vergleichen als Wahrheitswert verwendet. Die Funktionen sind ein besonderer Datentyp. Funktionen sind ausführbarer Programmcode. Sie werden einmal geschrieben und können dann beliebig oft benutzt werden. Einer Funktion lassen sich Argumente und Parameter übergeben, die dann in der Berechnung zur Verwendung kommen. Eine Funktion wird durch das Schlüsselwort *function* eingeleitet, gefolgt von einem optionalen Bezeichner und einer durch Kommata getrennten Liste von Argumenten und Parametern, die in runden Klammern eingeschlossen ist. Dann gibt es noch die Objekte, die eine Sammlung von nicht nummerierten Eigenschaften darstellen. Um auf eine Eigenschaft eines Objekts zuzugreifen wird an den Objektbezeichner ein Punkt an gehangen und dann der Name der Eigenschaft. Im Gegensatz dazu kann auf die Eigenschaften eines Arrays nur über den Index zugegriffen werden. Das Schlüsselwort *null* ist ein spezieller Wert. Er steht für *kein Wert* und wird häufig bei der Überprüfung von Variablen und Objekten verwendet.(vgl. [Fla07, S.22ff])

Variablen „Eine Variable ist ein Name, der mit einem Wert verbunden ist. Man spricht davon, dass die Variable den Wert speichert oder enthält. Variablen ermöglichen es [...], Daten in [...] Programmen zu speichern und zu bearbeiten.“[Fla07, S.51] Ein grundlegender Unterschied von JavaScript zu anderen Programmiersprachen besteht darin, dass Variablen nicht typisiert werden müssen. So kann man einer JavaScript Variablen ohne weiteres erst einen Zahlenwert zuweisen und später eine Zeichenkette. Diese Art von Typisierung nennt man dynamische Typisierung(Loose Typing), da erst zur Laufzeit der tatsächliche Datentyp der Variablen feststeht. In anderen stark typisierten Sprachen wie C oder Java sind solche Konstrukte nicht zulässig, da einer Variable auch nur ein Wert, der ihrem Datentyp entspricht, zugewiesen werden kann. Eine Variable wird in JavaScript mit dem Schlüsselwort *var* und einem Bezeichner deklariert. Das Schlüsselwort *var* kann auch weggelassen werden, dann wird es vom Interpreter implizit gesetzt. So deklarierte Variablen werden automatisch als globale Variabeln deklariert. Soll eine Variable jedoch nur innerhalb eines Funktionsblocks Gül-

tigkeit haben ist die verwenden des Schlüsselworts *var* unerlässlich. Aus diesem Grund sollte das Schlüsselwort immer zur Deklaration von Variablen genutzt werden.
Initialisiert werden Variablen durch die Zuweisung eines Wertes mittels des Zuweisungsoperator (=). Dies kann auch mit der Deklaration in einem Schritt zusammengefasst werden.(vgl. [Fla07, S.52ff])

Operatoren „Durch Operatoren wird eine gewisse Anzahl von Variablen miteinander kombiniert. Beispiele für Operatoren sind die Grundrechenarten. Durch den Plus-Operator werden zwei Zahlen miteinander kombiniert, und als Ergebnis erhält man die Summe dieser beiden Zahlen. Man unterscheidet - auch je nach Typ der beteiligten Variablen - verschiedene Arten von Operatoren.“[Wen08, S.69] Mit den arithmetischen Operatoren lassen sich numerische Variablen miteinander verknüpfen und berechnen. Durch die dynamische Typisierung sollte man stets sicher sein das die Operanden auch Zahlen Variablen sind und keine String Variablen. Zu den arithmetischen Operatoren gehören die Addition (+), Subtraktion (-), Multiplikation (*), Division (/), die Restwertberechnung Modulo (%) sowie die Negation (-). Außerdem sind noch zwei Operatoren zur In- (++) und Dekrementation (–) vorhanden. Der Plus (+) Operator hat noch eine zusätzliche Funktion. Er kann neben der arithmetischen Addition auch zur Zeichenverkettung verwendet werden.
Neben den arithmetischen Operatoren stehen in JavaScript auch Operatoren zur Verarbeitung von booleschen Werten bereit. Mit ihnen lassen sich Wahrheitswerte verknüpfen und vergleichen. Mit dem logischen UND (&&) beispielsweise wird ein boolescher Ausdruck darauf geprüft ob beide Operanden als Wert *true* liefern. Ist dies der Fall wird der Wert *true* für den booleschen Ausdruck zurück gegeben, ansonsten der Wert *false*. Das logische ODER (||) prüft einen booleschen Ausdruck im Grunde genauso wie ein logisches Und mit der Abweichung, dass bei einem logischen Oder auch der Wert *true* für den gesamten booleschen Ausdruck zurück geliefert sollte nur der erste Operand den Wert *true* haben. Weiter gibt es boolesche Vergleichsoperatoren die bei Zahlenwerten aber auch mit Strings verwendet werden können. Zu ihnen gehören der Gleichheitsoperator (==), Ungleich (!=), Größer als (>), Kleiner als (<) und zuletzt Größer gleich (>=) und Kleiner gleich (<=).(vgl. [Wen08, S.71ff])

Schleifen Um eine Anweisung innerhalb von JavaScript mehrfach ausführen zu lassen bietet JavaScript Schleifenkonstrukte an. Darunter zählen die For-Schleife, die Do-While-Schleife und die While-Schleife.Jede Schleifenart hat in bestimmten Situationen Vor- und Nachteile gegenüber den anderen beiden Arten.
Eine For-Schleife wird mit einem Startwert initialisiert und enthält außerdem eine Ab-

bruchbedingung sowie eine Befehlsfolge die nach jedem Schleifendurchlauf ausgeführt wird. Listing 21 zeigt die Syntax einer For-Schleife. Der Startwert ist auch die sogenannte Zählervariable, welche mit jedem Durchlauf durch die Befehlsfolge verändert wird. Die Abbruchbedingung überprüft diese Zählervariable und beendet die Schleife sollte die Bedingung nicht mehr zutreffen. Eine For-Schleife akzeptiert auch mehr als einen Startwert, welche durch Kommata von einander getrennt werden.(vgl. [Wen08, S.75f])

```
1 for (Startwert; Abbruchbedingung; Befehlsfolge) {
2   //Anweisung
3 }
```

Listing 21: Syntax For-Schleife

For-Schleifen eignen sich vor allem, wenn man im vor hinein weiß wie oft die Anweisung wiederholt werden muss. Ist dem nicht der Fall bietet sich die Do-While-Schleife an. Diese Art der Schleifen führt eine Anweisung mindestens einmal aus und überprüft dann zum ersten mal die Abbruchbedingung. Listing 22 zeigt die Syntax einer Do-While-Schleife.

```
1 do {
2   //Anweisung
3 } while (Abbruchbedingung);
```

Listing 22: Syntax Do-While-Schleife

In einigen Fällen kann die definitive Ausführung, vor der ersten Überprüfung der Bedingung, einer Anweisung nicht gewünscht sein. Dann sollte eine While-Schleife verwendet werden. Sie ist fast identisch der Do-While-Schleife. Ihr unterschied besteht darin, dass die Abbruchbedingung als erstes überprüft wird und dann, sollte die Bedingung zutreffen, die Anweisung ausgeführt wird. So kann verhindert werden, dass die Anweisung fälschlicherweise ausgeführt wird obwohl bestimmte Zuweisungen oder Funktionsaufrufe noch nicht durch geführt wurden. Das Listing 23 zeigt die Syntax der While-Schleife.

```
1 while (Abbruchbedingung) {
2   //Anweisung
3 }
```

Listing 23: Syntax While-Schleife

Für den Fall das der aktuelle Schleifendurchgang oder die gesamte Schleife vorzeitig beendet werden soll gibt es die beiden Schlüsselwörter *break* und *continue*. Die *break*

Anweisung bewirkt, dass die gesamte Schleife beendet und das Programm nach der Schleife fortgeführt wird. Möchte man jedoch nur den aktuellen Schleifendurchgang beenden ist die *continue* Anweisung das Mittel der Wahl. Beide Anweisungen sind nur innerhalb von Schleifen gültig und erzeugen außerhalb dieser einen Syntaxfehler.(vgl. [Fla07, S.103f])

Kontrollstrukturen Um den Programmablauf anhand von Fallunterscheidungen steuern zu können sind von JavaScript die *if*-Anweisungen vorgesehen. Listing 24 zeigt die Syntax einer *if-else*-Anweisung. Ist die Bedingung wahr wird die Anweisung ausgeführt. Sofern eine *else* Anweisung vorhanden ist - sie ist optional - würde die folgende Anweisung ausgeführt werden.(vgl. [Wen08, S.80])

```
1 if (Bedingung) {
2   //Anweisung
3 } else {
4   //Anweisung
5 }
```

Listing 24: Syntax If-else-Anweisung

Um den Programmcode übersichtlich zu halten existiert noch eine *switch* Anweisung, die als eine Art Zusammenfassung für mehrere *if* Anweisungen zu verstehen ist. Listing 25 beschreibt die Syntax einer *switch* Anweisung. Zu beachten ist, dass in einer *switch* Anweisung eine *break* Anweisung als Abbruchbedingung unabdingbar ist. Lässt man sie weg würde nach Ausführung der Anweisung des zuständigen *case* Blocks eventuell direkt der nächste passende *case* Block mindestens aber der *default* Block der *switch* Anweisung ausgeführt werden.

```
1  switch (Ausdruck) {
2    case Wert1:
3      //Anweisung
4      break;
5    case WertN:
6      //Anweisung
7      break;
8    default:
9      //Anweisung
10 }
```

Listing 25: Syntax Switch-Anweisung

Einbindung Um ein in JavaScript geschriebenes Skript innerhalb eines HTML Dokuments verwenden zu können muss es erst einmal in dieses eingebunden werden. Eine Möglichkeit ist es, dass Skript als separate Datei in das *head*-Element des HTML Dokuments einzubinden. Listing 26 zeigt das *script*-Element mit dem *src*-Attribut. Dieses Attribut verweist auf das externe Skript.

```
1 <script src="script.js" type="text/javascript"></script>
```

Listing 26: JavaScript Einbindung als separate Datei im *head*-Element

Die andere Möglichkeit besteht darin ein Skript direkt in das HTML Dokument zu schreiben und mit einem *script*-Element zu umschließen. Dies muss auch nicht zwingend im *head*-Element passieren, sondern kann auch an anderer Stelle im Dokument sein. Das ist je nach Anwendungsfall unterschiedlich. Listing 27 zeigt diese Möglichkeit.(vgl. [, S.])

```
1 <script type="text/javascript"></script>
```

Listing 27: JavaScript Einbindung in *script*-Element

Document Object Model Das DOM ist eine Schnittstelle um mit Skriptsprachen auf die Struktur eines HTML Dokuments Einfluss nehmen zu können. Spezifiziert wurde das DOM vom W3C um eine einheitliche Funktionsweise zu ermöglichen. Das DOM ist wie ein umgedrehter Baum aufgebaut. Jedes HTML Element und auch Texte die nicht von HTML Elementen umschlossen sind werden im DOM als Knoten(engl. node) dargestellt. HTML Elemente, die innerhalb eines anderen HTML Elements liegen, werden im DOM als Kindknoten(engl. child nodes) eingegliedert. Dadurch ist im DOM auch eine klare Hierarchie gegeben.(vgl. [Wen08, S.350])
Jeder Knoten im DOM beinhaltet zum einen Informationen über sich selbst und zum anderen Informationen über seinen Elternknoten und seine Kinderknoten. JavaScript hat dafür eigene Eigenschaften je Knoten definiert, mit denen auf diese Informationen zugegriffen werden kann. Mit den Eigenschaften *firstChild* und *lastChild* erhält man eine Referenz auf den ersten bzw. letzten Kindknoten des aktuellen Knotens. *nextSibling* und *previousSibling* liefern eine Referenz auf den nächsten bzw. vorherigen Kindknoten. *parentNode* ermöglicht den Zugriff auf den Elternknoten und die beiden Eigenschaften *nodeName* und *nodeType* sind zur bestimmung des Knoten selbst zuständig. Listing 28 zeigt eine in HTML implementierte Tabelle die in das entsprechende DOM umgewandelt wird, welches in Abbildung 3 zu sehen ist.

```
<table>
  <thead>
    <tr>
      <th>Produkt</th>
      <th>Preis</th>
    </tr>
  </thead>
  <tbody>
    <tr>
      <td>XYZ</td>
      <td>50,00</td>
    </tr>
  </tbody>
</table>
```

Listing 28: DOM5 Beispiel Definition

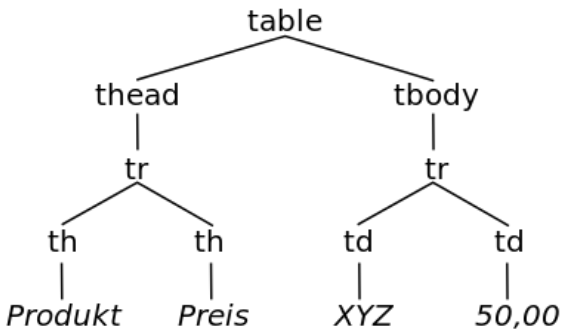

Abbildung 3: DOM Beispielbaum

Selbstverständlich bietet das DOM die Möglichkeit der Modifizierung. So lassen sich Knoten nicht nur ändern, sondern auch entfernen und neu hinzufügen. Außerdem ist auf Grund der Baumstruktur ein Löschen eines Knotens möglich ohne die Integrität des Baumes zu zerstören.

//Schnittstelle zum HTML Aufbau, W3C Spezifikation unterschieldich implementiert, Knoten Beziehungen, Verarbeitung des DOM, Generierung von HTML durch Serialisierung, Listing 28 beschreiben und zur Baumstruktur hinleiten

Ereignisse Interaktive JavaScript Applikationen kommunizieren mit dem Browser über Events. Der Browser erzeugt für eine vielzahl von Nutzeraktionen Events, welche dann wiederum im Applikationscode abgefangen und verarbeitet werden können.

In der DOM Level 0 Spezifikation werden Events lediglich an die Elemente verteilt an denen sie auftreten. Ist an diesem Element dann ein Event-Listener registriert wird dieser ausgeführt. DOM Level 2 hat die sogenannte Event-Propagation eingeführt. Diese Event Verteilung ist in drei Phasen aufgeteilt. Das Abfangen, dabei werden Events vom Document-Objekt den Dokumentbaum hinunter gereicht bis sie am Zielelement angelangt sind. Besitz ein Vorgänger Element im Dokumentbaum allerdings einen abfangenden Event-Listener wird auch dieser ausgelöst. Als nächstes wird löst das Event den Listener am Zielelement aus, was der DOM Level 0 Spezifikation gleicht. Die dritte Phase ist das sog. *bubbling*. Hierbei steigt das Event wie ein Bläschen in der Dokument-Hierarchie zum Document-Objekt auf. Das Aufsteigen eines Events ist aber je nach Event unterschiedlich. Einige steigen auf andere nicht. Ein Event mit dem ein Formular beispielsweise abgeschickt wird muss nicht weiter im Dokumentbaum nach oben propagiert werden. Mausklick Events hingegen können für das gesamte Dokument sinnvoll verarbeitet werden und werden daher immer im Dokument aufsteigen.
Ein Event-Listener kann auf drei unterschiedliche Arten an ein HTML Element gebunden werden. Eine Möglichkeit ist den Event-Listener direkt an das HTML Element mittels eines entsprechenden Event Attributs zu binden. So zu sehen in Zeile 1 in Listing 29.

```
1 <input type="button" name="b1" value="Drueck mich"
2                              onclick="alert('Button gedrueckt!');">
3
4 document.b1.onclick = function() { alert('Button gedrueckt!'); };
5
6 document.b1.addEventListener("click", click, false);
7 function click() { alert('Button gedrueckt!'); };
```

Listing 29: JavaScript Event-Handler Beispiek

Da HTML statisch ist, ist auch der Event Listener in dem Fall statisch. Komplexe Applikationen verlangen aber eine dynamische Bindung von Listenern. Aus diesem Grund wurden mit dem DOM Level 2 zwei weitere Möglichkeiten einen Event-Listener zu registrieren standardisiert. So kann der Listener einerseits einfach einer Eigenschaft des DOM Objekts zugewiesen werden. Daraus resultiert ein klar strukturierter Code im HTML und JavaScript. Die Wartbarkeit wird außerdem verbessert. Andererseits lässt sich ein Listener über eine Objekt Methode binden. Diese Methode erwartet drei Argumente. Das erste bestimmt das Event auf welches der Listener reagieren soll. Das zweite Argument ist die Funktion die beim Eintreten des Events ausgeführt werden soll. Das dritte Argument, ein Boolescher Wert, legt fest ob der Event-Listener das Event nur abfängt wenn es direkt am Element oder dessen Kind Elementen auftritt.

Dann muss dieser Wert *false* sein. Ansonsten kann der Event-Listener auch Events abfangen die im Geltungsbereich ihm über geordnet sind. Diese beiden Möglichkeit sind in Listing 29 ab Zeile 4 zu sehen. Nachfolgend ist eine Übersicht einiger wichtiger Events.

- onabort (bei Abbruch)
- onblur (beim Verlassen)
- onchange (bei erfolgter Änderung)
- onclick (beim Anklicken)
- ondblclick (bei doppeltem Anklicken)
- onerror (im Fehlerfall)
- onfocus (beim Aktivieren)
- onkeydown (bei gedrückter Taste)
- onkeypress (bei gedrückt gehaltener Taste)
- onkeyup (bei losgelassener Taste)
- onload (beim Laden einer Datei)
- onmousedown (bei gedrückter Maustaste)
- onmousemove (bei weiterbewegter Maus)
- onmouseout (beim Verlassen des Elements mit der Maus)
- onmouseover (beim Überfahren des Elements mit der Maus)
- onmouseup (bei losgelassener Maustaste)
- onreset (beim Zurücksetzen des Formulars)
- onselect (beim Selektieren von Text)
- onsubmit (beim Absenden des Formulars)
- onunload (beim Verlassen der Datei)

AJAX // AJAX

jQuery

// jQuery Bibliothek beinhaltet Elementselektion, Funktionen zum DOM, Animationen und Effekte, AJAX Funktionalitäten

// Selektoren

// Ereignisse - unterschiede zum JS Standard bei der Definierung, Einfachheit

// Übersicht der wichtigsten Funktionen zu Events

- .bind – Handler an Event binden
- .on – Handler an Event binden
- .blur – Ereignis, wenn ein Element den Fokus verliert
- .click – Klick mit der Maustaste
- .dbclick – Doppelklick mit der Maustaste
- .hover – Mauszeiger bewegt sich über ein Element
- .mousemove – Mauszeiger bewegt sich in einem Element
- .keypress – eine Taste der Tastatur wird gedrückt
- .keyup – eine Taste der Tastatur wird losgelassen
- .change – ein Formularfeld wird verändert

// DOM-Manipulation

2.4 SAP UI5 Framework

2.4.1 Definition

SAP UI5 ist ein Software Development Kit (SDK) zur Entwicklung von Desktop- und mobilen Anwendungen die in einem Browser ausgeführt werden. Nicht zu verwechseln mit SAP Fiori - was nur eine Sammlung webbasierter Anwendungen, die mit SAP UI5 entwickelt wurden, darstellt. Das Framework bündelt eine vielzahl an Technologien und Bibliotheken um den Entwicklungsprozess solcher Anwendungen bestmöglich zu unterstützen. Grundsätzlich basieren die, mit dem SDK entwickelten, Anwendungen auf den aktuellen Web-Entwicklungsstandards. Dazu gehören HTML 5, CSS 3 und JS. HTML 5 und CSS 3 werden, wie in den vorherigen Kapiteln geschildert, dafür verwendet um Struktur und Aussehen der Applikation zu gestalten. Bei JS hat man sich außerdem dazu entschieden zusätzlich die überaus populäre Erweiterungsbibliothek jQuery zu nutzen.(vgl. [Bui]) Das komplette SAP UI5 SDK ist freie Software und kann ohne jegliche SAP Lizenz bezogen und betrieben werden.

2.4.2 Architektur

„Das Model-View-Controller-Architekturmuster strukturiert die Softwareentwicklung in die drei Einheiten Datenmodell (Model), Präsentation (View) und Steuerung (Controller). Durch diese Trennung können die einzelnen Komponenten leichter erweitert, ausgetauscht oder wiederverwendet werden. Abbildung [... 5] zeigt dieses Architekturmuster.

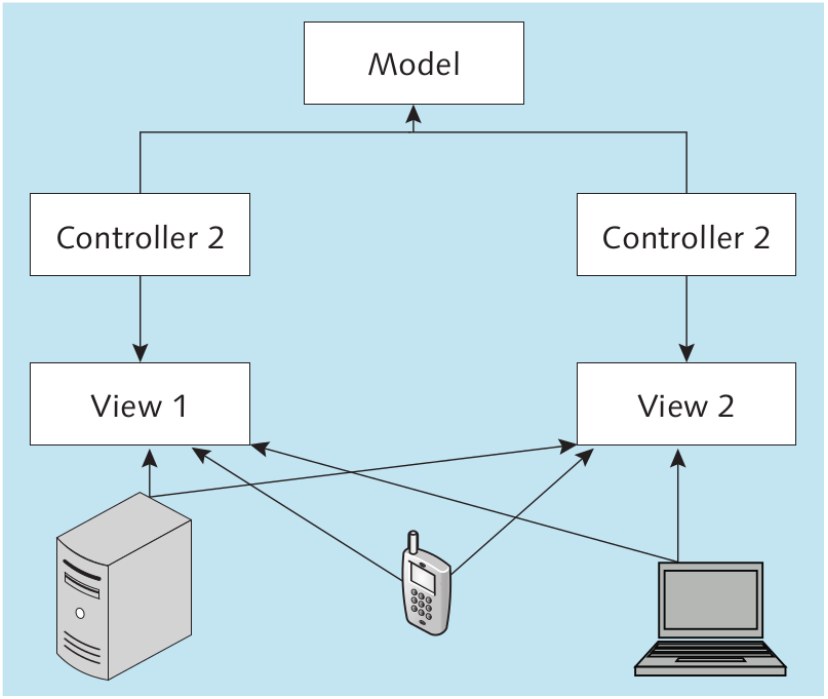

Abbildung 4: Model-View-Controller-Architekturmuster[Ant14, S.123]

Durch diese Trennung können z. B. zwei verschiedene Endgeräte das gleiche Model verwenden; der View wird z. B. einmal für die Desktop-Anwendung und einmal für das mobile Endgerät implementiert.“[Ant14, S.123]

Model

View

Controller // Vergleich

//Abbildung 5

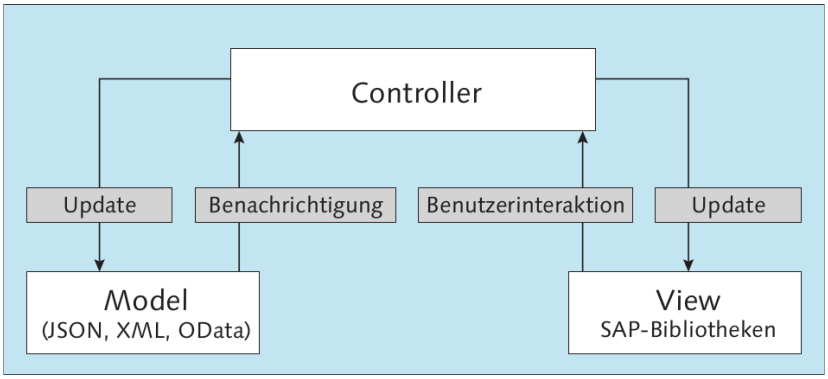

Abbildung 5: MVC-Architekturmuster[Ant14, S.124]

2.4.3 OData Protokoll

„Das Open Data Protocol (OData) ist ein von Microsoft veröffentlichtes Protokoll. Das Protokoll basiert auf HTTP und baut auf den älteren Protokollen Open Database Connectivity (ODBC) und Java Database Connectivity (JDBC) auf. OData ist primär für die sogenannten CRUD-Operationen, (Create, Read, Update und Delete) implementiert worden.“[Ant14, S.168]

// SAP Netweaver Gateway OData Services

3 Software Ergonomie

// Beleg für die Wichtigkeit von Software Ergonomie
// Kurze Übersicht über das Themenfeld Software Ergonomie
// Wichtigsten Aspekte nennen und näher erläutern

3.1 Definition

Kognitionspsychologie // Modellierung und Simulation von menschlichen Denk- und Wahrnehmungsprozessen

Arbeitsphysiologie, Industrieanthropologie // Beschäftigung mit grundlegenden menschlichen Fähigkeiten zur Informationsaufnahme und Informationsverarbeitung

Arbeitspsychologie // Untersuchung der Wechselbeziehungen zwischen Arbeit, deren Schnittstellen und psychischen Faktoren (unter anderem Arbeitszufriedenheit und -unlust)

3.2 DIN EN ISO 9241

// DIN Norm zur Software Ergonomie
// Die 7 Grundsätze der Dialoggestaltung:

- Aufgabenangemessenheit
- Selbstbeschreibungsfähigkeit
- Erwartungskonformität
- Fehlertoleranz
- Steuerbarkeit
- Individualisierbarkeit
- Lernförderlichkeit

DIN EN ISO 14915

// Erweiterung der ISO 9241

3.3 Analyse Methoden

Eye Tracking // Funktionsweise und Ergebnis

Mouse Clicking // Funktionsweise und Ergebnis

3.4 SAP Technologien in Bezug auf Software Ergonomie

3.4.1 Business Server Pages

// Business Server Pages (BSP) ist old school Technik
// geklaut von Java Server Pages (JSP)

3.4.2 Web Dynpro for ABAP

// Aktuelle Technik
// ABAP Code generiert HTML
// statischer und dynamischer Teil

3.4.3 SAP Fiori / SAP UI5 / SAP Screen Personas

// cutting edge
// aktuelle SAP UI Strategie
// SAP Präsi Chart Fiori/SP renew, etc. pp
// SAP Fiori einerseits Name des Themes/Guideline
// andererseits Bündel der gängigsten TAs/GPs als fertige
// Mobile First/Responsive Design Applikationen
// SAP UI5 - SAPs Framework zur Entwicklung von eigenen Applikationen im Fiori Style
// Nicht zu tief auf JS, HTML etc eingehen, dass kommt im nächsten Kapitel
// SAP SP - Zusätzliche Schicht um Standard Dynpro zu Personalisieren und so

4 Fallbeispiel SAP UI5

Lorem ipsum dolor sit amet.

4.1 Beschreibung

// Frontend - Browser, Elemente
// Backend - JSON, OData Model
// Analyse der wichtigen Arbeitsschritte

4.2 Hilfsmittel

4.2.1 Entwicklungsumgebung

Eclipse Zum Einsatz in der Implementierung des Fallbeispiels ist die quelloffene integrierte Entwicklungsumgebung Eclipse gekommen. Diese Integrated Development Environment (IDE) wurde von der Eclipse Foundation entwickelt und ist plattformunabhängig. Geschrieben wurde Eclipse in Java. Die aktuelle Version 4.4 ist am 25. Juni 2014 erschienen und trägt den Namen Luna. Eclipse zeichnet sich durch ein Plugin System aus mit welchem eine erhebliche Anzahl an Anwendungsfälle mit dieser IDE abgedeckt werden können.(vgl. [Wikb]) So auch die Entwicklung von SAP UI5 Applikationen. Dafür hat die SAP AG ein spezielles SAP UI5 Plugin bereitgestellt. Entsprechende Plugins müssen nicht umständlich über eine Webseite bezogen und installiert werden. Sie können über die integrierte Plugin Funktion installiert und eingerichtet werden. Eine URL zum Plugin ist vollkommen ausreichend. Für die Entwicklung von SAP UI5 Applikationen wurden folgende Plugins benötigt:

- UI Development Toolkit for HTML5
- ABAP Development Tools for SAP NetWeaver
- (SAP HANA Tools)

Bezogen werden können diese Plugins mittels der URL *https://tools.hana.ondemand.com/luna* und der erwähnten Plugin Funktion von Eclipse. Neben dem reinen Code Editor werden allerdings weitere Tools benötigt um eine SAP UI5 Applikation zu entwickeln.

Chrome Developer Tools Die SAP UI5 Dokumentation schlägt vor zum Testen der entwickelten Applikationen Google Chrome oder Mozilla Firefox anstatt Microsoft Internet Explorer zu verwenden. Das vorliegende Fallbeispiel wurde mit Google Chrome getestet. Google Chrome bietet dazu ein Tool mit dem Namen Developer Tools, welches in jeder Standard Installation des Browsers enthalten ist. Mit diesem Tool lässt

sich beispielsweise das DOM der aktuellen Webseite anzeigen. Weiter kann man JavaScript Breakpoints setzen und so effizient debuggen. Es bietet eine Konsole um direkte JavaScript Befehle abzusetzen und die Ergebnisse zu analysieren. Um eine Anwendungen nicht zwingend auf verschiedenen Endgeräten, mit verschiedenen Displaygrößen, testen zu müssen lassen sich jegliche Art von Endgeräten mit den Developer Tools emulieren.(vgl. [Inc])

Neptune Application Designer

4.2.2 UI Design und Prototyping

// Wireframing als Prototyping
// Abbildung Wireframesketcher

4.3 Implementierung

4.3.1 View

// Auszugsweise Coding bringen um bestimmte Elemente aus der Theorie zu zeigen
// Generellen Aufbau der Views erklären
// Kapselung wird dadurch verdeutlicht
// Listing 30

```
1  sap.ui.jsview("abat.Mockup.view.App", {
2
3    getControllerName: function () {
4      return "abat.Mockup.view.App";
5    },
6
7    createContent: function (oController) {
8      // to avoid scroll bars on desktop
9      this.setDisplayBlock(true);
10
11     // create app
12     this.app = new sap.m.SplitApp();
13
14     // load the master page
15     var master = sap.ui.xmlview("Master", "abat.Mockup.view.Master");
16     master.getController().nav = this.getController();
17     this.app.addPage(master, true);
18
19     // load the empty page
```

```
20      var empty = sap.ui.xmlview("Empty", "abat.Mockup.view.Empty");
21      this.app.addPage(empty, false);
22
23      // wrap app with shell
24      return new sap.m.Shell("Shell", {
25        title : "{i18n>ShellTitle}",
26        showLogout : false,
27        app : this.app
28      });
29    }
30  });
```

Listing 30: Root View der Applikation

```
// Master/Detail Applikation mit Fragment und Chart View Aufbau
// TODO: Visio Diagramm oder vergleichbares erstellen
// sap.ui.view
// |- sap.m.Shell
// | |- sap.m.SplitApp
// | | |- sap.m.Page
// | | | |- sap.m.Bar
// | | | |- sap.m.Bar
// | | | |- sap.m.List
// | | | | |- sap.m.ObjectListItem
// | | | |- sap.m.Bar
// | | |- sap.m.Page
// | | | |- sap.m.ObjectHeader
// | | | | |- sap.m.ObjectAttribute
// | | | | |- sap.m.ObjectAttribute
// | | | | |- sap.m.ObjectAttribute
// | | | | |- sap.m.ObjectAttribute
// | | | | |- sap.m.ObjectStatus
// | | | |- sap.IconTabBar
// | | | | |- sap.IconTabFilter
// | | | | | |- sap.ui.core.Fragment
// | | | | | | |- sap.ui.core.FragmentDefinition
// | | | | | | | |- sap.viz.ui5.Bar
// | | | | |- sap.IconTabFilter
// | | | | | |- sap.ui.core.Fragment
// | | | | | | |- sap.ui.core.FragmentDefinition
```

// | | | | | | | |- sap.viz.ui5.Bar
// | | | |- sap.m.Bar

4.3.2 Model und Controller

// die Verbindung von beiden Anhand von Coding zeigen
// TODO: OData Modell einbinden
// Listing 31

```
1  ...
2  // JSON Modell an die Root View binden
3  var oModel = new sap.ui.model.json.JSONModel("model/mock.json");
4  oView.setModel(oModel);
5
6  // OData Modell
7  var oModel = new sap.ui.model.odata.ODataModel(<URL>);
8  oView.setModel(oModel);
9
10 // I18N(Lokalisierung) Modell
11 var i18nModel = new sap.ui.model.resource.ResourceModel({
12   bundleUrl : "i18n/messageBundle.properties"
13 });
14 oView.setModel(i18nModel, "i18n");
15
16 // Geraetespezifisches Modell
17 var deviceModel = new sap.ui.model.json.JSONModel({
18   isPhone : jQuery.device.is.phone,
19   listMode : (jQuery.device.is.phone) ? "None" : "SingleSelectMaster",
20   listItemType : (jQuery.device.is.phone) ? "Active" : "Inactive"
21 });
22 deviceModel.setDefaultBindingMode("OneWay");
23 oView.setModel(deviceModel, "device");
24 ...
```

Listing 31: Component.js - Datenmodell an die Root View binden

4.3.3 Backend

// ABAP Stack der den RESTful Service bereitstellt zeigen
// Beispielhafte Implementation des HTTP Responses

5 Analyse

5.1 Heatmap

// Angewandte Analyse mit Heatmap

5.2 UI-Objekte

// Mobile First/Responsive Design

5.3 PLATZHALTER

// PLATZHALTER

6 Schluss

Lorem ipsum dolor sit amet.

Zusammenfassung // Arbeitsgebiete, Produktions & Dienstleistungsbereiche
// Arbeitsergebnisse
// Projektziele, Projektergebnisse, Projekttermine
// Mitwirkungszeiträume
// Liste aller selbst wahrgenommen Aufgaben und Tätigkeiten
// Projektmeilensteine
// Ablauforganisation & Beteiligte
// Arbeitsformen, Arbeitsmittel, Arbeitsabläufe
// Kommunikations- / Informationsgewohnheiten
// Auswertung relevanter Literatur
// Themen aus Lehrveranstaltungen

Bewertung // Wesentliche Erkenntnisse und Erfahrungen
// Folgerungen und Konsequenzen
// Vorschläge für Verbesserung und Veränderung
// Auswirkungen auf persönliche Berufs- und Karriereplanung
// Bezug zum Studium
// hilfreiche Studieninhalte
// neu gewonnenes Interesse

7 Quellenverzeichnis

[Ant14] Antolovic, Miroslav: *Einführung in SAPUI5: [Einführung in das SAP UI Development Toolkit für HTML5 ; moderne Benutzeroberflächen gestalten und erweitern ; Programmiermodell, Controls und UI-Elemente in der Praxis einsetzen].* 1. Aufl. Bonn [u.a.] : Galileo Press, 2014 (SAP PRESS). – ISBN 9783836227537 und 3836227533

[Bui] BuiltWith®: *Alexa 10k Technology Usage Statistics.* `http://trends.builtwith.com/tech-reports/Alexa-10k`. – Zugriff: 11.12.2014, Archiviert mit WebCite®: `http://www.webcitation.org/6UfdHXMrx`

[CG12] Clemens Gull, Stefan M.: *HTML5-Handbuch: [die neuen Features von HTML5 ; Webseiten für jedes Endgerät: Media Queries für mobile Devices ; so setzen Sie anspruchsvolle Web-Layouts mit HTML5 und CSS um ; umfangreicher Referenzteil für HTML und CSS zum Nachschlagen ; zukunftsorientierte Webseiten erstellen].* 2., aktualisierte und erw. Aufl. Haar bei Muüchen : Franzis, 2012 (Know-how ist blau). – ISBN 3645601511 und 9783645601511

[Fla07] Flanagan, David: *JavaScript: das umfassende Referenzwerk ; [behandelt Ajax und DOM-Scripting].* 3. Aufl., dt. Ausg. der 5. Aufl. Beijing [u.a.] : O'Reilly, 2007. – ISBN 3897214911 and 9783897214910

[Inc] Inc., Google: *Chrome DevTools Overview.* `https://developer.chrome.com/devtools`. – Zugriff: 04.01.2015, Archiviert mit WebCite®: `http://www.webcitation.org/6UfdHXMrx`

[Kro] Kroener, Peter: *HTML5 Spezifikations Graph.* `https://github.com/SirPepe/SpecGraph`. – Zugriff: 11.12.2014, Archiviert mit WebCite®: `http://www.webcitation.org/6UfdHXMrx`

[Krö11] Kröner, Peter: *HTML5: Webseiten innovativ und zukunftssicher ; [neu: Web Workers, File API, IE 9 uvm.].* 2. Aufl. Muüchen : Open Source Press, 2011 (Professional reference). – ISBN 3941841343 und 9783941841345

[ML05] Meyer, Eric A. ; Lang, Jørgen W.: *Cascading Style Sheets: das umfassende Handbuch ; [behandelt CSS2 und CSS2.1].* Dt. Ausg., 1. Aufl. Beijing [u.a.] : O'Reilly, 2005. – ISBN 3897213869

[Sch11] Schulten, Lars: *Durchstarten mit HTML5: [tauchen Sie ein in die Webentwicklung der Zukunft].* 1. Aufl. Köln [u. a.] : O'Reilly, 2011. – ISBN 9783897215719

[Sela] SelfHTML: *CSS Box Modell*. http://wiki.selfhtml.org/wiki/CSS/Box-Modell. – Zugriff: 09.12.2014, Archiviert mit WebCite®: http://www.webcitation.org/6UfdHXMrx

[Selb] SelfHTML: *CSS Einbindung*. http://wiki.selfhtml.org/wiki/CSS/Einbindung. – Zugriff: 07.12.2014, Archiviert mit WebCite®: http://www.webcitation.org/6UfdHXMrx

[Selc] SelfHTML: *Internet Explorer Fallback*. http://wiki.selfhtml.org/wiki/HTML/Tutorials/HTML5-Grundstruktur#Fallback_f.C3.BCr_.C3.A4ltere_Internet_Explorer. – Zugriff: 05.12.2014, Archiviert mit WebCite®: http://www.webcitation.org/6Ub78TIUq

[Seld] SelfHTML: *JavaScript / Einführung in JavaScript*. http://de.selfhtml.org/javascript/intro.htm. – Zugriff: 09.12.2014, Archiviert mit WebCite®: http://www.webcitation.org/6UfdHXMrx

[Sele] SelfHTML: *JavaScript / Objektreferenzen*. http://de.selfhtml.org/javascript/objekte/index.htm. – Zugriff: 17.12.2014, Archiviert mit WebCite®: http://www.webcitation.org/6UfdHXMrx

[Self] SelfHTML: *Physische Auszeichnungen im Text*. http://de.selfhtml.org/html/text/physisch.htm. – Zugriff: 01.12.2014, Archiviert mit WebCite®: http://www.webcitation.org/6UUqSW9ao

[Selg] SelfHTML: *Suchmaschinenoptimierung*. http://wiki.selfhtml.org/wiki/Suchmaschinenoptimierung. – Zugriff: 04.12.2014, Archiviert mit WebCite®: http://www.webcitation.org/6UZA8MTZe

[w3S] w3Schools: *CSS Selectors*. http://www.w3schools.com/css/css_selectors.asp. – Zugriff: 09.12.2014, Archiviert mit WebCite®: http://www.webcitation.org/6UfdHXMrx

[Wen08] Wenz, Christian: *JavaScript und Ajax: das umfassende Handbuch ; [Einführung, Praxis, Referenz ; browserübergreifende Lösungen ; Web 2.0: DOM, CSS, XML, Web Services ; neu in der 8. Auflage: Microsoft Silverlight, ASP.NET AJAX 1.0, Ausblick auf Firefox 3 und JavaScript 1.8]*. 8., aktualisierte und erw. Aufl. Bonn : Galileo Press, 2008 (Galileo computing). – ISBN 3836211289 and 9783836211284

[Wika] Wikipedia: *CSS Box Modell*. http://commons.wikimedia.org/wiki/File:Boxmodell-detail.png. – Zugriff: 09.12.2014, Archiviert mit WebCite®: http://www.webcitation.org/6UfdHXMrx

[Wikb] Wikipedia: *Eclipse (IDE)*. http://de.wikipedia.org/wiki/Eclipse_(IDE). – Zugriff: 02.01.2015, Archiviert mit WebCite®: http://www.webcitation.org/6UfdHXMrx

[Wikc] Wikipedia: *JavaScript - Sandbox-Prinzip*. http://de.wikipedia.org/wiki/JavaScript. – Zugriff: 11.12.2014, Archiviert mit WebCite®: http://www.webcitation.org/6UfdHXMrx

Anhang

A GUI

Ein toller Anhang.

Screenshot

Unterkategorie, die nicht im Inhaltsverzeichnis auftaucht.

Erklärung

Hiermit versichere ich, dass ich meine Abschlussarbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

Datum: .....

(Unterschrift)