



Clientseitige Generierung von PDF-Dokumenten aus
Webinhalten mit HTML-Markup und CSS-Layout

Masterarbeit

zur Erlangung des akademischen Grades Master of Science (M.Sc.)

vorgelegt an der Technischen Hochschule Köln (Campus Gummersbach)

im Studiengang
Medieninformatik
mit dem Schwerpunkt
Weaving the Web

ausgearbeitet von:

BENEDIKT ENGEL

Matrikelnummer: 11122021

Prüfer: Prof. Christian Noss (TH Köln)
Prof. Dr. Hoai Viet Nguyen (TH Köln)

Gummersbach, 11.06.2024

Erklärung

Erklärung

Ich versichere, die von mir vorgelegte Arbeit selbstständig verfasst zu haben. Alle Stellen, die wörtlich oder sinngemäß aus veröffentlichten oder nicht veröffentlichten Arbeiten anderer oder der Verfasserin/des Verfassers selbst 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 beziehungsweise in wesentlichen Teilen noch keiner anderen Prüfungsbehörde vorgelegen.

Kehl, 11.06.24

Ort, Datum

B. Engell

Unterschrift

Abstract

This thesis examines the client-side generation of PDF documents from HTML markup and CSS layout. The approach aims to simplify the maintenance process of web content and PDF documents by specifying information in the web content and generating the PDF document from this information as required.

Firstly, the technologies and options for client-side generation available at the time of this work are considered and examined for their advantages and disadvantages. The results show that all options have disadvantages that must be taken into account when using them. On the other hand, it was examined how a library can be developed that minimizes the disadvantages found. The partial question of how the content and layout of HTML elements can be mapped in PDF documents is answered. It is shown how texts can be read from web content line by line and transferred to a PDF document using the CSS rules applied. Furthermore, the possibilities of transferring backgrounds and borders of elements are discussed.

In addition, it is shown how document outlines can be generated from the web content, which represent a table of contents of the document and how a possibility for headers and footers, which are usually used in documents, can be implemented without being displayed in the web content.

Kurzfassung

Diese Arbeit untersucht die clientseitige Generierung von PDF-Dokumenten aus HTML-Markup und CSS-Layout. Der Ansatz zielt darauf ab, den Pflegeprozess von Webinhalten und PDF-Dokumenten zu vereinfachen, indem Informationen im Webinhalt angegeben werden und aus diesem bei Bedarf das PDF-Dokument generiert werden kann.

Zum einen werden die zum Zeitpunkt dieser Arbeit vorhandenen Technologien und Möglichkeiten zur clientseitigen Generierung betrachtet und auf ihre Vor- und Nachteile untersucht. Dabei zeigt sich, dass für alle Möglichkeiten Nachteile bestehen, welche bei einer Nutzung beachtet werden müssen. Zum anderen wurde untersucht wie sich eine Bibliothek entwickeln lässt, welche die ermittelten Nachteile minimiert. Es wird dabei die Teilfrage beantwortet, wie sich der Inhalt und das Layout von HTML-Elementen in PDF-Dokumenten abbilden lassen. Es wird gezeigt, wie sich Texte aus Webinhalten zeilenweise auslesen und mit den angewendeten CSS-Regeln in ein PDF-Dokument übertragen werden können. Des weiteren wird auf Möglichkeiten des Transfers von Hintergründen und Umrandungen von Elementen eingegangen. Zudem wird gezeigt wie sich aus dem Webinhalt die Document Outlines generieren lassen, welche ein Inhaltsverzeichnis des Dokuments darstellen und wie eine Möglichkeit für Kopf- und Fußzeilen, welche in der Regel in Dokumenten genutzt werden, umgesetzt werden können ohne das diese in dem Webinhalt dargestellt werden.

Anmerkung

In dieser Arbeit wird in den meisten Fällen eine geschlechtsneutrale Formulierung verwendet. An einigen Stellen wurde zur besseren Lesbarkeit das generische Maskulinum verwendet. In diesem Fall bezieht sich dieses auf alle Geschlechter.

Begleitrepository

Die aus dieser Arbeit resultierende Bibliothek zur clientseitigen Generierung von PDF-Dokumenten ist in einem Begleitrepository auf GitHub abgelegt:

<https://github.com/BenediktEngel/htmlWebsite2pdf>

Inhaltsverzeichnis

Erklärung	I
Abstract	II
1. Einleitung	1
1.1. Zielsetzung	1
1.2. Aufbau	2
2. Grundlagen des Portable Document Formats	4
2.1. Versionen des Portable Document Formats	4
2.2. Varianten des Portable Document Formats	5
2.3. Aufbau von PDF-Dateien	6
2.3.1. Objekte	7
2.3.2. Direkte und Indirekte Objekte	10
2.3.3. Grundstruktur	11
2.3.4. Inkrementelle Updates	15
2.3.5. Boxmodel und Boxes	15
2.4. Spezifische PDF-Objekte	17
2.4.1. Document Catalog	18
2.4.2. Pages und Page-Tree	19
2.4.3. Resource Dictionary	22
2.4.4. Schriften	22
2.4.5. Content Stream	23
2.4.6. Document Outline	27
2.4.7. Annotations	30
3. Technologien und Möglichkeiten zur clientseitigen Generierung	33
3.1. Druck-Dialog im Browser	34
3.2. jsPDF	38
3.3. html2pdf.js	41
3.4. Vergleich und Fazit	44
4. Konzeption der Bibliothek	49
4.1. Zielsetzung	49

4.2. Anforderungsermittlung	50
4.2.1. Funktionale Anforderungen	50
4.2.2. Nichtfunktionale Anforderungen	53
4.3. Architekturentwurf	54
4.3.1. Programmiersprache der Bibliothek	54
4.3.2. PDF-Writer	55
4.3.3. Bibliothek zur Verarbeitung von Schrift-Dateien (Font-Engine) . . .	55
4.3.4. Bibliothek zur Kompression (Compression-Engine)	56
5. Umsetzung und Entwicklung der Bibliothek	58
5.1. PDF-Datenmodell und PDF-Writer	58
5.1.1. Klassen der PDF-Objekte	58
5.1.2. Klasse PdfDocument	60
5.1.3. Klasse CrossReferenceSection und CrossReferenceSubSection .	65
5.1.4. Weitere Klassen	65
5.2. Transfer der HTML-Elemente und zugehörigem CSS-Layout	66
5.3. Transfer der Schriften aus dem Webinhalt in das PDF-Dokument	69
5.4. Document Outlines anhand der Überschriften-Hierarchie	70
5.5. Hinzufügen von Kopf- und Fußzeilen	72
5.6. Komplikationen bei der Umsetzung	73
5.6.1. Auslesen der Texte	74
5.6.2. Bilder verschiedener Formate	78
5.6.3. Interaktion während der Generierung	79
5.6.4. Schriften aus dem Webinhalt	79
5.6.5. Transfer des CSS-Layouts von Elementen	81
5.7. Verwendung der Bibliothek	82
6. Evaluation	84
7. Fazit und Ausblick	87
Abbildungsverzeichnis	89
Tabellenverzeichnis	90
Quellcodeverzeichnis	92

Abkürzungsverzeichnis	93
Literaturverzeichnis	95
Anhang	99
A. Portable Document Format	99
A.I. Hello World PDF-Datei	99
A.II. Standard Schriftarten in PDF-Dokumenten	102
B. Quellcode der zur Generierung verwendeten Webseite	103
C. Generierte PDF-Dokumente zu Kapitel 3	107
C.I. Druck-Dialog im Browser	107
C.II. jsPDF	115
C.III.html2pdf.js	126
D. Konzeption	133
D.I. Funktionale Anforderungen	133
D.II. Nichtfunktionale Anforderungen	139
D.III. Architecture Decision Records (ADRs)	140
E. Implementierung	150
E.I. Quellcode der Bibliothek	150
E.II. Optionen-Objekt für die Generierung	159
F. Generierte PDF-Dokumente zu Kapitel 5	164

1. Einleitung

Das Portable Document Format (PDF) hat sich in den letzten Jahrzehnten zum de facto-Standard für die Speicherung und Verteilung von Dokumenten entwickelt. Mittlerweile existieren laut Adobe mehr als 3 Billionen PDF-Dokumente und im Jahr 2022 wurden allein mit dem PDF-Betrachtungsprogramm *Adobe Acrobat* nach eigenen Angaben mehr als 400 Milliarden PDF-Dokumente geöffnet [1].

Informationen werden jedoch nicht nur als Dokumente verfügbar gemacht, sondern sind auch über Webseiten im Internet schnell abrufbar. Sowohl Informationen zu Produkten, als auch Jobanzeigen, journalistische Artikel und vieles Weiteres stehen zur Verfügung. Diese Information können jedoch jederzeit durch den Webseitenbetreibenden verschoben, verändert und gelöscht werden, wodurch sie entweder nicht mehr oder verändert auffindbar sind.

Zudem stehen die Informationen beispielsweise bei Datenblättern von Produkten oft auch als Download in Form eines PDF-Dokuments bereit. Von diesen Downloads profitieren Nutzende, da die Daten dadurch unveränderbar, lokal vorliegen. Die Generierung dieser Dokumente und das Nachhalten der gleichen Informationen im Webinhalt stellt jedoch einen doppelten Arbeitsaufwand für den Webseitenbetreibenden dar.

Die clientseitige Generierung von PDF-Dokumenten aus dem vorliegen Webinhalt bietet hierfür Lösungen. Zum einen wird der Prozess der Datenpflege vereinfacht, da die Informationen nur an einer Stelle für den Webinhalt gepflegt werden müssen. Hierdurch stehen die Daten in der Regel im Design des Webseitenbetreibenden als Website zur Verfügung stehen. Zum anderen werden die entsprechenden Dokumente nur bei Bedarf durch den Nutzenden generiert, was ein weiteres Einsparungspotenzial für die Betreibenden von Webhalten bietet.

1.1. Zielsetzung

Ziel dieser Arbeit ist es, die clientseitigen Generierung von PDF-Dokumenten aus Hypertext Markup Language (HTML)-Markup und durch Cascading Style Sheet (CSS) definiertes Layout zu betrachten. Dabei werden zunächst die zum Zeitpunkt dieser Arbeit existierenden Technologien und Möglichkeiten auf ihr Anwendbarkeit und darauf die generierten PDF-Dokumente auf Vor- und Nachteile überprüft und verglichen. Darauf auf-

bauend wird anhand der festgestellten Vor- und Nachteile eine neue Bibliothek konzipiert, welche die Nachteile der bisher existierenden Möglichkeiten reduzieren und die Vorteile vereinen soll.

Daraus ergibt sich die übergeordnete Forschungsfrage „Wie lassen sich clientseitig PDF-Dokumente aus HTML-Markup und CSS-Layout generieren?“. Für die Beantwortung dieser Forschungsfrage ergeben sich die nachfolgenden Teilfragen:

1. Welche Technologien und Möglichkeiten stehen zum Zeitpunkt dieser Arbeit zur Generierung von PDF-Dokumenten zur Verfügung und welche Vor- und Nachteile bieten diese?
2. Wie lassen sich mögliche Nachteile in einer neuen Bibliothek minimieren?
3. Wie lassen sich Inhalt und Layout von HTML-Elementen in PDF-Dokumenten abbilden?

1.2. Aufbau

Die vorliegende Arbeit behandelt die bereits genannten Forschungsfragen in zwei Teilen. Der erste Teil befasst sich mit den theoretischen Grundlagen, die für das Verständnis des darauffolgenden praktischen Teils notwendig sind. Dazu wird in Kapitel 2 das Portable Document Format vorgestellt. Neben einem Überblick über die Versionsgeschichte und Varianten des Formats werden die in PDF-Dateien verwendeten Objekt-Typen, sowie der zugrunde liegende interne Aufbau vorgestellt. Des Weiteren werden in Kapitel 2.4 einige spezifische Objekte vorgestellt, welche in PDF-Dateien Anwendung finden und in späteren Kapiteln für die Umsetzung benötigt werden.

In Kapitel 3 werden die zum Zeitpunkt dieser Arbeit verfügbaren Möglichkeiten und Technologien der clientseitigen PDF-Generierung aus HTML-Markup unter Verwendung des CSS-Layouts vorgestellt und ein Vergleich dieser durchgeführt. Anhand der durchgeföhrten Analyse und der abgeleiteten Vor- und Nachteile wird in Kapitel 4 eine Konzeption für eine neue Bibliothek definiert, welche die Nachteile der clientseitige Generierung von PDF-Dokumenten minimieren soll. Die Implementierung der dargelegten Konzeption wird daraufhin in Kapitel 5 vorgestellt und dokumentiert. Es wird sowohl auf den Aufbau der Funktionalität des PDF-Writers, als auch auf den Ablauf des Transfers der Webinhale eingegangen. Des Weiteren werden in Kapitel 5.6 Komplikationen, die während der Umsetzung aufgetreten sind, und deren Lösungsansätze vorgestellt. Anschließend wird

1. Einleitung

in Kapitel 6 die entwickelte Bibliothek evaluiert und die Zielerreichung der aufgestellten Anforderungen überprüft.

Abschließend wird im letzten Kapitel 7 sowohl ein Fazit gezogen, als auch ein Ausblick für die zukünftige wissenschaftliche Bearbeitung der Thematik gegeben.

2. Grundlagen des Portable Document Formats

In diesem Kapitel werden das PDF-Dateiformat und dessen Grundlagen vorgestellt. Kapitel 2.1 gibt einen Überblick über die Geschichte und die Versionen des Portable Document Formats. Des Weiteren werden in Kapitel 2.2 unterschiedliche Varianten von PDF-Dokumenten für verschiedene Anwendungsfälle vorgestellt. Darauf folgend stellt das Kapitel 2.3 den internen Aufbau von PDF-Dokumenten vor. Neben der Betrachtung der allgemeinen Grundstruktur werden die verschiedenen Datenobjekte und das Boxmodell dargestellt. Abschließend werden in Kapitel 2.4 einige spezifische Objekte betrachtet, welche beispielsweise zur Definition von Seiten in einem PDF-Dokument Verwendung finden.

2.1. Versionen des Portable Document Formats

Die Geschichte des Portable Document Formats begann im Jahr 1991, mit der Entwicklung von *Project Camelot* durch Adobe-Mitgründer Dr. John Warnock [2]. Ziel war eine Platform-neutrale Methode für den Austausch von Dokumenten, welche sowohl die Betrachtung von Dokumenten auf dem Computer ermöglichte, als auch das Drucken dieser Dokumente [3]. Mit PostScript existierte bereits eine Definition für Dokumente, welche in vielen Druckern und Textverarbeitungsprogrammen implementiert war. Diese hatte jedoch zum Nachteil, dass für die Darstellung der Dokumente leistungsstarke Computer benötigt wurden. Laut Warnock wären PostScript und Display PostScript die richtige Lösung für die Zukunft, da Computer immer leistungsstärker werden würden, dies aber nicht die Probleme der Kunden zur damaligen Zeit löste [4].

Bereits 1993 wurde das Ergebnis des Projekts als PDF-Format und die zugehörigen Programme *Acrobat Destiller*, zum Erstellen und Bearbeiten, sowie *Acrobat Reader*, zum Ansehen von PDF-Dokumenten veröffentlicht [3]. Zu Beginn waren die Programme kostenpflichtig, erst 1994 wurde der Acrobat Reader kostenlos zur Verfügung gestellt, was die Verbreitung beschleunigte [5]. In den folgenden Jahren wurde das PDF-Format kontinuierlich durch Adobe weiterentwickelt und Funktionalitäten wie der Passwortschutz von Dokumenten und Formulare hinzugefügt (siehe Versionen in Tabelle 1).

2007 übergab Adobe die PDF-Spezifikation an die International Organization for Standardization (ISO), welche die Version 1.7 als internationalen Standard verfügbar machte [6]. Im Jahr 2017 veröffentlichte die ISO die Version 2.0, welche seit 2020 in ihrer aktuellen Fassung als ISO 32000-2:2020 vorliegt [7].

Version	Veröffentlichung	Neuerungen (Auszug)
Adobe PDF 1.0	1993	Erste Veröffentlichung
Adobe PDF 1.1	März 1996	Passwortschutz; Actions; Geräteunabhängige Farben; Neues Datumsformat (ASN.1) [8].
Adobe PDF 1.2	November 1996	Interaktive Elemente und Formulare; Support für Text in Chinesisch, Japanisch und Korea-nisch; Multimedia-Support (Video und Ton) [9].
Adobe PDF 1.3	2000	Name und Number Trees, Einbettung von Da-teien, Support von JavaScript (JS), Digitale Si-gnaturen [10].
Adobe PDF 1.4	2001	Viewer Preferences; Metadata Streams; Tagged PDF [11].
Adobe PDF 1.5	2003	Bilder mit JPEG2000-Komprimierung; Cross-Reference Streams; Object Streams [12].
Adobe PDF 1.6	2004	Integration von 3D-Grafiken [13].
Adobe PDF 1.7	2006	
PDF 1.7 (ISO 32000-1:2008)	2008	Erste Veröffentlichung als ISO-Standard
PDF 2.0 (ISO 32000-2:2017)	2017	256-bit AES Verschlüsselung; Überarbeitung von Tagged PDF; Anpassungen auf die aktuel-len technischen Möglichkeiten [14]
PDF 2.0 (ISO 32000-2:2020)	2020	Diverse Aktualisierungen von Anhängen; Aktua-lisierung des PDF-Zeichensatz [7]

Tabelle 1: Versionen des Portable Document Formats

2.2. Varianten des Portable Document Formats

Neben dem übergeordneten PDF-Standard, welcher den generellen Aufbau von PDF-Dokumenten beschreibt, existieren weitere PDF-Varianten. Diese sind wie das PDF-Format selbst, teilweise auch durch die ISO standardisiert. In der Regel sind die Varianten an bestimmte Verwendungszwecke gekoppelt. Im nun Folgenden werden einige kurz vorgestellt:

PDF/X *PDF for Exchange* (zu Deutsch: PDF für den Austausch) ist die erste Variante des PDF-Formats, welche durch die ISO im Jahr 2001 standardisiert wurde. Ziel der Variante war es ein Format für den Austausch von Dokumenten in der Druck-Industrie zu schaffen. Aus diesem Grund schreibt der PDF/X-Standard unter anderem vor, dass

sowohl verwendete Schriftarten als auch alle Bilder und Grafiken in das PDF-Dokument eingebunden sind und nicht extern referenziert werden. Des Weiteren dürfen PDF/X-Dokumente keinen Passwortschutz, keine Formulare und keine multimedialen Inhalte besitzen, da diese im Druck nicht abgebildet werden können [15].

PDF/A Bei *PDF for Archiving* (zu Deutsch: PDF für die Archivierung) handelt es sich um einen ISO-Standard, welcher vorgibt wie PDF-Dokumente aufgebaut sein sollen, wenn sie zur Langzeitarchivierung verwendet werden. Der erste Normteil wurde am 1. Oktober 2005 veröffentlicht und wurde seitdem durch weitere Normteile kontinuierlich erweitert. Dabei legt der Standard unter anderem fest, dass in einem Dokument, welches dem PDF/A-Standard gerecht werden soll, alle benötigten Inhalte wie Bilder und Schriftarten eingebettet sind und somit immer korrekt dargestellt werden können. Außerdem dürfen dem PDF/A-Standard entsprechende Dokumente keinen Passwortschutz besitzen [16][17].

PDF/UA steht für *PDF for Universal Accessibility* (zu Deutsch: PDF für den universellen Zugang) und ist seit 2012 als ISO-Standard ISO 14289-1 standardisiert. Dieser schreibt vor, wie PDF-Dokumente aufgebaut werden sollen, dass sie barrierefrei nutzbar sind. Zudem wird die Verwendung der *Tagged PDF*-Funktionalität, welche in Version 1.4 eingeführt wurde, vorgeschrieben. Hierfür wird angegeben, wie diese angewendet werden muss, um dem Standard zu entsprechen [18].

Neben den vorgestellten Varianten existieren noch einige weitere. Beispielsweise sind mit PDF/H (*PDF for Health Care Industry*) für das Gesundheitswesen und PDF/E (*PDF for Engineering*) für das Ingenieurwesen, Spezifikationen für spezielle Wirtschaftsbeziehungsweise Industriezweige vorhanden. Außerdem liegt mit PDF/R (*PDF for raster images*) ein Standard für Dokumente vor, welche nur aus Bildern bestehen, wie beispielsweise gescannte Dokumente [19][6].

2.3. Aufbau von PDF-Dateien

Das vorliegende Unterkapitel befasst sich mit dem internen Aufbau von PDF-Dateien. Dieser wird sichtbar, wenn eine PDF-Datei mit einem Texteditor, wie beispielsweise *TextEdit* unter MacOS oder *Notepad* unter Windows, anstelle eines PDF-Viewers geöffnet

wird. Unter einem *PDF-Viewer* versteht sich ein Programm zur Darstellung von PDF-Dokumenten. Im Weiteren wird das Schlüsselwort *PDF-Datei* verwendet, wenn sich auf den internen Inhalt eines PDF-Dokuments bezogen wird. Analog dazu wird das Schlüsselwort *PDF-Dokument* angewendet, wenn auf die Ausgabe, das eigentliche Dokument, eingegangen wird.

2.3.1. Objekte

Alle Daten, welche für ein PDF-Dokument benötigt werden, liegen in der PDF-Datei als Objekte vor. Je nach Datentyp des Werts und dessen Verwendung wird eines der folgenden neun Objekte verwendet:

Boolean Object Ein *Boolean Object* stellt logische Werte dar und kann genau zwei Werte annehmen: `true` oder `false` [20].

Numeric Object Für numerische Werte beschreibt die PDF-Spezifikation zwei Arten von *Numeric Objects*. Zum einen das *Integer Object*, welches einen ganzzahligen Wert besitzt und zum anderen das *Real Object* für Gleitkommazahlen. Bei beiden Objekten sind sowohl positive als auch negative Werte möglich, welche über das Vorzeichen unterschieden werden. Handelt es sich um eine positive Zahl kann das Vorzeichen weggelassen werden [20]. Quellcode 1 stellt in Zeile 2 einige beispielhafte Integer Objects dar, analog dazu handelt es sich bei den Werten in Zeile 4 um Real Objects.

```

1 % Integer Objects:
2 1      -50     190     +1426
3 % Real Objects:
4 20.3   +0.2    -1.2    1.
```

Quellcode 1: Beispiel Werte von Integer und Real Objects.

String Object Für Zeichenketten wird das *String Object* verwendet. Der Wert kann hierbei entweder als *Literal String* (im Deutschen: wörtliche Zeichenkette) oder als *Hexadezimal String* angegeben werden. Wenn eine Zeichenkette als literal String angegeben wird, so muss diese zur Kennzeichnung von runden Klammern umschlossen sein. Analog dazu, muss ein String in der Hexadezimal-Schreibweise, von spitzen Klammern umschlossen werden. Sofern der Wert eines literal Strings runde Klammern enthält müssen diesen das Maskierungszeichen *Backslash* (\) vorangestellt werden. Durch die Verwendung des

Maskierungszeichens ist, wie in Zeile 2 von Quellcode 2 dargestellt, auch ein Text mit Klammern möglich. Soll ein Datum in einem String Object gespeichert werden, muss die Abstract Syntax Notation One (ASN.1) verwendet werden. Das String Object, welches das Datum enthält, wird daraufhin als wörtliche Zeichenkette in die PDF-Datei gesetzt [20].

```

1 (Text)
2 (Ein Text mit \((Klammern\))
3 <45696E204865782D54657874>
4 (D:20240612235959-00'00')

```

Quellcode 2: Beispielhafte String Objects.

Name Object Das *Name Object* ist laut der PDF-Spezifikation ein „atomic symbol uniquely defined by a sequence of any characters“ [20]. Das bedeutet, sollten mehrere Name Objects aus den gleichen Zeichen bestehen, sie das selbe Objekt darstellen. Das Name Object wird immer von einem Schrägstrich eingeleitet, welcher jedoch nicht Teil des Namens ist [20]. In der Regel werden Name Objects als Schlüssel in Dictionary Objects verwendet und sind dabei durch die PDF-Spezifikation vorgegeben. Nur in wenigen Fällen müssen eigene Name Objects erstellt werden. Das ist beispielsweise bei zusätzlichen Metadaten der Fall [6].

Array Object Das *Array Object* stellt eine Liste von Objekten dar. Ein Array Object kann sowohl nur gleiche Objekt-Typen als auch mehrere verschiedene Objekt-Typen beinhalten. In der PDF-Datei wird das Array Object immer von eckigen Klammern umschlossen und die einzelnen Werte durch ein Leerzeichen getrennt [20].

```

1 [ ] % Leeres Array
2 [ (eins) 2 (3) 4. ] % Array Object mit String und Numeric Objects
3 [ [ 1 2 ] [ 3 4 ] ] % Array Object aus Array Objects mit Integer Objects

```

Quellcode 3: Verschiedene Array Objects.

Dictionary Object Das *Dictionary Object* stellt eine Zuordnungstabelle dar. Es enthält ausschließlich Key-Value-Paare, wobei der Key immer vom Typ Name Object ist. Der zugehörige Wert kann von jedem anderen Typen sein, ist aber in den meisten Fällen von der PDF-Spezifikation vorgegeben. Eingeleitet wird ein Dictionary Object immer von zwei

öffnenden spitzen Klammer (<<). Analog dazu wird es durch die Verwendung von zwei schließenden spitzen Klammer (>>) beendet [20].

Ein Beispiel für ein Dictionary Object ist in Quellcode 4 zu sehen. Dieses Dictionary Object stellt die Definition einer Seite im PDF-Dokument dar¹. Das Dictionary Object besteht aus drei Key-Value-Paaren. In Zeile 2 wird dem Schlüssel /Type das Name Object /Page zugeordnet. Der Schlüssel /Parent besitzt als Wert eine Referenz² zu dem Objekt mit der ID 2. Des weiteren wird dem Schlüssel /MediaBox in Zeile 4 ein Array Object, bestehend aus vier Integer Objects, zugeordnet.

```

1 <<
2 /Type /Page
3 /Parent 2 0 R
4 /MediaBox [ 0 0 595 842 ]
5 >>
```

Quellcode 4: Beispiel Dictionary Object welches eine Seite im PDF-Dokument darstellt.

Es ist anzumerken, dass die Zeilenumbrüche und Leerzeichen nur für die bessere Lesbarkeit in den Beispielen verwendet werden. Quellcode 5 stellt ein Dictionary Object mit den gleichen Werten dar, jedoch ohne die Verwendung von zusätzlichen Zeilenumbrüchen und Leerzeichen.

```
1 <</Type/Page/Parent 2 0 R/MediaBox[0 0 595 842]>>
```

Quellcode 5: Beispiel Dictionary Object aus Quellcode 4 ohne die Verwendung von Leerzeichen und Zeilenumbrüchen.

Stream Object Ein *Stream Object* ermöglicht es, Binärdaten von unbegrenzter Länge in die PDF-Datei einzubetten. Dieses Objekt wird unter anderem verwendet, um Dateien, wie Schrift-Dateien oder Bilder, in ein PDF-Dokument einzubinden. Das Objekt besteht aus zwei Teilen, zum einen dem sogenannten *Stream-Dictionary* und zum anderen den eigentlichen Daten. Das Stream-Dictionary verhält sich wie jedes andere Dictionary Object und muss mindestens ein Key-Value-Paar besitzen. Es besteht aus dem Name Object /Length, welchem als Wert die Länge des Streams in Byte zugeordnet ist.

Zusätzlich kann der Inhalt eines Streams durch verschiedene Kompressionsalgorithmen komprimiert werden. Ist dies der Fall, benötigt das Dictionary das weitere Key-Value-

¹Eine ausführlichere Definition von Page Objects findet sich in Kapitel 2.4.2.

²Referenzen von Objekten werden in Kapitel 2.3.2 beschrieben.

Paar mit dem Name Object `/Filter` als Schlüssel und dem zugehörigen Name Object des Dekompressionsalgorithmus, welcher angewendet werden muss, um den Stream zu dekomprimieren. Wurden mehrere Algorithmen angewendet, muss ein Array Object angegeben werden, welches alle Name Objects der Algorithmen beinhaltet. Diese müssen in der Reihenfolge angegeben werden, in welcher sie auf den Stream angewendet werden müssen. Auf das Stream-Dictionary folgt eingeleitet durch das Schlüsselwort `stream` und einem Zeilenumbruch der eigentliche Byte-Stream. Abgeschlossen wird er durch einen Zeilenumbruch und das Schlüsselwort `endstream` [20].

```

1 <<
2 /Length 324
3 /Filter /FlateDecode
4 >>
5 stream
6   % 324 Bytes an Daten
7 endstream

```

Quellcode 6: Stream Object mit einer Länge von 324 Bytes, welcher mit FlateDecode (zlib/decode) entkomprimiert werden muss.

Null Object Das Null Object besitzt immer den Wert `null`. Die Verwendung des Null Objects in einem Dictionary Object als Wert ist gleichbedeutend mit dem Weglassen des Key-Value-Paars aus dem Dictionary [20].

2.3.2. Direkte und Indirekte Objekte

Die vorgestellten Objekte können in einer PDF-Datei auf zwei verschiedenen Weisen verwendet werden. Zum einen als *direkte Objekte*, dabei wird der Wert des Objekts, beispielsweise als Wert eines Key-Value-Paars in einem Dictionary Object, eingetragen. Der Wert des `/MediaBox`-Keys in Quellcode 4 ist beispielsweise ein direktes Array Object mit vier direkten Integer Objects.

Zum anderen kann ein Objekt als *indirektes Objekt* angegeben werden. Wird ein Objekt als indirektes Objekt in der PDF-Datei angegeben, so wird statt des Objekts die Referenz zu diesem Objekt angegeben. Dies ist beispielsweise bei dem Wert des `/Parent`-Keys in Quellcode 4 der Fall. Die Referenz besteht dabei immer aus der zugehörigen Objekt-ID, der Generationsnummer des referenzierten Objekts und dem Zeichen `R`.

Die Objekt-ID ist eine positive Ganzzahl, welche das indirekte Objekt eindeutig zuordnet.

Die Generationsnummer ist zu Beginn immer 0 und wird erhöht, wenn das PDF-Dokument bearbeitet wird (siehe Kapitel 2.3.4) [20]. Der Wert des `/Parent`-Keys aus Quellcode 4 ist somit im Objekt mit der ID 2 in der nullten Generation zu finden. Der eigentliche Wert des Objekts findet sich im Body der PDF-Datei (siehe Kapitel 2.3.3). Ein indirektes Objekt wird eingeleitet von der Objekt-ID, der Generationsnummer und dem Schlüsselwort `obj`. Darauf folgt der Wert des jeweiligen Objekts. Abschließend wird das indirekte Objekt mit dem Schlüsselwort `endobj` abgeschlossen [20].

Das Beispiel in Quellcode 7 zeigt, wie das Dictionary Object aus Quellcode 4 aussehen würde, wenn der Wert des Schlüssels `/MediaBox` als indirektes, anstatt als direktes Objekt angegeben wird.

```

1 % Das Page Object selber ist auch ein indirektes Objekt:
2 3 0 obj
3 <<
4 /Type /Page
5 /Parent 2 0 R
6 /MediaBox 4 0 R % Referenz zu Objekt 4
7 >>
8 endobj
9 % Neues Indirektes Objekt für den Wert von /MediaBox
10 4 0 obj
11 [ 0 0 595 842 ]
12 endobj

```

Quellcode 7: Indirektes Objekt für den MediaBox-Wert aus Quellcode 4.

Das Stream Object stellt in diesem Zusammenhang eine Besonderheit dar. Für dieses schreibt die Spezifikation vor, dass es immer als indirektes Objekt angegeben werden muss. Das korrespondierende Stream Dictionary muss dabei als direktes Objekt angegeben werden [20].

2.3.3. Grundstruktur

Im nun Folgenden wird die Grundstruktur von PDF-Dateien vorgestellt. Diese ist in der Regel immer gleich, wird jedoch bei der Verwendung der durch die PDF-Spezifikation festgelegten Funktionalität *Linearization* angepasst. Durch diese Funktionalität wird ermöglicht, dass ein PDF-Dokument nicht komplett heruntergeladen werden muss, um darauf zugreifen und es betrachten zu können. Aufgrund des Umfangs dieser Arbeit wird im

Folgenden nur die allgemeine Grundstruktur vorgestellt und die Linearization nicht weiter betrachtet.

Allgemein unterteilt sich eine PDF-Datei in vier Teile: **Header**, **Body**, **Cross-Reference-Table** und **Trailer**. Die PDF-Datei beginnt mit dem **Header**. Dieser besteht in der Regel aus zwei Zeilen, welche einen Kommentar beinhalten. Kommentare werden in PDF-Dateien mit einem Prozentzeichen eingeleitet. Die erste Zeile startet immer mit %PDF- und gibt damit an, dass es sich um eine PDF-Datei handelt. Folgend auf den Bindestrich wird die PDF-Version angegeben, welche die Datei verwendet. Auch die zweite Zeile ist ein Kommentar und startet somit mit einem Prozentzeichen. Auf dieses folgen mindestens vier Zeichen, welche eine American Standard Code for Information Interchange (ASCII)-Nummer besitzen die größer als 127 ist [20].

Die zweite Zeile wird nur benötigt, wenn die PDF-Datei Binär-Daten enthält [20]. Dies ist in den meisten aktuellen PDF-Dateien der Fall [6]. Der Zweck der Zeile ist es, dass Programme, welche eine ASCII-Binär-Prüfung am Anfang der Datei durchführen, die PDF-Datei immer als Binär-Datei behandeln [20].

¹ %PDF - 1 . 7

² %ääïö

Quellcode 8: PDF-Header für eine PDF-Datei in der Version 1.7.

Auf den Header folgt der **Body**. Dieser stellt den größten Teil der PDF-Datei dar und beinhaltet alle für das PDF-Dokument benötigten Daten in Form von indirekten Objekten. Dies umfasst alle Komponenten eines PDF-Dokuments, wie Seiten, Schriftarten, Bilder und Texte.

An dritter Stelle folgt die **Cross-Reference-Table**. Sie liefert Informationen zu allen indirekten Objekten und Objekt-IDs der PDF-Datei. Wird ein PDF-Dokument erstmalig erstellt besteht die Cross-Reference-Table aus einer *Cross-Reference-Section*. Diese besteht wiederum aus einer *Cross-Reference-Subsection*.

Die Cross-Reference-Section wird mit dem Schlüsselwort `xref` gefolgt von einem Zeilenumbruch eingeleitet. Darauf folgen die zur Cross-Reference-Section gehörigen Cross-Reference-Subsections. In der ersten Zeile wird die erste Objekt-ID der Subsection angegeben, sie ist im Falle eines unbearbeiteten Dokuments immer null. Darauf folgt die Anzahl an Objekten in der Cross-Reference-Subsection. Für ein unbearbeitetes PDF-Dokument ist diese die höchste Objekt-ID plus 1 [20].

Es folgt für jede Objekt-ID eine Zeile, welche immer nach dem in Quellcode 9 angegebenen

Schema aufgebaut ist.

```
1 aaaaaaaaaaa bbbbb c
```

Quellcode 9: Schema einer Zeile in einer Cross-Reference-Subsection.

Für die Übersichtlichkeit werden in der nachfolgenden Erklärung die ersten 10 Zeichen *Teil A*, die darauffolgenden 5 Zeichen *Teil B* und das letzte Zeichen *Teil C* genannt.

Wenn für die Objekt-ID, welche die Zeile darstellt, ein Objekt in der PDF-Datei vorhanden ist, gibt Teil A den Byte-Offset an, wo das Objekt in der Datei zu finden ist. Dieser wird immer durch vorangestellte Nullen auf zehn Stellen aufgefüllt. Der zweite Teil B gibt die Generationsnummer des Objekts an, welche auf fünf Stellen mit vorangestellten Nullen aufgefüllt wird. Als letztes folgt für Teil C der Buchstabe n. Dieser gibt an, dass die Objekt-ID verwendet wird [20].

Wird die Objekt-ID, welche der Zeile zugeordnet ist, nicht verwendet, so gibt Teil A die nächste freie Objekt-ID an. Auch diese wird auf zehn Stellen durch vorangestellte Nullen aufgefüllt. Ist keine weitere freie Objekt-ID vorhanden wird die Objekt-ID 0 angegeben. Teil B gibt die Generationsnummer an, welche benutzt werden soll, wenn die Objekt-ID wieder verwendet wird. Die höchst mögliche Generationsnummer ist 65535. Wird diese angegeben, kann die Objekt-ID nicht erneut verwendet werden. Teil C wird für eine freie Objekt-ID durch ein f für free (frei) ersetzt.

Einen Sonderfall stellt die Zeile für die Objekt-ID 0 dar, da in einer PDF-Datei kein Objekt mit dieser Objekt-ID existieren darf. Sie wird lediglich als Beginn der verketteten Liste von freien Objekt-IDs verwendet und besitzt somit immer die Generationsnummer 65535 [20].

Die Cross-Reference-Table bietet dadurch zwei Informationen. Zum einen darüber, an welcher Position sich Objekte in der PDF-Datei befinden. Zum anderen, welche Objekt-IDs frei sind und bei einem Update der Datei wieder verwendet werden können (mehr dazu in Kapitel 2.3.4).

Die in Quellcode 10 dargestellte Cross-Reference-Section sagt somit aus, dass die PDF-Datei vier indirekte Objekte mit den Objekt-IDs 1, 2, 3 und 5 besitzt. Außerdem wird angegeben, dass beispielsweise das Objekt mit der ID 1 beim 21 Byte der Datei beginnt. Die Objekt-ID 4 wird nicht verwendet, kann aber mit der Generationsnummer 1 wieder verwendet werden.

```
1 xref
```

```
2 0 6
```

```

3 0000000004 65535 f
4 0000000021 00000 n
5 0000000300 00000 n
6 0000032403 00000 n
7 0000000000 00001 f
8 0000009327 00000 n

```

Quellcode 10: Beispielhafte Cross-Reference-Table mit 6 Objekten, wovon vier verwendet werden und eines frei ist.

Mit der PDF-Version 1.5 wurde mit dem *Cross-Reference-Stream* eine Alternative zur Cross-Reference-Table eingeführt. Er ermöglicht unter anderem größere maximale Dateigrößen, da der Byte-Offset nicht mehr auf maximal zehn Stellen beschränkt ist [6]. Der Cross-Reference-Stream wird in dieser Arbeit nicht näher betrachtet.

Als letzter Teil einer PDF-Datei, folgt der **Trailer**. Dieser wird mit dem Schlüsselwort **trailer** eingeleitet. Auf dieses folgt ein Dictionary Object, welches mindestens folgende Key-Value-Paare beinhaltet [20]:

/Size - Anzahl der indirekten Objekte der PDF-Datei als Integer Object.

/Root - Indirekte Referenz zum Document-Catalog (siehe Kapitel 2.4.1).

Neben den zwingend erforderlichen Werte-Paaren gibt es noch andere Paare, die angegeben werden können. Bei diesen handelt es sich unter anderem um zwingend notwendige Parameter, wenn das PDF-Dokument verschlüsselt ist. Sie werden in dieser Arbeit nicht näher vorgestellt³.

Auf das zum Trailer gehörende Dictionary Object folgt das Schlüsselwort **startxref**. Dieses kennzeichnet, dass in der nächsten Zeile der Byte-Offset angegeben wird, der den Beginn der Cross-Reference-Table darstellt. Abschließend folgt der End-Of-File Marker, welcher aus zwei Prozentzeichen und der Zeichenkette **EOF** besteht [20].

```

1 trailer
2 <<
3 /Size 15
4 /Root 1 0 R
5 >>
6 startxref
7 200
8 %%EOF

```

Quellcode 11: Beispielhafter Trailer einer PDF-Datei.

³Eine ausführliche Beschreibung findet sich in der PDF-Spezifikation [20] auf Seite 43).

Der in Quellcode 11 dargestellte Beispiel-Trailer gibt an, dass die PDF-Datei aus 15 indirekten Objekten besteht, wovon der Document Catalog (siehe Kapitel 2.4.1) das Objekt mit der ID 1 ist. Zudem wird angegeben, dass die Cross-Reference-Table beim zweihundertsten Byte der Datei beginnt.

In Anhang A.I ist der Quellcode für eine *Hello-World-PDF-Datei* zu finden. Dieser wendet den im vorherigen beschriebenen Aufbau und die Objekt-Typen an, um ein Dokument zu erzeugen. Das Dokument besteht aus einer Seite auf welcher der Text „Hello World“ dargestellt wird.

2.3.4. Inkrementelle Updates

Wird ein bestehendes PDF-Dokument bearbeitet, erfolgt die Sicherung der Änderungen mittels der Methode der Inkrementellen Updates. Hierbei bleibt der Inhalt der Ausgangs-PDF-Datei unverändert bestehen. Für die vorgenommenen Änderungen werden ein weiterer Body, eine weitere Cross-Reference-Section und ein weiterer Trailer an das Ende der Datei angehängt. Der neue Body beinhaltet dabei ausschließlich Objekte, die bearbeitet wurden und Objekte, welche durch das Update neu hinzugekommen sind. Die Cross-Reference-Section beinhaltet ebenso nur die Objekt-IDs, welche bearbeitet oder neu verwendet werden. Ab dem ersten Inkrementellen Update bezieht sich die Bezeichnung *Cross-Reference-Table* nicht mehr nur auf die erste Cross-Reference-Section, sondern bezeichnet alle Cross-Reference-Sections [20].

Der Trailer des Updates übernimmt alle Werte-Paare, welche auch im Original Trailer vorhanden sind und passt diese gegebenenfalls an. Zudem wird der Trailer um einen Eintrag mit dem Schlüssel */Prev* erweitert, der Wert ist der Byte-Offset der Cross-Reference-Section der vorherigen Version. Der Aufbau der PDF-Datei nach einem, beziehungsweise mehreren Updates wird in Abbildung 1 dargestellt⁴.

2.3.5. Boxmodel und Boxes

Da das Portable Document Format auch im Druck Verwendung findet, können für die Seiten in der PDF-Datei bis zu fünf verschiedene Boxen definiert werden. Diese werden in Abbildung 2 dargestellt. Alle Boxen werden, wenn sie verwendet werden, für eine oder

⁴Ein Ausführliches Beispiel zu inkrementellen Updates findet sich in der PDF-Spezifikation [20] im Anhang H7 ab Seite 710.

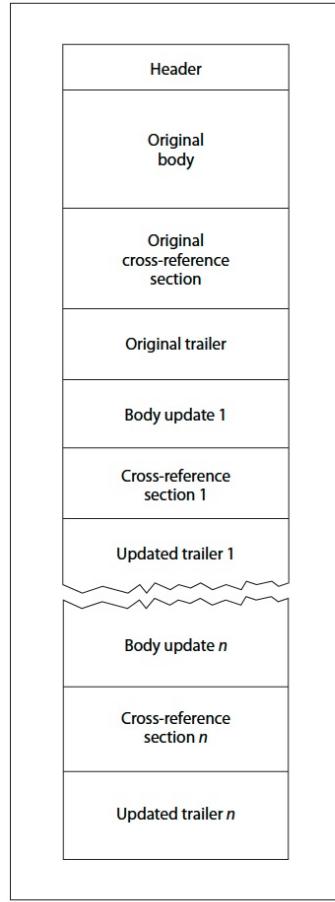


Abbildung 1: Aufbau einer PDF-Datei mit Inkrementellen Updates. Entnommen aus [20].

mehrere Seiten im Page Object oder Pages Object (siehe Kapitel 2.4.2) als Array Object bestehend aus vier Numeric Objects dargestellt. Die numerischen Werte geben zwei gegenüberliegende Eckpunkte an, der Aufbau ist $[x_1 \ y_1 \ x_2 \ y_2]$ [20].

Media Box Die Media Box stellt die maximale Fläche einer Seite dar und muss für jede Seite angegeben werden. In der Druckindustrie ist sie meistens die Größe der bedruckbaren Fläche [20].

Crop Box Die Crop Box definiert den Bereich der Media Box, welcher in einem PDF-Viewer angezeigt werden soll beziehungsweise beim Druck gedruckt werden sollen. Alle Elemente die über diese Box hinausragen werden nicht gedruckt oder angezeigt. Ist keine Crop Box angegeben, so besitzt sie die gleiche Größe wie die Media Box [20].

Bleed Box Ab der Version 1.3 kann zusätzlich eine Bleed Box angegeben werden. Sie gibt den Bereich an, an welcher das bedruckte Medium beschnitten werden soll, um beispiels-

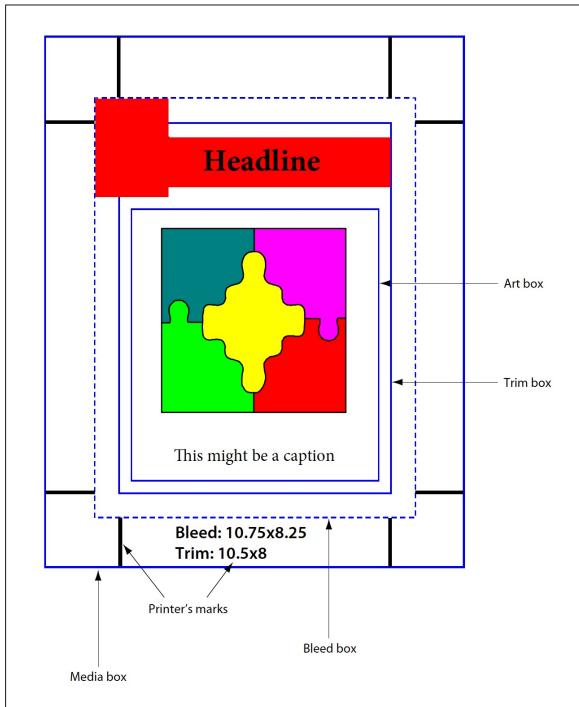


Abbildung 2: Darstellungen aller Boxen in einem PDF. Entnommen aus [20].

weise Druck-Markierung abzuschneiden. Ist keine Größe für die Bleed Box angegeben, wird die Größe der Crop Box verwendet [20].

Trim Box Ebenso mit der Version 1.3 eingeführt wurde die Trim Box. Sie stellt die Größe der eigentlichen Seite dar. Auch diese nimmt den Wert der Crop Box an, wenn keine Größe definiert ist [20].

Art Box Die Art Box gibt den Bereich des in der Spezifikation als „page’s meaningful content“ bezeichneten Inhalts an [20]. Diese hat sich, laut Rosenthal, jedoch nicht durchgesetzt [6].

2.4. Spezifische PDF-Objekte

Das nun folgende Kapitel stellt einige spezifische Objekte vor, welche in einer PDF-Datei Verwendung finden beziehungsweise verwendet werden können. Diese basieren auf den grundlegenden Objekt-Typen, welche bereits im vorherigen Unterkapitel vorgestellt wurden. In den meisten Fällen gibt die PDF-Spezifikation für die verschiedenen Objekte genau vor, welche Werte benötigt werden und in welchem Datentyp sie angegeben werden müssen. Dies ist der Fall, damit alle PDF-Viewer, welche der PDF-Spezifikation entsprechen,

diese gleich behandeln.

Es ist anzumerken, dass die im Folgenden aufgeführten Objekte, zum einen nicht die Gesamtheit aller in PDF-Dateien möglichen Objekte darstellen. Zum anderen haben alle Auflistungen möglicher Key-Value-Paare keinen Anspruch auf Vollständigkeit. Dies resultiert aus dem umfassenden Umfang der PDF-Spezifikation, welcher den Rahmen dieser Arbeit übersteigt. Für einen umfassenden Überblick über anwendbare Objekte und deren Ausprägungen sollte deshalb die PDF-Spezifikation verwendet werden.

2.4.1. Document Catalog

Der *Document Catalog* stellt den Einstiegspunkt zu allen anderen Objekten in der PDF-Datei dar und bildet damit den Wurzelknoten eines gerichteten Graphen, wie in Abbildung 3 dargestellt.

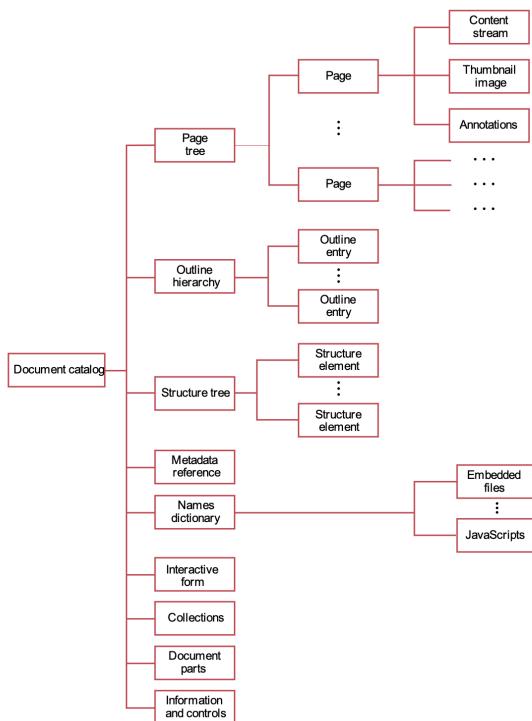


Abbildung 3: Darstellung der PDF-Datei als gerichteter Graph mit dem Document Catalog als Wurzelknoten. Entnommen aus [21].

Aus diesem Grund wird das zugehörige indirekte Objekt im Trailer der PDF-Datei unter dem Schlüsselwort `/Root` referenziert. Bei dem Document Catalog handelt es sich, um ein Dictionary Object, welches mindestens die folgenden zwei Key-Value-Paare beinhaltet: Zum einen den Schlüssel `/Type`, welcher immer als Wert das Name Object `/Catalog` sein muss. Dieser gibt an, dass es sich um den Document Catalog handelt. Zum anderen

muss dem Schlüssel `/Pages` eine Referenz zum Page-Tree (siehe Kapitel 2.4.2) zugeordnet werden [20].

Neben den erforderlichen Werten können noch weitere Key-Value-Paare angegeben werden, wie das Outline-Dictionary (siehe Kapitel 2.4.6) oder Informationen zur Darstellung des Dokuments in einem PDF-Viewer. Mit dem Schlüssel `/PageLayout` kann beispielsweise angegeben werden, ob standardmäßig eine oder zwei Seiten angezeigt werden sollen [20].

Der in Quellcode 12 dargestellte Document Catalog gibt in Zeile 3 an, dass der Page-Tree im Objekt mit der Objekt-ID 2 zu finden ist. Außerdem wird in Zeile 4 angegeben, dass beim Betrachten des Dokuments mit einem PDF-Viewer immer nur eine Seite angezeigt werden soll.

```
1 <<
2 /Type /Catalog
3 /Pages 2 0 R
4 /PageLayout /SinglePage
5 >>
```

Quellcode 12: Beispielhafter Document Catalog, durch welchen die Seiten des PDF-Dokument in einer Spalte angezeigt werden.

2.4.2. Pages und Page-Tree

Der *Page-Tree* stellt einen Teil des beim Document Catalog startenden gerichteten Graphen dar. Er beinhaltet alle Seiten und deren Reihenfolge im PDF-Dokument. Alle Knoten des Page-Trees sind entweder Zwischenknoten, die sogenannten *Page Tree Nodes* oder Blattknoten, den *Page Objects*. Ein Page Object, stellt dabei eine bestimmte Seite dar.

Der Wurzelknoten des Page-Tree muss immer ein Zwischenknoten sein [20].

Der Zwischenknoten ist ein Dictionary Object, welches die in Tabelle 2 dargestellten Key-Value-Paare besitzen muss.

Neben diesen Paaren kann ein Zwischenknoten auch weitere Key-Value-Paare besitzen. Dabei handelt es sich um Wertepaare, die zu den Blattknoten gehören, jedoch für alle auf den Zwischenknoten folgenden Seiten gleich sind. Wenn beispielsweise alle Seiten, welche auf einen Zwischenknoten folgen, um 90 Grad gedreht werden sollen kann für den Schlüssel `/Rotate` das Integer Object 90 angegeben werden. Somit muss es nicht in allen Page Objects definiert werden [20].

Schlüssel	Wert
/Type	Muss immer das Name Object /Pages sein.
/Parent	Indirekte Referenz zum übergeordneten Knoten. Wird bei allen Zwischenknoten außer dem Wurzelknoten benötigt.
/Kids	Ein Array Object von indirekten Referenzen zu allen direkten nachfolgenden Knoten.
/Count	Die Anzahl von nachfolgenden Page Objects als Integer Object.

Tabelle 2: Notwendige Key-Value-Paare eines Zwischenknotens. Eigene Darstellung nach [20].

Analog zu den Zwischenknoten sind auch die Blattknoten Dictionary Objects mit den in Tabelle 3 dargestellten erforderlichen Key-Value-Paaren.

Schlüssel	Wert
/Type	Ist immer das Name Object /Page.
/Parent	Indirekte Referenz zum übergeordneten Pages Object.
/Resources	Dictionary Object mit allen für die Seite benötigten Ressourcen, beispielsweise Schriftarten oder Bilder (siehe Kapitel 2.4.3).
/MediaBox	Die Größe der Seite (siehe Kapitel 2.3.5).

Tabelle 3: Notwendige Key-Value-Paare in Dictionary Objects vom Typ Page. Eigene Darstellung nach [20].

Das Key-Value-Paar mit dem Schlüssel /Content ist zwar optional, ist in der Regel aber bei allen Page Objects vorhanden. Der zugehörige Wert wird als Content Stream bezeichnet und ist entweder ein Stream Object oder ein Array von Stream Objects. Ist kein Content Stream angegeben, handelt es sich bei der Seite um eine leere Seite. Ist ein Content Stream angegeben, so enthält dieser alle Informationen zu den auf der Seite dargestellten Inhalten (siehe Kapitel 2.4.5)⁵.

In Quellcode 13 ist beispielhaft ein Page-Tree für ein PDF-Dokument mit drei Seiten abgebildet. Dieser besteht aus dem Wurzelknoten Objekt 2, auf den der Zwischenknoten Objekt 3 folgt. Dieser Zwischenknoten enthält als Kinder die Blattknoten mit den Objekt-IDs 5 und 6, welche die erste und zweite Seite darstellen. Der letzte Blattknoten ist Objekt 4, welches die dritte Seite darstellt.

⁵ Alle weiteren möglichen Key-Value-Paare eines Page Object finden sich in der PDF-Spezifikation [20] ab Seite 77.

```

1 2 0 obj      % Wurzelknoten des Page-Trees
2 <<
3 /Type /Pages
4 /Kids [ 3 0 R 4 0 R ]
5 /Count 3
6 >>
7 endobj
8 3 0 obj      % Zwischenknoten
9 <<
10 /Type /Pages
11 /Parent 2 0 R
12 /Kids [ 5 0 R 6 0 R ]
13 /Count 2
14 >>
15 endobj
16 4 0 obj      % Seite 3
17 <<
18 /Type /Page
19 /Parent 2 0 R
20 /Resources 7 0 R
21 /MediaBox [ 0 0 595 842 ]
22 >>
23 endobj
24 5 0 obj      % Seite 1
25 <<
26 /Type /Page
27 /Parent 3 0 R
28 % [ ... ] weitere Werte
29 >>
30 endobj
31 6 0 obj      % Seite 2
32 <<
33 /Type /Page
34 /Parent 3 0 R
35 % [ ... ] weitere Werte
36 >>
37 endobj

```

Quellcode 13: Beispielhafter Page-Tree mit drei Seiten.

2.4.3. Resource Dictionary

Das *Resource Dictionary* gibt alle Ressourcen an, welche für die zugehörige Seite beziehungsweise Seiten benötigt werden. Unter Ressourcen verstehen sich unter anderem Schriftarten und Bilder. Im Resource Dictionary werden diese unterteilt nach der Ressourcenart angegeben. Dabei sind die Schlüssel des Dictionary Objects die Name Objects der verschiedenen Ressourcenarten. Beispielsweise `/Font` für Schriftarten oder `/XObject` für externe Objekte, wie Bilder. Der zu den Schlüsseln gehörige Wert ist in der Regel auch ein Dictionary Object. Dieses ordnet den im Content Stream verwendeten Ressourcennamen, über eine indirekte Referenz das entsprechende Objekt zu [20]. Das Beispiel in Quellcode 14 zeigt ein Resource Dictionary, welches für Schriften das Name Object `/F1` dem Objekt mit der ID 10 zuordnet und `/Helvetica` dem Objekt mit der Objekt-ID 11. Analog dazu wird für externe Objekte das Name Object `/Bild` dem Objekt mit der ID 12 zugeordnet.

```

1 <<
2 /Font <<
3 /F1 10 0 R
4 /Helvetica 11 0 R
5 >>
6 /XObject <<
7 /Bild 12 0 R
8 >>
9 >>
```

Quellcode 14: Ein beispielhaftes Resource Dictionary mit 2 Schriftarten und einem externen Objekt.

2.4.4. Schriften

Soll Text in PDF-Dokumenten dargestellt werden muss für diesen eine Schriftart angegeben werden. Das Portable Document Format erlaubt die Verwendung folgender Formate [6]:

- Type 1
- TrueType
- OpenType
- Type 3

- Type 0 (auch CIDFont oder Composite Font genannt)

Neben diesen Formaten gibt die PDF-Spezifikation 14 Standard Type1-Schriftarten vor⁶.

Diese Schriftarten sollen von jedem PDF-Viewer bereitgestellt werden, wodurch die Nutzung dieser vereinfacht wird.

Je nach verwendetem Format der Schriftart werden unterschiedliche Objekte benötigt, um diese in einer PDF-Datei anzugeben und zu nutzen. Da eine ausführliche Vorstellung den Umfang dieser Arbeit übersteigen würde, werden diese im Folgenden nur vage definiert und vorgestellt⁷. Wird ein Name Object für eine Schriftart in einem Resource-Dictionary angegeben, so ist dessen Wert das zu der Schriftart gehörige *Font Dictionary*.

```

1 <<
2 /Type /Font
3 /Subtype /Type1
4 /BaseFont /Helvetica
5 >>
```

Quellcode 15: Beispielhaftes FontDictionary für eine Type1 Schriftart, welche in den Standardschriften vorhanden ist.

Dieses Dictionary Object muss je nach Typ der Schriftart verschiedene Key-Value-Paare besitzen. Für die Verwendung der Standard-Schriftarten sind die in Quellcode 15 dargestellten Werte ausreichend. Für die Verwendung anderer Schriften werden zusätzliche Key-Value-Paare benötigt. Dies sind je nach Format unter anderem die Breiten der einzelnen Zeichen oder der *FontDescriptor*. Bei einem FontDescriptor handelt es sich um ein weiteres Dictionary Object, welches weitere Metriken der Schrift beinhaltet. Unter anderem wird in diesem, wenn die Schriftart in das PDF-Dokument eingebettet ist, für den Schlüssel */FontFile*⁸ eine Referenz zu einem Stream Object angegeben in welchem sich die Binärdaten der Schriftart-Datei befinden [3].

2.4.5. Content Stream

Der *Content Stream* stellt eine Sequenz von Anweisungen dar, wie Inhalte auf der Seite platziert werden sollen. Wie schon unter Kapitel 2.4.2 beschrieben, kann der zu einer Seite

⁶Eine Liste aller Standard-Schriftarten findet sich in Anhang A.II

⁷Eine ausführliche Darstellung findet sich in der PDF-Spezifikation [20] ab Seite 253.

⁸Je nach Typ der Schriftart muss als Schlüssel stattdessen */FontFile2* oder */FontFile3* verwendet werden

gehörige Content Stream entweder ein einzelnes Stream Object oder ein Array Object von Stream Objekten sein. Liegt nur ein Stream Object vor, werden die im Stream angegeben Anweisungen nacheinander ausgeführt. Dies ist ebenso der Fall, wenn es sich um ein Array Object von Stream Objects handelt. Dabei werden die Stream Objects in der Reihenfolge nacheinander ausgeführt wie sie im Array Object angegeben sind [20].

Die im Content Stream angegebenen Anweisungen werden durch die Verwendung von verschiedenen Operatoren mit zugehörigen Operanden dargestellt. Eine Auswahl der Operatoren werden für das Platzieren von Bildern und Texten im weiteren vorgestellt. Neben diesen existieren noch weitere, um beispielsweise Vektorgrafiken zu erstellen und zu platzieren. Die Verwendung dieser wird in dieser Arbeit nicht näher vorgestellt.

Bilder Für das Platzieren von Bildern auf einer Seite müssen zwei Voraussetzungen erfüllt sein. Einerseits muss für das Bild ein Stream Object existieren. Andererseits muss diesem Stream Object, im Resource Dictionary der Seite, ein Name Object zugeordnet sein. Der Stream des Stream Objects enthält die zur Darstellung benötigten Daten der Bilddatei. Mit Ausnahme von Bildern im Joint Photographic Experts Group (JPG / JPEG)-Format kann der Inhalt der Bilddatei jedoch nicht ohne Anpassungen in den Stream geschrieben werden [6]. Aufgrund der Komplexität der notwendigen Verarbeitung wird im Folgenden nur das Einbetten von JPEG-Bildern betrachtet.

Schlüssel	Wert
/Type	Muss als Wert das Name Object /XObject besitzen, wenn das Key-Value-Paar angegeben wird.
/Subtype	Ist für ein Bild immer das Name Object /Image.
/Width	Ein Integer Object, welches die Breite in Bildpixeln angibt.
/Height	Ein Integer Object, welches die Höhe in Bildpixeln angibt.
/ColorSpace	Ein Name Object, welches den verwendete Farbraum der Bildpixel angibt, beispielsweise DeviceRGB.
/BitsPerComponent	Ein Integer Object, welches die Anzahl der Bits für eine Farbe angibt.

Tabelle 4: Notwendige Key-Value-Paare eines Image Dictionary. Eigene Darstellung nach [20].

Das zugehörige Stream Dictionary, wird im Fall eines Bildes *Image Dictionary* genannt und enthält weitere Informationen zum Bild. Neben dem allgemein im Stream Dictionary

erforderlichen Key-Value-Paar mit dem Schlüssel `/Length` sind für das Image Dictionary weitere Paare erforderlich. Diese werden für ein Bild im JPEG-Format in Tabelle 4 erläutert [20].

Ist für das Bild ein Stream Object vorhanden und im Resource Dictionary der Seite einem Name Object zugeordnet, so kann es im Content Stream verwendet und somit auf der Seite platziert werden. Dafür muss im Content Stream durch die Verwendung des Operators `cm` die *Current Transformation Matrix* (CTM) angepasst werden. Mit diesem Operator lassen sich unter anderem Position, Rotation und Größe anpassen. Der Operator `cm` besitzt hierfür sechs Operanden in Form von sechs Numeric Objects, welche vor ihm angegeben werden. Je nach Aufbau der Operanden erfolgt eine der angegebenen Transformationen [3]:⁹

- `1 0 0 1 <X> <Y> cm` - Verschiebung der Position um den an fünfter Stelle (`<X>`) angegebenen Wert auf der X-Achse, analog dazu die sechste Stelle (`<Y>`) für die Y-Achse.
- `<Breite> 0 0 <Höhe> 0 0 cm` - Anpassung der Größe, wobei der Wert an der ersten Stelle die Breite angibt und die vierte Stelle die Höhe.
- `<cos(x)> <sin(x)> <-sin(x)> <cos(x)> 0 0 cm` - Rotation gegen den Uhrzeigersinn. Die Werte sind dabei die Ergebnisse der angegebenen Winkelfunktionen, wobei `x` den Radianen der Rotation darstellt.

Nachdem mit dem `cm`-Operator die Position, Größe und gegebenenfalls Rotation angegeben wurden kann der `Do`-Operator verwendet werden, um das Bild zu platzieren. Er erhält als vorangestellten Operanden das Name Object des Bildes [3].

```

1 1 0 0 1 200 200 cm
2 50 0 0 25 0 0 cm
3 \Bild Do

```

Quellcode 16: Anweisungen im Content Stream zum Platzieren eines Bildes.

Die in Quellcode 16 dargestellten Anweisungen führen somit folgende Aktionen aus:

1. Verschiebung der Position um 200pt in Richtung der X-Achse, sowie um 200pt auf der Y-Achse.

⁹In den Beispielen finden Platzhalter Verwendung, welche durch spitze Klammern dargestellt werden. Diese müssen bei der Verwendung in einer PDF-Datei durch die entsprechenden Werte ersetzt werden.

2. Skalierung auf 50pt Breite und 25pt Höhe.
3. Platzieren des Bildes mit dem Name Object `/Bild` an der Position und in der Größe wie diese von der CTM vorgegeben sind.

Text Damit Texte auf einer Seite platziert werden können, müssen wie auch bei Bildern einige Voraussetzungen erfüllt sein. Zum einen muss ein Font Dictionary für die zu verwendende Schriftart in der PDF-Datei existieren. Zum anderen müssen auch diese im Resource-Dictionary jeweils einem Name Object zugeordnet sein, sodass sie im Content Stream verwendet werden können. Des Weiteren muss im Content Stream mit dem Operator `BT` in den Text-Modus gewechselt werden. Analog dazu beendet der Operator `ET` den Text-Modus [6].

Wurde in den Text-Modus gewechselt, können textspezifische Operatoren verwendet werden, um sowohl Position, Schriftart und Schriftgröße für den Text anzugeben, als auch Texte auf der Seite zu platzieren [20]:¹⁰

- `<SCHRIFT> <GRÖÙE> Tf` - Festlegen der Schriftart durch Angabe des entsprechenden Name Objects an erster Stelle und Festlegen der Schriftgröße in Pt (zweite Stelle).
- `<R-WERT> <G-WERT> <B-WERT> rg` - Festlegen der Schriftfarbe im RGB-Format, wobei jeder Wert zwischen 0 und 1 sein muss.
- `<X> <Y> Td` - Festlegen der Startposition des zu platzierenden Texts.
- `<TEXT> Tj` - Platzieren des eigentlichen Texts, wobei der Wert ein String Object ist.

Dementsprechend werden folgende Aktionen für die Anweisungen aus Quellcode 17 durchgeführt:

1. Wechsel in den Text-Modus.
2. Festlegen der Schriftart, welche im Resource-Dictionary mit dem Name Object `/Helvetica` angegeben wurde und setzen der Schriftgröße auf 12pt.
3. Einstellung der Schriftfarbe auf Rot.
4. Festsetzung der Startposition auf 100pt auf der X-Achse und 100pt auf der Y-Achse.

¹⁰In den Beispielen finden Platzhalter Verwendung, welche durch spitze Klammern dargestellt werden. Diese müssen bei der Verwendung in einer PDF-Datei durch die entsprechenden Werte ersetzt werden.

5. Platzierung des String Objects „Hello World“ mit den in den vorherigen Operatoren angegebenen Werten.
6. Beenden des Textmodus.

```

1 BT
2 /Helvetica 12 Tf
3 1 0 0 rg
4 100 100 Td
5 (Hello World) Tj
6 ET

```

Quellcode 17: Anweisungen im Content Stream zum Platzieren des Texts „Hello World“ in rot mit der Schriftart Helvetica und der Größe 12pt.

2.4.6. Document Outline

Unter der *Document Outline* eines PDF-Dokuments versteht sich ein Inhaltsverzeichnis, welches bei der Betrachtung des Dokuments mit einem PDF-Viewer, meist auf der linken Seite, dargestellt wird. Dieses stellt Verlinkungen zu bestimmten Stellen des Dokuments bereit. Dabei können die einzelnen Elemente verschachtelt werden, sodass eine hierarchische Struktur entsteht. Abbildung 4 stellt einen Ausschnitt der Document Outline der PDF-Version dieser Arbeit in verschiedenen PDF-Viewern dar.

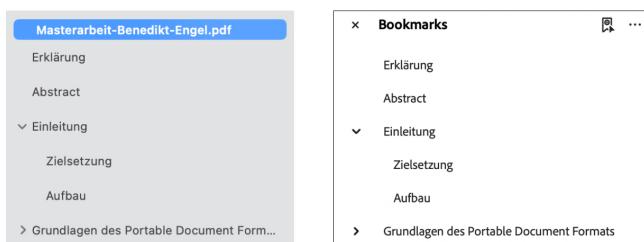


Abbildung 4: Ausschnitt der Darstellung von Document Outlines in Apple Preview (links) und Adobe Acrobat Reader (rechts). Eigene Darstellung.

In der PDF-Datei stellt das *Outlines-Dictionary* den Einstiegspunkt für die Definition der Outlines dar. Es wird im Document Catalog unter dem Schlüssel `/Outlines` entweder als direktes oder indirektes Objekt angegeben. Im Gegensatz zu anderen Dictionary Objects, ist für das Outlines-Dictionary der Schlüssel `/Type` optional. Dies resultiert daraus, dass das Outlines Dictionary nur einmal in der PDF-Datei existiert und der Typ bereits indirekt

durch den Schlüssel im Document Catalog angegeben ist. Sollte er dennoch vorhanden sein, muss er zwangsweise den Wert `/Outlines` besitzen [20]. Darüber hinaus besitzt das Dictionary die in Tabelle 5 dargestellten weiteren Key-Value-Paare.

Schlüssel	Wert
<code>/First</code>	Indirekte Referenz zum Objekt des ersten Outline-Elements, welches in der Liste dargestellt wird.
<code>/Last</code>	Indirekte Referenz zum Objekt des letzten Outline-Elements, welches in der Liste dargestellt wird.
<code>/Count</code>	Integer Object mit der Anzahl der insgesamt sichtbaren Outline-Elemente. Sollen keine Elemente angezeigt werden soll dieses Key-Value-Paar nicht angegeben werden.

Tabelle 5: Key-Value-Paare welche in einem Outlines-Dictionary möglich sind. Eigene Darstellung nach [20].

Jedes Outline-Element wird dabei durch ein *Outline-Item-Dictionary* dargestellt. Dieses Dictionary muss je nach Position unterschiedliche Key-Value-Paare besitzen, welche in Tabelle 6 beschrieben werden.

Dadurch, dass sowohl das vorherige Element auf derselben Ebene, als auch des nachfolgenden Element angegeben werden, entsteht eine doppelt verkettete Liste der Outline-Items, welche im PDF-Viewer dargestellt werden. Des Weiteren übernimmt der Schlüssel `/Count` zwei Aufgaben. Zum einen gibt dieser, wie schon in der Tabelle 6 dargelegt, an, wie viele Unterelemente auf ein Element folgen. Zum anderen wird durch das Vorzeichen angegeben, ob die Liste der Unterelemente angezeigt werden sollen oder nicht [6].

Das Beispiel in Quellcode 18 zeigt ein Outline-Dictionary und Outline-Items, für ein PDF-Dokument. In den Document Outlines ist durch die Angabe eines positiven Werts für den Schlüssel `/Count` die erste Ebene geöffnet (siehe Zeile 5). Diese zeigt zwei Elemente, das Outline-Item mit der ID 31 und das Item mit der ID 32. Dabei besitzt das zweite Element zwei Kind-Elemente, welche durch den negativen Wert von `/Count` in Zeile 23 nicht ausgeklappt dargestellt werden.

Sollen die Document Outlines standardmäßig beim Betrachten des PDF-Dokuments in einem PDF-Viewer geöffnet werden, kann im Document Catalog für den Schlüssel `/PageMode` das Name Object `/UseOutlines` angegeben werden. Ist dieses Key-Value-Paar gesetzt, soll der PDF-Viewer automatisch die Outlines anzeigen [20]. Es ist zu beachten, dass dies nicht alle PDF-Viewer unterstützen. Beispielsweise belässt *Apple Preview* die Seitenleiste auf der zuletzt verwendeten Ansicht, auch wenn ein PageMode angegeben ist.

Schlüssel	Wert
/Titel	String Object, welches den anzuzeigenden Text beinhaltet.
/Parent	Indirekte Referenz zum Eltern-Element oder, wenn sich das Outline-Element auf der obersten Ebene befindet, zum Outlines-Dictionary.
/Prev	Indirekte Referenz zum vorherigen Outline-Item-Dictionary, falls eines vorhanden ist.
/Next	Indirekte Referenz zum nächsten Outline-Item-Dictionary, falls eines vorhanden ist.
/First	Indirekte Referenz zum ersten Kind Outline-Item-Dictionary.
/Last	Indirekte Referenz zum letzten Kind Outline-Item-Dictionary.
/Count	Wenn Kind-Objekte vorhanden sind und diese angezeigt werden sollen ist der Wert ein Integer Object mit der Anzahl aller sichtbaren untergeordneten Objekte. Sollen die Kind-Objekte nicht angezeigt werden, so ist das Integer Object negativ und der absolute Wert stellt die Anzahl an sichtbaren Elementen dar, wenn es geöffnet wäre.
/Dest	Internes Ziel des Outline-Items, hier sind verschiedene Objekt-Typen möglich. Beispielsweise ein Array Object bestehend aus einer indirekten Referenz zum Page Object, und dem Name Object /Fit. Dies würde die entsprechende Seite den PDF-Viewer füllend öffnen.

Tabelle 6: Ausschnitt von Key-Value-Paare welche in einem Outline-Item-Dictionary möglich sind. Eigene Darstellung nach [20].

```

1 30 0 obj      % Outlines - Dictionary
2 <<
3 /First 31 0 R
4 /Last 32 0 R
5 /Count 2
6 >>
7 endobj
8 31 0 obj      % Outline - Item (Position 1.)
9 <<
10 /Title (Erstes Element)
11 /Parent 30 0 R
12 /Next 32 0 R
13 /Dest [40 0 R /Fit]
14 >>
15 endobj

```

```

16 32 0 obj      % Outline-Item (Position 2.)
17 <<
18 /Title (Zweites Element)
19 /Parent 30 0 R
20 /Prev 31 0 R
21 /First 33 0 R
22 /Last 34 0 R
23 /Count -2
24 /Dest [41 0 R /Fit]
25 >>
26 endobj
27 33 0 obj      % Outline-Item (Position 2.1)
28 <<
29 /Title (Erstes Kind-Element des zweiten Elements)
30 /Parent 32 0 R
31 /Next 34 0 R
32 /Dest [42 0 R /Fit]
33 >>
34 endobj
35 34 0 obj      % Outline-Item (Position 2.2)
36 <<
37 /Title (Zweites Kind-Element des zweiten Elements)
38 /Parent 32 0 R
39 /Prev 33 0 R
40 /Dest [43 0 R /Fit]
41 >>
42 endobj

```

Quellcode 18: Beispielhafte Document-Outlines einer PDF-Datei mit vier Elementen.

2.4.7. Annotations

Unter *Annotations* verstehen sich Objekte, welche interaktive Elemente auf Seiten darstellen. Bei Annotations handelt es sich unter anderem um Notizen, Töne, Links und abspielbare Videos. Im Weiteren werden nur Annotations für interne und externe Verlinkungen vorgestellt¹¹. Eine Annotation wird durch ein Dictionary Object, dem *Annotation-Dictionary*, abgebildet. Dieses muss mindestens die Key-Value-Paare */Rect* und */Subtype*

¹¹Eine vollständige Liste aller möglichen Annotations findet sich in der PDF-Spezifikation [20] auf Seite 390.

besitzen. Der Wert für den Schlüssel `/Subtype` ist dabei sowohl bei internen als auch externen Verlinkungen immer das Name Object `/Link`. Der Wert von `/Rect` ist ein Array Object von vier Numeric Objects. Diese geben, analog zu den Arrays der Boxen von Seiten, zwei gegenüberliegende Eckpunkte des klickbaren Bereichs der Annotation an [20]. Für eine interne Verlinkung wird zusätzlich das Key-Value-Paar mit dem Schlüssel `/Dest` benötigt [20]. Der Wert von diesem ist, wie schon aus den Document-Outlines bekannt, ein Array Object, welches die entsprechende Ziel-Seite und die Darstellung angibt.

Die in Quellcode 19 dargestellte erste Annotation (Zeile 1-8) spannt mit dem in Zeile 5 dargestellten Array Object eine klickbare Fläche auf, welche die komplette untere Hälfte einer A4-Seite bedeckt. Wird auf diese geklickt, öffnet sich die Seite, welche unter dem in `/Dest` angegebenen Objekt mit der ID 50 definiert ist. Durch den im Weiteren angegebenen Parameter `/Fit` wird die Seite den PDF-Viewer füllend dargestellt.

Soll stattdessen eine externe Ressource geöffnet werden, so erhält das Annotation-Dictionary statt dem Key-Value-Paar `/Dest` ein sogenanntes *Action-Dictionary* als Wert des Schlüssels `/A`. Dieses Dictionary Object kann, den Schlüssel `/Type` besitzen. Ist er vorhanden muss er als Wert das Name Object `/Action` besitzen. Des Weiteren wird mit dem Schlüssel `/S` die Art der Aktion angegeben. Soll ein Uniform Resource Identifier (URI) geöffnet werden, so ist der Wert das Name Object `/URI`. Ist dies der Fall, wird die entsprechende URI in einem weiteren Key-Value-Paar mit dem Schlüssel `/URI` als String Object angegeben [20]. Somit stellt die zweite Annotation aus Quellcode 19 in Zeile 9-20 eine Schaltfläche da, welche beim Klick auf die untere Hälfte der Seite die Webseite www.example.com öffnet.

```

1 40 0 obj      % Interne Verlinkung
2 <<
3 /Type /Annotation
4 /Subtype /Link
5 /Rect [ 0 0 595 421 ]
6 /Dest [ 50 0 R /Fit ]
7 >>
8 endobj
9 41 0 obj      % Externe Verlinkung
10 <<
11 /Type /Annotation
12 /Subtype /Link
13 /Rect [ 0 421 595 842 ]
14 /A <<
```

```
15 /Type /Action  
16 /S /URI  
17 /URI (https://www.example.com)  
18 >>  
19 >>  
20 endobj
```

Quellcode 19: Zwei beispielhafte Annotations. Eine verlinkt intern, die andere extern.

Die Zuordnung, auf welcher Seite die Annotations platziert werden sollen, erfolgt im Page-Dictionary der entsprechenden Seite. Dieses wird dabei um das Key-Value-Paar mit dem Schlüssel **/Annots** erweitert. Als Wert besitzt er ein Array Object mit allen Annotations, welche auf der Seite ausgegeben werden sollen [20]. Die Seite, welche durch das in Quellcode 20 dargestellte Page-Dictionary beschrieben wird, verwendet somit die in Quellcode 19 definierten Annotations.

```
1 39 0 obj  
2 <<  
3 /Type /Page  
4 /Annots [ 40 0 R 41 0 R ]  
5 % [ ... ] weitere Key-Value-Paare der Seite  
6 >>  
7 endobj
```

Quellcode 20: Page-Dictionary, welches die Annotations aus Quellcode 19 besitzt.

3. Technologien und Möglichkeiten zur clientseitigen Generierung

Für die Beantwortung der Teilfrage, welche Technologien und Möglichkeiten zum Zeitpunkt dieser Arbeit für die Generierung von PDF-Dokumenten aus Webinhalten zur Verfügung stehen, werden diese im folgenden Kapitel sowohl vorgestellt als auch miteinander verglichen. Es ist anzumerken, dass auch serverseitige Lösungen existieren, diese aber aufgrund der Eingrenzung der Arbeit auf clientseitige Methoden nicht betrachtet werden. In Unterkapitel 3.1 wird die Generierung von PDF-Dokumenten durch den Druck-Dialog des Browsers betrachtet. Das Kapitel 3.2 stellt daraufhin die Generierung von PDF-Dokumenten unter Verwendung der JavaScript-Bibliothek *jsPDF* vor. Kapitel ?? befasst sich mit der Nutzung der Bibliothek *html2pdf*. Abschließend werden im letzten Unterkapitel 3.4 die verschiedenen Möglichkeiten miteinander verglichen und ein Fazit gezogen. Neben den Technologien und Möglichkeiten, welche in den Unterkapiteln vorgestellt werden, existieren noch weitere Alternativen, wie beispielsweise *pdfmake*¹², *pdf-lib*¹³ oder *PDFkit*¹⁴. Auch diese Bibliotheken ermöglichen es PDF-Dokumente mit JavaScript zu erstellen. Der Funktionsumfang dieser beschränkt sich jedoch auf Funktionalitäten zum Erstellen der Dokumente und das Platzieren von Schriften, Text, Bildern und Linien. Um mit diesen Bibliotheken aus einem HTML-Webinhalt ein PDF-Dokument zu erzeugen, müsste eine eigene Implementierung angestrebt werden. Daher werden auch diese Alternativen in dieser Arbeit nicht näher betrachtet.

Um einen einheitlichen Vergleich der Endresultate zu ermöglichen wurde eine einfache Webseite mit gängigen HTML-Elementen entwickelt. Dieser enthält unter anderem Überschriften, Bild-Elementen, interne und externe Verlinkungen durch Anchor-Elemente und Text. Die Gestaltung der Elemente ist einfach gehalten und an das Layout dieser Arbeit angelehnt. Eine solche Seite könnte beispielsweise einen Blogbeitrag oder Artikel darstellen, aber auch einen Ausschnitt einer wissenschaftlichen Arbeit als Webinhalt. Der Quellcode dieser Seite ist im Anhang B angegeben.

¹²GitHub-Repository der Bibliothek: <https://github.com/bpampuch/pdfmake>

¹³GitHub-Repository der Bibliothek: <https://github.com/Hopding/pdf-lib>

¹⁴GitHub-Repository der Bibliothek: <https://github.com/foliojs/pdfkit>

3.1. Druck-Dialog im Browser

Eine Möglichkeit, ein PDF-Dokument aus einem vorliegenden Webinhalt zu generieren, ist die Nutzung des Druck-Dialogs, welcher durch den Browser bereitgestellt wird. Dieser lässt sich auf zwei Arten öffnen. Zum einen, manuell durch den Nutzenden entweder über die Menü-Leiste der Anwendung (in der Regel: Datei -> Drucken) oder durch die Verwendung einer Tastenkombination¹⁵. Zum anderen kann er auch programmatisch über die JavaScript-Funktion `window.print()` geöffnet werden. Wurde der Druck-Dialog geöffnet kann, statt die Webseite auszudrucken, die Option ausgewählt werden, den Webinhalt als PDF-Dokument zu sichern.

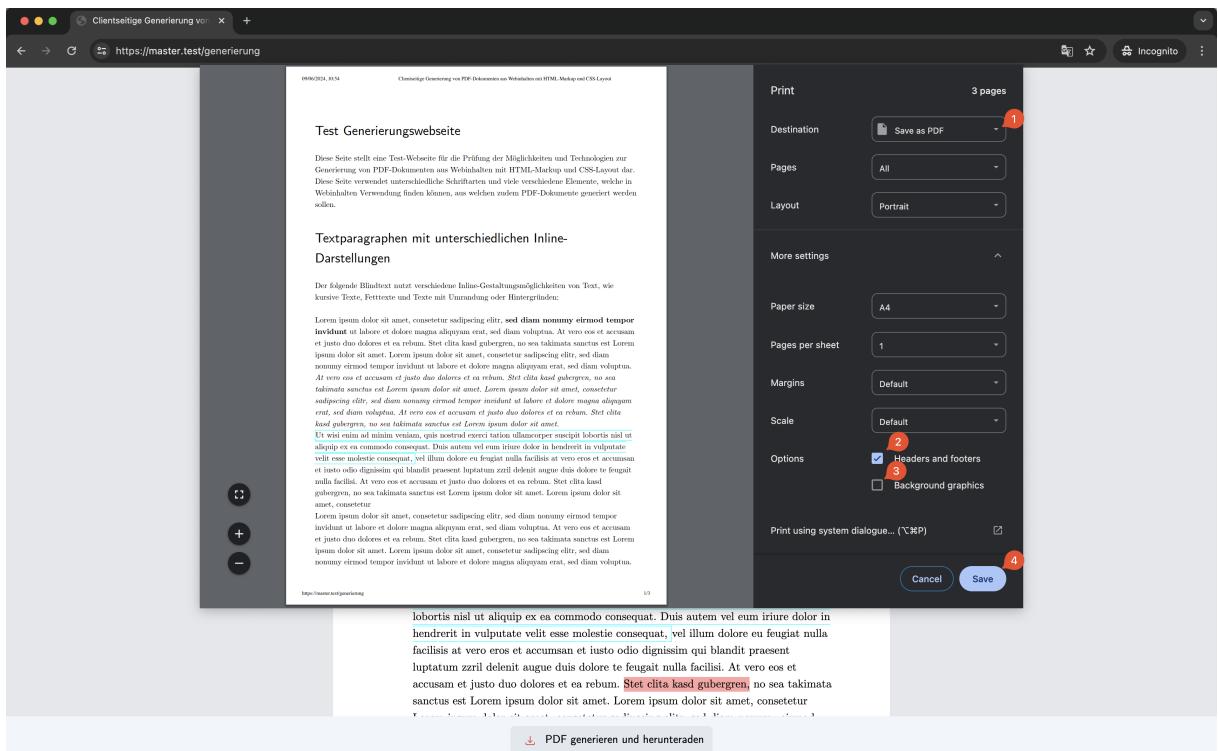


Abbildung 5: Screenshot des Druck-Dialogs im Browser Google Chrome. Eigene Darstellung.

In auf *Chromium* basierenden Browsern, wie *Google Chrome* oder *Arc*, muss dafür in der Einstellungsmöglichkeit *Ziel*, anstelle eines Druckers die Option *Als PDF speichern* ausgewählt werden (siehe Markierung 1 in Abbildung 5). Ist die Einstellung getroffen, kann das PDF-Dokument über die *Speichern*-Schaltfläche (siehe Markierung 4 in Abbildung 5) heruntergeladen und auf dem Computer gespeichert werden.

Standardmäßig ist dabei die Option *Kopf- und Fußzeile* aktiviert (Markierung 2 in Abbil-

¹⁵CMD+P unter MacOS beziehungsweise STRG+P in Windows.

3. Technologien und Möglichkeiten zur clientseitigen Generierung

dung 5). Dadurch wird zu jeder Seite sowohl eine Kopfzeile, bestehend aus dem aktuellen Datum, Uhrzeit und dem Webseiten-Titel (Inhalt des Title-Elements im HTML-Head), als auch eine Fußzeile mit dem Uniform Resource Locator (URL) der Webseite und der Seitenzahl hinzugefügt. Außerdem ist die Option Hintergrund-Grafiken (Markierung 3 in Abbildung 5) standardmäßig deaktiviert. Diese gibt an, dass Hintergründe von Elementen (wie beispielsweise Hintergrundfarben oder Hintergrundbilder), sofern vorhanden, nicht verwendet werden. Es ist davon auszugehen, dass diese Option für das eigentliche Drucken mit einem Drucker gedacht ist, um den Farbbedarf zu reduzieren, beispielsweise bei Webseiten mit farbigem Hintergrund. Dass die Einstellung standardmäßig deaktiviert ist, kann zu Problemen führen, falls der Webinhalt beispielsweise eine dunkle Hintergrundfarbe mit weißem Text verwendet. Durch die Entfernung der Hintergrundfarbe ist der Text in diesem Fall nicht mehr oder nur schwer lesbar.

Ähnlich verhält es sich bei anderen Browsern, wie beispielsweise *Safari*. Um den Webinhalt mit dem Druck-Dialog als PDF-Dokument zu speichern muss entweder auf die Schaltfläche *PDF* geklickt oder über den Pfeil *Speichern als PDF* (siehe Markierung 3 in Abbildung 6) ausgewählt werden. Auch in Safari sind dabei sowohl die Optionen für *Kopf- und Fußzeilen* als auch das Unterdrücken von Hintergründen standardmäßig eingestellt (siehe Markierungen 1 und 2 in Abbildung 6).

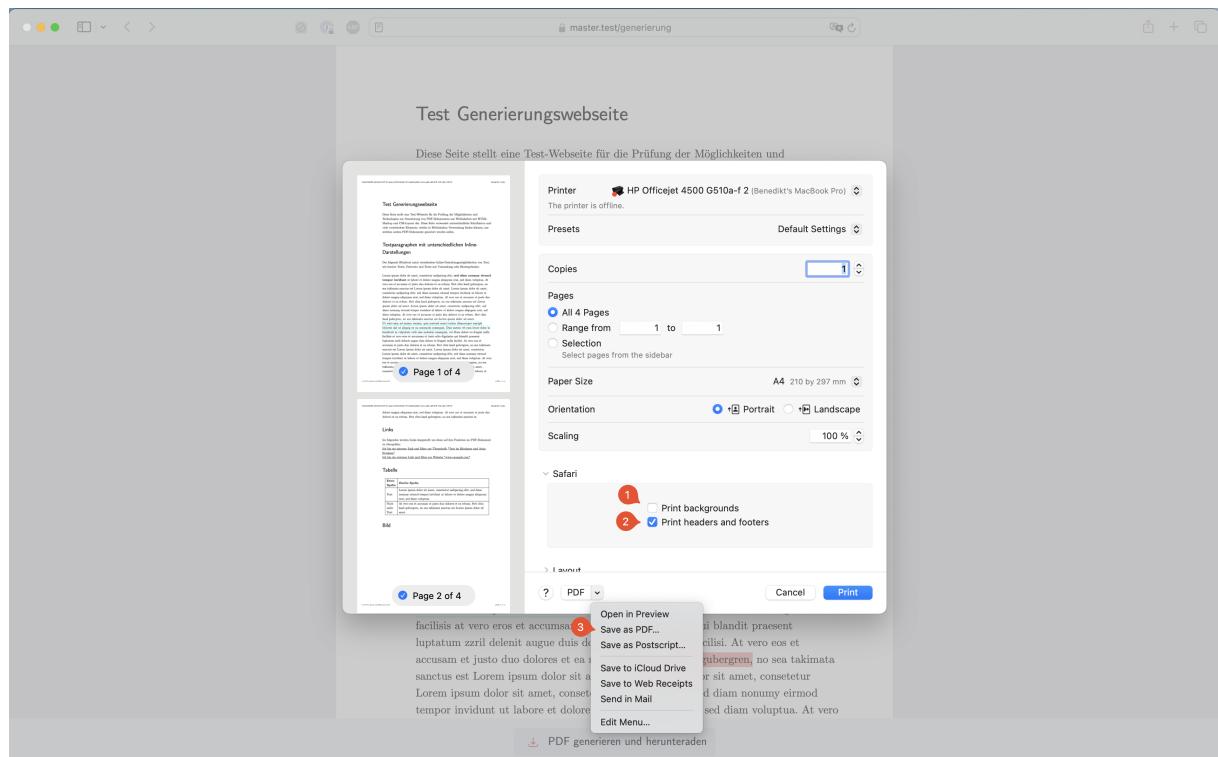


Abbildung 6: Screenshot des Druck-Dialogs im Browser Safari. Eigene Darstellung.

Nach dem erfolgreichen Abschließen des Druck-Dialogs erhält man als Endresultat ein PDF-Dokument, welches die Inhalte der Webseite enthält. Die Darstellung der Inhalte kann sich durch die Verwendung von Print-Stylesheets, beziehungsweise der Print Media-Query von der im Browser dargestellten Variante unterscheiden.

Wenn das PDF-Dokument¹⁶ mit einem PDF-Reader, wie *Acrobat Reader* oder *Apple Preview* geöffnet wird, lässt sich feststellen, dass Verlinkungen die auf der Webseite vorhanden waren auch weiterhin im PDF-Dokument eine Verlinkung darstellen. Handelt es sich bei einer Verlinkung um eine Verlinkung zu einem Element, welches ebenso in das resultierende PDF-Dokument überführt wurde, so verweist die Verlinkung innerhalb des Dokuments zur entsprechenden Stelle. Zudem lassen sich die Texte in dem PDF-Dokument auch weiterhin durch den Cursor markieren und kopieren, da diese weiterhin als Texte vorliegen. Wenn Texte oder Bilder auf einen Seitenumbruch fallen, werden diese unterteilt. Dabei bleibt der obere Teil auf der bisherigen Seite bestehen und der untere Teil wird auf die nächste Seite verschoben. Dabei kann es zu einem Informationsverlust kommen, da Text, wie in Abbildung 7, schwerer bis gar nicht mehr lesbar sein können oder ein Bild unterteilt wird. Um dies für Elemente zu vermeiden kann jedoch in den CSS-Regeln der Elemente die CSS-Eigenschaft **break-inside** mit dem Wert **avoid** verwendet werden.

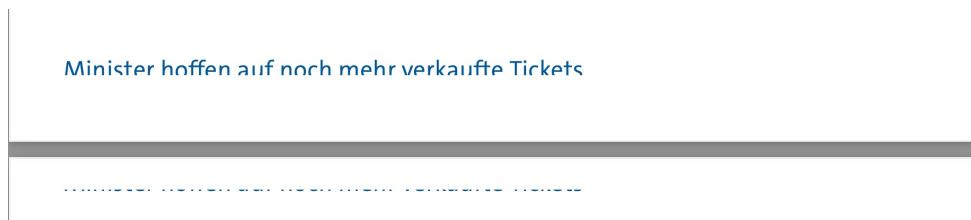


Abbildung 7: Screenshot eines Texts, welcher in der Mitte unterteilt wurde, da er auf dem Seitenumbruch liegt. Eigene Darstellung aus einem generierten PDF-Dokument von <https://www.tagesschau.de/inland/gesellschaft/deutschlandticket-preis-100.html>.

Eine weitere Problematik bildet die Tatsache, dass Webseiten in der Regel nicht nur auf Desktop-Geräten aufgerufen werden, sondern auch mobil auf Smartphones oder Tablets. Auch hier sind, je nach verwendetem Browser und Betriebssystem, verschiedene Ausgaben möglich. Beispielsweise fügt Safari unter iOS standardmäßig eine Fußzeile bestehend aus der Webseiten-URL, dem aktuellen Datum sowie Uhrzeit und einer Seitenzahl hinzu¹⁷. Es ist, wie in Abbildung 8 zu sehen, jedoch nicht möglich, die Fußzeile mithilfe einer

¹⁶PDF-Dokument findet sich im Anhang C.I.1 und mit angepassten Einstellungen in Anhang C.I.2

¹⁷PDF-Dokument findet sich im Anhang C.I.3

Einstellung zu entfernen. Google Chrome unter iOS¹⁸ hingegen fügt keine Fußzeile hinzu, entfernt jedoch teilweise Hintergründe und nutzt den selben von dem Betriebssystem bereitgestellten Druckdialog. Dieser ist in Abbildung 8 dargestellt. Auch das Entfernen der Hintergründe lässt sich nicht deaktivieren. Neben diesen fehlenden Einstellungsmöglichkeiten ist auch die Darstellung der Inhalte unterschiedliche. Je nach verwendetem Browser laufen Textzeilen unterschiedlich lang und es entstehen somit unterschiedliche Dokumente. Des Weiteren lässt sich feststellen, dass im Gegensatz zur Generierung durch den Druckdialog auf einem Computer, unter iOS Verlinkungen keine Verlinkungen mehr darstellen. Das Verhalten der Generierung unter der Verwendung von anderen Smartphone-Betriebssystemen und Browsern wurde nicht näher betrachtet. Es ist davon auszugehen, dass auch diese weitere Unterschiede in den Resultaten untereinander aufweisen.

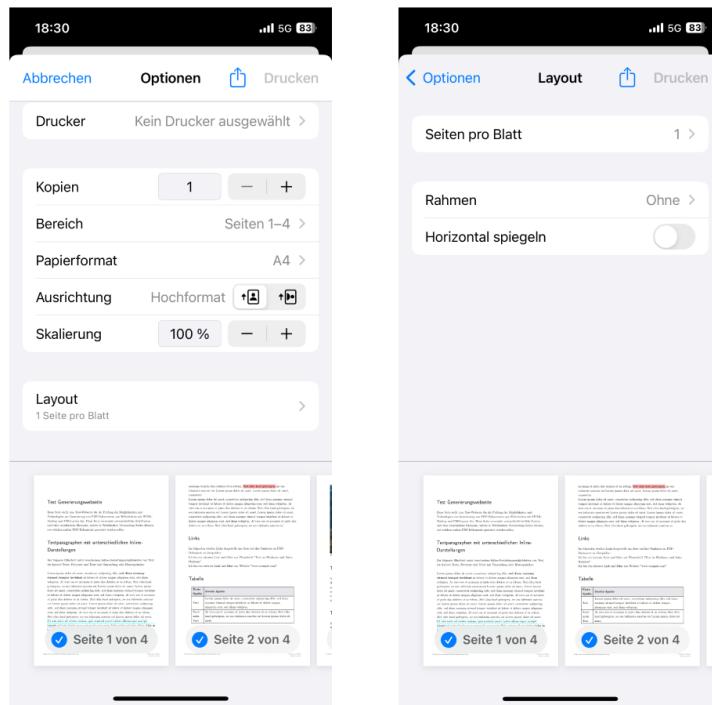


Abbildung 8: Screenshots des Druckdialogs unter iOS ohne weitere Einstellungsmöglichkeiten für die Verwendung von Hintergrundgrafiken und das Deaktivieren der Fußzeile. Eigene Darstellung.

Die fehlenden Einstellungsmöglichkeiten im Aufruf von `window.print()` wurden bereits im Mai 2022 in einem Issue im GitHub-Repository des HTML-Standards der *Web Hypertext Application Technology Working Group* (kurz: WHATWG) angemerkt und es wurde um eine Erweiterung der Funktionalität gebeten. In dem Issue wird auch ein Interface

¹⁸PDF-Dokument findet sich im Anhang C.I.4

beschrieben, welches die Funktion um ein Optionen-Objekt erweitern würde. Dieses Objekt würde unter anderem die Möglichkeiten beinhalten Seitenabstände, Seitenformat, die Verwendung von Kopf- und Fußzeilen und ähnliches zu steuern [22]. Das Issue ist zum Zeitpunkt dieser Arbeit noch nicht von der WHATWG beantwortet worden.

3.2. jsPDF

Die Open-Source zur Verfügung stehende Bibliothek *jsPDF* ermöglicht Entwickelnden das clientseitige Generieren von PDF-Dokumenten durch JavaScript. Die Bibliothek existiert seit 2010 und ist seit Januar 2022 in der Version 2.5.1 unter der MIT Lizenz verfügbar [23]. Über den Node Package Manager (NPM) wird sie wöchentlich etwa 1,19 Millionen Mal heruntergeladen [24].

jsPDF ist komplett in JavaScript geschrieben und hat dabei eine Dateigröße von 364 Kilobyte in der minimierten Version. Gewartet wird diese, laut der Readme-Datei des Repositorys, durch die britische *Parallax Agency Ltd.* und die deutsche Firma *yWorks GmbH*. Da das letzte Versionsupdate mehr als zwei Jahre zurückliegt und mehr als 50 Pull Requests noch nicht in den Main-Branch übertragen wurden [23] ist davon auszugehen, dass die Weiterentwicklung, zumindest vorerst, ausgesetzt wurde.

Neben Funktionalitäten zum Erstellen von PDF-Dokumenten über JS-Funktionen, wie das Hinzufügen von Seiten in verschiedenen Größen, das Zeichnen von Linien und das Platzieren von Bildern und Text in das PDF-Dokument, bietet *jsPDF* auch eine Funktion, um HTML-Webinhalte in ein PDF-Dokument zu überführen. Um diese Funktionalität zu nutzen, wird zusätzlich die Bibliothek *html2canvas* benötigt.

html2canvas wurde von Niklas von Hertzen entwickelt und steht seit Januar 2022 in der Version 1.4.1 zur Verfügung [25]. Durch die Verwendung der minimierten Version der Bibliothek kommen weitere 199 Kilobyte JS-Dateigröße hinzu. Somit werden 563 Kilobyte JavaScript für die Generierung benötigt.

Wenn beide Bibliotheken in die Webseite eingebunden sind, muss zunächst eine neue Instanz der Klasse *jsPDF* instanziert werden. Diese Objektinstanz bildet das zu erstellende Dokument ab. Bei der Instanziierung kann zusätzlich ein Objekt mit Optionen angegeben werden, welches beispielsweise die Seitengröße und Seitenausrichtung enthalten kann. Nach der Instanziierung muss über dieses Objekt die Methode `html()` aufgerufen werden. Als Übergabeparameter erhält diese das HTML-Element, dessen Inhalte in das PDF-Dokument übertragen werden sollen oder einen String, welcher das HTML enthält.

```

1 // Instanzierung eines neuen jsPDF-Objekts
2 const doc = new jsPDF();
3
4 // Generierung des PDF-Dokuments mit dem Dokumenten-Body als HTML-Input
5 doc.html(document.body, {
6   callback: function (doc) {
7     // Speichern/Herunterladen des erstellten PDF-Dokuments
8     doc.save("output.pdf");
9   }
10});

```

Quellcode 21: Funktionsaufruf zur Generierung eines PDF-Dokuments aus einem HTML-Webinhalt mit jsPDF. Übernommen aus: [26].

Betrachtet man das generierte PDF-Dokument¹⁹ mit einem PDF-Viewer stellt man fest, dass es sich bei Texten immer noch semantisch um Texte handelt, die markiert und kopiert werden können. Falls auf der Webseite für Texte eine andere Schriftart als die 14 Standard Type-1 Schriftarten verwendet wurde, werden diese nicht angewendet. Dies liegt daran, dass jsPDF die verwendeten Schriftarten und deren Quellen bei der Generierung nicht ausliest. Stattdessen muss ein Array von Objekten mit den verwendeten Schriftarten beim Aufruf übergeben werden. Dadurch werden diese in das PDF-Dokument eingebunden. Andernfalls werden die Schriften durch eine der Standard-Schriftarten ersetzt. Quellcode 22 zeigt die Erweiterung des Funktionsaufrufs, sodass die verwendeten Schriftarten auch im generierten PDF-Dokument verwendet werden²⁰.

```

1 // [...] Instanzierung des jsPDF-Objekts
2 doc.html(document.body, {
3   callback: function (doc) {
4     // Speichern/Herunterladen des erstellten PDF-Dokuments
5     doc.save("output.pdf");
6   }
7   fontFaces: [
8     {
9       family: "Custom Font",           // Font-Family der Schrift
10      style: "normal",                // Font-Style der Schrift
11      weight: "400",                  // Font-Weight der Schrift
12      stretch: "normal",              // Font-Stretch der Schrift

```

¹⁹PDF-Dokument findet sich im Anhang C.II.1

²⁰PDF-Dokument findet sich im Anhang C.II.2

```
13     src: [
14     {
15       url: "./assets/fonts/custom-font.ttf", // URL zur Schriftdatei
16       format: "truetype" // Format der Schriftdatei
17     }
18   ],
19   // [ ... ] Eventuelle weitere Schriften
20 ]
21 ]
22});
```

Quellcode 22: Erweiterter Funktionsaufruf zur Generierung eines PDF-Dokuments aus einem HTML-Webinhalt mit jsPDF und Verwendung von anderen Schriftarten.

Außerdem lässt sich feststellen, dass wie bei der Verwendung des Druck-Dialogs (siehe Kapitel 3.1) Texte und Bilder, die auf einen Seitenumbruch fallen, unterteilt werden. Dies kann für Texte, durch das Setzen der Option `autoPaging` mit dem Wert `"text"` verhindert werden. In diesem Fall wird der Text auf die nächste Seite verschoben²¹. In der Dokumentation wird angegeben, dass dieser Modus am Besten funktioniert, wenn nur eine Spalte Text existiert [26]. Es wurde jedoch festgestellt, dass auch bei der Nutzung dieser Option bei Webinhalten mit nur einer Spalte Text, Probleme auftreten. Der auftretende Fehler liegt in der Positionierung von Inline-Elementen, wenn Text auf die nächste Seite verschoben wird, sodass er nicht durch den Seitenumbruch unterteilt wird. Sobald dies der Fall ist, werden die Inline-Elemente auf der Seite vor dem Seitenumbruch tiefer positioniert als im Webinhalt. Dies ist im generierten PDF-Dokument bei der Bildunterschrift sichtbar. Dadurch, dass die erste Zeile des Texts im Blocksatz auf die nächste Seite verschoben wird werden die inline dargestellten Anchor-Elemente in der Bildunterschrift zu tief positioniert. Ein weiteres PDF-Dokument, welches diesen Fehler aufweist, findet sich in Anhang C.II.4. In diesem wird sichtbar, dass dieses Problem für jegliche Inline-Elemente gilt, welche in einem Paragraphen vorkommen können.

Ein ähnliches Fehlverhalten lässt sich bei der Nutzung von Hintergründen oder Umrandungen von Elementen feststellen. Diese werden, wie bei der Tabelle in Abbildung 9 zu sehen, unabhängig von der Einstellung `autoPaging`, immer höher als im Webinhalt positioniert.

²¹PDF-Dokument findet sich im Anhang C.II.3

Erste Spalte	Zweite Spalte
Text	Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua.
Noch mehr Text	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.

Abbildung 9: Screenshot der Tabelle aus dem PDF-Dokument in Anhang C.II.2, welche eine fehlerhafte Positionierung aufweist. Eigene Darstellung.

Zudem wurde festgestellt, dass Paragraphen, welche die CSS-Funktionalität `hyphens: auto` nutzen, fehlerhaft dargestellt werden. Dabei werden keine Bindestriche dargestellt, stattdessen wird das erste Zeichen der zweiten Zeile angezeigt. Ein weiteres Problem wurde bei Verlinkungen festgestellt. Bei diesen wird nur der Inhalt von vorhandenen Verlinkungen übertragen und somit sind sie im PDF-Dokument nicht mehr nutzbar. Die Gestaltung der Links, wie beispielsweise eine Unterstreichung oder die Verwendung einer anderen Farbe, werden dennoch in das PDF-Dokument übernommen, was zu Irritationen beim Lesenden führen kann.

Des Weiteren stellte sich heraus, dass das Layout der generierten PDF-Dokumente je nach verwendetem Endgerät abweichen²². Dies ist auf die Verwendung eines responsivem Designs zurückzuführen. Wird die Seite auf einem mobilen Gerät oder in einem Browser mit schmalem Viewport generiert, wird die Breite des übergebenen Elements auf die Seitenbreite skaliert. Es finden dabei weiter die mobilen CSS-Regeln Anwendung, was Unterschiede im Layout zur Folge hat.

3.3. `html2pdf.js`

Eine weitere JavaScript Bibliothek, welche es ermöglicht aus HTML-Webinhalten ein PDF-Dokument zu generieren, ist *html2pdf.js*. Diese Bibliothek steht bereits seit 3 Jahren in der aktuellen Version 0.10.1 zur Verfügung. Die Bibliothek wird von Erik Koopmans seit 2017 entwickelt. Genauso wie schon bei der Bibliothek *jsPDF* scheint es, dass die Bibliothek nicht aktiv gewartet und weiterentwickelt wird. Grund für die Annahme ist, dass zum Zeitpunkt dieser Arbeit über 30 Pull-Requests, welche teilweise mehrere Jahre alt sind, nicht mit dem Main-Branch zusammengeführt wurden [27]. *html2pdf.js* ist mit

²²PDF-Dokument findet sich im Anhang C.II.5

etwa 142.000 wöchentlichen Downloads laut npm nicht so verbreitet wie *jsPDF* [28]. Intern baut *html2pdf.js* auf der im vorherigen Kapitel 3.2 vorgestellten Bibliothek *jsPDF* auf und nutzt ebenso *html2canvas*. Hierbei wird von *html2canvas* die Version 1.0.0 und von *jsPDF* die Version 2.3.1 verwendet [29]. Die komplette Bibliothek kommt, mit allen benötigten abhängigen Bibliothek, dabei auf eine Größe von 906 KB in der minimierten Version.

Für die Generierung eines PDF-Dokuments aus dem Webinhalt muss die gleichnamige Funktion `html2pdf()` aufgerufen werden. Als Übergabeparameter muss auch hier, wie in Quellcode 23 dargestellt, das HTML-Element übergeben werden, welches in das PDF-Dokument überführt werden soll.

```
1 html2pdf(document.body);
```

Quellcode 23: Aufruf der PDF-Generierung für das Body-Element mittels *html2pdf.js*.

Führt man den Funktionsaufruf auf und betrachtet das generierte PDF-Dokument²³ mit einem PDF-Viewer ist festzustellen, dass Text und Bilder eine geringe Auflösung besitzen. Der Grund liegt darin, dass *html2pdf.js* alle Inhalte des übergebenen Elements mit *html2canvas* in ein Canvas-Element überführt und aus diesem Bilder in der Größe der PDF-Seite erstellt. *html2pdf.js* stellt die Möglichkeit bereit, neben dem HTML-Element noch ein Optionen-Objekt zu übergeben, um die Generierung anzupassen. Dies gilt auch für die Bild-Qualität. Mit der Angabe von `{image: {quality: 1}}` lässt sich die Qualität definieren²⁴, wobei 0 die geringste Auflösung und 1 die höchste darstellt. Standardmäßig ist dieser Wert mit 0.95 definiert [27]. Daher ist nur eine geringe Möglichkeit zur Verbesserung der Qualität gegeben.

Neben der Problematik bezogen auf die Auflösung der Inhalte birgt die Übertragung der Inhalte als Bild in das PDF-Dokument noch weitere Schwächen. Zum einen ist jeglicher Text nur noch Teil des Bildes und lässt sich somit weder markieren, kopieren, noch mit einer assistiven Technologie wie einem Screenreader erfassen.

Durch die Verwendung von Bildern weisen die generierten PDF-Dokumente eine hohe Dateigröße auf. Mit der Einstellungen der höchsten Qualität sind es für das resultierende Dokument etwa 2.2 MB bei der Verwendung von Bildern im JPEG-Format und 13.3 MB bei der Verwendung von Bildern im PNG-Format (Portable Network Graphics). Im Gegensatz hierzu weisen die Dokumente bei der Verwendung des Druckdialogs mit etwa

²³PDF-Dokument findet sich im Anhang C.III.1

²⁴PDF-Dokument findet sich im Anhang C.III.2

180 KB eine geringere Größe auf. Ebenso ist die Größe bei der Generierung mit *jsPDF* mit etwa 470 KB geringer.

Die Gestaltung und das Layout der Inhalte gleicht bei der Verwendung von *html2pdf.js* dem des Webinhalts. Jedoch ist festzustellen, dass auch in diesem Fall die Silbentrennung durch die CSS-Regel `hyphens: auto` nicht korrekt dargestellt wird. Dabei wird das zu trennende Wort nur in der zweiten Zeile dargestellt wodurch dieses sich mit dem weiteren Textinhalt überschneidet.

Des Weiteren wurde festgestellt, dass wenn auf der Webseite vor der Generierung nach unten gescrollt wurde und somit der `window.scrollY`-Wert größer als null ist, dieser nicht beachtet wird. Die Folge davon ist, dass das PDF-Dokument vor dem eigentlichen Inhalt einen leeren Abschnitt besitzt, welcher der Höhe des Scroll-Offsets entspricht. Zudem wird das Ende des Inhalts in gleicher Länge abgeschnitten²⁵.

Wenn der übertragene Webinhalt Verlinkungen beinhaltet, so sind diese auch im generierten PDF-Dokument vorzufinden. Handelte es sich um eine interne Verlinkung zu einem Element, welches ebenso in das PDF-Dokument übertragen wurde, verweist die Verlinkung, jedoch nicht an die Stelle des Elements in dem PDF-Dokument, sondern zurück auf die Webseite, aus welcher das Dokument generiert wurde. Wurde auf der Webseite vor der Generierung gescrollt, so werden die Annotations für die Verlinkungen an der selben Stelle positioniert wie wenn kein Scrollen stattgefunden hat. Für die Annotation wird somit der Scroll-Offset beachtet. Der eigentliche Inhalt, welcher die Verlinkung durch eventuell angewendete Styles darstellt, ist jedoch an einer anderen Position zu finden. Da die Verlinkung auch keine Umrandung besitzt, ist sie in diesem Fall nur durch die Veränderung des Cursors auffindbar.

Es ist außerdem anzumerken, dass *html2pdf.js* es erlaubt, eigene Seitenumbrüche zu definieren. Dafür können zum einen die CSS-Eigenschaften `break-before`, `break-after` und `break-inside` verwendet werden. Zum anderen kann auch im Optionen-Objekt für den Schlüssel `pagebreak` ein Objekt mitgegeben werden, welches Seitenumbrüche vor oder nach bestimmten CSS-Selektoren setzt beziehungsweise vermeidet [27]. Die Angabe durch das Optionen-Objekt wird in Quellcode 24 dargestellt. Dabei wird in Zeile 6 bis 9 definiert, dass vor dem HTML-Element `section` und dem Element mit der ID `myElement` ein Seitenumbruch stattfinden soll.

¹ `html2pdf(document.body, {`

²⁵PDF-Dokument findet sich im Anhang C.III.3

```

2   image: {
3     quality: 1
4   },
5   pagebreak: {
6     before: [
7       'section',
8       '#myElement'
9     ],
10    after: [
11      // Selektoren für Elemente nach welchen ein Seitenumbruch
12      stattfinden soll
13    ],
14    avoid: [
15      'img'
16    ]
17  }
18);

```

Quellcode 24: Erweiterter Aufruf der PDF-Generierung für das Body-Element mittels *html2pdf.js* mit gesetzten Optionen.

Ein weiteres Problem bei der Verwendung von *html2pdf.js* ist, dass wenn keine Seitenumbrüche angegeben werden die maximale Länge des zu transferierenden Inhalts beschränkt ist. Grund hierfür ist laut eines Issues, welches dieses Problem beschreibt, die maximale Höhe des Canvas-Elements, welches zur Erstellung der Bilder des Inhalts verwendet wird [30]. Die maximale Höhe variiert dabei je nach verwendetem Browser. Ist der Inhalt zu lang wird zwar ein PDF-Dokument generiert, dieses beinhaltet jedoch nur leere Seiten. Darüber hinaus tritt auch bei der Verwendung von *html2pdf.js* das Problem auf, dass sich das Layout des PDF-Dokuments je nach der für die Generierung verwendeten Browserbreite durch responsives Design unterscheidet²⁶. Auch in diesem Fall liegt der Grund darin, dass die Canvas-Breite auf die Seiten-Breite der PDF-Seiten gesetzt wird und die CSS-Regeln des verwendeten Viewports angewendet werden.

3.4. Vergleich und Fazit

Vergleicht man nun die in den vorherigen Unterkapiteln vorgestellten Möglichkeiten, lässt sich feststellen, dass alle Möglichkeiten nicht zu vernachlässigende Vor- und Nachteile

²⁶PDF-Dokument findet sich im Anhang C.III.4

aufweisen. Diese werden im Folgenden zusammenfassend vorgestellt.

Druckdialog Der Druck-Dialog erstellt ein qualitativ hochwertiges Resultat, dessen Layout sich nur in wenigen Ausnahmen vom vorliegenden Webinhalt unterscheidet. Dabei werden Texte so übertragen, dass diese auch in der PDF-Version Texte darstellen. Weitere Vorteile sind außerdem, dass keine weiteren Ressourcen nachgeladen werden müssen und Print-Stylesheets für Anpassung des Dokuments, wie das Ausblenden von Elementen, verwendet werden können. Den Vorteilen stehen jedoch die Nachteile der Verwendung des Druckdialogs gegenüber. Aufgrund des Konfigurationsaufwands auf Seiten des Nutzenden, sowie den Unterschieden in den generierten PDF-Dokumenten, welche sich aus den Einstellungsmöglichkeiten und der Verwendung unterschiedlicher Browser und Endgeräten ergeben, stellt diese Möglichkeit keine Alternative zu einem Download eines zuvor generierten PDF-Dokuments dar. Der Transfer der Verlinkungen, welche sowohl interne als auch externe Ziele haben können, stellt bei der Verwendung des Browsers auf einem Computer einen Vorteil dar. Da die Verlinkungen unter iOS jedoch nicht übertragen werden kann dies nicht als Vorteil gewertet werden und wird aus diesem Grund in der Gegenüberstellung in Tabelle 7 nicht aufgeführt.

Vorteile	Nachteile
Laden von zusätzlichen Skripten nicht erforderlich, da Funktionalität vorinstalliert	Ergebnis kann je nach verwendetem Browser und Endgerät unterschiedlich sein
Textinhalte stellen im PDF-Dokument auch Textinhalte dar, mit welchen der Nutzende interagieren kann	Notwendigkeit der Konfiguration durch den Webseiten-Nutzenden (Einstellungen für Kopf- und Fußzeilen, Hintergrund-Grafiken)
Anpassungen an das neue Format durch Print-Stylesheets möglich	
Kleinste resultierende Dateigröße des PDF-Dokuments im Vergleich zu den anderen Optionen	
Dokument wird unmittelbar beim Be-tätigen der Speichern-Schaltfläche gene-riert (keine Wartezeit)	

Tabelle 7: Gegenüberstellung der Vor- und Nachteile des Druckdialogs.

Die Gegenüberstellung zeigt, dass die Vorteile quantitativ überwiegen. Die unterschiedliche Qualität in Form von unterschiedlichen Darstellungen und nicht funktionierenden

Verlinkungen der resultierenden Dokumente überwiegt diese jedoch.

jsPDF Auch für die Verwendung von *jsPDF* sind die Vorteile zu benennen, dass Text aus dem Webinhalt auch in dem generierten PDF-Dokument Text darstellt. Ebenso ist positiv zu betonen, dass sich das Layout des PDF-Dokuments am Layout des Webinhalts orientiert. Dem entgegen steht jedoch die Tatsache, dass bei schmalen Viewports die Layout-Regeln für jene angewendet werden. Dadurch unterscheidet sich das generierte PDF-Dokument je nach verwendetem Endgerät. Des Weiteren stellt die teilweise fehlerhafte Positionierung, sowohl der Gestaltung von Elementen als auch von Inline-Elementen bei der Nutzung der Option `autoPaging` einen Nachteil dar. Dies gilt ebenso für die fehlende Übertragung von Verlinkungen in das PDF-Dokument und der fehlerhaften Darstellung bei der Verwendung von `hyphens: auto`.

Vorteile	Nachteile
Textinhalte stellen im PDF-Dokument auch Textinhalte dar	Schriften müssen separat definiert werden
Layout orientiert sich am angewendeten CSS-Layout	Verlinkungen werden nicht übernommen
Unterteilung von Text durch Seitenumbroch vermeidbar	Unterschiedliche Resultate bei Verwendung von responsivem Layout Teilweise fehlerhafte Positionierung von Elementen Angabe von gewollten Seitenumbrüchen nicht möglich Unterteilung von Bildern durch einen Seitenumbroch nicht deaktivierbar relativ große Bibliothek (906 KB)

Tabelle 8: Gegenüberstellung der Vor- und Nachteile der Nutzung von *jsPDF*.

Bei der Gegenüberstellung der Vor- und Nachteile für die Nutzung von *jsPDF* überwiegen die Nachteile (siehe Tabelle 8), da sich besonders die fehlerhafte Positionierung von Elementen auf die Nutzung auswirkt. Die separate Angabe der Schriften für die Übertragung stellt an sich zwar kein Nachteil dar, kann jedoch als dieser durch den zusätzlichen Konfigurationsaufwand durch die Entwickelnden gesehen werden.

html2pdf.js Für die Bibliothek *html2pdf.js* stellte sich positiv heraus, dass Verlinkungen übertragen werden. Weitere Vorteile stellen die Möglichkeit Seitenumbrüche manuell

zu definieren und das Vermeiden des Zerteilens von Elementen durch einen Seitenumbruch dar. Dem gegenüber stehen jedoch die Nachteile: Zum einen werden alle Inhalte nur als Bild in das PDF-Dokument übertragen. Dadurch lässt sich einerseits Text nicht mehr markieren und kopieren, noch durch assistive Technologien vorlesen. Andererseits wirkt sich dies auf die Dateigröße des generierten PDF-Dokuments aus. Unvorteilhaft sind desweiteren, dass Scrollen auf der Webseite vor der Generierung eine Verschiebung der Inhalte verursacht, wodurch Teile nicht korrekt dargestellt werden. Ebenso werden, wie schon bei *jsPDF* der Fall, bei schmalen Viewports und der Verwendung von responsivem Design andere CSS-Regeln für das Layout angewendet.

Vorteile	Nachteile
<p>Verlinkungen werden übertragen</p> <p>Manuelle Seitenumbrüche möglich</p> <p>Möglichkeit um Elemente von Unterteilung durch Seitenumbruch auszuschließen</p>	<p>Große Dateigröße, da alle Inhalte als Bild übertragen werden</p> <p>Textinhalte sind nur Teil eines Bilds</p> <p>Wenn Elemente größer als die maximale Canvas-Größe sind wird ein leeres PDF-Dokument erstellt</p> <p>Unterschiedliche Resultate bei Verwendung von responsivem Layout</p> <p>Scrollen des Webinhals führt zur Verschiebung der Inhalte im PDF-Dokument und Informationsverlust</p> <p>Verlinkungen, welche eine interne Verlinkung darstellen, verlinken aus dem PDF-Dokument auf die Webseite</p>

Tabelle 9: Gegenüberstellung der Vor- und Nachteile der Nutzung von *html2pdf.js*.

Auch die Gegenüberstellung der Vor- und Nachteile in Tabelle 9, welche bei der Nutzung der Bibliothek *html2pdf.js* ausgefunden wurden, überwiegen die Nachteile. Dies ist sowohl auf die Größe der resultierenden Dateien als auch die Tatsache, dass das PDF nur Bilder beinhaltet, zurückzuführen.

Die vorgestellten Möglichkeiten haben zudem gemein, dass sie keine Funktionalität bieten, ein Inhaltsverzeichnis in Form von Outlines anhand der verwendeten Überschrift-Elemente zu generieren. Solche Inhaltsverzeichnisse sind jedoch gerade bei längeren Dokumenten von hohem Nutzen für den Lesenden, da die Navigationsgeschwindigkeit innerhalb des

3. Technologien und Möglichkeiten zur clientseitigen Generierung

PDF-Dokuments deutlich erhöht wird. Darüber hinaus ist es nicht möglich, Elemente zu definieren, die Seitenzahlen ausgeben, die mit der Position eines anderen Elements korrespondieren oder die aktuelle Seitenzahl angeben. Eine solche Funktionalität würde die Möglichkeit bieten, ein Inhaltsverzeichnis mit Seitenzahlen zu erstellen, welches ebenso in das PDF-Dokument überführt werden kann. Außerdem ist es nicht möglich, Elemente zu definieren, welche als Kopf- oder Fußzeilen auf jeder Seite verwendet werden sollen. Solche Kopf- und Fußzeilen sind in druckbaren Dokumenten üblich, beispielsweise für die Angabe von Seitenzahlen oder den Kapitelnamen. Diese Elemente stellen gängige Komponenten von Dokumenten dar.

Somit lässt sich abschließend feststellen, dass keine der betrachteten Möglichkeiten und Technologien ein Resultat erzeugt, welches über verschiedene Browser und Endgeräte hinweg identisch ist. Zudem weisen alle Möglichkeiten einen Mangel an Funktionalitäten auf, von welchen die generierten PDF-Dokumente beziehungsweise dessen Lesende profitieren würden.

4. Konzeption der Bibliothek

Der Vergleich im vorherigen Kapitel zeigt, dass mit den zum Zeitpunkt dieser Arbeit verfügbaren Technologien und Möglichkeiten zur clientseitigen Generierung von PDF-Dokumenten aus einem Webinhalt keine Variante existiert, welche die Problemstellung aus der Einleitung ohne Einschränkungen löst. Daher wird in den folgenden Unterkapiteln die Konzeption für eine JavaScript-Bibliothek formuliert, welche die aufgefundenen Nachteile behebt und somit eine Lösung für die Problemstellung bietet. In Kapitel 4.1 wird dafür die Zielsetzung definiert. Darauf aufbauend wird in Kapitel 4.2 die Anforderungs-ermittlung an die Bibliothek vorgestellt und einige Kern-Anforderungen beschrieben. Abschließend stellt Kapitel 4.3 die Architekturentscheidungen und den Architekturentwurf für die Implementierung vor.

4.1. Zielsetzung

Die Zielsetzung lässt sich aus der Problemstellung ableiten. Somit soll eine Möglichkeit entwickelt werden, welche es ermöglicht clientseitig ein PDF-Dokument aus vorliegendem HTML-Markup zu generieren. Dabei soll der Nutzende bis auf das Starten des Generierungsprozesses oder aus dessen Sicht das Starten des Downloads, nicht weiter tätig werden müssen. Des Weiteren soll das Layout der Inhalte in dem PDF-Dokument so nahe wie möglich am Layout des Webinhalts sein. Die Formulierung „so nahe wie möglich“ bezieht sich darauf, dass durch das neue Ausgangsformat eine originalgetreue Übertragung unter anderem durch Seitenumbrüche nicht möglich ist. Des Weiteren soll das resultierende PDF-Dokument unabhängig von dem für die Generierung verwendetem Browser und Endgerät immer ein gleiches Ergebnis erstellen. Textinhalte sollen weiterhin als diese vorliegen und somit durch den Nutzenden des PDF-Dokuments markiert und kopiert werden können. Ebenso sollen Verlinkungen in das Dokument so übertragen werden, dass Verlinkungen zu Elementen, welche ebenfalls in das PDF-Dokument transferiert wurden, an die entsprechende Position dieser Elemente im PDF-Dokument verweisen. Verlinkungen zu externen Inhalten, beispielsweise zu anderen Webinhalten, sollen in diesem Fall ebenfalls externe Verlinkungen darstellen. Außerdem soll die Bibliothek den Entwickelnden, welche diese in ihr Webangebot integrieren, umfassende Möglichkeiten bieten die zu generierenden PDF-Dokumente anzupassen. Zu diesen Möglichkeiten zählen:

- Angabe von Seitenumbrüchen vor oder nach spezifizierten Elementen

- Angabe von Elementen, welche nicht transferiert werden sollen
- Definition eigener Kopf- und Fußzeilen durch HTML-Elemente und zugehörigem CSS-Layout
- Angabe von Seitenzahlen
- Abstände des Inhalts zu den Seitenrändern
- Erstellung der Document-Outlines anhand der im Webinhalt vorhandenen Überschriften

4.2. Anforderungsermittlung

Aus der zuvor definierten Zielsetzung lassen sich sowohl funktionale als auch nichtfunktionale Anforderungen an die neue Bibliothek ableiten. Durch die Verwendung der Anforderungen lässt sich sowohl während als auch nach der Implementierung die Zielerreichung überprüfen. Die an die Bibliothek gestellten Anforderungen wurden mithilfe der *FunktionsMAStEr-Schablone* der Sophisten definiert. Diese gibt den Aufbau von Anforderungen vor [31]. Wie von der Schablone empfohlen wurden zur Priorisierung die drei folgenden Schlüsselwörter verwendet:

Muss Anforderungen, welche dieses Schlüsselwort beinhalten, müssen für die Zielerreichung zwingend umgesetzt werden.

Sollte Dieses Schlüsselwort definiert Anforderungen, welche zur Zielerreichung nicht zwingend notwendig sind, jedoch Vorteile in der Nutzung oder weitere zusätzliche Funktionen bieten.

Wird Mit diesem Schlüsselwort werden Anforderungen definiert, welche nicht für die initiale Zielerreichung benötigt werden und welche erst in der Zukunft implementiert beziehungsweise erreicht werden.

4.2.1. Funktionale Anforderungen

Für die zu entwickelnde Bibliothek konnten im Rahmen dieser Arbeit 92 funktionale Anforderungen herausgearbeitet werden. Diese lassen sich in zwei Kategorien aufteilen. Zum einen in Anforderungen, welche sich an den Teilbereich des PDF-Writers richten, zum anderen in Anforderungen, welche den Transfer der als Webinhalt vorliegenden Daten definieren. Hierbei beschreiben die Anforderungen an den PDF-Writer alle Funktionalitäten, welche benötigt werden, dass das Endresultat der Generierung ein valides und

der PDF-Spezifikation entsprechendes PDF-Dokument darstellt. Analog dazu geben die Anforderungen an den Transfer vor, dass beispielsweise das Layout des resultierenden PDF-Dokuments dem Layout des Webinhals entspricht (siehe Anhang D.I).

Anforderungen an den PDF-Writer Auf die Komponente des PDF-Writers beziehen sich 50 der 92 funktionalen Anforderungen. Dabei beziehen sich die Anforderungen **F001** bis **F010** auf die Erstellung von PDF Objekten und geben an, dass diese sowohl als direkte als auch als indirekte Objekte ausgegeben werden können. Die weiteren sechs Anforderungen **F011** bis **F016** gehen darauf ein, dass die allgemeine Struktur von PDF-Dateien umgesetzt werden muss, sodass Header, Body, Cross-Reference-Table und Trailer korrekt platziert werden. Zudem wurden die weitere Anforderungen **F017** bis **F023** aufgestellt, welche das Setzen von Metadaten, wie dem Dokumententitel, Autor und anderen ermöglichen.

Darauf folgend spezifizieren die Anforderungen **F024** bis **F032** die Umsetzung von Verlinkungen in Form von Annotations, das Hinzufügen von neuen Seiten und dem Aufbau von Document Catalog und des Info Dictionarys. Die Anforderungen **F033** bis **F037** beschreiben, dass der PDF-Writer das Einbetten von Schriften im True Type Format (TTF) ermöglichen soll, sodass diese für Textinhalte verwendet werden können. Weitere Formate werden für die ersten Umsetzung durch die Verwendung des Wird-Schlüsselworts ausgeschlossen. Die Anforderung **F034** beschreibt die Möglichkeit, dass der PDF-Writer Subsets von Schriften ermöglichen wird. Von der Umsetzung dieser Anforderung wird auch die nicht-funktionale Anforderung **NF003** profitieren. Unter Subsets verstehen sich Schriftart-Dateien, welche lediglich die für die Darstellung des Texts benötigten Zeichen enthalten.

Für die Einbettung von Bildern wurden die Anforderungen **F038** bis **F042** aufgestellt. Diese sehen für die initiale Version nur die Bilder im JPEG-Format vor. Weitere Bildformate werden in den Anforderung angeführt, diese sind ebenso wie die Schriftarten durch die Verwendung des Schlüsselworts *wird* für die Zukunft geplant. Die weiteren Anforderungen Anforderungen **F043** bis **F050** gehen auf die Platzierung von Texten und Bildern auf den Seiten und das Erstellen der zugehörigen Resource-Dictionarys und ContentStreams ein.

Anforderungen an den Transfer Für den Teilbereich des Transfers eines Webinhals wurden weitere 42 Anforderungen aufgestellt. Dabei gehen die ersten fünf Anforderungen

F100 bis **F105** auf die Funktionalität der Erstellung der Document Outlines anhand der Überschriften des Webinhalts ein. Die Anforderungen **F106** bis **F109** beziehen sich auf die Funktion zur Ersetzung von Textinhalten von Elementen durch entsprechende Seitenzahlen. Dies ermöglicht unter anderem die in der Zielsetzung beschriebene Möglichkeit der Erstellung von Inhaltsverzeichnissen.

Die Anforderungen **F110** bis **F117** beziehen sich auf die Möglichkeit, dass Entwickelnde Elemente anhand von Klassennamen und Tagnames aus dem Transfer ausschließen können. Analog dazu wird zudem beschrieben, dass Elementen ebenso anhand von Klassennamen und Tagnames für die Generierung eingeblendet werden können. Diese Funktionalitäten ermöglichen es, dass der Webinhalt oder das PDF-Dokument zusätzliche Informationen beinhalten können. Hierfür wurde zudem spezifiziert, dass Elemente auch durch die Verwendung von spezifischen Attributen ignoriert beziehungsweise hinzugefügt werden können. Während das Ausblenden eine Muss-Anforderung darstellt, wurde für das zusätzliche Einblenden das Schlüsselwort soll verwendet und diese Funktionalität somit nicht für die initiale Umsetzung umgesetzt werden muss.

Auf diese folgen die weiteren sechs Anforderungen **F118** bis **F123**, welche sich auf mögliche Seitenumbrüche beziehen. Einerseits wird gefordert, dass durch die Angabe von Tagnames bestimmt werden kann, ob vor beziehungsweise nach den spezifizierten Elementen ein Seitenumbruch stattfinden soll. Auch für diese Funktionalität wird in Form von zwei Soll-Anforderungen angegeben, dass dies auch durch ein Attribut von Elementen ermöglicht werden soll. Zudem gibt Anforderung **F118** an, dass die Bibliothek automatisch eine neue Seite hinzufügen soll, sobald der zu übertragende Inhalt nicht mehr auf der aktuellen Seite platziert werden kann.

Des weiteren wird für den Generierungsprozess in den Anforderungen **F124** bis **F125** vorgeschrieben, dass der Nutzende auch während der Generierung weiterhin mit der Seite interagieren kann und kein Layoutshift durch das Aus- und Einblenden von Elementen für die Generierung stattfindet. Die Anforderungen **F133** bis **F134** beziehen sich auf die notwendigen Eigenschaften des Transfers. Dabei wird zum einen spezifiziert, dass der Text in der gleichen Darstellung in das PDF-Dokument übertragen werden soll wie er im Webinhalt auftritt. Zum anderen wird für alle Elemente spezifiziert, dass deren Gestaltung in das PDF-Dokument überführt werden soll.

Zudem beschreiben Anforderung **F135** und **F136**, dass ein gleiches Endresultat unabhängig vom verwendeten Endgerät und Browser ermöglicht wird und der Nutzende nicht

in den Generierungsprozess eingreifen muss.

Für die Funktionalität des Hinzufügens von Kopf- und Fußzeilen wurde in den Anforderungen spezifiziert, dass diese als Template-Elemente angegeben werden können und diese, wenn sie vorkommen, jeder neuen Seite hinzugefügt werden.

Des Weiteren wird in den Anforderungen **F138** bis **F140** auf den Transfer von Anchor-Elementen eingegangen. Für diese wird definiert, dass Verlinkungen innerhalb des Webinhalts, welche ebenso in das PDF-Dokument übertragen werden, auch innerhalb des Dokuments auf die entsprechende Seite verweisen sollen. Analog dazu wird angegeben, dass externe Verlinkungen zu anderen Webseiten oder zu Elementen, welche nicht übertragen wurden, die jeweilige URL öffnen sollen.

4.2.2. Nichtfunktionale Anforderungen

An die Bibliothek wurden drei nichtfunktionale Anforderungen aufgestellt (siehe Anhang D.II). Dabei gibt die Anforderungen **NF001** an, dass die Bibliothek selbst eine möglichst geringe Dateigröße aufweisen soll. Grund hierfür ist, dass die Ladezeit der zusätzlich benötigten JavaScript-Datei so gering wie möglich sein soll und die Funktionalität somit für den Endnutzenden schnell zur Verfügung steht. Als Richtwert für die Zielerreichung wird hierbei der Durchschnitt der Größen der beiden betrachteten Bibliotheken von 734,5 KB festgelegt.

Des Weiteren gibt die Anforderung **NF002** an, dass die Dauer, welche benötigt wird, um das PDF-Dokument zu generieren, möglichst gering sein soll. Grundlage dafür ist, dass davon auszugehen ist, dass die Benutzungserfahrung des Endnutzenden sich verbessert je geringer die Generierungsdauer ist. Da die Dauer unter anderem von der Anzahl der Elemente im zu transferierenden Webinhalt und der Internetverbindung für das Herunterladen von Schriftarten abhängig ist, wurde für die Zielerreichung eine Generierungsdauer von maximal 500ms für die in Kapitel 3 beschriebene Test-Webseite festgelegt.

Zudem wurde die dritte Anforderung **NF003** aufgestellt, welche angibt, dass neben der Bibliothek auch das resultierende PDF-Dokument eine möglichst geringe Dateigröße aufweisen soll. Auch hiervon profitiert der Endnutzende, da weniger Speicherplatz zum Sichern der Datei benötigt wird. Da auch die resultierende Dateigröße abhängig von dem vorliegenden Webinhalt durch die Anzahl von verwendeten Schriftarten und Bildern, sowie deren Auflösung ist, lässt sich auch für diese Anforderung nur schwer ein Wert für die Zielerreichung festlegen. Aus diesem Grund wird die Dateigröße des bei der Verwen-

dung von *jsPDF* generierten Dokuments von 578 KB als Richtwert für die Zielerreichung festgelegt.

4.3. Architekturentwurf

Anhand der Zielsetzung und der aufgestellten Anforderungen lässt sich die Funktionalität der clientseitigen Generierung von PDF-Dokumenten anhand von HTML-Markup und CSS-Layout in zwei Teilstrukturen unterteilen:

Zum einen wird die Funktionalität benötigt, welche ein PDF-Dokument erstellen kann, welche die PDF-Spezifikation erfüllt. Man spricht von einem *Conforming Writer*. Zum anderen wird die Funktionalität benötigt, welche das vorliegende HTML und angewendete CSS-Regeln ausliest und dem PDF-Writer zum Schreiben in die PDF-Datei übergibt.

Für alle Architekturentscheidungen wurden zur Dokumentation der Entscheidungen *Architectural Decision Records* (ADRs) aufgestellt. Als Vorlage für diese wurde in dieser Arbeit das *ADR-Template des Markdown Any Decision Records (MADR) Projekts* verwendet [32].

4.3.1. Programmiersprache der Bibliothek

Für die Architektur musste entschieden werden, welche Programmiersprachen für die Entwicklung der Bibliothek genutzt wird²⁷. Dabei wurden die Programmiersprachen *JavaScript (JS)* und *TypeScript (TS)* betrachtet.

Entschieden wurde sich für die Nutzung von TypeScript. Grundlage hierfür waren mehrere Faktoren. Einerseits profitiert die zu entwickelnde Bibliothek von der Typisierung. Dies ist auf die erwartbare Größe und den Umfang zurückzuführen. Andererseits ermöglicht Typescript eine einfachere Wartbarkeit und die Bibliothek wird durch die Typensicherheit fehlerresistenter. Zudem handelt es sich bei Typescript um ein Superset von Javascript. Dadurch lassen sich durch die Verwendung des TypeScript Compliers die TypeScript-Dateien in Javascript-Dateien kompilieren, wodurch diese auch im Browser ausgeführt werden können.

²⁷Der vollständige ADR findet sich im Anhang D.III - ADR 1

4.3.2. PDF-Writer

Ein weiterer ADR wurde für die Teilfunktionalität des PDF-Writers aufgestellt²⁸. Diese Funktionalität wird benötigt, um die Anforderungen **F001** bis **F050** zu erfüllen und somit die Erstellung der eigentlichen PDF-Datei und das Speichern als PDF-Dokument zu ermöglichen. Betrachtet wurden dabei einerseits die Nutzung der externen Bibliotheken *jsPDF* und *PDF-Lib*, sowie andererseits die Entwicklung einer eigenen Funktionalität zum Schreiben von PDF-Dateien.

In diesem Architectural Decision Record wurde sich für die eigene Implementierung der Funktionalität des PDF-Writers entschieden. Dies ist unter anderem auf die nichtfunktionale Anforderung zurückzuführen, dass die Bibliothek eine möglichst kleine Dateigröße aufweisen soll. Da die betrachteten Bibliotheken Funktionalitäten bereitstellen, wie beispielsweise Formulare, welche für die Generierung von PDF-Dokumenten aus einem Webinhalt nicht benötigt werden, ist davon auszugehen, dass eine eigene Implementation eine kleinere Dateigröße aufweisen wird. Außerdem lassen sich bei einer Eigenentwicklung die Funktionalitäten des PDF-Writers auf die Nutzung für den Transfer anpassen. Zudem hat der Entwickelnde ein umfassendes Verständnis der Funktionalitäten und deren Anwendung. Ein weiterer Punkt, welcher gegen die Verwendung der externen Bibliotheken spricht ist, dass diese nicht mehr aktiv weiterentwickelt und gewartet werden. Des Weiteren sprach gegen die Verwendung von *PDF-Lib*, dass keine Funktionalität für das Hinzufügen von Document Outlines und Verlinkungen besteht.

4.3.3. Bibliothek zur Verarbeitung von Schrift-Dateien (Font-Engine)

Resultierend aus der Entscheidung, die Funktionalität des PDF-Writers mit einer eigenen Implementierung abzubilden, wird eine Möglichkeit benötigt, um Schriftarten zu verarbeiten und Daten aus den zugehörigen Schriftdateien auslesen zu können. Grund dafür ist, dass diese Daten für die Font-Dictionarys der verwendeten und somit eingebundenen Schriftarten benötigt werden. Dafür wurden im Desicion Record²⁹ die beiden externen Bibliotheken *fontkit* und *font-reader* betrachtet.

Die Entscheidung fiel auf die Bibliothek *fontkit*. Grundlage hierfür war, dass die *font-reader*-Bibliothek einerseits lediglich Metadaten wie den Schriftnamen auslesen kann und andererseits nur auf Schriften im TrueType-Format anwendbar ist. Da für die Font-

²⁸Der vollständige ADR findet sich im Anhang D.III - ADR 2

²⁹Der vollständige ADR findet sich im Anhang D.III - ADR 3

Dictionarys Daten wie die Höhe von Ascent und Descent benötigt werden, stellte dies ein Ausschlusskriterium für die Bibliothek *font-reader* dar. *Fontkit* hingegen unterstützt eine Vielzahl von verschiedenen Schriftformaten und kann alle für die Erstellung der Font-Dictionarys benötigten Daten auslesen.

Zudem bietet die Verwendung von *fontkit* den Vorteil, dass die Bibliothek zusätzlich die Möglichkeit der Erstellung von *Font Subsets* unterstützt. Dabei handelt es sich um die Erstellung von einer neuen Schriftart-Datei, welche ausschließlich die benötigten Zeichen enthält. Eine solche Funktionalität wird für die Erfüllung der Wird-Anforderung **F034** benötigt. Die Umsetzung dieser Wird-Anforderung verbessert dabei auch das Ergebnis der nichtfunktionalen Anforderung **NF003**, da eine geringere Länge des Font-Streams auch eine geringe Dokumenten-Größe zur Folge hat.

4.3.4. Bibliothek zur Kompression (Compression-Engine)

Die Notwendigkeit einer Bibliothek für die Kompression resultiert aus der nichtfunktionalen Anforderung **NF003** und der Entscheidung aus dem ADR 2. Da die resultierende Dateigröße durch die Komprimierung des Bytestreams von Stream-Objekten sinkt, muss die Implementierung des PDF-Writers dies ermöglichen. Daher wurden im Architectural Decision Record die Nutzung von *node:zlib* und der Bibliothek *pako* betrachtet³⁰.

Dabei wurde sich für die Nutzung von *node:zlib* entschieden. Grundlage hierfür ist, dass die Nutzung von *node:zlib* dem Autor bereits bekannt war und diese sich einfach verwenden lässt. Des weiteren wurde zur Beginn der Umsetzung die Funktionalität des PDF-Writers in einer Node-Umgebung ausgeführt. Durch die Verwendung von *node:zlib* konnte diese ohne die Integration weiterer Module verwendet werden. Da Browserify als Bundler verwendet wird kann die *node:zlib* Implementierung auch im Browser verwendet werden, da diese mit in das Bundle übertragen wird.

Während der weiteren Implementierung wurde die weitere Bibliothek *fflate* aufgefunden. Auch diese bietet die Funktionalität der Komprimierung mit dem *zlib/deflate*-Algorithmus. Aufgrund dessen wurde eine weitere Betrachtung der Möglichkeiten in ADR 4.1 durchgeführt³¹. Dieser Decision Record löste damit seinen Vorgänger ADR 4 ab.

Durch die geringe Größe von 8 KB und dessen Einfluss auf die Nichtfunktionalen Anforderung **NF001** fiel die Entscheidung darauf, *fflate* anstelle von *node:zlib* zu verwenden.

³⁰Der vollständige ADR findet sich im Anhang D.III - ADR 4

³¹Der vollständige ADR findet sich im Anhang D.III - ADR 4.1

Durch diese Entscheidung konnte, bei gleichbleibendem Resultat der Generierung, die Dateigröße der Bibliothek in minimierter Form um 189 Kilobyte reduziert werden.

5. Umsetzung und Entwicklung der Bibliothek

In diesem Kapitel wird die Entwicklung und damit die Umsetzung der neuen Bibliothek zur Generierung von PDF-Dokumenten aus einem vorliegenden Webinhalt beschrieben. Da die Bibliothek aus einer Vielzahl von Funktionalitäten und Komponenten besteht, werden in den folgenden Unterkapiteln nur einige Aspekte davon vorgestellt.

In Kapitel 5.1 wird die Überführung der PDF-Objektsyntax und dem Aufbau von PDF-Dokumenten in eine objektorientierte Form in TypeScript vorgestellt. Zudem wird dabei die Funktionalität des Schreibens einer PDF-Datei aus dem PDF-Datenmodell beschrieben. Darauf folgend beschreibt Kapitel 5.2 den Ablauf, welcher den Transfer der HTML-Elemente und deren CSS-Layout in das PDF-Datenmodell durchführt. Dabei wird beantwortet, wie sich Inhalt und Layout von HTML-Elementen in ein PDF-Dokument überführt werden können. In Kapitel 5.3 wird außerdem die Funktionalität des Auslesens der benötigten Schriftarten aus der Webseite und der daraufhin stattfindende Transfer in das PDF-Dokument vorgestellt. In Kapitel 5.4 wird die Generierung der Document Outlines anhand der im Webinhalt vorhandenen Überschriftenhierarchie beschrieben. Das Kapitel 5.5 stellt außerdem die Implementierung für das Hinzufügen von Kopf- und Fußzeilen vor. Des Weiteren befasst sich das Unterkapitel 5.6 mit Komplikationen, die während der Implementierung aufgetreten sind. Diese werden sowohl vorgestellt, als auch die Lösungsansätze beschrieben. Abschließend wird im letzten Unterkapitel 5.7 die Nutzung der Bibliothek vorgestellt.

5.1. PDF-Datenmodell und PDF-Writer

Für die Implementierung des PDF-Datenmodells und die Implementierung des dazugehörigen PDF-Writers wurde ein objektorientierter Ansatz gewählt. Dadurch stehen für alle Objekttypen, das PDF-Dokument selbst, die Cross-Reference-Sections mit zugehörigen Subsections und weitere Komponenten von PDF-Dateien, Klassen zur Verfügung. Diese werden im Folgenden vorgestellt.

5.1.1. Klassen der PDF-Objekte

Für die verschiedenen PDF-Objekte wurde jeweils eine Klasse implementiert, welche dem Objekttypen entsprechend spezifische Eigenschaften und Methoden besitzt. Alle diese Klassen erben dabei sowohl Eigenschaften als auch Methoden von der übergeordneten

abstrakten Klasse `BaseObject`. Die Vererbung und entsprechende Eigenschaften und Methoden der Klassen sind dabei im Klassendiagramm in Abbildung 10 dargestellt. Durch die Vererbung besitzt jede Instanz einer Klasse mindestens die folgenden drei Eigenschaften:

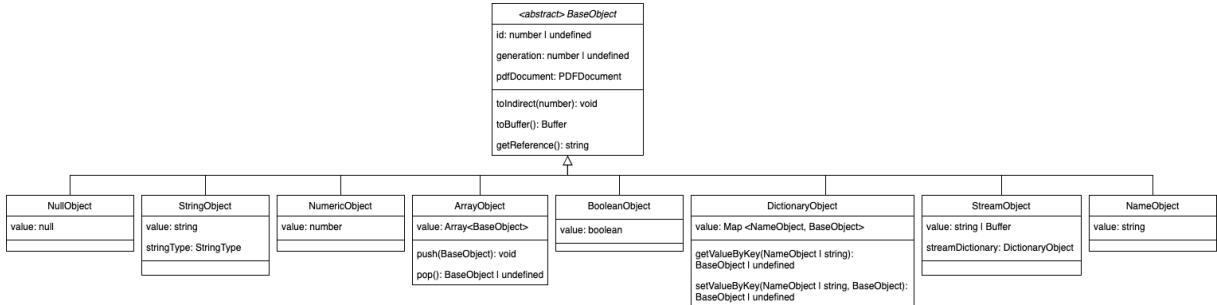


Abbildung 10: Klassendiagramm, welches die verschiedenen Objekttypen darstellt. Eigene Darstellung.

pdfDocument Diese Eigenschaft wird beim Aufruf des Konstruktors durch einen Übergabeparameter auf die Referenz zur zugehörigen Instanz der Klasse `pdfDocument` (siehe Kapitel 5.1.2) gesetzt. Diese Referenz ermöglicht unter anderem, dass Objekte in indirekte Objekte im Sinne des Portable Document Formats überführt werden können.

id Diese Eigenschaft ist standardmäßig `undefined`. Wenn die Instanz ein indirektes Objekt darstellt, wird in diesem Attribut die entsprechende Objekt-ID gespeichert.

generation Diese Eigenschaft ist ebenso standardmäßig `undefined`. Wenn das Objekt ein indirektes Objekt darstellt, so wird in dieser Eigenschaft die verwendete Generationsnummer gespeichert. Dadurch, dass die Bibliothek nur neue PDF-Dokumente erstellt, ist diese bei indirekten Objekten immer 0.

Dadurch, dass die Attribute `id` und `generation` standardmäßig `undefined` sind, stellen die Instanzen der Klassen immer direkte Objekte im Sinne der PDF-Spezifikation dar. Soll eine Instanz ein indirektes Objekt darstellen, so kann beim Aufruf des Konstruktors der Wert `true` für den Übergabeparameter `shouldBeIndirect` angegeben werden. Dadurch wird bereits während des Instanziierens die Methode `toIndirect()` aufgerufen. Diese nutzt die von der Instanz des zugehörigen PDF-Dokuments zur Verfügung gestellte Methode `addIndirectObject()`, um zum einem dem Dokument das Objekt als indirektes Objekt bekannt zu machen und zum anderen um eine eindeutige Objekt-ID zu erhalten.

Eine weitere Methode der Klassen, welche die PDF-Objekte darstellen, ist `toBuffer()`. Diese wird beim eigentlichen Schreiben der PDF-Datei für jedes Objekt aufgerufen und gibt den Wert der Objekte in der von der PDF-Spezifikation vorgegebenen Semantik zurück. Das bedeutet, dass beispielsweise indirekte Objekte von den Zeilen `<ObjektID>` `<Generationsnummer>` `obj` und `endobj` umschlossen werden und Name Objects den benötigten Schrägstrich vorangestellt bekommen.

Je nach Objekttyp stehen noch weitere Methoden zum Bearbeiten des eigentlichen Objektwerts zur Verfügung. Unter anderem ist das bei der Klasse `DictionaryObject` der Fall. Durch die Methode `getValueByKey()` lässt sich dabei der Wert eines Eintrags durch die Übergabe eines Name Objects oder Strings abfragen. Analog dazu lassen sich Wertepaare über die Methode `setValueByKey()` setzen und überschreiben.

5.1.2. Klasse PdfDocument

Die Klasse `pdfDocument` stellt den wichtigsten Teil des PDF-Writers dar. Jede Instanz dieser Klasse entspricht einem PDF-Dokument, welches erstellt werden soll. Beim Aufruf des Konstruktors können Optionen für das zu erstellende Dokument wie dessen Titel, den Autor und das Pagelayout angegeben werden. Der Konstruktor erstellt daraufhin anhand der übergebenen Werten den `DocumentCatalog`, den Wurzelknoten des Page-Trees und das Info Dictionary. Neben dem Konstruktor stellt die Klasse weitere Methoden bereit, welche beim Transfer des Webinhalts in das PDF-Dokument für das Erstellen von Seiten, Hinzufügen von Schriften und das Platzieren von Inhalten wie Text und Bilder genutzt werden. Im Weiteren werden der Aufbau und die Verwendung einiger Methoden vorgestellt:

Methode addPage() Diese Methode ermöglicht das Hinzufügen von Seiten an das Ende des Page-Trees. Hierbei kann lediglich die Seitengröße als Übergabeparameter angegeben werden. Wird keine Seitengröße angegeben, so wird standardmäßig die Din-A4-Größe im Hochformat (entspricht 595 x 842Point (pt)) verwendet. Zudem stellt die Bibliothek die Konstante `PageDimensions` bereit, welche eine vereinfachte Nutzung gängiger Formate ermöglicht, ohne die genauen Größen in Point angeben zu müssen (siehe Anhang E.I.1). Wie in Zeile 8 des Quellcodes 25 zu sehen, wird im derzeitigen Stand der Bibliothek immer der Wurzelknoten als Wert des Schlüssels `/Parent` genutzt. In einer weiteren Iteration der Bibliothek sollte dies angepasst werden, sodass durch die Verwendung von

Zwischenknoten der Page-Tree einen balanced Tree (zu deutsch: ausgeglichenen Baum) darstellt. Dabei handelt es sich um eine Empfehlung der Autoren der PDF-Spezifikation, sodass Dokumente bei der Betrachtung in einem PDF-Viewer performant bleiben [20]. Des Weiteren bietet die Methode keine Möglichkeit, Seiten an beliebigen Positionen hinzuzufügen. Grund hierfür ist, dass der implementierte PDF-Writer explizit für die Verwendung mit der zugehörigen Webinhalt-Transformation entwickelt wurde. Dieser transferiert die Inhalte durch deren Position im Webinhalt von oben nach unten, und fügt somit weitere Seiten nur an das Ende hinzu³². Dadurch besteht die Notwendigkeit dieser Möglichkeit nicht.

```

1 addPage(pageDimensions: [number, number] = PageDimensions.A4): void {
2   if (!this.pageTree) this.createPageTree();
3   const nextPage = new Page(
4     this,
5     new Map([
6       [NameObject.getName(this, 'MediaBox'), new Rectangle(this, 0, 0,
7         pageDimensions[0], pageDimensions[1])],
8       [NameObject.getName(this, 'Parent'), this.pageTree as
9        DictionaryObject],
10      [NameObject.getName(this, 'Type'), NameObject.getName(this, 'Page'
11        )],
12      [NameObject.getName(this, 'Resources'), new DictionaryObject(this)
13        ],
14      ]),
15      true,
16    );
17    const pageTreeKids = this.pageTree.getValueByKey('Kids');
18    if (pageTreeKids && pageTreeKids instanceof ArrayObject) {
19      pageTreeKids.push(nextPage);
20    } else {
21      this.pageTree.setValueByKey('Kids', new ArrayObject(this, [nextPage])
22        );
23    }
24    const count = this.pageTree.getValueByKey('Count');
25    if (count && count instanceof NumericObject) {
26      count.value += 1;
27    } else {
28      this.pageTree.setValueByKey('Count', new NumericObject(this, 1));
29    }
30  }
31}

```

³²Das Vorgehen der Transformation des Webinhalts wird in Kapitel 5.2 beschrieben.

```
24     }  
25 }
```

Quellcode 25: Methode der Klasse `pdfDocument` für das Hinzufügen eines neuen Page-Objects.

Methode addFont() Mit dieser Methode lassen sich neue Schriftarten der Instanz hinzufügen. Als Übergabeparameter erwartet die Methode die Schriftart-Datei in Form eines Buffers und einen String. Der String stellt dabei den Wert des Name Objects dar, mit welchem die Schriftart in ContentStreams verwendet wird. Die Methode erstellt ein neues leeres Font-Dictionary und speichert für die Schrift die Daten in Form eines Objekts im zur `pdfDocument`-Instanz gehörenden Array `includedFonts`. Grund dafür ist, dass beim Hinzufügen der Schrift nicht feststeht, ob diese überhaupt Verwendung findet. Deshalb werden die benötigten Werte-Paare des Font-Dictionarys und weitere Objekte erst beim Aufruf von `outputFileAsBuffer()` angelegt. Das Font-Dictionary muss angelegt werden, sodass eine Objekt-ID besteht, welche bei der Nutzung der Schriftart für das Resource Dictionary verwendet werden kann.

Methode addTextToPage() Um Textinhalte auf einer Seite zu platzieren, findet diese Methode Verwendung. Sie erwartet als Übergabeparameter mindestens die Koordinaten der Startposition, den zu platzierenden Text, den Namen der Schriftart und die Schriftgröße. Des Weiteren kann ein Seiten-Objekt oder die Seitennummer übergeben werden, auf welcher der Text platziert werden soll. Ist dies nicht der Fall, wird die letzte Seite verwendet. Ebenso kann ein Optionen-Objekt angegeben werden. Dieses kann weitere Eigenschaften wie die Textfarbe beinhalten.

Die Methode führt dabei folgenden Ablauf aus: Zu Beginn wird anhand des Namens die zu verwendende Schrift im Array `includedFonts` gesucht. Dem aufgefundenen zugehörigen Objekt werden daraufhin die verwendeten Buchstaben des Texts hinzugefügt. Falls die Textausrichtung nicht links ist, findet eine Berechnung der Textbreite statt. Nachdem dies erfolgt ist, wird das Seitenobjekt und dessen Resource Dictionary ausgelesen und die zu verwendende Schriftart diesem hinzugefügt. Bevor der Text nun dem ContentStream der Seite hinzugefügt werden kann wird jedes Zeichen des Texts in den Hexadezimalwert der Glyph-ID des Zeichens umgewandelt. Dadurch entsteht ein String in Hexadezimaler Schreibweise. Falls im Optionen-Objekt weitere Optionen wie die Textfarbe angegeben wurden, werden hierfür die Operatoren mit den zugehörigen Operanden als String er-

stellt. Gemeinsam mit den Operatoren zum Platzieren des Texts werden diese an das Ende des ContentStreams der Seite hinzugefügt.

Methode addImageToPage() Diese Methode platziert Bilder auf einer Seite des PDF-Dokuments. Als Übergabeparameter werden dabei die Position, die Bilddatei als Buffer, dessen Breite und Höhe, die Maße für die Platzierung und ein Bildname benötigt. Ebenso kann eine Seite angegeben werden, auf welcher das Bild platziert werden soll. Dies ist wie beim Hinzufügen von Text optional.

Sind alle benötigten Übergabeparameter gegeben, wird ein neues Stream Object für das Bild angelegt. Dabei werden die benötigten Key-Value-Paare des Bildes wie Höhe und Breite dem Image Dictionary hinzugefügt. Dieses wird daraufhin, ebenso wie beim Hinzufügen von Text, mit dem angegebenen Bildnamen dem Resource Dictionary hinzugefügt. Abschließend werden die benötigten Operatoren und Operanden sowohl für das Verschieben und Skalieren als auch das eigentliche Platzieren des Bildes dem ContentStream der Seite hinzugefügt.

Methode addLink() Für das Hinzufügen von Verlinkungen findet diese Methode Verwendung. Als Übergabeparameter erwartet sie alle für ein Annotation Dictionary benötigten Daten wie Position, Höhe, Breite und das Page Object der Seite, zu welcher die Annotation gehört. Zusätzlich kann ein Objekt an Optionen übergeben werden. In diesem können Optionen für eine mögliche Umrandung wie Farbe und Stärke angegeben werden. Obwohl diese Methode als privat deklariert ist, kann über die Methoden `addInternalLink()` und `addExternalLink()` diese außerhalb der Instanz verwendet werden. Dabei erstellt die Methode für interne Verlinkungen das Array, welches als Ziel benötigt wird. Analog dazu wird in der Methode für externe Verlinkungen das Action Dictionary anhand der übergebenen URL erstellt. Diese beiden Methoden werden beim Transfer des Webinhalts je nach Ziel für aufgefundene Anchor-Elemente verwendet.

Methode outputFileAsBuffer() Diese Methode stellt eine der Kern-Funktionalitäten der `pdfDocument`-Klasse dar. Sie ist für das Erstellen des Buffers zuständig, welcher den Inhalt der PDF-Datei darstellt. Vor dem eigentlichen erstellen des Buffers wird durch den Aufruf der privaten Methode `createDocumentOutline()` das Outline-Dictionary und zugehörige Outline-Item-Dictionarys erstellt³³. Daraufhin wird für alle Schriftarten, welche

³³Das genaue Vorgehen zur Erstellung der Document-Outlines wird in Kapitel 5.4 vorgestellt.

mit der Methode `addFont()` zur Instanz hinzugefügt wurden, überprüft, ob sie Verwendung finden. Dies erfolgt anhand des Arrays von benutzten Zeichen, welche der Schriftart zugeordnet sind. Ist das der Fall, werden mit der Funktion `addFontToDocument()` die für das Einbetten der Schriftart benötigten Objekte erstellt.

```

1 // [ ... ]
2 outputFileAsBuffer(): Buffer {
3     this.createDocumentOutline();
4     this.includedFonts.forEach((font, name) => {
5         if (font.usedChars.size === 0) {
6             this.indirectObjects.delete(font.fontDictionary.id!);
7         } else {
8             fontHelper.addFontToDocument(this, font);
9         }
10    });
11    const body = this.outputHeaderAndBody();
12    const bytesToStartxref = body.byteLength;
13    return Buffer.concat([
14        body,
15        Buffer.from(`#${this.createCrossReferenceTable()}`)trailer\r\n#${this.
16        trailer.toBuffer()}`)\r\nstartxref\r\n${bytesToStartxref}`\r\n%\EOF`),
17    ]);
18}
19// [ ... ]

```

Quellcode 26: Methode, welche die eigentliche PDF-Datei als Buffer zurückgibt.

Ist dies erfolgt, wird die eigentliche Erstellung des Buffers durchgeführt. Hierzu wird die private Methode `outputHeaderAndBody()` verwendet. Diese erstellt sowohl den Header der PDF-Datei anhand der spezifizierten Version als auch den Body (siehe Anhang E.I.3). Dabei werden alle in der Eigenschaft `indirectObjects` vorgehaltenen Objekte in einer Schleife durchlaufen. In dieser wird die Länge des bisherigen Buffers in Byte ausgelesen und gespeichert, da diese Information für die Erstellung der Cross-Reference-Table benötigt wird. Daraufhin wird die Methode `toBuffer()` aufgerufen und die Rückgabe an den bisherigen Buffer angehängt. Wenn alle Objekte im Buffer vorliegen wird die Cross-Reference-Table mit der Methode `createCrossReferenceTable()` erstellt und ebenso dem Buffer hinzugefügt. Abschließend wird, wie in der PDF-Spezifikation vorgegeben, der Trailer erstellt und an den Buffer angehängt. Der dabei erstellte Buffer wird zurückgegeben und kann daraufhin in eine Datei geschrieben werden (siehe Schritt 8 in Kapitel

5.2).

5.1.3. Klasse CrossReferenceSection und CrossReferenceSubSection

Für die Abbildung der Cross-Reference-Table im TypeScript-Datenmodell wurden zwei Klassen angelegt. Diese sind, wie die Klassennamen bereits aussagen, für die Datenhaltung der Cross-Reference-Sections und Cross-Reference-Subsections vorgesehen. Beim Instantiieren eines neuen Dokuments wird dabei in der Eigenschaft `crossReferenceTable` eine leere Instanz der Klasse `CrossReferenceSection` gespeichert. Diese wird erst, wie zuvor bereits beschrieben, beim Aufruf der Methode `createCrossReferenceTable()` während der Erstellung des Buffers genutzt. Dabei werden alle indirekten Objekte aus dem in der Eigenschaft `indirectObjects` gespeicherten Array in einer Schleife durchlaufen. In dieser werden die Objekte mit der Methode `addEntry()` der `CrossReferenceSection` hinzugefügt. Dabei wird, wie in Quellcode 27 zu sehen, überprüft ob eine Subsection vorhanden ist, welcher das Objekt am Anfang oder Ende hinzugefügt werden kann. Ist dies der Fall, so wird das Objekt der gefundenen Subsection, wie in Zeile 8 dargestellt, hinzugefügt. Andernfalls wird eine neue Subsection erstellt, welche als ersten Eintrag das Objekt angibt.

```

1 // [ ... ]
2 addEntry(id: number, byteOffset: number, generation: number, inUse =
3   true): void {
4   let section = this._sections.find((sec) => sec.firstId === id + 1 ||
5     sec.firstId + sec.entries.length === id);
6   if (!section) {
7     section = new CrossReferenceSubSection(this.pdf, id, byteOffset,
8       generation, inUse);
9     this.addSection(section);
10 } else {
11   section.addEntry(id, byteOffset, generation, inUse);
12 }
13 }
14 // [ ... ]

```

Quellcode 27: Methode `addEntry` der Klasse `CrossReferenceSection`.

5.1.4. Weitere Klassen

Neben den vorgestellten Klassen existieren noch weitere Klassen wie `Page` und `PageTree`. Diese Klassen erben von der Klasse `DictionaryObject` und setzen bereits bei der Aus-

führung des Konstruktors Key-Value-Paare, welche feste Werte besitzen. Dabei handelt es sich unter anderem um den Typen des Dictionary Objects oder ein leeres Dictionary Object unter dem Schlüssel `/Resources` für Seiten. Diese Klassen bieten zum Zeitpunkt dieser Arbeit jedoch keine weiteren Funktionalitäten.

5.2. Transfer der HTML-Elemente und zugehörigem CSS-Layout

Dieses Kapitel beschreibt den Ablauf des Transfers des Webinhalts in das PDF-Dokument. Dieser erfolgt in einer bestimmten Reihenfolge, sodass ein bestmögliches Ergebnis erzielt wird. Im Folgenden werden die dafür durchgeführten Schritte vorgestellt:

1. Verschieben des Inhalts in ein iFrame-Element Der erste Schritt ist die Verschiebung des Inhalts in ein separates iFrame-Element. Dies ist notwendig, um die Anforderungen **F124** und **F125** zu erfüllen. Diese Anforderungen geben einerseits vor, dass der Nutzende des Webinhalts während der Generierung weiterhin die Möglichkeit hat, mit diesem zu interagieren. Andererseits wird vorgegeben, dass es zu keiner sichtbaren Veränderung des Webinhalts kommt. Somit darf es nicht zu einem Layoutshift, also das Verschieben von Inhalten, kommen, wenn Elemente, welche nicht übertragen werden sollen, in einem späteren Schritt ausgeblendet werden. Zudem ist dies notwendig, sodass auch bei der Nutzung eines mobilen Endgeräts oder eines schmalen Viewports das Layout des PDF-Dokuments gleich bleibt.

Um dies zu ermöglichen, stellt die Bibliothek die Option `iFrameWidth` zur Verfügung. Mit dieser kann die optimale Vierport-Breite für ein gleichbleibendes Ergebnis angegeben werden. Es wird empfohlen, diese so zu wählen, dass das zu transferierende Element eine Breite besitzt, welche das Ergebnis von *Seitenbreite - Margin links - Margin rechts* ist. Wird keine iFrame-Breite in den Optionen definiert, so wird das Ergebnis der vorherigen Rechnung als Breite verwendet.

2. Ausblenden von Elementen In diesem Schritt werden alle Elemente ausgeblendet, welche beim Funktionsaufruf für die Generierung ausgeschlossen wurden. Dies erfolgt durch das Anwenden der CSS-Eigenschaft `display` mit dem Wert `none`. Dies hat zur Folge, dass die Elemente beim eigentlichen Transferieren ignoriert werden und sich alle anderen Elemente neu anordnen und leerer Weißraum verhindert wird.

3. Auslesen der Schriften Im dritten Schritt erfolgt das Auslesen aller Schriftarten, welche in den Stylesheets über `FontFace`-Regeln definiert wurden. Dabei wird ein Array an Schriften und zugehörigen URLs zu den Schriftdateien erstellt. Dieses wird bei der späteren Übertragung von Text-Inhalten verwendet, um die verwendeten Schriftarten zuordnen zu können und nur einmal in das PDF-Dokument übertragen zu müssen. Der Ablauf wird in Kapitel 5.3 vorgestellt.

4. Durchlaufen und hinzufügen aller Nodes Nachdem in den vorherigen Schritten die Voraussetzungen erfüllt wurden, folgt in diesem Schritt der Transfer der Inhalte. Dafür wird die Funktion `goThroughElements()` aufgerufen, startend mit dem bei dem Funktionsaufruf der Generierung angegebenen Element. Diese Funktion wird für alle Kind-Nodes des zu betrachtenden Elements rekursiv aufgerufen. Dabei finden für jede Node vorab einige Prüfungen statt. Als erstes wird geprüft, ob die Node ein Element-Node darstellt und ob diese ausgeblendet wird. Dafür werden die Werte der CSS-Eigenschaften `display` und `visibility` überprüft. Ist dies der Fall, werden das Element und dessen Kind-Elemente ausgelassen und die nächste Node überprüft. Des Weiteren wird geprüft, ob die Node das Attribut `data-htmlWebsite2pdf-pageNumberById` besitzt. Ist dies der Fall, wird das Element für die Bearbeitung am Ende des Transfers gespeichert und übersprungen (siehe Schritt 5). Eine weitere, letzte Untersuchung prüft, ob der Textinhalt des Elements durch die aktuelle Seitenzahl ersetzt werden soll. Ist dies der Fall wird dies ausgeführt. Nach diesen Überprüfungen wird die Node anhand ihres Typs weiterverarbeitet:

Verarbeitung von Element-Nodes: Für Nodes, welche HTML-Elemente darstellen, wird zuerst geprüft, ob neue Kopf- und Fußzeilen verwendet werden sollen. Der Prozess der dies durchführt, wird in Kapitel 5.5 beschrieben. Des Weiteren wird überprüft, ob zum einen vor dem Element ein Seitenumbruch stattfinden soll und zum anderen ob auf der Seite genug Platz für das Element vorhanden ist. In beiden Fällen wird falls notwendig eine neue Seite zur `pdfDocument`-Instanz hinzugefügt.

Anschließend findet die Übertragung des Layouts statt, welches beispielsweise Umrandungen und Hintergründe beinhaltet. Dies erfolgt in der Regel durch die Übertragung des Elements in ein SVG, welches über das Canvas-Element in ein Bild übertragen wird. Sowohl der Grund für dieses Vorgehen, als auch das Vorgehen selbst wird in Kapitel 5.6.5 beschrieben.

Nachdem die Gestaltung des Elements übertragen wurde, wird wenn das Element eine ID besitzt diese mit der aktuellen Seite in einem Array gespeichert. Dies ergibt sich daraus,

dass für das Hinzufügen der internen Verlinkungen in Schritt 6 die Seiten auf welcher diese platziert wurden benötigt werden. Anhand des Tagnames des Elements wird in einem Switch-Case-Block geprüft, ob für das Element weitere Funktionen ausgeführt werden müssen. Dies ist nur für `img`-Elemente, für das Hinzufügen und Platzieren der Bilder, Anchor-Elemente, für das Hinzufügen der Verlinkungen und Überschrift-Elemente, für die Generierung der Document Outlines, der Fall.

Abschließend wird wie bereits angeführt für die Kind-Nodes des betrachteten Elements erneut die Methode `goThroughElements()` ausgeführt.

Verarbeitung von Text-Nodes: Handelt es sich bei der betrachteten Node um eine Textnode, wird von dieser der Text zeilenweise ausgelesen, sodass dieser auf der Seite platziert werden kann³⁴. Des Weiteren wird aus den Computed Styles der Parent-Node die für den Text definierte Schriftart und Schriftstärke sowie der Schriftstil ausgelesen. Diese werden verwendet, um die entsprechende Schriftart aus den ausgelesenen Schriftarten zu finden und falls notwendig wird die gefundene Schrift der `pdfDocument`-Instanz hinzugefügt. Ist dies erfolgt, wird der Text der Zeilen wortweise mit der Methode `addTextToPage()` dem Dokument hinzugefügt.

5. Hinzufügen von übersprungenen Elementen Wurden im vorherigen Schritt Elemente aufgefunden, welche nicht direkt platziert werden konnten, so werden diese in diesem Schritt zur PDF-Datei hinzugefügt. Dies ist für Elemente der Fall, welche mit dem Data-Attribut `data-htmlWebsite2pdf-pageNumberById` gekennzeichnet sind. Mit diesem Attribut kann der Webseitenbetreibende eine ID angeben, für welche die Seitennummer, auf welcher das Element zu finden ist, ausgegeben wird. Da es sein kann, dass beim initialen Durchlaufen aller Elemente das entsprechende Element noch nicht übertragen wurde, werden diese erst in diesem Schritt hinzugefügt. Dabei wird falls notwendig der Inhalt durch die Seitennummer ausgetauscht und daraufhin für das Element erneut die Funktion `goThroughElements()` ausgeführt, um diese zu übertragen.

6. Hinzufügen von internen Verlinkungen Wurden alle Elemente in das PDF-Dokument übertragen werden den Seiten die Annotations für interne Verlinkungen hinzugefügt. Ebenso wie im letzten Schritt werden diese erst zum Ende des Transfers in das PDF-Dokument übertragen, um vorzubeugen, dass die Ziel-Elemente noch nicht übertragen wurden.

³⁴Der Ablauf des Auslesens wird in Kapitel 5.6.1 vorgestellt.

7. Entfernen des iFrame-Elements In diesem Schritt wird der Ausgangszustand des Webinhals wieder hergestellt, da der Transfer aller Elemente aus dem Webinhalt abgeschlossen ist. Dies erfolgt, indem das iFrame aus dem Document Object Model (DOM) entfernt wird.

8. Herunterladen des PDF-Dokuments Abschließend wird die Methode `downloadPdf()` aufgerufen. Diese ruft die bereits beschriebene, zur `pdfDocument`-Instanz gehörende Methode `outputFileAsBuffer()` auf. Aus dem zurückgegebenen Buffer wird daraufhin ein Blob und eine entsprechende Blob-URL erstellt. Für diese URL wird daraufhin ein Anchor-Element mit einem Download-Attribut erstellt und dieses programmatisch geklickt. Dies führt somit das Herunterladen des neu erstellten PDF-Dokuments aus. Eine Ausnahme bilden dabei Browser unter iOS und iPadOS, da diese durch JavaScript gestartete Downloads blockieren. Aus diesem Grund wird in diesem Fall alternativ die Blob-URL in einem neuen Tab geöffnet.

5.3. Transfer der Schriften aus dem Webinhalt in das PDF-Dokument

Wie schon im letzten Unterkapitel beschrieben findet vor dem eigentlichen Transfer des Webinhals das Auslesen aller Schriften statt. Dies erfolgt durch den Aufruf der Methode `getFontsOfWebsite()`. In dieser werden über die Schnittstelle `document.styleSheets` des DOMs alle Stylesheets des Webinhals ausgelesen. Nachdem Stylesheets, welche nicht geladen wurden oder keine CSS-Regeln beinhalten, herausgefiltert wurden, werden alle Regeln des Typen *CSSFontFaceRule* in einer Schleife durchlaufen. In dieser werden für jede Regel aus dem Source-Parameter die URLs und das dazugehörige Schriftformat der Schriftdateien ausgelesen. Des Weiteren werden der Schriftname, und falls angegeben die Schriftstärke, der Schriftstil und die Schriftdehnung ausgelesen. Da die Schriftstärke sowohl als String mit den Werten `bold` oder `normal` als auch als numerischer Wert angegeben werden kann, findet dabei falls notwendig eine Umformung statt. Dies liegt daran, dass die beim Transferieren ausgelesene Schriftstärke einzelner Elemente immer als numerischer Wert angegeben wird. Für jede Schrift werden die ausgelesenen Werte in ein Objekt überführt, welches den in Quellcode 28 dargestellten Aufbau besitzt. Diese Objekte werden in dem als Eigenschaft zur Generator-Instanz gehörigen Array gespeichert. Die zuvor beschriebene Vorgehensweise birgt jedoch eine Komplikation für das Auslesen

aller Schriften. Diese wird gesondert in Kapitel 5.6.4 behandelt.

```

1 {
2   fontFamily: '<FONT-NAME>',
3   fontStyle: '<FONT-STYLE>',
4   fontWeight: 400,
5   fontStretch: '',
6   name: '<FONT-NAME>-<FONT-WEIGHT>-<FONT-STYLE>', // String welcher als
    Name Object verwendet wird
7   inPDF: false, // Angabe ob die Schrift bereits dem PDF-Dokument
    hinzugefügt wurde
8   src: [ { url: '<URL-TO-FONTFILE>', format: '<FONT-FORMAT>' } ]
9 }
```

Quellcode 28: Aufbau eines Schrift-Objekts.

Wie bereits in Schritt 4 beschrieben wird vor dem Platzieren von Text aus diesem Array die zugehörige Schrift verwendet um den Text und gegebenenfalls die Schriftart dem PDF-Dokument hinzuzufügen.

5.4. Document Outlines anhand der Überschriften-Hierarchie

Die Generierung der Document Outlines für das resultierende PDF-Dokument stellt einen optionalen Mehrwert dar. Da diese bei kürzeren Dokumenten nicht benötigt oder vom Webseitenbetreibenden nicht erwünscht sein können, ermöglicht die Bibliothek, dies über eine Option zu steuern. Standardmäßig ist diese auf `false` gesetzt³⁵. Infolgedessen muss für die Generierung der Outlines im Funktionsaufruf der Schlüssel `outlineForHeadings` mit dem Wert `true` im Optionen-Objekt angegeben werden. Des Weiteren bietet die Bibliothek durch die Verwendung des Data-Attributs `data-htmlWebsite2pdf-no-outline` die Möglichkeit, einzelne Überschriften aus den Document Outlines auszuschließen.

```

1 // [ ... ]
2 case 'h1':
3 case 'h2':
4 case 'h3':
5 case 'h4':
6 case 'h5':
7 case 'h6':
```

³⁵Siehe Anhang E.II für mögliche Optionen und deren Standardwerte.

```

8   if (this.outlineForHeadings && element.el.dataset.
9     htmlwebsite2pdfNoOutline === undefined) {
10    await this.enoughSpaceOnPageForElement(element.el, element.rect.
11      bottom, element.rect.top);
12    const positionResult = this.findBookmarkPosition(this.bookmarks,
13      parseInt(element.el.tagName[1]));
14    positionResult.positions.pop();
15    this.bookmarks = positionResult.bookmarks;
16    this.pdf.addBookmark(element.el.innerText!.trim().replace(/\s+/g, '),
17      this.pdf.getCurrentPage(), positionResult.positions);
18  }
19  break;
20 // [ ... ]

```

Quellcode 29: Teil des Switch-Case-Statements für Überschriften bei der Überprüfung von ElementNodes.

Wie bereits beschrieben wird in der Funktion `handleElementNode()` in einem Switch-Case-Block anhand des Tagnames überprüft, ob für das betrachtete Element Besonderheiten in der Generierung vorliegen. Das ist durch die Document Outlines für Überschriften der Fall. Dabei wird wie in Quellcode 29 zu sehen in Zeile 8 zuerst überprüft, ob die Document Outlines erstellt werden sollen und das Element nicht ausgeschlossen wird. Ist dies der Fall, wird nach der Überprüfung, ob ausreichend Platz für das Element auf der Seite vorhanden ist, mit der Methode `findBookmarkPosition()` die Position im Zusammenhang mit allen anderen Überschriften ermittelt³⁶. Diese Funktion gibt unter anderem die Position der Überschrift im Verhältnis zu den schon betrachteten Überschriften in Form eines Arrays von Zahlen zurück. Wenn beispielsweise die Überschriften `h1`, `h2`, `h3`, `h2` bereits betrachtet wurden, besitzt das Position-Array für eine weitere h3-Überschrift den folgenden Inhalt: `[1, 2, 1]`. Die letzte Zahl wird in Zeile 11 aus diesem Array entfernt, da die Bookmarks in der Methode der `pdfDocument`-Instanz immer an das Ende der Ebene angehängt werden.

Wurde die Position ermittelt, wird die Methode `addBookmark()` der `pdfDocument`-Instanz aufgerufen. Dabei werden als Übergabeparameter die aktuelle PDF-Seite, der Textinhalt des Überschriften-Elements und das Postions-Array angegeben. Diese Funktion speichert daraufhin die Daten an der zugehörigen Stelle in der internen Repräsentation der Outlines der Instanz.

³⁶siehe im Anhang E.I.4

Erst beim Aufruf der Methode `outputFileAsBuffer()` wird vor dem eigentlichen Schreiben der PDF-Datei diese interne Repräsentation in das entsprechende Outline-Dictionary und die zugehörigen Outline-Item-Dictionarys überführt³⁷.

In dem zum Zeitpunkt dieser Arbeit vorliegenden Stand der Bibliothek sind bei der Erstellung der Document-Outlines noch keine weiteren Einstellungsmöglichkeiten vorhanden, wie sie in den Anforderungen **F103** bis **F105** beschrieben werden. Somit ist es nicht möglich, alternative Titel für die Outline-Items anzugeben und alle Ebenen mit Ausnahme der ersten Ebene werden zusammengeklappt dargestellt.

5.5. Hinzufügen von Kopf- und Fußzeilen

Sowohl die Zielsetzung in Kapitel 4.1 als auch die daraus resultierenden Anforderungen **F126** bis **F129** geben vor, dass die Bibliothek die Funktionalität bieten soll, Kopf- und Fußzeilen für Seiten in Form von HTML-Elementen und zugehörigen CSS-Regeln zu definieren. Um dies zu ermöglichen, stellt die Bibliothek den Nutzenden die Optionen `usePageHeaders` und `usePageFooters` zur Verfügung. Werden diese angegeben, so wird in Schritt 4 des Transfer-Prozesses beim Durchlaufen aller Elemente eine zusätzliche Überprüfung durchgeführt. Dabei wird das aktuell betrachtete Element auf ein direktes Kind-Element durchsucht, welches ein Template-Element ist und das Attribut `data-htmlWebsite2pdf-header` beziehungsweise `data-htmlWebsite2pdf-footer` besitzt. Wird ein Element gefunden, welches diesen Kriterien entspricht, so wird der HTML-Inhalt von diesem in einem neuen Div-Element an das für die Generierung genutzte iFrame angehängt. Durch das Hinzufügen des Elements wird dieses vom Browser gerendert, wodurch benötigte Daten wie Größe des Elements und angewendete CSS-Eigenschaften zur Verfügung stehen. Dieses neue Element wird für die spätere Verwendung in einem Array der aufgefundenen Kopf- und Fußzeilen gespeichert.

Beim Hinzufügen neuer Seiten mit der Methode `addPageToPdf()` werden die zuvor genannten Arrays dahingehend überprüft, ob diese Elemente beinhalten. Ist dies der Fall, so werden einerseits Werte von Variablen für die Positionsberechnung auf die zugehörige Werte desjenigen HTML-Elements gesetzt, welches als Kopf- oder Fußzeile verwendet werden soll. Andererseits wird die Methode `goThroughElements()` für das Element ausgeführt, welche den Inhalt des Elements in das PDF-Dokument transferiert. Wurden Kopf- beziehungsweise Fußzeilen gesetzt, wird daraufhin der verfügbare Bereich und dessen Po-

³⁷Die verwendeten Funktionen sind in Anhang E.I.5 abgebildet.

sition für den eigentlichen Seiteninhalt neu berechnet.

Wurde das letzte Kind-Element des Elements, welches Kopf- beziehungsweise Fußzeilen besitzt, transferiert, wird die Zeile aus dem iFrame-Element und dem Array an Kopf- und Fußzeilen entfernt. Die Nutzung eines Array ermöglicht hierbei, dass zwischenzeitlich andere Kopf- und Fußzeilen verwendet werden können und danach wieder die vorherigen Zeilen angewendet werden.

Da die Kopf- und Fußzeilen direkt beim Hinzufügen einer weiteren Seite auf dieser platziert werden, resultieren einige Einschränkungen für die Verwendung von Elementen und Ersetzungen durch die Bibliothek. Zum einen kann das Attribut `data-htmlWebsite2pdf-pageNumberById` nicht verwendet werden. Für Elemente mit diesem Attribut wird der Text-Inhalt durch die Seitenzahl, auf welcher sich das Zielelement befindet, ersetzt. Da Elemente mit dieser Ersetzung erst in Schritt 5, wenn alle Elemente transferiert sind, in das Dokument transferiert werden, ist dies für Kopf- und Fußzeilen nicht möglich. Außerdem können keine Anchor-Elemente verwendet werden, welche als Ziel eine interne Verlinkung aufweisen. Grund hierfür ist, dass die Position von internen Verlinkungen in Schritt 6 anhand der Position des Elements im Webinhalt und dem der Seite zugeordneten Ausschnitt des Webinhalts (durch Start und Ende auf der Y-Achse) berechnet wird. Da die HTML-Elemente, welche für die Kopf- beziehungsweise Fußzeile verwendet werden sollen, sich nicht im eigentlich übertragenen Inhalt befinden, kann diese Berechnung nicht stattfinden. Zudem sollte vermieden werden, dass Elemente in der Kopf- oder Fußzeile eine ID besitzen, welche als Ziel einer internen Verlinkung verwendet wird. Grund hierfür ist, dass Kopf- und Fußzeilen in der Regel auf mehreren Seiten verwendet werden und in diesem Fall nur zur ersten Seite verlinkt wird, auf welcher diese Zeile verwendet wird. Eine weitere Einschränkung bezieht sich auf die Nutzung von Überschriften, wenn die Option der Erstellung der Document-Outlines aktiviert ist. Hierdurch würde für jede Kopf- beziehungsweise Fußzeile ein Element zu den Document-Outlines hinzugefügt werden. Um Überschriften-Elemente trotzdem zu verwenden, sollte in diesem Fall das Element durch die Verwendung des Data-Attributs `data-htmlWebsite2pdf-no-outline` von den Document-Outlines ausgeschlossen werden.

5.6. Komplikationen bei der Umsetzung

Während der Implementierung der Bibliothek kam es zu einigen Komplikationen, welche gelöst werden mussten. Im nun Folgenden wird auf einige dieser eingegangen und die

angewendeten Lösungsiterationen und ihre etwaigen Nachteile vorgestellt.

5.6.1. Auslesen der Texte

Das Auslesen von Textinhalten stellte an sich kein Problem dar, da sich dieser über die Eigenschaft `textContent` des Node-Interfaces des DOMs auslesen lässt. Dieser beinhaltet neben dem dargestellten Text zusätzlich die im HTML-Markup verwendeten Whitespaces und Zeilenumbrüche. Durch die Anwendung des regulären Ausdrucks `/[\n\t\r\s]/g` kann dieser jedoch entfernt werden. Für das korrekte Platzieren des Texts im PDF-Dokument ist dieser zurückgegebene Text jedoch unzureichend, da die Verwendung der Text-Operatoren nicht automatisch Zeilenumbrüche ausführt. Würde der aus einer Text-Node ausgelesene Text als Operand für den `Tj`-Operator angegeben werden, so würde der komplette Text in einer Zeile auf der Seite platziert werden. Aus diesem Grund musste eine Lösung gefunden werden, mit welcher sich der Text auf die dargestellten Zeilen aufteilen lässt. Die Lösung, welche in einer ersten Umsetzung angestrebt wurde, war die Nutzung des *DOM Range Interfaces*. Dabei wurde mit der Methode `selectNodeContents()` der Bereich der Range auf den zu verarbeitenden Inhalt der Textnode gesetzt. Durch den Aufruf der Methode `getClientRects()` konnten daraufhin die Rechtecke, welche der Inhalt aufspannt, ausgelesen werden. Dies ist dargestellt in Zeile 9 bis 11 des Quellcodes 30. Diese Rechtecke stellen dabei die einzelnen Zeilen dar. In einem weiteren Schritt mussten daraufhin die Wörter ausgelesen werden, welche sich in diesen Rechtecken befinden. Hierfür wurde der Textinhalt der Node anhand der vorhandenen Leerzeichen und Bindestriche in einzelne Wörter unterteilt (Zeile 30-36). Daraufhin wurde der Textinhalt des Elements auf das erste Wort gesetzt und solange weitere Worte hinzugefügt, bis die Höhe des Elements sich verändert. Dies zeigte an, dass eine weitere Zeile hinzugekommen ist.

```

1 / [ ... ]
2 function getTextNodes(node: Node): Array<any> {
3   const nodes: Array<TTTextNodeData> = [];
4   if (window.getComputedStyle(node.parentNode as Element).display !== 'none') {
5     if (node.nodeType === Node.TEXT_NODE) {
6       if (node.nodeValue && node.nodeValue.trim() === '') {
7         return nodes;
8       } else {
9         let range = document.createRange();
10        range.selectNodeContents(node);

```

```

11     let rects = range.getClientRects();
12     nodes.push({
13         text: node.nodeValue!.trim(),
14         styles: window.getComputedStyle(node.parentNode as Element),
15         position: node.parentNode!.getBoundingClientRect(),
16         lines: splitTextFromElement(node.parentNode!) ,
17     });
18 }
19 } else {
20     for (let child of Array.from(node.childNodes)) {
21         nodes.push(...getTextNodes(child));
22     }
23 }
24 }
25 return nodes;
26 }
27
28 function splitTextFromElement(el: HTMLElement): Array<string> {
29     let wordBreak = [];
30     const words = el.innerText.split(' ');
31     words.forEach((word, idx) => {
32         if (word.includes('-')) {
33             let splittedWord = word.split('-');
34             words.splice(idx, 1, `${splittedWord[0]}-`, splittedWord[1]);
35         }
36     });
37     el.innerText = words[0];
38     let beforeBreak = words[0];
39     let height = el.offsetHeight;
40
41     for (let i = 1; i < words.length; i++) {
42         el.innerText += ' ' + words[i];
43         if (el.offsetHeight > height) {
44             height = el.offsetHeight;
45             wordBreak.push(beforeBreak);
46             beforeBreak = words[i];
47         } else {
48             if (beforeBreak.charAt(beforeBreak.length - 1) === '-') {
49                 beforeBreak += words[i];
50             } else {

```

```
51         beforeBreak += ' ' + words[i];
52     }
53 }
54 }
55 wordBreak.push(beforeBreak);
56 return wordBreak;
57 }
58 // [ ... ]
```

Quellcode 30: Erste Umsetzung für das zeilenweise Auslesen der Textinhalte.

Dieser Ansatz bildete eine Lösung für Textinhalte, welche im Flattersatz dargestellt werden. Textinhalte, welche durch die CSS-Eigenschaft `text-align` mit dem Wert `justify` im Blocksatz dargestellt werden, wurden in diesem Fall jedoch ebenfalls als Flattersatz in das PDF-Dokument übertragen. Dieses Problem wurde in einer weiteren Iteration des zeilenweisen Auslesens angegangen. Der Lösungsansatz bestand darin, mit einer Range die Breite der Wörter auszulesen, diese zu addieren und von der Gesamtbreite der Zeile abzuziehen. Die Differenz wurde daraufhin durch die Anzahl der Leerzeichen geteilt. Das Ergebnis stellte dabei die Breite eines Leerzeichens dar. Daraufhin wurde jedes Wort einzeln in die PDF-Datei übertragen. Die Position auf der x-Achse berechnete sich dabei wie folgt:

X-Position des Zeilenbeginns + Breiten der vorherigen Worte + (Anzahl der vorherigen Worte * Breite eines Leerzeichens)

At vero eos et accusam et justo duodolores 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. Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed

Abbildung 11: Screenshot eines Ausschnitts einer PDF-Seite mit der implementierten Iteration für Blocksatz. Eigene Darstellung.

Dieser Lösungsansatz besaß jedoch den Nachteil, wie durch die Textmarkierung in Abbildung 11 zu sehen, dass die Zeilen nicht genau an derselben Position endeten. Dies ist darauf zurückzuführen, dass jede Zeile mit einem Leerzeichen endet, welches je nach Zeile eine unterschiedliche Breite besitzt. In der zweiten Iteration des zeilenweisen Auslesen wurde dies gelöst, indem über die `setStart()`-Funktionalität des Range-Interfaces

der Start des zu betrachtenden Inhalts auf das erste Zeichen des Worts und das Ende auf das letzte Zeichen des Worts eingestellt wurde. Anhand des Rechtecks, welches das Wort beinhaltet, kann daraufhin die genaue Startposition von jedem Wort bestimmt werden. Anhand dieser Position werden daraufhin die Wörter weiterhin einzeln in das PDF-Dokument übertragen.

Das letzte Problem, welches weiterhin bestand, war die Nutzung von Wortumbrüchen durch die CSS-Eigenschaft `hyphens: auto`. Durch die Verwendung dieser CSS-Eigenschaft werden Wörter automatisch an einem Zeilenende mit einem Bindestrich getrennt. Da der Textinhalt der Textnode diesen Bindestrich nicht enthält und somit die Wortteile nicht einzeln betrachtet werden, resultiert dies in dem in Abbildung 12 dargestellten Ergebnis.

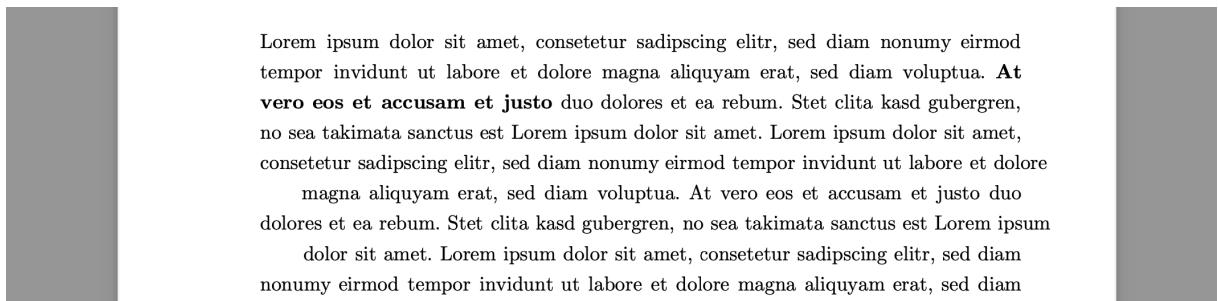


Abbildung 12: Screenshot eines Ausschnitts einer PDF-Seite mit der zweiten Iteration für Blocksatz und fehlerhaften Wortumbrüchen. Eigene Darstellung.

Dies konnte in einer dritten Iteration behoben werden. In dieser wurde eine Überprüfung der Rückgabe der Methode `getClientRects()` implementiert. Wenn diese Rückgabe ein Array mit mehreren Rechtecken darstellt, musste es sich um einen Zeilenumbruch handeln. Dies ist darauf zurückzuführen, dass jeweils ein Rechteck für den Teil in der ersten, ein weiteres für den Teil in der zweiten Zeile und ein Rechteck für den Bindestrich zurückgegeben wird. Um nun die Position des Zeilenumbruchs festzustellen, wird eine Schleife durchlaufen, welche die Rechtecke für eine Range überprüft. Der Inhalt der Range startete hierbei nur mit dem ersten Buchstaben des Worts und wird in der Schleife jeweils um den nächsten Buchstaben erweitert. Sobald mehr als ein Rechteck zurückgegeben wird, ist somit der Zeilenumbruch erreicht. Das Wort wurde daraufhin in zwei Teile unterteilt und dessen Positionen zurückgegeben³⁸.

Durch diese Umsetzung lassen sich Inhalt und Layout der vorhandenen Textinhalte von HTML-Elementen in das PDF-Dokument übertragen.

³⁸Der Quellcode dieser Funktion findet sich in Anhang E.I.7

5.6.2. Bilder verschiedener Formate

In den Anforderungen wurde spezifiziert, dass die Funktionalität des PDF-Writers für die initiale Zielerreichung nur Bilder im JPEG-Format unterstützen muss (siehe Anforderungen **F038** bis **F041**). Da jedoch auf Webseiten auch Bilder anderer Formate genutzt werden können, musste eine geeignete Lösung gefunden werden.

Die Lösung dieses Problems ist die Konvertierung der Bilder aus anderen Formaten in ein Bild im JPEG-Format. Hierfür wurde das Canvas-Element verwendet. In dieses kann ein vorhandenes Image-Element mit der Funktion `drawImage()` gezeichnet werden. Daraufhin kann mit der Funktion `toDataURL()` eine Data-URL für den Inhalt des Canvas-Elements erzeugt werden. Wird diese Data-URL mit `fetch` aufgerufen, so wird die jeweilige Bild-Datei zurückgegeben. Diese kann daraufhin der `pdfDocument`-Instanz zum Platzieren übergeben werden. Der dargestellte Quellcode 31 zeigt den Ausschnitt aus der Funktion `handleElementNode()` welche dies durchführt.

```

1 // [ ... ]
2 const canvas = document.createElement('canvas');
3 canvas.width = (element.el as HTMLImageElement).naturalWidth;
4 canvas.height = (element.el as HTMLImageElement).naturalHeight;
5 const ctx = canvas.getContext('2d');
6 ctx.fillStyle = 'rgb(255, 255, 255)';
7 ctx.fillRect(0, 0, canvas.width, canvas.height);
8 ctx.drawImage(element.el as HTMLImageElement, 0, 0);
9 const imageData = canvas.toDataURL('image/jpeg', 1.0);
10 const imgData = await fetch(imageData).then((res) => res.arrayBuffer());
11 this.pdf.addImageToPage(
12   imgData as Buffer,
13   // [ ... ]
14   // [ ... ]
15 );

```

Quellcode 31: Ausschnitt der Funktion `handleElementNode()`, welche für das Hinzufügen von Bildinhalten Verwendung findet.

Diese Lösung besitzt jedoch den Nachteil, dass transparente Flächen, wie sie in einem Bild im PNG-Format vorkommen können, nicht mehr transparent sind, da Transparenz vom JPEG-Format nicht unterstützt wird. Da der Hintergrund eines Canvas-Elements standardmäßig transparent schwarz ist, wird dieser bei der Überführung eines Canvas-

Elements in ein JPEG-Bild schwarz dargestellt [33]. Um dies zu verhindern, wird in Zeile 6 und 7 des Quellcodes 31 auf das Canvas-Element ein weißes Rechteck gezeichnet. Dies stellt jedoch nur eine Behelfslösung dar und sollte in einer weiteren Iteration der Bibliothek durch die Farbe des Hintergrunds ersetzt werden bis Bilder anderer Formate unterstützt werden.

5.6.3. Interaktion während der Generierung

Zu Beginn der Entwicklung wurde der Inhalt zunächst nicht in ein iFrame verschoben. Dies führte zu mehreren Komplikationen. Einerseits führte das Entfernen von Elementen, welche nicht in das PDF-Dokument übertragen werden sollten, zu sichtbaren Veränderungen im Seitenlayout. Andererseits mussten Kopf- und Fußzeilen für die Verwendung dem Webinhalt hinzugefügt werden, sodass diese vom Browser gerendert und somit die benötigten Werte wie Position und Größe berechnet werden. Zudem führte das Scrollen des Webinhals, während der Transfer durchgeführt wird, zu fehlerhaften Positionierungen, da der Scroll-Offset nur zu Beginn des Generierungsprozesses ausgelesen wurde.

Dies stand im Widerspruch zu der Anforderung **F124**, welche explizit die weitere Nutzung der Seite fordert, sowie der Anforderung **F125**, welche angibt, dass sich die Darstellung des Webinhals nicht durch die Generierung ändern soll. Um diesen Anforderungen gerecht zu werden, wurde die Verschiebung aller Inhalte in ein iFrame-Element, welches an den Webinhalt angehängt wird, implementiert. Das iFrame-Element wird durch die Verwendung der CSS-Eigenschaft `position` mit dem Wert `fixed` und den Positions値en `top: -9999px` und `left: -9999px` außerhalb des sichtbaren Bereichs des Webinhals platziert. Diese Lösung bietet zudem den Vorteil, dass für das iFrame-Element zusätzlich noch eine Breite angegeben werden kann. Dadurch, dass diese Viewport-unabhängig von der Bibliothek auf den gleichen Wert gesetzt wird, werden so auch bei einer Verwendung von responsivem Design immer die gleichen CSS-Regeln angewendet. Dadurch wird auch die Anforderung **F135** erfüllt, welche ein einheitliches PDF-Dokument unabhängig des verwendeten Browsers und Endgeräts als Resultat der Generierung fordert.

5.6.4. Schriften aus dem Webinhalt

Das Auslesen der verwendeten Schriftarten eines Webinhals stellt ebenfalls eine Komplikation dar. Infolgedessen, dass die Schriftarten-Dateien benötigt werden, um diese in das PDF-Dokument einzubetten, müssen diese unter einer URL verfügbar und in einem

Stylesheet mit einer Font-Face-Rule angegeben werden. Dies ist nicht bei allen Webseiten der Fall, da auch lokal installierte Schriften Verwendung finden können. Eine Lösung für dieses Problem bildet die Nutzung der *Local Font Access API*. Mit dieser lassen sich alle lokal installierten Schriften als ein Array von `FontData`-Instanzen auslesen. Die Instanzen enthalten dabei eine Methode, mit welcher die Schriftdatei als Blob zur Verfügung gestellt wird [34].

Diese Lösung beinhaltet jedoch selbst weitere Komplikationen. Zum einen steht diese Funktionalität laut *Can I Use* nur in neueren Versionen von *Google Chrome* und *Microsoft Edge* zur Verfügung. Dies entspricht nur 26.9% aller globalen Nutzenden [35]. Zum anderen öffnet sich bei Aufruf der Funktion ein Pop-Up Fenster des Browsers (siehe Abbildung 13). In diesem Fall müsst der Nutzende dies akzeptieren und somit tätig werden, was Anforderung **F136** widerspricht. Vielmehr kann das Popup-Fenster dazu führen, dass der Nutzende verunsichert wird und den Zugriff ablehnt.

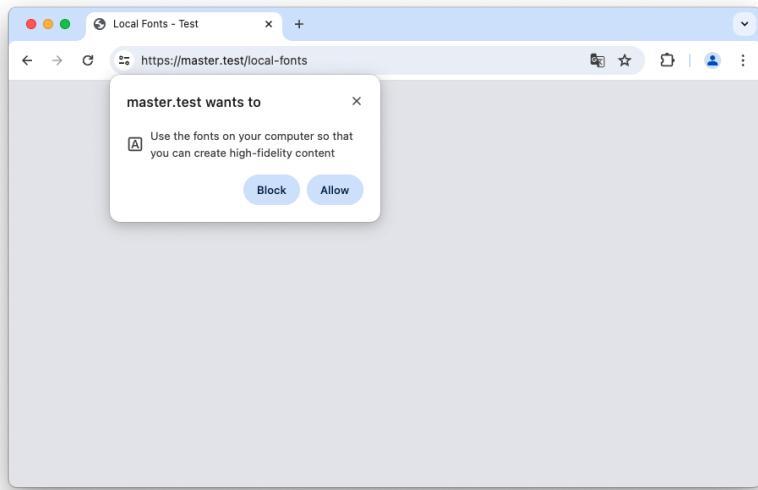


Abbildung 13: Screenshot des Browsers Google Chrome mit der Warnung, dass die Webseite auf die Schriften des Computers zugreifen will. Eigene Darstellung.

Aus diesen Gründen wurde sich gegen die Nutzung dieser Funktionalität entschieden. Somit stellt jedoch die Nutzung von Font-Face-Rules mit einer URL zu den Schriftdateien eine Vorbedingung dar (siehe Kapitel 5.7).

5.6.5. Transfer des CSS-Layouts von Elementen

Eine weitere Komplikation stellte die Übertragung der Gestaltungen von Elementen dar. Für diese stellt CSS eine Vielzahl von Möglichkeiten durch die Verwendung von Eigenschaften und dazugehörigen Werten zur Verfügung. Hierbei wurde zu Beginn der Implementierung die Übertragung durch die Erstellung von farbigen Flächen und Linien mithilfe der in der Spezifikation definierten Vektor-Operatoren angestrebt. Dies ließ sich für einfarbige, rechteckige Hintergründe und gerade, rechtwinklige Umrandungen umsetzen. Da CSS jedoch viele weitere Möglichkeiten wie abgerundete Ecken und Farbverläufe zur Verfügung stellt wurde festgestellt, dass dies nicht im Rahmen dieser Arbeit umsetzbar ist. Aus diesem Grund musste eine andere Lösung gefunden werden, welche eine dem Webinhalt gleichende Darstellung ermöglicht, um die Anforderung **F135** zu erfüllen.

Um dieses Problem behelfsmäßig zu lösen und die Übertragung des entsprechenden Layouts zu ermöglichen, wurde sich dafür entschieden die Darstellungen als Bilder in das PDF-Dokument zu übertragen. Dabei wurde die Methode `handleElementNode()` dahingehend erweitert, dass für das Element, welches verarbeitet wird, alle Computed Styles ausgelesen werden. Diese werden daraufhin einem neuen Div-Element hinzugefügt³⁹. Dieses Div-Element wird daraufhin mithilfe des foreignObject-Elements in eine Scalable Vector Graphic (SVG) übertragen. Das foreignObject-Element erlaubt hierbei die Nutzung von HTML-Elementen in SVGs. Das daraus entstandene SVG wird einem neuen Image-Element hinzugefügt. Sobald dieses geladen ist, wird dieses wie bereits in Kapitel 5.6.2 beschrieben auf ein Canvas gezeichnet und als Bild im JPEG-Format exportiert.

Diese Behelfslösung besitzt jedoch einige Nachteile. Zum einen steigt durch die Übertragung der Computed Styles und das Erstellen des Bildes die Generierungsduer an. Zum anderen erhöht sich durch die Verwendung von Bildern anstelle von Vektoren die Dateigröße des resultierenden Dokuments. Dies widerspricht zwar den nichtfunktionalen Anforderungen **NF002** und **NF003**, jedoch wurde der möglichst genauen Übertragung des Layouts und der Gestaltung eine höhere Priorität zugesprochen.

Außerdem stellte sich beim Testen in verschiedenen Browsern heraus, dass diese Lösung in Safari teilweise zu fehlerhaften Darstellungen führt. Dabei enthalten die generierten Bilder der Layouts der Elemente teilweise Weißraum auf der linken und rechten Seite, was zu einer fehlerhaften Darstellung führt. Dieser Fehler wird beispielsweise bei der Gestaltung von Tabellen wie in Abbildung 14 dargestellt zwischen den einzelnen Zellen sichtbar.

³⁹Der entsprechende Quellcode findet sich in Anhang E.I.6.

Erste Spalte	Zweite Spalte
Text	
Noch mehr Text	

Abbildung 14: Screenshot eines Ausschnitts aus einem in Safari generierten PDF-Dokument, in welchem die Gestaltung von Tabellenelementen fehlerhaft übertragen werden. Eigene Darstellung.

Aufgrund der angeführten Nachteile wurde für die Verwendung dieser Funktionalität die weitere Option `imagesForLayout` für das Optionen-Objekt hinzugefügt. Diese ist standardmäßig `true` und aktiviert somit den Transfer der Gestaltung von Elementen als Bilder. Wird für diese der Wert `false` angegeben, so werden nur einfarbige Hintergründe in Form von farbigen Rechtecken und gerade Linien für Umrandungen übertragen⁴⁰.

5.7. Verwendung der Bibliothek

Nachdem in den vorherigen Unterkapiteln auf die Umsetzung der Bibliothek eingegangen wurde, wird in diesem Kapitel vorgestellt wie die Bibliothek verwendet werden kann. Zum einen muss die JavaScript-Datei der Bibliothek in die Webseite eingebunden werden. Dies kann, wie in Quellcode 32 dargestellt, über die Verwendung eines Script-Elements geschehen.

```
1 <script src="/htmlWebsite2pdf.min.js"></script>
```

Quellcode 32: Importieren der entwickelten Bibliothek mit einem Script-Element.

Wurde die Datei in die HTML-Datei eingebunden, so stehen den Entwickelnden zwei Funktionen zur Generierung eines PDF-Dokuments zur Verfügung. Eine, welche den kompletten Inhalt des Body-Elements überführt (Zeile 2 in Quellcode 33), sowie eine weitere, bei welcher ein Element angegeben werden muss, dessen Inhalte übertragen werden sollen (Zeile 4 in Quellcode 33).

```
1 // Funktionsaufruf für die Generierung eines PDF-Dokuments mit dem
   gesamten Body als Inhalt:
```

⁴⁰PDF-Dokument mit der Nutzung der Option findet sich in Anhang F.I und mit der Verwendung von Vektoren in Anhang F.II.

```
2 htmlWebsite2Pdf.fromBody();  
3 // Funktionsaufruf für die Generierung welches nur den Inhalt eines  
// Elements (in diesem Fall des Main-Elements) enthält:  
4 htmlWebsite2Pdf.fromElement(document.querySelector('main'));
```

Quellcode 33: Funktionsaufruf der Generierung eines PDF-Dokuments durch die entwickelte Bibliothek.

Zusätzlich kann, wie bereits beschrieben, bei diesen Aufrufen ein Objekt an Optionen übergeben werden. Mit diesem können einerseits Funktionen wie die Erstellung der Document Outlines oder Kopf- und Fußzeilen aktiviert werden. Andererseits können Metadaten für das Dokument und Optionen für den genutzten PDF-Viewer angegeben werden. Eine vollständige Liste aller möglichen Optionen findet sich in Anhang E.II.

Bei der Nutzung der Bibliothek ist jedoch zu beachten, dass einige Voraussetzungen erfüllt sein müssen, sodass ein best mögliches Ergebnis generiert werden kann. Diese resultieren aus den im vorherigen Unterkapitel 5.6 vorgestellten Lösungen der Komplikationen und dem aktuellen Stand der Umsetzung. Zu diesen Voraussetzungen zählt einerseits die Verwendung von FontFace-Regeln mit angegebenen Schriftartdateien im TrueType-Format, da zum Zeitpunkt dieser Arbeit noch keine weiteren Schrifttypen unterstützt werden. Andererseits muss, wie in Kapitel 5.6.2 beschrieben, beachtet werden, dass Bilder in das JPEG-Format überführt werden, wodurch es zu fehlerhaften Darstellungen kommen kann. Dies gilt ebenso für die Verwendung der Option der Übertragung der Gestaltung von Layouts als Bilder bei der Verwendung von Safari. Eine weitere Voraussetzung bildet die Reihenfolge der Elemente im Markup, da diese während des Transfers sequentiell verarbeitet werden.

6. Evaluation

Um die in dieser Arbeit umgesetzte Bibliothek zu evaluieren wurde diese ebenso zur Generierung eines PDF-Dokuments aus dem Main-Element der in Kapitel 3 angegebenen Test-Webseite ohne die Verwendung weiterer Optionen angewendet⁴¹.

In dem resultierenden PDF-Dokument ist zu sehen, dass das Layout großteils der Darstellung des Webinhals gleicht und Elemente, und Textzeilen, welche nicht mehr auf der Seite platziert werden können auf die nächste Seite verschoben werden. Wie in Abbildung 15 zu sehen werden dabei die Texte gemäß ihrer Darstellung im Webinhalt übertragen, dies gilt sowohl für Texte im Flatter- als auch Blocksatz. Lediglich Unterstreichungen, welche bei den Verlinkungen Verwendung finden, werden nicht übertragen. Ein weiterer Unterschied ist bei den Gestaltungen von Inline-Text-Elementen vorzufinden. Wenn der Text des Elements über mehrere Zeilen läuft, wie es bei der Umrandung der Fall ist wird dieser bei der Übertragung des Layouts durch die Verwendung von Bildern nicht korrekt übertragen.

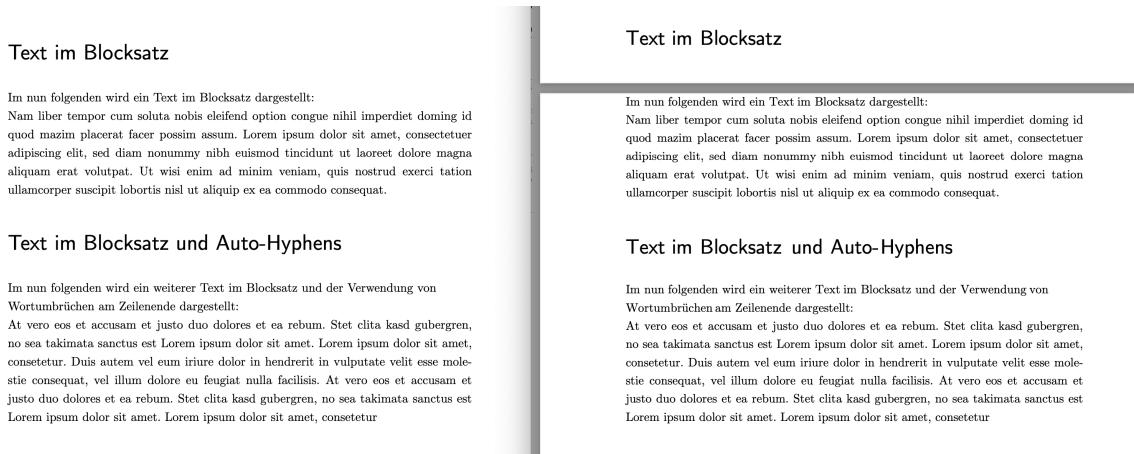


Abbildung 15: Ausschnitte des zur Generierung verwendeten Webinhals (links) und des durch die Bibliothek generierten PDF-Dokuments (rechts). Eigene Darstellung.

Führt man die Generierung für den Webinhalt mit der Nutzung von Vektoren, anstelle von Bildern für die Gestaltung von Elementen aus, so werden auch diese Gestaltungen, da sie nur aus Hintergründen und geraden Umrandungen bestehen, korrekt übertragen⁴². Auch Inline-Text-Elemente werden in diesem Fall korrekt dargestellt. Ein Vergleich der Dateigrößen der PDF-Dokumente zeigt, dass die Verwendung von Vektoren Speicherplatz

⁴¹PDF-Dokument findet sich in Anhang F.I

⁴²PDF-Dokument findet sich in Anhang F.II

sparender ist. Der Unterschied beläuft sich hierbei auf 34 KB.

Um die Generierung unter der Verwendung einer zusätzlichen Fußzeile zu untersuchen wurde das Main-Element um das in Quellcode 34 dargestellte Template-Element erweitert. Damit dieses Element als Fußzeile verwendet werden kann müssen noch einige Optionen angepasst werden. Zum einen muss die Option `usePageFooters` durch den Wert `true` genutzt werden. Zum anderen muss die Option `addFirstPage` deaktiviert und ein Seitenumbruch vor Section-Elementen hinzugefügt. Dies führt dazu, dass bereits auf der ersten Seite die Fußzeile verwendet wird.

```
1 <template data-htmlWebsite2pdf-footer>
2   <div class="md:w-[210mm] w-full text-center text-base pb-[10mm] pt-[5
3     mm] mb-10">
4     <div class="md:mr-[20mm] md:ml-[30mm]">
5       <div class="font-serif border-t border-black">
6         <span data-htmlWebsite2pdf-currentPageNumber>X</span>
7       </div>
8     </div>
9   </div>
10 </template>
```

Quellcode 34: Template-Element, welches für die Verwendung als Footer dem Main-Element hinzugefügt wurde.

Betrachtet man das aus dieser Generierung resultierende Dokument⁴³, so findet sich am Fußende jeder Seite die durch das Template-Element spezifizierte Fußzeile, welche aus einer zentrierten Seitenzahl und einer horizontalen Trennlinie besteht. Des weiteren ist auf der zweiten Seite eine freie Fläche unter der Überschrift *Bild* sichtbar. Diese resultiert daraus, dass durch die Verwendung der Fußzeile nicht ausreichend Platz für das Bild gegeben ist und dieses deshalb auf die nächste Seite verschoben wird um eine Zerteilung zu vermeiden.

Wenn die Generierung in einem Browser mit schmalen Viewport durchgeführt wird, so wird trotz der Verwendung eines responsivem Design ein PDF-Dokument mit dem gleichen Inhalt und der gleichen Darstellung generiert. Dies zeigt, dass Anforderung **F135** erfüllt ist.

Wie bereits in Kapitel 5.7 beschrieben existieren trotz der im vorherigen angeführten Ergebnisse zu den Resultaten der Generierung einige Funktionalitäten welche noch nicht feh-

⁴³PDF-Dokument findet sich in Anhang F.III

lerfrei ausgeführt werden. Zu diesen zählt beispielsweise die Übertragung der Gestaltung von Elementen als Bilder. Diese funktioniert hierbei nicht fehlerfrei für Inline-Elemente, als auch bei der Durchführung der Generierung im Browser Safari. Ebenso ist es wichtig, dass die Schriftarten im TrueType-Format angegeben werden und die Struktur des HTML-Markups der Reihenfolge der Darstellung folgt.

Trotz dieser Fehlerquellen konnten für die Teilfunktionalität des PDF-Writers alle in der Anforderungsermittlung aufgeführten Muss-Anforderungen erfüllt werden. Für diese wurden jedoch zum Abschluss dieser Arbeit noch keine der Soll- und Wird-Anforderungen umgesetzt. Ähnlich verhält es sich mit den Anforderungen an den Transfer des Webinhalts. Hierbei werden bis auf vier Muss-Anforderungen alle weiteren erfüllt. Nicht erreicht wurde die Anforderung **F133**, welche vorgibt, dass Textinhalte in der gleichen Darstellung wie im Webinhalt übertragen werden sollen. Da jedoch keine Unterstreichungen übertragen werden ist diese Anforderung nicht erfüllt. Selbiges gilt für Anforderung **F134**, welche den Transfer der Gestaltung von Elementen fordert, sowie Anforderung **F135** welche eine einheitliche Darstellung unabhängig des Endgeräts und Browsers fordert. Dadurch, dass zum einen die Darstellung unter Safari teilweise fehlerhaft übertragen wird und zum Anderen die Darstellung für Inline-Elemente nicht korrekt transferiert wird, werden diese nicht erfüllt.

Die nichtfunktionalen Anforderungen wurden in der Umsetzung ebenso nicht alle erfüllt. Dabei liegt die Dateigröße der Bibliothek bei 774 KB in einer minimierten Version und verfehlt somit den in Kapitel 4.2.2 für die Dateigröße festgelegten Richtwert von 734,5 KB. Selbiges lässt sich auch für die Dateigröße des resultierenden PDF-Dokument feststellen, da dieses bei der Verwendung von Bildern für die Gestaltung der Elemente eine Größe von 821 KB, beziehungsweise 787 KB bei der Verwendung von Vektoren, aufweist. Lediglich der für Anforderung **NF002** definierte Richtwert von 500ms konnte mit einer Dauer von 327ms beziehungsweise 354ms je nach verwendeten Optionen erfüllt werden. Es ist jedoch davon auszugehen, dass durch die Implementierung der in Anforderung **F034** geforderten Verwendung von Subsets für die benötigten Schriftarten die resultierende Dateigröße auf einen Wert reduziert werden kann, sodass die nichtfunktionale Anforderung bezüglich der Dokumenten-Größe in Zukunft erfüllt wird.

7. Fazit und Ausblick

In dieser Arbeit wurde die clientseitige Generierung von PDF-Dokumenten aus vorliegenden Webinhalten betrachtet. Es wurde einerseits die PDF-Spezifikation, welche den Aufbau von PDF-Dateien vorgibt, betrachtet und Teile dieser vorgestellt. Des Weiteren wurden zum Zeitpunkt dieser Arbeit zur Verfügung stehende Möglichkeiten und Technologien zur Generierung von PDF-Dokumenten betrachtet. Für diese wurden dabei die Nutzung und resultierenden Dokumente überprüft und daraus Vor- und Nachteile für die Nutzung der Möglichkeiten abgeleitet. Dabei stellte sich heraus, dass alle Möglichkeiten einige Nachteile aufweisen, welcher einer Nutzung entgegenstehen beziehungsweise welche bei der Nutzung beachtet werden müssen. Nach Ansicht des Autors ist dies für die Verwendung des Druck-Dialogs die notwendigen Einstellungen welche durch den Nutzenden getroffen werden müssen. Für die Bibliothek *jsPDF* stellt die fehlerhafte Positionierung der zum Layout von Elementen gehörenden Hintergründe und Umrandungen einen Nachteil dar. In Verbindung mit der fehlerhaften Positionierung von Inline-Elementen bei der Nutzung der Seitenumbruchs-Option für Texte führt die Generierung mit *jsPDF* zu einem fehlerhaften Layout. Bei der Bibliothek *html2pdf.js* spricht aus Sicht des Autors die Tatsache, dass alle Inhalte als Bild übertragen werden was dazu führt, das Texte nicht markiert oder kopiert werden können und die Dokumente eine große Dateigrößen aufweisen gegen eine Nutzung.

Anhand dieser aufgefundenen Nachteile konnte eine Konzeption für eine neue Bibliothek aufgestellt werden, mit welcher die Nachteile verringert werden und weitere Funktionalitäten für die Generierung von PDF-Dokumenten aus Webinhalten hinzu kommen. Dieses Konzept konnte in einer ersten Version im Rahmen dieser Arbeit implementiert werden und zeigt die Möglichkeiten dieser. Durch das Einsetzen von Seitenzahlen können dabei Seitennummerierungen und Inhaltsverzeichnisse geschaffen werden. Zusätzlich ermöglicht die Nutzung der Funktion für Kopf- und Fußzeilen die Erstellung eines Dokuments mit einem einheitlichen Seitenlayout.

In der Umsetzung wurde festgestellt, dass das Ermöglichen der Übertragung aller Möglichkeiten des CSS-Layouts von HTML-Webinhalten eine umfangreiche Implementierung benötigt welche in dieser Arbeit nicht zu erreichen war. Trotzdem konnten der Transfer der Gestaltung von Elementen und Bildern in Formaten, welche noch nicht in den PDF-Writer implementiert sind, durch die Behelfslösung der Transformation in JPEG-Bilder ermöglicht werden. Hierdurch wurde jedoch die nichtfunktionalen Anforderungen zur Da-

teigröße und Generierungsdauer missachtet. Sowohl diese, als auch die Umsetzung für die Überführung von Textinhalten zeigen jedoch, wie sich Inhalt und Layout von HTML-Elementen in PDF-Dokumenten abbilden lassen. Die Umsetzung zeigt außerdem wie sich Textinhalte von Elementen, den angegebenen CSS-Regeln entsprechend, zeilenweise auslesen und in das PDF-Dokument übertragen lassen.

Die entwickelte Bibliothek bietet die Möglichkeit HTML-Markup in PDF-Dokumente unter Einbezug des CSS-Layouts zu übertragen. Jedoch existieren noch einige Hürden und Voraussetzungen welche bei einer Nutzung in der zum Abschluss dieser Arbeit verfügbaren Version beachtet werden müssen. Nach Ansicht des Autors sollte vor einer produktiven Nutzung das Verhalten und die resultierenden Dokumente ausgiebig in verschiedenen Browsern getestet werden, um Fehler in der Generierung zu vermeiden.

Für eine weitere wissenschaftliche Auseinandersetzung bietet der Themenbereich der clientseitigen Generierung von PDF-Dokumenten aus Webinhalten viele weitere Möglichkeiten. Beispielsweise kann durch die Betrachtung der Möglichkeit der Übertragung von Formularen in das PDF-Dokument unter der Verwendung der in der PDF-Spezifikation beschriebenen *AcroForms* die Bibliothek erweitert werden. Besonders die Übertragung des angewendeten CSS-Layouts der Form-Elemente auf die Elemente von AcroForms bieten weiteren Raum für eine Untersuchung. Des Weiteren bietet auch die Erweiterung des Transfers um die Nutzung von Tagged PDF und der Einhaltung der Vorschriften aus dem PDF/UA-Standards einen spannenden Teilbereich. Hierdurch könnten aus dem Inhalt und Layout von barrierefreien Webinhalten ebenso barrierefreie PDF-Dokumente erstellt werden. Außerdem bieten Verbesserungen, welche die nichtfunktionalen Anforderungen betreffen, weitere Betrachtungsfelder. Dabei könnte beispielsweise sowohl die Geschwindigkeit, als auch die resultierende Dateigröße durch die Erweiterung der Überführung von Layouts von Elementen beeinflusst werden. Hierbei können die verschiedenen Vektor-Operatoren betrachtet werden, welche der PDF-Standard zur Verfügung stellt. In diesem Zusammenhang stellt auch der Vergleich des Aufbaus von SVGs zu dem der Operatoren der PDF-Spezifikation einen weiteren Betrachtungsbereich dar.

Abbildungsverzeichnis

1.	Aufbau einer PDF-Datei mit Inkrementellen Updates.	16
2.	Darstellungen aller Boxen in einem PDF.	17
3.	Darstellung der PDF-Datei als gerichteter Graph mit dem Document Catalog als Wurzelknoten.	18
4.	Ausschnitt der Darstellung von Document Outlines in Apple Preview (links) und Adobe Acrobat Reader (rechts).	27
5.	Screenshot des Druck-Dialogs im Browser Google Chrome.	34
6.	Screenshot des Druck-Dialogs im Browser Safari.	35
7.	Screenshot eines Texts, welcher in der Mitte unterteilt wurde, da er auf dem Seitenumbruch liegt.	36
8.	Screenshots des Druckdialogs unter iOS ohne weitere Einstellungsmöglichkeiten für die Verwendung von Hintergrundgrafiken und das Deaktivieren der Fußzeile.	37
9.	Screenshot der Tabelle aus dem PDF-Dokument in Anhang C.II.2, welche eine fehlerhafte Positionierung aufweist.	41
10.	Klassendiagramm, welches die verschiedenen Objekttypen darstellt. . . .	59
11.	Screenshot eines Ausschnitts einer PDF-Seite mit der implementierten Iteration für Blocksatz.	76
12.	Screenshot eines Ausschnitts einer PDF-Seite mit der zweiten Iteration für Blocksatz und fehlerhaften Wortumbrüchen.	77
13.	Screenshot des Browsers Google Chrome mit der Warnung, dass die Webseite auf die Schriften des Computers zugreifen will.	80
14.	Screenshot eines Ausschnitts aus einem in Safari generierten PDF-Dokument, in welchem die Gestaltung von Tabellenelementen fehlerhaft übertragen werden.	82
15.	Ausschnitte des zur Generierung verwendeten Webinhalts (links) und des durch die Bibliothek generierten PDF-Dokuments (rechts).	84

Tabellenverzeichnis

1.	Versionen des Portable Document Formats	5
2.	Notwendige Key-Value-Paare eines Zwischenknotens.	20
3.	Notwendige Key-Value-Paare in Dictionary Objects vom Typ Page.	20
4.	Notwendige Key-Value-Paare eines Image Dictionary.	24
5.	Key-Value-Paare, welche in einem Outlines-Dictionary möglich sind.	28
6.	Ausschnitt von Key-Value-Paare welche in einem Outline-Item-Dictionary möglich sind.	29
7.	Gegenüberstellung der Vor- und Nachteile des Druckdialogs	45
8.	Gegenüberstellung der Vor- und Nachteile der Nutzung von <i>jsPDF</i>	46
9.	Gegenüberstellung der Vor- und Nachteile der Nutzung von <i>html2pdf.js</i>	47

Quellcodeverzeichnis

1.	Beispiel Werte von Integer und Real Objects.	7
2.	Beispielhafte String Objects.	8
3.	Verschiedene Array Objects.	8
4.	Beispiel Dictionary Object welches eine Seite im PDF-Dokument darstellt.	9
5.	Beispiel Dictionary Object aus Quellcode 4 ohne die Verwendung von Leerzeichen und Zeilenumbrüchen.	9
6.	Stream Object mit einer Länge von 324 Bytes, welcher mit FlateDecode (zlib/decode) entkomprimiert werden muss.	10
7.	Indirektes Objekt für den MediaBox-Wert aus Quellcode 4.	11
8.	PDF-Header für eine PDF-Datei in der Version 1.7.	12
9.	Schema einer Zeile in einer Cross-Reference-Subsection.	13
10.	Beispielhafte Cross-Reference-Table mit 6 Objekten, wovon vier verwendet werden und eines frei ist.	13
11.	Beispielhafter Trailer einer PDF-Datei.	14
12.	Beispielhafter Document Catalog, durch welchen die Seiten des PDF-Dokument in einer Spalte angezeigt werden.	19
13.	Beispielhafter Page-Tree mit drei Seiten.	21
14.	Ein beispielhaftes Resource Dictionary mit 2 Schriftarten und einem externen Objekt.	22
15.	Beispielhaftes FontDictionary für eine Type1 Schriftart, welche in den Standardschriften vorhanden ist.	23
16.	Anweisungen im Content Stream zum Platzieren eines Bildes.	25
17.	Anweisungen im Content Stream zum Platzieren des Texts „Hello World“ in rot mit der Schriftart Helvetica und der Größe 12pt.	27
18.	Beispielhafte Document-Outlines einer PDF-Datei mit vier Elementen. . . .	29
19.	Zwei beispielhafte Annotations. Eine verlinkt intern, die andere extern. . . .	31
20.	Page-Dictionary, welches die Annotations aus Quellcode 19 besitzt.	32
21.	Funktionsaufruf zur Generierung eines PDF-Dokuments aus einem HTML-Webinhalt mit jsPDF	39
22.	Erweiterter Funktionsaufruf zur Generierung eines PDF-Dokuments aus einem HTML-Webinhalt mit jsPDF und Verwendung von anderen Schriftarten. . . .	39
23.	Aufruf der PDF-Generierung für das Body-Element mittels <i>html2pdf.js</i>	42

24.	Erweiterter Aufruf der PDF-Generierung für das Body-Element mittels <i>html2pdf.js</i> mit gesetzten Optionen.	43
25.	Methode der Klasse <code>pdfDocument</code> für das Hinzufügen eines neuen Page-Objects.	61
26.	Methode, welche die eigentliche PDF-Datei als Buffer zurückgibt.	64
27.	Methode <code>addEntry</code> der Klasse <code>CrossReferenceSection</code>	65
28.	Aufbau eines Schrift-Objekts.	70
29.	Teil des Switch-Case-Statements für Überschriften bei der Überprüfung von <code>ElementNodes</code>	70
30.	Erste Umsetzung für das zeilenweise Auslesen der Textinhalte.	74
31.	Ausschnitt der Funktion <code>handleElementNode()</code> , welche für das Hinzufügen von Bildinhalten Verwendung findet.	78
32.	Importieren der entwickelten Bibliothek mit einem Script-Element.	82
33.	Funktionsaufruf der Generierung eines PDF-Dokuments durch die entwickelte Bibliothek.	82
34.	Template-Element, welches für die Verwendung als Footer dem Main-Element hinzugefügt wurde.	85

Abkürzungsverzeichnis

ASCII American Standard Code for Information Interchange

ASN.1 Abstract Syntax Notation One

CSS Cascading Style Sheet

DOM Document Object Model

HTML Hypertext Markup Language

ISO International Organization for Standardization

JPG / JPEG Joint Photographic Experts Group

JS JavaScript

NPM Node Package Manager

PDF Portable Document Format

PDF/A PDF for Archiving (PDF für die Archivierung)

PDF/E PDF for Engineering (PDF für das Ingenierwesen)

PDF/H PDF for Health Care Industry (PDF für das Gesundheitswesen)

PDF/R PDF for raster-image documents (PDF für Raster-Bilder Dokumente)

PDF/UA PDF for Universal Accessibiliy (PDF für den universellen Zugang)

PDF/X PDF for Exchange (PDF für den Austausch)

png Portable Network Graphics

pt Point

SVG Scalable Vector Graphic

TS TypeScript

TTF True Type Format

URI Uniform Resource Identifier

URL Uniform Resource Locator

Literaturverzeichnis

- [1] Abhigyan Modi. *PDF und Adobe Acrobat: 30 Jahre im Zeichen der Kommunikation und der digitalen Transformation.* de. Juni 2023. URL: <https://blog.adobe.com/de/publish/2023/06/15/pdf-und-adobe-acrobat-30-jahre-im-zeichen-der-kommunikation-und-der-digitalen-transformation> (besucht am 09.04.2024).
- [2] Adobe Systems Software Ireland Limited. *Was ist eine PDF-Datei? Portable Document Format.* de. URL: <https://www.adobe.com/de/acrobat/about-adobe-pdf.html> (besucht am 09.04.2024).
- [3] John Whitington. *PDF explained the ISO standard for document exchange.* en. O'Reilly, 2012.
- [4] John Warnock. *The Camelot Project.* en. 1991. URL: https://pdfa.org/norm-refs/warnock_camelot.pdf (besucht am 02.03.2024).
- [5] Peter Zschunke. *PDF wird 20 Jahre alt / heise online.* de. Juni 2013. URL: <https://www.heise.de/news/PDF-wird-20-Jahre-alt-1883567.html> (besucht am 30.04.2024).
- [6] Leonard Rosenthal. *Developing with PDF: Dive into the Portable Document Format.* en. O'Reilly, 2014.
- [7] International Organization for Standardization. *ISO - The standard for PDF is revised.* en. Jan. 2021. URL: <https://www.iso.org/news/ref2608.html> (besucht am 02.05.2024).
- [8] Tim Bienz, Richard Cohn und James R. Meehan. *Portable document format reference manual - Version 1.1.* en. Addison-Wesley Pub. Co., 1996.
- [9] Tim Bienz, Richard Cohn und James R. Meehan. *Portable document format reference manual - Version 1.2.* en. Addison-Wesley Pub. Co., 1996.
- [10] Jim Meehan u. a. *PDF Reference, Second Edition - Adobe Portable Document Format Version 1.3.* en. Addison-Wesley, 2000.
- [11] Jim Meehan u. a. *PDF Reference, Third Edition - Adobe Portable Document Format Version 1.4.* en. Addison-Wesley, 2001.
- [12] Jim Meehan u. a. *PDF Reference, Fourth Edition - Adobe Portable Document Format Version 1.5.* en. 2003. URL: https://opensource.adobe.com/dc-acrobat-sdk-docs/pdfstandards/pdfreference1.5_v6.pdf (besucht am 11.03.2024).

- [13] Jim Meehan u. a. *PDF Reference, Fifth Edition - Adobe Portable Document Format Version 1.6.* en. 2004. URL: <https://opensource.adobe.com/dc-acrobat-sdk-docs/pdfstandards/pdfreference1.6.pdf> (besucht am 11.03.2024).
- [14] International Organization for Standardization. *ISO - The worldwide standard for electronic documents is evolving.* en. Aug. 2017. URL: <https://www.iso.org/news/ref2199.html> (besucht am 02.05.2024).
- [15] Dietrich von Seggern u. a. *PDF/X in a Nutshell - PDF for printing – The ISO standard.* en. URL: <https://pdfa.org/wp-content/uploads/2017/05/PDFX-in-a-Nutshell.pdf> (besucht am 19.04.2024).
- [16] Adobe Systems Software Ireland Limited. *PDF/A-Format: Merkmale und Unterschiede zur PDF-Datei.* de. URL: <https://www.adobe.com/de/acrobat/resources/document-files/pdf-types/pdf-a.html> (besucht am 18.04.2024).
- [17] PDF Association. *PDF/A - Normreihe ISO 19005 für digitale Langzeitarchivierung.* de. URL: <https://pdfa.org/wp-content/uploads/2014/01/Flyer-PDFA-DEU.pdf> (besucht am 18.04.2024).
- [18] PDF Association. *PDF/UA – ISO 14289: Der neue Standard für barrierefreie PDF-Dokumente und PDF-Formulare.* de. URL: <https://pdfa.org/wp-content/uploads/2013/12/Flyer-PDFUA-DEU.pdf> (besucht am 18.04.2024).
- [19] PDF Association. *Glossary of PDF terms – PDF Association.* en. URL: <https://pdfa.org/glossary-of-pdf-terms/> (besucht am 17.04.2024).
- [20] Adobe Systems Incorporated. *Document Management — Portable Document Format — Part 1: PDF 1.7.* en. Hrsg. von International Organization for Standardization. ISO 32000-1:2008. Juli 2008.
- [21] PDF Association. *PDF - Basics - CheatSheet.* en. URL: <https://pdfa.org/wp-content/uploads/2023/08/PDF-Basics-CheatSheet.pdf> (besucht am 30.03.2024).
- [22] *Proposal: Allow authors to enforce “save as PDF” when printing a document - Issue #7946 - whatwg/html.* en. Mai 2022. URL: <https://github.com/whatwg/html/issues/7946> (besucht am 21.04.2024).
- [23] Parallax Agency Ltd. *parallax/jsPDF: Client-side JavaScript PDF generation for everyone.* en. URL: <https://github.com/parallax/jsPDF> (besucht am 21.03.2024).

- [24] *jsPDF - npm*. en. URL: <https://www.npmjs.com/package/jspdf> (besucht am 22.04.2024).
- [25] Niklas von Hertzen. *niklasvh/html2canvas: Screenshots with JavaScript*. en. URL: <https://github.com/niklasvh/html2canvas/tree/master> (besucht am 24.03.2024).
- [26] Parallax Agency Ltd. *html - Documentation*. en. URL: <https://rawgit.com/MrRi/o/jsPDF/master/docs/module-html.html#~html> (besucht am 30.03.2024).
- [27] Erik Koopmans. *eKoopmans/html2pdf.js: Client-side HTML-to-PDF rendering using pure JS*. en. URL: <https://github.com/eKoopmans/html2pdf.js> (besucht am 24.02.2024).
- [28] *html2pdf.js - npm*. en. URL: <https://www.npmjs.com/package/html2pdf.js> (besucht am 25.02.2024).
- [29] Erik Koopmans. *html2pdf.js/package.json at master - eKoopmans/html2pdf.js*. en. URL: <https://github.com/eKoopmans/html2pdf.js/blob/master/package.json> (besucht am 24.02.2024).
- [30] *html2pdf failed with too many pages - Issue #19 - eKoopmans/html2pdf.js*. en. 2017. URL: <https://github.com/eKoopmans/html2pdf.js/issues/19> (besucht am 28.03.2024).
- [31] SOPHIST GmbH Die SOPHISTEN. »Schablonen für alle Fälle«. de. URL: https://www.sophist.de/fileadmin/user_upload/Bilder_zu_Seiten/Publikationen/Wissen_for_free/MASTeR_Broschuere_3-Auflage_interaktiv.pdf (besucht am 12.02.2024).
- [32] Joel Parker Henderson. *ADR template of the Markdown Any Decision Records (MADR) project*. en. URL: <https://github.com/joelparkerhenderson/architecture-decision-record/tree/main/locales/en/templates/decision-record-template-of-the-madr-project> (besucht am 02.04.2024).
- [33] Web Hypertext Application Technology Working Group (WHATWG). *HTML Living Standard — Last Updated 24 April 2024*. en. URL: <https://html.spec.whatwg.org/multipage/canvas.html#the-canvas-element> (besucht am 28.04.2024).
- [34] Mozilla Corporation. *Local Font Access API - Web APIs / MDN*. en. URL: https://developer.mozilla.org/en-US/docs/Web/API/Local_Font_Access_API (besucht am 19.05.2024).

- [35] Can I use. *Window API: queryLocalFonts / Can I use... Support tables for HTML5, CSS3, etc.* en. URL: https://caniuse.com/mdn-api_window_querylocalfonts (besucht am 19.05.2024).

Anhang A Portable Document Format

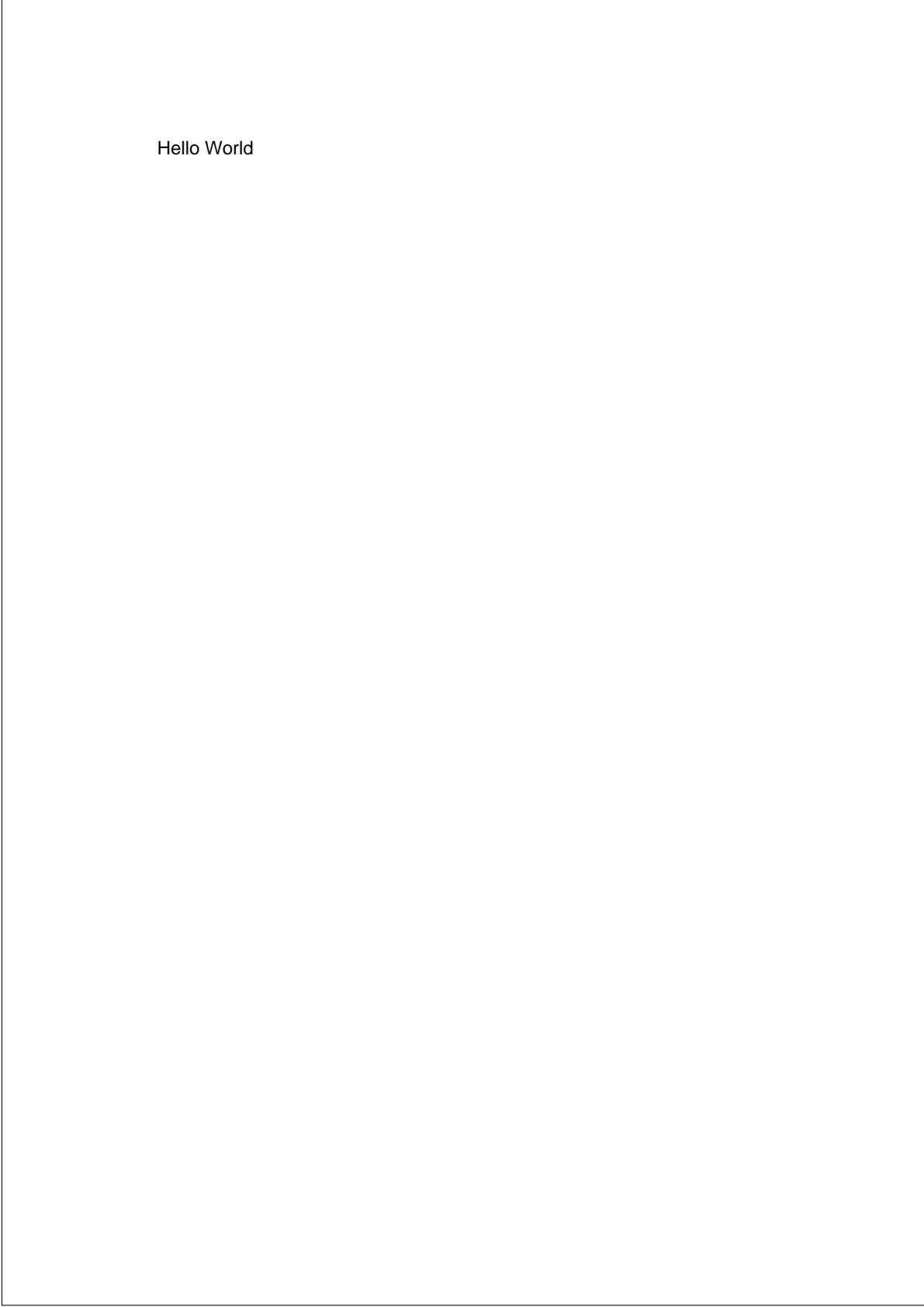
A.I Hello World PDF-Datei

Inhalt der PDF-Datei

```
1 %PDF-1.4
2 1 0 obj           % Document Catalog
3 <<
4 /Type /Catalog
5 /Pages 2 0 R
6 >>
7 endobj
8 2 0 obj           % Page-Tree Wurzelknoten
9 <<
10 /Type /Pages
11 /Count 1
12 /Kids [ 3 0 R ]
13 >>
14 endobj
15 3 0 obj           % Page-Dictionary für Seite 1
16 <<
17 /Type /Page
18 /MediaBox [ 0 0 595 842 ]
19 /Resources 4 0 R
20 /Contents 5 0 R
21 >>
22 endobj
23 4 0 obj           % Resource-Dictionary für Seite 1
24 <<
25 /Font << /F1 6 0 R >>
26 >>
27 endobj
28 5 0 obj           % Contentstream für Seite 1
29 <<
30 /Length 53
31 >>
32 stream
33 BT
34 /F1 12 Tf
35 1 0 0 1 100 742 Tm
```

```
36 (Hello World)Tj
37 ET
38 endstream
39 endobj
40 6 0 obj % Font Dictionary für /F1
41 <<
42 /Type /Font
43 /Subtype /Type1
44 /BaseFont /Helvetica
45 >>
46 endobj
47 xref % Cross-Reference-Table
48 0 7
49 0000000000 65535 f
50 0000000009 00000 n
51 0000000082 00000 n
52 0000000171 00000 n
53 0000000300 00000 n
54 0000000383 00000 n
55 0000000520 00000 n
56 trailer % Trailer
57 <<
58 /Size 7
59 /Root 1 0 R
60 >>
61 startxref
62 660
63 %%EOF
```

Resultierendes PDF-Dokument



Hello World

A.II Standard Schriftarten in PDF-Dokumenten

- Times-Roman
- Helvetica
- Courier
- Symbol
- Times-Bold
- Helvetica-Bold
- Courier-Bold
- ZapfDingbats
- Times-Italic
- Helvetica- Oblique
- Courier-Oblique
- Times-BoldItalic
- Helvetica-BoldOblique
- Courier-BoldOblique

Anhang B Quellcode der zur Generierung verwendeten Webseite

```
1 <html lang="de" class="scroll-smooth">
2 <head>
3   <meta charset="UTF-8">
4   <meta name="viewport" content="width=device-width, initial-scale=1.0">
5   <link href="./assets/css/styles.css" rel="stylesheet">
6   <script src="./assets/js/html2pdf.bundle.min.js"></script>
7   <script src="./assets/js/html2canvas.min.js"></script>
8   <script src="./assets/js/jspDF.umd.min.js"></script>
9   <script src="./assets/js/htmlWebsite2Pdf.min.js"></script>
10  <script src="./assets/js/main.js"></script>
11  <title>Clientseitige Generierung von PDF-Dokumenten aus Webinhalten
12    mit HTML-Markup und CSS-Layout</title>
13 </head>
14 <body class="md:bg-gray-200 relative">
15 <main class="md:w-[210mm] w-full mx-auto mb-20">
16   <section class="py-10 md:pr-[20mm] md:pb-[10mm] md:pl-[30mm] px-6
17     bg-white leading-normal font-serif">
18     <h1 class="font-normal font-sans text-h2 mt-[28.8pt] mb-[18pt]">
19       Test Generierungswebseite</h1>
20     <div>
21       <p>Diese Seite stellt eine Test-Webseite für die Prüfung der Mö
22         glichkeiten und Technologien zur Generierung von PDF-Dokumenten aus
23         Webinhalten mit HTML-Markup und CSS-Layout dar. Diese Seite verwendet
24         unterschiedliche Schriftarten und viele verschiedene Elemente,
25         welche in Webinhalten Verwendung finden können, aus welchen zudem
26         PDF-Dokumente generiert werden sollen.</p>
27     <h2 class="font-normal font-sans text-h2 mt-[28.8pt] mb-[18pt]">
28       Textparagraphen mit unterschiedlichen Inline-Darstellungen</h2>
29     <p>Der folgende Blindtext nutzt verschiedene Inline-Gestaltungsmö
30       glichkeiten von Text, wie kursive Texte, Fetttexte und Texte mit
31       Umrandung oder Hintergründen:<br><br>
32       Lorem ipsum dolor sit amet, consetetur sadipscing elitr, <strong>sed
33         diam nonumy eirmod tempor invidunt</strong> ut labore et dolore
34         magna aliquyam erat, sed diam voluptua. At vero eos et accusam et
35         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.  
<em>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.</em><br>  
23 <span style="border: 1px solid rgb(0, 255, 255);">Ut wisi enim ad  
minim veniam, quis nostrud exerci tation ullamcorper suscipit  
lobortis nisl ut aliquip ex ea commodo consequat. Duis autem vel eum  
iriure dolor in hendrerit in vulputate velit esse molestie consequat,  
</span>  
24 vel illum dolore eu feugiat nulla facilisis at vero eros et accumsan  
et iusto odio dignissim qui blandit praesent luptatum zzril delenit  
augue duis dolore te feugait nulla facilisi. At vero eos et accusam  
et justo duo dolores et ea rebum. <span class="bg-red-300">Stet clita  
kasd gubergren,</span> no sea takimata sanctus est Lorem ipsum dolor  
sit amet. Lorem ipsum dolor sit amet, consetetur  
<br>  
25 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. <em>At vero eos et  
accusam et justo duo dolores</em> et ea rebum. Stet clita kasd  
gubergren, no sea takimata sanctus es. </p>  
27 <h2 class="font-normal font-sans text-h2 mt-[28.8pt] mb-[18pt]">  
Links</h2>  
28 Im folgenden werden Links dargestellt um diese auf ihre Funktion im  
PDF-Dokument zu überprüfen: <br>  
29 <a class="underline" href="#test">Ich bin ein interner Link und fü  
hre zur Überschrift "Text im Blocksatz und Auto-Hyphens"</a><br>  
30 <a class="underline" href="https://www.example.com">Ich bin ein  
externer Link und führe zur Website "www.example.com"</a>  
31 <h2 class="font-normal font-sans text-h2 mt-[28.8pt] mb-[18pt]">
```

```

Tabelle</h2>
32   <table class="block-table w-[95%] mx-auto border-collapse">
33     <thead class="">
34       <tr class="border border-solid border-black">
35         <th class="border border-solid border-black px-2 font-bold
36           bg-gray-200 text-left">Erste Spalte</th>
37         <th class="border border-solid border-black px-2 font-bold
38           bg-gray-200 text-left">Zweite Spalte</th>
39     </thead>
40     <tbody>
41       <tr>
42         <td class="border border-solid border-black px-2">Text</td>
43         <td class="border border-solid border-black px-2">Lorem ipsum
44           dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod
45           tempor invidunt ut labore et dolore magna aliquyam erat, sed diam
46           voluptua.</td>
47       </tr>
48       <tr>
49         <td class="border border-solid border-black px-2">Noch mehr Text</
50           td>
51         <td class="border border-solid border-black px-2"> At vero eos et
52           accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren,
53           no sea takimata sanctus est Lorem ipsum dolor sit amet.</td>
54       </tr>
55     </tbody>
56   </table>
57
58   <h2 class="font-normal font-sans text-h2 mt-[28.8pt] mb-[18pt]">Bild
59   </h2>
60   <figure></figure>
62   Photo by <a href="https://unsplash.com/@stp_com?utm_content=
63   creditCopyText&utm_medium=referral&utm_source=unsplash">Tim
64   Peterson</a> on <a href="https://unsplash.com/photos/
65   a-view-of-the-grand-canyon-from-the-top-of-a-mountain--j0ic-c0jk0?
66   utm_content=creditCopyText&utm_medium=referral&utm_source=
67   unsplash">Unsplash</a>
68   <h2 class="font-normal font-sans text-h2 mt-[28.8pt] mb-[18pt]">Text
69   im Blocksatz</h2>
70   Im nun folgenden wird ein Text im Blocksatz dargestellt:

```

```
55 <div class="text-justify">  
56     Nam liber tempor cum soluta nobis eleifend option congue nihil  
57     imperdiet doming id quod mazim placerat facer possim assum. Lorem  
58     ipsum dolor sit amet, consectetuer adipiscing elit, sed diam nonummy  
59     nibh euismod tincidunt ut laoreet dolore magna aliquam erat volutpat.  
60     Ut wisi enim ad minim veniam, quis nostrud exerci tation ullamcorper  
61     suscipit lobortis nisl ut aliquip ex ea commodo consequat.  
62 </div>  
63 <h2 id="test" class="font-normal font-sans text-h2 mt-[28.8pt] mb-[18pt]">Text im Blocksatz und Auto-Hyphens</h2>  
64     Im nun folgenden wird ein weiterer Text im Blocksatz und der  
65     Verwendung von Wortumbrüchen am Zeilenende dargestellt:  
66 <p class="text-justify hyphens-auto">  
67     At vero eos et accusam et justo duo dolores et ea rebum. Stet  
68     clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor  
69     sit amet. Lorem ipsum dolor sit amet, consetetur. Duis autem vel eum  
70     iriure dolor in hendrerit in vulputate velit esse molestie consequat,  
71     vel illum dolore eu feugiat nulla facilisis. At vero eos et  
72     accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren,  
73     no sea takimata sanctus est Lorem ipsum dolor sit amet. Lorem ipsum  
74     dolor sit amet, consetetur  
75 </p>  
76 </div>  
77 </section>  
78 </main>  
79  
80 <!-- Footer mit Downloadbuttons und JS für die Übersichtlichkeit  
81     ausgelassen. Die angewendeten Funktionsaufrufe finden sich im Anhang  
82     der Resultate -->  
83  
84 </body>  
85 </html>
```

Anhang C Generierte PDF-Dokumente zu Kapitel 3

C.I Druck-Dialog im Browser

C.I.1 Mit Voreinstellungen - Google Chrome Desktop

Einstellungen: Kopf- und Fußzeilen: aktiviert; Hintergrundgrafiken: deaktiviert

URL⁴⁴: <https://master.benediktengel.de/generierung>

Test Generierung Webseite

Diese Seite stellt eine Test-Webseite für die Prüfung der Möglichkeiten und Technologien zur Generierung von PDF-Dokumenten aus Webinhalten mit HTML-Markup und CSS-Layout dar. Diese Seite verwendet unterschiedliche Schriftarten und viele verschiedene Elemente, welche in Webinhalten Verwendung finden können, aus welchen zudem PDF-Dokumente generiert werden sollen.

Textparagraphen mit unterschiedlichen Inline-Darstellungen

Der folgende Blindtext nutzt verschiedene Inline-Gestaltungsmöglichkeiten von Text, wie kursive Texte, Fetttexte und Texte mit Umrandung oder Hintergründen:

Lorem ipsum dolor sit amet, consectetur adipisicing elit, 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, consecetur 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, consecetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua.*

kasd gubergren, no sea takimata sanctus est. Lorem ipsum dolor sit amet.

Ut wisi enim ad minim veniam, quis nostrud exerci tation ullamcorper suscipit lobortis nisl ut aliquip ex ea commodo consequat. Duis autem vel eum iriure dolor in hendrerit in vulputate velit esse molestie consequat, vel illum dolore eu feugiat nulla facilisis at vero eros et accusan et justo odio dignissim qui blanditi praesent luptatum zzril delenit augue dls dolore te fenguit nulla facilisi. 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, consecetur

Lorem ipsum dolor sit amet, consectetur adipisicing elit, 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, consecetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua.

⁴⁴Zugangsdaten für den Zugriff:

Benutzername: master Passwort: Thesis24

<p>31.05.2024, 18:02</p> <p>Clieiszeitige Generierung von PDF-Dokumenten aus Webinhalten mit HTML-Markup und CSS Layout</p> <h2>Text im Blocksatz</h2> <p>Im nun folgenden wird ein Text im Blocksatz dargestellt: Nam liber tempor cum solita nobis eleifend option congue nisl imperdiet doming id quod mazim placerat facer possim assum. Lorem ipsum dolor sit amet, consectetuer adipiscing elit, sed diam nonummy nibh euismod tincidunt ut laoreet dolore magna aliquam erat volutpat. Ut wisi enim ad minim veniam, quis nostrud exerci tation ullamcorper suscipit lobortis nisl ut aliquip ex ea commodo consequat.</p> <h2>Text im Blocksatz und Auto-Hyphens</h2> <p>Im nun folgenden wird ein weiterer Text im Blocksatz und der Verwendung von Wortumbrüchen am Zeileende dargestellt: At vero eos et accusam et justo duo dolores et ea rebum. Sint clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet. Lorem ipsum dolor sit amet, consecetur. Duis autem vel eum iriure dolor in hendrerit in vulputate velit esse molestie consequat, vel illum dolore en fengiat nulla facilisis. At vero eos et accusam et justo duo dolores et ea rebum. Sint clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet. Lorem ipsum dolor sit amet, consecetur</p>	<p>31.05.2024, 18:02</p> <p>Clieiszeitige Generierung von PDF-Dokumenten aus Webinhalten mit HTML-Markup und CSS Layout</p> <p><i>At vero eos et accusam et justo duo dolores et ea rebum. Sint clita kasd gubergren, no sea takimata sanctus es.</i></p> <h2>Links</h2> <p>Im nun folgenden werden Links dargestellt, um diese auf ihre Funktion im PDF-Dokument zu überprüfen:</p> <p>Ich bin ein interner Link und führe zur Überschrift "Text im Blocksatz und Auto-Hyphens" Ich bin ein externer Link und führe zur Website "www.example.com"</p>
---	--

<p>31.05.2024, 18:02</p> <p>Clieiszeitige Generierung von PDF-Dokumenten aus Webinhalten mit HTML-Markup und CSS Layout</p> <h2>Text im Blocksatz</h2> <p>Im nun folgenden wird ein Text im Blocksatz dargestellt: Nam liber tempor cum solita nobis eleifend option congue nisl imperdiet doming id quod mazim placerat facer possim assum. Lorem ipsum dolor sit amet, consectetuer adipiscing elit, sed diam nonummy nibh euismod tincidunt ut laoreet dolore magna aliquam erat volutpat. Ut wisi enim ad minim veniam, quis nostrud exerci tation ullamcorper suscipit lobortis nisl ut aliquip ex ea commodo consequat.</p> <h2>Text im Blocksatz und Auto-Hyphens</h2> <p>Im nun folgenden wird ein weiterer Text im Blocksatz und der Verwendung von Wortumbrüchen am Zeileende dargestellt: At vero eos et accusam et justo duo dolores et ea rebum. Sint clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet. Lorem ipsum dolor sit amet, consecetur. Duis autem vel eum iriure dolor in hendrerit in vulputate velit esse molestie consequat, vel illum dolore en fengiat nulla facilisis. At vero eos et accusam et justo duo dolores et ea rebum. Sint clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet. Lorem ipsum dolor sit amet, consecetur</p>	<p>31.05.2024, 18:02</p> <p>Clieiszeitige Generierung von PDF-Dokumenten aus Webinhalten mit HTML-Markup und CSS Layout</p> <p><i>At vero eos et accusam et justo duo dolores et ea rebum. Sint clita kasd gubergren, no sea takimata sanctus es.</i></p> <h2>Links</h2> <p>Im folgenden werden Links dargestellt, um diese auf ihre Funktion im PDF-Dokument zu überprüfen:</p> <p>Ich bin ein interner Link und führe zur Überschrift "Text im Blocksatz und Auto-Hyphens" Ich bin ein externer Link und führe zur Website "www.example.com"</p>	<p>31.05.2024, 18:02</p> <p>Clieiszeitige Generierung von PDF-Dokumenten aus Webinhalten mit HTML-Markup und CSS Layout</p> <h2>Links</h2> <p>Im folgenden werden Links dargestellt, um diese auf ihre Funktion im PDF-Dokument zu überprüfen:</p> <p>Ich bin ein interner Link und führe zur Überschrift "Text im Blocksatz und Auto-Hyphens" Ich bin ein externer Link und führe zur Website "www.example.com"</p>												
<p>31.05.2024, 18:02</p> <p>Clieiszeitige Generierung von PDF-Dokumenten aus Webinhalten mit HTML-Markup und CSS Layout</p> <h2>Tabelle</h2> <p>Im nun folgenden wird eine Tabelle dargestellt:</p> <table border="1"> <thead> <tr> <th>Erste Spalte</th> <th>Zweite Spalte</th> </tr> </thead> <tbody> <tr> <td>Text</td> <td> Lorem ipsum dolor sit amet, consecetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua.</td> </tr> <tr> <td>Noch mehr Text</td> <td>At vero eos et accusam et justo duo dolores et ea rebum. Sint clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet.</td> </tr> </tbody> </table>	Erste Spalte	Zweite Spalte	Text	Lorem ipsum dolor sit amet, consecetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua.	Noch mehr Text	At vero eos et accusam et justo duo dolores et ea rebum. Sint clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet.	<p>31.05.2024, 18:02</p> <p>Clieiszeitige Generierung von PDF-Dokumenten aus Webinhalten mit HTML-Markup und CSS Layout</p> <h2>Tabelle</h2> <p>Im nun folgenden wird eine Tabelle dargestellt:</p> <table border="1"> <thead> <tr> <th>Erste Spalte</th> <th>Zweite Spalte</th> </tr> </thead> <tbody> <tr> <td>Text</td> <td> Lorem ipsum dolor sit amet, consecetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua.</td> </tr> <tr> <td>Noch mehr Text</td> <td>At vero eos et accusam et justo duo dolores et ea rebum. Sint clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet.</td> </tr> </tbody> </table>	Erste Spalte	Zweite Spalte	Text	Lorem ipsum dolor sit amet, consecetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua.	Noch mehr Text	At vero eos et accusam et justo duo dolores et ea rebum. Sint clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet.	<p>31.05.2024, 18:02</p> <p>Clieiszeitige Generierung von PDF-Dokumenten aus Webinhalten mit HTML-Markup und CSS Layout</p> <h2>Bild</h2> 
Erste Spalte	Zweite Spalte													
Text	Lorem ipsum dolor sit amet, consecetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua.													
Noch mehr Text	At vero eos et accusam et justo duo dolores et ea rebum. Sint clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet.													
Erste Spalte	Zweite Spalte													
Text	Lorem ipsum dolor sit amet, consecetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua.													
Noch mehr Text	At vero eos et accusam et justo duo dolores et ea rebum. Sint clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet.													

C.I.2 Mit angepassten Einstellungen - Google Chrome Desktop

Einstellungen: Kopf- und Fußzeilen deaktiviert; Hintergrundgrafiken aktiviert

Test Generierungswebseite

Links

At vero eos et accusam et justo duo dolores et ea rebum. Sint clita kasd gubergren, no sea takimata sanctus es.

Im folgenden werden Links dargestellt um diese auf ihre Funktion im PDF-Dokument zu überprüfen:

Ich bin ein interner Link und führt zur Überschrift "Text im Blocksatz und Auto-Hyphens"
Ich bin ein externer Link und führt zur Webseite "www.example.com"

Tabelle

Erste Spalte	Zweite Spalte
Text	<p>LOREM ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor eiusmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua.</p> <p>Noch mehr Text</p> <p>At vero eos et accusam et justo duo dolores et ea rebum. Sint clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet.</p>

Textparagraphen mit unterschiedlichen Inline-Darstellungen

Diese Seite stellt eine Test-Webseite für die Prüfung der Möglichkeiten und Technologien zur Generierung von PDF-Dokumenten aus Webinhalten mit HTML-Markup und CSS-Layout dar. Diese Seite verwendet unterschiedliche Schriftarten und viele verschiedene Elemente, welche in Webinhalten Verwendung finden können, aus welchen zudem PDF-Dokumente generiert werden sollen.

Der folgende Blindtext nutzt verschiedene Inline-Gestaltungsmöglichkeiten von Text, wie kursive Texte, Fetttexte und Texte mit Umrandung oder Hintergründen:

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. Sint clita kasd gubergren, no sea takimata sanctus est 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. Sint clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet.

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. Sint clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet.

Ut wisi enim ad minim veniam, quis nostrud exerci tation ullamcorper suscipit lobortis nisl ut aliquip ex ea commodo consequat. Duis autem vel eum iriure dolor in hendrerit in vulputate velit esse molestie consequat, vel illum dolore eu feugiat nulla facilisis at vero eros et accusam et justo odio dignissim qui blandit praesent ttipatum zzril delenit augue dhis dolore te fengait nulla facilisi. At vero eos et accusam et justo duo dolores et ea rebum. Sint clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet. 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. Sint clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua.

Tabelle

Im folgenden werden Links dargestellt um diese auf ihre Funktion im PDF-Dokument zu überprüfen:

Ich bin ein interner Link und führt zur Überschrift "Text im Blocksatz und Auto-Hyphens"
Ich bin ein externer Link und führt zur Webseite "www.example.com"

Image



Bild

Photo by Tim Peterson on Unsplash

Text im Blocksatz

Im nun folgenden wird ein Text im Blocksatz dargestellt:

Nam liber tempor cum soluta nobis eleifend option congue nihil imperdiet doming id quod mazim placerat facer possim assum. Lorem ipsum dolor sit amet, consecetur adipiscing elit, sed diam nonumy nibh euismod tincidunt ut laoreet dolore magna aliquan erat volutpat. Ut wisi enim ad minim veniam, quis nostrud exerci tation ullamcorper suscipit lobortis nisl ut aliquip ex ea commodo consequat.

Text im Blocksatz und Auto-Hyphens

Im nun folgenden wird ein weiterer Text im Blocksatz und der Verwendung von Worttrennrichen am Zeilenende dargestellt:

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, consecetur. Duis autem vel eum iriure dolor in hendrerit in vulputate velit esse molestie consequat, vel illum dolore eu feugiat nulla facilisis. 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, consecetur

C.I.3 Safari iOS

Keine Einstellungen möglich

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

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 vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est.

Links

Im folgenden werden Links dargestellt um diese auf ihre Funktion im PDF-Dokument zu überprüfen:

Ich bin ein interner Link und führe zur Überschrift "Text im Blocksatz und Autohyphens"

Ich bin ein externer Link und führe zur Website "www.example.com"

Tabelle

Im folgenden werden Links dargestellt um diese auf ihre Funktion im PDF-Dokument zu überprüfen:

Erste Spalte	Zweite Spalte
Text	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.
Noch mehr Text	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.

Bild

https://master.benediktengel.de/generierung
31.05.24, 18:10
Seite 2 von 4

31.05.24, 18:10
https://master.benediktengel.de/generierung
Seite 1 von 4

111



Photo by Tim Peterson on Unsplash

Text im Blocksatz

Im nun folgenden wird ein Text im Blocksatz dargestellt:
Nam liber tempor cum soluta nobis eleifend option congue nihil imperdiet doming id
quod nazim placerat facer possim assum. Lorem ipsum dolor sit amet, consecetur
adipiscing elit, sed diam nonummy nibh euismod tincidunt ut laoreet dolore magna
aliquam erat volutpat. Ut wisi enim ad minim veniam, quis nostrud exerci tation
ullamcorper suscipit lobortis nisl ut aliquip ex ea commodo consequat.

Text im Blocksatz und Auto-Hyphens

Im nun folgenden wird ein weiterer Text im Blocksatz und der Verwendung von
Worttrennrichen am Zeilenende dargestellt:
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, consecetur. Duis autem vel eum iriure dolor in hendrerit in vulputate velit

esse molestie consequat, vel illum dolore eu feugiat nulla facilisis. At vero eos et ac-
cusam 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, consecetur

C.I.4 Google Chrome iOS

Keine Einstellungen möglich

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 es.

Links

Im folgenden werden Links dargestellt um diese auf ihre Funktion im PDF-Dokument zu überprüfen:

Ich bin ein interner Link und führe zur Überschrift "Text im Blocksatz und Auto-Hyphens"

Ich bin ein externer Link und führe zur Website "www.example.com"

Tabelle

Erste Spalte	Zweite Spalte
Text	
Noch mehr Text	

Bild

Diese Seite stellt eine Test-Webseite für die Prüfung der Möglichkeiten und Technologien zur Generierung von PDF-Dokumenten aus Webinhalten mit HTML-Markup und CSS-Layout dar. Diese Seite verwendet unterschiedliche Schriftarten und viele verschiedene Elemente, welche in Webinhalten Verwendung finden können, aus welchen zudem PDF-Dokumente generiert werden sollen.

Textparagraphen mit unterschiedlichen Inline-Darstellungen

Der folgende Blindtext nutzt verschiedene Inline-Gestaltungsmöglichkeiten von Text, wie kursive Texte, Fetttexte und Texte mit Umrandung oder Hintergründen:

Loren 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. Loren 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. Loren 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. Loren 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. Loren ipsum dolor sit amet.

Ut wisi enim ad minim veniam, quis nostrud exerci tation ullamcorper suscipit lobortis nisl ut aliquip ex ea commodo consequat. Duis autem vel eum iriure dolor in hendrerit in vulputate velit esse molestie consequat, vel illum dolore eu feugiat nulla facilisis at vero eros et accumsan et iusto odio dignissim qui blanditi praesent luptatum zzril delenit augue duis dolore te feugait nulla facilisi. 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. Loren ipsum dolor sit amet, consetetur

Loren ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor

113



Photo by Tim Peterson on Unsplash

Text im Blocksatz

Im nun folgenden wird ein Text im Blocksatz dargestellt:

Nam liber tempor cum soluta nobis eleifend option congue nihil imperdiet doming id quod
mazim placerat facer possim assum. Lorem ipsum dolor sit amet, consecetur adipiscing
elit, sed diam nonummy nibh euismod tincidunt ut laoreet dolore magna aliquam erat
voluptat. Ut wisi enim ad minim veniam, quis nostrud exerci tation ullamcorper suscipit
lobortis nisl ut aliquip ex ea commodo consequat.

Text im Blocksatz und Auto-Hyphens

Im nun folgenden wird ein weiterer Text im Blocksatz und der Verwendung von
Wortumbrüchen an Zeilenende dargestellt:

At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea
takinata sanctus est Lorem ipsum dolor sit amet. Lorem ipsum dolor sit amet, consecetur.
Duis autem vel eum iuriare dolor in hendrerit in vulputate velit esse molestie consequat, vel
illum dolore eu feugiat nulla facilisis. At vero eos et accusam et justo duo dolores et ea re-
bum. Stet clita kasd gubergren, no sea takinata sanctus est Lorem ipsum dolor sit amet.
Lorem ipsum dolor sit amet, consecetur

C.II jsPDF

C.II.1 Ohne Voreinstellungen

URL⁴⁵: <https://master.benediktengel.de/generierung?type=jspdf&option=1>

Funktionsaufruf:

```
1 const doc = new jsPDF();
2 doc.html(document.body, {
3   callback: function (doc) { doc.save("output.pdf"); }
4});
```

Test Generierungswebsite

Diese Seite stellt eine Test-Website für die Prüfung der Möglichkeiten und Technologien zur Generierung von PDF-Dokumenten aus Webinhalten mit HTML-, Markup und CSS-Layout dar. Diese Seite verwendet unterschiedliche Schriftfarben und viele verschiedene Elemente, welche in Webinhalten Verwendung finden können, aus welchen zudem PDF-Dokumente generiert werden sollen.

Textparagraphen mit unterschiedlichen Inline-Darstellungen

Der folgende Blindtext nutzt verschiedene Inline-Gestaltungsmöglichkeiten von Text, wie kursive Texte, Fetttexte und Texte mit Umrandung oder Hintergrundfarbe:

Tempor invidunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercit tation ullamcorper suscipit lobortis nisl ut aliquip ex ea commodo consequat. Duis autem vel eum iriure dolor in hendrerit in vulputate velit esse molestie consequat, vel illum dolore eu fugiat nulla facilisis at vero eos et accusam et justo odio dignissim qui blandit praesent luptatum zzril delenit augue duis dolore te feugait nulla facilisi. At vero eos et accusam et justo duo dolores et ea rebum. Sunt clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet. Lorem ipsum dolor sit amet, consecetur adipiscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliqua. Ut vero eos et accusam et justo duo dolores et ea rebum. Sunt clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet. Lorem ipsum dolor sit amet, consecetur adipiscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliqua. Ut vero eos et accusam et justo duo dolores et ea rebum. Sunt clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet. Lorem ipsum dolor sit amet, consecetur adipiscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliqua. Ut vero eos et accusam et justo duo dolores et ea rebum. Sunt clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet. Lorem ipsum dolor sit amet, consecetur adipiscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliqua. Ut vero eos et accusam et justo duo dolores et ea rebum. Sunt clita kasd gubergren, no sea takimata sanctus es.

⁴⁵ Zugangsdaten für den Zugriff:

Benutzername: master Passwort: Thesis24

<p>Links</p> <p>Im nun folgenden wird ein Text im Blocksatz dargestellt :</p> <p>Nam liber tempor cum soluta nobis eleifend option congue nihil imperdiet doming id quod mazim placerat facer possim assum. Lorem ipsum dolor sit amet, consecetur adipiscing elit, sed diam nonummy nibh euismod tincidunt ut laoreet dolore magna aliquan erat volupat. Ut wisi enim ad minim veniam, quis nostrud exercitatio ullamcorper suscipit lobortis nisl ut aliquip ex ea commodo consequat.</p> <p>Text im Blocksatz und Auto-Hyphens</p> <p>Im nun folgenden wird ein weiterer Text im Blocksatz und der Verwendung von Wortumbrüchen am Zeilende dargestellt :</p> <p>At vero eos et accusam et justo duo dolores et ea rebum. Sint clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet. Lorem ipsum dolor sit amet, consecetur .Duis autem vel eum iriure dolor in hendrerit in vulputate velit esse moles tie consequat , vel illum dolore eu feugiat nulla facilisis. At vero eos et accusam et justo duo dolores et ea rebum. Sint clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet. Lorem ipsum dolor sit amet, consecetur</p>	<p>Im folgenden werden Links dargestellt um diese auf ihre Funktion im PDF -Dokument zu überprüfen :</p> <p>Ich bin ein interner Link und führe zur Überschrift "Text im Blocksatz und Auto-Hyphens"</p> <p>Ich bin ein externer Link und führe zur Website "www.example.com"</p>
---	---

<p>Tabelle</p> <p>Im nun folgenden wird ein weiterer Text im Blocksatz und der Verwendung von Wortumbrüchen am Zeilende dargestellt :</p> <p>At vero eos et accusam et justo duo dolores et ea rebum. Sint clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet. Lorem ipsum dolor sit amet, consecetur .Duis autem vel eum iriure dolor in hendrerit in vulputate velit esse moles tie consequat , vel illum dolore eu feugiat nulla facilisis. At vero eos et accusam et justo duo dolores et ea rebum. Sint clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet.</p>	<table border="1"> <thead> <tr> <th>Erste Spalte</th><th>Zweite Spalte</th></tr> </thead> <tbody> <tr> <td>Text</td><td> </td></tr> <tr> <td>Noch mehr Text</td><td> </td></tr> </tbody> </table>	Erste Spalte	Zweite Spalte	Text		Noch mehr Text	
Erste Spalte	Zweite Spalte						
Text							
Noch mehr Text							



Photo by Tim Peterson on Unsplash

Text im Blocksatz

C.II.2 Mit angegebenen Schriften

URL⁴⁶: <https://master.benediktengel.de/generierung?type=jsPDF&option=2>

Funktionsaufruf:

```

1 const doc = new jsPDF();
2 doc.html(document.body, {
3   callback: function (doc) { doc.save("output.pdf"); },
4   fontFaces: [
5     {
6       family: 'Computer Modern Serif',
7         style: 'normal',
8         weight: '400',
9         stretch: 'normal',
10        src: [{url: "./assets/fonts/computer-modern/Serif/cmunrm.ttf",
11      format:'truetype'}]
12    },
13      family: 'Computer Modern Serif',
14      style: 'normal',
15      weight: '600',
16      stretch: 'normal',
17      src: [{url: "./assets/fonts/computer-modern/Serif/cmunbx.ttf", format
18      : 'truetype'}]
19    },
20      family: 'Computer Modern Serif',
21      style: 'italic',
22      weight: '400',
23      stretch: 'normal',
24      src: [{url: "./assets/fonts/computer-modern/Serif/cmunti.ttf", format
25      : 'truetype'}]
26    },
27      family: 'Computer Modern Sans',
28      style: 'normal',
29      weight: '600',
30      stretch: 'normal',
31      src: [{url: "./assets/fonts/computer-modern/Sans/cmunsx.ttf", format:
32      'truetype'}]
33  ],
34 });

```

⁴⁶Zugangsdaten für den Zugriff:

Benutzername: master Passwort: Thesis24

Links

Im folgenden werden Links dargestellt um diese auf ihre Funktion im PDF-Dokument zu überprüfen:
Ich bin ein interner Link und führe zur Überschrift „Text im Blocksatz und Antizypationspfeil.“
Ich bin ein externer Link und führe zur Website „www.example.com“

Tabell

Erste Spalte	Zweite Spalte
Text	<p>Loreum ipsum dolor sit amet, consectetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua.</p>
Noch mehr Text	<p>At vero eos et accusam et justo duo dolores et ea rebum. Siet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum</p> <p>1. 2. 3. 4. 5. 6. 7. 8. 9. 10.</p>

B
1



Photo by Tim Peterson on Unsplash

Text im Blocksatz

Test Generierungswebseite

Diese Seite stellt eine Test-Webseite für die Prüfung der Möglichkeiten und Technologien zur Generierung von PDF-Dokumenten aus Webinhalten mit HTML-Markup und CSS-Layout dar. Diese Seite verwendet unterschiedliche Schriftarten und viele verschiedene Elemente, welche in Webinhalten Verwendung finden können, aus welchen zudem PDF-Dokumente generiert werden sollen.

Textparagraphen mit unterschiedlichen Inline-Darstellungen

Der folgende Blindtext nutzt verschiedene Inline-Gestaltungsmöglichkeiten von Text, wie kursive Texte, Fetttexte und Texte mit Umrundung oder Hintergründen:

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. 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

Uf, wisi enim ad minim veniam, quis nostrud exerci tation ullamcorper suscipit lobortis nisl ut aliquip ex ea commodo consequat. Duis autem vel eum iriure dolor in hendrerit in vulputate velit esse molestie consequat, vel illum dolore eu feugiat nulla facilisis at vero eros et accumsan et iusto odio dignissim qui blanditi present luptatum zzril delenit angue dñis dolore te feugiat nulla facilisi. At vero eos et accusam et justo duo dolores et ea rebum. Sicut clita kasd gubergren, no sea takimata sanctus est. Lorem ipsum dolor sit amet. Lorem ipsum dolor sit amet, consetetur 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 es.

In nun folgenden wird ein Text im Blocksatz dargestellt:
Nam liber tempor cum soluta nobis eleifend option congue nihil imperdiet doming id
quod mazim placerat facer possim assum. Lorem ipsum dolor sit amet, consectetuer
adipiscing elit, sed diam nonummy nibh euismod tincidunt ut laoreet dolore magna
aliquam erat volutpat. Ut wisi enim ad minim veniam, quis nostrud exerci tation
ullamcorper suscipit lobortis nisl ut aliquip ex ea commodo consequat.

Text im Blocksatz und Auto-Hyphens

Im nun folgenden wird ein weiterer Text im Blocksatz und der Verwendung von
Wortumbrüchen am Zeilenende dargestellt:
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. Duis autem vel eum iriure dolor in hendrerit in vulputate velit esse moles
tie consequat, vel illum dolore eu feugiat nulla facilisis. 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

C.II.3 Mit angegebenen Schriften und Textumbrüchen

URL⁴⁷: <https://master.benediktengel.de/generierung?type=jsPDF&option=3>

Funktionsaufruf:

```

1 const doc = new jsPDF();
2 doc.html(document.body, {
3   callback: function (doc) { doc.save("output.pdf"); },
4   fontFaces: [
5     {
6       family: 'Computer Modern Serif',
7         style: 'normal',
8         weight: '400',
9         stretch: 'normal',
10        src: [{url:"./assets/fonts/computer-modern/Serif/cmunrm.ttf",
11      format:'truetype'}]
12    },
13    {
14      family: 'Computer Modern Serif',
15      style: 'normal',
16      weight: '600',
17      stretch: 'normal',
18      src: [{url:"./assets/fonts/computer-modern/Serif/cmunbx.ttf", format
19        : 'truetype'}]
20    },
21    {
22      family: 'Computer Modern Serif',
23      style: 'italic',
24      weight: '400',
25      stretch: 'normal',
26      src: [{url:"./assets/fonts/computer-modern/Serif/cmunti.ttf", format
27        : 'truetype'}]
28    },
29    {
30      family: 'Computer Modern Sans',
31      style: 'normal',
32      weight: '600',
33      stretch: 'normal',
34      src: [{url:"./assets/fonts/computer-modern/Sans/cmunsx.ttf", format:
35        'truetype'}]
36    },
37    autoPaging = 'text',
38  });

```

⁴⁷Zugangsdaten für den Zugriff: Benutzername: master Passwort: Thesis24

Links

Im folgenden werden Links dargestellt, um diese auf ihre Funktion im PDF-Dokument zu überprüfen:

Ich bin ein interner Link und führe zur Überschrift "Text im Blocksatz und Auto-

Ich bin ein externer Link und führe zur Website "www.example.com"
Hyphens!"

Tabelle

Erste Spalte	Zweite Spalte
Text	<p>Loreum ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua.</p>
Noch mehr Text	<p>At vero eos et accusamus et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet.</p>

1



Photo by Tim Peterson on Unsplash

Text im Blocksatz

Test Generierungswebseite

Diese Seite stellt eine Test-Webseite für die Prüfung der Möglichkeiten und Technologien zur Generierung von PDF-Dokumenten aus Webinhalten mit HTML-Markup und CSS-Layout dar. Diese Seite verwendet unterschiedliche Schriftarten und viele verschiedene Elemente, welche in Webinhalten Verwendung finden können, aus welchen zudem PDF-Dokumente generiert werden sollen.

Textparagraphen mit unterschiedlichen Inline-Darstellungen

Uf. wisi enim ad minim veniam, quis nostrud exerci tation ullamcorper suscipit lobortis nisl ut aliquip ex ea commodo consequat. Duis autem vel eum iriure dolor in hendrerit **in vulputate velit esse molestie consequat**, **vel illum dolore eu feugiat nulla facilisis at vero eros et accumsan et iusto odio dignissim qui blandit present luptatum zzril delenit angue duis dolore et ea rebum.** Siet clita kasd gubergren, no sea takimata sanctus est. Lorem ipsum dolor sit amet. Lorem ipsum dolor sit amet, consetetur 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 es.

In nun folgenden wird ein Text im Blocksatz dargestellt:

Nam liber tempor cum soluta nobis eleifend option congue nihil imperdiet doming id quod mazim placerat facer possim assum. Lorem ipsum dolor sit amet, consecetur adipiscing elit, sed diam nonummy nibh euismod tincidunt ut labore magna aliquam erat volutpat. Ut wisi enim ad minim veniam, quis nostrud exerci tation ullamcorper suscipit lobortis nisl ut aliquip ex ea commodo consequat.

Text im Blocksatz und Auto-Hyphens

Im nun folgenden wird ein weiterer Text im Blocksatz und der Verwendung von Worttrennrichen am Zeilenende dargestellt:

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, consecetur. Duis autem vel eum iriure dolor in hendrerit in vulputate velit esse moles tie consequat, vel illum dolore eu feugiat nulla facilisis. 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, consecetur

C.II.4 Fehlerhafte Positionierung durch autoPaging

URL⁴⁸: <https://master.benediktengel.de/generierung-fehlerhaft-positionierung?type=jspdf&option=3>

Funktionsaufruf: Gleich wie bei Anhang C.II.3

aliquam erat voluptat. Ut wisi enim ad minim veniam, quis nostrud exerci tation
ullancorper suscipit tortoritis nisl ut aliquip ex ea commodo consequat.
Lorem ipsum dolor sit amet, consecetur adipisicing elit, 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,
consecetur adipisicing elit, sed diam nonumy eirmod tempor invidunt ut labore et
dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo quo
dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus es.
Nam liber tempor cum soluta nobis eleifend option congue nihil imperid, doming id
quod maxin placerat facer possim assun. Lorem ipsum dolor sit amet, consecetur
adipisicing elit, sed diam nonumy nibh euismod tincidunt ut laoreet dolore magna
aliquam erat voluptat. Ut wisi enim ad minim veniam, quis nostrud exerci tation
ullancorper suscipit tortoritis nisl ut aliquip ex ea commodo consequat.

⁴⁸Zugangsdaten für den Zugriff: Benutzername: master Passwort: Thesis24

C.II.5 Generierung auf einem Smartphone (mobile Viewport)

Funktionsaufruf: Gleich wie bei Anhang C.II.3

Test Generierungswebseite

Textparagraphen mit unterschiedlichen Inline-Darstellungen

Diese Seite stellt eine Test-Webseite für die Prüfung der Möglichkeiten und Technologien zur Generierung von PDF-Dokumenten aus Webinhalten mit HTML-Markup und CSS-Layout dar. Diese Seite verwendet unterschiedliche Schriftarten und viele verschiedene Elemente, welche in Webinhalten Verwendung finden können, aus welchen zudem PDF-Dokumente generiert werden sollen.

Links

Im folgenden werden Links dargestellt um diese auf ihre Funktion im PDF-Dokument zu überprüfen:

- Ich bin ein interner Link und führe zur Überschrift "Text im Blocksatz und Auto-Hyphens".
- Ich bin ein externer Link und führe zur Website "www.example.com".

Tabelle

Erste Spalte	Zweite Spalte
Text	<p>Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed diam nonummy enimod tempor incidunt ut labore et dolore magna aliquyam erat, sed diam voluptua.</p>
Noch mehr Text	<p>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.</p>

Bild



Photo by Tim Botsford on Unsplash

Text im Blocksatz

Im nun folgenden wird ein Text im Blocksatz dargestellt:

Nam liber tempor cum soluta nobis eleifend option congue nihil imperdiet doming id quod mazim placerat facer possim assum. Lorem ipsum dolor sit amet, consecetur adipiscing elit, sed diam nonummy nibh euismod tincidunt ut laoreet dolore magna aliquan erat volutpat. Ut wisi enim ad minim veniam, quis nostrud exerci tation ullamcorper suscipit lobortis nisl ut aliquip ex ea commodo consequat.

Text im Blocksatz und Auto-Hyphens

Im nun folgenden wird ein weiterer Text im Blocksatz und der Verwendung von Wortumbrüchen am

Links

Im folgenden werden Links dargestellt um diese auf ihre Funktion im PDF-Dokument zu überprüfen:

Ich bin ein interner Link und fühne zur Überschrift "Text im Blocksatz und Auto-Hyphens"

Zeilenden dargestellt.

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, consecetur. Duis autem vel eum iriure dolor in hendrerit in vulputate velit esse molestie consequat, vel illum dolore eu feugiat nulla facilisis. 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, consecetur

C.III html2pdf.js

C.III.1 Ohne Voreinstellungen

URL⁴⁹: <https://master.benediktengel.de/generierung?type=html2pdf&option=1>

Funktionsaufruf:

```
1 html2pdf(document.querySelector('main'), {
2   filename: 'html2pdf.pdf',
3 }) ;
```

Test Generierungswebsite

Diese Seite stellt eine Test-Webseite für die Prüfung der Möglichkeiten und Technologien zur Generierung von PDF-Dokumenten aus Webinhalten mit HTML-Markup und CSS-Layout dar. Diese Seite verwendet unterschiedliche Schriftarten und viele verschiedene Elemente, welche in Webinhalten Verwendung finden können, aus welchen zudem PDF-Dokumente generiert werden sollen.

Textparagraphen mit unterschiedlichen Inline-Darstellungen

Der folgende Blindtext nutzt verschiedene Inline-Gestaltungsmöglichkeiten von Text, wie kursive Texte, Fetttexte und Texte mit Umrandung oder Hintergründen:

 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. 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. Ut wisi enim ad minim veniam, quis nostrud exerci tation ullamcorper suscipit lobortis nisl ut aliquip ex ea commodo consequat. Duis autem vel eum iriure dolor in hendrerit in vulputate velit esse molestie consequat, vel illum dolore eu feugiat nulla facilisis at vero eros et accumsan et, iusto odio dignissim, qui blandit praesent luptatum zzril delenit augue duis dolore te feingait nulla facilisi. 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. 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 es.

⁴⁹ Zugangsdaten für den Zugriff:

Benutzername: master Passwort: Thesis24

Links

Nam liber tempor cum solita nobis eleifend option congue nibil imperdiet doming id quod nazim placerat facer possim assum. Lorem ipsum dolor sit amet, consecetur adipiscing elit, sed diam nonummy nibh euismod tincidunt ut laoreet dolore magna aliquam erat volutpat. Ut wisi enim ad minim veniam, quis nostrud exerci tation ullamcorper suscipit lobortis nisl ut aliquip ex ea commodo consequat.

Text im Blocksatz und Auto-Hyphens

Im nun folgenden wird ein weiterer Text im Blocksatz und der Verwendung von Wortumbrüchen am Zeilenende dargestellt:
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, consecetur. Duis autem vel eum iriure dolor in hendrerit in vulputate velit esse molestie consequat, vel illum dolore eu feugiat nulla facilisis. 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. Lorem ipsum dolor sit amet, consecetur

Im folgenden werden Links dargestellt um diese auf ihre Funktion im PDF-Dokument zu überprüfen:
Ich bin ein interner Link und führe zur Überschrift "Text im Blocksatz und Auto-Hyphens"
Ich bin ein externer Link und führe zur Website "www.example.com"

Tabelle

Erste Spalte	Zweite Spalte
Text	Lorem ipsum dolor sit amet, consecetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua.
Noch mehr Text	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.



Photo by Tim Peterson on Unsplash

Text im Blocksatz

Im nun folgenden wird ein Text im Blocksatz dargestellt.

C.III.2 Mit maximaler Auflösung

URL⁵⁰: <https://master.benediktengel.de/generierung?type=html2pdf&option=2>

Funktionsaufruf:

```
1 html2pdf(document.querySelector('main'), {  
2   filename: 'html2pdf.pdf',  
3   image: {  
4     type: 'jpeg',  
5     quality: 1  
6   },  
7 }) ;
```

⁵⁰Zugangsdaten für den Zugriff:

Benutzername: master Passwort: Thesis24

Links

Nam liber tempor cum solita nobis eleifend option congue nibil imperdiet doming id quod nazim placerat facer possim assum. Lorem ipsum dolor sit amet, consecetur adipiscing elit, sed diam nonummy nibh euismod tincidunt ut laoreet dolore magna aliquam erat volutpat. Ut wisi enim ad minim veniam, quis nostrud exerci tation ullamcorper suscipit lobortis nisl ut aliquip ex ea commodo consequat.

Text im Blocksatz und Auto-Hyphens

Im nun folgenden wird ein weiterer Text im Blocksatz und der Verwendung von Wortumbrüchen am Zeilenende dargestellt:
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, consecetur. Duis autem vel eum iriure dolor in hendrerit in vulputate velit esse molestie consequat, vel illum dolore eu feugiat nulla facilisis. 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. Lorem ipsum dolor sit amet, consecetur

Im folgenden werden Links dargestellt um diese auf ihre Funktion im PDF-Dokument zu überprüfen:
Ich bin ein interner Link und führe zur Überschrift "Text im Blocksatz und Auto-Hyphens"
Ich bin ein externer Link und führe zur Website "www.example.com"

Tabelle

Erste Spalte	Zweite Spalte
Text	Lorem ipsum dolor sit amet, consecetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua.
Noch mehr Text	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.



Photo by Tim Peterson on Unsplash

Text im Blocksatz

Im nun folgenden wird ein Text im Blocksatz dargestellt.

C.III.3 Webinhalt vor Generierung gescrollt

URL⁵¹: <https://master.benediktengel.de/generierung?type=html2pdf&option=3>

Funktionsaufruf: Gleich wie bei Anhang C.III.2

Test Generierungswebseite

Diese Seite stellt eine Test-Webseite für die Prüfung der Möglichkeiten und Technologien zur Generierung von PDF-Dokumenten aus Webinhalten mit HTML-Markup und CSS-Layout dar. Diese Seite verwendet unterschiedliche Schriftarten und viele verschiedene Elemente, welche in Webinhalten Verwendung finden können, aus welchen zudem PDF-Dokumente generiert werden sollen.

Textparagraphen mit unterschiedlichen Inline-Darstellungen

Der folgende Blindtext nutzt verschiedene Inline-Gestaltungsmöglichkeiten von Text, wie kursive Texte, Fetttexte und Texte mit Umrandung oder Hintergründen:

Erste Spalte

Zweite Spalte

Etiam 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 es. Aliquam 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.

Textparagrahen mit unterschiedlichen Inline-Darstellungen

Etiam 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.

Ut wisi enim ad minim veniam, quis nostrud exerci tation ullamcorper suscipit lobortis nisl ut aliquip ex ea commodo consequat. Duis autem vel eum iriure dolor in hendrerit in vulputate velit esse molestie consequat, vel illum dolore eu feugiat nulla facilisis at vero eros et accumsan et iusto odio dignissim qui blanditi priesent luptatum zzril delenit augue duis dolore te feugait nulla facilisi. 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

Aliquam erat, 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,

Links

Im folgenden werden Links dargestellt um diese auf ihre Funktion im PDF-Dokument zu überprüfen:

[Ich bin ein interner Link und führe zur Überschrift "Text im Blocksatz und Auto-Hyphens"](#)

[Ich bin ein externer Link und führt zur Website "www.example.com"](#)

Erste Spalte	Zweite Spalte
Text	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.
Noch mehr Text	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.

Tabelle

Bild



Photo by Tim Peterson on Unsplash

⁵¹Zugangsdaten für den Zugriff: Benutzername: master Passwort: Thesis24

Text im Blocksatz

Im nun folgenden wird ein Text im Blocksatz dargestellt:
Nam liber tempor cum soluta nobis eleifend option congue nihil imperdiet doming id
quod mazim placerat facer possim assum. Lorem ipsum dolor sit amet, consecetur

C.III.4 Generierung auf einem Smartphone (mobile Viewport)

Funktionsaufruf: Gleich wie bei Anhang C.III.2

Tabelle

Erste Spalte	Zweite Spalte
Text	<p>Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua.</p>
Noch mehr Text	<p>At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergen, no sea takimata sanctus est Lorem ipsum dolor sit amet.</p>

Bild



Photo by Tim Peterson on Unsplash

Text im Blocksatz

Im nun folgenden wird ein Text im Blockssatz dargestellt:

Nam liber tempor cum soluta nobis eleifend option congue nihil imperdiet doming id quod mazim placerat facer possim assum. Lorem ipsum dolor sit amet, consecetur adipiscing elit, sed diam nonumy nihii euismod tincidunt ut laoreet dolore magna aliquam erat volutpat. Ut wisi enim ad minim veniam, quis nostrud exercitATION ullamcorper suscipit lobortis nisl ut aliquip ex ea commodo consequat.

links

Im folgenden werden Links dargestellt um diese auf ihre Funktion im PDF-Dokument zu überprüfen:
Ich bin ein interner Link und führe zur Überschrift "Text im Blocksatz und Auto-Hyphens"
Ich bin ein externer Link und führe zur Website "www.example.com"

Anhang D Konzeption

D.I Funktionale Anforderungen

PDF-Writer

- F001** Die Bibliothek muss fähig sein einen gegebenen boolischen Wert als ein der PDF-Spezifikation entsprechendes *Boolean Object* auszugeben.
- F002** Die Bibliothek muss fähig sein einen gegebenen numerischen Wert als ein der PDF-Spezifikation entsprechendes *Numeric Object* auszugeben.
- F003** Die Bibliothek muss fähig sein einen gegebenen Null Wert als ein der PDF-Spezifikation entsprechendes *Null Object* auszugeben.
- F004** Die Bibliothek muss fähig sein einen gegebenen String Wert als ein der PDF-Spezifikation entsprechendes *String Object* in der litaral Schreibweise auszugeben.
- F005** Die Bibliothek muss fähig sein einen gegebenen String Wert als ein der PDF-Spezifikation entsprechendes *String Object* in der hexadezimal Schreibweise auszugeben.
- F006** Die Bibliothek muss fähig sein ein gegebenes Array von PDF-Objekten als ein der PDF-Spezifikation entsprechendes *Array Object* auszugeben.
- F007** Die Bibliothek muss fähig sein gegebene Key-Value-Paare bestehend aus einem Name Object als Key und einem weiteren PDF-Objekt als Value als ein der PDF-Spezifikation entsprechendes *Dictionary Object* auszugeben.
- F008** Die Bibliothek muss fähig sein einen gegebenen Byte-Stream als ein der PDF-Spezifikation entsprechendes *Stream Object* und etwaige zusätzliche Key-Value-Paare als zugehöriges der PDF-Spezifikation entsprechendes *Stream Dictionary* auszugeben.
- F009** Die Bibliothek muss fähig sein ein gegebenen String als ein der PDF-Spezifikation entsprechendes *Name Object* auszugeben.
- F010** Die Bibliothek muss fähig sein PDF-Objekte sowohl als direkte, als auch als indirekte Objekte auszugeben.
- F011** Die Bibliothek muss fähig sein den PDF-Header einer Datei mit der spezifizierten PDF-Version auszugeben.

- F012** Die Bibliothek muss fähig sein eine Cross-Reference-Tabelle anhand aller indirekten Objekte eines Dokuments auszugeben.
- F013** Die Bibliothek muss fähig sein für indirekte Objekte eine eindeutige Objekt-ID zu vergeben.
- F014** Die Bibliothek muss fähig sein alle indirekten Objekte in Form des Bodys der PDF-Datei auszugeben.
- F015** Die Bibliothek muss fähig sein den Trailer für die PDF-Datei zu erstellen und diesen auszugeben.
- F016** Die Bibliothek muss fähig sein Header, Body, CrossReferenceTabel und Trailer in der richtigen Reihenfolge als Inhalt der PDF-Datei auszugeben.
- F017** Die Bibliothek muss die Fähigkeit bieten einen String anzugeben, welcher als Dokumenten Titel verwendet werden soll.
- F018** Die Bibliothek muss fähig sein einen spezifizierten String als Titel des PDF-Dokuments zu verwenden.
- F019** Die Bibliothek muss die Fähigkeit bieten eine PDF-Version anzugeben, welche für das PDF-Dokument verwendet werden soll.
- F020** Die Bibliothek muss die Fähigkeit bieten einen String anzugeben, welcher als Autor des PDF-Dokuments verwendet werden soll.
- F021** Die Bibliothek muss fähig sein einen spezifizierten String als Autor des PDF-Dokuments anzugeben.
- F022** Die Bibliothek muss die Fähigkeit bieten einen String anzugeben, welcher als Betreff (Subject) des PDF-Dokuments verwendet werden soll.
- F023** Die Bibliothek muss fähig sein einen spezifizierten String als Betreff (Subject) des PDF-Dokuments anzugeben.
- F024** Die Bibliothek muss die Fähigkeit bieten eine neue Seite an das Ende der bisherigen Seiten hinzufügen.
- F025** Die Bibliothek muss die Fähigkeit bieten für eine neue Seite die Seitengröße anzugeben.
- F026** Die Bibliothek soll fähig sein den Page-Tree in Form eines balanced trees zu erstellen.

- F027** Die Bibliothek muss fähig sein Annotations einer Seite hinzu zufügen.
- F028** Die Bibliothek muss die Fähigkeit bieten eine Link-Annotation zu einer Seite hinzuzufügen, welche zu einer Seite verweist.
- F029** Die Bibliothek muss die Fähigkeit bieten eine Link-Annotation zu einer Seite hinzuzufügen, welche eine URI aufruft.
- F030** Die Bibliothek muss fähig sein ein Outlines Dictionary mit entsprechenden Outline-Item-Dictionarys anzulegen.
- F031** Die Bibliothek muss fähig sein den Document Catalog der PDF-Spezifikation entsprechend zu erstellen.
- F032** Die Bibliothek muss fähig sein anhand der vorhandenen Angaben ein Info-Dictionary zu erstellen.
- F033** Die Bibliothek muss die Fähigkeit bieten eine Schriftart im TrueType-Format in das PDF-Dokument einzubetten.
- F034** Die Bibliothek wird die Fähigkeit bieten für Schriftarten Subsets zu erstellen.
- F035** Die Bibliothek wird die Fähigkeit bieten eine Schriftart im OpenType-Format in das PDF-Dokument einzubetten.
- F036** Die Bibliothek wird die Fähigkeit bieten eine Schriftart im Type3-Format in das PDF-Dokument einzubetten.
- F037** Die Bibliothek wird die Fähigkeit bieten eine Schriftart im Type1-Format in das PDF-Dokument einzubetten.
- F038** Die Bibliothek muss die Fähigkeit bieten ein Bild im JPEG-Format in das PDF-Dokument einzubetten.
- F039** Die Bibliothek wird die Fähigkeit bieten ein Bild im PNG-Format in das PDF-Dokument einzubetten.
- F040** Die Bibliothek wird die Fähigkeit bieten ein Bild im TIFF-Format in das PDF-Dokument einzubetten.
- F041** Die Bibliothek wird die Fähigkeit bieten ein Bild im WebP-Format in das PDF-Dokument einzubetten.
- F042** Die Bibliothek muss die Fähigkeit bieten ein eingebettetes Bild an einer übergebenen Position und Größe auf einer Seite zu platzieren.

F043 Die Bibliothek muss die Fähigkeit bieten Text auf einer Seite an einer bestimmten Position zu platzieren.

F044 Die Bibliothek muss die Fähigkeit bieten für das Platzieren von Texten die Schriftart und Schriftgröße anzugeben.

F045 Die Bibliothek muss die Fähigkeit bieten gerade Linien in Form von Vektoren einer Seite hinzuzufügen.

F046 Die Bibliothek muss fähig sein einen ContentStream einer Seite hinzuzufügen.

F047 Die Bibliothek muss fähig sein Anweisungen an einen gegebenen ContentStream anzuhängen.

F048 Die Bibliothek muss fähig sein ein RessourceDictionary einer Seite hinzuzufügen.

F049 Die Bibliothek muss fähig sein verwendete Bilder dem RessourceDictionary einer Seite hinzuzufügen.

F050 Die Bibliothek muss fähig sein verwendete Schriftarten dem RessourceDictionary einer Seite hinzuzufügen.

Transfer des Webinhalts

F100 Die Bibliothek muss fähig sein aus der Überschriften-Hierarchie des Webinhalts die Document-Outlines des PDF-Dokuments zu erstellen.

F101 Die Bibliothek muss die Fähigkeit bieten das Erstellen der Document-Outlines durch eine Option zu aktivieren beziehungsweise deaktivieren.

F102 Die Bibliothek muss die Fähigkeit bieten Überschriften aus den Document-Outlines auszuschließen.

F103 Die Bibliothek soll die Fähigkeit bieten für Überschriften anzugeben, dass dessen untergeordnete Überschriften in der Document-Outline eingeklappt sind.

F104 Die Bibliothek soll die Fähigkeit bieten für Überschriften einen alternativen Titel, anstelle des Text-Inhalts der Überschrift, anzugeben.

F105 Die Bibliothek wird die Fähigkeit bieten in den Optionen anzugeben wie die Seite dargestellt werden soll, wenn auf den Link in den Document-Outlines geklickt wird.

F106 Die Bibliothek muss die Fähigkeit bieten für ein Element anzugeben, dass dieses die aktuelle Seitenzahl beinhalten soll.

- F107** Die Bibliothek muss fähig sein ein Element, welches markiert ist, dass es die aktuelle Seitenzahl beinhalten soll, diese in das Element einzusetzen.
- F108** Die Bibliothek muss die Fähigkeit bieten für ein Element anzugeben, dass dieses die Seitenzahl beinhalten soll, auf welcher ein angegebenes Element zu finden ist.
- F109** Die Bibliothek muss fähig sein ein Element, welches markiert ist, dass es die Seitenzahl auf welcher ein angegebenes Element zu finden ist beinhalten soll, diese in das Element einzusetzen.
- F110** Die Bibliothek muss die Fähigkeit bieten Tagnames von Elementen anzugeben die ignoriert werden sollen.
- F111** Die Bibliothek muss die Fähigkeit bieten Klassennamen von Elementen anzugeben die ignoriert werden sollen.
- F112** Die Bibliothek muss die Fähigkeit bieten Elemente anhand eines Attributs anzugeben, dass dieses ignoriert werden soll.
- F113** Die Bibliothek muss fähig sein die Elemente, welche ignoriert werden sollen, vor der Generierung auszublenden und nicht in das PDF-Dokument übertragen.
- F114** Die Bibliothek soll die Fähigkeit bieten Klassennamen von Elementen die nicht im Webinhalt sichtbar sind anzugeben, welche für die Generation zusätzlich eingeblendet werden sollen.
- F115** Die Bibliothek soll die Fähigkeit bieten Tagnames von Elementen die nicht im Webinhalt sichtbar sind anzugeben, welche für die Generation zusätzlich eingeblendet werden sollen.
- F116** Die Bibliothek soll die Fähigkeit bieten Elemente anhand eines Attributs anzugeben, dass diese für die Generierung zusätzlich eingeblendet werden sollen.
- F117** Die Bibliothek soll fähig sein die Elemente, welche für die Generierung zusätzlich eingeblendet werden sollen, vor der Generierung einzublenden und somit zusätzlich in das PDF-Dokument zu übertragen.
- F118** Die Bibliothek muss fähig sein, neue Seiten zum PDF-Dokument hinzuzufügen, falls für das zu platzierende Element nicht mehr ausreichend Platz auf der Seite vorhanden ist.

- F119** Die Bibliothek muss die Fähigkeit bieten Tagnames von Elementen anzugeben vor welchen ein Seitenumbruch erfolgen soll.
- F120** Die Bibliothek muss die Fähigkeit bieten Tagnames von Elementen anzugeben nach welchen ein Seitenumbruch erfolgen soll.
- F121** Die Bibliothek soll die Fähigkeit bieten für ein Element mit einem Attribut anzugeben, dass vor ihm ein Seitenumbruch stattfinden soll.
- F122** Die Bibliothek soll die Fähigkeit bieten für ein Element mit einem Attribut anzugeben, dass nach ihm ein Seitenumbruch stattfinden soll.
- F123** Die Bibliothek muss fähig sein vor oder nach einem Element eine neue Seite hinzuzufügen wenn dies angegeben wurde.
- F124** Die Bibliothek muss fähig sein die Generierung durchzuführen während der Nutzende mit dem Webinhalt interagiert.
- F125** Die Bibliothek muss fähig sein Elemente für die Generierung auszublenden ohne das sich die Darstellung des Webinhalts für den Nutzenden ändert.
- F126** Die Bibliothek muss die Fähigkeit bieten Kopfzeilen für Seiten über ein Template-Element anzugeben.
- F127** Die Bibliothek muss die Fähigkeit bieten Fußzeilen für Seiten über ein Template-Element anzugeben.
- F128** Die Bibliothek muss fähig sein eine angegebene Kopfzeile auf jeder Seite zu platzieren.
- F129** Die Bibliothek muss fähig sein eine angegebene Fußzeile auf jeder Seite zu platzieren.
- F130** Die Bibliothek muss fähig sein Bilder anderer Formate in das JPEG-Format zu überführen.
- F131** Die Bibliothek muss es ermöglichen Elemente anzugeben welche nicht von einem Seitenumbruch geteilt werden sollen.
- F132** Die Bibliothek muss fähig sein die Elemente welche nicht geteilt werden sollen auf die nächste Seite zu verschieben.
- F133** Die Bibliothek muss fähig sein Texte in der gleichen Darstellung in das PDF-Dokument zu übertragen.

F134 Die Bibliothek muss fähig sein die Gestaltung von Elementen in das PDF-Dokument zu übertragen.

F135 Die Bibliothek muss fähig sein unabhängig von Endgerät und Browser ein einheitliches PDF-Dokument zu erstellen.

F136 Die Bibliothek muss fähig sein die Generierung ohne das Eingreifen des Nutzenden durchzuführen.

F137 Die Bibliothek muss fähig sein die verwendeten Schriften aus dem Webinhalt auszulesen.

F138 Die Bibliothek muss fähig sein Anchor-Elemente in das PDF-Dokument als Verlinkungen zu übertragen.

F139 Die Bibliothek muss fähig sein Anchor-Elemente mit einem anderen übertragenden Element als Ziel als eine Verlinkung zu der entsprechenden Seite auf welcher das Element zu finden ist zu übertragen.

F140 Die Bibliothek muss fähig sein Anchor-Elemente mit URLs als externe Verlinkungen in das PDF-Dokument zu übertragen.

F141 Die Bibliothek muss die Fähigkeit bieten die Seitengröße/das Format von Seiten zu definieren.

F142 Die Bibliothek muss die Fähigkeit bieten den Seitenabstand welcher auf allen Seiten freigelassen werden soll anzugeben.

D.II Nichtfunktionale Anforderungen

NF001 Die Bibliothek soll eine möglichst kleine Dateigröße vorweisen.

NF002 Die Bibliothek soll in möglichst geringer Zeit das PDF-Dokument aus der Webseite generieren.

NF003 Das PDF-Dokument, welches aus der Generierung resultiert soll eine möglichst kleine Dateigröße aufweisen.

D.III Architecture Decision Records (ADRs)

ADR 1: Programmiersprache der Bibliothek

Status: angenommen

Entscheidende: Benedikt Engel

Datum: 04.02.2024

Kontext und Problemstellung:

Für die Entwicklung der Bibliothek für die Generierung von PDF-Dokumenten aus einem vorliegenden Webinhalt muss die Entscheidung getroffen werden in welcher Programmiersprache diese programmiert werden soll.

Entscheidungsfaktoren:

Ermöglichen von einer hohen Codequalität, Reduzierung der Fehleranfälligkeit, für große und komplexe Anwendungsfelder geeignet.

Betrachtete Optionen:

- JavaScript
- TypeScript

Ergebnis der Entscheidung: TypeScript

Positive Konsequenzen:

- Superset von JavaSciprt
- Typensicherheit
- Für größere Projekte gut geeignet
- Klare Strukturen durch Typisierung

Negative Konsequenzen:

- Erhöhter Entwicklungsaufwand
- Compilierung für die Nutzung im Browser erforderlich

Pro und Kontra der Optionen:

Option 1 - JavaScript:

- keine Typisierung und keine Typensicherheit möglich
- + Einfache Nutzung

- + Kein Entwicklungs-Overhead
- + Kann direkt im Browser verwendet werden

Option 2 - TypeScript:

- Erhöhter Entwicklungsaufwand
- Muss für die Verwendung im Browser kompiliert werden
- + Typensicherheit
- + Besonders für große Projekte gut geeignet
- + Klare Strukturen durch Typisierung
- + Da es sich um ein Superset von JS handelt, lässt sich der TypeScript-Code in JavaScript kompilieren.

ADR 2: PDF Writer

Status: angenommen

Entscheidende: Benedikt Engel

Datum: 04.02.2024

Kontext und Problemstellung:

Für den Transfer der Inhalte des Webinhalts wird eine Funktionalität eines PDF-Writers benötigt. Diese muss es ermöglichen ein PDF-Dokument zu erstellen, welches der PDF-Spezifikation entspricht. Diesem PDF-Dokument müssen Schriften, Bilder und Texte hinzugefügt werden und diese an bestimmten Positionen auf einzelnen Seiten des PDF-Dokuments platziert werden können.

Entscheidungsfaktoren:

- Verfügbarkeit von Funktionalitäten für das Hinzufügen von allen benötigten PDF-Objekten wie Document Outlines, Verlinkungen (Annotations), etc.
- Verfügbarkeit zum Auslesen benötigter Informationen, wie Seitennummern
- Möglichst geringe Dateigröße
- Verständliche und einfache Nutzung

Betrachtete Optionen:

- jsPDF (<https://github.com/parallax/jsPDF>)
- PDF-Lib (<https://github.com/Hopding/pdf-lib>)
- Eigene Implementierung

Ergebnis der Entscheidung: Eigene Implementierung

Positive Konsequenzen:

- Funktionen und Funktionalitäten können an den Prozess des Transfers angepasst werden
- Möglichkeit der kleineren Dateigröße, dadurch, dass nur Funktionalitäten entwickelt werden die von benötigt werden.

Negative Konsequenzen:

- Erhöhter Entwicklungsaufwand

- Tiefes Verständnis der PDF-Spezifikation notwendig
- Endgültige Dateigröße ungewiss

Pro und Kontra der Optionen:

Option 1 - jsPDF:

- Viele nicht benötigte Funktionalitäten, wie AcroForms
- Wartung und weitere Entwicklung scheint vorerst eingestellt
- In purem JavaScript entwickelt
- Teils sind Funktionalitäten schlecht dokumentiert (siehe Document Outlines)
- + Bietet alle Funktionalitäten die benötigt werden
- + Einfach zu verwendende API

Option 2 - PDF-Lib:

- Viele nicht benötigte Funktionalitäten, wie AcroForms
- Wartung und weitere Entwicklung scheint vorerst eingestellt
- Keine Funktionalität zur Erstellung von Annotations
- 525 KB Größe
- Keine Funktionalität zur Erstellung von Document Outlines
- + Einfach zu verwendende API

Option 3 - Eigene Implementierung:

- Erhöhter Entwicklungsaufwand
- Tiefes Verständnis der PDF-Spezifikation notwendig
- Endgültige Dateigröße ungewiss
- Keine Funktionalität zur Erstellung von Document Outlines
- + Funktionen und Funktionalitäten können an den Prozess des Transfers angepasst werden
- + Möglichkeit der kleineren Dateigröße, dadurch, dass nur Funktionalitäten entwickelt werden die von benötigt werden.
- + Autor hat durch eigene Entwicklung tiefes Verständnis für die Nutzung

ADR 3: Bibliothek zur Verarbeitung von Schrift-Dateien (Font-Engine)

Status: angenommen

Entscheidende: Benedikt Engel

Datum: 04.02.2024

Kontext und Problemstellung:

Durch die Entscheidung in ADR 2 wird eine Bibliothek benötigt, welche es ermöglicht alle Informationen aus vorliegenden Schriftart-Dateien auszulesen sodass die entsprechenden FontDictionarys erstellt werden können. Diese beinhalten unteranderem den PostScript-Namen, den Winkel von kursiven Schriften und den Ascent und Descent Werten.

Entscheidungsfaktoren:

Auslesen aller benötigten Informationen der Schriften; möglichst kleine Dateigröße; einfache Nutzung.

Betrachtete Optionen:

- fontkit (<https://github.com/foliojs/fontkit>)
- font-reader (<https://github.com/Elity/FontReader>)

Ergebnis der Entscheidung: fontkit

Positive Konsequenzen:

- Subsets von Schriftarten möglich
- Unterstützung vieler Formate
- Auslesen aller benötigten Informationen möglich

Negative Konsequenzen:

- Zusätzliche Dateigröße

Pro und Kontra der Optionen:

Option 1 - Fontkit

- Dateigröße
- + Subsets von Schriftarten möglich
- + Unterstützung vieler Formate (otf, ttf, woff, woff2, ttc, dfont)
- + Auslesen aller benötigten Informationen möglich

Option 2 - font-reader

- Nur für Schriftarten im TrueType Format nutzbar
- Liefert nur Metadaten der Schriften, welche nicht ausreichend sind.
- + Kleine Dateigröße

ADR 4: Bibliothek zur Kompression (Compression-Engine)

Status: ersetzt durch ADR 4.1

Entscheidende: Benedikt Engel

Datum: 04.02.2024

Kontext und Problemstellung:

Für die Implementierung des PDF-Writers (siehe ADR 2) wird eine Möglichkeit benötigt um Datenstreams von Stream Objekten zu komprimieren. Dies wirkt sich positiv auf die resultierende Dateigröße des PDF-Dokuments aus, welche so gering wie möglich ausfallen soll. Hierfür wird die Verwendung von zlib flate und deflate angestrebt. **Entscheidungsfaktoren:**

Möglichst kleine Dateigröße; Funktionalität um Datenstreams mit zlib-Deflate zu komprimieren; Einfache Handhabung.

Betrachtete Optionen:

- node:zlib (<https://nodejs.org/api/zlib.html>)
- pako (<https://github.com/nodeca/pako>)

Ergebnis der Entscheidung: node:zlib

Positive Konsequenzen:

- Bekannte Nutzung
- Einfach anzuwenden

Negative Konsequenzen:

- Muss durch den Bundler für die Nutzung im Browser übertragen werden

Pro und Kontra der Optionen:

Option 1 - node:zlib:

- Eigentlich für den Node Kontext gedacht
- Muss durch den Bundler für die Nutzung im Browser übertragen werden
- + Lässt sich durch den Bundler *Browserify* auch im Browser nutzen
- + Anwendung dem Entwickelnden bereits bekannt
- + Einfache Anwendung

Option 2 - pako:

- Zusätzliche, externe Bibliothek und somit Abhängigkeit
- + Einfache Nutzung

ADR 4.1: Bibliothek zur Kompression (Compression-Engine)

Status: angenommen

Entscheidende: Benedikt Engel

Datum: 18.05.2024

Vorgänger: ADR 4

Kontext und Problemstellung:

Für die Implementierung des PDF-Writers (siehe ADR 2) wird eine Möglichkeit benötigt um Datenstreams von Stream Objekten zu komprimieren. Dies wirkt sich positiv auf die resultierende Dateigröße des PDF-Dokuments aus, welche so gering wie möglich ausfallen soll. Hierfür wird die Verwendung von zlib flate und deflate angestrebt.

Da eine weitere Bibliothek gefunden wurde, welche die Bibliothek verbessern kann wird eine erneute Betrachtung durchgeführt.

Entscheidungsfaktoren:

Möglichst kleine Dateigröße; Funktionalität um Datenstreams mit zlib-Deflate zu komprimieren; Einfache Handhabung.

Betrachtete Optionen:

- node:zlib (<https://nodejs.org/api/zlib.html>)
- pako (<https://github.com/nodeca/pako>)
- fflatte (<https://github.com/101arrowz/fflatte>)

Ergebnis der Entscheidung: fflatte

Positive Konsequenzen:

- Sehr kleine Dateigröße
- Einfache Nutzung

Negative Konsequenzen:

- Es sind keine Negativen Konsequenzen zu erwarten.

Pro und Kontra der Optionen:

Option 1 - node:zlib:

- Eigentlich für den Node Kontext gedacht
- + Lässt sich durch den Bundler *Browserify* auch im Browser nutzen

- + Anwendung dem Entwickelnden bereits bekannt
- + Einfache Anwendung

Option 2 - pako:

- Zusätzliche, externe Bibliothek und somit Abhängigkeit
- + Einfache Nutzung

Option 3 - fflat:

- + Sehr kleine Dateigröße (8 KB)
- + Einfache Nutzung

Anhang E Implementierung

E.I Quellcode der Bibliothek

Im Weiteren werden einige Ausschnitte des Quellcodes angeführt, auf welche in der Arbeit Bezug genommen wurde. Der vollständige Quellcode der Bibliothek ist im Github-Repository einsehbar: <https://github.com/BenediktEngel/htmlWebsite2pdf>

E.I.1 Konstante PageDimensions

/src/constants/PageDimensions.ts

```
1 export const PageDimensions = {  
2     A6: [297, 420] as [number, number],  
3     A5: [420, 595] as [number, number],  
4     A4: [595, 842] as [number, number],  
5     A3: [842, 1191] as [number, number],  
6     A2: [1191, 1684] as [number, number],  
7     A1: [1684, 2384] as [number, number],  
8     A0: [2384, 3370] as [number, number],  
9     LETTER: [612, 792] as [number, number],  
10    LEGAL: [612, 1008] as [number, number],  
11    LEDGER: [792, 1224] as [number, number],  
12    TABLOID: [792, 1224] as [number, number],  
13};  
14  
15 export default PageDimensions;
```

E.I.2 Erstellung des Buffers für die PDF-Datei

Ausschnitt aus: /src/pdfDocument.ts

```
1 // [ ... ]  
2 outputFileAsBuffer(): Buffer {  
3     this.createDocumentOutline();  
4     this.includedFonts.forEach((font, name) => {  
5         if (font.usedChars.size === 0) {  
6             this.indirectObjects.delete(font.fontDictionary.id!);  
7         } else {  
8             fontHelper.addFontToDocument(this, font);  
9         }  
10    }  
11    );  
12    return this.outputFileAsBuffer();  
13}
```

```

10 });
11 const body = this.outputBody();
12 const bytesToStartxref = body.byteLength;
13 return Buffer.concat([
14   body,
15   Buffer.from(` ${this.crossReferenceTableAsString()}trailer\r\n${this.
16 trailer.toBuffer()} \r\nstartxref\r\n${bytesToStartxref}\r\n%EOF `),
17 ]);
18 // [ ... ]

```

E.I.3 Generierung von Header und Body des PDF-Dokuments

Ausschnitt aus: /src/pdfDocument.ts

```

1 // [ ... ]
2 private outputHeaderAndBody(): Buffer {
3   let outputBuffer = Buffer.from(`%PDF-${this.version}\r\n%ääßö\r\n`);
4   this.indirectObjects.forEach((indirectObject, id) => {
5     const temp = indirectObject;
6     const objectOutput = indirectObject.obj.toBuffer();
7     temp.byteOffset = outputBuffer.byteLength;
8     this.indirectObjects.set(id, temp);
9     outputBuffer = Buffer.concat([outputBuffer, objectOutput, Buffer.
10      from('\r\n')]);
11   });
12   return outputBuffer;
13 }

```

E.I.4 Auslesen der Position einer Überschrift im Verhältnis zu den bisherigen Überschriften

Ausschnitt aus: /src/Generator.ts

```

1 // [ ... ]
2 findBookmarkPosition(
3   bookmarks: Array<TBookmarkObject>,
4   level: number,
5   positionArray: Array<number> = [],
6 ): { bookmarks: Array<TBookmarkObject>; positions: Array<number> } {

```

```

7  if (!bookmarks.length || bookmarks.at(-1)!.level === level) {
8    bookmarks.push({ level, children: [] });
9    return { bookmarks, positions: [...positionArray, bookmarks.length] };
10   };
11 } else {
12   let result = this.findBookmarkPosition(bookmarks.at(-1)!.children,
13   level, [...positionArray, bookmarks.length]);
14   bookmarks.at(-1)!.children = result.bookmarks;
15   return { bookmarks, positions: result.positions };
16 }
16 // [ ... ]

```

E.I.5 Methoden zur Erstellung der Objekte für die Document Outlines

Ausschnitt aus: /src/pdfDocument.ts

```

1 // [ ... ]
2 private createDocumentOutline(): void {
3   if (!this.bookmarkStructure.length) return;
4   const outline = new DictionaryObject(this, new Map(), true);
5   this.bookmarkStructure.forEach((bookmark) => {
6     this.createOutlineItem(this, bookmark, outline);
7   });
8   outline.setValueByKey('First', this.indirectObjects.get(this.
9     bookmarkStructure[0].objectId!)?.obj as DictionaryObject);
10  outline.setValueByKey(
11    'Last',
12    this.indirectObjects.get(this.bookmarkStructure[this.
13      bookmarkStructure.length - 1].objectId!)?.obj as DictionaryObject,
14  );
15  if (this.bookmarkStructure.length > 1) {
16    this.bookmarkStructure.forEach((bookmark, index) => {
17      if (index > 0) {
18        const obj = this.indirectObjects.get(bookmark.objectId!)?.obj as
19          DictionaryObject;
20        const prevObj = this.indirectObjects.get(this.bookmarkStructure[
21          index - 1].objectId!)?.obj as DictionaryObject;
22        // Set Prev to the current object
23        obj.setValueByKey('Prev', prevObj);
24        // Set Next to the previous object
25      }
26    });
27  }
28}

```

```

21         prevObj.setValueByKey('Next', obj);
22     }
23 }
24 }
25 this.catalog?.setValueByKey('Outlines', outline);
26 }
27
28 private createOutlineItem(pdf: PDFDocument, bookmark: TBookmark, parent:
29     DictionaryObject): void {
30     const outlineItem = new DictionaryObject(pdf, new Map(), true);
31     bookmark.objectId = outlineItem.id;
32     outlineItem.setValueByKey('Title', new StringObject(pdf, bookmark.
33         title));
34     outlineItem.setValueByKey('Parent', parent);
35     outlineItem.setValueKey(
36         'Dest',
37         new ArrayObject(pdf, [this.indirectObjects.get(bookmark.pageObjectId
38             !)?.obj as Page, NameObject.getName(pdf, 'Fit')]),
39     );
40     if (bookmark.children.length) {
41         bookmark.children.forEach((child, index) => {
42             this.createOutlineItem(pdf, child, outlineItem);
43             if (index > 0) {
44                 // Set Prev to the current object
45                 const obj = this.indirectObjects.get(child.objectId!)?.obj as
46                 DictionaryObject;
47                 const prevObj = this.indirectObjects.get(bookmark.children[index
48                     - 1].objectId!)?.obj as DictionaryObject;
49                 obj.setValueByKey('Prev', prevObj);
50                 // Set Next to the previous object
51                 prevObj.setValueByKey('Next', obj);
52             }
53         });
54         outlineItem.setValueByKey('First', this.indirectObjects.get(bookmark.
55             children[0].objectId!)?.obj as DictionaryObject);
56         outlineItem.setValueByKey('Last', this.indirectObjects.get(bookmark.
57             children[bookmark.children.length - 1].objectId!)?.obj as
58             DictionaryObject);
59     }
60 }

```

```
53 // [ ... ]
```

E.I.6 Überführung des CSS-Layouts von Elementen

Ausschnitt aus: /src/Generator.ts

```

1  async handleElementNode(element: TElement, page?: Page): Promise<void> {
2  // [ ... ]
3  let newEl = document.createElement('div');
4  let scaling = 1.5;
5  let createInlineStyles = (styles) => {
6      let out = '';
7      [...styles].forEach((style) => {
8          if (style !== 'display' && style !== 'width' && s !== 'height'
9              && !style.startsWith('-') && !style.startsWith('margin')) {
10             out += `${style}:${styles[style]};`;
11         } else if (style === 'display') {
12             out += `${style}:block;`;
13         }
14     });
15     out += `width:${element.rect.width * scaling}px;`;
16     out += `height:${element.rect.height * scaling}px;`;
17     return out;
18 };
19 newEl.setAttribute('style', createInlineStyles(element.styles));
20
21 const data = `<svg xmlns="http://www.w3.org/2000/svg" width="${
22     element.rect.width * scaling}px" height="${element.rect.height *
23     scaling}px"><foreignObject width="100%" height="100%"><div xmlns="
24     http://www.w3.org/1999/xhtml">${newEl.outerHTML}</div></foreignObject
25     ></svg>`;
26
27 function toImage(svg): Promise<{ url: string; width: number; height:
28     number }> {
29     return new Promise((resolve, reject) => {
30         const svgDataUrl = `data:image/svg+xml;base64,` + btoa(
31             unescape(encodeURIComponent(svg)));
32         const img = new Image();
33         img.onload = function () {
34             const canvas = document.createElement('canvas');
35             canvas.width = img.naturalWidth;
36             canvas.height = img.naturalHeight;
37         }
38     });
39 }
```

```

29         const ctx = canvas.getContext('2d');
30         ctx!.fillStyle = 'rgb(255, 255, 255)';
31         ctx!.fillRect(0, 0, canvas.width, canvas.height);
32         ctx!.drawImage(img, 0, 0);
33         canvas.toBlob(
34             function (blob) {
35                 if (blob) {
36                     const jpegUrl = URL.createObjectURL(blob);
37                     resolve({ url: jpegUrl, width: img.naturalWidth,
38 height: img.naturalHeight });
39                 } else {
40                     reject(new Error('Canvas toBlob failed'));
41                 }
42             },
43             'image/jpeg',
44             1.0,
45         );
46         img.src = svgDataUrl;
47     );
48 }
49 const newImgData = await toImage(data);
50
51 const imgData = await fetch(newImgData.url).then((res) => res.
arrayBuffer());
52 this.pdf.addImageToPage(
53 {
54     x: px.toPt(element.rect.x - this.offsetX),
55     y: this.pageTopOffset - px.toPt(element.rect.y + this.
scrollTop - this.offsetY + element.rect.height),
56 },
57 imgData as Buffer,
58 newImgData.width,
59 newImgData.height,
60 px.toPt(element.rect.width),
61 px.toPt(element.rect.height),
62 ImageFormats.JPEG,
63 hashCode(newImgData.url.toString().slice(0, 100)).toString(),
64 page,
65 );

```

```

66     }
67 // [ ... ]

```

E.I.7 Textinhalte zeilenweise auslesen

Ausschnitt aus: /src/Generator.ts

```

1 // [ ... ]
2
3 getTextLinesOfNode(element: Text): Array<TTextLine> {
4   // If the textNode is empty we return an empty array
5   if (element.data.replace(/[\n\t\r\s]/g, '').replace(/\s/g, '').length
6     === 0) {
7     return [];
8   }
9   const textWithStyles: Array<TTextNodeData> = [];
10
11  const text = element.textContent;
12  const parentStyle = window.getComputedStyle(element.parentElement!);
13  const range = this.iframeDoc!.createRange();
14
15  // Get the lines of the text
16  const lineRange = this.iframeDoc!.createRange();
17  lineRange.selectNodeContents(element);
18  const lines = lineRangeClientRects();
19
20  let words = text!.split(/\s+/);
21  // Also split words with hyphens into several parts to also split them
22  // at end of a line
23  let replacements: Array<{ wordIndex: number; parts: Array<string> }> =
24    [];
25  words.forEach((word, wordIndex) => {
26    if (word.includes('-') && word.length > 1) {
27      let parts = word.split('-');
28      parts.forEach((part, index) => {
29        if (part !== parts.at(-1)) {
30          parts[index] = `${parts[index]}-${`;
31        }
32      });
33      replacements.push({ wordIndex, parts });
34    }
35  });

```

```

32  if (replacements.length > 0) {
33    replacements.reverse().forEach((replacement) => {
34      words.splice(replacement.wordIndex, 1, ...replacement.parts);
35    });
36  }
37  // remove empty words
38  words = words.filter((word) => word.trim().length > 0);
39
40  let startOffset = 0;
41  words.forEach((word) => {
42    const wordOffset = text!.indexOf(word, startOffset);
43    range.setStart(element, wordOffset);
44    range.setEnd(element, wordOffset + word.length);
45
46    const rects = rangeClientRects();
47    // If we have more than one rect we have a line break
48    if (rects.length > 1) {
49      // This schould be a separate function to make more then one
50      // linebreak possible
51      let newParts: Array<{ rect: DOMRect; text: string }> = [];
52      for (let i = 1; i <= word.length; i++) {
53        range.setStart(element, wordOffset);
54        range.setEnd(element, wordOffset + i);
55        const newRects = rangeClientRects();
56        const isSafari = /~((?!chrome|android).)*safari/i.test(navigator
57          .userAgent);
58        if (newRects.length > 1) {
59          if (parentStyle.hyphens === 'auto') {
60            // Hyphanation is activated so we should split the word
61            // Caused by the reason that the added hyphen gets its own
62            // rect we now that i chars are before.
63
64            newParts = [
65              { rect: rects[0], text: ` ${word.slice(0, i)}-` },
66              { rect: rects[isSafari ? 1 : 2], text: ` ${word.slice(i)}-` }
67            ],
68          } else {
69            // No hyphanation so we split the word at the char which is
70            // at the end of the line
71          }
72        }
73      }
74    }
75  }
76}

```

```

67         newParts = [
68             { rect: rects[0], text: `${word.slice(0, i - 1)}` },
69             { rect: rects[1], text: `${word.slice(i - 1)}` },
70         ];
71     }
72     break;
73 }
74 }
75 Array.from(newParts).forEach((part) => {
76     textWithStyles.push({
77         text: part.text,
78         position: part.rect,
79     });
80 });
81 } else {
82     textWithStyles.push({
83         text: word,
84         position: rects[0],
85     });
86 }
87
88 startOffset = wordOffset + word.length;
89 });
90
91 // Add the Words to the lines
92 const lineElements = Array.from(lines).map((line) => {
93     return {
94         position: line,
95         words: [],
96         styles: parentStyle,
97     };
98 });
99 textWithStyles.forEach((element) => {
100     // Check in which line the element is and add it to words
101     if (element.position.top) {
102         let line = lineElements.find((line) => line.position.top ===
103             element.position.top);
104         line!.words.push(element);
105     }
106 });

```

```
106     return lineElements;
107 }
108 // [ ... ]
```

E.II Optionen-Objekt für die Generierung

Folgende Optionen lassen sich über das Optionen-Objekt für die Generierung des PDF-Dokuments übergeben:

```
1 {
2     /*
3      Margin of each page in the PDF in pt. Can be a number, a four number
4      array or a two number array.
5      One number: The margin will be applied to all four sides of the page.
6      Array of two numbers: The first number will be applied as the margin
7      at the top and bottom, the second as the left and right margin.
8      Array of four numbers: The numbers correspond to the top, right,
9      bottom and left margins respectively (like CSS).
10     Default: [0, 0, 0, 0]
11     */
12     margin: number | [number, number] | [number, number, number, number];
13     /*
14      The filename of the PDF file.
15      Can include the suffix '.pdf', but is not required.
16      Default: 'output.pdf'
17     */
18     filename: string;
19     /*
20      The title of the PDF file.
21      Default: ''
22     */
23     title: string;
24     /*
25      The PDF-Version which the output should be compatible with.
26      Default: '1.4'
27     */
28     version: PdfVersion;
29     /*
30      The size of the pages in the PDF in pt.
31      Default: 'A4' => [595, 841]
```

```
29  */
30  pageSize: [number, number] | PageDimensions;
31  /*
32   Elements which should be ignored based on classnames.
33   Default: []
34  */
35  ignoreElementsByClass: string | string[];
36  /*
37   Elements which should be ignored based on tag names.
38   Default: []
39  */
40  ignoreElements: string | string[];
41  /*
42   Whether or not to include custom page headers in the PDF.
43   If true, the header has to be defined by a template with the data-
44   attribute 'data-htmlWebsite2pdf-header'. For each page with elements
45   which are in the same parent element as the header, the header will
46   be used on the page.
47  Default: false
48  */
49  usePageHeaders: Boolean;
50  /*
51   Whether or not to include custom page footers in the PDF.
52   Same usage as the option for headers, but the template needs the data-
53   attribute 'data-htmlWebsite2pdf-footer'.
54  Default: false
55  */
56  usePageFooters: Boolean;
57  /*
58   Elements which should get a page break before them based on the tag
59   name.
60  Default: []
61  */
62  pageBreakBeforeElements: string | string[];
63  /*
64   Elements which should get a page break after them based on the tag
65   name.
66  Default: []
67  */
68  pageBreakAfterElements: string | string[];
```

```
63  /*
64   Whether or not to generate the Document-Outline for the PDF based on
65   the headings in the HTML.
66   If true, the outline will be generated based on the headings and their
67   hierarchy in the HTML. Therefore the headings have to be in the
68   correct order.
69   Default: false
70 */
71 outlineForHeadings: Boolean;
72 /*
73 The author of the PDF file.
74 Default: ''
75 */
76 author: string;
77 /*
78 The subject of the PDF file.
79 Default: ''
80 */
81 subject: string;
82 /*
83 The keywords of the PDF file.
84 Default: ''
85 */
86 keywords: string;
87 /*
88 The width of the iframe in px which is used for the generation. Should
89 be a size where the input element is the width of the available
90 pagewidth so everything is placed and sized correctly. (pageWidth -
91 marginLeft - margin-Right)
92 Default: pageWidth - marginLeft - margin-Right
93 */
94 iFrameWidth: number;
95 /*
96 Whether or not a page should be added at the begin of the generation.
97 If the first element to set is in the list of pagebreak-before you
98 should set this to false. Otherwise the first page will be a blank
99 page.
100 Default: true
101 */
102 addFirstPage: Boolean;
```

```

94  /*
95   Names of elements which should be avoided to break in the middle of
96   them if there is not enough space on the page. Based on the tag name.
97   If the element is bigger than the page, this option will be ignored.
98   Also lines of text will always be avoided, so there is no need to
99   add paragraphs or spans to this list.
100  Default: []
101  */
102  avoidBreakingElements: string | string [];
103  pdfOptions: {
104    viewerPreferences: {
105      /*
106       Whether or not to hide the toolbar.
107       Default: false
108     */
109     hideToolbar: Boolean;
110     /*
111      Whether or not to hide the menubar.
112      Default: false
113    */
114     hideMenubar: Boolean;
115     /*
116      Whether or not to hide the window UI, like scrollbars.
117      Default: false
118    */
119     hideWindowUI: Boolean;
120     /*
121      Whether or not to display the document title instead of the
122      filename.
123      Default: false
124    */
125    displayDocTitle: Boolean;
126  };
127  /*
128   The page mode which should be used in a PDF-Viewer.
129   Default: 'UseNone'
130 */
131  pageMode: PdfPageMode;
132  /*
133   The layout of the pages when displayed in a PDF-Viewer.

```

```
129     Default: 'SinglePage'  
130     */  
131     pageLayout: PdfPageLayout;  
132 };  
133 }
```

Anhang F Generierte PDF-Dokumente zu Kapitel 5

F.I Ohne Voreinstellungen

URL⁵²: <https://master.benediktengel.de/generierung?type=htmlwebsite2pdf&option=1> Funtionsaufruf:

```
1 htmlWebsite2Pdf.fromElement(document.querySelector('main'), {
2   filename: 'htmlWebsite2Pdf.pdf',
3 })
```

Test Generierungswebsite

Diese Seite stellt eine Test-Website für die Prüfung der Möglichkeiten und Technologien zur Generierung von PDF-Dokumenten aus Webinhalten mit HTML-Markup und CSS-Layout dar. Diese Seite verwendet unterschiedliche Schriftarten und viele verschiedene Elemente, welche in Webinhalten Verwendung finden können, aus welchen zudem PDF-Dokumente generiert werden sollen.

Textparagraphen mit unterschiedlichen Inline-Darstellungen

Der folgende Blindtext nutzt verschiedene Inline-Gestaltungsmöglichkeiten von Text, wie kursive Texte, Fetttexte und Texte mit Umrandung oder Hintergründen:

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. Loren 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. Loren 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. Loren 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. Loren 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. Loren 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. Loren 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 es.

⁵²Zugangsdaten für den Zugriff:

Benutzername: master Passwort: Thesis24

Im nun folgenden wird ein Text im Blocksatz dargestellt:

Nam liber tempor cum soluta nobis eleifend option congue nihil imperdiet doming id quod mazim placerat facer possim assum. Lorem ipsum dolor sit amet, consecetur adipiscing elit, sed diam nonummy nibh euismod tincidunt ut laoreet dolore magna aliquam erat volutpat. Ut wisi enim ad minim veniam, quis nostrud exerci tation ullamcorper suscipit lobortis nisl ut aliquip ex ea commodo consequat.

Text im Blocksatz und Auto-Hyphens

Im nun folgenden wird ein weiterer Text im Blocksatz und der Verwendung von Wortumbrechen am Zeileende dargestellt:

At vero eos et accusamus et iusto duo dolores et ea rebum. Sicut clita kasd gubergren, no sea takinata sanctus est Lorem ipsum dolor sit amet. Lorem ipsum dolor sit amet, consecetur. Duis autem vel eum iriure dolor in hendrerit in vulputate velit esse mole- stie consequat, vel illum dolore eu fereiat nulla facilisis. At vero eos et accusam et justo duo dolores et ea rebum. Sicut clita kasd gubergren, no sea takinata sanctus est Lorem ipsum dolor sit amet. Lorem ipsum dolor sit amet, consecetur

Links

Im folgenden werden Links dargestellt um diese auf ihre Funktion im PDF-Dokument zu überprüfen:

Ich bin ein interner Link und führe zur Überschrift "Text im Blocksatz und Auto-Hyphens"

Hyphens"

Ich bin ein externer Link und führe zur Website "www.example.com"

Tabelle

Erste Spalte	Zweite Spalte
Text	Lorem ipsum dolor sit amet, consecetur sadipscing elitr, sed diam nonummy eiusmod tempor iuvhnt ut labore et dolore magna aliquyam erat, sed diam voluptua.
Noch mehr Text	At vero eos et accusam et justo duo dolores et ea rebum. Sicut clita kasd gubergren, no sea takinata sanctus est Lorem ipsum dolor sit amet.

Bild



Photo by Tim Peterson on Unsplash

Text im Blocksatz

F.II Verwendung von Vektoren für Hintergründe und Umrundungen

URL⁵³: <https://master.benediktengel.de/generierung?type=htmlwebsite2pdf&option=2> **Funktionsaufruf:**

```
1 htmlWebsite2Pdf.fromElement(document.querySelector('main'), {  
2   filename: 'htmlWebsite2Pdf.pdf',  
3   imagesForLayout: false  
4 })
```

⁵³Zugangsdaten für den Zugriff:

Benutzername: master Passwort: Thesis24

Im nun folgenden wird ein Text im Blocksatz dargestellt:
 Nam liber tempor cum soluta nobis eleifend option congue nihil imperdiet doming id
 quod mazim placerat facer possim assum. Lorem ipsum dolor sit amet, consecetur
 adipiscing elit, sed diam nonummy nibh euismod tincidunt ut laoreet dolore magna
 aliquam erat volutpat. Ut wisi enim ad minim veniam, quis nostrud exerci tation
 ullamcorper suscipit lobortis nisl ut aliquip ex ea commodo consequat.

Text im Blocksatz und Auto-Hyphens

Im nun folgenden wird ein weiterer Text im Blocksatz und der Verwendung von
 Wortumbrechen am Zeileende dargestellt:
 At vero eos et accusamus et iusto duo dolores et ea rebum. Sicut clita kasd gubergren,
 no sea takimata sanctus est Lorem ipsum dolor sit amet. Lorem ipsum dolor sit amet,
 consecetur. Duis autem vel eum iriure dolor in hendrerit in vulputate velit esse mole-
 stie consequatur, vel illum dolore eu feugiat nulla facilisis. At vero eos et accusam et
 iusto duo dolores et ea rebum. Sicut clita kasd gubergren, no sea takimata sanctus est
 Lorem ipsum dolor sit amet. Lorem ipsum dolor sit amet, consecetur

Links

Im folgenden werden Links dargestellt um diese auf ihre Funktion im PDF-Dokument
 zu überprüfen:
 Ich bin ein interner Link und führe zur Überschrift "Text im Blocksatz und Auto-
 Hyphens"
 Ich bin ein externer Link und führe zur Website "www.example.com"

Tabelle

Erlste Spalte	Zweite Spalte
Text	Lorem ipsum dolor sit amet, consecetur sadipscing elitr, sed diam nonummy eiusmod tempor iwhridut ut labore et dolore magna aliquyam erat, sed diam voluptua.
Noch mehr Text	At vero eos et accusam et iusto duo dolores et ea rebum. Sicut clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet.

Bild



Photo by Tim Peterson on Unsplash

Text im Blocksatz

F.III Verwendung von Fußzeilen und Outlines

URL⁵⁴: <https://master.benediktengel.de/generierung?type=htmlwebsite2pdf&option=3> Funtionsaufruf:

```
1 htmlWebsite2Pdf.fromElement(document.querySelector('main'), {
2   filename: 'htmlWebsite2Pdf.pdf',
3   imagesForLayout: false,
4   usePageFooters: true,
5   outlineForHeadings: true,
6   addFirstPage: false,
7   pageBreakBeforeElements: ['section'],
8 })
```

Test Generierungswebsite

Diese Seite stellt eine Test-Website für die Prüfung der Möglichkeiten und Technologien zur Generierung von PDF-Dokumenten aus Webinhalten mit HTML-Markup und CSS-Layout dar. Diese Seite verwendet unterschiedliche Schriftarten und viele verschiedene Elemente, welche in Webinhalten Verwendung finden können, aus welchen zudem PDF-Dokumente generiert werden sollen.

Textparagraphen mit unterschiedlichen Inline-Darstellungen

Der folgende Blindtext nutzt verschiedene Inline-Gestaltungsmöglichkeiten von Text, wie kursive Texte, Fetttexte und Texte mit Umrandung oder Hintergründen:

 Lorem ipsum dolor sit amet, consectetur adipisicing elit, 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. Sint clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet. Lorem ipsum dolor sit amet, consectetur adipisicing elit, 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. Sint clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet.

 Ut wisi enim ad minim veniam, quis nostrud exerci tation ullamcorper suscipit lobortis nisl ut aliquip ex ea commodo consequat. Duis autem vel eum iriure dolor in hendrerit in vulputate velit esse molestie consequat, vel illum dolore eu feugiat nulla facilisis at vero eros et accumsan et justo odio dignissim qui blandit praesent luptatum zzril delenit augue duis dolore te feugait nulla facilisi. At vero eos et accusam et justo duo dolores et ea rebum. Sint clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet. Lorem ipsum dolor sit amet, consectetur adipisicing elit, 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. Sint clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet. Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed diam nonumy eirmod tempor invidunt ut labore et.

⁵⁴Zugangsdaten für den Zugriff:

Benutzername: master Passwort: Thesis24

<p>dolor 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 es.</p> <h3>Links</h3> <p>Im folgenden werden Links dargestellt, um diese auf ihre Funktion im PDF-Dokument zu überprüfen:</p> <ul style="list-style-type: none"> Ich bin ein interner Link und führt zur Überschrift "Text im Blocksatz und Auto-Hyphens" Ich bin ein externer Link und führt zur Website "www.example.com" 	<h3>Tabelle</h3> <table border="1"> <thead> <tr> <th>Erste Spalte</th><th>Zweite Spalte</th></tr> </thead> <tbody> <tr> <td>Text</td><td> Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed diam nonummy nibh euismod tempor incididunt ut labore et dolore magna aliqua. Ut 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.</td></tr> <tr> <td>Noch mehr Text</td><td></td></tr> </tbody> </table>	Erste Spalte	Zweite Spalte	Text	Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed diam nonummy nibh euismod tempor incididunt ut labore et dolore magna aliqua. Ut 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.	Noch mehr Text	
Erste Spalte	Zweite Spalte						
Text	Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed diam nonummy nibh euismod tempor incididunt ut labore et dolore magna aliqua. Ut 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.						
Noch mehr Text							

 <p>Photo by Tim Peterson on Unsplash</p>	<h3>Text im Blocksatz</h3> <p>Im nun folgenden wird ein Text im Blocksatz dargestellt.</p> <p>Nam liber tempor cum solita nobis eleifend option congue nihil imperdiet doming id quod nazim placerat facer possim assum. Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed diam nonummy nibh euismod tincidunt ut laoreet dolore magna aliquam erat volutpat. Ut wisi enim ad minim veniam, quis nostrud exerci tation ullamcorper suscipit lobortis nisl ut aliquip ex ea commodo consequat.</p> <h3>Text im Blocksatz und Auto-Hyphens</h3> <p>Im nun folgenden wird ein weiterer Text im Blocksatz und der Verwendung von Wortumbrüchen am Zeilenende dargestellt:</p> <p>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, consectetur. Duis autem vel eum iriure dolor in hendrerit in vulputate velit esse molestie consequat, vel illum dolore eu feugiat nulla facilisis. 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, consectetur</p>
--	--