Fall 2015

A JavaScript and PHP EPUB reader web application

Xiaqing He

A JavaScript and PHP EPUB reader web application

A Project Report

Presented to

The faculty of Department of Computer Science

San Jose State University

In Partial Fulfillment

Of the Requirements for the Degree

Master of Science

By

Xiaqing He

December 2015

The Undersigned Project Committee Approves the Project Titled

A JavaScript and PHP EPUB reader web application

By

Xiaqing He

APPROVED FOR THE DEPARTMENT OF COMPUTER SCIENCE

Dr. Chris Pollett, Department of Computer Science	Date
Dr. Thomas Austin, Department of Computer Science	Date
Dr. Chris Tseng, Department of Computer Science	Date

Acknowledgement

First, I would like to give my deep thanks to my project advisor, Dr. Chris Pollett, for his encouragement, suggestions and support during the whole process of this project. Without his technical advisement, guidance and thoughtful insight, I would not have been able to complete the project on time.

Second, I would like to thank my committee members, Dr. Chris Tseng and Dr. Thomas Austin, for their efforts, time and feedback.

Last, I would like to thank my husband, Faya Wang, who is always my strong support for everything, my parents, who raised me, and my lovely kids, Audrey and Bryan, who make my life full of fun.

Table of Contents

Abstract

A JavaScript and PHP EPUB reader web application

By Xiaqing He

EPUB is one of the most popular ebook formats. It is supported by many e-Readers, such as Apple's iBooks, BlackBerry Playbooks, Sony Reader, Kobo eReader, Amazon Kindle Fire, and the Mozilla Firefox add-on EPUBReader.

In this report, we describe our implementation of a JavaScript and PHP EPUB reader web application. This web application allows users to read the EPUB format books easily across multiple devices without any specified platform lock-in.

Our application provides an EPUB library. When a user logs in, he can borrow an EPUB book from the library and save into his own bookshelf, then choose any one from his own bookshelf to read. Our application tracks where a user has been reading so users can easily pick up where they left off if they leave the application.

1.Introduction

EPUB, like PDF and Word, is a format for representing documents in electronic form. It is vendor-independent XML-based and is widely supported by many eReaders, such as iBooks, Blackberry Playbooks, Sony Reader, and so on. There is also Mozilla Firefox add-on, EPUBReader, that reads EPUB files. However, there are some shortages for these existing devices and applications. A drawback of eReaders is Platform lock-in, and one for an add-on is the inability to easily sync page positions across multiple devices. In this project, we will implement a web application for the EPUB format to get rid of these drawbacks.

In my application, users can read the EPUB format books easily, synchronized across multiple devices and without any specified platform lock-in. We stored massive EPUB books in servers and provided them to the user as a library entry. User can borrow those free books, which will be saved into user's bookshelf. He can then access any book on his own bookshelf at any time. The application will track his last reading page to let him pick up easily next time. He can also access the application from any devices, that have been synchronized among these devices.

In Chapter 2, I'll present the background for the project with an introduction of EPUB and a comparison with existing EPUB reader devices and Apps. In Chapter 3, I will discuss the preliminary work of the project, which includes how JavaScript compression/decompression works. The core of the project will be described in Chapter 4, which includes the design, the implementation, and our user experience experiments. Finally, I will explain my conclusion and future work for the project in Chapter 5.

2.Background

In this chapter, I will explain what EPUB is, why it has become popular recently and the difference between EPUB and PDF. Then I will describe the existing e-reader hardware devices and software applications that support EPUB format and their issues. Furthermore, I will explain why I choose to create an EPUB reader web application, and how my application will solve those issues.

2.1 What is EPUB?

EPUB is a short for Electronic PUBlication and is sometime written as ePub[1]. Like PDF and Word, it is a format for representing documents in electronic form. It has become the most common of all the formats and is widely supported across all platforms.

The EPUB format is based on the original Open eBook(OEB) format, which was used between 1997 and 2007. The International Digital Publishing Forum (IDPF) had announced in late 2007 that OEB format would be superseded by the PUB format. The EPUB format is a free and open standard and is designed for reflowable content, meaning that the text display is not static and fixed, but rather can be optimized for particular rendering device. This feature differentiates EPUB from PDF and Word. The content, which was shown in EPUB format, is not pre-formatted as fixed book pages. When a reader device renders the contents of an EPUB book, it formats the content into pages based on the display size and text font. At this level, EPUB defines both the format for the contents and how the reading systems will render the contents.

2.1.1 What a EPUB file includes?

The EPUB file is XHTML content wrapped as a zip file package with a .epub extension, and is similar to a website. It has three major parts:

- Open Publication Structure (OPS)
- Open Packaging Format (OPF)
- OEBPS (Open eBook Publication Structure) Container Format (OCF)

In general, an EPUB file is a collection of OPS Documents, an OPF package file, and other files of different media types, which include structured text and graphics that were packaged in an OCF container.

Figure 1 shows an EPUB file's structure.

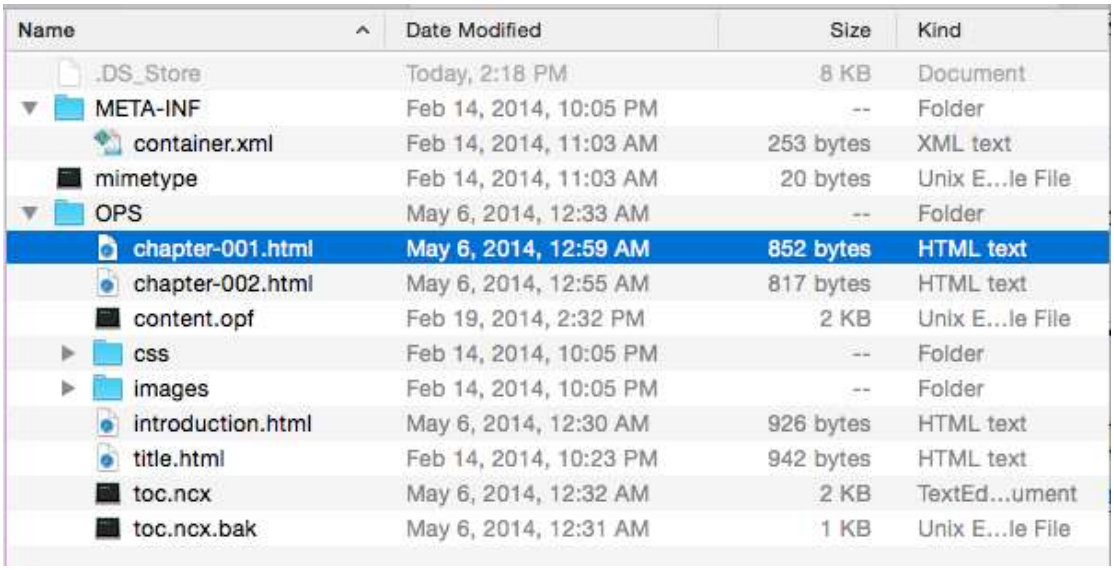

Figure 1. An EPUB file's structure

Typically, the Open Packaging Format file (OPF) serves as a container manifest and defines all of the files included in the EPUB zip file. The OPF file is an XML file, which consists of three elements:

1) <metadata>: Bibliography and rights information for the book;

2) <manifest>: Defines the pages and resources which were used in the <spine> element, including content documents, CSS style sheets, images and navigation control file (NCX);

3) <spine>: Specifies the order of pages of the book.

Figure 2 shows an example of the OPF file.

content.opf

```
<?xml version="1.0" encoding="UTF-8"?>

<package xmlns="http://www.idpf.org/2007/opf" unique-identifier="EPB-UUID" version="2.0">
   <metadata xmlns:opf="http://www.idpf.org/2007/opf"
             xmlns:dc="http://purl.org/dc/elements/1.1/">
      <dc:title>The First EPUB Book</dc:title>
      <dc:creator opf:role="aut" opf:file-as="He, Xiaqing">Xiaqing He</dc:creator>
      <dc:date opf:event="original-publication">2014</dc:date>
      <dc:publisher>epubBooks (www.epubbooks.com)</dc:publisher>
      <dc:date opf:event="epub-publication">2014-02-12</dc:date>
      <dc:subject>Mystery</dc:subject>
      <dc:source>Project Gutenberg</dc:source>
      <dc:rights>
            Provided by Xiaqing He for master graduate project. Not for commercial use.
            This EPUB eBook is released under a Creative Commons (BY-NC-ND/3.0) Licence.
            Source text and images are in the Public Domain.
      </dc:rights>
      <dc:identifier id="EPB-UUID">urn:uuid:216635F8-6BFE-1014-8FEA-F72C22E1D637</dc:identifier>
      <dc:language>en-gb</dc:language>
   </metadata>
   <manifest>
      <!-- Content Documents -->
      <item id="titlepage" href="title.html" media-type="application/xhtml+xml"/>
      <item id="epubbooksinfo" href="epubbooksinfo.html" media-type="application/xhtml+xml"/>
      <item id="introduction-001" href="introduction-001.html" media-type="application/xhtml+xml"/>
      <item id="introduction-002" href="introduction-002.html" media-type="application/xhtml+xml"/>

      <!-- CSS Style Sheets -->
      <item id="title-page-css" href="css/titlepage.css" media-type="text/css"/>
      <item id="main-css" href="css/book.css" media-type="text/css"/>

      <!-- Images -->
      <item id="epubbooks-logo" href="images/epubbooks-logo.png" media-type="image/png"/>

      <!-- NCX -->
      <item id="ncx" href="toc.ncx" media-type="application/x-dtbncx+xml"/>
   </manifest>
   <spine toc="ncx">
      <itemref idref="titlepage" linear="yes"/>
      <itemref idref="epubbooksinfo" linear="yes"/>
      <itemref idref="introduction-001" linear="yes"/>
      <itemref idref="chapter-001" linear="yes"/>
      <itemref idref="chapter-002" linear="yes"/>
   </spine>
</package>
```

Figure 2. An example of the OPF file

The content.opf file serves a very import role in the process of rendering EPUB format files. It tells EPUB Reader devices or applications where to find files in the EPUB ZIP package, what role each file plays, and in which order to display the pages of the EPUB format book.

The Open Container Format packages the whole set of EPUB files together. The container.xml file specifies the locations for the OEBPS folders that contain the content files and the OPF XML files. It has only one function: telling the eReader app where to find OPF file. We show an example of the container.xml file in Figure 3.

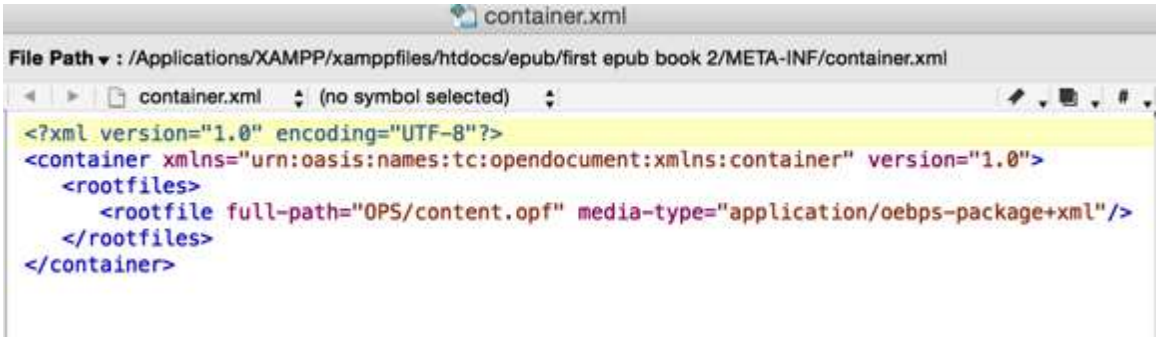

Figure 3. An example of container.xml

2.1.2 How to create a EPUB file

There are different options for how to create an EPUB file, and I will go through two basic ways due to limited space.

1) Creating from scratch

We have described in the previous section what is included in an EPUB file. We can follow that standard to create an EPUB file from scratch. Even though this "creating from scratch" method is not recommended, it is still doable, and we can demonstrate the process in a simple way. We can find an EPUB file, which can't have DRM (digital rights management) included, and change the extension to .zip. Now we will get a folder containing all of the files that comprise the EPUB book. First, we can try to manipulate the contents of one chapter file and check how it was modified using an EPUB reader application (like iBooks for Mac Users), and then modify the whole contents as expected.

This method will require a huge amount of labor and can easily lead to mistakes. That's the reason why it is not recommended or commonplace to create an EPUB file from scratch.

I used this method to create a dummy EPUB file. Figure 4 shows how it looks like when rendered by the Firefox add-on EPUBReader.

Figure 4. An example of a dummy EPUB book created from scratch

2) Convert from another format (like Word, TXT) using an application

There are a few applications, both open-source and commercial, that can help to convert your Word, ODF (OpenOffice Document File), or RTF (Rich Text Format) doc into an EPUB file[2] .

There is an application named Sigil. It is a free, open-source editor for EPUB files. However, a Word file can't be handled by Sigil directly, since Sigil can only handle basic text or HTML files. Consequently, we need another software to prepare the Word file to import into Sigil while retaining the basic formatting. If you are a Mac user, it is very easy to use Mac's pre-installed TextEdit application to open a

Word document and then save it as an HTML file. Figure 5 shows how to set up the HTML Saving Options of TextEdit.

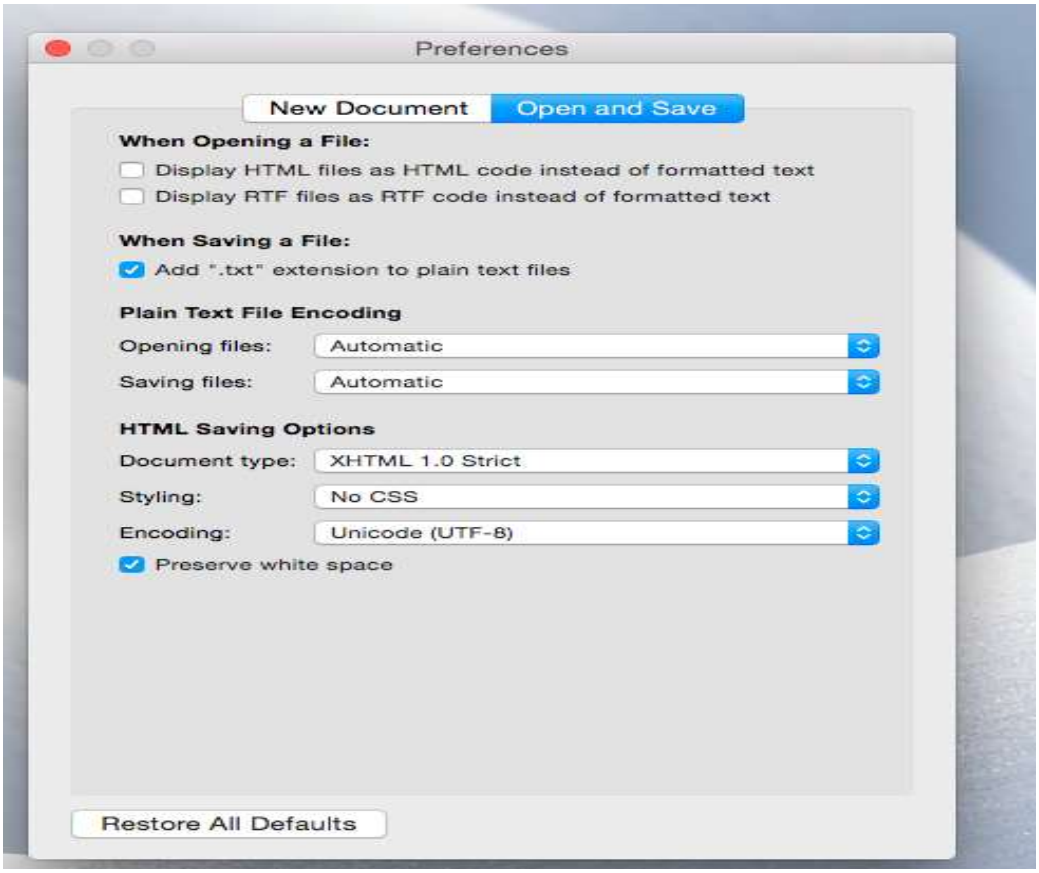

Figure 5. An example of how to set up the HTML Saving Options of TextEdit.

Or you can import the Word document into Google Docs and then save to HTML. It can then be opened in Sigil as an EPUB file. There is another option, Calibre, which is a free eBook management software that has conversion and editing capabilities. In this section, I have introduced these applications that you can utilize to convert an existing document in another format to an EPUB file. I won't describe too many details of how to do it, as you can easily search online for a topic like How to use Sigil or Calibre to get the instructions.

2.1.3 EPUB 3

EPUB3 is the latest version of the EPUB format, and is based on the latest HTML5 standard. Therefore, EPUB publication can now contain video, audio, and interactivity – just like websites in modern browsers[3]. The feature is achieved by these key technologies in EPUB 3[4]:

1) XHTML5: for representing text and multimedia content, which now has native support for MathML equations, ruby pronunciation markup, and embedded SVG images;
2) SVG 1.1: for representing graphical works, like manga and comics;
3) CSS 2.1 and 3: to facilitate visual display and rendering of content;
4) JavaScript: for interactivity and automation;
5) TrueType and WOFF: to provide font support beyond the minimal base set included in a typical reading system;
6) SSML/PLS/CSS 3 Speech: for improved text-to-speech rendering;
7) SMIL 3: for synchronizing text and audio playback;
8) RDF vocabularies: for embedding semantic information about the publication and content;
9) XML: a number of specialized grammars to facilitate the discovery and processing aspects of EPUBs;
10) ZIP: to wrap all the resources up into a single file

2.2 Existing EPUB reader

Since EPUB has become popular recently, there are now a lot of existing eBook rendering platforms and applications that support the EPUB format. In this section, I will describe them separately and analyze their drawbacks. Then I will explain why we will promote an EPUB reader web application and how it will solve the issues of the existing services.

2.2.1 iBooks and Google Play Books

iBooks is an e-book application developed by Apple for iOS and OSX operating system based devices only. With the release of iOS8, iBooks became an integrated app without extra installation. A user can open any EPUB file with iBooks, which also provides a link for the user to receive EPUB content from the iBooks Store.

Google Play Books, originally named Google eBooks, is a cross-platform eBook application developed by Google. The user must have an active Google Play account to log in, and then can purchase or download an eBook from Google Play, which offers over 5 million eBooks. The user may also upload up to 1000 PDF or EPUB books to his Google Play Books account in cloud storage and synchronize them between multiple devices. However, all those PDF or EPUB books must be purchased from Google Play Books, otherwise they will be prohibited.

2.2.2 Firefox's add-on EPUBReader

EPUBReader is Firefox's add-on. With EPUBReader installed, the user can import pre-downloaded EPUB files via Firefox's "File/Open File" dialog, and EPUBReader will render the file automatically. The EPUBReader will organize all those books in ePub-Catalog. The user can set up different tag to classify them as a

private library. The user can also click a link to an EPUB file online via the Firefox browser, which has installed EPUBReader extension. EPUBReader will then download the file and process the file for reading automatically, exactly the same way as pre-downloaded files. Figure 6 shows an example of how an EPUB book looks in the Firefox EPUBReader, and Figure 7 shows how those imported EPUB books were organized in an ePub-Catalog.

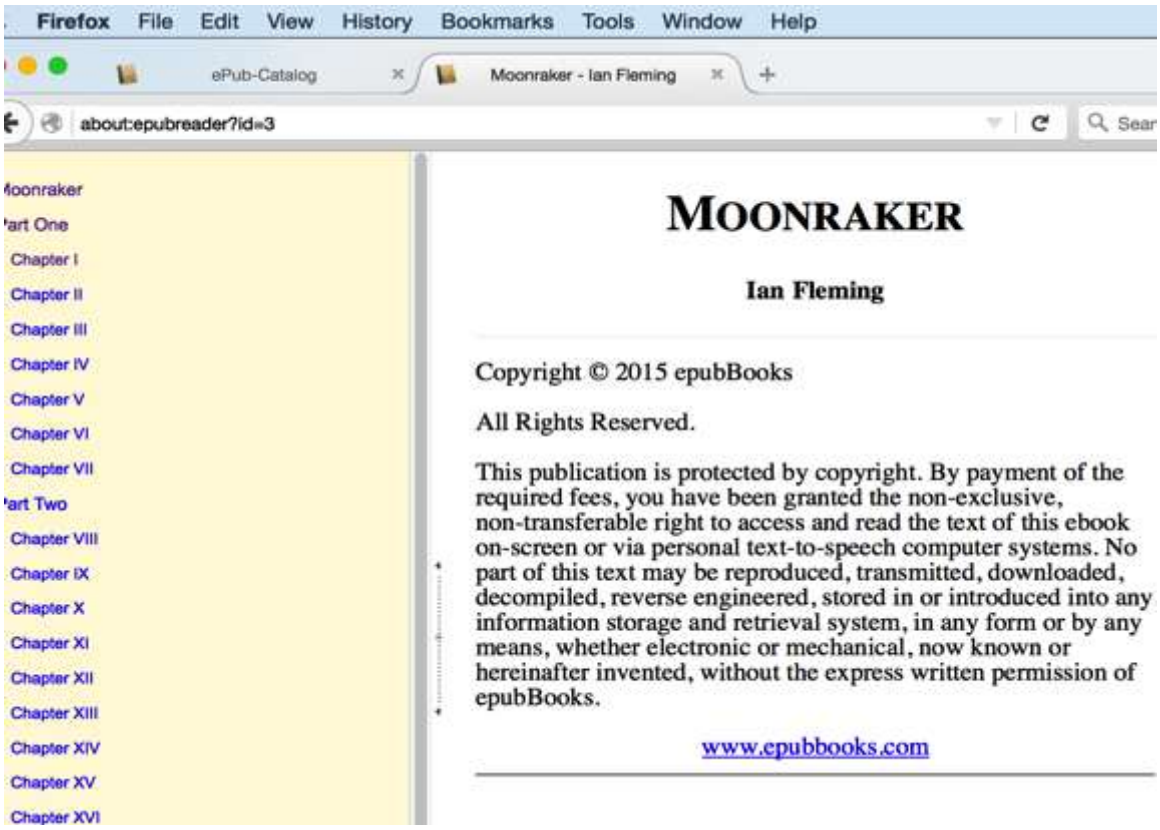

Figure 6. An example of how EPUB book looks in Firefox EPUBReader.

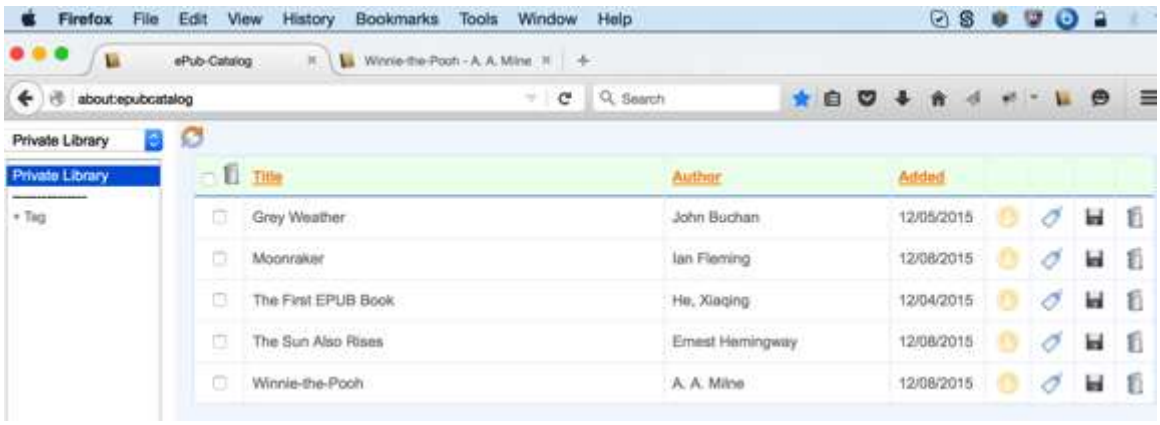

Figure 7. How those imported EPUB books were organized in an ePub-Catalog.

2.2.3 eReader devices

There are many popular eReader devices, such as Amazon Kindle, Barnes & Noble Nook, Sony Reader and so on.

Amazon Kindle is a series of e-book readers developed by Amazon. The oldest version of Kindle can't read EPUB files. The most popular format support by Kindle is MOBI. Kindle has supported EPUB since the release of the Amazon Kindle Fire.

The Barnes & Noble Nook is developed by American book retailer Barnes & Noble, and is based on Android platform. Barnes & Noble also provides a free e-reader application to allow the user to read EPUB books on devices other than Nooks.

Sony Reader is a line of e-book readers manufactured by Sony, who invented the first commercial E Ink e-book reader with the Sony Librie in 2004.

There are a few other nice eReader devices on the market, but we won't describe them here due to space limitations. For more details and comparisons between these devices, you can visit https://en.wikipedia.org/wiki/Comparison_of_e-book_readers. From the point of view of this application, all the eReader devices have the same function: that is, they all support EPUB format.

2.3 Why is an EPUB reader web application?

There are EPUB reading application like iBooks, browser extensions like EPUBReader for Firefox, and eBook reading devices like the Amazon Kindle Fire. They all support EPUB format, so why still promote an EPUB reader web application?

The reason is that those existing services have their own drawbacks. iBook functions on iOS and OSX operation systems only. Window users and Android phone users can't utilize this great application, though there are other options available for Windows and the Android platform. Firefox's add-on EPUBReader solves the issues of iBook and can be accessed by both Mac and PC users. However, it is still flawed due to its inability to easily synchronize page positions across multiple devices. A user may be reading an EPUB book via EPUBReader on device A and then change to device B. He may need to install EPUBReader if it is not installed already. And there is not any infomation on device B about which book he is reading and where he had left off. This is a big defect im many eBook reader systems. eBook devices do track where a user has been reading and provide other useful features. The device itself, however, costs money and it has its own limitations; for example, it does not support multiple users reading at the same time.

In my project, we are going to create a JavaScript and PHP EPUB reader web application that supports multiple platforms/systems and allows the user to read the EPUB book easily across multiple devices, while allowing multiple users to read the book simultaneously.

3. Preliminary Work

In this chapter, we will go through the preliminary work of the project, which includes an introduction of two compression algorithms, DEFLATE and SEQUITUR, how to implement this SEQUITUR algorithm using JavaScript, why we have chosen JSZip, and how to use it decompress a compressed .epub file. As we demonstrated in the previous chapter, an EPUB file is actually a compressed ZIP file. Since our project is to create a JavaScript EPUB reader web application, we need to understand how the file is compressed and how to decompress it, which algorithm to apply and how it works, whether we can implement one algorithm using JavaScript, which algorithm we will choose, and how to implement it using JavaScript. Lastly, I will show why we choose JSZip and which experiment we did to ensure it was good choice for our project.

3.1 DEFLATE algorithm

Compression plays an important role in computer world, and it is widely used in data processing. There are many compression methods, like GZIP and DEFLATE, that are called lossless compression, since there is no data is lost during the compression and decompression, and the decoded output is exactly the same as the original input[10]. Any other compression method will create smaller copies of the original that can't be decompressed back to the original format. For example, during JPEG compression, colors that are very similar (but not quite the same) are converted into one single color. And for MP3 compression, inaudible sounds are removed from an audio file. Even though data is lost during JPEG and MP3 compression, the image

quality is still good enough for normal printing, and the audio quality is also good for average use.

In our project, we need to decompress the compressed EPUB file and render it correctly in the web browser, after doing some research on lossless compression methods, we chose DEFLATE.

Deflate is a data compression algorithm and associated file format that uses a combination of LZ77 algorithm and Huffman codes. It was originally designed by Phil Katz for his PKZIP archiving tool. The file format was later specified in RFC 1951. Deflate compression basically consists of two parts: duplicate string elimination with LZ77 algorithm and compression though Huffman coding trees[6].

3.1.1 Huffman Coding

A Huffman code is a prefix code prepared by the Huffman algorithm[8]. Each code is a series of bits, either 0 or 1, and represents an element in a specific "alphabet" (or Huffman tree). The core of Huffman coding is: the characters most used in the original file get the least amount of bits inside the compressed file. Alphabet (or Huffman tree) is a lookup table that stores how many bits a certain character gets. There will be different alphabets for different files, or even for the same file[9].

I will explain how Huffman coding works using a simple example. Say we have an input string: 'FFDBABFCFFAFBE'. The F is the most used, so it gets the least amount of bits. The B comes second. One thing we need to pay attention to is to avoid making an overlap. Finally, we can get a Huffman tree as shown in Figure 8.

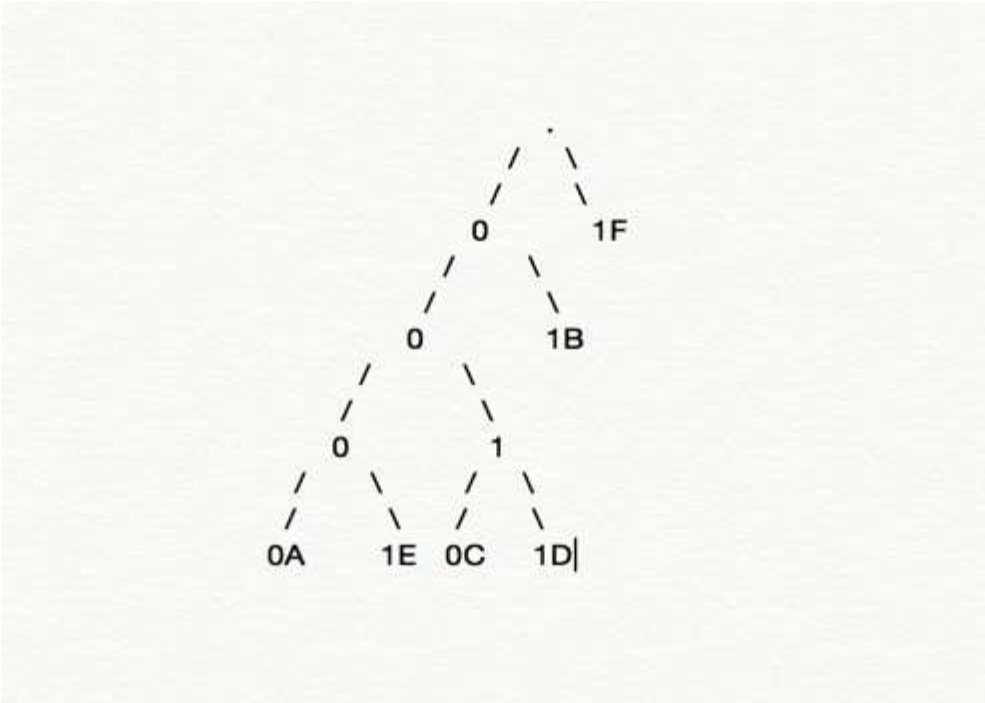

Figure 8. An example of a Huffman tree

After getting the Huffman tree, it will be easy to encode and decode of the string. Now we can compress the original string ‘FFDBABFCFFAFBE’ into:

1 1 0011 01 0000 01 1 0010 1 1 0000 1 01 0001; and, grouped into bytes:

1100 1101 | 0000 0110 | 0101 1000 | 0101 0001

C D 0 6 5 8 3 1 (hex code)

We have compressed 14 bytes string into 4 bytes. Now, we begin to decompress. The rule of decompressing is to read a bit and follow down the Huffman tree until we get to a character or the end of the tree. When on the top of a tree, we need to read the follow bit to decide which branch will continue. Then we can determine that the output of decompression is ‘FFDBABFCFFAFBE’, which is exactly the same as the original string.

3.1.2 LZ77 compression algorithm

LZ77 is an algorithm designed to compress repeated sequences of characters. It is used to analyze input data and determine how to reduce the size of that input data by replacing redundant information with metadata[7].

LZ77 algorithm uses the term “Sliding window”, which means that at any given point in the data, there is a record of what characters came before[9]. A 64K sliding window means that the compressor (and decompressor) has a record of what the last 65536 (64*1024) characters were. When the next sequence of characters to be compressed is the same as the one that can be found within the sliding window, the sequence of characters will be replaced by two numbers: a distance, representing how far back into the window the sequence starts, and a length, representing the number of characters for which the sequence is identical.

I will also explain how distance and length works using a simple example. For example, we have the following HTML code: <title>Test</title>. It can be compressed as <title>Test</[6;12]. 6 is the length and 12 is the distance. How can we decompress it? 12bytes back, the position begins at ‘t’ of <title>, and we will copy the next 6 (length) bytes since there to get the <title>Test</title>.

3.2 SEQUITUR, a grammar-based compression algorithm

The SEQUITUR algorithm was developed by Craig Nevill-Manning and Lan H. Witten in 1997. It is a method of using data compression to infer the structure of a sequence of symbols. It detects repetition and factors it out of the string by forming

rules in a grammar. The rules can be composed of non-terminals, giving rise to a hierarchy.

Here is general process of compressing with SEQUITUR[12]:

We will use three major variables, as follows:

1) An auxiliary string, Aux, that represents the results of compression process so far;
2) A hash table, Hash, which will have pairs of symbols that appear in Aux;
3) A table of productions that have been generated so far. The right-hand side of these productions will always have a length of at most 2 before the final step of this algorithm is applied.

In the beginning, all the three variables are empty by default. Let W= W1,W2,...,Wn be the string we are going to compress. The algorithm proceeds as follows:

First, for i = 1 to n: add Wi to the end of Aux and look at the last two symbols of Aux. These might be a mix of terminals and variables. If:

a) The last two symbols match one right-hand side of production in Production table, then replace in Aux these two symbols by the left-hand side of variable of that production. If the production has been used more than once, it will have a flag as marked;
b) The last two symbols match a pair of symbols in Hash, then replace the two symbols with a new variable. Add a new production to the production table consisting of this new variable going to these two symbols. Next, delete these

two symbols from Hash and add to Hash the two symbols, starting from where we replaced the earlier occurrence.

Now, we will place the last two symbols of Aux in Hash, then repeat the above two steps until nothing else can be applied.

Second, remove rules that are only used once in the grammar by compressing them. For example, if A -> BC, B -> de, and C -> fg and the last two rules were used only once, then we will delete these two rules and add the rule A -> defg.

Last, we output the grammar consisting of the rules in our production table together with a new start variable, S, that is used in the rule S -> final value of Aux.

For example, we have the input string as “abbababbabbabbabab”, Table 1 shows the state of Aux and the production table after each character is read:

Letter	Aux	Production Table	Hash
a	a	{ }	{ }
b	ab	{ }	{ab}
b	abb	{ }	{ab,bb}
a	abba	{ }	{ab,bb,ba}
b	Abbab becomes AbA	{A->ab}	{Ab,bA}
a	AbAa	{A->ab}	{Ab,bA,Aa}
b	AbAab becomes AbAA	{ A->ab marked }	{Ab,bA,AA}
b	AbAAb becomes BAB	{ A->ab marked, B->Ab }	{BA,AB}
a	BABa	{ A->ab marked, B->Ab }	{BA,AB,Ba}
b	BABab becomes BABA becomes CC	{ A->ab marked, B->Ab, C->BA }	{CC}
b	CCb	{ A->ab marked, B->Ab, C->BA }	{CC,Cb}
a	CCba	{ A->ab marked, B->Ab, C->BA }	{CC,Cb,ba}
b	CCbab becomes CCbA	{ A->ab marked, B->Ab, C->BA }	{CC,Cb,bA}
b	CCbAb becomes CCbB	{ A->ab marked, B->Ab marked, C->BA }	{CC,Cb,bB}
a	CCbBa	{ A->ab marked, B->Ab marked, C->BA }	{CC,Cb,bB,Ba}
b	CCbBab becomes CCbBA becomes CCbC	{ A->ab marked, B->Ab marked, C->BA marked }	{CC,Cb,bC}
a	CCbCa	{ A->ab marked, B->Ab marked, C->BA marked }	{CC,Cb,bC,Ca}
b	CCbCab becomes CCbCA	{ A->ab marked, B->Ab marked, C->BA marked }	{CC,Cb,bC,CA}

Table 1. An example of how to compress a string with SEQUITUR algorithm

Now, we get the final {A -> ab, B -> Ab, C -> BA, S -> CCbCA} since there is no rule that was used only once. We are going to use R to denote the start of a rule, and # followed by a sequence of digits for n to denote the n variable. In this example, the string “abbababbabbabbabab” was compressed as RabR#0bR#1#0R#2#2b#2#0. It looks like it was not too efficient here, while it works better on longer strings. For example, a string of 64 a’s can be encoded as RaaR#0#0R#1#1R#2#2R#3#3R#4#4, a string of length 28.

3.3 How to implement SEQUITUR algorithm using JavaScript

Since our purpose is to create a JavaScript EPUB reader web application, the most important issue is whether we can unzip compressed EPUB files using JavaScript. The research I am doing here is trying to implement the SEQUITUR algorithm using JavaScript. If our JavaScript code can compress and decompress correctly with SEQUITUR algorithm, then our proposal is doable.

To fulfill the compress function, we have an auxiliary string s1 which is exactly the same as Aux that was described in Sequitur algorithm, an array R[] to store the right-hand side of the Production table, array M[] to track whether a production has been used more than once, and array last2Arr[] as that Hash which was described in the previous section to track each combination of two characters of the input string. If there are only 2 characters in the input string, there is no need to apply the Sequitur algorithm, just use the original one. For a string consisting more than 2 characters, using variable C to track each character and add it to the end of the auxiliary sting S1. When getting a new last two characters of S1, which was represented as other String

variable last2char, we first check if it exists in Hash last2Arr already. If yes, we will push it to Production table R and replace the two symbols with index info of Production table in auxiliary string S1, and need to construct array last2Arr[] with the new combination of each two characters in S1. If the last2char does not exist in Hash last2Arr, we will check whether it matches anything that existed in array R[] as the right-hand side of Production table. If it matches an existing right-hand side production, we will first mark that production as true, and then replace in auxiliary string S1 those two characters by the index number of production and delete it from Hash. Furthermore, we need to deal with the new last 2 characters in auxiliary string S1 and repeat the above process. If it does not match anything in right-hand side of Production table, we will just push last2char to Hash last2Arr[].

When getting the final production array R, we need to eliminate the rules that were used only once. This step was fulfilled by checking whether M[index] is labeled as false when dealing with R[index]. If the rule was used only once, we need to replace in auxiliary string S1 the updated right-hand side production.

For the last step, we will construct the final compressed string with element in array R[]; that is, the final rules in production table.

It is relatively easy to implement the decompress function. We have the final compressed value, str, which consists with a lot of Rs as the start of a rule. We then we split the whole string by replacing it with position and rule info.

The source code of the compression and decompression functions is shown in Figure 9 and Figure 10.

```
<script type="text/javascript">
function compress() {
        var s =  document.getElementById("submit").value;
        if (s.length < 3) {document.getElementById("demo").innerHTML = s;}
        else {
           var s1 = s[0].concat(s[1]);
           var R = new Array();
           var M = new Array();
           var last2Arr = new Array();
           last2Arr[0] = s1;
           var last2char;
           var final="";
           for (var i=2;i<s.length;i++){
              var c = s[i];
              s1 =s1.concat(c);
              last2char = s1[s1.length-2].concat(s1[s1.length-1]);
              if (last2Arr.indexOf(last2char) < 0)   {

                  var index = R.indexOf(last2char);
                  if (index >= 0)
                     {
                        M[index] = true; s1 = s1.replace(last2char,index);last2Arr.pop();
                        var last2char = s1[s1.length-2].concat(s1[s1.length-1]);
                        if (last2Arr.indexOf(last2char) < 0 )
                         {last2Arr.push(last2char);}
                        else {
                         R.push(last2char);
                         var index = R.indexOf(last2char);

                         var re = new RegExp(last2char,'g');
                         s1 = s1.replace(re,index);
                         last2Arr = new Array();
                         for (var j=0;j<s1.length-1;j++)
                           {
                            last2Arr.push(s1[j].concat(s1[j+1]));
                           }
                        }
                     }
                  else {last2Arr.push(last2char); }
              }
              else {
                  R.push(last2char);
                  var index1 =  R.indexOf(last2char);

                  var re = new RegExp(last2char,'g');
                  s1 = s1.replace(re,index1);

                  last2Arr = new Array();
                  for (var j=0;j<s1.length-1;j++)
                     {
                        last2Arr.push(s1[j].concat(s1[j+1]));
                     }
               }
           }
      for (var i=0; i < R.length; i++){
        if (M[i] != true) {var re = new RegExp(i,'g'); s1 = s1.replace(re,R[i]); R[i] = i;}
      }

      for (var i=0;i<R.length;i++){
       final = final.concat("R" + R[i]);
      }
      final = final.concat("R" + s1);
      document.getElementById("demo").innerHTML = final;
    }
 }
```

Figure 9. Compress function with SEQUITUR

```
function decompress(){
    var str = document.getElementById("submit_again").value;
    str = str.substr(1);
    var res = str.split("R");
    for (var j=res.length-2;j>=0;j--){
      if (res[j] != j){var re = new RegExp(j,'g');res[res.length-1] = res[res.length-1].replace(re,res[j]);}
    }
   document.getElementById("origin").innerHTML = res[res.length-1];
}
```

Figure 10. Decompress function with SEQUITUR

3.4 Why did we choose JSZip?

Since have implemented Sequitur algorithm successfully in JavaScript. We are pretty confident that an EPUB file can be rendered in web browsers using JavaScript. While it may cost a lot of time and effort to create our own JavaScript code to Zip/Unzip, there may be some JavaScript library that has provided this feature already. After doing some research, we found JSZip on GitHub.

JSZip is a JavaScript library for creating, reading, and editing .zip files. In our project, we focused on how to use JSZip to read an EPUB book as a zip, especially how to get the binary data of that Zip file with an AJAX request from the browser, and then how to access the file content in the browser. From the examples shown in JSZip document, there is a collection of cross-browser utilities to go along with JSZip, JSZipUtils , that we can apply. To make sure it works well, we need to include both JSZip and JSZipUtils libraries in the source code.

First, we will utilize JSZipUtils to get the binary data of an EPUB book on the server with an AJAX request in the browser. We need to provide the correct file path; otherwise JSZipUtils can’t load the contents.

Then we make an instance of JSZip, which consists of all the files included in that EPUB file from which we just got binary data with a AJAX request. On this instance, we can update (add, remove, and modify) files and folders with .file(name, content) and .folder(name). They return the current JSZip instance as an output of JSZip(data).

In our application, we are just going to show them properly in the browser, so we will access the file content with .file(name) and its getters. We are using one getter

method asText(). It returns the content as a Unicode string. After that, we just do a normal front-end operation in JavaScript, as we expected.

There are some limitations of JSZip. One is that JSZip only supports UTF8. If the names of the files inside the Zip file are not in UTF8 (or ASCII), they won't be interpreted correctly. If the content is a text not encoded with UTF8 (or ASCII), the getter method .asText() won't decode it correctly either. And there are some performance issues that come from the browser. A compressed zip file of 10MB will be "easily" opened by Firefox/Chrome/Opera/IE10+, but will cause a crash in an older version of IE. So we suggest not using IE <= 9 when using JSZip.

Those limitations may be improved in the future version, or there may be another improved JavaScript library available to render Zip/EPUB files.

4. Core of the project

In previous sections, we described the background of this project and what we completed in preliminary work. In this chapter, I will present the core of the project: that is, how we design, implement, and test the JavaScript and PHP EPUB reader web application. First, I will introduce the goal of this project and its requirements. Secondly, I will go through the architecture design and framework options. Third, I will list all supported features and discuss features to improve. Last, I will show several experiments we performed to check this application's design and functionality.

4.1 Goal and Requirements

Our proposal is to create a web application that allows users to read an EPUB format book easily across multiple devices. To fulfill this feature, we need a database to store the user's account info, which book he is reading, and where he has been reading. Furthermore, we will provide a library service and bookshelf functionality.

In our application, a user can create an account. A registered user can borrow books from a library after login and then save them into his bookshelf. The user can only read books in his bookshelf, and the application will track his reading and save his last reading position in each book to enable him to easily pick up where he left off.

4.1.1 Database Architecture and Development Approach

We will use typical server-client architecture for this project. A server is normally a powerful computer or process dedicated to managing disk drives (file

servers), printers (print servers), or network traffic (network servers). A client can be any PC, smartphone, tablet, or workstation on which users can run an application[13].

LAMP (the Linux operation system, the Apache HTTP Server, the MySQL database and the PHP programming language) is a popular solution stack for building web applications. In this project, we are going to use XAMPP (version 5.6.3), a free and open source cross-platform web server solution stack. The X stands for cross-platform. It works equally well on Linux, Solaris, Windows, and Max OSX. A stands for Apache HTTP Server, M means MySQL database and PP means it can interpret scripts written in both the PHP and Perl programming languages[14]. It's not easy to install an Apache web server, and it gets more complicated when adding MySQL, PHP, and Perl. With XAMPP, it is easy to install Apache distribution containing MySQL, PHP, and Perl.

We chose the most popular open source MySQL (version 5.5.30) as our database option. For this application, we need the database to store user info, books/library info, and user-book info, which will track who are reading which book and the last page they were reading. We will use PHP for our server side development.

4.1.2 JavaScript JSZip & jQuery

We have implemented the SEQUITUR algorithm using JavaScript, ensuring us that we can write code in JavaScript to Zip/Unzip files. In our application, we will utilize the JSZip, a JavaScript library for creating, reading, and editing .zip files, to unzip the EPUB file with an AJAX request from the browser. Then, we will write functions in JavaScript to parse metadata and display the EPUB book's contents in the browser. Most importantly, we need to implement a feature that will send the user's

reading information to the server periodically and automatically. We will explain more details from a programming point of view as follows:

1. Using JSZip to unzip and read the EPUB book

   With the help of JSZip, not a lot of coding is required on our part. However, we spent a good deal of effort making JSZip and its utilities, JSZipUtils, work well with our code. The documentation of how to use JSZip is not clear enough and it is difficult to debug when something goes wrong.

   Getting binary data with an AJAX request in the browser is hard, but with JSZipUtils.getBinaryContent, we can achieve it. Figure 11 shows an example of how to use JSZipUtils.getBinaryContent.

```
JSZipUtils.getBinaryContent('path/to/content.zip', function(err, data) {
  if(err) {
    throw err; // or handle err
  }

  var zip = new JSZip(data);
});
```

Figure 11. An example of how to use JSZipUtils.getBinaryContent.

   In the callback function, we make an instance of JSZip with response data that includes all the files of the Zip package that are specified in the 'path/to/content.zip'. With this instance, we can update files and folders. We show an example of how to get file content with a JSZip instance in Figure 12.

```
var new_zip = new JSZip();
// more files !
new_zip.load(content);

// you now have every files contained in the loaded zip
new_zip.file("hello.txt").asText(); // "Hello World\n"
```

Figure 12. An example of how to get file content with a JSZip instance.

2. Using JavaScript to parse the metadata and display the pages

When getting the file contents in the browser, we need to write JavaScript code to parse the metadata and define how to display book contents properly in the browser. An EPUB book is like a website, and its pages are stored as .html file, so it is straightforward to display the pages in the browser. We will display table of contents also by parsing toc.ncx or the equivalent file. Furthermore, we implemented an on-click function when a user clicks any chapter in the table of contents, the page will show the contents accordingly.

3. Using jQuery to load data from the server using an HTTP Post request

jQuery is a cross-platform JavaScript library and is one of the most popular JavaScript libraries in use today. It makes things like HTML document traversal and manipulation, event handling, animation, and AJAX much simpler with an easy-to-use API that works across multiple browsers. In our application, we did not fully utilize its powerful functions. We just used jQuery to make an AJAX call and send JSON data to the server via an HTTP Post request. This step enables our application to track a user's data (user info, book info, chapter info and page info). The snippet code used to update the user's reading data periodically via jQuerey.post() is in Figure 13.

```
setInterval(function(){
  var data = {
     title : titleInfo,
    chap : new_chapInfo,
    page : window.pageYOffset
   };
  $.post("./?c=book", data, function(json){alert(json);}, "json");
},3000);
```

Figure 13. The snippet code used to update the user's reading data periodically via jQuerey.post().

4.1.3 PHP MVC framework

MVC (Model-View-Controller) is a very popular software design pattern for developing web applications. A Model View Controller pattern consists of following three parts[16]:

- Model - The lowest level of the pattern that is responsible for maintaining data.
- View - This is responsible for displaying all or a portion of the data to the user.
- Controller - Software code that controls the interactions between the Model and View.

In the MVC pattern, the application is broken down into three coherent parts: the Model, or Data, the View, or User Interface layer, and the Controller, which mediates user interface interactions and updates to Model. I will show the MVC process clearly in Figure 14.

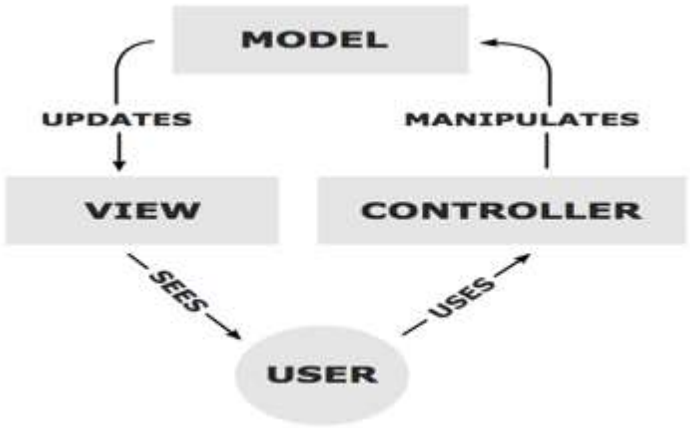

Figure 14. MVC process

There are many popular PHP MVC frameworks, like CakePHP, Symfony, Laravel, Zend, and so on. However, we will create our own PHP MVC framework in this project, just following the basic concepts.

We have specified only one entry point for the whole application as index.php. In index.php, we control the workflow based on the different $_REQUEST. To implement the application's features, we need to specify each feature in controller, model, and view. For example, we provide a feature to let a user register in our system. Then, we need a controller file (controller/register.php) which defines the workflow of register process, like how to deal with when a user did not fill in all the must-have info, how to retrieve data in the view and store it in the database via model file, and so on. A model file (model/register.php) will take care of all the operations within database, like checking whether a username was used already, inserting user information into the database, and so on; a view file (view/register_view.php) will just display the metadata following the HTML and CSS format, which was defined by itself.

4.2 Supported Features

In this section, I will describe the main functionalities of this application. This new EPUB reader application enables a user to, 1) read any available EPUB books without any specific platform/device or pre-installed add-ons, 2) track a user's reading page automatically for easily pick up form where he left off, 3) borrow free books from the EPUB book library and manage books in bookshelf. Lastly, I will list some features that need to be implemented or improved upon in the future.

4.2.1 Read EPUB book easily without any extra effort

This new EPUB reader web application is very user friendly. Like any other EPUB reader system, a user can read an EPUB format book in this application. The user does not need to pre-install anything or rely on any specific platform or device to access the books. After the user properly creates an account, he can borrow a book from the library, save it to his bookshelf, and begin to read.

4.2.2 Track the page position automatically

This is the nicest feature of the application. It serves as a bookmark without requiring a user's effort. That is, a user does not need to place it intentionally. During the reading process, the JavaScript code in the client side will communicate with the PHP script on the server side periodically to update the page position and store it in the database. Once the user logs in, the system will retrieve his latest reading record from the database and the server will send this response to the client. Eventually, it will display to the user the exact same page he left off at during last reading.

4.2.3 Library service & Bookshelf functionality

In this application, user can create an account and his username, password and some other registration info will be saved into database. When a user logs in, the system will check the legality of the user. After a successful login, the user now can browse all the available books and borrow any or all of them and save them into his own bookshelf.

The major issue here is not a technical issue, but rather a copyright issue. Where can we get so many EPUB books and then provide them to users to borrow for free?

Google Play Books offers over 5 million eBooks based on Google's powerful resources. In our case, this is not an issue we will try to solve in this project.

4.2.4 Features to be implemented or improved

Currently, this application just works in a basic way. There are a lot of features that can be implemented or improved.

For example, support for uploading a book. Before we do that, we need to assign a role for each user in the application. Basically, it will be guest, registered user, and admin. A guest can view a book without registration, but can not save the book to his bookshelf. None of his reading info will be tracked by the application. A registered user can borrow a book from the library to save into his bookshelf or upload books to his bookshelf. He can classify books of his bookshelf as public, which serves a similar purpose as the library, or label a book as private. Here we need to provide a search function for the user to easily search across both the library and his own bookshelf. Having more users sharing books will help solve the issue of limited books in this project.

Furthermore, as a web application, the user experience is very important. A user friendly UI is crucial. This prospect is still in the early beta version and it needs lots of effort to improve the UI design.

4.3 Experiment

In this section, we will show several experiments we performed to check this application's design and functionality.

4.3.1 Testing process

We designed the testing tasks as follows:

1) Whether a user can register an account.
2) Whether a user can log in.
3) After login, whether the book shows properly.

   Here we have different scenarios: for a user's first time log in, the page is empty, and there is a link to the library page. Then, the user can borrow a book from there and begin to read; for a user with an existing reading log in this system, the page should be exactly the same as where he left off last time.
4) Whether a user can borrow a book from a library and the book will shown up in his bookshelf.
5) Whether the recent book feature was updated during the reading.

And we had a group of 3 people execute testing in 3 rounds.

- Round 1: testing on Chrome, Firefox, and Safari for Mac OSX
- Round 2: testing on Chrome, Firefox, and IE for Windows
- Round 3: testing on iPhone, iPad, and Android Phone

4.3.2 Testing results & feedback

Testing results from round 1 and 2 are quite positive, and all those testing tasks were checked and fulfilled. In round 3, since our web application is not targeted for mobile devices, the user experience is not good. Figure 15-20 show those experiments we had done on different browsers for different operation systems and show those tests we had done on different smart phones and tablets.

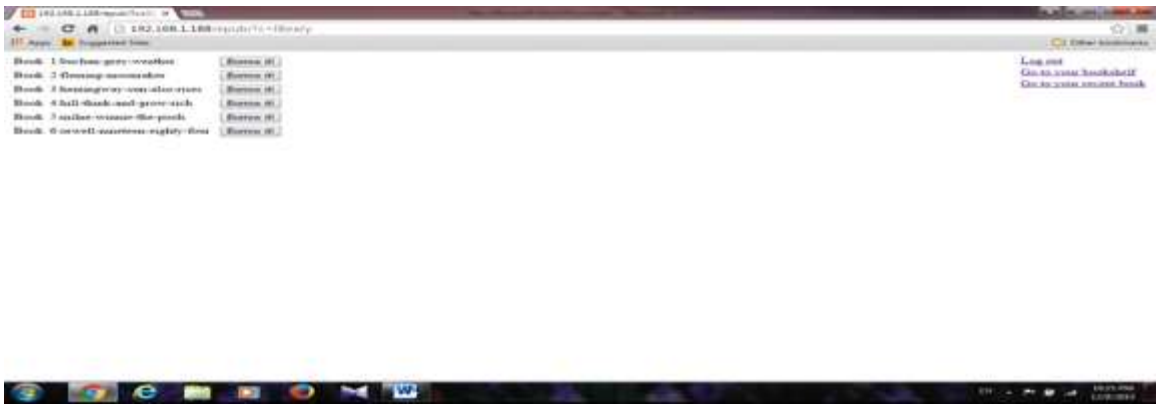

Figure 15. Testing EPUB reader on Chrome for Windows

Figure 16. Testing EPUB reader on IE for Windows

Figure 17. Testing EPUB reader on Firefox for Mac OSX

Figure 18. Testing EPUB reader on iPhone

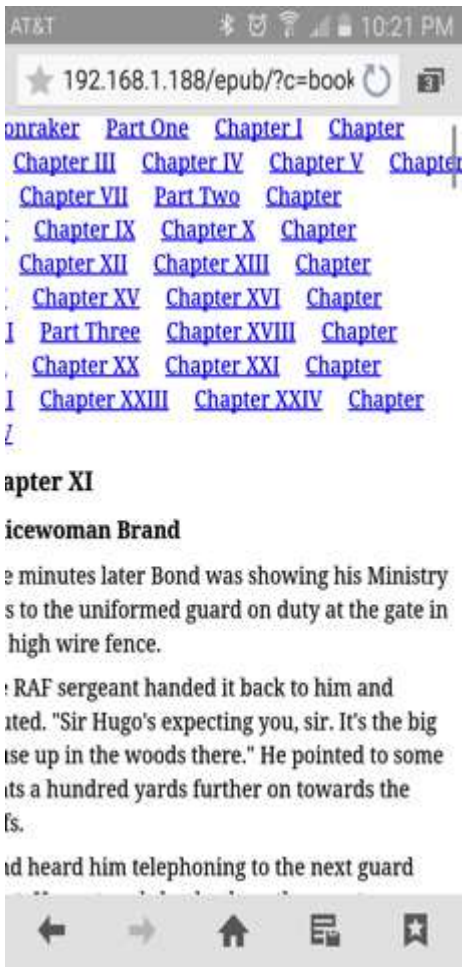

Figure 19. Testing EPUB reader on iPad Figure 20. Testing EPUB reader on Android phone

Overall, the design and functionality are acceptable, while all the feedback is to focus on the UI design. This part definitely needs to be improved.

5. Conclusion and Future Improvement

This paper contains the motivation, theoretical foundation, design pattern, detailed implementation, and testing process for our EPUB reader web application. We started our project doing research on EPUB. After learning that EPUB is just a Zip package, we began to do research on how the compression/decompression algorithm works and implemented the SEQUITUR algorithm successfully in JavaScript. When all preliminary work had been done, we were confident that we could create such a web application and focus on design and implementation. We went though many issues during the process. By solving each of them, we expanded our knowledge of compression algorithms, front-end technologies, and server-side programming.

This project aims to create an EPUB reader web application that supports a user to read EPUB format book easily across multiple devices. During user acceptance testing, this feature was implemented fully. However, there was a lot of feedback that focused on the user experience. In a web application, user experience is very important, and there are many features that need to be implemented and improved in the future release.

The application is only a beta version, and there is much room for improvement, especially in the UI design. In the future of this project, it may expand into an eBook community. It needs to provide more books for people to borrow, read, and share comments. Increasing the number of books is a big issue due to copyright protection. In that scenario, providing the upload feature is key, but may involve other issues like file management, role design and so on.

6. References

[1] "EPUB". [Online]. Retrieved from: https://en.wikipedia.org/wiki/EPUB

[2] Kudler, David. (July 10 2015). "4 Ways to Create an ePub eBook". Retrieved from: http://www.thebookdesigner.com/2015/07/4-ways-to-create-an-epub-ebook/

[3] "Understanding EPUB 3". [Online]. Retrieved from: http://epubzone.org/epub-3-overview/understanding-epub-3

[4] Garrish, Matt. (September 1 2011). "What is EPUB 3". O'Reilly Media, Inc. Retrieved from: http://www.oreilly.de/german/freebooks/epub3/What_Is_EPUB_3_.pdf

[5] Davies, Emma. "Creating and formatting documents for e-readers using ePub". Retrieved from: http://www2.le.ac.uk/projects/oer/oers/beyond-distance-research-alliance/creating-and-formatting-documents-for-e-readers-using-epub-a-guide

[6] "DEFLATE on Wikipedia". [Online]. Retrieved from: http://en.wikipedia.org/wiki/DEFLATE

[7] "LZ77 compression algorithm". [Online]. Retrieved from: https://msdn.microsoft.com/en-us/library/ee916854.aspx

[8] "Huffman coding". [Online]. Retrieved from: https://en.wikipedia.org/wiki/Huffman_coding

[9] Feldspar, Antaeus. (August 23 1997). "An Explanation of the Deflate Algorithm". Retrieved from: http://www.zlib.net/feldspar.html

[10] Thijssen, Joshua. (June 02 2010). "Deflating the universe". Retrieved from: https://www.adayinthelifeof.nl/2010/06/02/deflating-the-universe/

[11] "How does the DEFLATE compression algorithm work". [Online]. Retrieved from: https://www.quora.com/How-does-the-DEFLATE-compression-algorithm-work

[12] Pollett, Chris. (March 18 2013). "Grammar-based compression algorithm: SEQUITUR". [Online]. Retrieved from: http://www.cs.sjsu.edu/faculty/pollett/154.13.13s/Lec18032013.html

[13] "Client-server model". [Online]. Retrieved from: https://en.wikipedia.org/wiki/Client%E2%80%93server_model

[14] "XAMPP". [Online]. Retrieved from: https://en.wikipedia.org/wiki/XAMPP

[15] "Entity-relational model on Wikipedia". [Online]. Retrieved from: http://en.wikipedia.org/wiki/Entity%E2%80%93relationship_model

[16] Falkman, Drew. (March 16 2015). "MVC Frameworks for Building PHP Web Applications". Retrieved from: http://www.lynda.com/CakePHP-tutorials/MVC-Frameworks-Building-PHP-Web-Applications/315196-2.html