ENTWICKLUNG EINER FREIEN SOFTWARE ZUR VERWALTUNG VON DIGITALEN COMICS AUF BASIS VON WEB-TECHNOLOGIEN

Dokumentation des Praxis-Projekts “ComicLib” im Bachelor-Studiengang „Informatik“

André Hahn
TH KÖLN | andre.hahn1@smail.th-koeln.de

Abstract

Mit dem Aufkommen von E-Books sind auch Comics im digitalen Zeitalter angekommen. Digitale Comicsammlungen bestehen meistens aus Dateien unterschiedlicher, teils proprietärer Dateiformate. Da Comics in proprietären Formaten nur in den Anwendungen der Hersteller geöffnet werden können, gestaltet sich die gemeinsame Verwaltung aller Comics einer Sammlung schwierig. Da die Verwaltung der Comics bei diesen Herstellern zentralisiert auf deren Servern erfolgt, ist das Lesen von Comics bei diesen über deren Apps standortunabhängig möglich. Diesen Vorteil bieten Comics in nicht-proprietären Formaten nicht. Für den maximalen Komfort muss man sich also für einen einzelnen Anbieter entscheiden und eine Fragmentierung der eigenen Comicsammlung über mehrere Anbieter hinweg vermeiden. Diese Probleme soll dieses Projekt lösen und dazu mit ComicLib eine freie Software zur zentralen Verwaltung und Bereitstellung digitaler Comicsammlungen schaffen. Da ein Öffnen der proprietären Dateien nicht möglich ist, soll die Beschaffung von Informationen zu Comics, Serien und Verlagen über den Zugriff auf die ComicVine API[1] von Gamespot[2] erfolgen. Bei Comics in nicht-proprietären Dateiformaten soll das Lesen direkt in der Software möglich sein, bei den proprietären Formaten soll die Datei stattdessen nur zum Download bereitgestellt werden. Eine eigene Programmierschnittstelle soll die Erweiterung von ComicLib um mobile Apps ermöglichen. Die Implementation als Webanwendung soll eine Nutzung der Comicverwaltung auf Computern, Tablets und Smartphones ermöglichen.

[1] https://comicvine.gamespot.com/api
[2] https://www.gamespot.com/

Inhalt

1 Über das Projekt

Die Software ComicLib (kurz für „Comics Library“, also Comicbuch-Bibliothek) soll die Verwaltung und Nutzung digitaler Comics ermöglichen und entsteht im Rahmen des Praxis-Projekts des Bachelor-Studiengangs Informatik an der TH Köln. Im folgenden Kapitel sollen die Aufgabenstellung und ihr Lösungsansatz sowie die Motivation des Projekts dargelegt werden.

1.1 Motivation

Bildergeschichten stellen die älteste Erzählform der Menschheitsgeschichte dar. Mit der fortschreitenden, technischen Entwicklung haben sich auch die Medien zur Speicherung und Verbreitung stets weiterentwickelt. Was mit den Höhlenmalereien der Steinzeit begann, fand seine Fortsetzung unter anderem in den Papyri der antiken Ägypter, den Illustrationen handgeschriebener Bibeln des Mittelalters und den ersten industriell gedruckten Comicstrips in Zeitungen des 19. Jahrhunderts. Mit dem Aufkommen der digitalen Bücher hielt auch der Comic Einzug in das digitale Zeitalter.

In den letzten Jahren ist so ein Markt mit einer Vielzahl von Anbietern digitaler Comics entstanden, die zum Teil proprietäre oder mit digitalem Rechtemanagement (DRM) versehene Dateiformate zur Speicherung von Comics verwenden. Gerade bei kleineren Verlagen haben sich jedoch auch offene Dateiformate ohne digitales Rechtemanagement etablieren können.

Im Bereich der DRM-geschützten Comics haben sich, neben dem Verkauf digitaler Comics zum Download, auch Streaming-ähnliche Angebote etabliert. Bei diesen können Comics gekauft oder geliehen und auf unterstützten Endgeräten per App heruntergeladen und dort gelesen werden. Zum Teil kann man bei diesen Anbietern die Comics auch direkt im Webbrowser lesen. Dem Angebot DRM-geschützter Comics ist allerdings gemein, dass die Comics nur mit den Apps der Anbieter gelesen werden können. Zum Teil ist nicht einmal ein Export der Dateien zur externen Speicherung möglich. Der größte Vorteil für die Nutzer besteht bei diesen Anbietern darin, dass die Comic-Sammlung nicht selbst verwaltet werden muss und jeder Comic bei Bedarf einfach heruntergeladen werden kann. Bekannte Anbieter DRM-geschützter Comics sind Amazon Kindle[3] und Comixology[4] (inzwischen ebenfalls Amazon).

Bezüglich der DRM-freien Comics sind vor allem kleine und mittlere Verlage, wie zum Beispiel Valiant Entertainment[5] oder Image Comics[6], zu nennen. Diese vertreiben ihre Comics häufig auch DRM-geschützt über die bereits genannten Anbieter, bieten ihre Comics aber auch im Direktvertrieb ohne DRM-Schutz an. Außerdem sind für Comics dieser Anbieter zum Beispiel bei Comixology auch DRM-freie Sicherungskopien verfügbar. Eine wichtige Besonderheit bei DRM-freien Comics besteht darin, dass diese auch häufig in größeren Paketen zu Angebotspreisen verkauft werden, so zum Beispiel durch HumbleBundle[7].

Eine digitale Comicsammlung besteht meistens aus einer Mischung aus Comics dieser beiden Kategorien. Dadurch entstehen mehrere Nachteile gegenüber der ausschließlichen Nutzung von Anbietern der ersten Kategorie. Zum einen liegt die Comicsammlung so immer fragmentiert vor, da die Comics von Anbietern der ersten Kategorie nur beim Anbieter liegen, die Comics der zweiten Kategorie hingegen nur beim Nutzer. Zum anderen ist der Komfort bei der Nutzung DRM-freier Comics

[3] https://www.amazon.de/kindle-dbs/fd/kcp
[4] https://www.comixology.com/
[5] http://valiantentertainment.com/
[6] https://imagecomics.com/
[7] https://www.humblebundle.com/

geringer, da die Sammlung nicht zentralisiert als durchstöberbarer Katalog vorliegt und so ein spontanes Lesen von Comics unterwegs erschwert wird.

Diese beiden Hauptprobleme soll ComicLib lösen. Comics beider Kategorien sollen gemeinsam in einem Katalog verwaltet werden, der den persönlichen Bibliotheken der großen Anbieter nachempfunden ist. DRM-freie Comics sollen in ComicLib Bürger erster Klasse sein, die genauso verwaltet werden wie ihre DRM-behafteten Pendants. Comics beider Kategorien sollen an Hand ihrer Serien und Verlage organisiert und durch ihr Coverbild in der Sammlung visualisiert werden. Sie sollen beide durch die Software lesbar sein – im Falle der DRM-freien Comics direkt in der Software, im Falle der DRM-behafteten durch Download und Öffnen der Drittanbieteranwendung. Die Speicherung und Verwaltung sollen zentral erfolgen, um einen Zugriff von Unterwegs zu ermöglichen.

ComicLib soll eine freie Software sein, die den proprietären Systemen der großen Anbieter eine freie Alternative für offene Comic-Dateiformate entgegensetzt. Es soll jedem möglich sein, seine gesamte Comicsammlung mit ComicLib zu verwalten. Damit jeder die Software weiterentwickeln und an seine Bedürfnisse anpassen kann, soll ComicLib unter einer Open Source Lizenz stehen.

1.2 Aufgabenstellung

Mit dem Aufkommen digitaler Bücher hat auch der Comic Einzug ins digitale Zeitalter gehalten. Heute gibt es Comics in vielen verschiedenen digitalen Formaten. Manche davon sind proprietär oder enthalten Maßnahmen zur Erschwerung der Verbreitung (Digital Rights Management, DRM), andere basieren auf bereits zuvor allgemein verwendeten Formaten. Aus der Vielfalt und Unterschiedlichkeit der Speicherformate ergeben sich besondere Anforderungen an eine gemeinsame Verwaltung und Handhabung dieser digitalen Comics.

1.2.1 Anforderungen an die Software

Da proprietäre Formate meist nur mit dem Leseprogramm des Anbieters geöffnet werden können[8], muss die Verwaltungssoftware gegenüber dem Dateiformat eines Comics möglichst agnostisch sein - das Dateiformat darf für die Speicherung der Dateien und die Verwaltung der Metadaten also keinerlei Rolle spielen. Außerdem muss die Verwaltungssoftware die Dateien so zur Verfügung stellen, dass diese einfach mit dem jeweiligen Drittanbieterprogramm geöffnet werden können.

Da die nicht-proprietären (offenen) Formate auf etablierten Dateiformaten (wie zum Beispiel ZIP[9] oder RAR[10]) basieren, lassen sich diese leicht durch die Verwendung von Standardbibliotheken nutzen. Dies ermöglicht eine tiefere Integration in die Verwaltungssoftware, so dass zusätzlich zur Bereitstellung der Dateien für Drittprogramme auch eine Aufbereitung der Dateien zur Anzeige direkt in der Software möglich und auch sinnvoll ist.

Auf Grund der unterschiedlichen Nutzbarkeit proprietärer und freier Dateiformate ist es notwendig, die Beschaffung von Metadaten, wie Serientitel, Verlag oder Titelbild, unabhängig vom Dateiinhalt zu gestalten. Um ein Durchstöbern der Sammlung zu ermöglichen werden Metadaten benötigt, die eine Zuordnung von Einzelheften und ihren Dateien zu Serien, Titelbildern, Verlagen und ihrem aktuellen Lesestatus (gelesen/nicht gelesen, sowie aktuelle Seitenzahl als Lesezeichen) ermöglichen.

Aus der Vielzahl und Diversität möglicher digitaler Endgeräte ergibt sich, dass für die Software Technologien gewählt werden müssen, die auf möglichst vielen Endgeräten nutzbar sind. Auf Grund

[8] Vgl. (Sharpened Productions, .AZW File Extension, 2017)
[9] Vgl. (Sharpened Productions, .CBZ File Extension, 2018)
[10] Vgl. (Sharpened Productions, .CBR File Extension, 2018)

der zum Teil sehr unterschiedlichen Formfaktoren und Eingabegeräte ist es außerdem notwendig, die Benutzeroberfläche an die Gegebenheiten der verschiedenen Geräteklassen anzupassen.

Zusammengefasst ergeben sich also folgende Anforderungen:

- eine gemeinsame Verwaltung und Bereitstellung von Comic-Dateien unterschiedlicher Formate
- ein Lese-Modus für Comics in freien Dateiformaten
- die Beschaffung der Metadaten zu Comics über externe Datenquellen und nutzergenerierte Daten
- die Unterstützung möglichst vieler, gängiger Betriebssysteme und Endgeräteklassen
- eine Benutzeroberfläche, die sich an die unterschiedlichen Geräteklassen und Formfaktoren anpasst, ohne dabei die Benutzerführung stark zu ändern

1.2.2 Lösungsansätze zur Erfüllung der Anforderungen

Für die gemeinsame Verwaltung von Comics in unterschiedlichen Dateiformaten muss eine Möglichkeit gefunden werden, die Informationen zu den Comics zu erlangen, ohne die Dateien öffnen zu müssen. Dafür bietet sich die Programmierschnittstelle (API) von ComicVine, der größten Onlinedatenbank für Comics, an. Die dort verfügbaren Informationen müssen jedoch lokal den Comicdateien zugeordnet werden.

Die API bietet eine Suchfunktion an, die genutzt werden könnte, um den Dateien anhand der Dateinamen Details zum Comicheft zuzuordnen. Allerdings ist dieses Verfahren sehr fehleranfällig, da es häufig mehrere Serien mit identischem Namen gibt und die Schreibweise eines Seriennamen auch nicht immer eindeutig ist. Fehlzuordnungen wären so unvermeidbar. Zudem ist die Geschwindigkeit der Suchfunktion im Vergleich zum Zugriff auf einen Datensatz mit bekannter API-Adresse sehr langsam. Hinzu kommt, dass die Zugriffsrate auf die API limitiert ist, um Überlastungen zu vermeiden. Daher müssen die Zugriffe im Abstand von mindestens einer Sekunde erfolgen.[11] Eine Suche dauert zum Teil noch länger als eine Sekunde. Bei der Verwendung des suchbasierten Ansatzes muss für jede Datei zunächst die Suche verwendet werden, um im Anschluss die Details des gefundenen Datensatzes nachzuladen. Daraus ergibt sich also pro Comicheft ein langsamer Suchvorgang und ein regulär schneller Zugriff auf die API zum Holen der Details. Hinzu kommen ein regulärer Zugriff pro Serie und Verlag, da sich diese aus dem Heft-Datensatz ergeben und hier nicht die Suche genutzt werden muss.

Die bessere Alternative besteht darin, den Nutzer die Zuordnung einmalig pro Serie vornehmen zu lassen. Der Nutzer schaut für jede Serie der Sammlung einmalig die der Serie zugeordnete Identifikationsnummer (ID) im ComicVine-Wiki[12] nach und trägt diese in eine vorgegebene INI-Datei im Dateiverzeichnis der Serie ein. Diese wird dann durch das Programm ausgelesen und der Datensatz der Serie inklusive der Liste der in der Serie enthaltenen Comichefte von der API bezogen. Pro Serie darf es nur einen Dateiordner geben, in dem alle Comicdateien der Serie, nach einem vorgegebenen Format benannt, enthalten sein müssen. Durch die standardisierte Benennung der Dateien ist eine Zuordnung der Heftnummern der Dateien zu den Heftnummern der Serienhefte aus der API leicht möglich. So ergibt sich pro Heft, Serie und Verlag jeweils nur ein regulär schneller API-Zugriff, da sich Heft und Verlag aus der Serie herleiten lassen. Die Zuordnung kann hier also wesentlich schneller erfolgen und Fehlzuordnung sind ausgeschlossen. Dank der einmaligen, manuellen Zuordnung ist es auch möglich, mehrere Instanzen von ComicLib parallel zu betreiben, ohne die Zuordnungen neu vornehmen oder korrigieren zu müssen.

Der Lesemodus setzt voraus, dass die einzelnen Seiten aus den Comicdateien entpackt werden können. Im Falle der beiden beliebtesten Dateiformate, CBR und CBZ, ist dies einfach umzusetzen, da

[11] Vgl. (CBS Interactive Inc., API Rate Limiting, 2015)
[12] https://comicvine.gamespot.com/

diese reguläre RAR- bzw. ZIP-Archive nutzen, deren Dateierweiterungen lediglich umbenannt werden. Hier reicht ein einfaches Entpacken der Archivdatei, in der die Seiten bereits in der korrekten Reihenfolge als Bilddateien vorliegen. Im Falle von PDF, das ebenfalls recht häufig für Comics verwendet wird, besteht die Notwendigkeit, aus den Seiten Bilder zu erstellen. Dies erfordert deutlich mehr Rechenzeit, weshalb die beiden anderen Formate eher zu bevorzugen sind. Um ein Lesen der Comics zu ermöglichen ist es nach dem Entpacken der Bilder notwendig, diese in der korrekten Reihenfolge anzuzeigen. Es muss möglich sein, manuell zwischen den Seiten hin und her zu navigieren. Außerdem muss die aktuelle Seitenzahl festgehalten werden, um ein Unterbrechen und späteres Fortsetzen des Lesens zu unterstützen (automatisches Lesezeichen).

Zur Unterstützung möglichst vieler Endgeräteklassen und Betriebssysteme bietet es sich an, ComicLib als eine Webanwendung umzusetzen. ComicLib wird dann zusammen mit den Comic-Dateien von einem Server bereitgestellt. Die diversen Endgeräte greifen über einen Webbrowser auf die Software zu. Anpassungen an unterschiedliche Betriebssysteme der Endgeräte sind so nicht notwendig, allerdings müssen Anpassungen für unterschiedliche Webbrowser erfolgen.

Zur Unterstützung unterschiedlicher Formfaktoren und Eingabegeräte bietet sich die Verwendung von Bootstrap an. Dieses CSS- und Javascript-Framework von Twitter bringt umfassende Funktionen zur Gestaltung von Webseiten in Responsive Design mit. Das bedeutet, dass eine gemeinsame Benutzeroberfläche auf Geräten unterschiedlicher Formfaktoren, zum Beispiel PC und Smartphone, unterschiedlich und auf den Formfaktor optimiert dargestellt wird. Außerdem enthält Bootstrap auch Funktionen zur Unterstützung von Maus- wie auch Touch-basierter Bedienung.

2 Hilfsmittel und Technologien

Im Rahmen der Durchführung des Projekts kommen diverse Softwaresysteme zum Einsatz. Dabei kann unterschieden werden zwischen den Hilfsmitteln, die zur Erstellung der Software genutzt werden, und den Technologien, die als Basis der Software dienen und von denen diese abhängig ist. Das folgende Kapitel gibt einen Einblick in die genutzten Softwaresysteme und ihre Einsatzzwecke.

2.1 Hilfsmittel

In den folgenden Teilkapiteln werden die verwendeten Hilfsmittel vorgestellt. Zu den Hilfsmitteln zählen Softwaresysteme zur Unterstützung bei Entwicklung, Design, Planung und Testen des Softwareprojekts.

2.1.1 Git

Zur Versionierung der Software wird Git[13] verwendet. Git ist eine Software zur verteilten Versionsverwaltung von Dateien. Es wird vor allem zur Versionsverwaltung von Software-Quelltext verwendet. Git ermöglicht das verteilte Zusammenarbeiten an einer gemeinsamen Quelltext-Basis. Dank des verteilten Ansatzes lässt sich ein Git-Depot einfach an viele Orte spiegeln und Änderungen können von jedem dieser Orte wieder lokal eingespielt werden. In ComicLib erfüllt Git zwei wesentliche Aufgaben. Zum einen dient Git der Versionsverwaltung an sich. Änderungen an der Software werden in Git eingepflegt und schaffen so eine nachvollziehbare Historie der Entwicklung des Projekts. Zum anderen wird der verteilte Speicheransatz genutzt, um mehrere identische Kopien des Projekts an unterschiedlichen Orten synchron zu halten. Sollte die lokale oder eine der nicht-lokalen Kopien beschädigt werden oder verloren gehen, bleibt trotzdem der aktuelle Entwicklungsstand erhalten und kann aus einer der anderen Quellen wiederhergestellt werden (redundante Speicherung).

2.1.2 GitHub

GitHub[14] ist eine Plattform zur Bereitstellung von Quellcode. GitHub stellt neben Speicherplatz für Git-Depots auch noch andere Funktionen zur Verfügung. Softwarefehler können direkt auf der GitHub-Seite eines Softwareprojekts gemeldet und diskutiert werden. Außerdem können andere Nutzer Quelltext zur Behebung von Fehlern und zur Erweiterung der Funktionalität von Softwareprojekten beitragen. Weitere Funktionen umfassen die Dokumentation des Projekts durch ein Wiki, Projektplanung mit Hilfe von Kanban-Boards und mehr. In ComicLib wird GitHub für drei Dinge verwendet. Erstens dient GitHub als eines der entfernten Git-Depots zur redundanten Speicherung des Projekts. Dabei nimmt GitHub die Rolle des Hauptdepots ein. Zum zweiten können nach der Veröffentlichung des Projekts auf der GitHub-Projektwebseite Fehler gemeldet und Quelltext beigetragen werden. Drittens, und letztens, können nach der Veröffentlichung des Projekts häufig gestellte Frage und Besonderheiten der Software im Wiki-Bereich dokumentiert werden.

2.1.3 PhpStorm

Zur Entwicklung des Quelltexts kommt die integrierte Entwicklungsumgebung (IDE) PhpStorm[15] zum Einsatz. Sie unterstützt die Softwareentwicklung des Projekts durch Funktionen wie Syntaxhervorhebung, Vorschläge zur Textvervollständigung und Vorlagen für neue Dateien. Die gut integrierte Datenbank-Erweiterung bietet ebenfalls einige Vorteile. Datenbanktabellen sind direkt in der IDE einsehbar und veränderbar. Datenbank-Definitionen und -Anfragen können direkt in der IDE programmiert und getestet werden. Basierend auf den Namen von Datenbanktabellen und deren

[13] https://git-scm.com/
[14] https://github.com/
[15] https://www.jetbrains.com/phpstorm/

Feldern wird auch hier die automatische Textvervollständigung angeboten und es gibt Syntaxhervorhebung für verschiedene Datenbanksprachen (unter anderem MySQL und Oracle SQL). PhpStorm bringt außerdem eine integrierte Eingabeaufforderung mit, die sich vor allem bei der Nutzung von Git als sehr praktisch erweist. Bei der Entwicklung von ComicLib wird außerdem die Docker-Erweiterung für PhpStorm genutzt. Diese ermöglicht ein einfaches Erstellen, Starten, Stoppen und Löschen von Docker-Containern sowie Syntax-Unterstützung für Docker-Konfigurationsdateien.

2.1.4 Postman

Postman[16] ist eine Entwicklungsumgebung für Programmierschnittstellen (APIs). Es ermöglicht das Entwerfen, Testen, Simulieren und Dokumentieren von APIs. Für die Entwicklung von ComicLib wird Postman für mehrere Dinge eingesetzt. Die Anfragen an die ComicVine API werden erst mit Postman getestet, bevor sie in ComicLib programmiert werden. Dadurch können die Anfragen an die API und die ComicLib-interne Komponente zur Kommunikation mit der API unabhängig voneinander entwickelt und optimiert werden. Das erleichtert die Fehleranalyse in diesem Teil der Softwareentwicklung enorm. Bei der Entwicklung der ComicLib API kommt Postman zum Testen und Dokumentieren der programmierten API-Ressourcen zum Einsatz. Die unterschiedlichen Anfragen an die ComicLib API werden in Postman angelegt und deren Ergebnisse auf Korrektheit geprüft sowie die möglichen Ergebnisse dokumentiert. Dabei erstellt Postman automatisch eine Webseite mit der Dokumentation, die nach Veröffentlichung einfach auf der Projektwebseite referenziert werden kann.

2.1.5 Todoist

Die Durchführung der Softwareentwicklung von ComicLib erfordert ein gewisses Mindestmaß an zeitlicher und struktureller Planung. Zur Unterstützung der Planung kommt Todoist[17] zum Einsatz. Todoist ist eine Software zur Planung von Aufgaben und Projekten. Aufgaben können in Projekten organisiert und bis zu vier Ebenen tief verschachtelt werden. Dadurch lassen sich Komponenten und ihre Subkomponenten gut darstellen. Aufgaben können ein Fälligkeitsdatum, eine Priorität und Etiketten haben, wodurch sowohl eine zeitliche als auch eine strukturelle Planung unterstützt werden. So ist es möglich, in einem Projekt „Praxis-Projekt“ eine Aufgabe „Durchführung“ mit den unterschiedlichen Komponenten und Subkomponenten von ComicLib als Unteraufgaben anzulegen und diese durch Etiketten in „zu erledigen“ und „in Bearbeitung“ aufzuteilen, sowie die Aufgaben mit Fälligkeitsdaten und Prioritäten über den Projektzeitraum zu verteilen. Auf diese Weise kann mit geringem Aufwand das Projekt strukturiert und ein Überblick bewahrt werden.

2.1.6 Draw.io

Draw.io[18] ist eine Webanwendung zur Erstellung von Diagrammen unterschiedlichster Art. Sie unterstützt die Gestaltung von UML- und Venn-Diagrammen, aber auch von Mind-Maps, Flowcharts, Torten- und Balken-Diagrammen und vielem Mehr. Bei der Entwicklung von ComicLib kommt Draw.io zur Erstellung von Entity Relationship Diagrammen beim Datenbankdesign zum Einsatz. Dabei werden die Tabellen mit Hilfe der Krähenfuß-Notation in Beziehung zu einander gesetzt. Draw.io ermöglicht den Export von Diagrammen als Grafik sowie als XML-Datei. Die XML-Datei des ComicLib ER-Diagramms wird zusammen mit dem Quelltext im Git-Depot gespeichert, so dass ein späteres Überarbeiten des Diagramms mit Hilfe von Draw.io möglich ist.

---

[16] https://www.getpostman.com/
[17] https://todoist.com/
[18] https://www.draw.io/

2.2 Technologien

ComicLib ist eine datenbankbasierte Webanwendung. Daher werden zum Betrieb von ComicLib ein Webserver sowie ein Datenbankmanagementsystem benötigt. Diesen und den weiteren zum Einsatz kommenden Technologien widmet sich das folgende Kapitel.

2.2.1 Programmiersprachen

Am Anfang der Auswahl der Technologien zur Umsetzung von ComicLib steht die Wahl der richtigen Programmiersprachen. Da manche der häufig zur Entwicklung von Webanwendungen genutzten Back-End-Programmiersprachen ihr eigenes Framework inklusive eines eigenen Webservers mitbringen (zum Beispiel Ruby on Rails[19] oder Django[20]), ist es ratsam, die Wahl dieser Sprache an den Anfang zu setzen, da sich so die Wahl eines Webservers mitunter erübrigt.

ComicLib verwendet im Back-End PHP[21] in Version 7. Heutzutage werden Webanwendungen auch häufig in Javascript (über Node.js[22]), Python (über Django) oder Ruby (über Ruby on Rails) entwickelt, jedoch ist PHP weiterhin die klassische Wahl. Daher finden sich Inhalte zu integrierten Funktionen, Verhaltensweisen, Best Practices und Standardbibliotheken sehr zahlreich und in allgemein guter Qualität in Internet[23] und Literatur.

Zur Bereitstellung dynamischer Inhalte kommt im Front-End Javascript[24] zum Einsatz. Alternativen zu Javascript sind zum Beispiel Typescript oder Dart, um nur zwei zu nennen. Jedoch ist all diesen Sprachen gemein, dass sie zu Javascript kompiliert werden müssen, da die Webbrowser nur Javascript interpretieren können. Da Javascript, wie PHP, die klassische Wahl und entsprechend gut dokumentiert[25] ist, spricht nichts dagegen, auf die Nutzung einer der anderen Sprachen zu verzichten.

Sowohl Javascript als auch PHP werden in ComicLib um einige Bibliotheken erweitert. Das ist notwendig, um manche der Funktionen bereitstellen zu können. Dazu zählen unter anderem die Anbindung an die Datenbank sowie das Entpacken der Comic-Dateien. Genaueres dazu ist dem Kapitel 3.4.1 Verwendete Bibliotheken zu entnehmen.

2.2.2 Apache

Da PHP als Programmiersprache verwendet wird, ist es notwendig, zusätzlich eine Webserver-Software zu wählen. Die beiden meistgenutzten Webserver im Internet sind NGINX[26] und Apache[27]. Best Practices, Konfiguration und Co. sind für Apache 2 wie auch für NGINX gut und zahlreich in Internet und Literatur dokumentiert. Auf Grund von Erfahrungen mit Apache aus vorangegangenen Projekten fiel die Wahl auf den Apache Webserver in Version 2.

2.2.3 MySQL

Zur Bereitstellung der Datenbank kommt MySQL[28] zum Einsatz. MySQL ist, ähnlich wie PHP und Javascript, die klassische Wahl. Auch hier ist die Dokumentation in Literatur und Internet recht gut. Alternative Datenbanksysteme sind beispielsweise MariaDB, PostgreSQL oder Oracle SQL. Auch nicht-SQL-basierte Datenbanksysteme kämen in Frage. Auf Grund von Vorerfahrungen wurde für ComicLib

[19] https://rubyonrails.org/
[20] https://www.djangoproject.com/
[21] https://www.php.net/
[22] https://nodejs.org/en/
[23] Vgl. (The PHP Group, 2019)
[24] https://developer.mozilla.org/en-US/docs/Web/JavaScript
[25] Vgl. (Mozilla, 2019)
[26] https://www.nginx.com/
[27] https://httpd.apache.org/
[28] https://www.mysql.com/de/

MySQL gewählt. Das Datenbankmanagementsystem kommt in Version 5.5 zum Einsatz. Diese Version ist veraltet, die aktuelle Version trägt die Versionsnummer 8.0. Da ComicLib auf Computern mit ARM-Prozessor lauffähig sein soll, muss MySQL 5.5 verwendet werden. Eine aktuellere Version ist für ARM schlicht und einfach nicht verfügbar. Glücklicher Weise sind alle durch ComicLib benötigten Datenbankfunktionen bereits in MySQL 5.5 enthalten. Mit dem Verfügbarwerden neuerer MySQL-Versionen für die ARM-Architektur ist ein Upgrade der im Projekt verwendeten Version ratsam, um Sicherheitsprobleme zu vermeiden. Bei Nutzung der Software auf Computern mit AMD64-Prozessorarchitektur ist schon jetzt der Einsatz einer aktuelleren Version möglich und auch zu empfehlen.

2.2.4 Docker

ComicLib ist auf den Betrieb in einer Docker-Umgebung ausgelegt. Docker[29] ist eine Software zur Containerisierung. Bei der Containerisierung werden eine oder mehrere Instanzen eines Betriebssystem parallel auf einem unterliegenden Host-Betriebssystem betrieben. Docker erstellt dabei die Container aus so genannten Images, die aus fertigen Betriebssystemabbilder inklusive vorinstallierter Software bestehen. Jeder Container ist eine lauffähige Instanz eines Betriebssystemabbilds. Durch die Virtualisierung der Betriebsumgebung ist ComicLib auf Linux, MacOS und Windows lauffähig, ohne speziell an jedes der Betriebssysteme angepasst werden zu müssen.

[29] https://www.docker.com/

3 Projektdurchführung

Das folgende Kapitel widmet sich der Durchführung des Projekts. Es erläutert Designentscheidungen und Details der Softwarearchitektur sowie Eigenheiten der Projektumgebung und der Implementierung.

3.1 Infrastruktur

Die Server-Infrastruktur, auf der ComicLib als Webanwendung aufsetzt, ist mit Docker virtualisiert. Docker ist eine Software zur Container-Virtualisierung, mit der mehrere Instanzen eines Betriebssystems auf einem Host-Betriebssystem betrieben werden können. Im Vergleich zur herkömmlichen Virtualisierung mittels Hypervisor ist Container-Virtualisierung deutlich ressourcenschonender, wodurch auch die wirtschaftliche Nutzung von Virtualisierung zur Bereitstellung einzelner Anwendungen machbar ist.[30] Für den Webserver und den Datenbankserver jeweils einen eigenen Container zu verwenden, stellt so kein Problem dar.

Docker-Container basieren auf so genannten Images. Dabei handelt es sich um Systemabbilder mit bereits vorinstallierter Software. Docker Images können aus dem Docker Hub[31] Online-Depot bezogen werden, wo sie von der Community, sowie im Falle mancher spezialisierter Images von den Entwicklern der jeweiligen Spezialsoftware, betreut und aktualisiert werden. Außerdem ist es möglich, Docker Images auf Basis existierender Images selbst zu bauen. Für ComicLib werden vorgefertigte, spezialisierte Images genutzt. Im Falle des Images für den Webserver wird aus dem Basis-Image allerdings ein neues Image gebaut, das zusätzlich die für ComicLib benötigten PHP-Erweiterungen sowie deren Abhängigkeiten enthält.

ComicLib nutzt zwei Container, die jeweils ein auf Debian 9 basierendes Image verwenden. Der Container zur Bereitstellung des Webservers betreibt Apache 2 und nutzt dazu das offizielle „PHP“ Image, das von der Docker Community gepflegt wird. Der Container zur Bereitstellung der Datenbank auf AMD64-basierten Computern betreibt MySQL 5.5 und nutzt dazu das offizielle „MySQL“ Image, das von der Docker Community und den MySQL Entwicklern gepflegt wird. Der Container für die Datenbank zum Betrieb auf ARM-basierten Computern nutzt als einziger kein offizielles Image, da es kein offizielles MySQL Image für ARM gibt. Stattdessen kommt eine angepasste Version des offiziellen Images zum Einsatz, das von den Entwicklern von HypriotOS, einem speziell für Containerisierung per Docker auf Raspberry Pi Minicomputern ausgelegten Betriebssystem, gepflegt wird.

Die beiden Container werden durch eine Docker-Compose-Konfiguration zusammengehalten. Diese erstellt und verwaltet beide Container zusammen und sorgt dafür, dass sie beide im selben, virtualisierten Netzwerk laufen. Außerdem sorgt Docker-Compose dafür, dass die diversen Konfigurationsdateien und das Verzeichnis mit dem Quelltext in den Webserver-Container hinein verlinkt wird. Zudem wird das Datenverzeichnis von MySQL ins lokale Projektverzeichnis verlinkt, wodurch ein Backup der Datenbankdateien möglich wird.

Der größte Vorteil des Betriebs von ComicLib über Docker-Compose besteht darin, dass die Installation der Software stark vereinfacht wird. Bei konventioneller Installation müssten erst MySQL, Apache 2, PHP und die diversen PHP-Erweiterungen installiert werden[32], bevor dann im Anschluss alle Konfigurationsdateien in die entsprechenden Verzeichnisse kopiert werden müssten. Danach müsste dann noch der Webserver neu gestartet werden. Bei konventioneller Installation müssten außerdem die Namen der benötigten Pakete womöglich dem Host-Betriebssystem angepasst werden, eventuell

[30] Vgl. (Docker Inc., 2019)
[31] https://hub.docker.com/
[32] Vgl. (Drake, 2018)

wären manche der Pakete auch gar nicht verfügbar. Bei Nutzung von Docker-Compose reduziert sich die Anzahl der zu installierenden Pakete auf zwei (Docker und Docker-Compose) und die gesamte Installation erfolgt mit nur einem Befehl auf der Kommandozeile. Dank fester Versionen der zugrunde liegenden Images ist ein Fehlen von Paketen dabei ausgeschlossen.

Ein weiterer Vorteil der Nutzung von Docker und Docker-Compose besteht darin, dass zwischen der Entwicklungsumgebung und der Produktivumgebung der Software kein Unterschied besteht[33]. Ein häufiges Problem im Bereich der Softwareentwicklung liegt darin, dass eine Software nur auf dem System des Entwicklers lauffähig ist. Das passiert, wenn dieser bei Problemen zwar sein eigenes System anpasst, um den Fehler zu beheben (zum Beispiel durch Nachinstallation einer fehlenden Abhängigkeit), dabei aber vergisst, diese Anpassung zu dokumentieren oder zu implementieren. Da bei der Nutzung von Docker-Compose die Konfigurationsdateien in die Container hinein verlinkt werden, können diese zusammen mit dem Quelltext verwaltet werden. Da die Images und Container auf Basis von Skripten erstellt und mit den benötigten Abhängigkeiten versehen werden, können auch alle Abhängigkeiten zusammen mit dem Quelltext verwaltet werden. Quelltext, Systemkonfiguration und Abhängigkeiten können dank Docker-Compose zusammen im selben Projekt gepflegt werden.

Dank der Nutzung von Docker-Compose ist ComicLib auf Linux, MacOS und Windows gleichermaßen lauffähig, da Docker-Compose für alle drei Betriebssysteme verfügbar ist. Zudem läuft so ComicLib immer auf einem Debian Betriebssysteme, wodurch in der Software keine Rücksicht auf unterschiedliche Betriebssysteme genommen werden muss.

3.2 Datenbankdesign

Das nachfolgende Kapitel gibt Einblick in das Datenbankdesign hinter ComicLib. Da große Teile der Daten zu den Comics aus der ComicVine API bezogen werden, wird zunächst auf die verwendeten API-Ressourcen und -Felder eingegangen.

3.2.1 ComicVine API

ComicLib bezieht die meisten Daten zu den Comicheften, -serien und -verlagen aus der ComicVine API von GameSpot. Bei ComicVine handelt es sich um die größte Online-Datenbank für Comics. Die Datensätze der Datenbank werden von der Community in Form eines Wikis gepflegt. Die API stellt diese Daten strukturiert und durchsuchbar als JSON- und XML-Datensätze zur Verfügung. Die API ist gemäß dem REST (Representational State Transfer) Paradigma gestaltet. Die API-Ressourcen sowie ihre Felder und die anwendbaren Filter sind auf der ComicVine API Webseite[34] dokumentiert.

Verwendete Felder der API-Ressource *Issue* (Heft)

Feldname	Inhalt	Resultierender Datentyp
api_detail_url	URL der Ressource mit den Details dieses Hefts	TEXT
description	Beschreibung des Inhalts des Hefts	TEXT
id	Einzigartige ID des Hefts	VARCHAR (min. 6)
image:	Titelbild des Hefts in verschiedenen Größen (Objekt)	-
small URL	URL des Titelbilds in kleiner Auflösung	TEXT
issue number	Heftnummer innerhalb der Serie	VARCHAR (min 5.)
volume:	Serie, deren Teil das Heft ist (Objekt)	Eigene Tabelle
id	Einzigartige ID der Serie	VARCHAR (min. 5)

Tabelle 1: Verwendete Felder der API-Ressource Issue

[33] Vgl. (Perry, 2016)
[34] Vgl. (CBS Interactive Inc., ComicVine API Documentation, 2019)

Die Ressource *Issue* enthält die Daten zu einzelnen Comicheften. Um ein Aktualisieren der Datensätze zu ermöglichen, wird das Feld *api_detail_url* benötigt, das die URL zum jeweiligen Datensatz enthält. Das Feld *description* enthält die Beschreibung des jeweiligen Hefts, die dem Leser Informationen zum Inhalt gibt. Das Feld *id* enthält die eindeutige Identifikationsnummer des Datensatzes in der ComicVine API. Diese wird benötigt, um den Datensatz, zum Beispiel innerhalb der dazugehörigen Serie, eindeutig zu identifizieren und zu verknüpfen. Aus dem Objekt *image* wird das Feld *small_url* benötigt. Dieses Feld enthält die URL des Titelbilds des Comichefts in kleiner Auflösung. Das *image*-Objekt enthält Links für mehrere unterschiedlich große und unterschiedlich zugeschnittene Versionen desselben Bilds. *small_url* enthält die kleinste Version des Bilds im normalen Coverbild-Seitenverhältnis. Die Bildauflösung ist groß genug für die Darstellung auf allen unterstützten Geräteklassen – vom Smartphone bis Desktop-Monitor – ohne sichtbare Qualitätsmängel. Die Nutzung der nächst größeren Variante würde die Aufbaugeschwindigkeit der Weboberfläche bereits deutlich verringern. Für die Zuordnung von Comicdateien zu ihren Heftdaten und zur Sortierung innerhalb der Serie wird die Heftnummer benötigt, die dem Feld *issue_number* entnommen werden kann. Dabei muss beachtet werden, dass eine klassische Heftnummer zwar eine Ganzzahl ist, die ComicVine API jedoch auch andere Zeichen in Heftnummern zulässt, da es auch Serien mit Heftnummern wie beispielsweise „1/2“, „1a“, „α“ oder in römischer Zählweise geben kann. Aus dem Objekt *volume* wird das Feld *id* benötigt, das die Zuordnung zur übergeordneten Serie des Comichefts herstellt.

Verwendete Felder der API-Ressource *Volume* (Serie)

Feldname	Inhalt	Resultierender Datentyp
api_detail_url	URL der Ressource mit den Details dieser Serie	TEXT
description	Beschreibung des Inhalts der Serie	TEXT
id	Einzigartige ID der Serie	VARCHAR (min. 5)
image:	Titelbild der Serie in verschiedenen Größen (Objekt)	-
small_url	URL des Titelbilds in kleiner Auflösung	TEXT
name	Name der Serie	TEXT
publisher:	Verlag der Serie (Objekt)	Eigene Tabelle
id	Einzigartige ID des Verlags	VARCHAR (min. 4)
issues	Liste der Hefte der Serie als Objekte	Eigene Tabelle
start_year	Jahr des Serienstarts	INTEGER

Tabelle 2: Verwendete Felder der API-Ressource Volume

Die Ressource *Volume* enthält die Daten zu Comicserien. Viele Felder sind hier identisch zu den gleichnamigen Feldern der *Issue* Ressource. Das Feld *api_detail_url* enthält die URL zum jeweiligen Datensatz, die zur Aktualisierung benötigt wird. Das Feld *description* enthält die Beschreibung der Serie, die in ComicLib zusammen mit den Heften der jeweiligen Serie angezeigt werden soll. Das Feld *id* enthält die eindeutige Identifikationsnummer des Datensatzes in der ComicVine Datenbank, die unter anderem die Verknüpfung mit den Heften der Serie ermöglicht. Dem Objekt *image* kann aus dem Feld *small_url* der Link zum Coverbild der Serie entnommen werden. Dabei handelt es sich in der Regel um das Coverbild des ersten Hefts der Serie, allerdings ist bei Sammelschubern auch die Frontseite des Schubers als Serienbild möglich. Das Feld *name* enthält den Namen der Serie, anhand dessen die Serie in ComicLib in der Serienübersicht einsortiert wird. Der Serienname wird dort zusammen mit dem Serientitelbild angezeigt. Außerdem kann in ComicLib anhand des Seriennamens nach Serien gesucht werden. Das Objekt *publisher* enthält im Feld *id* die eindeutige Identifikationsnummer des Verlags. Darüber ist die Zuordnung von Serien zum herausgebenden Verlag möglich. Diese wird in ComicLib zum Generieren einer Sammelansicht aller Serien eines Verlags genutzt. Das Objekt *issues* enthält Informationen zu allen Heften einer Serie, allerdings mit weniger Details als in der *Issue* Ressource. ComicLib nutzt die Heftnummern und Heft-IDs aus dieser Liste, um

die Dateinamen der Comics den Heften zuzuordnen. Das Feld *start_*year enthält das Erscheinungsjahr des ersten Hefts einer Serie und wird in der Serienübersicht zusätzlich zu Name und Coverbild angezeigt.

Verwendete Felder der API-Ressource *Publisher* (Verlag)

Feldname	Inhalt	Resultierender Datentyp
api_detail_url	URL der Ressource mit den Details dieses Publishers	TEXT
description	Beschreibung des Verlags	TEXT
id	Einzigartige ID des Verlags	VARCHAR (min. 5)
image:	Logo des Verlags in verschiedenen Größen (Objekt)	-
small_url	URL des Logos in kleiner Auflösung	TEXT
name	Name des Verlags	TEXT

Tabelle 3: Verwendete Felder der API-Ressource Publisher

Die Ressource *Publisher* enthält Daten zu den Comicverlagen. Auch hier finden sich wieder Felder, die aus den anderen Ressourcen bereits bekannt sind. Das Feld *api_detail_url* enthält die URL zum jeweiligen Datensatz, die zur Aktualisierung benötigt wird. Das Feld *description* enthält die Beschreibung des Comicverlags, die in ComicLib zusammen mit den Serien eines Verlags angezeigt werden soll. Das Feld *id* enthält die eindeutige Identifikationsnummer des Datensatzes in der ComicVine API. Das Objekt *image* enthält im Feld *small_url* den Link auf ein Bild des Unternehmenslogos des Verlags in kleiner Auflösung.

3.2.2 Datenbankarchitektur

ComicLib verwaltet die Comics, Serien und Verlage in einer MySQL Datenbank. Das folgende Entity Relationship Diagramm (Abbildung 1: Entity Relationship Diagramm der ComicLib Datenbank) gibt die Tabellen und deren Beziehungen zueinander wieder.

Die Tabellen *Issues*, *Publishers* und *Volumes* enthalten Daten, die aus der ComicVine API bezogen werden. Die Tabellen *Issues* und *Volumes* enthalten zudem Informationen zu den lokalen Speicherpfaden der Serien und ihrer Comicdateien. Die Tabellen *Users* und *UserGroups* enthalten Informationen zu Nutzern und Gruppen. Die Tabelle *ReadStatus* enthält Informationen zum Lesestatus von Comicheften.

Publishers: Die Tabelle *Publishers* enthält die Informationen zu den Comicverlagen der in der Comicsammlung enthaltenen Comichefte. Als Primärschlüssel *PublisherID* wird die ID aus der ComicVine API verwendet. Die Felder *APIDetailURL*, *Description* und *Name* enthalten die unveränderten Informationen ihrer Entsprechungen aus der API. Das Feld *ImageFileName* enthält den Dateinamen der lokal gespeicherten Kopie des Verlagslogos, während das Feld *ImageURL* die dazugehörige URL aus dem API-Feld *small_url* enthält. Die Tabelle steht in einer 1:0-n-Beziehung zu *Volumes* (jeder *Publisher* verlegt 0 – n *Volumes*), die über *PublisherID* als Fremdschlüssel in *Volumes* angelegt ist.

Volumes: Die Tabelle *Volumes* enthält die Informationen zu den Serien der lokalen Comicsammlung. Als Primärschlüssel *VolumeID* wird wie bei *Publishers* die ID aus der ComicVine API verwendet. Als Fremdschlüssel enthält die Tabelle die *PublisherID* des die Serie herausgebenden Verlags in einer 0-n:1-Beziehung (jedes *Volume* wird von genau einem *Publisher* verlegt). Die Felder *APIDetailURL*, *Description*, *Name* und *StartYear* enthalten, wie bei *Publishers*, jeweils die Informationen aus ihren API-Pendants. Analog zu *Publishers* enthalten die Felder *ImageFileName* und *ImageURL* den Dateinamen der lokalen Bilddatei des Seriencoverbild sowie dessen API-URL. Das Feld *VolumeLocalPath* enthält den Pfad zum Speicherort des Ordners mit den Comicdateien der Serie. Pro

Serie darf es nur genau einen Ordner geben. Die Tabelle steht in einer 1:0-n-Beziehung zu *Issues* (jedes *Volume* enthält 0 – n *Issues*), die über *VolumeID* als Fremdschlüssel in *Issues* angelegt ist.

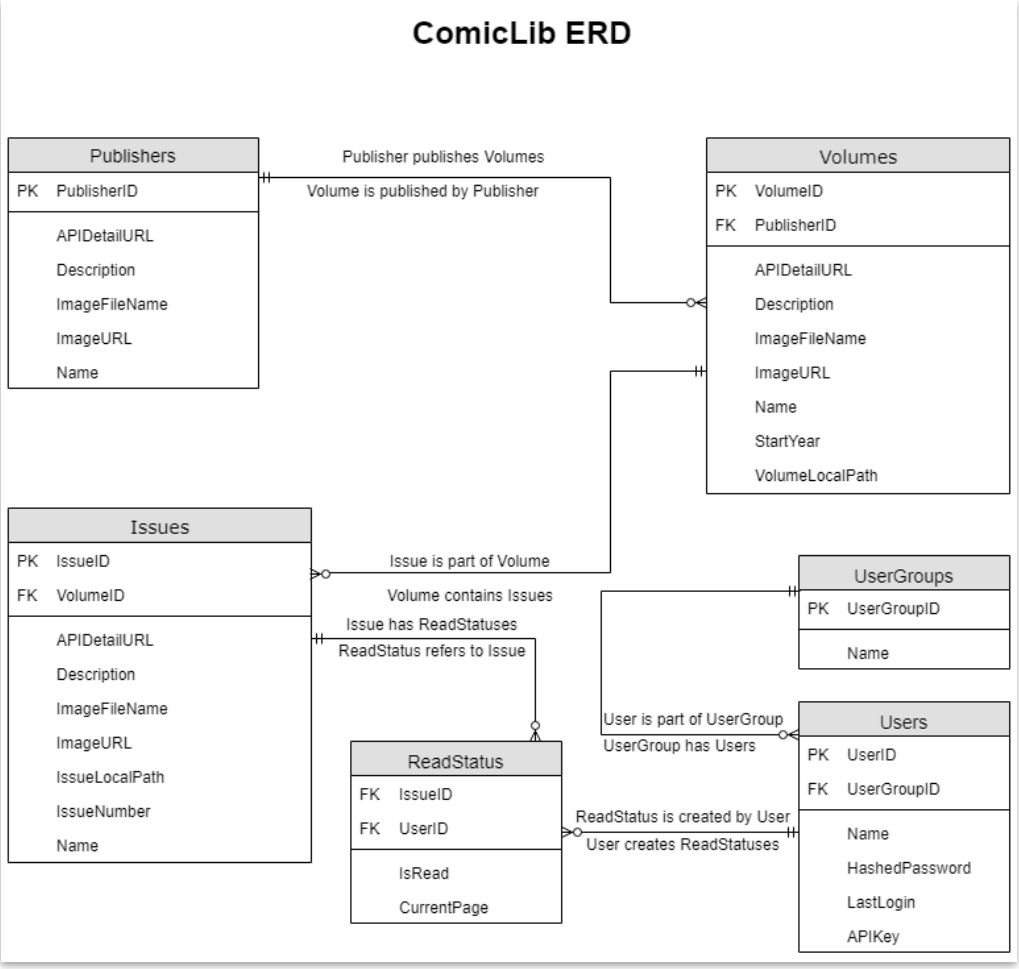

Abbildung 1: Entity Relationship Diagramm der ComicLib Datenbank

Issues: Die Tabelle *Issues* enthält die Informationen zu den Comicheften der Comicsammlung. Als Primärschlüssel *IssueID* dient hier, analog zu den beiden vorhergegangenen Tabellen, die ID aus der ComicVine API. Als Fremdschlüssel enthält die Tabelle die *VolumeID* der Serie, in der das Comicheft erschienen ist, in einer 0-n:1-Beziehung (jedes Issue ist Teil genau eines *Volumes*). Analog zu den beiden vorangegangenen Tabellen enthält *Issues* die Felder *APIDetailURL*, *Description*, *Name* und *IssueNumber* für die Daten aus den entsprechenden Feldern der ComicVine API-Datensätze. Analog zu den anderen beiden Tabellen enthalten *ImageFileName* und *ImageURL* den Dateinamen der lokal gespeicherten Bilddatei mit dem Coverbild des Comichefts sowie die API-URL des Coverbilds. Das Feld *IssueLocalPath* enthält den Dateinamen der Comicdatei des Comichefts. Der Pfad zu dieser Datei ergibt sich aus dem Pfad zum Stammverzeichnis der Comicsammlung (durch ComicLib vorgegeben), dem *VolumeLocalPath* der Serie und dem *IssueLocalPath* des Comichefts. Daraus ergibt sich, dass alle Hefte einer Serie im selben Verzeichnis liegen müssen. Die Tabelle steht in einer 1:0-n-Beziehung zu *ReadStatus* (jede *Issue* hat 0 – n *ReadStati*, abhängig von der Anzahl der Benutzer), die über *IssueID* als Fremdschlüssel in *ReadStatus* angelegt ist.

ReadStatus: Die Tabelle *ReadStatus* dient der Verwaltung des Lesestatus von Comicheften. Da der Lesestatus festhält, ob ein Benutzer ein Heft bereits gelesen hat, enthält die Tabelle die beiden

Fremdschlüssel *IssueID* und *UserID*, die sich auf die Tabellen *Issues* und *Users* beziehen. Zwischen *Issues* und *ReadStatus* besteht eine 0-n:1-Beziehung (jeder *ReadStatus* hat genau eine *Issue*). Zwischen *ReadStatus* und *Users* besteht eine 0-n:1-Beziehung (jeder *ReadStatus* hat genau einen *User*). Datenbanktrigger stellen sicher, dass es für jede Kombination aus einem Benutzer und einem Comicheft einen Eintrag in *ReadStatus* gibt und dieser Zustand auch im Falle des Hinzufügens eines Benutzers oder Hefts konsistent bleibt. Das Feld *IsRead* hält fest, ob das Heft bereits gelesen wurde oder nicht. Das Feld *CurrentPage* hält die aktuelle Seitenzahl fest. Wurde der Comic noch nicht zum Lesen geöffnet, ist die Seitenzahl 0. Wird der Comic gelesen, wird in *CurrentPage* die Seitenzahl der zuletzt betrachteten Seite gespeichert, um so ein Fortsetzen an dieser Stelle zu ermöglichen. Wird die letzte Seite erreicht, wechselt *IsRead* zum Wahrheitswert „wahr“ und die Seitenzahl wird auf 0 zurückgesetzt.

Users: Die Tabelle *Users* speichert die Informationen zu den in ComicLib angelegten Benutzern. Diese Daten werden zur Authentifizierung vor der Nutzung der Webanwendung sowie der ComicLib API benötigt. Die Tabelle enthält als Primärschlüssel *UserID* eine automatisch erstellte, einzigartige Identifikationsnummer. Der Fremdschlüssel *UserGroupID* bezieht sich auf die Tabelle *UserGroups* und enthält die eindeutige Identifikationsnummer der Gruppe des Benutzers. Die Beziehung zu *UserGroups* ist eine 0-n:1-Beziehung (jeder *User* hat genau eine *UserGroup*). Das Feld *Name* enthält den Benutzernamen des Benutzers. Das Feld *HashedPassword* enthält das gehashte und gesalzene Passwort des Benutzers. Das Hashing erfolgt dabei über die PHP-eigene Funktion *password_hash* mit dem BCrypt-Algorithmus. Das Feld *LastLogin* enthält den Zeitstempel des letzten erfolgreichen Logins und dient dazu, einen Benutzer automatisch von der Webanwendung abzumelden, wenn ein festgelegter Zeitraum überschritten wird. Das Feld *APIKey* enthält einen einzigartigen, automatisch generierten Schlüssel in Form einer langen Hexadezimal-Zeichenkette und dient der Authentifizierung gegenüber der ComicLib API. Zurzeit ist ein Anlegen von Benutzern aus ComicLib heraus noch nicht möglich. Die Tabelle wurde im Vorausblick auf einen zukünftigen Mehrbenutzerbetrieb angelegt. Die Authentifizierung und den Lesestatus nicht bereits jetzt auf einen Mehrbenutzerbetrieb auszulegen, würde später einen deutlich höheren Änderungsaufwand bedeuten. Zurzeit wird automatisch anhand der ComicLib Konfigurationsdatei ein Standardbenutzer mit den dort festgelegten Zugangsdaten angelegt und auch aktualisiert. Eine Änderung des Passworts oder Benutzernamen in der Konfigurationsdatei führt also auch zur Änderung der entsprechenden Information in der Datenbank. Die Tabelle steht in einer 1:0-n-Beziehung zu *ReadStatus* (jeder *User* hat 0 – n *ReadStati*, abhängig von der Anzahl der Hefte), die über *UserID* als Fremdschlüssel in *ReadStatus* angelegt ist.

UserGroups: Die Tabelle *UserGroups* enthält die Informationen zu den Benutzergruppen. Der Primärschlüssel *UserGroupID* enthält eine eindeutige, automatisch generierte Identifikationsnummer. Das Feld *Name* enthält den Namen der Gruppe. Diese Tabelle wurde im Vorausblick auf einen zukünftigen Mehrbenutzerbetrieb von ComicLib angelegt. Zurzeit werden Benutzergruppen noch nicht genutzt. ComicLib legt jedoch automatisch zwei Gruppen an. Die Gruppe mit der ID 0 und dem Namen „Administrators“ wird automatisch dem Standardbenutzer zugewiesen. In einem zukünftigen Mehrbenutzerbetrieb könnte so zum Beispiel das Aktualisieren der Datenbank und das Anlegen neuer Benutzer den Mitgliedern der „Administrators“-Gruppe vorbehalten bleiben. Die Gruppe mit der ID 1 und dem Namen „Users“ hat zurzeit keine Benutzer, könnte in Zukunft aber alle Nicht-Administratoren beherbergen. Die Tabelle steht in einer 1:0-n-Beziehung zu *Users* (jede *UserGroup* hat 0 – n *Users*), die über *UserGroupID* als Fremdschlüssel in *Users* angelegt ist.

Zusätzlich zu den oben erläuterten Tabellen verwendet die ComicLib-Datenbank Trigger und Views. Der Trigger *CreateReadStatusAfterUserInsert* wird aktiviert, wenn ein neuer Benutzer angelegt wird und erstellt für den Benutzer einen *ReadStatus* für jedes existierende Comicheft. Der Trigger

CreateReadStatusAfterIssueInsert wird ausgeführt, wenn ein neues Comicheft angelegt wird und erstellt für jeden Benutzer einen neuen *ReadStatus* für das neue Comicheft.

Die Datenbankviews werden verwendet, um Daten aus mehreren Tabellen zusammenzufügen und so die Beschaffung von zusammengesetzten Datensätzen im PHP-Teil zu vereinfachen. Dadurch fallen die MySQL-Statements im PHP-Teil einfacher aus, wodurch die Fehlerbehebung vereinfacht wird. Die View *VolumeIssueCount* berechnet zu jeder *VolumeID* die Anzahl der *Issue*-Datensätze, die diesem *Volume* zugeordnet sind. Sie dient der Bereitstellung der Heftanzahl für die Serienübersicht in ComicLib und wird außerdem verwendet, um zu berechnen, ob alle Hefte einer Serie gelesen wurden. Die View *PublisherVolumes* fügt die Tabellen *Publishers* und *Volumes* anhand der *PublisherID* zusammen und bezieht für die *Volumes* auch die Heftanzahl aus *VolumeIssuesCount* mit ein. *VolumeIssues* fügt die Tabellen *Volumes* und *Issues* anhand der *VolumeID* zusammen, wodurch jeder *Issues*-Datensatz um das Feld *VolumeLocalPath* erweitert wird. Die View *VolumeReadStatus* dient dazu, für jede Kombination aus einem Benutzer und einer Serie den *ReadStatus* der Serie zu bestimmen. Dazu muss berechnet werden, ob die Anzahl der Hefte der Serie, die als gelesen markiert sind, mit der Gesamtzahl der Hefte der Serie übereinstimmt. Diese Information wird benötigt, um in der Serien-Übersicht von ComicLib die Serien mit „New“ zu markieren, die noch nicht komplett gelesen wurden. Die View *IssueReadStatus* fügt für jeden Benutzer jedem Heft den Lesestatus an. Dadurch wird im PHP-Teil zum Beschaffen des Lesestatus eines oder mehrerer Hefte nur noch ein WHERE-Statement mit der *UserID* des anfragenden Nutzers benötigt.

3.3 Back-End

Das folgende Kapitel widmet sich dem Design und den Designentscheidungen bei der Entwicklung des Back-End-Teils von ComicLib. Das Back-End ist für die Verwaltung der Comics, die Kommunikation mit der ComicVine API und die Bereitstellung der Views der Webanwendung sowie der Ressourcen der ComicLib API zuständig. Es ist in PHP 7 programmiert.

3.3.1 Architektur

ComicLib setzt auf eine geordnete Verzeichnisstruktur zur Organisation der Klassen, Abhängigkeiten und sonstigen Dateien und Ordner. Auf die Verwendung von Namespaces in PHP wurde verzichtet. Das Quelltext-Stammverzeichnis von ComicLib gliedert sich in die Verzeichnisse *cache*, *php_includes*, *resources* und *storage*. Außerdem befinden sich die Dateien *.htaccess*, *index.php*, *log.txt* und *updater.lock* im Stammverzeichnis.

Das Verzeichnis *cache* dient als Stammverzeichnis für alle Inhalte, die durch ComicLib zwischengespeichert werden müssen. Das Unterverzeichnis *comics* enthält die zur Anzeige im Lesemodus entpackten Comicdateien in jeweils einzelnen Ordnern. Das Unterverzeichnis *images* enthält die für die Anzeige in der Webanwendung zwischengespeicherten Verlagslogos, Serientitelbilder und Heftcoverbilder.

php_includes enthält alle PHP-Dateien von ComicLib. In diesen spielt sich die wesentliche Programmlogik ab. Die Unterverzeichnisse organisieren die PHP-Dateien in Funktionsgruppen. *Authentication* enthält die Klassen für die Authentifizierung über die Web-Oberfläche sowie über die API. In *Caching* befinden sich die Klassen für die Zwischenspeicherung von Bildern der ComicVine API sowie von Comicdateien für den Lesemodus. *ComicLibAPI* enthält die Klassen zur Bereitstellung der ComicLib API. Dabei wird zwischen den API-Controllern (*APIControllers*) für die verschiedenen API-Versionen und den generischen Methoden und API-Ressourcen (*API*) geteilt. *ComicVineAPI* stellt die Klassen zur Nutzung der ComicVine API, unterteilt in Klassen zur Verwaltung der API-Zugriffe (*Management*), Verarbeitung der erhaltenen Datensätze (*Processing*) und Zugriff auf die einzelnen API-Ressourcen (*Resources*). *Configuration* enthält die Klassen zum Einlesen und Bereitstellen der

ComicLib Konfigurationsdatei. *Controllers* sammelt die Controller des MVC-Patterns der Web-Oberfläche. In *Database* sind die Klassen zur Kommunikation mit der MySQL Datenbank enthalten. Dabei wird geteilt zwischen den Klassen zur Bereitstellung der Konfiguration, Datenbankinitialisierung und -verbindung (*Management*) und den Klassen zur Bereitstellung der Datenbanktabellen und -views (*Resources*), welche die Models im MVC-Pattern darstellen. *Logging* stellt die Klassen zur Umsetzung des Logging von Nachrichten und Ereignissen in die Logdatei (*log.txt*). In *storage* sind die Klassen zur Verwaltung des Speichers enthalten. Dazu zählen die Verwaltung des Comicspeichers (und der Datenbanktabellen *Issues*, *Volumes* und *Publishers*) sowie die Bereitstellung von Comicdateien zum Download. u*pdater* enthält die Klassen zur Steuerung der Datenbankupdates. *views* sammelt die Views des MVC-Patterns der Web-Oberfläche.

resources dient als Stammverzeichnis für alle Nicht-PHP-Dateien von ComicLib: CSS- und Javascript-Bibliotheken sowie -Skripte, ComicLib-eigene (Hintergrund-)Bilder und HTML-Dateien.

Das Verzeichnis *storage* dient als Stammverzeichnis des Comicspeichers. Hierin wird die Comicsammlung abgelegt.

.htaccess sperrt den direkten Zugang zum Verzeichnis mit den Comicdateien, um unautorisierte Downloads zu verhindern. Außerdem leitet *.htaccess* alle Aufrufe, die nicht auf eine tatsächlich existierende Datei gerichtet sind, auf *index.php* um.

ComicLib verwendet für die Bereitstellung der Weboberfläche das Model-View-Controller-Pattern (MVC-Pattern). Das bedeutet, dass die Verwaltung der Daten (Model), die Anzeige der Daten (View) und die Aufbereitung von Daten für die Anzeige sowie Speicherung als Vermittlung zwischen diesen Beiden (Controller) getrennt voneinander stattfinden.[35] Um dies besser umsetzen zu können, werden alle Webseitenaufrufe auf *index.php* umgeleitet, wo dann der in der URL angeforderte Controller geladen wird, welcher dann mittels der View die Webseite generiert.

Die Datei *log.txt* ist die zentrale Logdatei für ComicLib. Hier laufen alle Ereignismeldungen und sonstigen Nachrichten für den Administrator der Installation zusammen. Wenn ein Fehler oder ungewöhnliches Verhalten auftritt, ist dies die erste Anlaufstelle.

Die Datei *updater.lock* dient als Sperrdatei für den Updateprozess. Sie stellt sicher, dass immer nur ein Updateprozess gleichzeitig läuft. Ist die Datei gesperrt, ist ein Update im Gange und der Nutzer kann auf der Web-Oberfläche über den Status des Updates informiert werden.

3.3.2 Anbindung an die ComicVine API

ComicLib greift zur Beschaffung der Informationen zu Comics, Serien und Verlagen auf die ComicVine API von Gamespot zu. Die ComicVine API ist eine REST-konforme Web-API. Der Zugriff auf die API erfolgt über das HTTP-Protokoll auf Port 80, ein Testen von API-URLs ist also auch über die Adresszeile eines Internet-Browsers möglich. Bei der Entwicklung von ComicLib kommt stattdessen Postman zum Einsatz, da es viele Zusatzfunktionen abseits des bloßen Durchführens von GET-Anfragen bietet und Anfragen sowie deren Antworten speichern und in Ordnerstrukturen organisieren kann.

Die ComicVine API bietet als Antwortformate XML (Extensible Markup Language) und JSON (JavaScript Object Notation) an. Für ComicLib werden nur Antworten im JSON-Format genutzt, da der geringere Formatoverhead zu geringeren Datenmengen führt und so im Vergleich zu XML die Übertragung großer Datensätze beschleunigt. Dank der Filter-Funktion der ComicVine API werden durch ComicLib

[35] Vgl. (Reimers, 2017)

nur die jeweils benötigten Felder eines Datensatzes angefordert, wodurch die zu übertragende Datenmenge weiter reduziert wird. Die genutzten Felder lassen sich 3.2.1 ComicVine API entnehmen.

Der Zugriff auf die API ist durch eine Token-Authentifizierung beschränkt. Zur Nutzung muss erst ein Benutzerkonto unter https://comicvine.gamespot.com/api/ angelegt werden. Danach wird dort der persönliche Zugangsschlüssel in Form einer 40-stelligen, hexadezimalen Zeichenkette, angezeigt. Diese muss an jede API-Anfrage in Form eines GET-Parameters angehangen werden. Im Falle eines fehlenden oder ungültigen Schlüssels antwortet die API mit einer Fehlermeldung und einem HTTP-Code 401 („Unauthorized“). Zur Nutzung von ComicVine muss also nach der Installation ein gültiger API-Schlüssel hinterlegt werden, was ein ComicVine Benutzerkonto voraussetzt.

Für die API-Zugriffe besteht eine Zugriffsratenbegrenzung, um eine Überlastung der Server zu vermeiden. Zwischen zwei API-Zugriffen muss immer mindestens eine Sekunde verstreichen. Eine Nichteinhaltung führt nicht sofort zur Sperrung des Zugriffs, allerdings werden die Antworten verzögert. Eine dauerhafte Nichteinhaltung führt zu einer temporären Sperre, während derer keine Datensätze mehr bezogen werden können.[36] Daher wartet ComicLib nach dem Erhalten einer Antwort von der API immer zwei Sekunde, bevor eine neue Anfrage gesendet werden kann. Die Wartezeit von zwei Sekunden ist bewusst großzügig gewählt, da es mit nur einer Sekunde Wartezeit zwar noch keine Sperre, aber eine Warnmeldung in der Zugriffsübersicht des ComicVine Benutzerkontos gab.

Eine Ausnahme bezüglich der Authentifizierung sowie der Zugriffsratenbegrenzung stellt die *image*-Ressource dar. Diese stellt die in den Datensätzen der anderen Ressourcen verlinkten Bilddateien bereit. Da hier keine Zugriffsratenlimitierung vorliegt, bezieht ComicLib die Bilder ohne künstliche Pause zwischen den API-Zugriffen. Die Authentifizierung entfällt bei dieser Ressource ebenfalls.

Die ComicVine API stellt eine Suchfunktion in Form einer eigenen Ressource zur Verfügung. Die Suchfunktion ist im Vergleich zum Zugriff auf die anderen Ressourcen deutlich langsamer. Da es häufig Serien mit ähnlichen oder gleichen Seriennamen gibt (zum Beispiel durch Serien-Neustarts), sind die Ergebnisse für eine Suche nach einer Serie häufig nicht eindeutig. Die Einbeziehung des Startjahrs der Serie kann dieses Problem deutlich entschärfen, aber auch hier kann es noch zu Fehlzuordnungen kommen. Daher wurde für ComicLib ein anderer Ansatz gewählt. Mehr dazu im nächsten Kapitel, 3.3.3 Verwaltung der Comics.

3.3.3 Verwaltung der Comics

Die Speicherung der Comics erfolgt bei ComicLib im Verzeichnis *storage*. Pro Serie darf es nur genau ein Verzeichnis geben. Zur Zuordnung des Verzeichnisses zu einer Serie muss es eine Datei *volume.ini* enthalten, in die die ID des *Volumes* aus der ComicVine Datenbank eingetragen werden muss. Der Inhalt der Datei besteht dabei aus einer ersten Zeile mit dem Abschnittsnamen „[Volume]“ und einer zweiten Zeile mit der ID in der Form „ID=123“. Da die *Volume-IDs* aus einem Präfix „4050-“ und der eigentlichen ID bestehen, wird hier das Präfix ausgespart und erst beim Zugriff auf die API durch ComicLib ergänzt. Die *Volume-ID* muss der Besitzer/Verwalter der Sammlung also für jede Serie einmalig im ComicVine-Wiki nachsehen. Dadurch ist eine präzise Zuordnung gewährleistet und die Anzahl der API-Zugriffe auf ein Minimum reduziert, wodurch der Zeitbedarf für die Aktualisierung der Sammlung minimiert wird.

Damit die Comicdateien den einzelnen Heften aus den Datensätzen der ComicVine API zugeordnet werden können, müssen deren Dateinamen mit der Heftnummer im Format „#123“ enden („123“ dient hier nur als Beispielheftnummer). Fehlt dieser Teil des Dateinamens, kann das Heft nicht in die

[36] Vgl. (CBS Interactive Inc., API Rate Limiting, 2015)

Datenbank aufgenommen werden, ebenso wenn die Heftnummer zu keinem Heft der Serie aus der ComicVine API passt.

Die Aktualisierung der Datenbank läuft in mehreren Phasen ab. Zunächst werden drei Listen gebildet: Serien, die in der Sammlung, aber nicht in der Datenbank enthalten sind; Serien, die in beiden enthalten sind; und Serien, die in der Datenbank, aber nicht in der Sammlung enthalten sind.

Die Serien der ersten Liste werden zur Datenbank hinzugefügt, indem die Comicdateien im Serienverzeichnis mit dem *Volume*-Datensatz aus der ComicVine API abgeglichen und im Falle einer Übereinstimmung zur Datenbank hinzugefügt werden. Die Zuordnung des Serienverzeichnisses zum *Volume*-Datensatz erfolgt dabei mit Hilfe der *Volume-ID* aus der *volume.ini*. Bevor die Serie und dann die Hefte in die Datenbank geschrieben werden, muss der Verlag hinzugefügt werden, falls er noch nicht in der Datenbank enthalten ist. Nach dem Hinzufügen von Verlag, Serie und Heften wird das jeweilige Bild des Datensatzes heruntergeladen und in *cache* zwischengespeichert.

Die Serien der zweiten Liste sind bereits, zumindest teilweise, in der Datenbank enthalten. Falls das Serienverzeichnis umbenannt wurde, wird der Verzeichnisname in der Datenbank aktualisiert. Für jede Comicdatei muss nun überprüft werden, ob sie bereits in der Datenbank liegt. Falls eine Comicdatei umbenannt wurde, wird der Dateiname im Datensatz aktualisiert – vorausgesetzt, die Heftnummer ist weiterhin zuzuordnen. Ist die Comicdatei eines Hefts aus der Datenbank nicht mehr auffindbar, wird dieses aus der Datenbank entfernt. Sollte dadurch die Serie Heft-los werden, wird sie ebenfalls gelöscht. Sollte durch das Löschen der Serie der Verlag Serien-los werden, wird auch dieser gelöscht. Für jeden gelöschten Datensatz wird das zwischengespeicherte Bild ebenfalls gelöscht. Für jedes neue Heft wird nach dem Hinzufügen zur Datenbank das Coverbild heruntergeladen und zwischengespeichert.

Die Serien der dritten Liste sind in der Datenbank enthalten, aber in der Sammlung nicht mehr auffindbar. Da die Zuordnung über die *Volume-ID* erfolgt, ist die *volume.ini* möglicherweise beschädigt oder gelöscht worden, oder das Verzeichnis ist im Ganzen nicht mehr vorhanden. Daher werden erst die Hefte der Serie und dann die Serie aus der Datenbank gelöscht. Sollte dadurch der Verlag keine Serien mehr haben, wird auch dieser gelöscht. Für alle gelöschten Datensätze wird auch das zwischengespeicherte Bild gelöscht.

Zum Abschluss der Aktualisierung wird noch einmal für jeden Datensatz in *Issues*, *Publishers* und *Volumes* geprüft, ob die jeweilige Bilddatei im Bildercache enthalten ist. Falls das nicht der Fall ist, wird die Datei in den Zwischenspeicher heruntergeladen. Dadurch wird sichergestellt, dass verloren gegangene Bilder wiederhergestellt werden können. Sollte ein Bild beschädigt sein, kann es durch den Benutzer gelöscht und über die Aktualisierung erneut heruntergeladen werden.

3.4 Front-End

Das folgende Kapitel widmet sich dem Front-End von ComicLib. Da ComicLib eine Webanwendung ist, ist die Trennung zwischen Front-End (Benutzeroberfläche) und Back-End (Anwendungslogik) sehr starr. Im folgenden Kapitel geht es um die verwendeten Bibliotheken, die Sichten der Benutzeroberfläche, den integrierten Lesemodus, die ComicLib Programmierschnittstelle und die Authentifizierung zur Nutzung von Benutzeroberfläche und API.

3.4.1 Verwendete Bibliotheken

Zur Bereitstellung von Design-Vorlagen und -Elementen kommen in ComicLib mehrere Bibliotheken zum Einsatz. Zur Verbesserung der Verständlichkeit der Benutzeroberfläche werden Icons genutzt, die

durch Font Awesome[37] 5 zur Verfügung gestellt werden. Diese sind dank mitgelieferter CSS- und Javascript-Klassen leicht in der Größe, Farbe und Orientierung anpassbar und animierbar. Die Designbausteine für die Weboberfläche stammen größtenteils aus Bootstrap[38] 4. Dank der enthaltenen Designs und Funktionen passen sich die Designelemente gut an Bildschirme unterschiedlicher Größen, Formfaktoren und Rotationen an. Auch die ausklappbaren Menüs und die Seitendarstellung des Lesemodus profitieren sehr von der Vorarbeit in Bootstrap. Als Abhängigkeiten benötigt Bootstrap zwei weitere Bibliotheken. Popper.js[39] wird von Bootstrap zur Anzeige von Pop-ups und einblendbaren Hilfetexten verwendet. Da einige der Javascript-Funktionen in Bootstrap jQuery[40] verwenden, muss diese Bibliothek ebenfalls von ComicLib mitgebracht werden. jQuery kommt auch in ComicLibs eigenem Javascript-Quelltext vor: die Ansteuerung des Bootstrap *Carousel*-Elements, das zur Anzeige der Comicseiten im Lesemodus verwendet wird, setzt jQuery-Code voraus.[41] Abseits davon wird reines Javascript verwendet.

3.4.2 Sichten

Die Weboberfläche von ComicLib basiert auf dem Model-View-Controller-Pattern (MVC-Pattern). Das bedeutet, dass streng zwischen Anwendungslogik (Controller), Darstellung (View) und Datenmodell (Model) getrennt wird.[42] Der Model-Teil wird durch die Klassen zur Datenbankanbindung realisiert. Für jede Tabelle und jede View gibt es eine Klasse, die die Operationen auf dieser Tabelle oder View implementiert. Der Controller-Teil wird über die Klassen im *Controllers*-Ordner realisiert. Für jede Seite der Weboberfläche gibt es einen Controller, der die für die Seite nötigen Daten aufbereitet und im Anschluss die zur Seite gehörige View mit den aufbereiteten Daten befüllt und an den anfordernden Internetbrowser sendet. So gibt es für die Serienübersicht unter dem Pfad „/volumes“ einen *VolumesController*, der die Seriendatensätze aus der Datenbank holt, aufbereitet und dann die *VolumesView* mit den Daten bestückt als Antwort sendet. Der Controller dient aber auch als Rückkanal für Änderungen aus der View. Wird zum Beispiel an einer der Serien in der *VolumesView* der Lesestatus geändert, wird ein Formular an den Server gesendet, dass durch den *VolumesController* verarbeitet wird. Der neue Lesestatus wird dann vom *VolumesController* über die entsprechende Model-Klasse gespeichert.

Die Weboberfläche besteht im Wesentlichen aus 10 unterschiedlichen Sichten, die jeweils durch einen Controller und eine View implementiert werden.

Die Loginsicht (*/login*) besteht aus einem einfachen Login-Formular mit Feldern für Benutzername und Passwort. Unter dem Formular befindet sich der Senden-Button. Falls beim Anmelden ein Fehler auftritt klappt außerdem unter dem Passwortfeld eine Fehlermeldung auf. Siehe Abbildung 3: Login-Sicht.

Die Serienübersicht (*/volumes*) stellt alle Serien der Comicsammlung in einer Albumansicht dar. Dabei wird jede Serie durch ihr Serientitelbild und ihren Namen repräsentiert. Über einen Button kann die Sicht mit den Seriendetails samt der Serienhefte aufgerufen werden. Ein Dropdown-Menü daneben ermöglicht das Markieren der Serie als (un-)gelesen und den Zugriff auf die Verlagssicht des Verlags der Serie. Unter den Buttons wird in gedämpfter Schrift die Anzahl der Hefte der Serie angezeigt. In der oberen, rechten Ecke wird ein „New“-Schriftzug angezeigt, falls die Serie ungelesene Hefte enthält.

---

[37] https://fontawesome.com/
[38] https://getbootstrap.com/
[39] https://popper.js.org/
[40] https://jquery.com/
[41] Vgl. (getbootstrap.com, 2019)
[42] Vgl. (Reimers, 2017)

Die Serienübersicht zeigt nicht alle Serien auf einmal an – bei größeren Sammlungen würde das Laden der Seite sonst zu lange dauern. Stattdessen werden in alphabetischer Reihenfolge der Seriennamen immer 24 Serien pro Seite angezeigt und über ein Menüband am oberen und unteren Ende der Albumansicht kann zwischen den Seiten navigiert werden. Da diese Sicht auf den unterschiedlichen Displaygrößen mit zwei, drei oder sechs Serien nebeneinander angezeigt wird, empfiehlt es sich, die Seitengröße als Vielfaches von Sechs zu wählen. Die Festlegung auf 24 Elemente pro Seite stellt einen Kompromiss zwischen guter Nutzbarkeit auf kleinen Displays und guter Platzausnutzung auf großen Displays dar. Die Serienübersicht dient in ComicLib als Startseite. Siehe Abbildung 4: Serienübersicht.

Die Seriendetailsicht (*/volume/volumeid*) enthält im oberen Bereich der Seite den Titel der Serie als Überschrift, gefolgt von einem erweiterbaren Textbereich mit der Beschreibung der Serie. Darunter folgt, ähnlich der Serienübersicht, die Übersicht der Comichefte der Serie als Albumansicht. Im Gegensatz zur Serienübersicht werden hier alle Elemente auf einmal angezeigt, da die Anzahl der Comichefte pro Serie in der Regel gering ist. Die Comichefte werden mit Coverbild und Namen präsentiert. Darunter folgt ein Button zum Öffnen des Comics im Lesemodus. Wenn die Comicdatei des Hefts nicht in einem der unterstützten, freien Formate (CBR, CBZ, PDF, RAR, ZIP) vorliegt, ist dieser Button ausgegraut. Neben dem Button befindet sich ein Dropdown-Menü, das das Markieren des Hefts als (un-)gelesen und das Herunterladen der Comicdatei ermöglicht. Siehe Abbildung 5: Seriendetailsicht.

Die Verlagsübersicht (*/publishers*) ist ähnlich wie die Serienübersicht gestaltet. Jedoch kommt hier keine Einteilung der Elemente in Seiten zum Einsatz, da diese Liste in der Regel recht kurz ist. Unter dem Bereich mit dem Verlagslogo und -namen wird ein Button angezeigt, der zur Verlagsdetailsicht mit den Verlagsserien führt. Darunter wird in gedämpfter Schrift die Anzahl der Serien des Verlags in der Sammlung angezeigt. Siehe Abbildung 6: Verlagsübersicht.

Die Verlagsdetailsicht (*/publisher/publisherid*) enthält im oberen Bereich neben einander das Verlagslogo und den Verlagsnamen. Darunter folgt, wie in der Seriendetailsicht, ein erweiterbarer Textbereich mit dem Beschreibungstext des Verlags. Darunter folgt die aus der Serienübersicht bekannte Albumansicht der Serien, hier allerdings beschränkt auf die Comicserien eines einzelnen Verlags. Siehe Abbildung 7: Verlagsdetailsicht.

Die Leselistensicht (*/readinglist*) enthält die Liste der Comichefte, die ungelesen sind, deren aktuelle Seitenzahl aber nicht 0 ist. Diese werden als Album entsprechend dem Layout der Hefte aus der Seriendetailsicht angezeigt. Über der Albumansicht erklärt eine Überschrift samt Untertitel, wozu die Leseliste dient. Siehe Abbildung 8: Leselistensicht.

Die Aktualisierungssicht (*/update*) startet im Hintergrund die Datenbankaktualisierung, falls diese nicht schon läuft. Danach wird auf die Aktualisierungsstatussicht umgeleitet (*/updates*), wo der Status des aktuellen Updatevorgangs alle drei Sekunden über AJAX (Asynchronous JavaScript and XML) im Hintergrund aktualisiert wird. Der Updatestatus wird dabei als Text in der Bildschirmmitte angezeigt. Links daneben wird bei laufender Datenbankaktualisierung ein sich drehendes Aktualisierungssymbol angezeigt. Ist die Aktualisierung abgeschlossen, wird stattdessen ein Hakensymbol angezeigt. Die Seite wird auch nach dem Ende der Aktualisierung weiter mit dem aktuellen Status versorgt. Wird in einem anderen Tab oder Browser die Aktualisierung der Datenbank gestartet, wird der neue Status auch hier angezeigt. Siehe Abbildung 9: Aktualisierung im Gange und Abbildung 10: Aktualisierung abgeschlossen.

Die Suchergebnissicht (*/search*) zeigt die zu einem Suchtext passenden Serien an. Gesucht werden kann nur anhand des Serientitels. Die Suchergebnissicht verwendet dasselbe Albumlayout wie die

Serienübersicht. Die Suche ist der einzige Bereich der Weboberfläche, in dem statt der POST-Methode die GET-Methode zur Übertragung von Daten an den Server genutzt wird. Dadurch bleibt in der URL ersichtlich, nach welchem Suchbegriff gesucht wurde und ein Neuladen der Seite nach dem Ändern des Lesestatus einer der Serien stellt kein Problem dar. Im Falle von POST wäre beides nicht möglich. Siehe Abbildung 11: Suchergebnissicht.

Die Seite-Nicht-Gefunden-Sicht ist eine an das Design von ComicLib angepasste Variante einer klassischen Fehlerseite. In der Seitenmitte wird groß der Fehlertext angezeigt. Daneben steht ein Suchsymbol in Form einer Lupe. Zusätzlich zum eigentlichen Inhalt wird bei dieser Sicht der HTTP-Statuscode 404 („Not Found“) im Header mitgesendet. Diese Sicht wird immer dann angezeigt, wenn ein angeforderter Controller nicht existiert (also die Seite nicht existiert) oder für die Sicht keine Datensätze gefunden wurden (die Sicht also leer wäre, z.B. weil ein Verlag in der Datenbank nicht gefunden wurde). Siehe Abbildung 12: Seite-nicht-gefunden-Sicht.

Das Herzstück von ComicLib ist sicherlich die Lesesicht (*/issue/issueid*). Daher wird dieser und den Mechanismen dahinter ein eigenes Kapitel gewidmet. Dieses findet sich unter 3.4.3 Lesemodus.

3.4.3 Lesemodus

Der Lesemodus ist das Herzstück von ComicLib. Der Wunsch, eine freie Software als Alternative zu Amazon Kindle und Comixology für Comicsammlungen auf Basis von Comics in freien Dateiformaten zu schaffen, treibt das ganze Projekt an. Mit dem Lesemodus werden die Comics in freien Dateiformaten zu Bürgern erster Klasse in ComicLib. Die Verwaltung von Comics ist in ComicLib so ausgelegt, dass Comics beliebiger Formate genutzt werden können, aber das Lesen von Comics direkt in der Weboberfläche bleibt den freien Formaten vorbehalten. Für alle anderen Formate bleibt nur die Nutzung des Downloadbuttons und das Öffnen in einer Drittanbietersoftware.

Wird die Leseansicht aufgerufen, erscheint, falls die Comicdatei noch nicht im Cache liegt, erst eine Ladesicht (siehe Abbildung 13: Ladesicht), auf die dann die eigentliche Leseansicht (siehe Abbildung 14: Leseansicht) folgt, sobald die Datei in den Cache entpackt wurde. Das Entpacken von PDF-Dateien dauert besonders lange - einige Minuten bei PDF, hingegen nur ein paar Sekunden bei CBR/CBZ/etc. Daher wird in der Ladesicht ein zusätzlicher Hinweis auf die lange Ladedauer angezeigt, wenn die Datei im PDF-Format vorliegt. Die Leseansicht enthält in der Mitte den Bereich zur Anzeige der einzelnen Comicseiten. Als ausblendbares Overlay werden an den Rändern die Steuerelemente eingeblendet. Rechts und links gibt es große Buttons zum Vor- und Zurückblättern. Unten befindet sich ein Schieberegler, mit dem schnell an eine beliebige Stelle im Comic gesprungen werden kann. Oben befindet sich ein kleiner Button mit einem „Vollbild“-Symbol, der den Internetbrowser in den Vollbildmodus befördert. Besonders auf mobilen Endgeräten ist das sehr praktisch, da diese in vertikaler Rotation annähernd das Seitenverhältnis eines Comics aufweisen. Die Vor- und Zurück-Buttons werden beim Ausblenden der Steuerelemente nur unsichtbar, sind aber weiterhin benutzbar. Dadurch bietet ComicLib eine Touchnavigation durch Tippen auf die Seitenränder, wie sie von vielen mobilen Dokumentenbetrachtern und E-Book-Readern bekannt ist. Ein Vor- und Zurückblättern per Wischgeste ist auf Geräten mit Touchscreen ebenfalls möglich. Dazu kommt jedoch nicht die Touchunterstützung des, für die Anzeige und Seitennavigation verwendeten, Bootstrap-Carousel zum Einsatz, sondern eine eigens entwickelte Touchsteuerung, da die Bootstrap-eigene Steuerung sporadisch die falsche Seitenzahl ansteuerte.

Bei jedem Seitenwechsel wird im Hintergrund per AJAX die aktuelle Seitenzahl an den Server gesendet. Dieser aktualisiert dann in der Datenbank die Seitenzahl, so dass der Comic jederzeit an dieser Stelle weitergelesen werden kann. Wird die letzte Seite des Comics erreicht, wird durch den Server der Lesestatus auf „gelesen“ gesetzt und die Seitenzahl auf 0 zurückgesetzt.

Bei der Anzeige der Comicseiten kommt Lazy Loading zum Einsatz. Da die Comics häufig mehrere Dutzend bis Hunderte Megabyte groß sind, kommt ein Laden aller Bilder beim Laden der Leseansicht nicht in Frage. Der Seitenaufbau würde bei einer langsamen Verbindung sehr lange dauern und die Seite würde unnötig viel Platz im Speicher belegen. Deshalb werden die Bilder bei Bedarf nachgeladen.[43] Die Links der Bilder der Comicseiten werden nicht als Source-Attribut in den HTML-Elementen hinterlegt, sondern als Data-Source-Attribut. Dadurch werden die Bilder nicht automatisch geladen. Nur die aktuelle Comicseite hat beim Laden der Ansicht ein Source-Attribut, damit deren Bild beim Laden der Seite mitgeladen wird. Wird nun zu einer neuen Comicseite gewechselt, wird die URL des Bilds der neuen Seite per Javascript in das Source-Attribut kopiert, wodurch es durch den Internetbrowser geladen wird. Damit beim Übergang zur nächsten oder vorherigen Seite keine Ladezeiten auftreten, werden für diese ebenfalls die Bilder nachgeladen. Wird der Comic normal linear gelesen, treten im Allgemeinen keine spürbaren Ladezeiten auf. Wird im Comic per Schieberegler gesprungen, können Ladezeiten auftreten, falls die angesteuerte Seite nicht bereits zuvor geladen wurde.

Damit die Comics in der Leseansicht angezeigt werden können, müssen sie zuvor entpackt werden. Für Dateien in den Formaten CBR, CBZ, RAR und ZIP gestaltet sich das Entpacken geradezu trivial. Da es sich um normale Archivdateien handelt, reicht ein einfaches Entpacken in den Comics-Cache. Das Hinzufügen von Comics im PDF-Format zum Cache gestaltet sich deutlich aufwändiger. Hier muss die PDF-Datei erst als Ganzes in ImageMagick eingelesen werden. Dieser Vorgang kann sich über einige Minuten hinziehen. Danach generiert ImageMagick aus jeder Seite ein Bild und speichert es im Cache. Das Caching von PDF-Dateien ist sehr speicheraufwändig. Die standardmäßige Speicherbegrenzung von ImageMagick auf 256MB RAM war im Test zu klein, um Comicdateien mit mehr als 30 Seiten zu konvertieren. ImageMagick exportierte dabei immer nur knapp über 20 Seiten. Eine Erhöhung der Speicherbegrenzung auf 512MB genügte im Test, um auch den größten zum Testen verfügbaren Comic mit mehr als 600 Seiten zu konvertieren. Die Konvertierung dauerte allerdings eine halbe Stunde auf einem Rechner mit Intel Core i5 CPU. Insgesamt kann von der Verwendung von PDF zur Speicherung von Comics nur abgeraten werden. Das Format hat gegenüber den auf Archivdateien basierenden Formaten keine Vorteile, bringt dafür aber große Performanceprobleme mit sich.

3.4.4 ComicLib API

Um die zukünftige Entwicklung einer mobilen Lese-App für ComicLib zu ermöglichen, enthält ComicLib eine Web-API, die sich am REST-Paradigma orientiert. Diese stellt die Daten der Datenbanktabellen *Issues*, *Publishers* und *Volumes* zum Lesen zur Verfügung. Der Lesestatus für Serien und Hefte kann über Subressourcen der *Issues*- und *Volumes*-Ressourcen gelesen und geändert werden. Die Hefte einer Serie sind als Subressource der jeweiligen Serie verfügbar. Die Comicdateien können ebenfalls über eine Subressource des jeweiligen Hefts direkt per API heruntergeladen werden. Für die Bilder wird keine eigene API-Ressource verwendet, stattdessen wird direkt auf die gecacheten Bilddateien, wie sie auch in der Weboberfläche genutzt werden, verlinkt.

Die API-Ressourcen wurden zusammen mit den auf ihnen erlaubten HTTP-Verben und der jeweils benötigten Authentifizierungsart mit Hilfe von Postman dokumentiert[44]. Für jede Ressource wurden API-Zugriffe mit den HTTP-Methoden GET, POST, PUT und DELETE dokumentiert. Die GET-Methode ist auf allen Ressourcen erlaubt und gewährt lesenden Zugriff auf die Datenbanktabellen. Die PUT-Methode ist nur für den Lesestatus von Serien und Heften erlaubt, da alle anderen Tabellen nicht durch den Benutzer, sondern nur durch die Comicsammlungsaktualisierung geändert werden dürfen.

[43] Vgl. (Wagner, 2019)
[44] Vgl. (Hahn, 2019)

Die POST- sowie DELETE-Methode sind auf allen Ressourcen verboten und nur dokumentiert, da sie Teil der obligatorischen Methoden einer jeden REST-API sind. Ihre Dokumentation soll vor allem Eindeutigkeit schaffen. Die Dokumentation umfasst für jede Methode jeder Ressource die API-URL eines Beispielzugriffs, einen Beispieldatensatz für den Fall des erfolgreichen API-Zugriffs (falls erlaubt) und die möglichen Fehlermeldungen im Falle des Scheitern des API-Zugriffs.

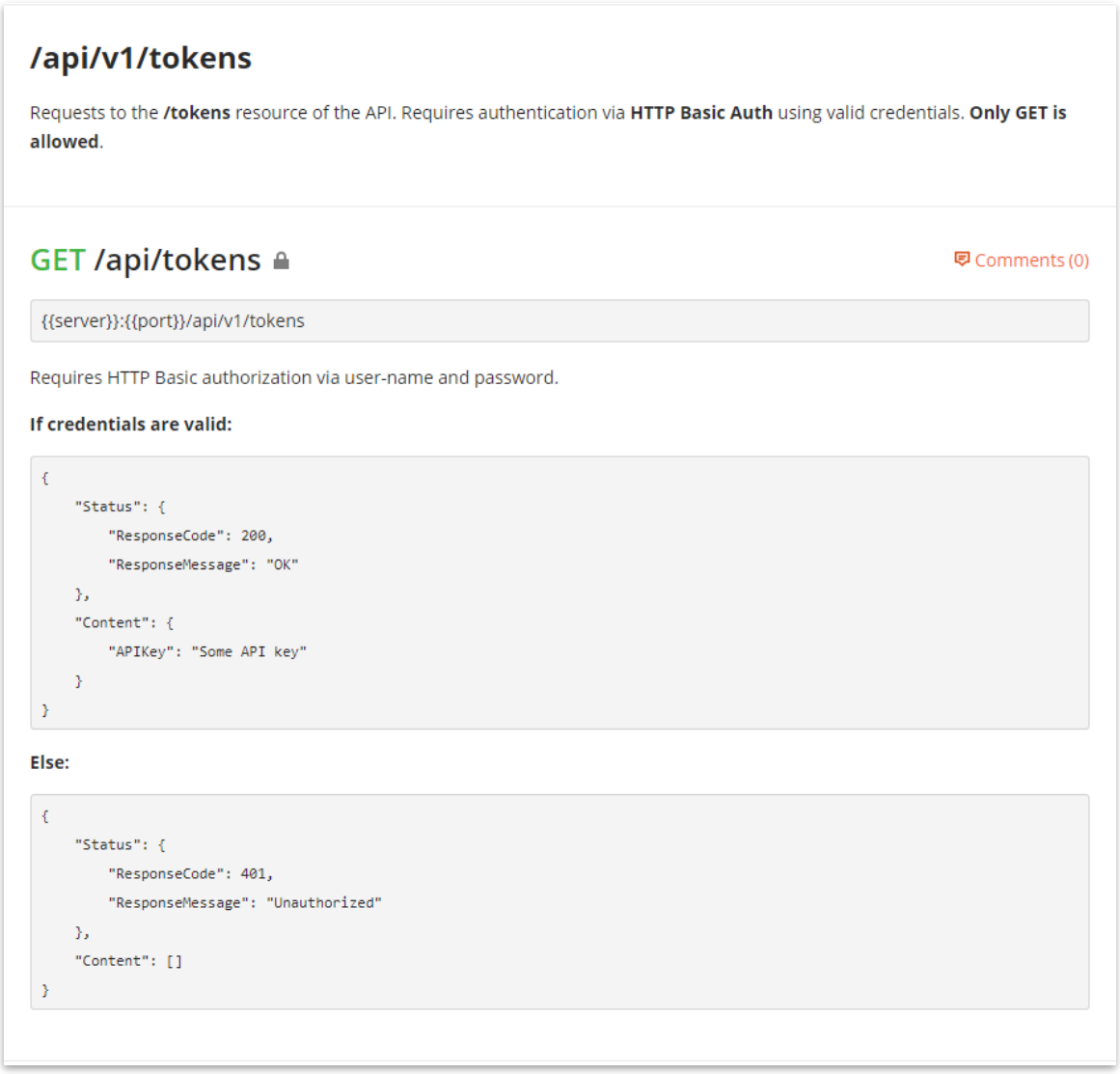

Abbildung 2: Ausschnitt aus der API-Dokumentation.
In der URL sind Platzhalter für Serveradresse und Port angegeben, dahinter folgt der Pfad innerhalb der API. Die in den Antwortdatensätzen angegebenen Fehlercodes werden ebenfalls als HTTP-Statuscode im Header gesendet. Das Feld „ResponseMessage" enthält den Namen des HTTP-Statuscodes. Das Feld „Content" enthält die Datensätze der Antwort des API-Servers.

Die ComicLib API orientiert sich am *Representational State Transfer*-Paradigma. Das REST-Paradigma ist ein häufig genutztes Paradigma zur Gestaltung von Programmierschnittstellen. REST definiert fünf obligatorische Anforderungen an eine API[45]:

[45] Vgl. (Fielding, 2000)

Client-Server-Architektur: Der API-Server stellt einen Server im Sinne einer Client-Server-Architektur dar. Er stellt einen Dienst zur Nutzung durch Clients bereit. Da ComicLib im Allgemeinen als Server-Software ausgelegt ist, ist die Anforderung erfüllt.

Zustandslosigkeit: Anfragen an und Antworten durch den Server sind in sich geschlossen und voneinander unabhängig. Weder der Server noch der Client muss zwischen zwei Anfragen zusätzliche Informationen über den Zustand speichern, um die zweite Anfrage interpretieren zu können. Diese Anforderung ist in ComicLib erfüllt.

Caching: Eine REST-API soll HTTP Caching nutzen. Dadurch wird es ermöglicht, Datensatz nur dann zu aktualisieren, wenn sie seit der letzten Abfrage auch wirklich geändert wurden. Diese Anforderung ist in ComicLib – zumindest zurzeit – nicht erfüllt.

Mehrschichtige Systeme: Eine REST-API soll nur eine Schnittstelle anbieten, hinter der die eigentliche Architektur verborgen bleibt. Das bedeutet, dass zwischen dem eigentlichen API-Server und dem Nutzer auch ein Proxy oder ein Loadbalancer geschaltet werden kann, ohne die Funktionstüchtigkeit und Erreichbarkeit der API einzuschränken. Grundsätzlich ist der Betrieb der ComicLib API hinter einem Proxy möglich, auf Grund der zu erwartenden geringen Größe von ComicLib-Installationen ist ein Loadbalancing aber tendenziell nicht nötig. Diese Anforderung ist in ComicLib also erfüllt.

Einheitliche Schnittstelle: Diese Anforderung besteht aus vier Teilanforderungen, die dafür sorgen sollen, dass die API einheitlich gestaltet und leicht nutzbar ist:

> Adressierbarkeit von Ressourcen: Jede über die API zugängliche Information hat eine eindeutige Adresse. Diese setzt sich aus der Adresse der API und dem Pfad zur Ressource mit der Information zusammen. Eine gültige Adresse einer Ressource in ComicLib ist zum Beispiel „127.0.0.1:8081/api/v1/issues/*{id}*“ (*{id}* dient als Platzhalter für eine gültige *IssueID*). Diese Anforderung ist in ComicLib erfüllt.
>
> Veränderung von Ressourcen durch Repräsentationen: Besitzt ein Client die Repräsentation einer Ressource einschließlich aller dazugehörigen Metadaten, besitzt er alle Informationen, die zum Ändern oder Löschen dieser Ressource notwendig sind. Bei der Repräsentation einer Ressource handelt es sich um einen Datensatz der API, der in einem durch die API nutzbaren Format (Repräsentation) vorliegt. Die innere Repräsentation der API (z.B. Datenbanktabellen, PHP-Objekte) und die äußere Repräsentation (z.B. JSON, XML, HTML) sind streng voneinander getrennt. Die meisten APIs bieten mehrere Repräsentationsformen an. Die ComicLib API bietet nur JSON als Format an, da der Speicherbedarf von XML im Vergleich höher ist. Ein Beispiel für die Veränderung einer Ressource durch eine Repräsentation stellt in ComicLib das Aktualisieren eines Lesestatus dar. Zum Aktualisieren wird der aktuelle Lesestatus von der API geholt, die Werte der Felder im *Content*-Teil der im JSON-Format erhaltenen Antwort aktualisiert und als JSON-Objekt in einem PUT-Request auf die Adresse des Datensatzes an die API gesendet. Diese aktualisiert den Datensatz in der Datenbank und sendet den aktualisierten Datensatz als Antwort. Diese Anforderung ist in ComicLib also erfüllt.
>
> Selbstbeschreibende Nachrichten: Anfragen an und Antworten von einer REST-API sollen selbstbeschreibend sein. Die wichtigsten Mittel zur Erreichung der Selbstbeschreibung sind die Verwendung von HTTP-Verben zur Angabe der Verarbeitungsart, die Nutzung des Medientyp-Headers zur Angabe des zu erwartenden Nachrichtenformats des Request-Körpers und die Verwendung von HTTP-Statuscodes. Die ComicLib API nutzt die vier obligatorischen HTTP-Verben GET, POST, PUT und DELETE zur Angabe der Verarbeitungsart. Wirklich genutzt werden allerdings nur GET und PUT, das Erstellen und Löschen von Datensätzen soll über diese

API nicht möglich sein. Da die API nur mit JSON arbeitet, entfällt die Spezifikation des Formats via Medientyp-Header. ComicLib nutzt zur Übermittlung des Status einer Anfrage HTTP-Statuscodes. Ist beispielsweise eine Methode (zum Beispiel DELETE) nicht erlaubt, wird im Header der Antwort der HTTP-Statuscode *405 Method Not Allowed* gesetzt. Außerdem werden der Statuscode und der Statustext im Körper der Antwort eingetragen. Für die Repräsentationen von Ressourcen ist in der Antwort ein eigenes Feld *Content* angelegt, das im Falle eines Fehlers leer bleibt, im Falle einer erfolgreichen Anfrage hingegen alle resultierenden Datensätze als JSON-Liste enthält. Verarbeitungsmethode, Anfragestatus und Inhalt der Antwort sind also wohldefiniert und die Nachrichten beschreiben sich selbst. Diese Anforderung ist also in ComicLib erfüllt.

„Hypermedia as the Engine of Application State“ (HATEOAS): Diese Eigenschaft ermöglicht es, in der API allein durch die in den Ressourcen enthaltenen URLs zu navigieren. In ComicLib enthält jeder Verweis auf eine andere Ressource immer auch die URL zu dieser Ressource. Die Repräsentation eines Datensatzes aus *Volumes* enthält so, zum Beispiel, neben der *PublisherID* des die Serie herausgebenden Verlags, immer auch die URL auf die Ressource dieses Verlags.

Eine weitere, wichtige Eigenschaft, die aber keine der Anforderungen von REST ist, stellt die Verwendung einer Versionierung der API dar. ComicLib definiert den API-Pfad als „servername:port/api/{version}/{resource}“, wobei „{version}“ den Bezeichner der API-Version und „{resource}“ den weiteren Pfad zur gewünschten Ressource vertritt. Die ComicLib API liegt zurzeit in Version 1 vor, ein gültiger API-Pfad wäre daher „servername:port/api/v1/{resource}“. Die Versionierung einer REST API ist wichtig, damit Änderungen und Neuerungen implementiert werden können, ohne die Funktionalität von existierenden Anwendungen, die die API nutzen, zu beschädigen.[46]

Für die Absicherung der API gegen unautorisierten Zugriff kommen zwei unterschiedliche Authentifizierungsverfahren zum Einsatz. Die Ressource */tokens* ist durch HTTP Basic Authentication gesichert. Dazu müssen Benutzername und Passwort genutzt werden. Für alle anderen Ressourcen kommt eine Bearer Token Authentication zum Einsatz, für die ein Zugangstoken benötigt wird. Die Details zur API-Authentifizierung können dem Kapitel 3.4.5 Authentifizierung entnommen werden.

3.4.5 Authentifizierung

Zur Nutzung der Web-Oberfläche von ComicLib müssen Benutzer erst authentifiziert werden. Die Authentifizierung arbeitet Session-basiert.[47] Der Internetbrowser übermittelt zusammen mit jeder Anfrage die eindeutige Identifikationsnummer seiner Session. Anhand dieser ordnet PHP der Anfrage automatisch das zu dieser ID gehörende Session-Objekt zu. In diesem wird nun nachgesehen, ob für die Session bereits ein Benutzername hinterlegt ist. Ist das der Fall, hat sich der Benutzer bereits zuvor erfolgreich angemeldet. Daher wird nun überprüft, ob die Session abgelaufen ist. Dazu wird geprüft, ob der aktuelle Zeitstempel mehr als zwei Stunden hinter dem Zeitstempel im Feld *LastLogin* des Datenbankeintrags des Benutzers liegt. Ist das der Fall, wird die Session gelöscht und statt der angeforderten Sicht die Login-Sicht gesendet. Ist die Session noch nicht abgelaufen, wird die angeforderte Sicht generiert und gesendet.

Zum Anmelden in ComicLib müssen auf einer Login-Seite ein Benutzername und das dazugehörige Passwort eingegeben werden. Diese werden ans Back-End gesendet und dort verarbeitet. Zunächst wird versucht, das gehashte Passwort zum erhaltenen Benutzernamen aus der Datenbank zu holen.

[46] Vgl. (Spichale, 2016)
[47] Vgl. (Morris, 2017)

Sollte der Benutzer nicht existieren, wird erneut die Login-Sicht gesendet und eine zusätzliche Fehlermeldung angezeigt. Wenn der Benutzer gefunden wurde, wird nun das erhaltene Password per *password_verify*-Funktion mit dem gehashten Password aus der Datenbank verglichen. Stimmen die Passwörter nicht überein, wird wie beim falschen Benutzer die Loginsicht erneut angezeigt. Stimmen die Passwörter überein, werden der Name des Benutzers und dessen Benutzer-ID im Session-Objekt gespeichert und das Feld *LastLogin* des Datensatzes des Benutzers in der Datenbank mit dem aktuellen Zeitstempel überschrieben. Danach wird die angeforderte Sicht generiert und gesendet.

Die ComicLib API verwendet zur Authentifizierung zwei unterschiedliche Methoden. Beide nutzen zur Übertragung der Logindaten den Authorization-Header des HTTP-Protokolls. Für den Zugriff auf die Ressourcen der API wird im Allgemeinen die Bearer Token Authentication verwendet.[48] Dazu wird ein persönliches, einzigartiges Zugangstoken, in Base64 codiert, an den API-Server gesendet und durch diesen decodiert und in der Datenbank gesucht. Wird das Zugangstoken in der Datenbank gefunden, handelt es sich um ein gültiges Token und der Zugang zur angeforderten Ressource wird gestattet. Wird das Token nicht in der Datenbank gefunden, ist der Zugang nicht autorisiert und es werden eine Fehlermeldung und der HTTP-Status *401 Unauthorized* gesendet. ComicLib verwendet als Zugangstoken 128-stellige, hexadezimale Zeichenketten, welche im Feld *APIKey* der *Users*-Tabelle gespeichert werden.

Da das Zugangstoken im Vergleich zum normalen Benutzerpasswort sehr schlecht zu merken ist, ist es sinnvoll, das Token per API nach einer Authentifizierung per Benutzername und Passwort dem jeweiligen Benutzer zur Verfügung zu stellen. Zu diesem Zwecke wurde die API-Ressource */tokens* implementiert. Diese stellt einem Benutzer sein Zugangstoken zur Verfügung. An dieser Stelle kommt HTTP Basic Authentication zum Einsatz. Dabei werden der Benutzername und das Passwort durch einen Doppelpunkt zu einer Zeichenkette verbunden und Base64-codiert übertragen.[49] Der API-Server decodiert die Zeichenkette und zerlegt sie wieder in Benutzername und Passwort. Das Passwort wird nun per *password_verify*-Funktion mit dem Passworthash des Benutzers aus der Datenbank verglichen. Stimmen beide überein, erhält der Benutzer sein Zugangstoken in Form einer JSON-Zeichenkette zusammen mit einem HTTP-Status *200 OK*. Stimmen die Passwörter nicht überein, erhält der Nutzer stattdessen eine Fehlermeldung in Form einer JSON-Zeichenkette zusammen mit einem HTTP-Status *401 Unauthorized*.

Bearer Token Authentication und HTTP Basic Authentication sind Standardverfahren zur Authentifizierung über HTTP. Daher werden sie auch durch die meisten Hilfsprogramme zur API-Entwicklung unterstützt, darunter auch Postman. Beide Verfahren übertragen die Zugangsdaten allerdings unverschlüsselt. In einem ungesicherten Netzwerk ist es daher nötig, die Verbindung über TLS zu sichern. Da TLS domänenspezifische Zertifikate verwendet, kann ComicLib dem Betreiber der ComicLib-Installation den Aufwand, TLS einzurichten, nicht abnehmen.

Die Nutzung von Zugangstoken für den Zugang zu allen API-Ressourcen außer */token* soll ein Minimum an Sicherheit in ungesicherten Netzwerken ohne Verschlüsselung gewährleisten, da so die Kombination aus Benutzername und Passwort nur bei der ersten Nutzung der API von einem neuen Endgerät aus verwendet wird. So kann diese Zugangsdatenkombination, die von Benutzern häufig andernorts wiederverwendet wird, weniger leicht durch eine Man-in-the-Middle-Attacke abgefangen werden. Da bei der Nutzung der Weboberfläche Benutzername und Passwort nur einmal pro Session übertragen werden, ist auch hier ein Minimum an Sicherheit gegeben.

---

[48] Vgl. (Internet Engineering Task Force, The OAuth 2.0 Authorization Framework: Bearer Token Usage, 2012)
[49] Vgl. (Internet Engineering Task Force, Hypertext Transfer Protocol (HTTP/1.1): Authentication, 2014)

4 Fazit

Insgesamt konnte das Projekt wie geplant realisiert werden[50]. Die lokale Comicsammlung kann mit Hilfe der ComicVine API um Informationen zu Comicheften, -serien und -verlagen angereichert werden. Nach dem Einloggen auf der Weboberfläche steht dem Benutzer eine vollwertige Alternative zu Amazon Kindle und Comixology für Comics in offenen Dateiformaten zur Verfügung.

Comicserien können in der Serienübersicht durchstöbert sowie per Suchfunktion gefunden werden. Noch nicht komplett gelesene Comicserien werden durch ein „New“-Etikett hervorgehoben. Für jede Serie gibt es eine Detailseite mit einem Beschreibungstext und der Übersicht aller in der Sammlung enthaltenen Hefte der Serie. Die Hefte können, sofern sie ein offenes Dateiformat nutzen, direkt im Browser gelesen werden. Alle Comicdateien können auch heruntergeladen werden, wodurch Comics in proprietären Dateiformaten einfach mit der Hersteller-Anwendung geöffnet werden können. Comichefte wie auch ganze Serien können als (un-)gelesen markiert werden. Angelesene Comics erhalten automatisch ein Lesezeichen an der zuletzt gelesenen Stelle, so dass das Lesen jederzeit an dieser Stelle fortgesetzt werden kann. Angelesene Comics haben eine eigene Übersicht erhalten, wodurch auch das parallele Lesen von mehreren Comics kein Problem darstellt. Dank der Verlagsübersicht und den Verlagsdetailseiten ist das Finden von anderen Serien desselben Verlags einfach. So können Vorgänger- und Nachfolgeserien, Crossover-Serien und Spin-offs leicht gefunden werden. Auf Grund der durchgehenden Nutzung von Bootstrap zur Gestaltung der Weboberfläche passt sich diese sehr gut an die unterschiedlichen Bildschirmgrößen und -formate von Smartphones, Tablets und PCs an - siehe Abbildung 15: Die Serienübersicht auf Geräten unterschiedlicher Größe und Rotation. Bei Nutzung des Vollbildmodus in der Leseansicht gestaltet sich das Lesen besonders auf Tablets sehr bequem, da die Bildgröße so annähernd der in einem konventionellen Comicheft gleicht.

Dank der ComicLib API steht der Anbindung mobiler Apps nichts im Wege. Durch die konsequente Nutzung versionierter API-Pfade wird es in Zukunft möglich sein, die API um weitere Funktionen zu erweitern, ohne bestehende Anwendungen zu beschädigen.

Ein paar kleinere Dinge konnten leider im Rahmen des Projekts nicht mehr implementiert werden. Das Startjahr wird bei Serien noch nicht mit angezeigt und die Beschreibungstexte der Comichefte werden ebenfalls noch nirgendwo genutzt. Im Layout ergab sich für beide im Laufe der Entwicklung einfach kein guter Platz, so dass diese bisher außenvor blieben. Sobald sich ein guter Platz findet, werden beide nachträglich eingebaut werden. Ein weiteres, kleines Ärgernis besteht in der Art, wie der Lesestatus in der Weboberfläche aktualisiert wird. Zurzeit werden die ID der Serie bzw. des Hefts – je nachdem, wofür der Lesestatus aktualisiert wird – und der neue Lesestatus einfach per POST an den Server gesendet und danach die Seite neu geladen. Dadurch entstehen aber zwei Probleme. Zum einen muss die Seite so sichtbar neu aufgebaut werden, was den Eindruck, eine normale, lokale Anwendung zu nutzen, stört. Zum anderen geht durch das Neuladen der Scrollfortschritt auf der Seite verloren. Das Ändern des Lesestatus mehrerer Serien oder Hefte im unteren Bereich der Seite gestaltet sich so unnötig schwierig, da nach jedem Neuladen wieder herunter gescrollt werden muss. Beide Probleme lassen sich durch die Nutzung von AJAX zur Aktualisierung des Lesestatus beheben. Dabei erfolgen die Übertragung des POST-Formulars und das Neuladen der Seite im Hintergrund.[51] Sobald das Neuladen abgeschlossen ist, wird nur der geänderte Teil der Seite durch den aktuelleren Teil aus der neu geladenen Seite ersetzt. Diese Änderung konnte aus Zeitgründen leider nicht mehr implementiert werden, wird aber zeitnah nachgeholt.

---

[50] Das öffentliche Git-Depot für ComicLib findet sich unter https://github.com/ahahn94/ComicLib.

[51] Vgl. (w3schools.com, 2019)

Für die Zukunft des Projekts sind viele Erweiterungen und Verbesserungen im Detail denkbar. Beispielsweise könnte eine eigene Ansicht nur für die ungelesenen Comicserien geschaffen werden, um das Auffinden neuer Comics weiter zu erleichtern. Mit der Einrichtung eines echten Mehrbenutzerbetriebs könnten auch Teilsammlungen für unterschiedliche Benutzergruppen eingeführt werden, um zum Beispiel zwischen Comicsammlungen für Kinder und für Erwachsene zu unterscheiden. Die ComicVine API bietet noch viele weitere Möglichkeiten, Comics unter einander zu verknüpfen, zum Beispiel anhand der auftretenden Charaktere oder der Autoren. So könnten auch, ähnlich der Verlagsseiten, Seiten für Charaktere oder Autoren implementiert werden, die neben einem Beschreibungstext die jeweiligen Comicserien oder -hefte enthalten (am besten sogar umschaltbar).

Eine mit Sicherheit sinnvolle Verbesserung wäre es, ComicLib auf HTTPS umzustellen. Da TLS jedoch auf Domänennamen-Basis arbeitet, könnte mit ComicLib nur ein selbstsigniertes Standardzertifikat ausgeliefert werden, was im Internetbrowser zu einer Warnmeldung führen würde. Zusätzlich könnte es daher sinnvoll sein, die Docker-Konfiguration so zu erweitern, dass der Let's Encrypt Zertifikatbot mitinstalliert wird, damit Benutzer mit diesem leicht ein korrektes TLS-Zertifikat für die eigene Domain erhalten können.

ComicLib ist in Hinblick auf die Weboberfläche eine vollwertige Alternative zu Amazon Kindle und Comixology, jedoch bieten diese Plattformen auch mobile Apps an, die das Mitnehmen und die Offline-Nutzung von einzelnen Comics der Sammlung ermöglichen.[52] Daher soll im Rahmen der Bachelorarbeit eine Android-App entwickelt werden, die diese Lücke in ComicLib schließt. Diese soll auf der ComicLib API aufsetzen und neben dem Herunterladen und Lesen von Comics auch die Synchronisation des Lesestatus über mehrere Geräte hinweg ermöglichen.

[52] Vgl. (Amazon, 2019)

Anhang

Anhang 1 - Screenshots der Weboberfläche

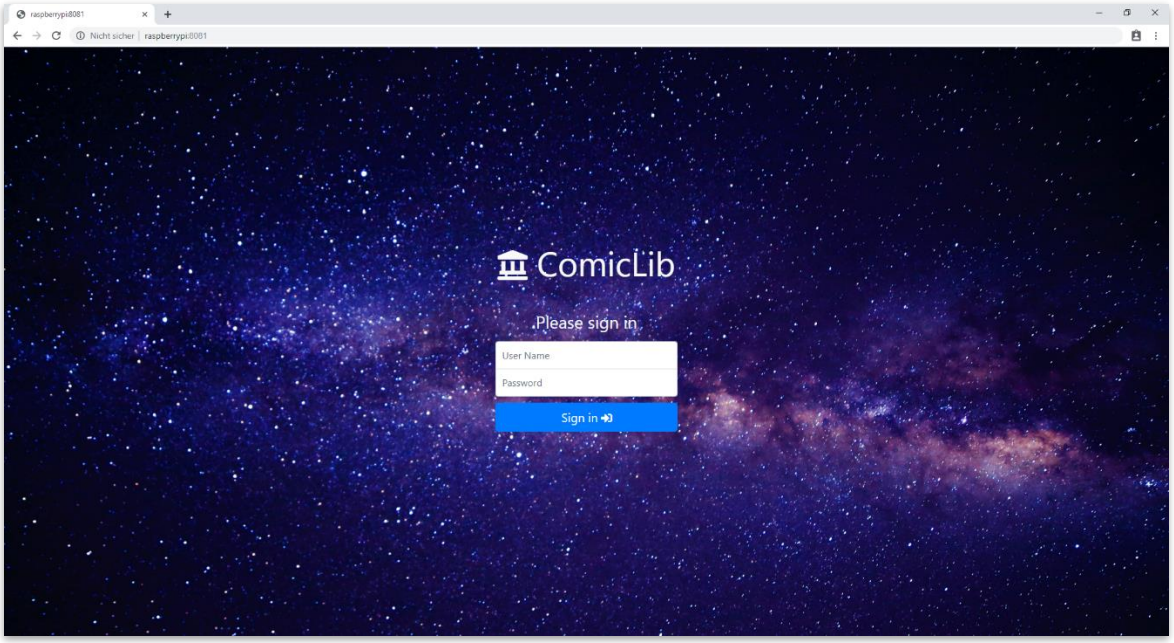

Abbildung 3: Login-Sicht

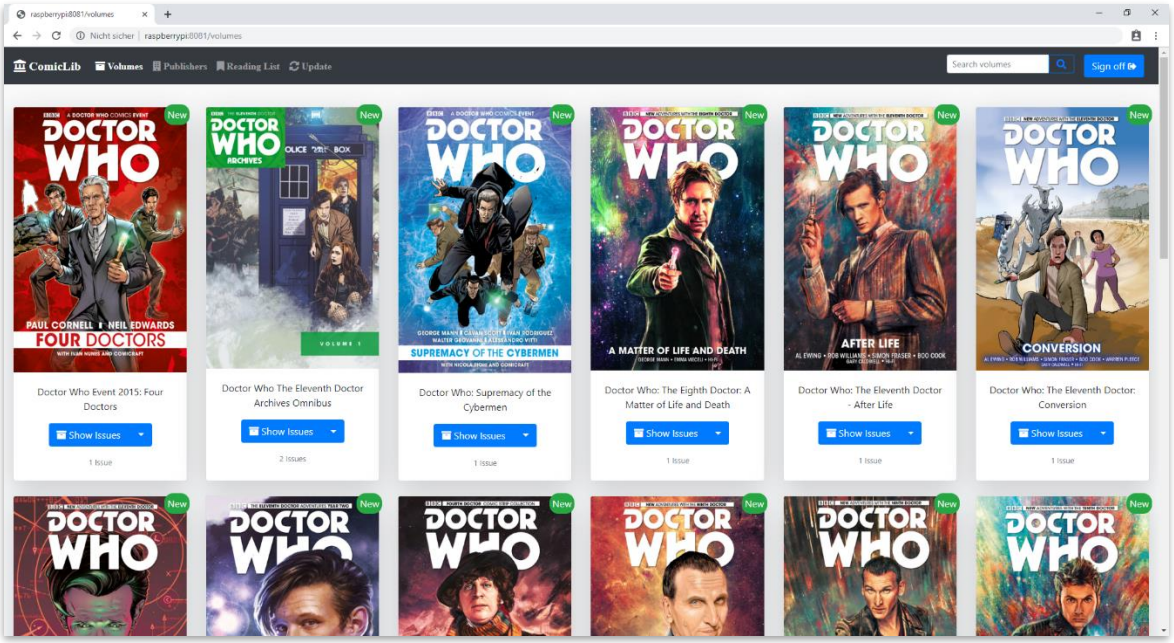

Abbildung 4: Serienübersicht
Die Seitennavigation wird erst angezeigt, wenn die Sammlung mehr als eine Seite füllt. Da die Beispielsammlung weniger als 24 Serien umfasst, ist diese Menüleiste ausgeblendet.

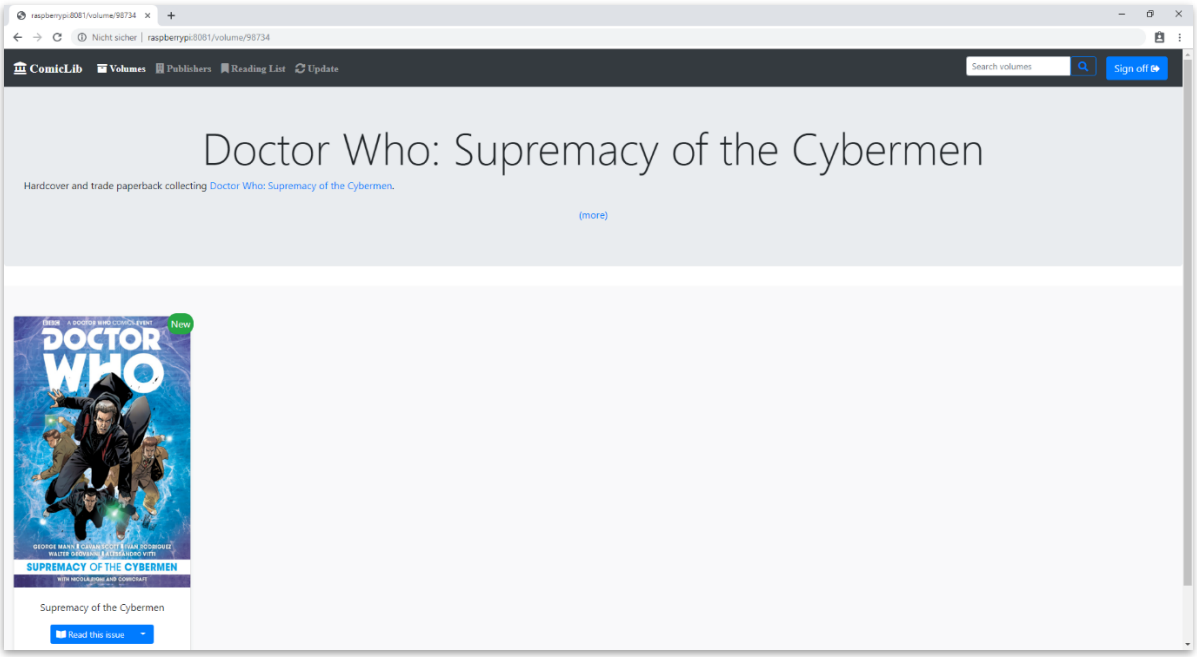

Abbildung 5: Seriendetailsicht

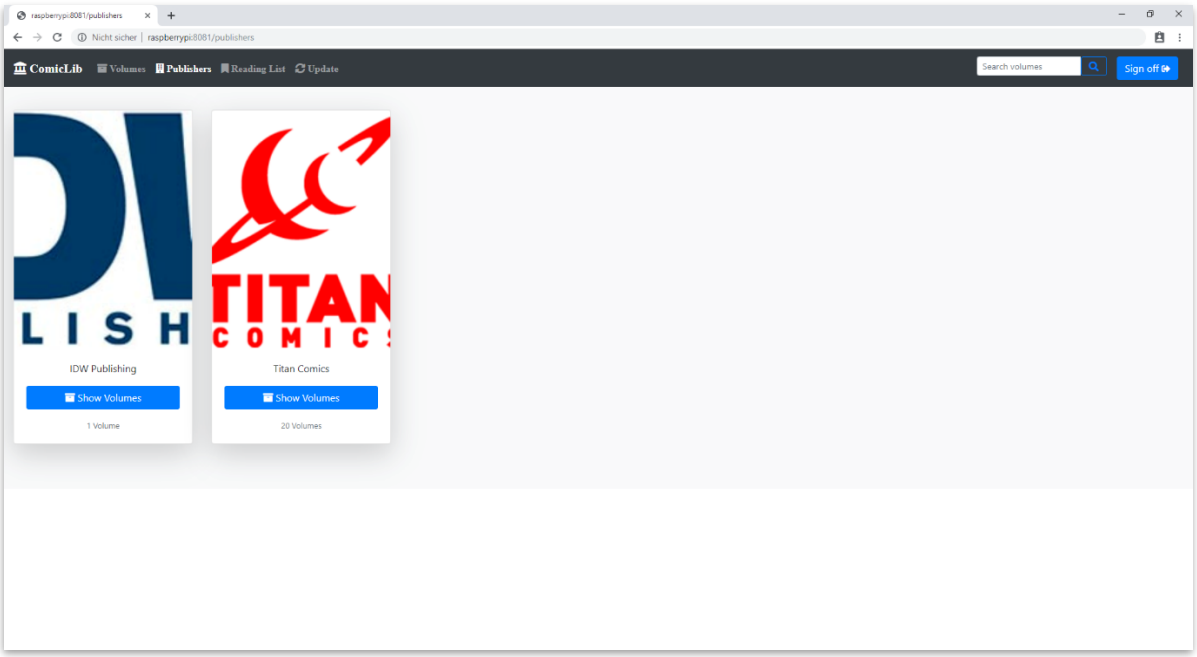

Abbildung 6: Verlagsübersicht

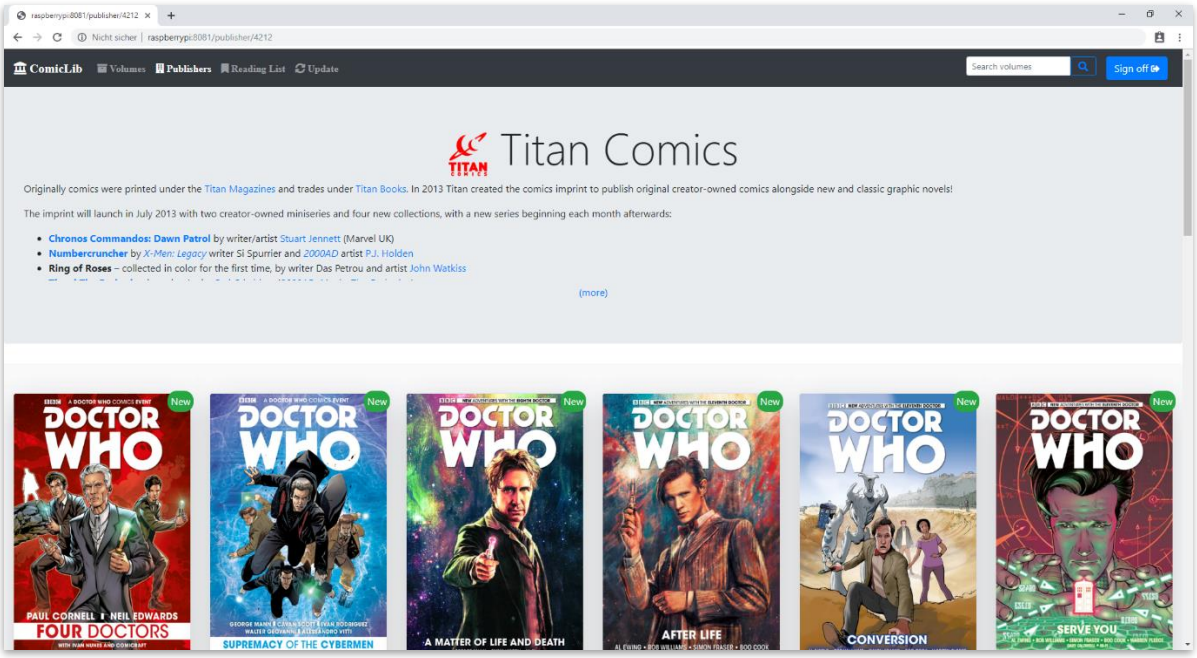

Abbildung 7: Verlagsdetailsicht

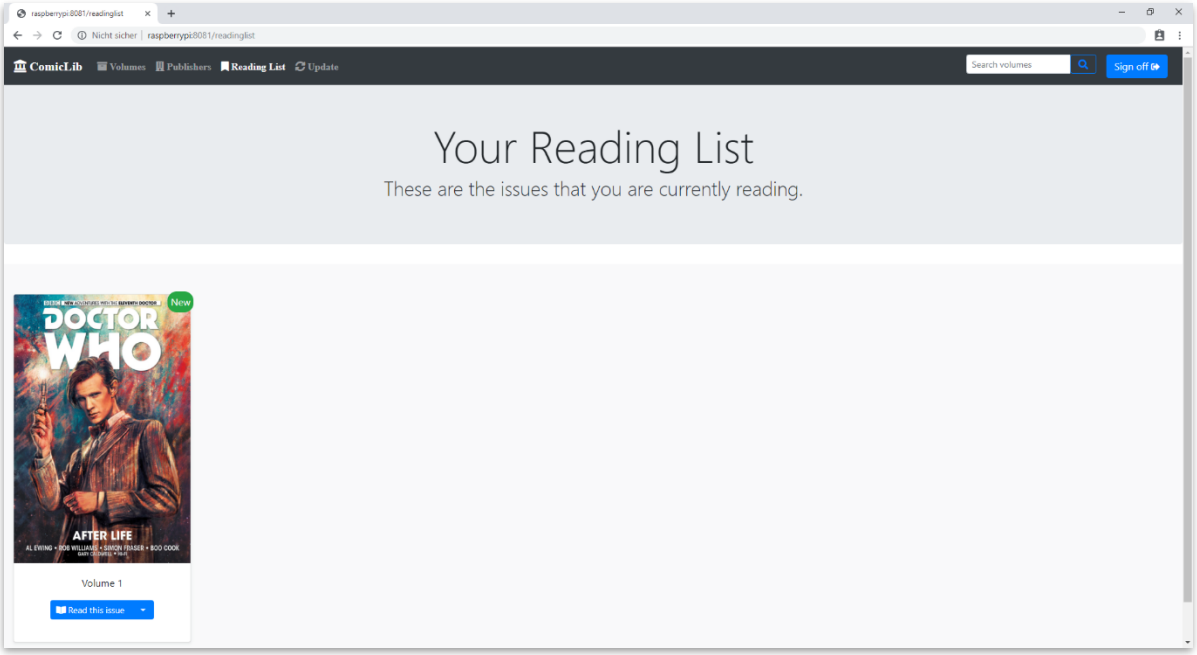

Abbildung 8: Leselistensicht

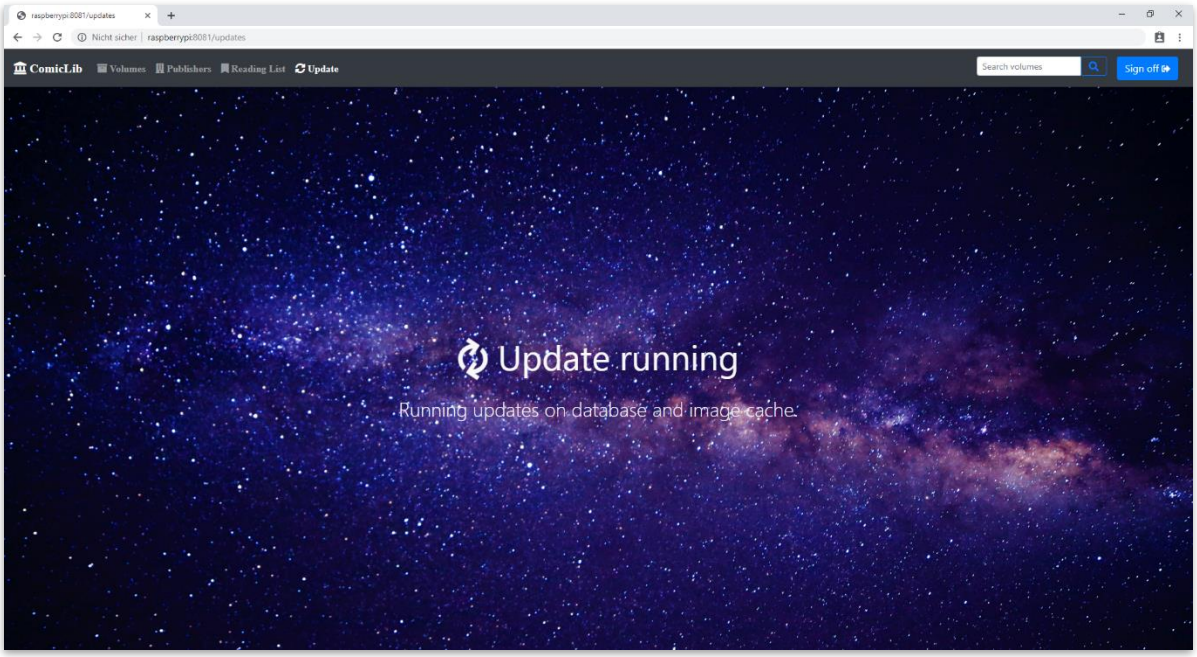

Abbildung 9: Aktualisierung im Gange

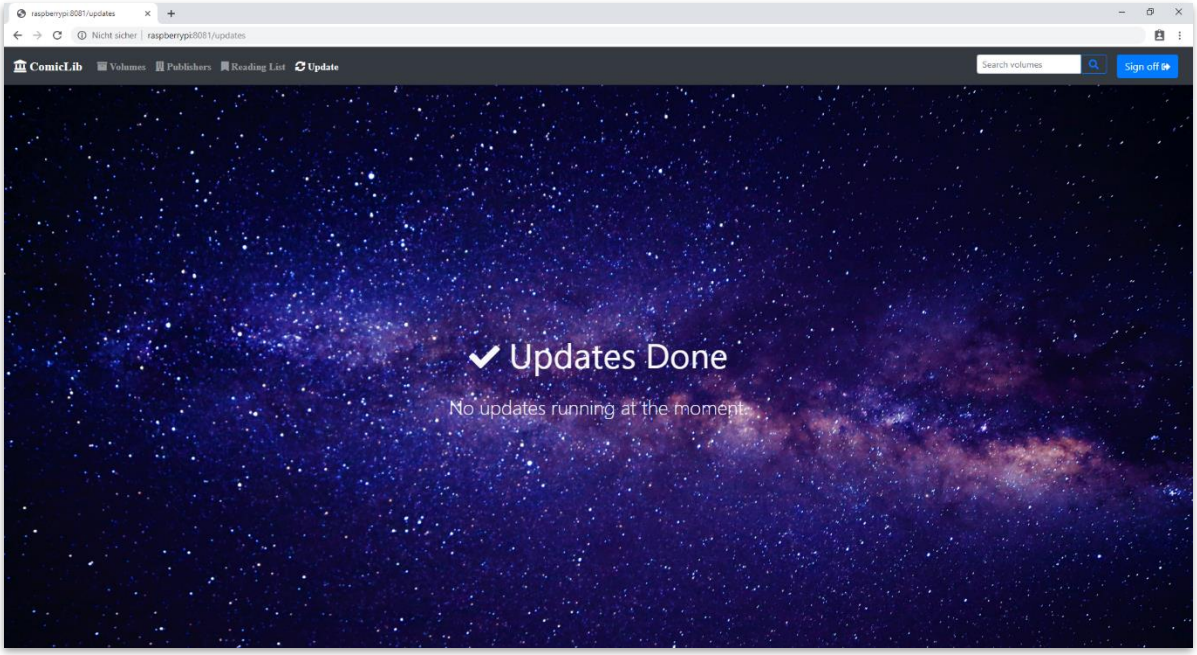

Abbildung 10: Aktualisierung abgeschlossen

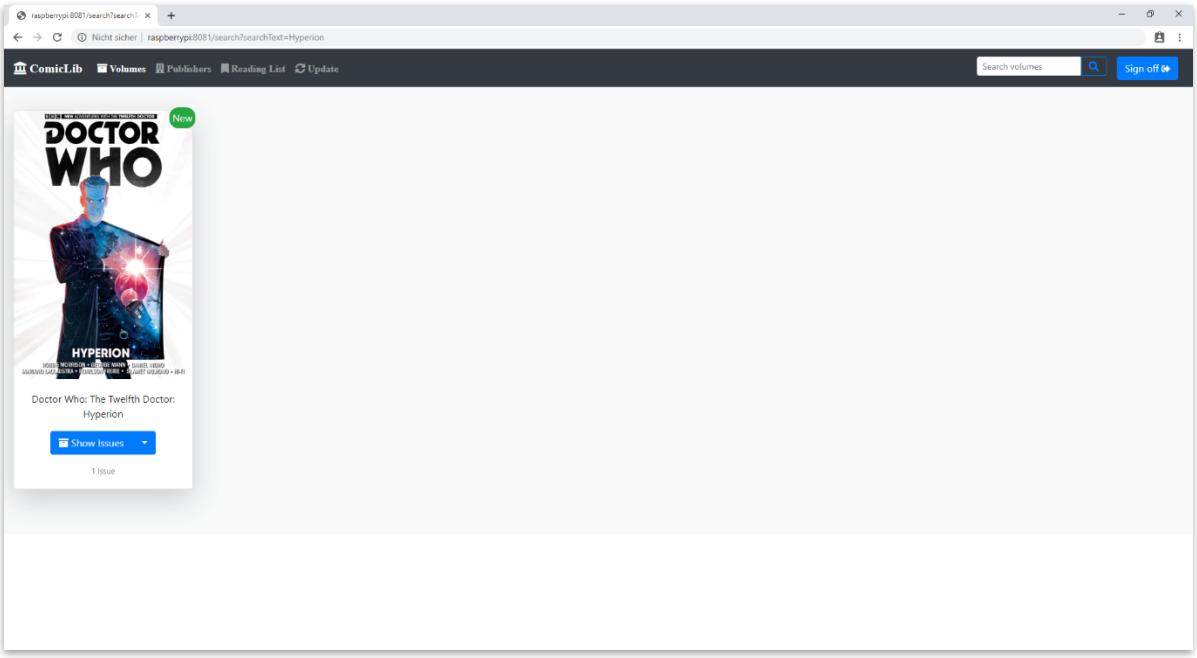

Abbildung 11: Suchergebnissicht

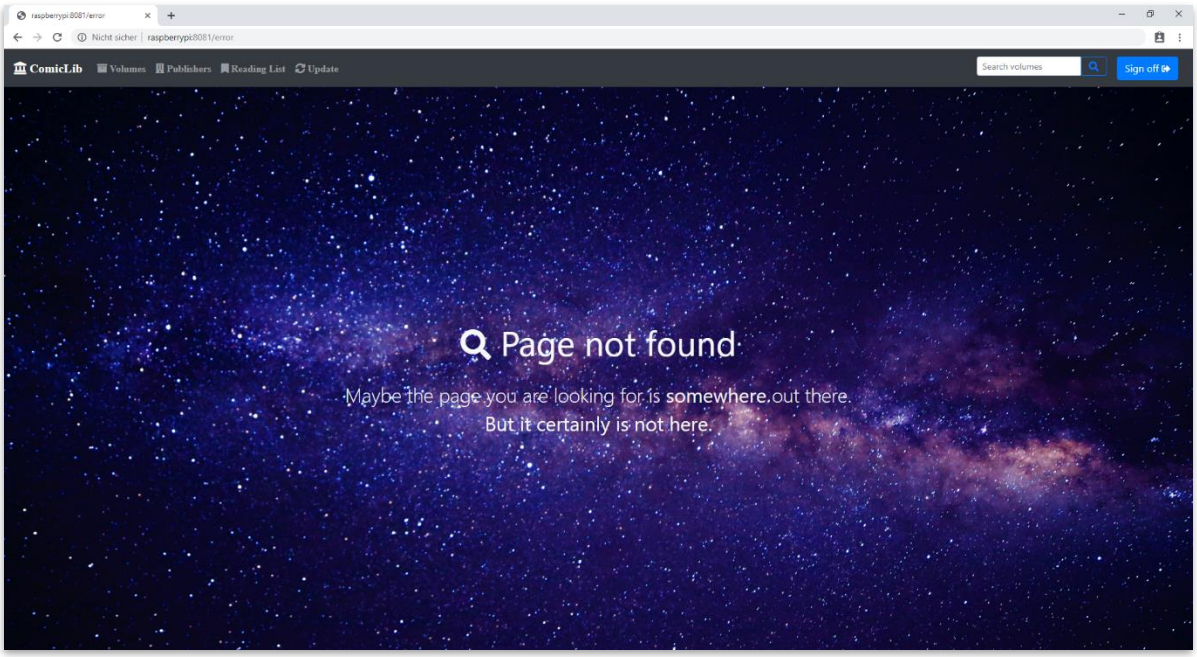

Abbildung 12: Seite-nicht-gefunden-Sicht

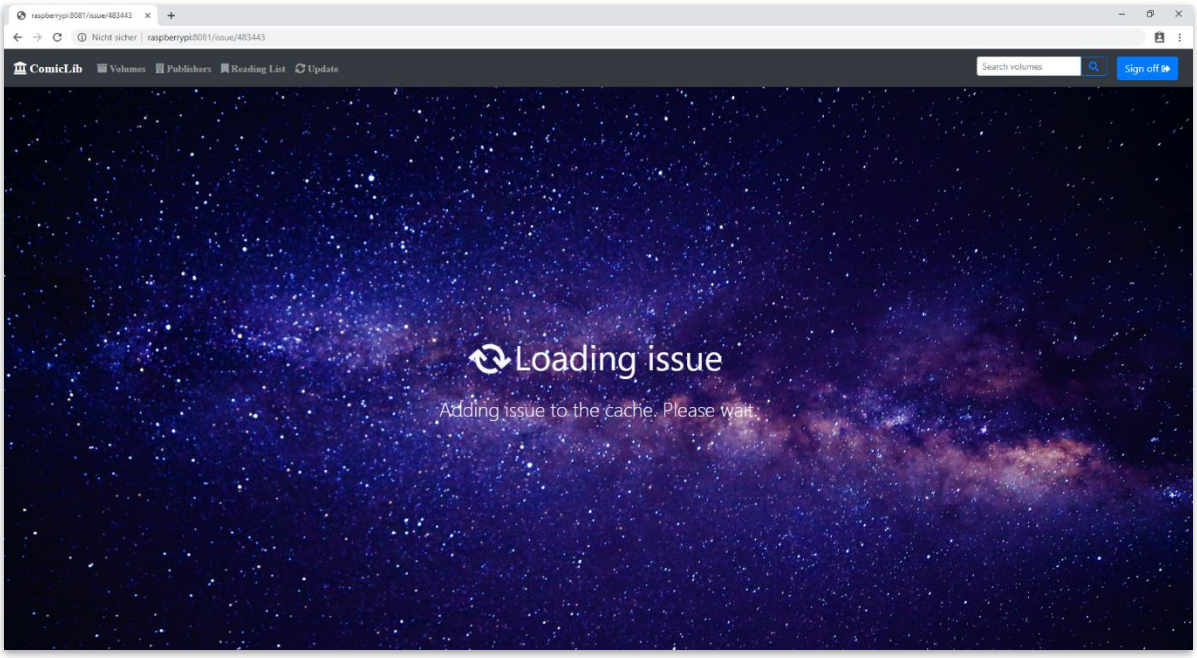

Abbildung 13: Ladesicht

Abbildung 14: Leseansicht

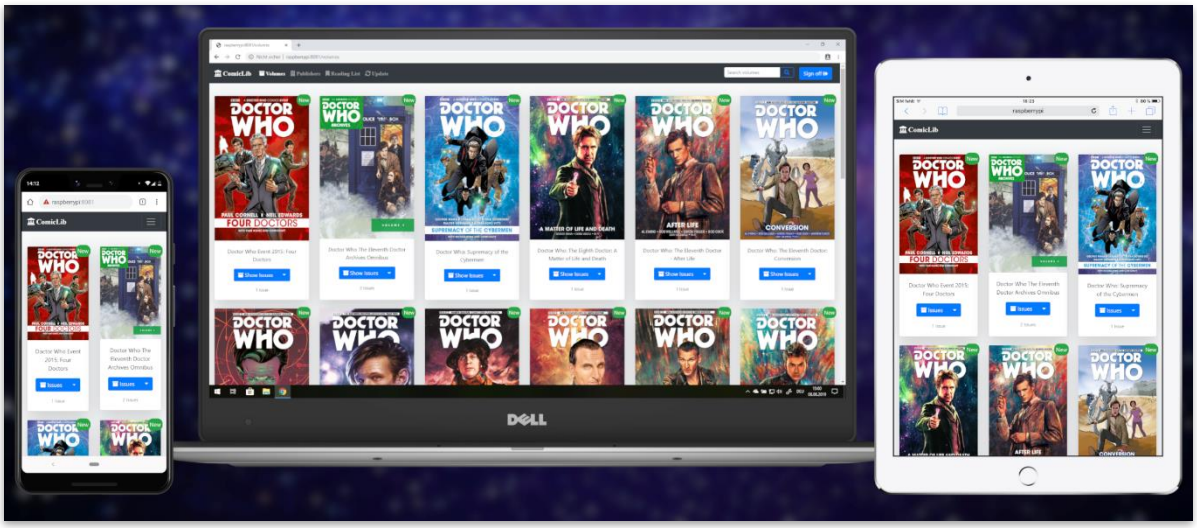

Abbildung 15: Die Serienübersicht auf Geräten unterschiedlicher Größe und Rotation

Literatur

Amazon. (2019). *Kindle Apps.* Abgerufen am 25. Juni 2019 von https://www.amazon.de/kindle-dbs/fd/kcp

CBS Interactive Inc. (2015). *API Rate Limiting*. Abgerufen am 25. Juni 2019 von https://comicvine.gamespot.com/forums/api-developers-2334/api-rate-limiting-1746419/?page=1#js-message-15811401

CBS Interactive Inc. (2019). *ComicVine API Documentation*. Abgerufen am 25. Juni 2019 von https://comicvine.gamespot.com/api/documentation

Docker Inc. (2019). *What is a Container?* Abgerufen am 25. Juni 2019 von https://www.docker.com/resources/what-container

Drake, M. (2018). *How To Install Linux, Apache, MySQL, PHP (LAMP) stack on Ubuntu 18.04*. Abgerufen am 25. Juni 2019 von https://www.digitalocean.com/community/tutorials/how-to-install-linux-apache-mysql-php-lamp-stack-ubuntu-18-04

Fielding, R. T. (2000). *Architectural Styles and the Design of Network-based Software Architectures*. Abgerufen am 25. Juni 2019 von https://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm

getbootstrap.com. (2019). *Carousel*. Abgerufen am 25. Juni 2019 von https://getbootstrap.com/docs/4.3/components/carousel/

Hahn, A. (2019). *ComicLib API Version 1*. Abgerufen am 25. Juni 2019 von https://documenter.getpostman.com/view/5715403/S1a35U3H

Internet Engineering Task Force. (2012). *The OAuth 2.0 Authorization Framework: Bearer Token Usage*. Abgerufen am 25. Juni 2019 von https://tools.ietf.org/html/rfc6750

Internet Engineering Task Force. (2014). *Hypertext Transfer Protocol (HTTP/1.1): Authentication*. Abgerufen am 25. Juni 2019 von https://tools.ietf.org/html/rfc7235

Morris, J. (2017). *How to Build a PHP Login Form Using Sessions*. Abgerufen am 25. Juni 2019 von https://johnmorrisonline.com/build-php-login-form-using-sessions/

Mozilla. (2019). *JavaScript Guide*. Abgerufen am 25. Juni 2019 von https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide

Perry, J. S. (2016). *What is Docker? A Primer on the Benefits of Containers for Applications*. Abgerufen am 25. Juni 2019 von https://developer.ibm.com/dwblog/2016/what-is-docker-containers/

Reimers, N. (2017). *Model-View-Controller in PHP*. Abgerufen am 25. Juni 2019 von https://www.php-einfach.de/experte/objektorientierte-programmierung-oop/php-design-patterns/model-view-controller-in-php/

Sharpened Productions. (2017). *.AZW File Extension*. Abgerufen am 25. Juni 2019 von https://fileinfo.com/extension/azw

Sharpened Productions. (2018). *.CBR File Extension*. Abgerufen am 25. Juni 2019 von https://fileinfo.com/extension/cbr

Sharpened Productions. (2018). *.CBZ File Extension*. Abgerufen am 25. Juni 2019 von https://fileinfo.com/extension/cbz

Spichale, K. (2016). *Best Practices bei der API-Versionierung*. Abgerufen am 25. Juni 2019 von https://www.heise.de/developer/artikel/Best-Practices-bei-der-API-Versionierung-3210728.html

The PHP Group. (2019). *PHP-Handbuch*. Abgerufen am 25. Juni 2019 von https://www.php.net/manual/de/

w3schools.com. (2019). *AJAX Introduction*. Abgerufen am 25. Juni 2019 von https://www.w3schools.com/xml/ajax_intro.asp

Wagner, J. (2019). *Lazy Loading Images and Video*. Abgerufen am 25. Juni 2019 von https://developers.google.com/web/fundamentals/performance/lazy-loading-guidance/images-and-video/

Abbildungen

Tabellen

Abkürzungen

Abkürzung	Bedeutung
AJAX	„Asynchronous JavaScript and XML“. Eine Methode zum Nachladen von Informationen einer Webseite im Hintergrund.
AMD64	Nach ihrem Entwickler „Advanced Micro Devices“ benannte 64-Bit Prozessorarchitektur. Häufig als Synonym für alle 64-Bit-Erweiterungen der x86-Architektur verwendet.
API	“Application Programming Interface”. Eine Programmierschnittstelle.
ARM	Nach ihrem Hersteller „Advanced RISC Machines“ benannte Prozessorarchitektur.
CBR	„Comic Book RAR“. Ein Dateiformat für Comichefte, das auf dem RAR-Archivformat basiert.
CBZ	„Comic Book ZIP“. Ein Dateiformat für Comichefte, das auf dem ZIP-Archivformat basiert.
CSS	„Cascading Style Sheets“. Eine Stylesheet-Sprache zur Gestaltung von Webseiten.
DRM	„Digital Rights Management“. Siehe Glossar.
HTTP	„Hypertext Transfer Protocol“. Ein Protokoll zur Übertragung von Dokumenten über das Internet. Meistens zum Laden von Webseiten verwendet.
HTTPS	„Hypertext Transfer Protocol Secure“. Um TLS-Verschlüsselung erweitertes HTTP-Protokoll.
ID	„Identifikationsnummer“.
IDE	„Integrated Development Environment“. Eine integrierte Entwicklungsumgebung vereint in sich einen Quelltext-Editor mit zusätzlichen Werkzeugen zur Softwareentwicklung.
INI	Vom Englischen „Initialization“. Ein Dateiformat zum Speichern von Konfigurationen als Schlüssel-Wert-Paare.
JSON	„JavaScript Object Notation“. Ein Datenformat zur Darstellung strukturierter Daten als Text.
MVC	„Model View Controller“. Ein Entwurfsmuster für Software.
PDF	„Portable Document Format“. Ein Dateiformat zur Speicherung von elektronischen Dokumenten.
PHP	„PHP Hypertext Processor“. Eine Skriptsprache zur Programmierung von Webanwendungen.
RAR	Von „Roshal Archive“. Ein Dateiformat für Archivdateien.
REST	„Representational State Transfer“. Ein Paradigma zur Gestaltung von APIs.
SQL	„Structured Query Language“. Eine standardisierte Datenbanksprache.
TLS	„Transport Layer Security“. Ein Verschlüsselungsprotokoll zur sicheren Datenübertragung.
UML	„Unified Modeling Language“. Eine Modellierungssprache zur Gestaltung von Diagrammen in der Softwareentwicklung.
URL	„Uniform Resource Locator“. Adresse des Links auf eine Ressource.
XML	„Extensible Markup Language“. Eine Auszeichnungssprache zur Darstellung strukturierter Daten als Text.
ZIP	Aus dem Englischen „Zipper“. Ein Dateiformat für Archivdateien.

Glossar

Begriff	**Bedeutung**
agnostisch	Neutral, unabhängig bezüglich eines Sachverhalts (abgeleitet aus dem technischen, englischen „agnostic“).
Apache	Apache HTTP Server ist ein quelloffener Webserver.
Asynchronous JavaScript and XML	Eine Methode zum Nachladen von Informationen einer Webseite im Hintergrund.
Authentifizierung	Prüfung der Identität eines Benutzers anhand eines von diesem bereitgestellten Identitätsbeweises.
Authorization-Header	Ein Teil des HTTP-Headers, der Informationen zur Authentifizierung enthält.
Back-End	Der serverseitige Teil einer Webanwendung.
Base64	Ein Verfahren zur Codierung von Binärdaten als Zeichenketten.
BCrypt	Eine kryptologische Hashingfunktion.
Bearer Token Authentication	Ein Verfahren zur Authentifizierung mittels einer kryptografischen Schlüsselzeichenkette.
Best Practice	Eine bewährte oder optimale Vorgehensweise.
Bootstrap	Eine CSS- und Javascript-Bibliothek zur Gestaltung von Webseiten mit etablierten Designs und Strukturen.
Button	Ein Druckknopf in einer Benutzeroberfläche.
Cache	Ein Zwischen-/Pufferspeicher.
Carousel	Ein Design-Element aus Bootstrap zur Darstellung von Bildern. Es ist an einen Diaprojektor angelehnt. Es ist immer genau ein Bild zu sehen und es kann zum jeweils nächsten oder vorherigen Bild navigiert werden.
Container	Eine virtuelle Maschine in der Containervirtualisierung.
Containerisierung	Synonym zu Containervirtualisierung.
Containervirtualisierung	Besonders ressourcenschonende Art der Virtualisierung.
Controller	Der Programmsteuerungsteil des MVC-Pattern.
Crossover-Serie	Eine Comic-Serie, die Charaktere aus mehreren Serien vereint.
Dart	Eine Programmiersprache, die als moderne Alternative zu Javascript entwickelt wird.
Data-Source-Attribut	Ein Attribut in HTML, dass auf eine Datei verlinkt, ohne dass diese automatisch beim Laden der Seite mitgeladen wird.
Debian	Ein auf Linux basierendes, freies Betriebssystem.
DELETE-Methode	Eine HTTP-Methode, die dem Löschen einer Ressource dient.
Digital Rights Management	Maßnahmen zum elektronischen Kopierschutz digitaler Güter.
Display	Englisch für „Bildschirm“.
Django	Ein Framework zur Entwicklung von Webanwendungen in Python.
Docker	Eine Software zur Containervirtualisierung von Anwendungen.
Docker-Compose	Software, die die gemeinsame Verwaltung mehrerer Docker-Container ermöglicht.
Domänenname	Ein Name, der stellvertretend für eine Netzwerkadresse steht. Zum Beispiel example.com.
Dropdown-Menü	Ein ausklappendes Menü. Wird meistens durch einen Button ein- und ausgeklappt.
E-Book	Ein digitales Buch.
E-Book-Reader	Eine Software zum Betrachten digitaler Bücher.

Entity-Relationship-Diagramm	Ein Diagrammtyp zur Darstellung von Entitäten und deren Relationen zu einander.
Flowchart	Ein Diagrammtyp zur Visualisierung von Abläufen.
Font Awesome	Eine CSS- und Javascript-Bibliothek zur Darstellung gängiger Icons.
Formatoverhead	Der Anteil am Datenaufkommen, der durch Informationen zum Format und nicht durch Nutzdaten entsteht.
Framework	Eine Softwarebibliothek, die als Rahmen zur Entwicklung anderer Anwendungen dient und zu diesem Zwecke Komponenten, Strukturen und Funktionen bereitstellt.
Front-End	Der clientseitige Teil einer Webanwendung.
Gesalzen	Bei einem gesalzenen Hash wurde beim Hashing an den Klartext ein so genanntes Salt (eine zufällige Zeichenfolge) angehangen.
GET-Anfrage	Eine Anfrage mit Hilfe des HTTP-Verbs GET. Die meisten Webseitenaufrufe finden über GET statt.
GET-Methode	Eine Methode zur Übertragung von Formulardaten an einen Webserver. Dabei wird das Formular als Zeichenkette in der URL übertragen.
GET-Parameter	Ein Parameter in einer GET-Anfrage. Man sieht sie häufig in der Adresszeile eines Browsers (der Teil hinter dem „?“).
Git	Eine Software zur verteilten Versionsverwaltung von Dateien.
Hashing	Abbildung eines Eingabewerts auf eine Zielmenge durch eine Einwegfunktion.
HTTP Basic Authentication	Ein Verfahren zur Authentifizierung mittels Benutzername und Passwort, das Teil des HTTP Protokolls ist.
HTTP-Caching	Das Zwischenspeichern von HTTP-Nachrichten.
HTTP-Header	Der Metadatenteil einer HTTP-Nachricht. Enthält unter anderem den Statuscode und den Inhaltstyp der Nachricht.
HTTP-Statuscode	Der Statuscode einer HTTP-Nachricht. Dieser gibt Auskunft darüber, ob eine Anfrage erfolgreich war oder welcher Fehler aufgetreten ist.
HTTP-Verben	Die HTTP-Methoden (GET, POST, PUT, DELETE, u.a.) werden auch als HTTP-Verben bezeichnet. Jede Methode definiert eine andere Aktion auf einer Ressource.
Hypervisor	Eine Abstraktionsschicht zwischen Hardware und virtueller Maschine, die der virtuellen Maschine virtuelle Hardware und Ressourcen zur Verfügung stellt.
HypriotOS	Ein auf Containervirtualisierung spezialisiertes Betriebssystem für Raspberry Pi Einplatinencomputer.
Icon	Ein Piktogramm/Symbol.
Image	Ein Betriebssystemabbild inklusive der installierten Software und Konfiguration.
ImageMagick	Eine Software zur Erstellung und Bearbeitung von Grafiken. Kann unter anderem PDF-Seiten als Bilddateien exportieren.
Intel Core i5 CPU	Ein Prozessor für Desktop-Rechner der Marke Intel.
Javascript	Eine Programmiersprache, die vor allem zur Umsetzung dynamischer Inhalte in Webseiten verwendet wird.
jQuery	Eine Javascript-Bibliothek zur Vereinfachung der Nutzung von Standard-Funktionen.

Kanban	Eine Methode zur Arbeitsplanung in der Softwareentwicklung, bei der die Anzahl paralleler Arbeiten begrenzt wird.
Krähenfußnotation	Eine Notation in Entity-Relationship-Diagrammen.
Lazy Loading	Daten werden erst bei Bedarf geladen.
Let's Encrypt	Eine Zertifizierungsstelle, die TLS-Zertifikate herausgibt.
Let's Encrypt Zertifikatbot	Eine Software, die TLS-Zertifikate von Let's Encrypt erstellen und automatisch erneuern kann.
Loadbalancer	Eine Software, die zur Reduzierung der individuellen Last Anfragen über mehrere Rechner verteilt.
Loadbalancing	Das Verteilen von Anfragen über mehrere Rechner zur Reduzierung der Last des individuellen Rechners.
Logging	Die automatische Protokollierung von Ereignissen und Nachrichten.
Man-in-the-Middle-Attacke	Eine Angriffsform, bei der der Angreifer die Kommunikation zweier Kommunikationspartner über sich selbst umleitet und diese so abhören und manipulieren kann.
MariaDB	Eine auf dem SQL-Standard basierende Datenbanksprache.
Medientyp-Header	Ein Teil des HTTP-Headers, der den Medientyp (z.B. JSON oder PDF) des Nachrichteninhalts angibt.
Mind-Map	Ein Diagrammtyp zur visuellen Organisation von Informationen.
Model	Der Datenmodell-Teil des MVC-Pattern.
Model-View-Controller-Pattern	Ein Muster zum Entwurf von Software, bei dem zwischen Datenmodell, Darstellung und Programmsteuerung getrennt wird.
MySQL	Eine auf dem SQL-Standard basierende Datenbanksprache.
NGINX	Ein quelloffener Webserver.
Node.js	Eine Software zum Betrieb von Serveranwendungen in Javascript.
Open Source	Software, bei der der Quelltext nicht unter Verschluss steht, sondern jedem zugänglich ist.
Oracle SQL	Eine auf dem SQL-Standard basierende Datenbanksprache.
Overlay	Eine grafische Komponente, die über anderen eingeblendet wird.
Pattern	Ein (Architektur-/Entwurfs-)Muster.
Popper.js	Eine Javascript-Bibliothek zur Darstellung von Pop-ups und Hilfetexten.
Pop-up	Ein Designelement, das plötzlich auftaucht und andere Elemente überdeckt.
PostgreSQL	Eine auf dem SQL-Standard basierende Datenbanksprache.
POST-Methode	Eine Methode zur Übertragung von Formulardaten an einen Webserver. Dabei wird das Formular im Anfragekörper gesendet.
Proxy	Eine Software, die durch sie hindurch geleiteten Datenverkehr unter anderem filtern und zwischenspeichern kann.
PUT-Methode	Eine HTTP-Methode, die dem Aktualisieren einer Ressource dient.
Python	Eine interpretierte Programmiersprache.
Raspberry Pi	Eine Modellreihe von Einplatinencomputern auf Basis der ARM-Prozessorarchitektur.
Repräsentation	Darstellungs- oder Speicherformat einer Datenmenge.

Representational State Transfer	Ein häufig genutztes Paradigma zur Gestaltung von APIs.
Request	Eine Anfrage an einen Server.
Responsive Design	Ein Paradigma zur Gestaltung von Webseiten. Dabei werden diese so gestaltet, dass sie ihr Aussehen an das Endgerät, z.B. PC oder Tablet, anpassen.
Ruby	Eine interpretierte Programmiersprache.
Ruby on Rails	Ein Framework zur Entwicklung von Webanwendungen in Ruby.
Scrollfortschritt	Der auf einer Webseite durch Scrollen zurückgelegte Weg und die daraus resultierende Position.
Selbstsigniert Zertifikat	Ein Zertifikat, das nicht durch eine Zertifizierungsstelle herausgegeben, sondern durch einen Benutzer generiert wurde.
Session	Englisch für „Sitzung“. Eine stehende Verbindung eines Clients zu einem Server.
Sicht	Eine Unterseite einer Webseite.
Source-Attribut	Ein Attribut in HTML, das auf eine Datei verlinkt, die beim Laden der Seite mitgeladen wird.
Spin-off	Englisch für „Ableger“. Eine Serie, die aus einer anderen hervorgeht.
TLS-Zertifikat	Ein digitales Zertifikat als Identitätsnachweis in TLS.
Token	Ein Sicherheitsschlüssel.
Token-Authentifizierung	Ein Identitätsnachweis anhand eines einzelnen Schlüssels (Token)
Touchnavigation	Die Nutzung eines Touchscreens zur Navigation.
Touchscreen	Ein berührungsempfindlicher Bildschirm.
Trigger	Eine in einer Datenbank hinterlegte Funktion, die beim Eintreten vorgegebener Bedingungen ausgeführt wird.
Typescript	Eine Programmiersprache, die auf Javascript basiert.
Update	Englisch für „Aktualisierung“.
Venn-Diagramm	Ein Diagrammtyp zur Darstellung von Relationen zwischen Mengen.
View	Eine virtuelle Datenbanktabelle, die auf Basis einer hinterlegten Anfrage bei jeder Anfrage neu generiert wird. Sie stellt also eine Art Alias zu einer hinterlegten Anfrage dar.
View	Der Darstellungsteil des MVC-Pattern.
Virtualisierung	Paralleler Betrieb mehrerer Betriebssysteme auf einem Computer.
Web-API	Eine über HTTP(S) bereitgestellte Programmierschnittstelle.
WHERE-Statement	Eine Filterregel in einer SQL-Anfrage.
Zugangstoken	Eine nach kryptografischen Anforderungen erstellte Zeichenkette als Erkennungsschlüssel für eine Authentifizierung.