GRUNDLAGEN • PRAXIS • ARCHITEKTUR • AGILE • INNOVATION

März-Juni 1 / 2017

# JAVA PRO

Magazin für professionelle Java Entwicklung in der Praxis

www.java-pro.de







www.java-pro.de

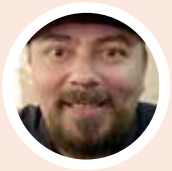
www.java-pro.de

#Javapro #WebComponents #Polymer

# Web-Components mit Polymer

Web-Components ist ein Standard, der komponentenbasierte Softwareentwicklung für das Web ermöglicht. Der folgende Workshop zeigt, wie man mittels Web-Components und Polymer wiederverwendbare Komponente entwickelt, die in sich gekapselt sind, die sich kombinieren und erweitern lassen.

**Autor:**



Marcus Fihlon arbeitet als Agile Coach und Software Entwickler bei der CSS Versicherung in Luzern und als Dozent für Web Engineering an der TEKO Schweizerische Fachschule in Olten.

Er organisiert den monatlichen Hackergarten in Luzern und gibt sein Wissen gerne mit Vorträgen und Workshops bei User-Groups und Konferenzen weiter. Als Ausgleich geht Marcus gerne in den Schweizer Bergen wandern und ist auch oft auf seinem heiß geliebten Klapprad anzutreffen.

## Web-Components sind HTML Standard

Die Zeiten, zu denen man als Java-Entwickler auch ausschliesslich Java-Technologien für Benutzeroberflächen eingesetzt hat, sind schon lange vorbei. Über die Jahre hinweg hat HTML immer mehr Einzug gehalten und kann nun als fester Bestandteil des Java-Ökosystems betrachtet werden.

Doch HTML ist nicht gleich HTML: die erste Version erschien bereits vor 25 Jahren, seitdem hat sich sehr viel getan. Quasi unbemerkt hat sich ein Standard für Web-Components regelrecht eingeschlichen, welcher aus vier verschiedenen Spezifikationen besteht.

Dieser Standard spezifiziert, wie wiederverwendbare Widgets und Komponenten für Web-Dokumente und Web-Applikationen erstellt und genutzt werden können. Die Idee dahinter ist, die komponentenbasierte Softwareentwicklung in das World-Wide-Web zu bringen. Das Komponentenmodell erlaubt die Kapselung von HTML-Elementen und deren Interoperabilität. Von vielen Web-Entwicklern nicht beachtet, bieten Web-Components enormes Potential, um auf einfache und schnelle Weise moderne und modulare Benutzeroberflächen mit HTML zu erstellen.

## Bücherwurm

Auf den folgenden Seiten entwickeln wir gemeinsam eine kleine Web-Applikation auf Basis von Web-Components und dem Polymer-Framework. Wir werden dabei sowohl bereits vorhandene Komponenten einsetzen als auch eigene Komponenten entwickeln.

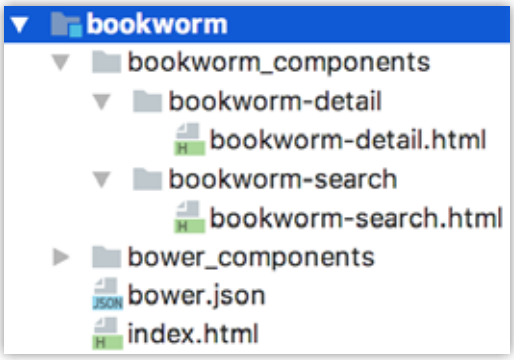
Der Business-Case stellt sich einfach dar: Mittels eines Eingabefeldes soll eine Buchsuche durchgeführt und die Resultate übersichtlich dargestellt werden. Doch was simpel erscheint, ist durchaus anspruchsvoll: Wir teilen die Funktionalitäten auf einzelne Komponenten auf, die miteinander interagieren.

## Projekt Setup

Zuerst müssen wir unser Projekt einrichten. Wir beginnen mit einem leeren Verzeichnis mit dem Namen **bookworm**.
Um uns das Management externen Abhängigkeiten in unserem Projekt einfacher zu machen, nutzen wir die Paketverwaltung Bower. Nach der Installation von Bower initialisieren wir unser Projekt im Projektverzeichnis:

```
bower init
```

Alle Fragen können mit dem Standardwert quittiert werden. Nach einer abschließenden Sicherheitsfrage wird automatisch die neue Datei **bower.json** angelegt.




Verzeichnisstruktur. (Abb. 1)

## Fremdkomponenten einbinden

Im weiteren Verlauf dieses Tutorials nutzen wir einige Komponenten aus dem Polymer-Project, die wir jetzt einbinden. Eine genauere Erläuterung der einzelnen Komponenten folgt später.

```
bower install --save Polymer > polymer PolymerElements > iron-ajax PolymerElements > paper-card PolymerElements > paper-input PolymerElements > paper-material PolymerElements > paper-styles
```

Es wird automatisch ein neues Verzeichnis mit dem Namen **bower_components** angelegt und zusätzlich zu unseren gewünschten Komponenten werden noch etliche Abhängigkeiten heruntergeladen. Alles landet in diesem neuen Verzeichnis und ist somit gleich von unserem eigenen Quelltext separiert.

Bevor wir mit der eigentlichen Entwicklung beginnen können, fehlt uns aber noch ein Browser und ein Editor. Die Wahl des Browsers ist an dieser Stelle sehr wichtig, denn einige Browser benötigen zum Testen von Web-Components eine HTTP-Verbindung. Während Safari auch mit Web-Components über **file://** URLs keinerlei Probleme hat, funktionieren Web-Components in Chrome ausschließlich über eine HTTP(S) Verbindung. Daher empfiehlt sich der Einsatz eines lokalen Webservers oder eines Editors, der über einen integrierten Webserver verfügt. Möglichkeit wäre hier der Einsatz des Editors Atom mit dem Live-Server-Package.

## Unsere erste eigene Komponente

Unsere kleine App realisieren wir als Komponente. Damit kann sie einfach und schnell auf jeder Webseite eingebunden werden. Für unsere eigenen Komponenten erstellen wir im Projektverzeichnis das Unterverzeichnis **bookworm_components.** Dort sammeln wir alle unsere eigenen Komponenten für diese Anwendung.

Als Best Practice hat sich hier erwiesen, für jede Komponente ein eigenes Verzeichnis zu erstellen und dort alle Dateien dieser Komponente abzulegen. Das Verzeichnis sollte dabei den Namen der Komponente tragen. Für jede Komponente wird uns ein Tag zur Verfügung gestellt, mit dem wir die Komponente im HTML-Quelltext einbauen. Dafür wird der Name der Komponente genutzt, der zwingend einen Bindestrich enthalten muss. Da normale Tags keinen Bindestrich enthalten, kann der Browser so erkennen, dass es sich um eine Komponente handelt. Unsere erste Komponente heisst **bookworm-search**, also erstellen wir im soeben erzeugten Verzeichnis noch das Unterverzeichnis **bookworm-search.** Darin legen wir eine leere Datei mit dem Namen **bookworm-search.html** an. Das ist ebenfalls Best Practice: Die Hauptdatei der Komponente hat den gleichen Namen wie die Komponente selbst. Beim späteren Import von Komponenten erleichtert uns diese Konvention der Benennung von Verzeichnissen und Dateien das Finden der entsprechenden Importdatei. In der soeben erstellten HTML Datei beginnen wir nun nicht mit der üblichen HTML5 Struktur, denn wir erstellen keine Webseite, sondern eine Komponente. Mit dem Tag **<dom-module>** definieren wir unsere Komponente und vergeben eine ID, die unsere Komponente benennt und auch als Tag für unsere Komponente fungiert:

```
<dom-module id="bookworm-search">
```

Nun erstellen wir den Inhalt unserer Komponente. Der Einfachheit halber benutzen wir erstmal einen statischen Text. Dazu fügen wir innerhalb des **<dom-module>**-Tag ein **<template>**-Tag ein:

```
<template>
  It works!
</template>
```

Bevor wir unsere Komponente nutzen können, müssen wir sie noch bei Polymer registrieren. Dazu sind zwei Schritte notwendig. Zuerst müssen wir Polymer importieren, damit es uns zur Verfügung steht. Anschließend können wir unsere Komponente mit JavaScript registrieren lassen. Der Import befindet sich dabei, ähnlich einer Java-Klasse, am Anfang der Komponente.

Fügen wir nun also folgenden Import ganz am Anfang unserer Datei vor dem **<dom-module>**-Tag hinzu:

```
<link rel="import“ href="../../bower_components/polymer/polymer.
html“ />
```

Dabei ist der Pfad zu beachten: Bei Imports ist immer die Position der aktuellen Komponente ausschlaggebend, nicht die der Webseite, in der unsere Komponente verwendet wird. Daher müssen wir erst zwei Verzeichnisebenen zurück! Nun ergänzen wir die Registrierung unserer Komponente bei Polymer. Wir übergeben dabei den Tag für unsere Komponente und Polymer kümmert sich darum, den Tag und unsere Komponente dem Browser bekannt zu machen. Dieser Code-Schnipsel gehört zwischen das schließende **</template>** und das schließende **</dom-module>**-Tag:

Kein Umbruch, das gehört zusammen: </dom-module>

```
<script>
  Polymer({
    is: "bookworm-search"
  });
</script>
```

Mit nur 11 Zeilen Code haben wir unsere allererste eigene Komponente auf Basis von Web-Components und Polymer entwickelt.

## Unsere Komponente einbauen

Jetzt wäre es toll, wenn wir unsere erste eigene Komponente in Aktion sehen könnten. Nichts leichter als das. Im Projektverzeichnis erstellen wir ein einfaches HTML5-Dokument mit dem Dateinamen **index.html** und folgendem Inhalt:

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <title>Bookworm</title>
  </head>
  <body>
  </body>
</html>
```

Soweit handelt es sich um ein klassisches HTML5-Grundgerüst. Wenn wir nun Web-Components nutzen möchten, ergibt sich ein kleines Problem: Dieser Standard wird noch nicht von jedem Browser vollumfänglich unterstützt. Chrome und Opera unterstützen diesen Standard in aktuellen Versionen komplett, alle anderen nur teilweise. Doch dafür gibt es eine einfache Lösung: Ein Polyfill ist ein in JavaScript geschriebener Code-Baustein, der in Browsern noch nicht unterstützte Funktionen mittels eines Workarounds nachrüstet. Einen solchen Polyfill für Web-Components binden wir nun im **<head>**-Bereich ein:

```
<script src="bower_components/webcomponentsjs/webcomponents.min.
js"></script>
```

Nun können wir damit starten, unsere Komponente einzubauen. Zuerst müssen wir unsere Komponente importieren, damit der Browser sie kennt. Den Import fügen wir im **<head>**-Bereich hinzu:

```
<link  rel="import"  href="bookworm_components/bookworm-search/
bookworm-search.html" />
```

Einer Nutzung unserer Komponente steht nun nichts mehr im Weg. Wir binden sie einfach mittels ihres eigenen Tags im **<body>**-Bereich ein:

```
<bookworm-search></bookworm-search>
```

Fertig! Rufen wir nun unsere Webseite auf und schauen uns das Ergebnis an: **It works!**

Beim Parsen unserer Webseite stößt der Browser auf den Import unserer Komponente. Ähnlich einem Stylesheet wird unsere Komponente nachgeladen und bringt dem Browser unser Tag **bookworm-search** bei. Später stößt der Browser dann auf unser Tag und fügt an dieser Stelle unsere Komponente ein, welche nichts anderes macht als **It works!** auszugeben. So einfach funktionieren Web-Components mit Polymer.

## Ein Eingabefeld in hübsch

Auch wenn wir auf unseren statischen Text stolz sein können, schaut unser heutiges Ziel etwas anders aus. Wir benötigen ein Eingabefeld. Da wir den Umgang mit Komponenten lernen möchten, nutzen wir nicht das HTML-Input-Element, sondern eine Komponente. In diesem Fall eine von Polymer, welche dem Materialdesign entspricht und etwas hübscher ausschaut. Diese Komponente heißt **paper-input** und wir haben sie bereits am Anfang des Tutorials heruntergeladen.

Erweitern wir also nun unsere eigene Komponente **bookworm-search** um die Fremdkomponente **paper-input**, indem wir diese importieren. Fügen wir unserer Komponente einen zweiten Import hinzu:

Kein Bindestrich an dieser Stelle!
Keine Silbentrennung im Quellcode!

```
<link rel="import" href="../../bower_components/paper-input/paper-input.html" />
```

Nun kennt der Browser diese Fremdkomponente und wir können unseren statischen Text durch das neue Eingabefeld ersetzen:

```
<paper-input label="enter a search term"></paper-input>
```

Diese zwei Schritte reichen schon aus. Nach dem Import und dem Einbau des Tags, funktioniert diese Komponente. Schauen wir uns die Anzeige in einem Browser an:


Das Suchfeld. (Abb. 2)

Wenn wir nun mit der Maus in das Eingabefeld klicken, sehen wir eine Animation. Wenn wir mit der Eingabe beginnen, eine weitere. Diese sind Bestandteil der neuen Komponente, die wir bei uns eingebunden haben.

## Databinding

Damit wir später eine Suche mit dem eingegebenen Begriff durchführen können, müssen wir an den Wert des Eingabefeldes gelangen. Das ist mittels Databinding von Polymer ziemlich einfach. Wir müssen das Eingabefeld nur bitten, den Wert in eine Variable zu schreiben. Dazu geben wir den Variablennamen in doppelten geschweiften Klammern im **value** Attribut an:

```
<paper-input label="enter a search term" value="{{searchterm}}"></paper-input>
```

Damit wir überprüfen können, ob das Databinding funktioniert, können wir den Inhalt der Variablen ausgeben lassen. Dazu können wir die Variable innerhalb des Templates angeben, wieder mit doppelten geschweiften Klammern:

```
{{searchterm}}
```

Kein Bindestrich an dieser Stelle!
Keine Silbentrennung im Quellcode!

Wenn wir nun in das Eingabefeld tippen, erscheint der Suchtext auch darunter als Ausgabe:


Das Suchfeld in Aktion. (Abb. 3)

Nachdem wir das Databinding überprüft haben, können wir die testweise Ausgabe des Suchbegriffes entfernen.

## AJAX mal einfach

Nun möchten wir eine Buchsuche per AJAX durchführen. Glücklicherweise müssen wir uns nicht mehr selbst mit dem **XMLHttpRequest** herumschlagen, das nimmt uns die Komponente **iron-ajax** ab, welche wir bereits am Anfang des Tutorials heruntergeladen haben. Fügen wir den entsprechenden Import hinzu:

```
<link rel="import" href="../../bower_components/iron-ajax/iron-ajax.html" />
```

Bauen wir nun die AJAX-Komponente ein und verwenden in der URL an der richtigen Stelle einfach unsere Variable für das Databinding mit dem Eingabefeld:

```
<iron-ajax url="https://www.googleapis.com/books/v1/volumes?q={{searchterm}}"></iron-ajax>
```

Kein Bindestrich an dieser Stelle!
Keine Silbentrennung im Quellcode!

Wenn wir jetzt etwas in das Eingabefeld tippen, wird es von der **paper-input** Komponente in die Variable searchterm geschrieben. Da sich die Variable ändert, aktualisiert Polymer automatisch die URL der **iron-ajax** Komponente. Wir können nun die AJAX-Komponente bitten, bei jeder Änderung der URL automatisch einen AJAX-Request abzusetzen. Dazu ergänzen wir einfach den Parameter «auto»:

```
<iron-ajax auto url="https://…q={{searchterm}}"></iron-ajax>
```

Wenn wir unsere Webseite im Browser neu laden und jetzt etwas in das Eingabefeld eintippen, dann können wir in der Netzwerkansicht unseres Browsers sehen, dass nach jedem Tastendruck ein neuer Request abgeschickt wird:

| Name | Status | Type | Initiator | Size | Time |
|---|---|---|---|---|---|
| volumes?q=T | 200 | xhr | iron-request.html:304 | 4.7KB | 1.03s |
| volumes?q=Te | 200 | xhr | iron-request.html:304 | 4.0KB | 805ms |
| volumes?q=Tes | 200 | xhr | iron-request.html:304 | 4.0KB | 561ms |
| volumes?q=Test | 200 | xhr | iron-request.html:304 | 6.0KB | 529ms |

Jeder Tastendruck löst einen Request aus. (Abb. 4)

Damit wir die Antwort auf unseren Request später weiterverarbeiten können, müssen wir diese Antwort speichern. Dazu geben wir an, in welchem Format wir die Antwort erwarten und dass wir den Inhalt der letzten (aktuellsten) Antwort in einer Variablen speichern möchten. Dazu ergänzen wir im <iron-ajax>-Tag folgende zwei Attribute:

Hier habe ich ein " vergessen, mein Fehler…

```
handle-as="json"
last-response="{{searchresult}}"
```

Mittels **handle-as** teilen wir mit, dass wir eine Antwort im JSON-Format erwarten und mittels **last-response** und Databinding lassen wir uns die letzte Antwort in die Variable **searchresult** schreiben. So einfach können wir einen AJAX-Request durchführen!

Kein Bindestrich an dieser Stelle!
Keine Silbentrennung im Quellcode!

## Verschachtelte Templates

Die Suchergebnisse warten jetzt darauf, angezeigt zu werden. Hierzu kombinieren wir das Databinding von Polymer mit einem weiteren Template, dass die Daten eines Buches anzeigt und so oft wiederholt eingebunden wird, wie es Bücher im Suchergebnis gibt. Innerhalb des Templates unserer Komponente fügen wir dazu folgendes neues Template hinzu:

```
<template is="dom-repeat" items="{{searchresult.items}}">
  <p>{{item.volumeInfo.title}}</p>
</template>
```

Das Attribut is mit dem Wert **dom-repeat** sorgt dafür, dass das Template selbst so oft eingebunden wird, wie es Einträge in dem Array gibt, das dem Attribut **items** zugewiesen wurde. Der jeweilige Eintrag aus dem Array steht automatisch in der Variable **item** zur Verfügung. Zum Test geben wir innerhalb des Templates den Titel des Buches aus. Wenn wir nun einen Test im Browser durchführen, sollte es wie in diesem Screenshot aussehen:


**Unsere Suchergebnisse werden angezeigt. (Abb. 5)**

## Interoperabilität

Innerhalb unserer eigenen Komponente könnten wir nun in der inneren Schleife die Ausgabe der Suchergebnisse vervollständigen und wären fertig. Doch das wäre zu einfach – und um die Interoperabilität zwischen eigenen Komponenten zu demonstrieren, brauchen wir mindestens zwei eigene Komponenten. Daher lagern wir die Anzeige eines Suchergebnisses in eine eigene Komponente aus. Daraus ergibt sich die interessante Frage, wie nun die Daten eines Buches von der einen Komponente in die andere gelangen, um dort angezeigt zu werden. Doch dazu später mehr, wir beginnen ein paar Schritte vorher und legen erst eine neue Komponente an.

Die neue Komponente soll **bookworm-detail** heißen. Entsprechend legen wir im Verzeichnis **bookworm_components** das Unterverzeichnis **bookworm-detail** an und darin eine neue Datei mit dem Namen bookworm-detail.html. Das Grundgerüst unserer neuen Komponente entspricht dem Grundgerüst unserer ersten Komponente:

```
<link rel="import" href="../../bower_components/polymer/
polymer.html“ />
<dom-module id="bookworm-detail">
  <template>
  </template>
  <script>
    Polymer({
      is: "bookworm-detail"
    });
  </script>
</dom-module>
```

Alle Komponenten haben in etwa das gleiche Grundgerüst. Unserer zweiten Komponente, die die Details eines Buches anzeigen soll, müssen wir die Daten eines Buches übergeben können. Dazu erweitern wir den JavaScript-Code am Ende der Komponente um ein Properties-Objekt:

```
<script>
  Polymer({
    is: "bookworm-detail",
    properties: {
      bookdata: {
        type: Object
      }
    }
  });
</script>
```

Innerhalb des Properties-Objekts definieren wir ein Objekt namens **bookdata** vom JavaScript-Typ **Object.** Die Properties-Objekte werden über die Attribute des Tags unserer Komponente gesetzt, so können wir die Daten eines Buches von einer Komponente an die nächste übergeben. Bevor wir unsere neue Komponente einbauen, ergänzen wir die Ausgabe des Buchtitels im Template:

```
<template>
  <p>{{bookdata.volumeInfo.title}}</p>
</template>
```

Wir können auf unser definiertes Property zugreifen wie auf eine Variable. Bauen wir nun unsere zweite Komponente in die erste Komponente ein. Dazu müssen wir in der Datei **bookworm-search.html** einen Import hinzufügen:

```
<link rel="import" href="../bookworm-detail/bookworm-detail.
html" />
```

Kein Bindestrich an dieser Stelle!
Keine Silbentrennung im Quellcode!

Dabei müssen wir beachten, dass wir, um zur Komponente **bookworm-detail** zu gelangen, nur eine Verzeichnisebene nach oben wechseln müssen. Ersetzen wir nun die direkte Ausgabe des Buchtitels durch den Einbau unserer neuen Komponente und übergeben dabei gleich die Daten des Buches:

```
<bookworm-detail bookdata="{{item}}"></bookworm-detail>
```

Wenn wir uns das Ganze nun erneut im Browser anschauen, sollte es immer noch ausschauen wie im vorherigen Screenshot, jedoch haben wir technisch gesehen eine schönere Kapselung erreicht.

## Material Design

Wenn wir genau hinschauen, entspricht unsere Applikation noch nicht den Regeln für Material-Design. Wir können nun mit sehr wenig Aufwand dafür sorgen, dass unsere App einen schönen Rahmen erhält und überall die gleiche Schriftart verwendet wird. Dazu ergänzen wir in unserer Such-Komponente folgende zwei Imports:

```
<link rel="import" href="../../bower_components/paper-material/
paper-material.html" />
<link rel="import" href="../../bower_components/paper-styles/
typography.html" />
```

Nun umschließen wir den kompletten Inhalt des Template mit der soeben importierten **paper-material**-Komponente:

```
<template>
  <paper-material>
    …
  </paper-material>
</template>
```

Außerdem kann die Buchsuche auch optisch angepasst werden. Eigene Styles müssen Bestandteil dabei des Template sein, d.h. innerhalb des Templates definiert werden:

```
<template>
  <style>
    * {
      @apply(--paper-font-common-base);
    }
    paper-material {
      padding: 1em;
    }
  </style>
  <paper-material>
    …
  </paper-material>
</template>
```

Wir übernehmen für alle HTML-Elemente die von Polymer für uns vordefinierte Schrift und setzen für die **paper-material**-Komponente einen Abstand zum enthaltenen Inhalt. Nun verwendet unsere App überall die gleiche Schriftart und hat einen schönen Rahmen.

## Responsive Design

Bisher zeigen wir das Suchergebnis untereinander an. Es funktioniert, ist jedoch nicht wirklich responsiv, da der zur Verfügung stehende Platz nicht optimal genutzt wird. Daher werden wir nun die Komponente für die Detailansicht erweitern. Dazu importieren wir die Komponente **paper-card**

```
<link rel="import" href="../../bower_components/paper-card/
paper-card.html“ />
```

und ersetzen im Template die Ausgabe des Titels als Absatz durch die **paper-card**, wobei wir gleich ein paar weitere Angaben ausgeben:

```
<template>
  <paper-card heading="{{bookdata.volumeInfo.title}}" image="{{-
bookdata.volumeInfo.imageLinks.thumbnail}}">
    <div id="subtitle">{{bookdata.volumeInfo.subtitle}}</div>
     <div id="details">{{bookdata.volumeInfo.pageCount}} Pages
({{bookdata.volumeInfo.publishedDate}})</div>
  </paper-card>
</template>
```

Kein Bindestrich an dieser Stelle!
Keine Silbentrennung im Quellcode!

Bei der **paper-card** definieren wir zwei Attribute, eines zur Ausgabe des Buchtitels und eines zur Anzeige des Buchcovers. Innerhalb der **paper-card** geben wir in einzelnen **div**-Elementen neu auch den Sub-Titel, die Seitenzahl und das Erscheinungsdatum aus.
Wenn wir das nun im Browser betrachten, sehen wir die entsprechenden Angaben und auch das Buchcover wird angezeigt. Leider alles andere als schön. Daher fügen wir am Anfang des Template noch folgendes Stylesheet hinzu:

```
<style>
  paper-card {
    margin: 1em;
    padding: 0.5em;
    width: 20em;
  }
  #subtitle {
    font-size: 1.2em;
    font-weight: bold;
  }
  #details {
    font-style: italic;
  }
</style>
```

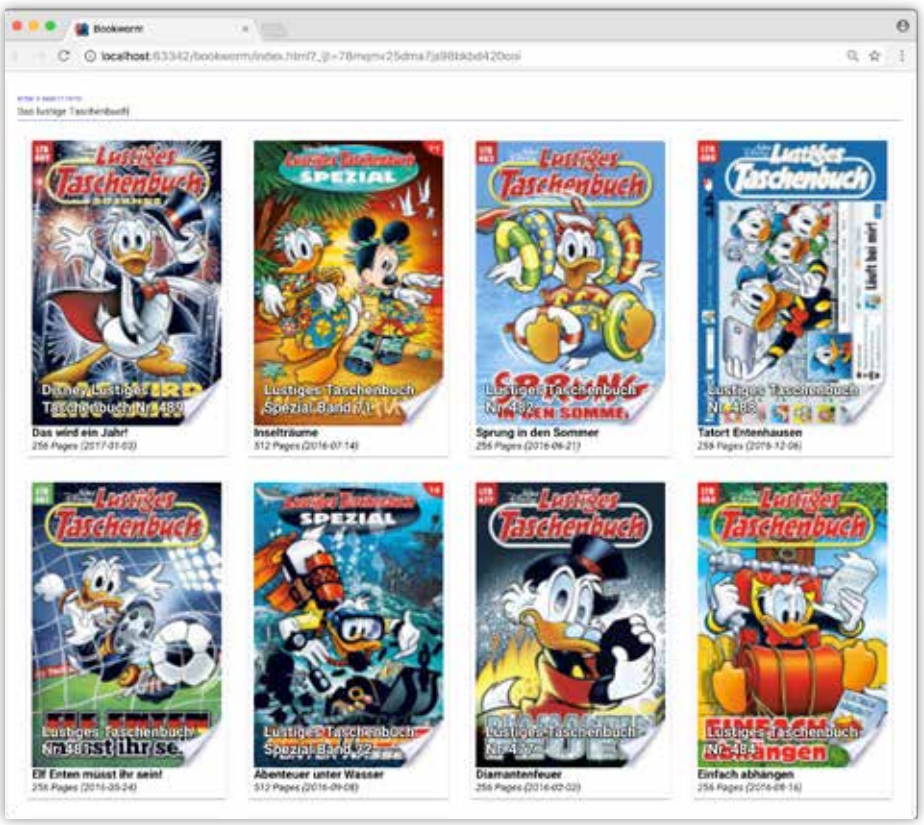
Wir definieren damit für die **paper-card** einige Abstände und beschränken deren Breite. Auch der Sub-Titel und die Anzeige von Seitenzahl und Erscheinungsdatum sind nun schon etwas gefälliger. Doch wir haben noch etwas erreicht: Durch die Beschränkung der Breite der **paper-card** werden diese nun nebeneinander angezeigt, je nach verfügbarem Platz. Unsere kleine Applikation reagiert jetzt bereits responsiv.

## Fremdkomponenten stylen

Der Buchtitel wird von der **paper-card** über dem Buchcover angezeigt. Das wäre nicht weiter schlimm, wenn der Titel jetzt nicht so schlecht zu lesen wäre! Je nach Buchcover ist der Titel gar nicht zu entziffern. Das müssen wir ändern. Doch wie?

Jede Komponente hat ihr eigenes kleines DOM, Shadow DOM genannt. Wir können HTML, CSS und JavaScript schreiben, ohne Gefahr zu laufen, dass dies unerwünschte Nebeneffekte auf andere Komponenten hat. Allerdings haben wir nun ein Problem: Wir können in unserer Komponente so viel CSS definieren, wie mir möchten, die **paper-card** sieht es schlicht nicht, da sie ihr eigenes DOM hat!

CSS unterstützt Variablen und sogenannte Mixins. Wenn der Autor einer Komponente vorgesehen hat, von extern Styles verändern zu dürfen, so stellt er dafür entsprechende Variablen bereit. Welche das sind und was man damit jeweils stylen kann, muss der Autor der Komponente dokumentieren. Google hat das glücklicherweise für alle Komponenten getan, auch für die **paper-card**. Gemäß Dokumentation müssen wir zum Stylen des Textes über dem Bild die Variable **--paper-card-header-image-text** setzen. Diese Variable ist ein Mixin, d.h. sie enthält nicht nur einen Wert, sondern eine komplette CSS-Definition (Name-Wert-Paare). Ergänzen wir nun das CSS für die **paper-card** Komponente wie folgt:




Unsere fertige Anwendung. (Abb. 6)

```
paper-card {
  margin: 1em;
  padding: 0.5em;
  width: 20em;
  --paper-card-header-image-text : {
    color: white;
    font-weight: bold;
    text-shadow:  1px  1px 1px black,
                  1px -1px 1px black,
                 -1px  1px 1px black,
                 -1px -1px 1px black;
  }
}
```

Wir setzen damit die Schriftfarbe auf weiß, machen den Text fett und fügen noch einen Schatten hinzu, so dass die Schrift schwarz umrandet wird. So ist sie nun auch über dem Buchcover sehr gut zu lesen – und wir haben über ein CSS-Mixin in das Styling einer anderen Komponente eingegriffen.

## Fazit

Für unsere erste Applikation auf Basis von Web-Components und Polymer haben wir einiges geleistet. Wir mussten zwei eigene Komponenten entwickeln und griffen auf Fremdkomponenten zurück. Mittels eines Polyfills sorgten wir dafür, dass alle aktuellen Browser mit unserer App funktionieren, auch wenn sie noch keine vollständige Web-Components Unterstützung implementiert haben.

Wir haben mehrere Komponenten über Databinding interagieren lassen und so auch gelernt, wie Daten von einer Komponente an eine andere weitergegeben werden können. Den AJAX-Request zur Suche nach Büchern konnten wir komplett ohne JavaScript realisieren. Schließlich haben wir noch auf das Styling einer Komponente Einfluss genommen und es unseren Wünschen angepasst.

Mittels Web-Components und Polymer entwickelten wir wiederverwendbare Komponenten, die in sich gekapselt sind und die sich kombinieren und erweitern lassen. Dabei dachten wir immer „in Komponenten" und schrieben auf effiziente Weise übersichtlichen, leicht verständlichen Quelltext.

Web-Components – die neue, standardisierte Art, Anwendungen für das Web zu entwickeln!

**Links:**

Der Quelltext der Beispielanwendung dieses Tutorials steht auf GitHub unter der AGPL zur Verfügung: https://bit.ly/wcpoltut