BYO-EPUB: an E-Book Builder

By Franklin True Martin

***Overview*—Create an Android application that is able to parse and build an e-book out of a publicly available web novel. This goal has been met and the application is in a working state. The application also serves as an e-reader, allowing users to read the e-books that they build. While the application is currently able to parse web novels, there is more that needs to be done. This report should serve as an introduction into the work that has been completed and provide the reader with an understanding of the goals and accomplishments of the project's team. This project was a culmination of open-source works and new ideas. The open-source pieces that were sourced can be found in the reference section below.**

The source code can be found here:
https://github.com/TrueFMartin/BYO-EPUB

I. Introduction

This report details the development of a mobile application known as BYO-EPUB; a unique tool designed to cater to the needs of web novel enthusiasts. The application's main goal is to simplify the transition from web novels to e-books. It uniquely offers two key features: converting web novels into .epub format and functioning as an e-book reader, providing users with a cohesive experience.

The need for such an application arises from the common issues faced by web novel readers, especially those who prefer mobile platforms. Traditional web novel reading often requires internet connectivity, lacks efficient progress tracking, and is plagued by intrusive advertisements. Current mobile apps lack the ability to convert web novels into e-books, a major limitation for those who favor traditional e-book formats. Additionally, manually changing web novels into e-books is a tedious and slow process, especially for frequently updated novels.

BYO-EPUB addresses these challenges by offering a convenient, integrated solution. Users can input the URL of a web novel's table of contents, and the application efficiently parses the content, converts it into an .epub file, and allows for immediate reading within the same application. This approach not only saves time but also enhances the reading experience by offering the functionalities of an e-reader, such as bookmarking and progress tracking.

The development of BYO-EPUB showcases the potential of integrating parsed and packaged web content with mobile applications, thereby enhancing user experience and accessibility. This report will cover the various aspects of BYO-EPUB's development, including its design, implementation, and the unique challenges overcome during its creation.

II. Related Work

The digital reading landscape is populated with a variety of e-reader applications, each offering unique features and functionalities. Among these, the Amazon Kindle app stands out as a prominent choice, allowing users to both purchase and read e-books within a single platform. Kindle's popularity stems from its seamless integration of an extensive e-book store with a highly functional reading interface. Other notable e-reader apps include ReadEra, eBoox, Libby, and FBReader, each catering to different user preferences. ReadEra is appreciated for its user-friendly interface and compatibility with various file formats. eBoox distinguishes itself with a clean, ad-free reading experience, although it offers fewer features compared to others. Libby, powered by OverDrive, is unique in providing access to digital books from local libraries. FBReader is a versatile option, supporting a wide range of formats and offering customizable reading experiences. Most of these e-reader apps are free, but some offer additional features through in-app purchases or subscriptions.

A common limitation of these apps is their focus on reading pre-existing .epub files, rather than creating them. For web novel enthusiasts, there are specialized apps like GoodNovel, Babel Novel, WebNovel, and Light Novel. These apps reformat web novels to mimic traditional e-reader interfaces, enhancing the reading experience. GoodNovel offers a vast collection of web novels, but its user interface can be cluttered with advertisements. Babel Novel and WebNovel provide a more immersive reading experience but require a constant internet connection. Light Novel, while offering a diverse library, sometimes struggles with maintaining user progress across sessions. The reliance on internet connectivity is a significant drawback for these web novel apps, limiting offline accessibility. These web novel apps often face challenges in chapter management and maintaining a smooth user experience. Advertisements are a common issue, often disrupting the reading flow in many of these applications.

In contrast, WebToEpub, a Chrome extension, focuses on converting web novels into .epub files. WebToEpub's open-source nature allows for a collaborative approach in developing site-specific parsers. This extension has gained popularity for its ability to transform web content into a more readable e-book format. However, WebToEpub's functionality is restricted to desktop browsers, lacking a mobile counterpart.

My application, BYO-EPUB, bridges this gap by incorporating WebToEpub's parser logic into a mobile-friendly format. BYO-EPUB stands out as a unique solution in the mobile domain, focusing on creating e-books while also offering the option to read .epub files. Unlike other apps, BYO-EPUB allows users to convert web novels directly on their mobile devices. This capability addresses the need for a more integrated and efficient approach to reading web novels on the go. BYO-EPUB's innovation lies in its ability to streamline the conversion process, making it user-friendly and accessible. The absence of other mobile-specific e-book packing applications highlights the uniqueness of BYO-EPUB in the market. BYO-EPUB thus represents a significant advancement in the realm of digital reading, particularly for web novel enthusiasts. Its development reflects a growing trend towards more versatile and comprehensive reading applications in the digital age.

III. Application Design

Our application employs the Model View ViewModel (MVVM) architecture along with SQLite for database management. This architecture is effectively used for handling the Wikipedia API and API calls, which are essential for storing and retrieving highlighted words, underlined phrases, and tracking the user's last read location in a book.

The application is structured into two main parts: the Framework and the Activities. The Framework contains all the logic and fragments for displaying, searching, and building Epub books, in addition to managing web calls and database operations. It does not include any activities. The Activities section, however, maintains a singleton instance of the Framework. It uses this instance to open books, display fragments, and manage all Model-related operations.

This structure not only makes the app more functional but also allows for easy integration of our Framework into other applications. Another project would be able to import our framework, and then build their own e-book reader needing to implement the e-book displayer, searcher, highlighter, parser, opener, or packer. They would instead only need to build the front end to host the framework.

A major focus of my work was on parsing web novels. When a user submits a URL, the app begins by parsing the Table of Contents of the requested page, extracting the title, author, and chapter links. This is the starting point for building an e-book.

For efficient parsing, our app manages multiple chapters in parallel. Without concurrency, the user may have to wait for hundreds of HTTP request calls to be done serially. The parsed chapters are then added to the e-book, which is subsequently written to a location chosen by the user using the systems File Picker.

The parser uses the factory pattern. It accepts a URL as input and returns the constructor of the corresponding Parser class. For example, given "https://wanderinginn.com/table-of-contents", the parser identifies the host name – *wanderinginn.com* – and retrieves the appropriate constructor for the Wandering Inn Parser. This method provides the necessary flexibility and scalability for handling different web novel sources.

Each web novel site requires a distinct parser, designed to accommodate its unique DOM structure. Currently, parsers have been developed for the top 10 web novel sites, plus a personal favorite. The process involves transforming the HTML of web novels into well-formatted XHTML, a task complicated by the diverse HTML styles used across different sites. This is the reason each site requires its own parser implementation.

Each parser in our app extends a base parser class that defines general interface methods for parsing web novels. Individual parsers then implement these methods with site-specific logic. For example, The Wandering Inn's parser targets the DOM class “entry-content” for chapter body content.

A significant part of the parsing process involves removing unwanted content. Since web novels are typically hosted in HTML format, converting them into the XHTML format used in ebooks is challenging, especially considering the unconventional HTML programming on some sites.

To aid each parser class, we have the Utils class. This class contains 99 methods that assist in content parsing. These methods range from manipulating DOM elements to removing HTML comments, which are crucial for converting content into XHTML without errors.

This project leverages several open-source resources and libraries, including FolioReader, JSoup, Epub4J and WebToEpub. However, integrating these resources required significant modifications.

The app is built on the FolioReader framework, which provides the basic structure of an e-reader without UI elements. FolioReader provides the ability to view and traverse e-books. However, it lacked any of the implementation needed to allow a user to select an e-book, modify settings, or open an e-book from internal storage.

The WebToEpub Chrome extension, used for building epub books, was a significant reference in developing our parser framework. While it is not a direct dependency, we modeled our e-book builder on the implementation of its parser factory pattern. This involved converting over 4000 lines of code from JavaScript to Java, trying to account for the differences between the Android Framework and the browser specific Chrome Extension Framework.

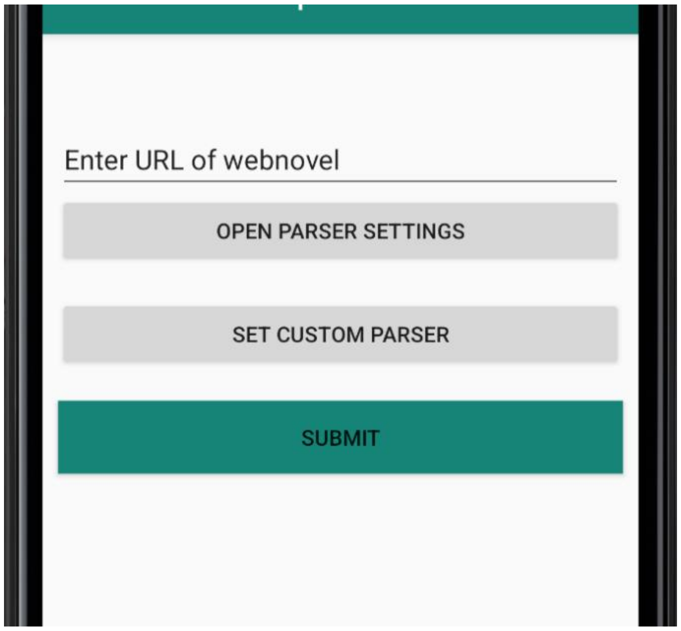
We also encountered compatibility challenges with Epub4J, a Java tool for packing XHTML files into an epub file, and our FolioReader framework. Adjustments were necessary to ensure compatibility between Epub4J and FolioReader's ebook parser. Epub4J's XHTML tag prefix in the ebook container file was not compatible with FolioReader's parsing requirements, leading me to modify Epub4J's source code.

The primary issue was Epub4J's tendency to prefix DOM tag names with ‘odf’. This would lead to a tag like: `<odf:package>`, instead of the standard `<package>`. This caused a NullPointerException when FolioReader would parse a packaged .epub file looking for the publication data within the `package` tag. After evaluating the options, I decided to modify

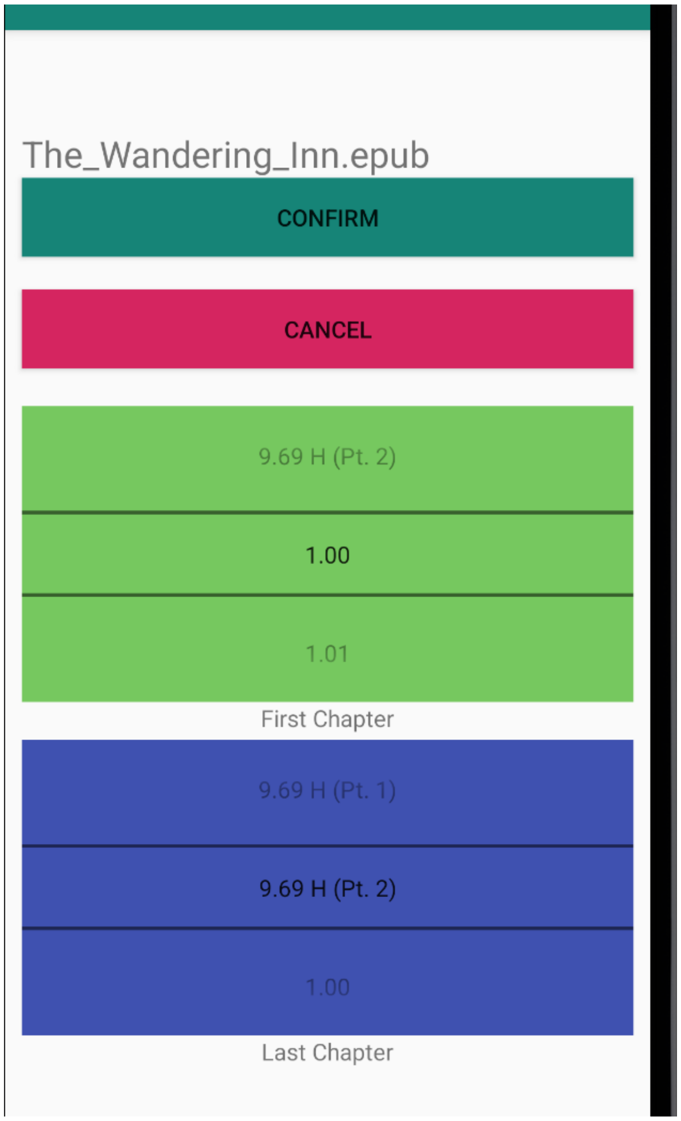
Epub4J's source code instead of FolioReader's parser. This involved cloning the Epub4J repository, identifying each location that Epub4J incorrectly prefixed a tag namespace, and making the necessary changes in multiple separate locations.

However, integrating the modified Epub4J into our application posed another challenge, as Android Studio initially rejected the custom jar file. After several attempts and exploring various alternatives, I had to directly incorporate Epub4J's source code into our library. This required adjusting every import statement in Epub4J's code and resolving dependency conflicts between libraries used by both FolioReader and Epub4J. Once these issues were resolved, the two libraries could be used together effectively.

When the user wishes to build their own e-book, they must first provide a URL starting point. This is done in the Build Book Setup Activity, where they could also add specific parser settings. This can be seen in the image below.



Once the user hits submit, a coroutine is spawned that will request the provided webpage. The parsing process starts with the Table of Contents (TOC), where we extract the web novel's title, author, and chapter links. Each chapter contains a chapter title, a chapter URL where it's content can be found, and whether it is a part of an *arc*. Once all of these are gathered, they are offered to the user. Users are then given the option to select chapters for their ebook, allowing them to customize the content according to their preferences. This can be seen in the next image.

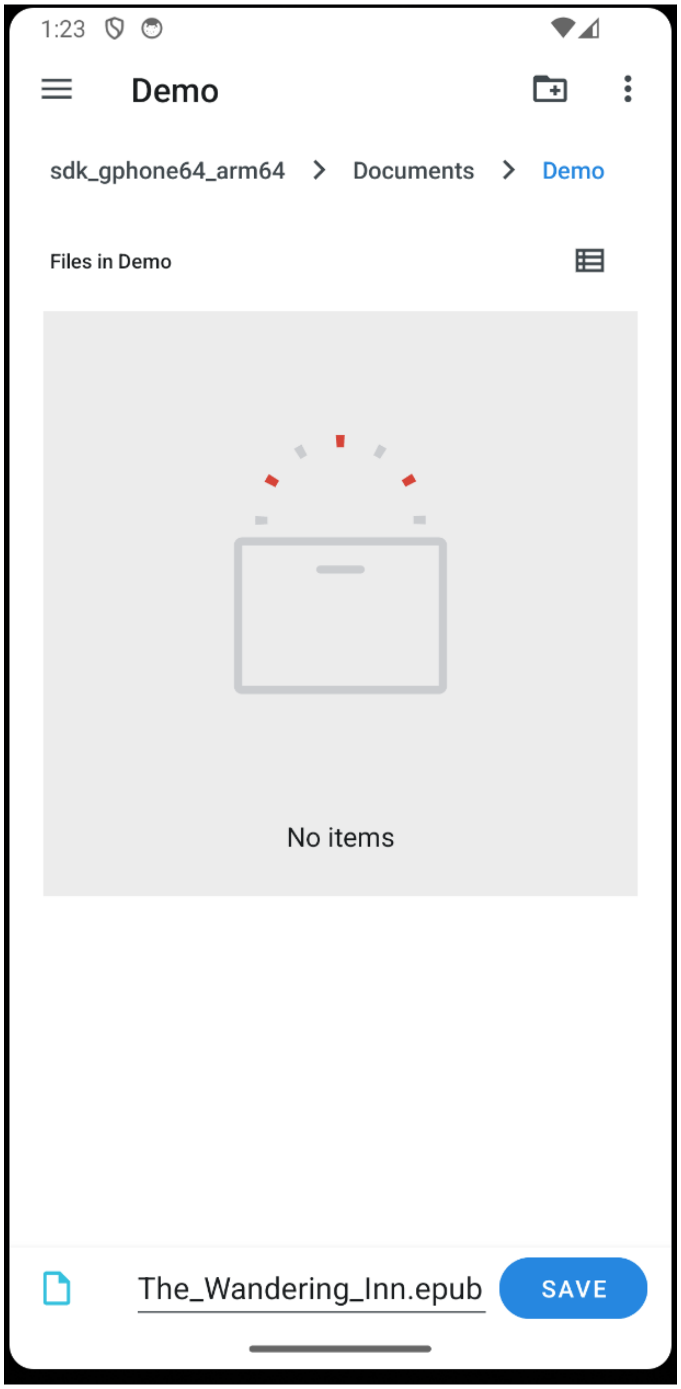


The image above shows the second phase of the Confirm New Book Activity. The first phase has every view greyed out except the Cancel Button and a spinning Progress Bar that disappears once the TOC is parsed.

In the image above, the user is presented with two Number Picker views. Each one displays an identical list of chapter titles. The first Number Picker, in light green, is used to select the initial chapter to start parsing. The second, in blue, selects the last chapter to parse. When the screen is first brought up the green picker is set to the first chapter available and the blue picker to the last chapter. From there, the user can narrow down the selection if they wish. This is useful for web novels with regular release schedules. The user can open the app when a new chapter is released and instead of having to download the entire book, they can download only the newest chapter.

Another thing to note in the activity above is the file name. In the image it is “The_Wandering_Inn.epub”. This name is based on the title parsed from the TOC. When the user chooses a file location, this is the initial filename that is offered to the user. It also serves to show the user that the parsing was successful. Otherwise, the name would display an error or “Still Parsing…” if the activity was in the first phase.

When the user clicks the Confirm Button, a system File Picker activity is started. This is system dependent but looks like the image below.



This activity is started using the intent action, ACTI-ON_CREATE_DOCUMENT. This allows users to choose a storage location through an implicit intent and the system’s File Picker. They create an initial empty file, assign it a name, and the URI of the file is returned as intent data.

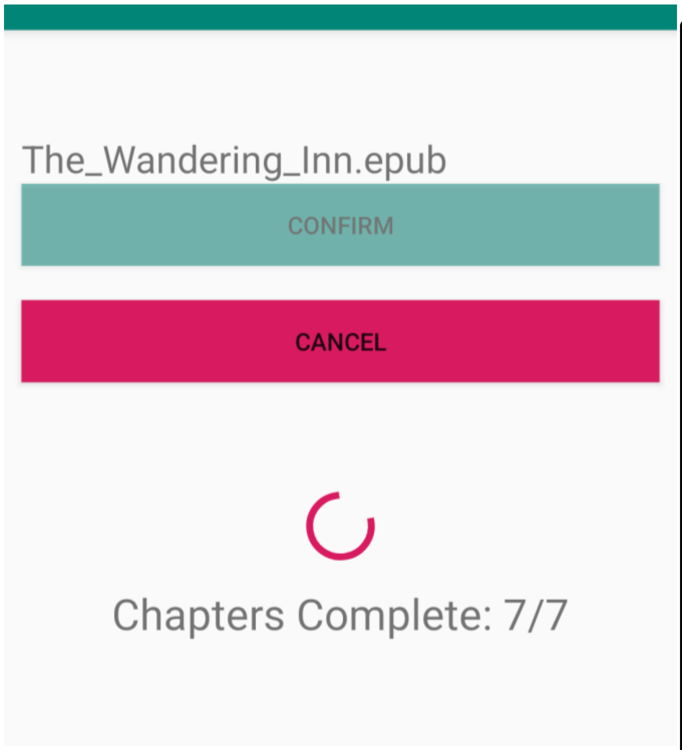
We receive this intent in the activity result listener and use the URI to later create a file output stream for writing the ebook. The user thus selects the chapters and the location for the ebook, and we collect URL addresses to those chapter locations and the URI pointing to the file.

A new thread is started which will handle the parsing of the chapters. It creates an ExecutorService with a fixed thread pool based on the number of available processors and the number of chapters to parse, whichever is less. This is also where the first Epub4J book object is created.

A list of callable tasks is created, each of which returns a JSoup Element (the HTML body of a chapter). Each task has the job of making an HTTP request to the chapter URL, running site parser specific logic, removing all unneeded HTML elements, converting each HTML tag into a tag that has an XHTML equivalent, and then finally returning the clean JSoup Element in the form of a Future.

Back on the thread that created the task, all tasks are invoked at once by the ExecutorService. This batches out as many tasks as it can at one time. When invoked, it returns a list of Future objects. Each Future holds either the returned value of the task, or the current state of the task if it is not finished or has thrown an exception.

When a task is successful, the returned JSoup Element ins converted into XHTML and then added to the Epub4J book object as a chapter. Each time a chapter is added to the book, a call back is performed which updates the UI on a main thread coroutine. This update is intended to let the user know the progress of the chapters that have been parsed. This stage of the Confirm New Book Activity can be seen in the next image.



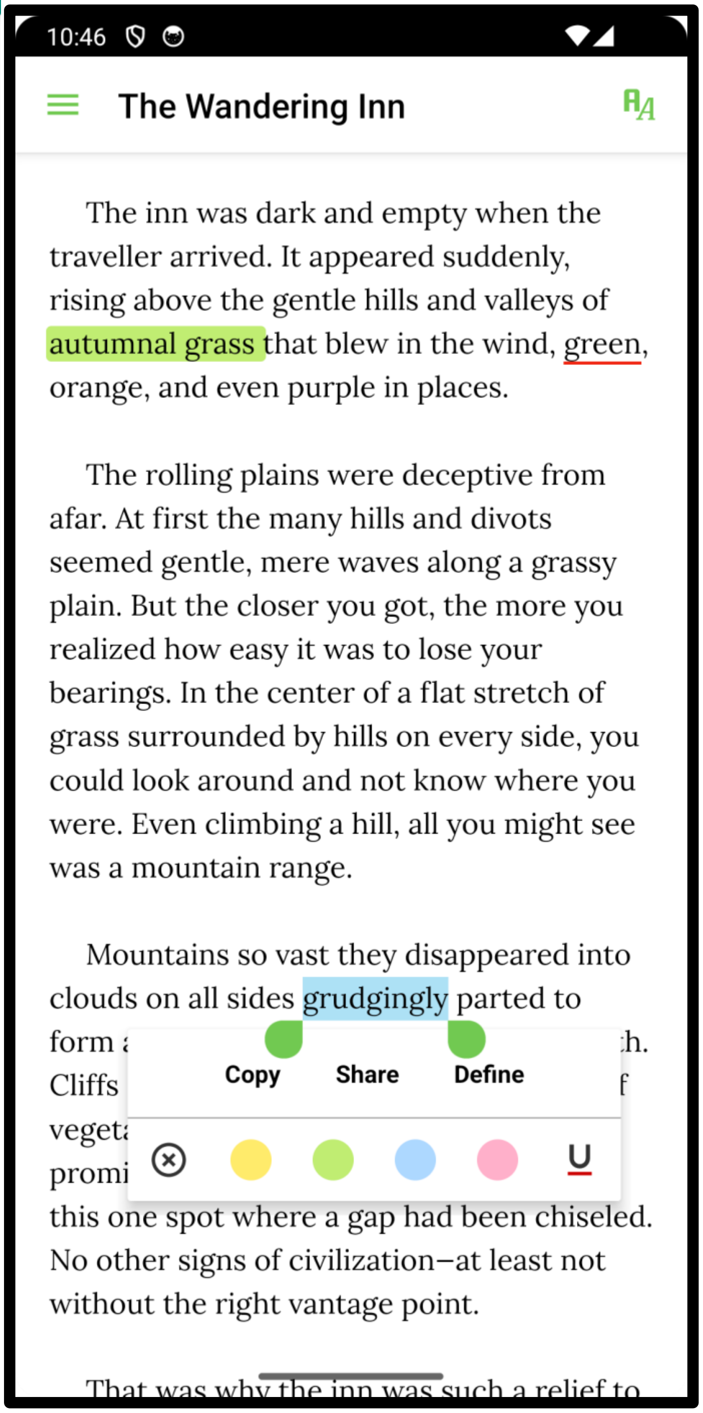
Once all chapters are added to the book object, a css style file is added which holds the fonts and formatting for the epub. Finally, the URI pointing to a file location that was provided by the user earlier is used to create a File Output Stream and write the e-book to disk. If successful, a toast message is created, and the user is returned to the application entry point activity. If the book building fails, a different toast message is presented, and the user is brought back to Build Book Setup Activity.

IV. Results

The above descriptions of the web novel parsing and the logic used to get there is the majority of my contribution to this application. I did not design the actual e-book viewer. However, there was a fair amount more that needed to be added to be able open a newly created e-book. One of these modifications has already been mentioned, the handling of incompatibility between FolioReader the e-book reader and Epub4J the e-book builder.

A second, and just as important, modification was the file system addition to pick a book from the user's storage. All the FolioReader framework provides is a way to retrieve the singleton Folio object, and then a call like: `folioObj.openBook(…)`, where `...` could be various arguments. One argument it does not accept is a URI, which is the only identifier we have when a user selects an item with the system File Picker.

To solve this, I needed to incorporate the path to open, parse, and display an e-book from a URI. This was a non-insignificant undertaking. FolioReader is designed to open a book using an absolute path to permissioned storage and a book identifier, which we have neither of.



The inn was dark and empty when the traveller arrived. It appeared suddenly, rising above the gentle hills and valleys of autumnal grass that blew in the wind, green, orange, and even purple in places.

The rolling plains were deceptive from afar. At first the many hills and divots seemed gentle, mere waves along a grassy plain. But the closer you got, the more you realized how easy it was to lose your bearings. In the center of a flat stretch of grass surrounded by hills on every side, you could look around and not know where you were. Even climbing a hill, all you might see was a mountain range.

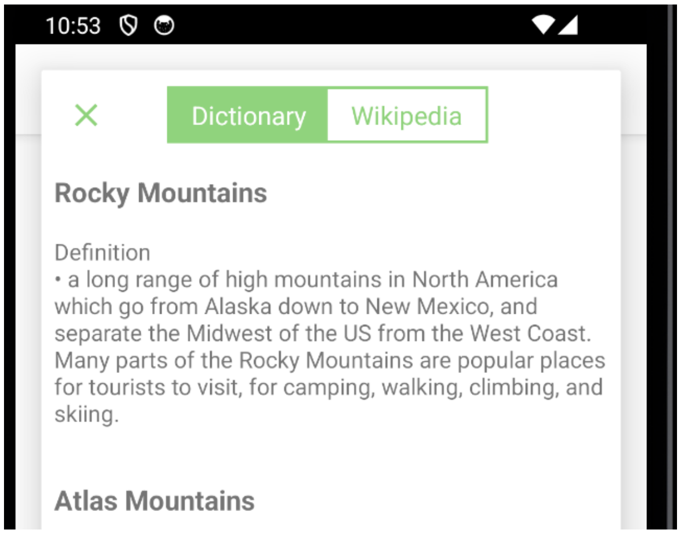
Mountains so vast they disappeared into clouds on all sides grudgingly parted to form a … h. Cliffs … f veget… promi… this one spot where a gap had been chiseled. No other signs of civilization—at least not without the right vantage point.

That was why the inn was such a relief to

Once the user chooses to open a book from internal storage and uses the File Picker, they are immediately brought to the cover page of the book – indicated by the .epub *spine* – or to the position the reader left off from. This can be seen in the next image.

In the image above, the opening page of an e-book selected from internal storage. Note the title at the top and the hamburger menu option at the top left. That button leads the user to the table of contents. Next notice the green highlighting and the underline word. Both markings will persist in the SQL Lite database, stored with the e-books unique identifier and their position in the text.

Another feature visible in the above image is the custom pop-


10:53

Dictionary | Wikipedia

Rocky Mountains

Definition

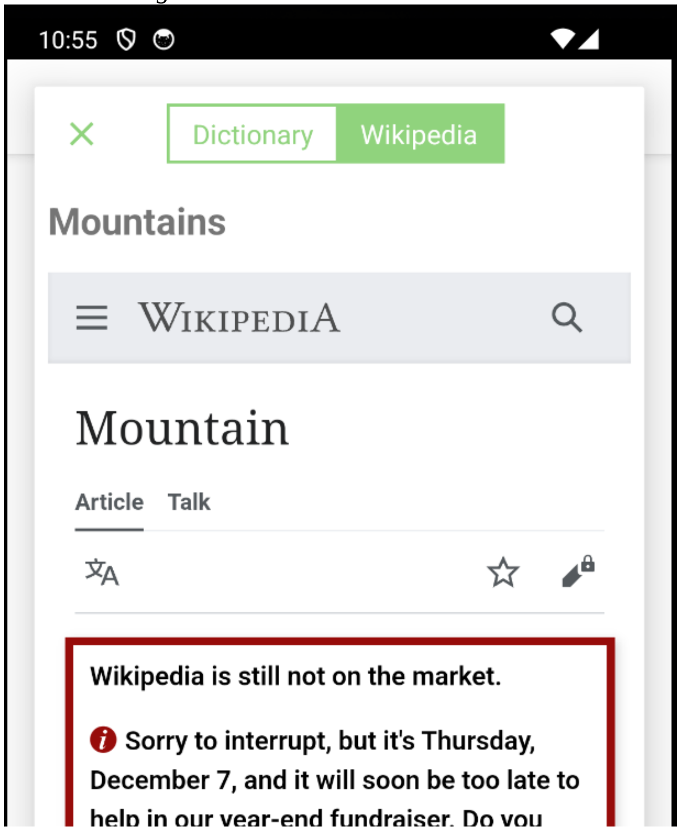
• a long range of high mountains in North America which go from Alaska down to New Mexico, and separate the Midwest of the US from the West Coast. Many parts of the Rocky Mountains are popular places for tourists to visit, for camping, walking, climbing, and skiing.

Atlas Mountains

up menu when a word is selected. When a user clicks on *Define*, the ViewModel contacts the Model which makes an API call to the Pearson Dictionary API. The fragment the user sees can be viewed in the next image.

In the image above, there is an option to switch from the *Dictionary* to the *Wikipedia* tab. When the user selects this option, the ViewModel contacts the Model which makes an API call to the Wikipedia API and parses the response body into what can be seen in the image below.

The last feature to demonstrate is an e-books TOC. This is what allows a user to switch between arcs, chapters, and sections. This view can be seen in the next image, with sections 1.00 to 1.07 being displayed. Most books have chapters with either chapter names or a numerical identifier starting at 1, such as "Chapter 1". However, the web novel *The Wandering Inn* uses the form "1.00… 1.69" to indicate


10:55

Dictionary | Wikipedia

Mountains

WIKIPEDIA

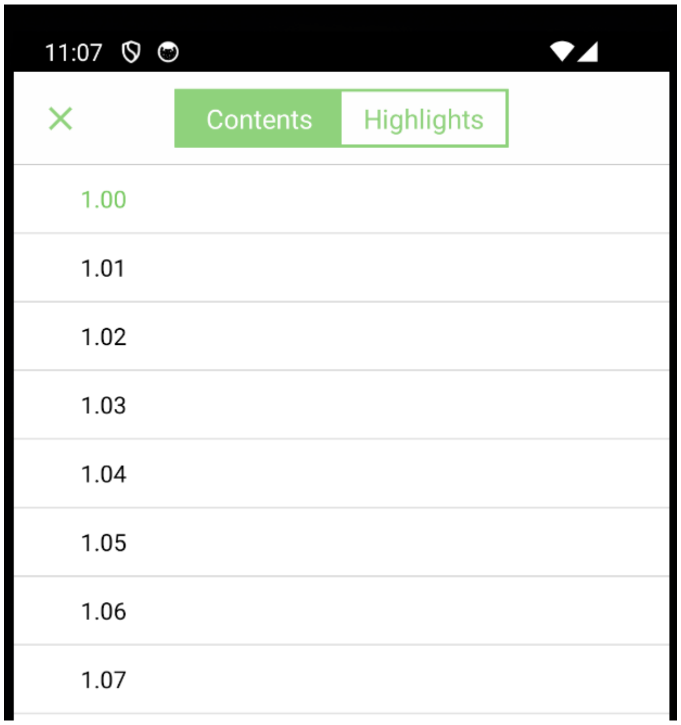
Mountain

Article Talk

Wikipedia is still not on the market.

Sorry to interrupt, but it's Thursday, December 7, and it will soon be too late to help in our year-end fundraiser. Do you

chapters 1 through 69 before moving on to the second arc, with chapter "2.00". In the image below, the user chose to only parse the first seven chapters.


11:07

Contents | Highlights

1.00

1.01

1.02

1.03

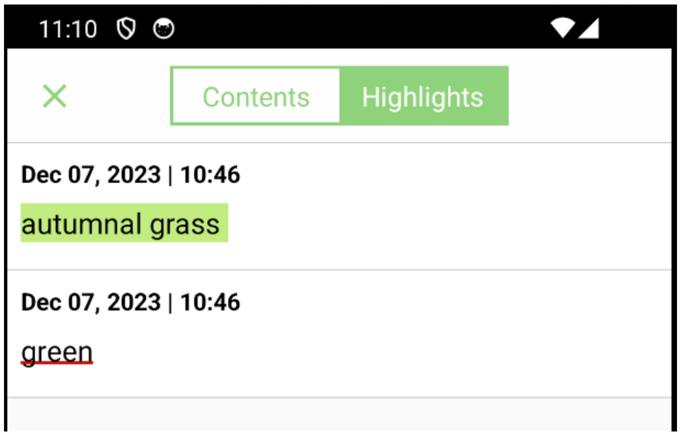
1.04

1.05

1.06

1.07

In the above image there is also an option to select *Highlights*. This will bring the user to a list of highlights and underlines made with in a particular e-book. This can be seen in the image below.



Dec 07, 2023 | 10:46
autumnal grass

Dec 07, 2023 | 10:46
green

The *Highlights* section in the image above also shows the date and time each highlight was done. This is assorted in ascending order. This section is useful if the user is reading a text-book or other non-fiction work in which they need to store certain sections for later review.

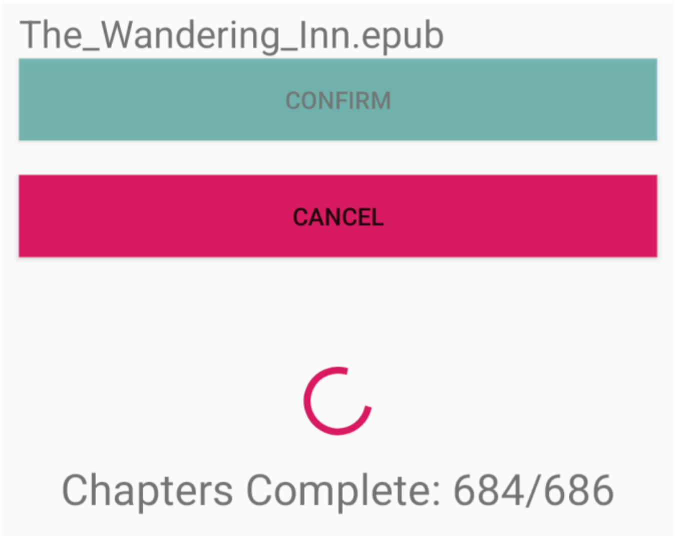
This project can be considered a success. The initial goal was to develop an application in which a user can build an e-book file of any web novel they have the rights to download. That is the ultimate goal. This could have been satisfied with a simple application with a single Activity in which the user submits a URL and receives an .epub file in exchange. Instead, we have built an application in which the user cannot only customize the building of an e-book, but also open and view the e-book in the same application. However, since the e-book is downloaded to the user's internal storage, they can open the e-book in any other e-reader application.

This will be the way I take advantage of BYO-EPUB. The e-reader application that I use has a *Text-to-Speech* configuration that I use every day. I have successfully downloaded an e-book file that I have built on this application. I no longer need to log into my desktop computer, use the Chrome browser to package an .epub file, and then transfer that file back to my phone.

I can now simply open BYO-EPUB, paste a hyperlink, select the most recently released chapter by swiping up once on the first Number Picker, and then opening that same book in my preferred reader. I even have the option to build an e-book of 686 chapters and over 12,000,000 words all from my phone. This is a significant feat of both mobile computing power and concurrency. Over 12 million words requested, parsed, cleaned, reformatted, converted to XHTML, compressed, and then downloaded in less than a minute. This can be seen in the image below and is an indicator of this project's success.

V. Future Work

Much of the work on BYO-EPUB e-book builder is complete. It is able to parse web novels from the top ten most popular web novel hosting sites. It provides the user with modification options for the book that is being parsed and built. However, the application is not in a finished state. The



Chapters Complete: 684/686

work that still needs to be done will be discussed below.

The first improvement that needs to be made on this application is to add a library viewer activity. This would be a display that shows all the e-books a user has built or opened in BYO-EPUB. It would display either the cover image of a book, or the title of the book if it did not have a cover image. Each image and title would be stored in app specific storage and accessed with a ViewModel. This would make a reader more likely to select BYO-EPUB for its reader features rather than just its e-book building capabilities.

The next addition to be added to the application is a custom parser. If the user has a web novel from a web-site that does not have a corresponding parser, they should be able to create their own. The back-end logic for this has already been written. However, I have not yet created an activity to implement it. This would require more work on the users end. They will need to view the HTML of their chosen web novel and find the HTML tag attributes that indicate the novels body content. They will also need to indicate any tags that need to be removed while parsing.

A third addition would be more parser settings. Options to indicate that the parser should spread out the chapter requests. Some web-sites restrict the number of HTTP requests than can be made in quick succession to prevent denial of service attacks. Our concurrent parser may trigger this. At the moment, there is no option to set a cool down on the parser. It instead tries to build an e-book as quickly as possible.

A fourth addition that needs to be implemented is more parsers. The parser logic was borrowed from WebToEpub. That program has over 100 individual parsers. Most were crowd-sourced by fellow readers who made pull-requests to the GitHub repository it is held in. This could be an approach that BYO-EPUB takes. It does not take a lot of work to add an additional parser. A programmer needs only to implement the parser interface and modify the abstract methods that are site-specific. While the Utils class mentioned above does have near 100 methods designed to handle DOM oddities, it is not perfect. Future parser additions might need to add additional utility methods to the Utils class if that parser's web-site contains a new DOM structure not handled by the utilities that are currently in place.

Another future modification is the improvement of the UI look and feel. Focus was placed on getting the e-book builder in place and interactable. The current UI lacks a modern, clean look and feel. This includes adding an application icon and adding back buttons on each activity. We also need to add layouts for landscape-mode and dark-mode users.

Finally, there needs to be research into the legal issues this application may face. Users should only create e-books for web novels they have the rights to. However, there is nothing in place to prevent users from creating an .epub file of a book they do not own. While the web-sites these books are built from are publicly available, they may have restrictions on the modification and redistribution of the web-site's content.

I have received express permissions from the author of *The Wandering Inn* web novel to build an e-book of their site's content. I regularly donate to the author's Patreon page and wanted to make sure I was not infringing on their distribution rights. As long as I do not distribute or sell their works, they are fine with the way I consume their content. Other web sites and authors may have a different standpoint. Effort should be made to make sure that we, the creator of this application, cannot be held legally liable for anything a user does with our application.

References

[1] B. Lusina, "Librum," [Online]. Available: https://github.com/materka/Librum/tree/master.

[2] I. Ivanenko, "LibreraReader," [Online]. Available: https://github.com/foobnix/LibreraReader.

[3] J. Hedley, "JSoup" [Online]. Available: https://jsoup.org/.

[4] Dteviot, "WebToEpub" [Online]. Available: https://github.com/dteviot/WebToEpub/tree/master.

[5] E. Carpes, "FolioReader," [Online] Available: https://github.com/FolioReader/FolioReader-Android

[6] ReadEra Team, "ReadEra," [Online]. Available: https://play.google.com/store/apps/details?id=org.readera

[7] MobiPups+, "eBoox," [Online]. Available: https://play.google.com/store/apps/details?id=com.mobisoft.knigopis

[8] OverDrive, Inc., "Libby," [Online]. Available: https://www.overdrive.com/apps/libby/

[9] FBReader.ORG Limited, "FBReader," [Online]. Available: https://fbreader.org/

[10] Singapore New Reading Technology Pte Ltd, "GoodNovel," [Online]. Available: https://www.goodnovel.com/

[11] Babel Novel, "Babel Novel," [Online]. Available: https://www.babelnovel.com/

[12] Cloudary Holdings Limited, "WebNovel," [Online]. Available: https://www.webnovel.com/

[13] Light Novel, "Light Novel," [Online]. Available: https://play.google.com/store/apps/details?id=com.light.novel

[14] Amazon Mobile LLC, "Amazon Kindle," [Online]. Available: https://www.amazon.com/kindle-dbs/fd/kcp

[15] Pirateaba, "The Wandering Inn," [Online] Available: https://www.wanderinginn.com