Fachhochschule Wilhelmshaven

Fachbereich Management, Information & Technologie

Bachelorarbeit

zur Erlangung des akademischen Grades
Bachelor of Science

Thema:	Prototypische Implementierung einer SAP UI5 Applikation im SAP Umfeld und Analyse eines effizienten Einsatz von UI-Objekten
Autor:	Nils Lutz Mat. Nr. 6002109
Version vom:	26. Januar 2015
1. Betreuer:	Prof. Dr. Hergen Pargmann
2. Betreuer:	Prof. Dr. Harald Schallner

Zusammenfassung

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

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.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Listingverzeichnis

Abkürzungsverzeichnis

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

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
// Begründen warum Technologien so ausführlich ist
// Liegt daran, dass die Implementierung vergleichsweise einfach ist
// sofern man die Technos verstanden und verinnerlicht hat

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 HTML, die Gestaltungssprache CSS und die Skriptsprache 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 HTML 5

In diesem Kapitel wird zuerst die Entstehung des aktuellen HTML Standards beschrieben. Von den Anfängen bis zur heute gültigen Spezifikation. Weiter werden die Ziele näher definiert, sowie der allgemeine Aufbau eines HTML Dokuments aufgezeigt. Abschließend sind die wichtigsten neuen Elemente der HTML 5 Sprache ausführlich dargelegt.

2.1.1 Historie

HTML 5 ist die aktuell empfohlene Spezifikation des W3C und stellt eine der Kernsprachen des 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 XHTML genannte Spezifikation entwickelt, die HTML mit dem 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 SGML - eine Metasprache zur Definition von Auszeichnungssprachen - sondern durch ein 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 HTML 5 dann vom W3C zum De-facto Standard des WWW erklärt. Heute existiert neben der Spezifikation des W3C auch noch ein sogenannter „lebender Standard“ der 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“ der WHATWG hinaus und beinhaltet zahlreiche Schnittstellen zu anderen Technologien. Abbildung 1 verdeutlicht diese Situation.

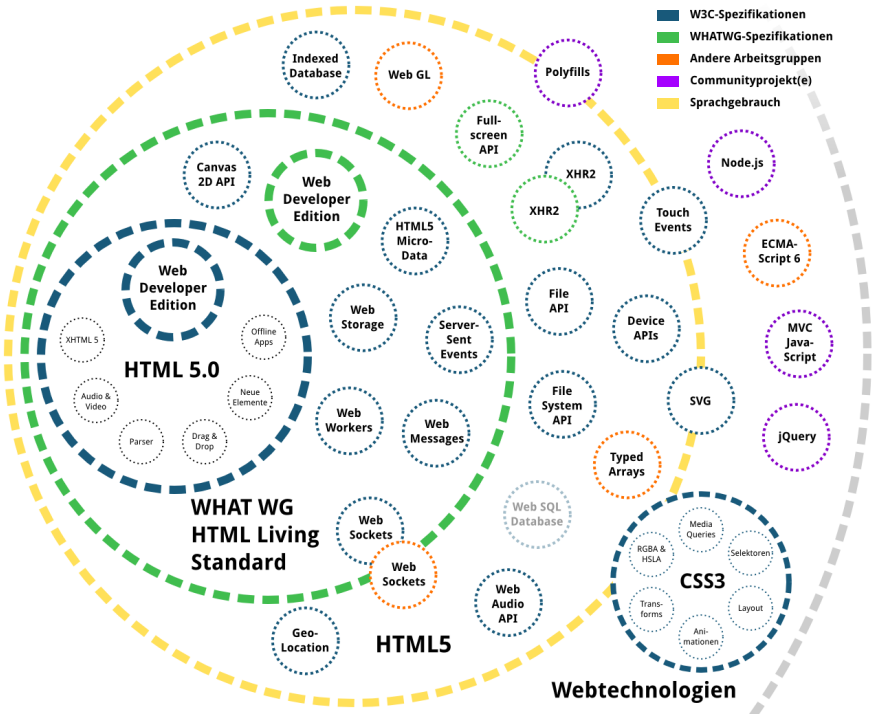

Abbildung 1: HTML 5 Spezifikationen Übersicht [Kroa]

2.1.2 Ziele

HTML 5 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 HTML 5 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: HTML 5 Browserkompatibilität

2.1.3 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 HTML 5 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 HTML 5 sind auch diese *Doctype* heute noch gültig und das Dokument wird korrekt vom Parser interpretiert werden.

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

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

Listing 2 hingegen zeigt das *Doctype* von HTML 5. 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*.

```
<!DOCTYPE html>
```

Listing 2: HTML 5 *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 HTML 5 Dokument mit den Mindestanforderungen.(vgl. [Krö11, S.58])

```
<!DOCTYPE html>
<html>
 <head>
  <title>Beispiel Seite</title>
 </head>
 <body>
  <h1>Beispiel Seite</h1>
  <p>Dies ist ein <a href="demo.html">einfaches</a> Beispiel.</p>
  <!-- Dies ist ein Kommentar -->
 </body>
</html>
```

Listing 3: HTML 5 Basis Dokument

2.1.4 Wichtige neue Sprachelemente

In HTML 5 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-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.

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

Listing 4: HTML 5 *meta*-Element

Zwei weitere neue Elemente in HTML 5 sind das *header*- und *footer*-Element. 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.

```
<body>
  <header>
    <img src="logo.gif" alt="logo">
    <h1>Ueberschrift</h1>
  </header>
  <footer>
     <a href="kontakt.html">Kontakt</a>
  </footer>
</body>
```

Listing 5: HTML 5 *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.

```
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 HTML 5-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: HTML 5 Struktur Elemente

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. „Damit auch ältere Internet Explorer der Versionen 6-8 die neuen HTML 5-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>HTML 5-Seite mit Grundstruktur</title>
7 </head>
```

Listing 7: HTML 5 Internet Explorer Fallback

Eingabefelder sind in den meisten Applikationen unabdingbar. Aus diesem Grund wurden in HTML 5 eine vielzahl an *input*-Elementen hinzugefügt. So müssen keine komplizierten Workarounds mit JavaScript oder anderen Skriptsprachen mehr verwendet werden. „Das typische HTML 5-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 Einga-

be 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: HTML 5 *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 CSS 3

Das Kapitel CSS 3 erläutert kurz die Notwendigkeit von CSS und geht dann weiter auf die Einbindung in einem HTML Dokument ein. Anschließend wird die Syntax der Gestaltungssprache beschrieben. Einige Eigenheiten wie Selektoren und Pseudoklassen werden erklärt um dann zum Box-Modell zu kommen. Abgerundet wird das Kapitel mit der Definition der spezifischen Stylesheets, die gerade in der Web Entwicklung ihre Stärken zeigen.

2.2.1 Einleitung

„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 ermöglicht 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]

2.2.2 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])

2.2.3 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.

```
h1, h2, p {
  color: black;
}
```

Listing 13: CSS3 Gruppierung

2.2.4 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 (>).

```
h1 strong {
  font-size: 14pt;
  font-weight: bold;
}
```

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.

```
<div id="ID">
  <p class="Klasse"></p>
</div>

div#ID {
  text-align: center;
  color:      red;
}
.Klasse {
  text-align: center;
  color:      red;
}
p.Klasse {
  text-align: center;
  color:      red;
}
```

Listing 15: CSS3 Klassen- und ID-Selektoren

2.2.5 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 einem 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* be-

zieht 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

2.2.6 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.

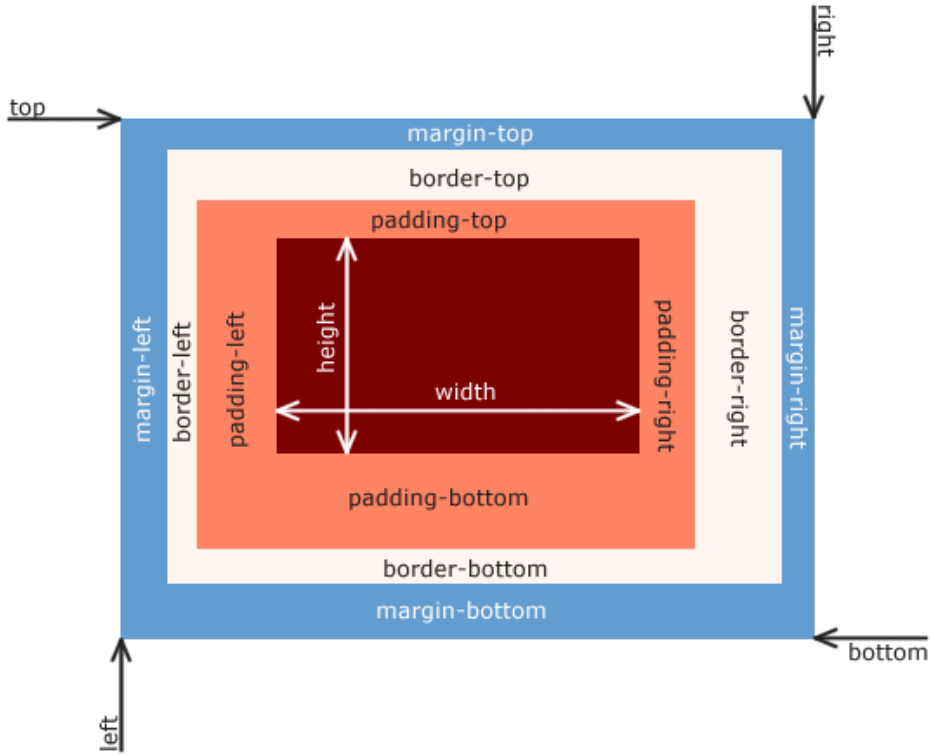

Abbildung 2: CSS-Boxmodell[1]

2.2.7 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="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

[1] Bildquelle: [Wika]

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.

```
#inhalt {
        width: 800px;
}

@media screen and (max-width: 1024px) {
        #inhalt {
                width: 600px;
        }
        aside {
                display: none;
        }
}
```

Listing 18: CSS3 eigenschaftsspezifisches Stylesheet

2.3 JavaScript

Dieses Kapitel soll ein grundlegendes Verständnis für die Skriptsprache JavaScript aufbauen. Beginnend mit der Historie und dem Sandbox-Prinzip hin zur Objektorientierung und den Sprachelementen werden außerdem die einzelnen Datentypen, Werte und Variablen beschrieben. Im Detail werden dann die Operatoren, Schleifen und Kontrollstrukturen erörtert und durch Listings verdeutlicht. Des Weiteren ist das Document-Object-Modell, die Ereignisverarbeitung und AJAX wichtig zum Verständnis des Frameworks jQuery, auf welchem letztendlich auch das SAP UI 5 Framework basiert.

2.3.1 Historie

JavaScript wurde 1995 von Netscape entwickelt, lizenziert und eingeführt. Um der Sprache von Anfang an einen Standardcharakter zu geben wurde die Organisation 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])

2.3.2 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])

2.3.3 Paradigma

„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])

2.3.4 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]

```
12                  // Die Zahl zwoelf
1.2                 // Die Zahl eins komma zwei
"Hallo Welt"        // Ein Text-String
'Hi'                // Noch ein String
true                // Ein Boolescher Wert
false               // Der andere Boolesche Wert
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

2.3.5 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} dar-

stellen. 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])

2.3.6 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 Funkti-

onsblocks Gültigkeit 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])

2.3.7 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])

2.3.8 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 Abbruchbedingung 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])

2.3.9 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

2.3.10 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

2.3.11 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.

```
<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

Listing 28 zeigt eine in HTML implementierte Tabelle die in das entsprechende DOM umgewandelt wird, welches in Abbildung 3 zu sehen ist.

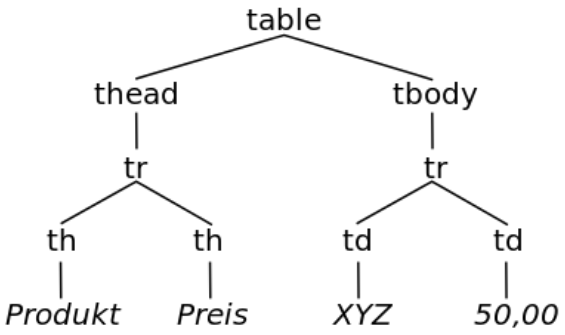

Abbildung 3: DOM Beispielbaum aus Listing 28

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.

2.3.12 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.(vgl. [Fla07, S.428ff])

- 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)

2.3.13 AJAX

AJAX steht als Akronym für *Asynchronous JavaScript and XML*. Geprägt wurde der Begriff AJAX von Jesse J. Garret.(vgl. [Gar]) Es ist keine eigenständige Technologie, sondern vielmehr eine Bündelung einiger der verbreitetsten Web Technologien. Dazu gehört das HTTP Protokoll, JavaScript, XML und neuerdings auch JSON. Entwickelt wurde diese Technik von Microsoft, speziell von den Outlook Entwicklern. Diese benötigten eine Möglichkeit um HTTP Anfragen an einen Server abzusetzen ohne ein permanentes Neuladen der kompletten Webseite zu verursachen, um zu prüfen ob neue Mails vorhanden sind. Diese Anfragen sollten im Hintergrund ausgewertet und der entsprechende Teil der Webseite durch JavaScript, mit den nachgeladenen Informationen, verändert werden. Dadurch muss der Anwender nicht mit seiner Dateneingabe warten bis der Server mit der Auswertung der vorher eingegeben Daten fertig ist.
Technisch erfolgt dies in drei Schritten. Zu erst wird ein AJAX Objekt erzeugt, dazu haben die Browser Hersteller das *XMLHttpRequest*-Objekt implementiert. Über dieses Objekt wird eine Verbindung zur Zielseite hergestellt. Als Parameter benötigt das Objekt die HTTP Übertragungsmethode, sprich *GET* oder *POST*. Außerdem die URL der Zielseite und zuletzt einen Booleschen Wert, der das Skript entweder

synchron oder asynchron ausführen lässt. Zuletzt wird eine Rückgabefunktion angegeben die aufgerufen wird sobald der Server mit der Verarbeitung fertig ist und seine Antwort auf der Client Seite angekommen ist.(vgl. [Wen08, S.392ff]) Abbildung 4 zeigt schematisch die Kommunikation zwischen Client und Server mit AJAX Technik. Bekanntestes Beispiel einer AJAX Implementation ist wohl der Vorschlag-Mechanismus der Google Suche.

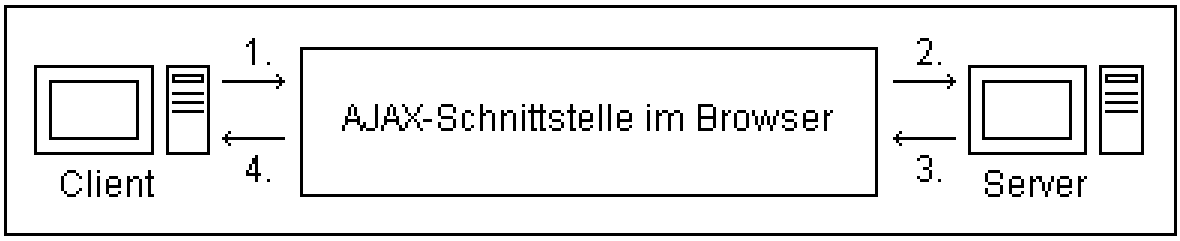

Abbildung 4: AJAX Client/Server Kommunikation [Krob]

2.3.14 jQuery

jQuery ist die meistverbreiteste JavaScript Bibliothek im Internet. Mit 94,7% Marktanteil, Stand 19.1.2015, ist jQuery klar der Marktführer bei den JavaScript Bibliotheken. Ganze 61,7% der vom W3 Technology Survey analysierten Webseiten verwenden jQuery in ihrer Implementierung. (vgl. [Sur]) Die freie JavaScript Bibliothek jQuery stellt unter anderem umfangreiche Funktionen zur Manipulierung des DOM bereit. Daneben ist die verwendung der AJAX Technik enorm vereinfacht worden von jQuery. Animationen und Effekte lassen sich schnell und unkompliziert mit der Bibliothek realsieren. Wie aus CSS bekannt verwendet jQuery Selektoren um auf die DOM Knoten zuzugreifen. Dank dem Programmierkonzept der *fluent interfaces*, was soviel bedeutet wie sprechende Schnittstellen, kann ein erzeugtes jQuery Objekt sehr einfach an andere Funktionen weitergegeben werden.(vgl. [Wikd]) „jQuery wird von `http://docs.jquery.com/` geladen und per Script-Tag in die HTML-Dokumente eingebunden.

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

Listing 30: jQuery Einbindung

Die grundlegende Funktion von jQuery ist die Funktion jQuery() oder auch mit verringertem Schreibaufwand $(), deren Verhalten abhängig ist von den jeweils gesetzten Parametern. Dabei fasst (sammelt) jede $()-Funktion einen oder mehrere Knoten eines DOM-Baumes zusammen. In der einfachsten Form wird dann nur ein Ausdruck übergeben - meistens ein CSS-Selektor -, der alle passenden Elemente im Dokument findet.“[Wis]

```
1 $("selektor").funktionsname({ function({ }); });
```

Listing 31: jQuery Selektor Syntax

Listing 31 zeigt die übliche Syntax einer jQuery Anweisung. Über den Selektor werden die Funktionen des erzeugten Objekts aufgerufen und eine entsprechende Rückgabe Funktion wird außerdem übergeben um auf das Ergebnis der Objekt Funktion reagieren zu können.

2.4 SAP UI5 Framework

Kapitel 2.4 beschreibt das SAP UI 5 Framework der SAP AG. Beginnend wird mit einem kurzen Exkurs über die aktuelle UI Strategie der SAP AG, um zu verstehen wieso die SAP AG dieses Framework überhaupt entwickelt hat. Im Anschluss kommt eine allgemeine Definition, sowie die Beschreibung der verwendeten Softwarearchitektur. Zuletzt folgt eine kurze Erklärung des OData Protokolls und die Positionierung des Frameworks innerhalb einer SAP Netweaver Landschaft.

2.4.1 SAP UI Strategie

Die SAP AG definiert ihre momentane *User Experience* Strategie durch den Styleguide SAP Fiori und drei Schlagwörter – *New*, *Renew* und *Enable*. Unter *New* wird verstanden, dass Entwicklern und Kunden neue Tools zur Entwicklung von SAP Applikationen und neue Applikationn selbst zur Verfügung gestellt werden. *Renew* bedeutet, dass die vorhandenen Kernszenarios auf das neue UX portiert werden. SAP Screen Personas, auf das im Rahmen der Bachelorarbeit nicht weiter eingegangen wird, gehört zum Schlagwort *Enable*. Kurz gesagt soll mit diesem Toolset das vorhandene UI der SAP Software an ein Unternehmen im Sinne des Corporate Design angepasst werden.(vgl. [AGd])

2.4.2 Definition

SAP UI5 ist ein SDK zur Entwicklung von Desktop- und mobilen Anwendungen die in einem Browser ausgeführt werden. 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

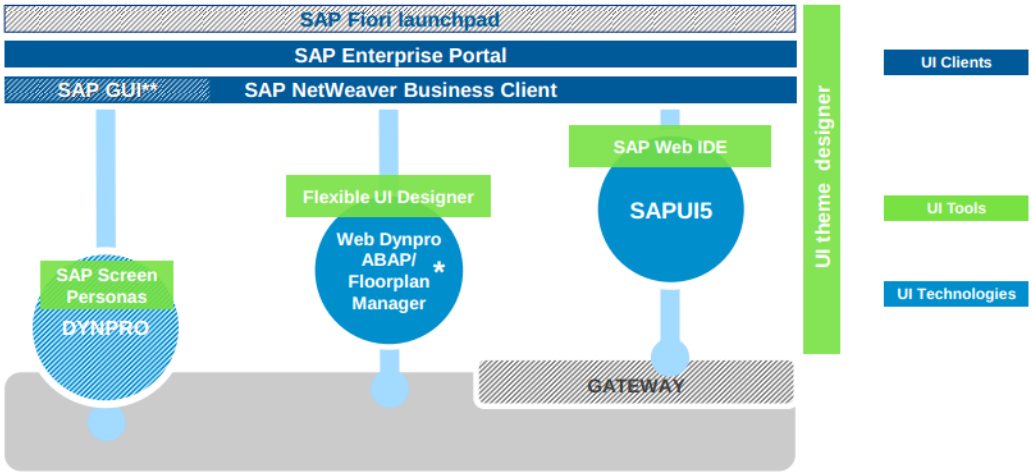

Abbildung 5: SAP Schlüssel UI Tools und Technologien [AGc]

Erweiterungsbibliothek jQuery zu nutzen.(vgl. [Bui]) Das komplette SAP UI5 SDK ist freie Software und kann ohne jegliche SAP Lizenz bezogen und betrieben werden.

2.4.3 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 [... 6] zeigt dieses Architekturmuster.

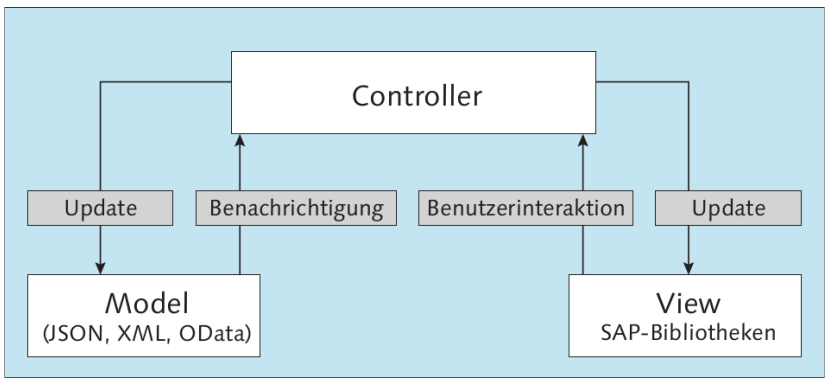

Abbildung 6: MVC-Architekturmuster [Ant14, S.124]

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

Mit dem Model wird das Datenmodell abgebildet. Neben dem Datenbankzugriffsme-

chanismus stellt es die Applikationsdaten bereit und kann zudem auch die dazugehörige Geschäftslogik enthalten.

View
Benutzeraktionen werden vom View erfasst und zur weiteren Verarbeitung an den Controller weitergegeben. Damit fungiert das View als Präsentationsschicht zur visuellen Darstellung auf den verschiedenen Endgeräten.

Controller
Der Controller ist die zentrale Steuereinheit. Er nimmt Benutzeraktionen vom View entgegen und verarbeitet diese weiter. Ein Controller kann mehrere Views verwalten, zu jeder View gehört mindestens ein Controller. Bei Datenänderungen durch den Anwender koordiniert der Controller die Kommunikation mit dem Model.(vgl. [Ant14, S.123f])

2.4.4 OData Protokoll

„Das Open Data Protocol (OData) ist ein von Microsoft veröffentlichtes Protokoll. Das Protokoll basiert auf HTTP und baut auf den älteren Protokollen ODBC und JDBC auf. OData ist primär für die sogenannten CRUD-Operationen, (Create, Read, Update und Delete) implementiert worden.“[Ant14, S.168] Von der SAP AG ist der Einsatz der SAP Netweaver Gateway Software vorgesehen, um einen entsprechenden OData Service im Backend bereit zustellen. Dadurch wird ein direkter Zugriff auf die dahinter liegenden Systeme verhindert. OData stellt eine API nach dem REST Prinzip bereit. Ein REST Service hat vier Eigenschaften die erfüllt sein müssen. Dazu gehört die Adressierbarkeit, jeder REST Service hat eine eindeutige Adresse den Uniform Resource Locator (URL). Der REST Server muss die angeforderten Daten in unterschiedlichen Formaten zurück geben können wie z.B HTML, JSON oder XML. Zustandslosigkeit ist eine weitere Eigenschaft. Sie besagt, dass weder der Server noch die Applikation Zustandsinformationen zwischen zwei Nachrichten speichert. Jede Anfrage an den Server ist in sich abgeschlossen. Die letzte Eigenschaft beschreibt die Operationen die ein REST Service bereitstellt. Bei einem Zugriff über HTTP werden die Methoden *GET*, *POST*, *PUT* und *DELETE* verwendeten um die CRUD Operationen zu ermöglichen.(vgl. [Wike])

2.4.5 Positionierung im Netweaver Stack

Eine mit SAP UI5 entwickelte Applikation wird als BSP Applikation auf dem SAP ABAP Application Server abgelegt. Von dort kann sie dann über die verschiedenen

Endgeräte abgerufen werden. Aufgrund der verwendeten Technologien läuft die Anwendung weitestgehend auf dem jeweiligen Endgerät. Der ABAP AS dient lediglich dazu eingehende Anfragen, die vorher vom SAP Netweaver Gateway entgegen genommen und weitergeleitet wurden, zu verarbeiten und eine entsprechende Antwort zurück zu schicken. Natürlich kann man auf das Gateway verzichtet, was aus Sicherheitsgründen jedoch nicht zu empfehlen ist. Auf Grund des verwendeten OData Protokolls und der REST Technik können problemlos Lastverteiler eingesetzt werden. Wodurch eine einfache Möglichkeit zur Skalierung geboten wird.(vgl. [AGb]) Abbildung 7 zeigt eine schematische SAP Landschaft in der SAP UI5 integriert ist.

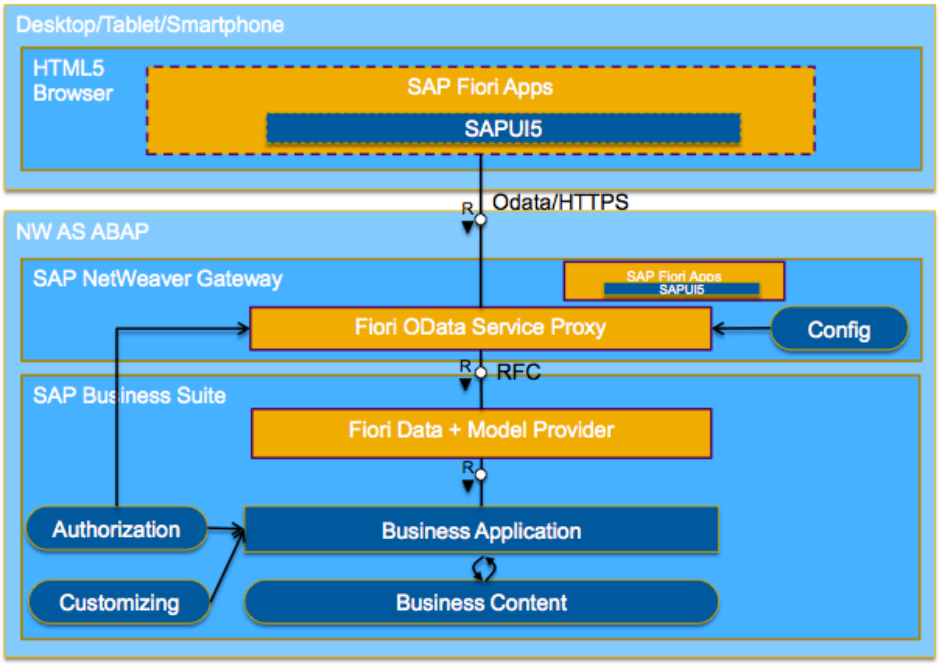

Abbildung 7: Einordnung von SAP UI5 in die SAP System Landschaft [AGb]

3 Fallbeispiel SAP UI5 Applikation

In diesem Kapitel wird das prototypische Fallbeispiel vorgestellt. Eine kurze Erläuterung des Front- und Backends der Applikation wird in der Beschreibung gegeben. Im Unterkapitel 3.2 werden die genutzten Hilfsmittel in ihrer Funktion und Zugehörigkeit zum Fallbeispiel erklärt. Danach folgt die Implementierung selbst in der Anhand von Screendumps und Applikationscode beschrieben wird wie das Fallbeispiel nach dem Model-View-Controller Muster umgesetzt worden ist.

3.1 Beschreibung

Die prototypische SAP UI5 Applikation ist dazu gedacht im Rahmen eines SAP SCM Systems eine Übersicht über die Produkte zu geben. Das Frontend wird im Browser ausgeführt, wohingegen das Backend von einem ABAP AS repräsentiert wird der zusätzlich noch durch ein SAP Netweaver Gateway geschützt ist. Nach Auswahl eines Produktes sollen weitere Details angezeigt werden. Auf dieser Detail Seite sind Informationen wie Name, Material Nummer, Brutto/Netto Preis und Lagermenge einsehbar. Außerdem können über zwei Reiter Charts angezeigt werden. Die erste Chart soll anhand historischer Absatzdaten eine Prognose über zukünftige Absatzzahlen geben. In der zweiten Chart wird eine Bestellvorschau bereitgestellt.

3.2 Hilfsmittel

Dieses Kapitel soll kurz aufzeigen welche Hilfsmittel zur Realisierung des Prototypen verwendet wurden. Zum einen gehört die Entwicklungsumgebung Eclipse dazu. Weiter wurden die Chrome Developer Tools des Chrome Browser genutzt, sowie das Tool Wireframesketcher. In den folgenden Unterkapitel werden diese Tools kurz vorgestellt.

3.2.1 Eclipse

Zum Einsatz in der Implementierung des Fallbeispiels ist die quelloffene integrierte Entwicklungsumgebung Eclipse gekommen. Diese 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.

3.2.2 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])

3.3 Implementierung

Es wird die Implementierung des Fallbeispiels beschrieben. Jede Schicht des Model-View-Controller Musters hat dabei ihr eigenes Unterkapitel. Damit werden jeweils die wichtigen Schlüsselpunkte der Applikation offen gelegt. Mehrfach verwendeter Code wird in dem Kapitel bewusst nicht gezeigt um keine unnötige Komplexität zu erzeugen. Der vollständige Programmcode ist im Anhang zu finden. Abbildung 8 zeigt eine schematische Übersicht der SAP UI5 Applikation und ihrer Komponenten.

3.3.1 Vorbereitungen

In Eclipse wird ein neues Projekt als SAP UI5 Applikation erstellt. Nach dem Anlegen einer neuen SAP UI5 Applikation ohne initial View findet man im Projekt Explorer von Eclipse die generierte Verzeichnisstruktur vor. In dieser Struktur finden

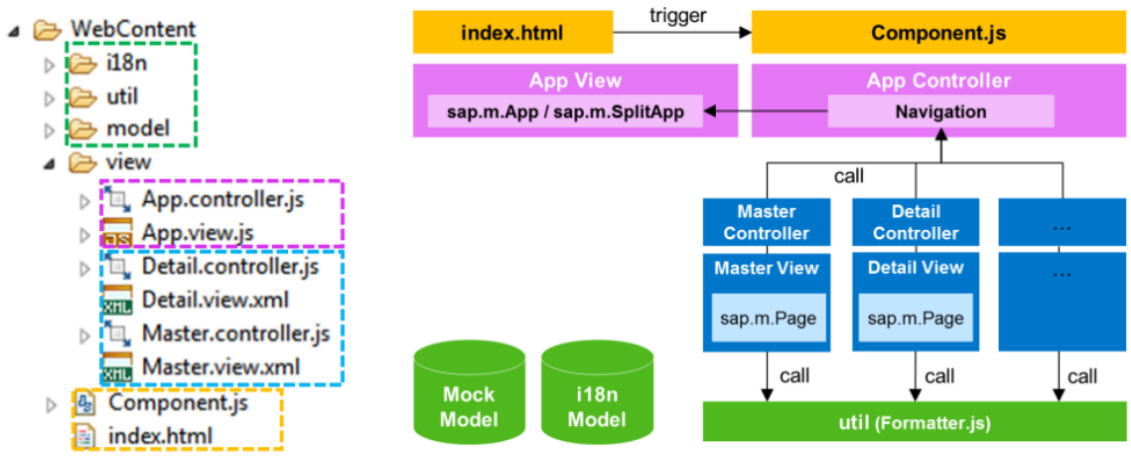

Abbildung 8: Übersicht der SAP UI5 Applikation [AGa]

sich zu beginn drei Verzeichnisse. Das *META-INF* Verzeichnis wird automatisch von Eclipse erstellt und dient lediglich als Informationsspeicher für den Umgang des Projekt im Zusammenspiel mit einigen Java Tools. Daneben existiert das *WEB-INF* Verzeichnis. In den Dateien dieses Verzeichnisses stehen Anweisungen für das Deployment des Projekts, sowie sind vorkompilierte Klassendateien und Hilfsbibliotheken in Unterordnern hinterlegt. Außerdem enthält das Projekt auch das *WebContent* Verzeichnis, welches als Arbeitsverzeichnis für das gesamte Projekt gilt. Dort sind sämtliche statische Dateien, wie z.B. HTML Dokumente, JavaScript und XML Dateien oder auch andere Ressourcen wie Bilder und Texte abgelegt. Im Arbeitsverzeichnis werden für die spätere ordentliche Struktur Verzeichnisse für die unterschiedlichen Zwecke angelegt – *i18n*, *model*, *util* und *view*. Im *i18n* Verzeichnis liegen später die Sprachdateien um die Applikation lokalisiert auszuliefern. Das *model* und *view* Verzeichnis sind selbsterklärend. Im *util* Verzeichnis werden Helferskripte hinterlegt wie z.B. *Formatter* oder *Grouper* Skripte. Als nächstes folgt die Implementierung der Views.

3.3.2 View

Die Datei *index.html* ist der Startpunkt der SAP UI5 Applikation. In dieser Datei wird mittels eines *title*-Element, direkt unter dem letzten *meta*-Element, der Seitentitel festgelegt. Innerhalb des dann folgenden *script*-Block werden Eigenschaften der SAP UI5 Applikation eingestellt. An dieser Stelle müssen die verwendeten SAP UI5 JavaScript Bibliotheken und die Art und Weise der Datenbindung des Modells hinzugefügt werden. Der Prototyp arbeitet mit der *complex* Datenbindung. In dem leeren *script*-Block wird dann letztendlich die Anweisung gesetzt, welche den *Component Container* lädt und damit das Bootstrapping der Applikation in Gang setzt. Die Datei *index.html* ist in verkürzter Form in Listing 5 zu sehen.

```
...
        <script
                id="sap-ui-bootstrap"
                src="resources/sap-ui-core.js"
                data-sap-ui-theme="sap_bluecrystal"
                data-sap-ui-libs="sap.m, sap.ui.layout"
                data-sap-ui-preload=""
                data-sap-ui-xx-bindingSyntax="complex"
                data-sap-ui-resourceroots='{
                        "abat.Mockup": "./"
                }' >
        </script>

        <script>
                new sap.ui.core.ComponentContainer({
                        name : "abat.Mockup"
                }).placeAt("content");
        </script>
...
```

Listing 32: Bootstrapping der SAP UI5 Applikation

Component Container

Der eben erwähnte *Component Container* wird nach den Änderungen an der Datei *index.html* im Arbeitsverzeichnis als Datei *Component.js* erstellt. Der *Component Container* ist ein Sammelbehälter für die unterschiedlichen SAP UI5 Komponenten. Er beinhaltet neben der Deklaration und Initialisierung der Model auch die Root View. Diese Root View ist die unterste Schicht des View Stacks. In ihr werden sämtliche weiteren Views platziert. Bei der Initialisierung wird auf die Datei *App.view.js* verwiesen. Zu den verwendeten Models sei an dieser Stelle noch nichts näher erläutert da dies im nächsten Kapitel folgt. Als Rückgabewert liefert dieser *Component Container* dann die Root View mit ihren Elementen. Listing 5 zeigt auszugweise die Datei *Component.js* und die Erstellung der Root View.

```
jQuery.sap.declare("abat.Mockup.Component");
sap.ui.core.UIComponent.extend("abat.Mockup.Component", {
        createContent : function() {

                // create root view
                var oView = sap.ui.view({
                        id : "app",
                        viewName : "abat.Mockup.view.App",
                        type : "JS",
                        viewData : { component : this }
                });

```

```
13                  // model declaration and initialisation goes here
14                  ...
15
16                  return oView;
17          }
18  });
```

Listing 33: Auszug aus der Component.js

Root View

Die Datei *App.view.js* repräsentiert die Root View aus dem *Component Container*. Der Rückgabewert ist eine *Shell* genannte SAP UI5 Komponente. Dieser *Shell* wird ein Titel über das Lokalisierungsmodell zugewiesen. Dazu später mehr. Ausdrücke die einen String darstellen können so leicht mit einer lokalen Version des Wertes ersetzt werden. Des Weiteren verlangt die *Shell* eine *App* Komponente. Diese Komponente wird am Anfang der Datei *App.view.js* erzeugt. Es handelt sich bei dem Prototyp um eine *App* Komponente namens *SplitApp*. Eine *SplitApp* besitzt eine Haupt- und Detailseite. Diese werden auch innerhalb der *App.view.js* erzeugt und der *SplitApp* Komponente zugewiesen. Allerdings verweisen die Haupt- und Detailseite auch wieder auf eigene Dateien auf Grund des Kapselungsprinzip zum Zwecke der besseren Wartbarkeit. Listing 5 zeigt den gerade beschriebenen Vorgang.

```
1  ...
2      // create app
3      this.app = new sap.m.SplitApp();
4
5      // load the master page
6      var master = sap.ui.xmlview("Master", "abat.Mockup.view.Master");
7      master.getController().nav = this.getController();
8      this.app.addPage(master, true);
9
10     return new sap.m.Shell("Shell", {
11       title : "{i18n>ShellTitle}",
12       showLogout : false,
13       app : this.app
14     });
15   }
16 });
```

Listing 34: Root View der Applikation

Hauptseite

Die *Master.view.xml* wird nicht, wie der Dateiname schon schließen lässt, gleich der Root View im JavaScript sondern im XML Format erstellt. Sie beinhaltet die Komponenten, die in der *SplitApp* die Hauptseite darstellt. Realisiert wird das über die

Komponente *Page*. In der *Page* liegen widerum weitere Komponenten die das Aussehen der Hauptseite definieren. Durch einen *customHeader* inklusive einer *Bar* und einem *Image* wird ein Firmenlogo in die Hauptseite ganz oben eingefügt. Ein *subHeader* mit einer *Bar* und einem *SearchField* sind für eine Suchfunktion innerhalb der Liste gesetzt. Eine *List* implementiert eine vertikale Liste, die mit Komponenten vom Typ *ObjectListItem* befüllt wird. Das *ObjectListItem* dient in dem Fall als Formatvorlage für die tatsächlichen Daten die später aus dem Model bezogen werden. Abgeschlossen wird die Seite mit der *footer* Komponente in der eine *Bar* und ein *Button* eingelassen sind. Der Button soll eine Gruppierungsfunktion für die Liste möglich machen. Listing 5 enthält die erläuterte Struktur der Hauptseite im XML Format.

```
<core:View
  controllerName="abat.Mockup.view.Master"
  xmlns="sap.m"
  xmlns:core="sap.ui.core" >
  <Page>
    <customHeader><Bar><contentLeft>
          <Image src="img/logo.png" width="173px" height="30px"></Image>
    </contentLeft></Bar></customHeader>
    <subHeader><Bar><contentLeft>
        <SearchField
          search="handleSearch"
          width="100%" >
        </SearchField>
    </contentLeft></Bar></subHeader>
    <List id="list"
      mode="{device>/listMode}"
      select="handleListSelect"
      items="{/Products}" >
      <ObjectListItem type="{device>/listItemType}"
        press="handleListItemPress"
        title="{MaterialName}"
        number="{Quantity}"
        numberUnit="{i18n&gt;QuantityUnit}"
        numberState="{parts : [ 'Quantity', 'MinimalQuantity' ],
              formatter : 'abat.Mockup.util.Formatter.numberState'}" >
        <attributes><ObjectAttribute text="{MatId}" /></attributes>
        <firstStatus><ObjectStatus text="{Status}"
            state="{path: 'Status',
            formatter: 'abat.Mockup.util.Formatter.statusState'}" />
        </firstStatus>
      </ObjectListItem>
    </List>
    <footer>
    <Bar><contentRight><Button icon="sap-icon://group-2"
```

```
35          press="handleViewSettings" />
36      </contentRight></Bar>
37      </footer>
38    </Page>
39 </core:View>
```

Listing 35: Hauptseite der SplitApp

Detailseite

Die Detailseite wird zwar in der Root View nicht direkt deklariert, dies erfolgt innerhalb eines Controllers, aber zur Vollständigkeit wird sie kurz erläutert. Vom generellen Aufbau gleicht sie der *Master.view.xml*. Sie ist auch im XML Format definiert, enthält aber ein paar andere Komponenten. Sämtliche Komponenten liegen wieder in einer *Page*. Begonnen wird mit einem *ObjectHeader*, dieser setzt den Titel des anzuzeigenden Listeneintrags der Hauptseite. Daneben werden noch weitere Information aufgelistet. Unter diesem Kopfteil ist eine *IconTabBar* platziert. Diese Leiste enthält zwei Einträge um so später Zugriff auf die beiden Charts zu erhalten. Um auch hier wieder zu kapseln besitzen die Einträge nur ein Bild und einen verweis auf sogenanntes Fragment. Mit diesem Fragment wird jeweils das Chart definiert. Bedeutet die Art des Diagramms und die jeweils dazu spezifischen Einstellungen. Auf ein Listing wird an dieser Stelle verzichtet.

3.3.3 Model

Für die prototypische Implementierung wurde auf ein JSON Model gesetzt. So kann die Applikation ohne eine tatsächliche Anbindung an ein SAP System im Backend getestet werden. Der Aufbau eines JSON Model im Vergleich zu einem OData Model ist überwiegend gleich bis auf die Tatsache, das ein OData Model noch Metadaten beinhaltet zum OData Service selbst. Diese Metadaten sind zum Testen der Applikation jedoch nicht von nöten. Bei einem späteren produktiv Einsatz dieser Applikation kann durch Veränderung einer Zeile Code das JSON Model durch ein OData Model ersetzt werden. Natürlich sollte das OData Model demzufolge die selbe Struktur besitzen wie das JSON Model, da es ansonsten auch zu komplikationen kommen könnte. Des Weiteren wurde bewusst kein OData Model verwendet, da es vom SAP Netweaver Gateway bereitgestellt wird und darauf im Rahmen dieser Arbeit nicht eingegangen werden sollte. Neben dem JSON Model, als eigentliche Datenbasis, kommt noch ein Resource Model und ein weiteres JSON Model zum Einsatz. Das Resource Model repräsentiert die Lokalisierung. Dadurch können Strings in den verschiedenen Views dynamisch gesetzt werden und je nach Spracheinstellung des dahinter liegenden SAP Systems wird dann die lokalisierte Variante des Stringwerts verwendet. Die Lokalisierungsdateien liegen in dem *i18n* Verzeichnis welches zu

beginn erstellt wurde. Das zweite JSON Model stellt ein gerätespezifisches Model dar. Das bedeutet es werden Metainformationen gespeichert über das Gerät mit dem die Applikation abgerufen wird. Die dazu nötigen Funktionen werden von der jQuery Bibliothek bereitgestellt. Die Applikation kann mit diesem Model ihr visuelles Erscheinungsbild je nach Situation anpassen. In Listing 36 sind die unterschiedlichen Modelle zu sehen, welche in der Datei *Component.js* angelegt werden. Jedes Model wird nach der Erzeugung an die Root View registriert damit es im gesamten Applikationskontext zur Verfügung steht.

```
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(<OData Service 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 36: Model an die Root View binden

3.3.4 Controller

Je View wird mindestens ein Controller vorausgesetzt. Durch den Aufbau der Applikation sind drei Controller nötig. Die Root View benötigt einen allgemeinen Controller der auch für die beiden anderen Views zuständig ist. Die Haupt- und Detailseite haben jeweils wieder ihren eigenen Controller der nur auf Benutzeraktionen der jeweiligen View reagiert. Einerseits um Daten aus dem Model zu beziehen, andererseits um Navigationsaktionen an den Root Controller weiterzugeben.

Controller Root View

Der Root Controller implementiert zwei Funktionen zur Navigation. Mit diesen beiden Funktionen lässt sich beispielsweise von der Detailseite zurück auf die Hauptseite navigieren und andersherum. Des Weiteren veranlasst die *to* Funktion die Applikation dazu beim ersten Aufruf direkt die Hauptseite zu laden damit kein leerer Bildschirm angezeigt wird. Dieser Mechanismus ist in Listing 5 gezeigt.

```
...
to : function (pageId, context) {
  var app = this.getView().app;

  // load page on demand
  var master = ("Master" === pageId);
  if (app.getPage(pageId, master) === null) {
    var page = sap.ui.view({
      id : pageId,
      viewName : "abat.Mockup.view." + pageId,
      type : "XML"
    });
        page.getController().nav = this;
        app.addPage(page, master);
        jQuery.sap.log.info("app controller > loaded page: " + pageId);
  }

  // show the page
  app.to(pageId);

  // set data context on the page
  if (context) {
    var page = app.getPage(pageId);
    page.setBindingContext(context);
  }
},
...
```

Listing 37: Navigationsmechanismus

Controller Hauptseite

Im Controller der Hauptseite sind sämtliche *Handler* untergebracht die zu Komponenten der Seite gehören. Ein relativ einfach zu implementierender, aber trotzdem absolut wichtiger *Handler* ist *handleListItemPress*. Dieser *Handler* wird ausgeführt sobald in der Liste der Hauptseite ein Eintrag gedrückt bzw. ausgewählt wird. Bevor direkt die Detailseite aufgerufen wird holt sich die Funktion noch den aktuellen Datenkontext des Eintrags. Diesen Kontext gibt er an die Navigationsfunktion aus dem Root Controller weiter. Dadurch muss der Controller der Detailseite nicht erst die

Daten beschaffen sondern die Detail View kann die Informationen direkt aus dem übergebenen Kontext entnehmen und darstellen. Das reduziert den Datentraffic zwischen Controllern und Model und dem entsprechend auch die Verbindungen zum Backend System. Ein weiterer *Handler* ist implementiert um die Suchfunktion der Liste zu realisieren. Sobald eine Suche gestartet wird reagiert diese Funktion darauf und filtert die Liste anhand des eingegeben Suchwortes. Dazu stellt das SAP UI5 Framework ein Filter Objekt zur Verfügung. Diesem Filter Objekt übergibt man die Eigenschaft aus dem Model nach der gefiltert werden soll. Der zweite Parameter ist eine Konstante aus dem SAP UI5 Framework, die den entsprechenden Filter Operator angibt. In diesem Fall wird geprüft ob die Eigenschaft den Suchbegriff enthält. Der letzte Parameter ist das Suchwort selbst. In Listing 38 sind die beschriebenen *Handler* Funktionen abgebildet.

```
...
  // Handler Listenelemnte
  handleListItemPress : function (evt) {
    var context = evt.getSource().getBindingContext();
    this.nav.to("Detail", context);
  },
  handleListSelect : function (evt) {
    var context = evt.getParameter("listItem").getBindingContext();
    this.nav.to("Detail", context);
  },

  // Handler Suchfunktion
  handleSearch : function (evt) {
    // create model filter
    var filters = [];
    var query = evt.getParameter("query");
    if (query && query.length > 0) {
      var filter = new sap.ui.model.Filter("MaterialName",
                  sap.ui.model.FilterOperator.Contains, query);
      filters.push(filter);
    }
    // update list binding
    var list = this.getView().byId("list");
    var binding = list.getBinding("items");
    binding.filter(filters);
  },
...
```

Listing 38: Navigation und Suchfunktion der Hauptseite

Da die Liste auch eine Gruppierungsmöglichkeit bieten soll wurde ein weiterer *Handler* implementiert. Dieser *Handler* reagiert sobald auf der Hauptseite der Button unter der Liste gedrückt wird. Die Funktion erzeugt daraufhin ein Dialogfens-

ter und stellt direkt dazu die Eigenschaften ein. So kann anhand des Bruttopreises und des Status gruppiert werden. Außerdem benötigt der Dialog noch eine Funktion, welche die Benutzereingabe entsprechend verarbeitet je nach dem für welche Gruppierung sich der Anwender entschieden hat. Um diese Entscheidung korrekt umzusetzen wurde ein Hilfsskript mit implementiert was im *util* Verzeichnis als Datei *Grouper.js* abgelegt ist. Dieses Hilfsskript liefert je nach Gruppierungskriterium die entsprechenden Werte aus dem Kontext. Da im Dialog auch eine Auswahl bezüglich der Gruppierungsrichtung vorhanden ist wird auch das mit in die Gruppierung einbezogen. SAP UI5 stellt auch dafür eine fertige Funktion zur Verfügung. Listing 39 zeigt die implementierte Funktion aus dem Controller.

```
...
  // Handler Listengruppierung
  handleViewSettings : function (evt) {
    var that = this;
    if (!this._lineItemViewDialog) {
      // create dialog
      this._lineItemViewDialog = new sap.m.ViewSettingsDialog({
        groupItems : [
          new sap.m.ViewSettingsItem({
          text : "Price",
          key : "GrossAmount"
          }),
          new sap.m.ViewSettingsItem({
           text : "Status",
           key : "Status"
          })],
        confirm : function (evt) {
          var aSorters = [];
          var mParams = evt.getParameters();
          if (mParams.groupItem) {
            var sPath = mParams.groupItem.getKey();
            var bDescending = mParams.groupDescending;
            var vGroup = abat.Mockup.util.Grouper[sPath];
            aSorters.push(new sap.ui.model.Sorter(sPath, bDescending, vGroup));
          }
          var oBinding = that.getView().byId("list").getBinding("items");
          oBinding.sort(aSorters);
        }
      });
    }
    this._lineItemViewDialog.open();
  }
...
```

Listing 39: Handler der Listengruppierung

Controller Detailseite

Der Controller der Detailseite hat als wichtigste Aufgabe die Charts zu konfigurieren, die unter der Kopfzeile angezeigt werden. Im SAP UI5 Framework ist eine eigene Chart Visualisierungs Klasse enthalten, welche einen großen Umfang an verschiedensten Chartstyles besitzt. Im Prototyp wurden zunächst nur Bar- und Line-Charts verwendet. Ein Chart benötigt eine Datenbasis auf die es sich stützt. Diese Datenbasis bildet im Prototyp das globale Model. Da die Detailseite bei einem Aufruf aus der Hauptseite automatisch den Datenkontext des gewählten Listeneintrags mitbekommt kann in der Datenbasis des Charts, *FlattenedDataset*, einfach auf einen Pfad innerhalb des JSON Model bezug genommen werden. Daneben werden noch die Achsen entsprechend der Verwendung eingestellt und benannt. Anhand der in der View festgelegten ID wird das Chart Objekt dann referenziert. Es wird ein Titel festgelegt, welcher auch über eine lokalisierte Variante verfügt. Zuletzt wird das erstellte *FlattenedDataset* an das Chart gebunden.

```
...
handleNavButtonPress : function (evt) {
  this.nav.back("Master");
},

...

configForecastChart : function () {
  var oDatasetForecast = new sap.viz.ui5.data.FlattenedDataset({
  dimensions : [{
    axis : 1,
    name : 'Month',
    value : "{Month}"
  }],
  measures : [{
    name : 'Menge1',
    value : '{Menge1}'
    },{
    name : 'Menge2',
    value : '{Menge2}'
  }],
  // data comes from model
  data : {
    path : "ForecastData"
  }
  });

  oLineChart = this.getView().byId("lineChart");
  oLineChart.setTitle(new sap.viz.ui5.types.Title(
        {visible:true, text:"{i18n>ForecastChartTitle}"}));
  oLineChart.setDataset(oDatasetForecast);
```

```
32 } ,
```

Listing 40: Chart Konfigurierung

3.3.5 Deployment

Sofern das Backend SAP System entsprechend vorbereitet ist, bedeutet das alle SAP UI5 Addons und Patches eingespielt und aktiviert sind, verläuft das Deployment der SAP UI5 Applikation relativ schnell und einfach. Eclipse bietet dazu eine eigene Funktion an mit der sich ein angelegtes Projekt auf verschiedenste Plattformen deployen lässt. In Abbildung 9 ist der entsprechende Menüeintrag zu sehen, welcher über einen Rechtsklick auf das Projekt zu erreichen ist.

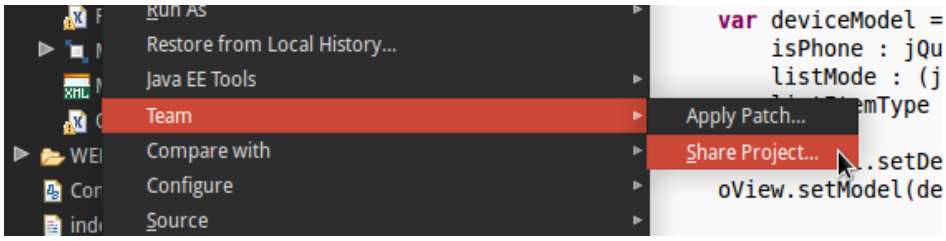

Abbildung 9: Eclipse Menüeintrag zum deployen des Projekts

Durch die SAP UI5 Plugins steht unter dieser Funktion auch ein SAP System als Zielplattform zur Verfügung. Nach der Auswahl des SAPUI5 ABAP Repository erscheint ein weiteres Fenster in welchem die Verbindungsdaten des SAP Systems eingetragen wurden. Nach erfolgreichem Verbindungstest werden noch einige zusätzliche Informationen zu dem SAP System angezeigt. Im nächsten Schritt verlangt Eclipse die Eingabe der SAP Logindaten.

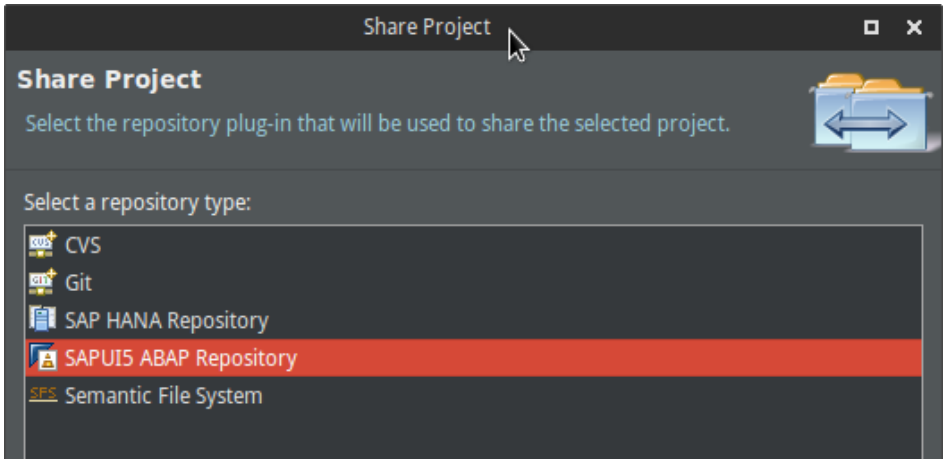

Abbildung 10: Remote Repository Auswahl in Eclipse

Ist Eclipse vollständig mit dem SAP System verbunden kann das Projekt entweder in eine schon existierende BSP Applikation deployed werden oder es kann auch eine komplett neue BSP Applikation über Eclipse auf dem SAP System angelegt werden. Letztere Möglichkeit wurde mit dem Prototyp verwendet. Name, Beschreibung und ein Paket sind dazu nötig. Der Name und das Paket müssen sich im Z* Namensraum befinden. Die Beschreibung kann frei gewählt werden. Einen Schritt weiter erfolgt die Auswahl eines Transportauftrags, sofern einer vorhanden ist im System. Daraufhin lässt sich über einen erneuten Rechtsklick auf das Projekt selbiges an das zuvor eingestellte SAP System schicken. Auf dem SAP System selbst wird danach die BSP Applikation angelegt und der entsprechende Service in der Transaktion *SICF* erstellt und aktiviert. Ab diesem Zeitpunkt ist der Prototyp unter der entsprechenden URL im Browser aufrufbar.

4 Objektorientierte Analyse

// Struktur des Kapitels

4.1 Objekt Beziehungen

// Klassendiagramm erstellen und beschreiben
// Komponentendiagramm erstellen und beschreiben

4.2 Nachrichtenfluss

// Sequenzdiagramm erstellen und beschreiben

4.3 PLATZHALTER

// Datenflussdiagramm erstellen und beschreiben // oder anderes gutes Diagramm

5 Fazit

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

Literaturverzeichnis

[AGa] AG, SAP: *Building SAP Fiori-like UIs with SAPUI5 in 10 Exercises*. http://scn.sap.com/docs/DOC-51167. – Zugriff: 26.1.2015

[AGb] AG, SAP: *SAP Fiori - Architecture Overview*. http://scn.sap.com/docs/DOC-46218. – Zugriff: 20.1.2015

[AGc] AG, SAP: *SAP User Interface Technologies - Road Map*. http://scn.sap.com/docs/DOC-41321. – Zugriff: 18.1.2015

[AGd] AG, SAP: *SAP's user experience strategy*. http://experience.sap.com/ux-strategy/. – Zugriff: 18.1.2015

[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

[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

[Gar] GARRET, Jesse J.: *Ajax: A New Approach to Web Applications*. http://www.adaptivepath.com/ideas/ajax-new-approach-web-applications/. – Zugriff: 18.1.2015

[Inc] INC., Google: *Chrome DevTools Overview*. https://developer.chrome.com/devtools. – Zugriff: 04.01.2015

[Kroa] KROENER, Peter: *HTML5 Spezifikations Graph*. https://github.com/SirPepe/SpecGraph. – Zugriff: 11.12.2014

[Krob] KROPFF, Peter: *Ajax Prinzip*. http://www.peterkropff.de/site/javascript/ajax.htm. – Zugriff: 19.1.2015

[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

[Selb] SELFHTML: *CSS Einbindung*. http://wiki.selfhtml.org/wiki/CSS/Einbindung. – Zugriff: 07.12.2014

[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

[Seld] SELFHTML: *JavaScript / Einführung in JavaScript*. http://de.selfhtml.org/javascript/intro.htm. – Zugriff: 09.12.2014

[Sele] SELFHTML: *JavaScript / Objektreferenzen*. http://de.selfhtml.org/javascript/objekte/index.htm. – Zugriff: 17.12.2014

[Self] SELFHTML: *Physische Auszeichnungen im Text*. http://de.selfhtml.org/html/text/physisch.htm. – Zugriff: 01.12.2014

[Selg] SELFHTML: *Suchmaschinenoptimierung*. http://wiki.selfhtml.org/wiki/Suchmaschinenoptimierung. – Zugriff: 04.12.2014

[Sur] SURVEY, W3 T.: *Usage Statistics and Market Share of JavaScript Libraries for Websites*. http://w3techs.com/technologies/overview/javascript_library/all. – Zugriff: 19.1.2015

[w3S] W3SCHOOLS: *CSS Selectors*. http://www.w3schools.com/css/css_selectors.asp. – Zugriff: 09.12.2014

[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

[Wikb] WIKIPEDIA: *Eclipse (IDE)*. http://de.wikipedia.org/wiki/Eclipse_(IDE). – Zugriff: 02.01.2015

[Wikc] WIKIPEDIA: *JavaScript - Sandbox-Prinzip*. http://de.wikipedia.org/wiki/JavaScript. – Zugriff: 11.12.2014

[Wikd] WIKIPEDIA: *jQuery*. http://de.wikipedia.org/wiki/JQuery. – Zugriff: 19.1.2015

[Wike] WIKIPEDIA: *Representational State Transfer (REST)*. http://de.wikipedia.org/wiki/Representational_State_Transfer. – Zugriff: 20.1.2015

[Wis] WISSEN, IT: *jQuery*. http://www.itwissen.info/definition/lexikon/jQuery.html. – Zugriff: 19.1.2015

Anhang

index.html

```
<!DOCTYPE html>
<html>
<head>
<meta http-equiv="X-UA-Compatible" content="IE=edge" />
<meta http-equiv='Content-Type' content='text/html;charset=UTF-8' />

<title>abat AG - Mockup Claas</title>

<script id="sap-ui-bootstrap" src="resources/sap-ui-core.js"
        data-sap-ui-theme="sap_bluecrystal"
        data-sap-ui-libs="sap.m, sap.ui.layout"
        data-sap-ui-preload=""
        data-sap-ui-xx-bindingSyntax="complex"
        data-sap-ui-resourceroots='{
                                    "abat.Mockup": "./"
                            }'>
</script>

<script>
  new sap.ui.core.ComponentContainer({
    name : "abat.Mockup"
  }).placeAt("content");
</script>

</head>
<body class="sapUiBody" role="application">
        <div id="content"></div>
</body>
</html>
```

Component.js

```
jQuery.sap.declare("abat.Mockup.Component");

sap.ui.core.UIComponent.extend("abat.Mockup.Component", {

  createContent : function() {

    // create root view
    var oView = sap.ui.view({
      id : "app",
      viewName : "abat.Mockup.view.App",
      type : "JS",
      viewData : {
        component : this
      }
    });

    // set data model on root view
    var oModel = new sap.ui.model.json.JSONModel("model/mock.json");
    oView.setModel(oModel);

    // set i18n model
    var i18nModel = new sap.ui.model.resource.ResourceModel({
      bundleUrl : "i18n/messageBundle.properties"
    });
    oView.setModel(i18nModel, "i18n");

    // set device model
    var deviceModel = new sap.ui.model.json.JSONModel({
      isPhone : jQuery.device.is.phone,
      listMode : (jQuery.device.is.phone) ? "None" : "SingleSelectMaster",
      listItemType : (jQuery.device.is.phone) ? "Active" : "Inactive"
    });
    deviceModel.setDefaultBindingMode("OneWay");
    oView.setModel(deviceModel, "device");

    // done
    return oView;
  }
});
```

App.view.js

```
sap.ui.jsview("abat.Mockup.view.App", {

  getControllerName : function() {
    return "abat.Mockup.view.App";
  },

  createContent : function(oController) {

    // to avoid scroll bars on desktop the root view must be set to block
    // display
    this.setDisplayBlock(true);

    // create app
    this.app = new sap.m.SplitApp();

    // load the master page
    var master = sap.ui.xmlview("Master", "abat.Mockup.view.Master");
    master.getController().nav = this.getController();
    this.app.addPage(master, true);

    // load the empty page
    var empty = sap.ui.xmlview("Empty", "abat.Mockup.view.Empty");
    this.app.addPage(empty, false);

    // wrap app with shell
    return new sap.m.Shell("Shell", {
      title : "{i18n>ShellTitle}",
      showLogout : false,
      app : this.app
    });
  }
});
```

App.controller.js

```
sap.ui.controller("abat.Mockup.view.App", {

  /**
   * Navigates to another page
   *
   * @param {string} pageId The id of the next page
   * @param {sap.ui.model.Context} context The data context
   */
  to : function(pageId, context) {

    var app = this.getView().app;

    // load page on demand
    var master = ("Master" === pageId);
    if (app.getPage(pageId, master) === null) {
      var page = sap.ui.view({
        id : pageId,
        viewName : "abat.Mockup.view." + pageId,
        type : "XML"
      });
      page.getController().nav = this;
      app.addPage(page, master);
      jQuery.sap.log.info("app controller > loaded page: " + pageId);
    }

    // show the page
    app.to(pageId);

    // set data context on the page
    if (context) {
      var page = app.getPage(pageId);
      page.setBindingContext(context);
    }
  },

  /**
   * Navigates back to a previous page
   *
   * @param {string} pageId The id of the next page
   */
  back : function(pageId) {
    this.getView().app.backToPage(pageId);
  }
});
```

Master.view.xml

```
<core:View controllerName="abat.Mockup.view.Master" xmlns="sap.m"
  xmlns:core="sap.ui.core">
  <Page>
    <customHeader>
      <Bar>
        <contentLeft>
          <Image src="img/logo.png" width="173px" height="30px"></Image>
        </contentLeft>
      </Bar>
    </customHeader>
    <subHeader>
      <Bar>
        <contentLeft>
          <SearchField search="handleSearch" width="100%">
          </SearchField>
        </contentLeft>
      </Bar>
    </subHeader>
    <List id="list" mode="{device>/listMode}" select="handleListSelect"
      items="{/Products}">
      <ObjectListItem type="{device>/listItemType}"
        press="handleListItemPress"
        title="{MaterialName}"
        number="{Quantity}"
        numberUnit="{i18n&gt;QuantityUnit}"
        numberState="{parts : [ 'Quantity', 'MinimalQuantity' ],
          formatter : 'abat.Mockup.util.Formatter.numberState'}">
        <attributes>
          <ObjectAttribute text="{MatId}" />
        </attributes>
        <firstStatus>
          <ObjectStatus text="{Status}"
            state="{path: 'Status',
              formatter: 'abat.Mockup.util.Formatter.statusState'}" />
        </firstStatus>
      </ObjectListItem>
    </List>
    <footer>
      <Bar>
        <contentRight>
          <Button icon="sap-icon://group-2" press="handleViewSettings" />
        </contentRight>
      </Bar>
    </footer>
  </Page>
</core:View>
```

Master.controller.js

```
jQuery.sap.require("abat.Mockup.util.Formatter");
jQuery.sap.require("abat.Mockup.util.Grouper");

sap.ui.controller("abat.Mockup.view.Master", {

  onExit : function() {
    if (this._lineItemViewDialog) {
      this._lineItemViewDialog.destroy();
      this._lineItemViewDialog = null;
    }
  },

  handleListItemPress : function(evt) {
    var context = evt.getSource().getBindingContext();
    this.nav.to("Detail", context);
  },

  handleSearch : function(evt) {
    // create model filter
    var filters = [];
    var query = evt.getParameter("query");
    if (query && query.length > 0) {
      var filter = new sap.ui.model.Filter("MaterialName",
          sap.ui.model.FilterOperator.Contains, query);
      filters.push(filter);
    }

    // update list binding
    var list = this.getView().byId("list");
    var binding = list.getBinding("items");
    binding.filter(filters);
  },

  handleListSelect : function(evt) {
    // get binding context and nav to detail page
    var context = evt.getParameter("listItem").getBindingContext();
    this.nav.to("Detail", context);
  },

  handleViewSettings : function(evt) {
    var that = this;
    if (!this._lineItemViewDialog) {
      // create dialog
      this._lineItemViewDialog = new sap.m.ViewSettingsDialog({
        groupItems : [ new sap.m.ViewSettingsItem({
          text : "Price",
```

```
47            key : "GrossAmount"
48          }), new sap.m.ViewSettingsItem({
49            text : "Status",
50            key : "Status"
51          }) ],
52          confirm : function(evt) {
53            var aSorters = [];
54            var mParams = evt.getParameters();
55            if (mParams.groupItem) {
56              var sPath = mParams.groupItem.getKey();
57              var bDescending = mParams.groupDescending;
58              var vGroup = abat.Mockup.util.Grouper[sPath];
59              aSorters.push(
60                new sap.ui.model.Sorter(sPath, bDescending, vGroup));
61            }
62            var oBinding = that.getView().byId("list").getBinding("items");
63            oBinding.sort(aSorters);
64          }
65        });
66      }
67
68      // open dialog
69      this._lineItemViewDialog.open();
70    }
71  });
```

Detail.view.xml

```
<core:View controllerName="abat.Mockup.view.Detail" xmlns="sap.m"
  xmlns:core="sap.ui.core">
  <Page title="{i18n>DetailTitle}"
    class="sapUiFioriObjectPage"
    showNavButton="{device>/isPhone}"
    navButtonPress="handleNavButtonPress">
    <ObjectHeader title="{MaterialName}" number="{Quantity}"
      numberUnit="{i18n&gt;QuantityUnit}"
      numberState="{parts: [ 'Quantity', 'MinimalQuantity' ],
        formatter: 'abat.Mockup.util.Formatter.numberState'}"
      icon="img/placeholder.gif">
      <attributes>
        <ObjectAttribute
          text="{i18n&gt;MatId}: {MatId}" />
        <ObjectAttribute
          text="{i18n&gt;GrossAmount}: {GrossAmount} {CurrencyCode}" />
        <ObjectAttribute
          text="{i18n&gt;TaxAmount}: {TaxAmount} {CurrencyCode}" />
        <ObjectAttribute
          text="{i18n&gt;NetAmount}: {NetAmount} {CurrencyCode}" />
      </attributes>
      <firstStatus>
        <ObjectStatus text="{Status}"
          state="{path: 'Status',
                  formatter: 'abat.Mockup.util.Formatter.statusState'}" />
      </firstStatus>
    </ObjectHeader>
    <IconTabBar expanded="{device>/isNoPhone}">
      <items>
        <IconTabFilter text="{i18n&gt;IconTab_1}"
          icon="sap-icon://vertical-bar-chart-2">
          <content>
            <core:Fragment type="XML"
              fragmentName="abat.Mockup.view.ForecastChart"></core:Fragment>
          </content>
        </IconTabFilter>
        <IconTabFilter text="{i18n&gt;IconTab_2}" icon="sap-icon://line-chart">
          <content>
            <core:Fragment type="XML"
              fragmentName="abat.Mockup.view.OrderProposalChart"></core:Fragment>
          </content>
        </IconTabFilter>
      </items>
    </IconTabBar>
    <footer>
      <Bar>
```

```
47          <contentRight>
48            <Button text="{i18n>ApproveButtonText}" icon="sap-icon://accept"
49              press="handleApprove" />
50          </contentRight>
51        </Bar>
52      </footer>
53    </Page>
54  </core:View>
```

Detail.controller.js

```
jQuery.sap.require("abat.Mockup.util.Formatter");
jQuery.sap.require("sap.m.MessageBox");
jQuery.sap.require("sap.m.MessageToast");

sap.ui.controller("abat.Mockup.view.Detail", {

  onInit : function(evt) {
    this.configForecastChart();
    this.configOrderProposalChart();
  },

  handleForecastDataSelect : function(evt) {
    var xAxisIndex = (evt.getParameter("data")[0]).data[0].ctx.path.dii_a1;
    var oContext = this.getView().byId("lineChart").getBindingContext();
    var oSelectedData = this.getView().getModel().getProperty(
        oContext.sPath + "/ForecastData/" + xAxisIndex);

    console.log(oSelectedData);
  },

  handleOrderProposalDataSelect : function(evt) {
    // TODO write code to make Chart dynamic
  },

  handleNavButtonPress : function(evt) {
    this.nav.back("Master");
  },

  handleApprove : function(evt) {
    // show confirmation dialog
    var bundle = this.getView().getModel("i18n").getResourceBundle();
    sap.m.MessageBox.confirm(bundle.getText("ApproveDialogMsg"), function(
        oAction) {
      if (sap.m.MessageBox.Action.OK === oAction) {
        // notify user
        var successMsg = bundle.getText("ApproveDialogSuccessMsg");
        sap.m.MessageToast.show(successMsg);
        // TODO call proper service method and update model
      }
    },

    bundle.getText("ApproveDialogTitle"));
  },

  configForecastChart : function() {
    var oDatasetForecast = new sap.viz.ui5.data.FlattenedDataset({
```

```
47        dimensions : [ {
48          axis : 1,
49          name : 'Month',
50          value : "{Month}"
51        } ],
52        measures : [ {
53          name : 'Menge1',
54          value : '{Menge1}'
55        }, {
56          name : 'Menge2',
57          value : '{Menge2}'
58        } ],
59        data : {
60          path : "ForecastData"
61        }
62      });
63
64      oLineChart = this.getView().byId("lineChart");
65      oLineChart.setTitle(new sap.viz.ui5.types.Title({
66        visible : true,
67        text : "{i18n>ForecastChartTitle}"
68      }));
69      oLineChart.setDataset(oDatasetForecast);
70    },
71
72    configOrderProposalChart : function() {
73      var oDatasetOrderProposal = new sap.viz.ui5.data.FlattenedDataset({
74        dimensions : [ {
75          axis : 1,
76          name : 'Month',
77          value : "{Month}"
78        } ],
79        measures : [ {
80          name : 'Menge1',
81          value : '{Menge1}'
82        } ],
83        data : {
84          path : "OrderProposalData"
85        }
86      });
87
88      oBarChart = this.getView().byId("barChart");
89      oBarChart.setTitle(new sap.viz.ui5.types.Title({
90        visible : true,
91        text : "{i18n>OrderProposalChartTitle}"
92      }));
93      oBarChart.setDataset(oDatasetOrderProposal);
94    }
```

```
95 });
```

ForecastChart.fragment.xml

```
1 <core:FragmentDefinition xmlns:core="sap.ui.core"
2   xmlns:mvc="sap.ui.core.mvc"
3   xmlns="sap.m"
4   xmlns:html="http://www.w3.org/1999/xhtml"
5   xmlns:viz="sap.viz.ui5"
6   xmlns:types="sap.viz.ui5.types">
7   <viz:Line id="lineChart" width="100%" height="400px"
8     selectData="handleForecastDataSelect"></viz:Line>
9 </core:FragmentDefinition>
```

OrderProposalChart.fragment.xml

```
1 <core:FragmentDefinition xmlns:core="sap.ui.core"
2   xmlns:mvc="sap.ui.core.mvc"
3   xmlns="sap.m" xmlns:html="http://www.w3.org/1999/xhtml"
4   xmlns:viz="sap.viz.ui5"
5   xmlns:types="sap.viz.ui5.types">
6   <viz:Bar id="barChart" width="100%" height="400px"
7     selectData="handleOrderProposalDataSelect"></viz:Bar>
8 </core:FragmentDefinition>
```

Empty.view.xml

```
1 <core:View xmlns="sap.m" xmlns:core="sap.ui.core">
2   <Page>
3     <footer>
4       <Bar>
5       </Bar>
6     </footer>
7   </Page>
8 </core:View>
```

Formatter.js

```
jQuery.sap.declare("abat.Mockup.util.Formatter");

jQuery.sap.require("sap.ui.core.format.DateFormat");

abat.Mockup.util.Formatter = {

  _statusStateMap : {
    "lieferbar" : "Success",
    "bald lieferbar" : "Warning",
    "ausverkauft" : "Error"
  },

  statusState : function(value) {
    var map = abat.Mockup.util.Formatter._statusStateMap;
    return (value && map[value]) ? map[value] : "None";
  },

  numberState : function(stock, minStock) {
    return (parseInt(stock) <= parseInt(minStock)) ? "Error" : "None";
  },

  date : function(value) {
    if (value) {
      var oDateFormat = sap.ui.core.format.DateFormat.getDateTimeInstance({
        pattern : "yyyy-MM-dd"
      });
      return oDateFormat.format(new Date(value));
    } else {
      return value;
    }
  },

  quantity : function(value) {
    try {
      return (value) ? parseFloat(value).toFixed(0) : value;
    } catch (err) {
      return "Not-A-Number";
    }
  }
};
```

Grouper.js

```
jQuery.sap.declare("abat.Mockup.util.Grouper");

abat.Mockup.util.Grouper = {

  Status : function(oContext) {
    var status = oContext.getProperty("Status");
    return {
      key : status,
      text : status
    };
  },

  GrossAmount : function(oContext) {
    var price = oContext.getProperty("GrossAmount");
    var currency = oContext.getProperty("CurrencyCode");
    var key, text;
    if (price <= 5000) {
      key = "A-LE10";
      text = "< 5000 " + currency;
    } else if (price <= 10000) {
      key = "B-LE100";
      text = "< 10.000  " + currency;
    } else {
      key = "C-GT100";
      text = "> 10.000 " + currency;
    }
    return {
      key : key,
      text : text
    };
  }
};
```

mock.json

```
{
    "Products": [
        {
            "MatId": "100000000002082980",
            "MaterialName": "Produkt 1",
            "Status": "lieferbar",
            "Quantity": "251",
            "MinimalQuantity": "100",
            "GrossAmount": "13224.47",
            "NetAmount": "11113.00",
            "TaxAmount": "2111.47",
            "CurrencyCode": "EUR",
            "CreatedAt": "2013-05-22T22:00:00",
            "ChangedAt": "2013-05-22T22:00:00.000Z",
            "CreatedByBp": "EPM USER",
            "ChangedByName": "EPM USER",
            "ForecastData": [
                {
                    "Month": "Januar",
                    "Menge1": "289",
                    "Menge2": "967"
                },
                ...
                {
                    "Month": "Dezember",
                    "Menge1": "719",
                    "Menge2": "857"
                }
            ],
            "OrderProposalData": [
                {
                    "Month": "Januar",
                    "Menge1": "343",
                    "Menge2": "837"
                },
                ...
                {
                    "Month": "Dezember",
                    "Menge1": "417",
                    "Menge2": "761"
                }
            ]
        }
```

messageBundle.properties

```
ShellTitle=Sales Orders App
MasterTitle=Products
DetailTitle=Sales Order
ApproveButtonText=Approve
ApproveDialogTitle=Approve Sales Order
ApproveDialogMsg=Do you want to approve this sales order now?
ApproveDialogSuccessMsg=The sales order has been approved
LineItemTableHeader=Products
LineItemTitle=Product
IconTab_1=Forecast
IconTab_2=Ordering proposal
ForecastChartTitle=Forecast
OrderProposalChartTitle=Ordering Proposal
QuantityUnit=pc.
GrossAmount=Gross Amount
TaxAmount=Tax Amount
NetAmount=Net Amount
MatId=Mat. ID
```

messageBundle_de.properties

```
ShellTitle=Sales Orders App
MasterTitle=Produkte
DetailTitle=Produkt
ApproveButtonText=Approve
ApproveDialogTitle=Approve Sales Order
ApproveDialogMsg=Do you want to approve this sales order now?
ApproveDialogSuccessMsg=The sales order has been approved
LineItemTableHeader=Products
LineItemTitle=Product
IconTab_1=Prognose
IconTab_2=Bestellvorschau
ForecastChartTitle=Prognose
OrderProposalChartTitle=Bestellvorschau
QuantityUnit=Stk.
GrossAmount=Brutto Betrag
TaxAmount=MwSt.
NetAmount=Netto Betrag
MatId=Mat. ID
```

Eidesstattliche Erklärung

Eidesstattliche Erklärung zur Bachelorarbeit

Ich versichere, die von mir vorgelegte Arbeit selbstständig verfasst zu haben. Alle Stellen, die wörtlich oder sinngemäß aus veröffentlichten oder nicht veröffentlichten Arbeiten anderer entnommen sind, habe ich als entnommen kenntlich gemacht. Sämtliche Quellen und Hilfsmittel, die ich für die Arbeit benutzt habe, sind angegeben. Die Arbeit hat mit gleichem Inhalt bzw. in wesentlichen Teilen noch keiner anderen Prüfungsbehörde vorgelegen.

$Unterschrift:$ $Ort, Datum:$